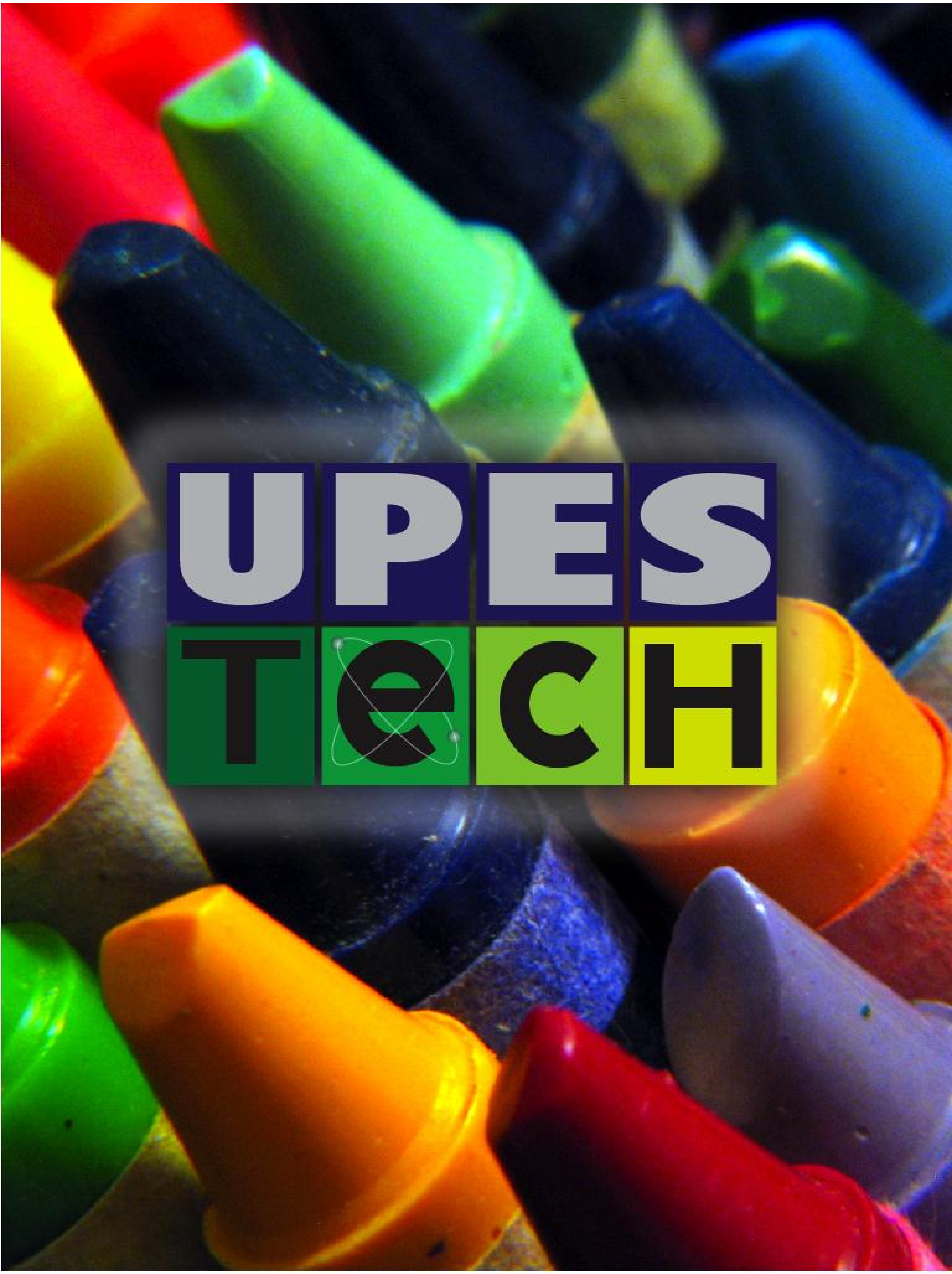


UPES TECH

Date

02.06.12

Cosmos

① Cormen
② Google

Algorithms

Syllabus

① Analysis
② Divide & Conquer
③ Greedy Technique
④ Dynamic Programming
⑤ Graph, Tree
⑥ Hashing
⑦ P, NP, NPC, NPH

Algorithm

Definition: It is a combination of sequence of finite ~~sets~~ steps to solve a particular problem.

Properties of Algorithm:-

① It should take finite time to produce O/P.
② It should produce the correct O/P.
③ Every step in the algo should be unambiguous (or) deterministic.
④ Every step in the algorithm should perform some task
⑤ Every algo should produce atlest 1 output.
⑥ Every algo should take zero or more I/P.
⑦ Algorithm should be independent of programming language.

Non-Deterministic Algo:-
Random no generator

Analysis Of Algorithm

Note: How to check in the available algorithms, which is the best one?

① Time ② Space

How to find time complexity of the algo?

T(A) :- time complexity of the algorithm A

$T(A) = C(A) + R(A)$

compile time of A
• (it depends on the compiler)
• S/W

Run-time of A
• (it depends on the processor)
• H/W

★ $\boxed{n}$ is considered to be very large

Analysis

Apostiory	Apriori
① It is programming of a compiler & h/w dependent analysis	① It is independent of programming language & h/w.
② ~~It is dependent~~ The complexity/ answer changes from comp. to computer.	② The answer will not change. (i.e. the complexity doesn't change from computer to computer)
③ Exact answer.	③ Approximated answer

depends pon mpiler & rocessor ←

Apriori Analysis:-

is the determination of order of magnitude of a stateme

```
Main()
{ int x,y,z;  ──→ order of magnitude of this statement is 1.
                  (which means that this statement is
                   executed once when the program
                   executes)
  x=y+z;
}
```

```
   Main()
1. { int x,y,z,i,n;        → order of magnitude=1
2.   x= y+z;               → order of magnitude=1
     for(i=1; i≤n; i++)
5.   { x=y+z;              → order of magnitude= n
     }
}
```

$$\boxed{n+2 = O(n)}$$

No of stater
nts & steps =
3

```
Main()
{ int x,y,z,i,j,n;   → order of magnitude=1
  x=y+z;             →   "     "     "   =1
  for(i=1;i≤n;i++)   →
     x=y+z;          →   "     "     "   = n
  for (i=1; i≤n; i++)  }
  { for(j= 1;j≤n;j++)  } → order of magnitude = n^2
      { x=y+z;         }
      }
  }
}
```

2+n

$$\boxed{n+2+n^2 = O(n^2)}$$

Cosmos

```
④ Main()
{int x,y,z,i,j,n;      ⟶ order of magnitude = 1
  x = y+z;             ⟶  "    "    "       = 1
  for(i=1 ton)
    x=y+z;             ⟶  "    "    "       = n
  for (i=1 to n)       ⟶  "    "    "       = n   } ⟶ n²/2
    for(j=1 to n/2)    ⟶  "    "    "       = n/2 }
      x=y+z;
}
```

$$\frac{n^2}{2}+n \to \frac{1}{2}n^2 = O(n^2)$$ → as n is c

therefore $\frac{n^2}{2}$

☆

```
⑤ Main()
{int x,y,z,n;      ⟶ order of magnitude = 1
  x=y+z;           ⟶  "    "    "       = 1
  While(n>1)
  {x=y+z  ]
   n= n/2 ]
  }
}
```

if n=16 then loop will be executed 4 times.	if n=8 then loop will be executed 3 times.	if n=32 then loop will be executed 5 times

$\boxed{\log_2 n}$

∴ 2($\log_2 n$) statements will be executed.

+ $2\log_2 n + 2$

but we can neglect +2 & multiple 2.

∴ complexity is $\log_2 n$

but if n=n/3	but if n=n-1
then $\log_3 n$	then O(n)
if n=n/10	if n= n-2
then $\log_{10} n$	then $\frac{n}{2}$, & O(n)

```
) Main()
{ int x,y,z,i,j,k,n;
  for (i=1 to n)          ⟶ order of magnitude = n
  { for (j=1 to i)        ⟶ 1, 2, 3, . . . . , n
    {for(k=1 to 135)      ⟶ order of magnitude = 135
      {x=y+z;
      }
} } }
```

$$135\times n\left[\frac{n(n+1)}{2}\right] = \frac{135\times n^2(n+1)}{2}$$

$$= (135n^3 + 135)/2$$

Cosmos

i = 1	i = 2	i = 3	. . . .	i = n
j = 1 to 1	j = 1 to 2	j = 1, 2, 3		j = 1, 2, 3, . . . , n
135	135 135	135 135 135		135 135 135 135

$T = 1 \times 135 + 2 \times 135 + \dots + n \times 135$

$= 135 [1 + 2 + \dots + n]$

$= 135 \frac{n(n+1)}{2} \Rightarrow O(n^2)$

```
Main()
{ int x, y, z, i, j, k, n;
   for(i = 1 to n)
     { for j=1 to i^2)
        { for (k=1 to n/2)
           { x = y+z;
           }
        }
     }
}
```

i = 1	i = 2	i = 3
j = 1 to 1	j = 1 to 4	j = 1 to 9
k = 1 to n/2	k = 1 to n/2	k = 1 to n/2

$T = 1 \times \frac{n}{2} + 4 \times \frac{n}{2} + 9 \times \frac{n}{2} + \dots + n^2 \times \frac{n}{2}$

$= \frac{n}{2} [1 + 4 + 9 + \dots + n^2] = \frac{n}{2} \left[\frac{n(n+1)(2n+1)}{6} \right]$

$\Downarrow$

$\boxed{O(n^4)}$

```
Main()
{ n = 2^(2^k);
   for (i=1; i≤n; i++)
     { j=2;
        while (j≤n)
          { j=j^2;
          }
     }
}
```

i = 1	i = 2	. . .	i = n
j = 2	j = 2		j = 2
2 ≤ n	$\log_2 n$		$\log_2 n$
$\log_2 n$			

n = 1
n = 4

j = 2

k = 2

16

n = 32
2
4
16

~~n x log~~

$n = 2^{2^k}$

$\log_2 n = 2^k$

$\log_2(\log_2 n) = k$

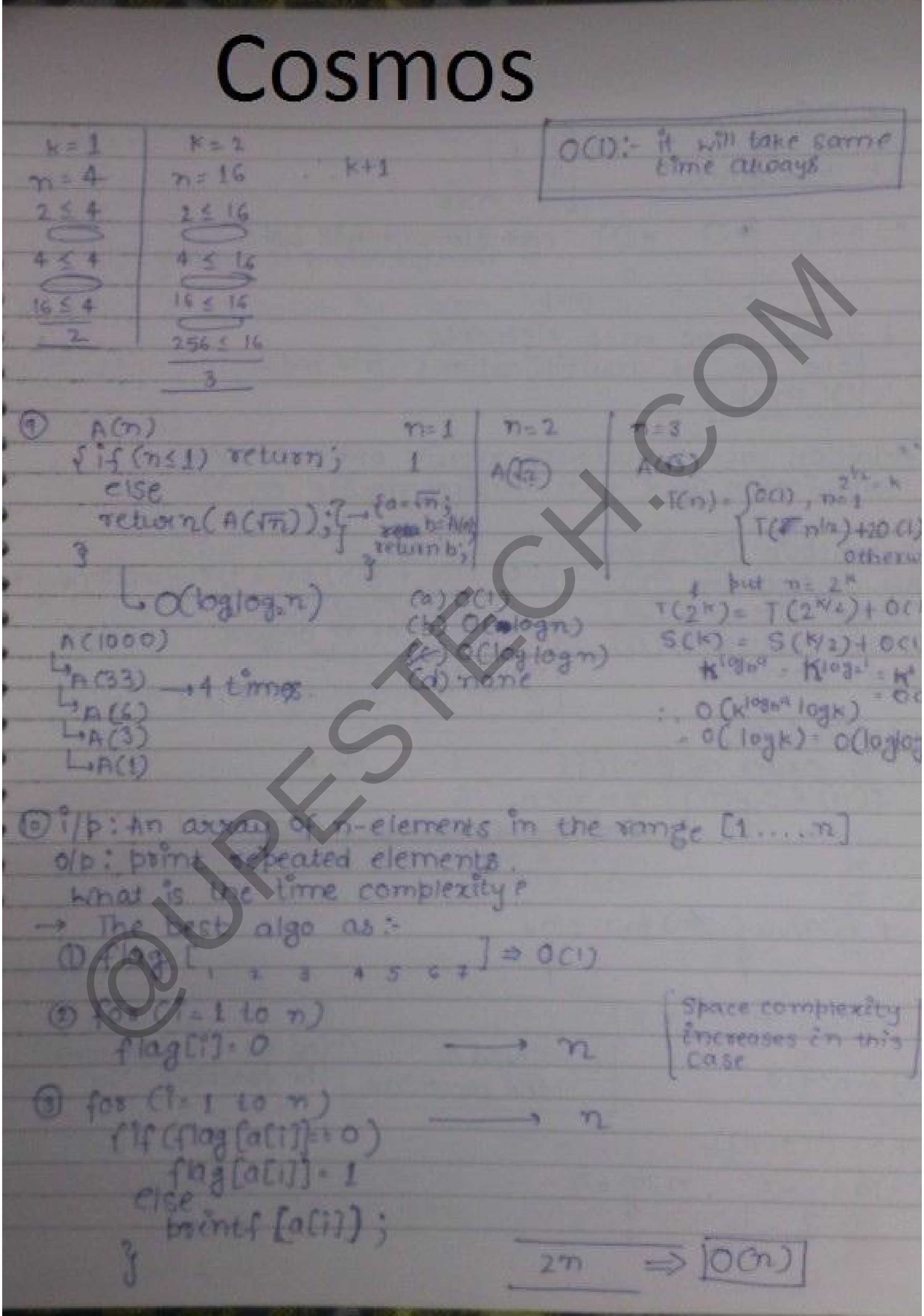
~~n x log~~

$n \times (k+1)$

$n \times [\log_2 \log_2 n] + n$

$\Rightarrow O(n \log_2 \log_2 n)$

Cosmos



k = 1	k = 2
n = 4	n = 16
2 ≤ 4	2 ≤ 16
4 ≤ 4	4 ≤ 16
16 ≤ 4	16 ≤ 16
2	256 ≤ 16
	3

k+1

O(1):- it will take same time always

⑦
```
A(n)
{ if (n≤1) return;
  else
  return(A(√n)); } → { a = √n;
                       b = A(a)
                       return b;
                     }
}
  └→ O(loglog₂n)
```

n=1	n=2	n=3
1	A(√2)	A(√3)

```
A(1000)
 └→ A(33) → 4 times
    └→ A(6)
     └→ A(3)
      └→ A(1)
```

(a) O(1)
(b) O(logn)
(c) O(loglogn)
(d) none

$$T(n) = \begin{cases} O(1), & n = 1 \\ T(n^{1/2}) + 2O(1) & \text{otherw} \end{cases}$$

$2^{1/k} = k$

put $n = 2^k$

$T(2^k) = T(2^{k/2}) + O($

$S(k) = S(k/2) + O($

$k^{\log_b a} = k^{\log_2 1} = k^0$

$= O($

$\therefore O(k^{\log_b a} \log k)$

$= O(\log k) = O(\log\log$

⑧ i/p: An array of n-elements in the range [1.....n]
o/p: print repeated elements.
What is the time complexity?
→ The best algo as:-

① flag [1 2 3 4 5 6 7] ⇒ O(1)

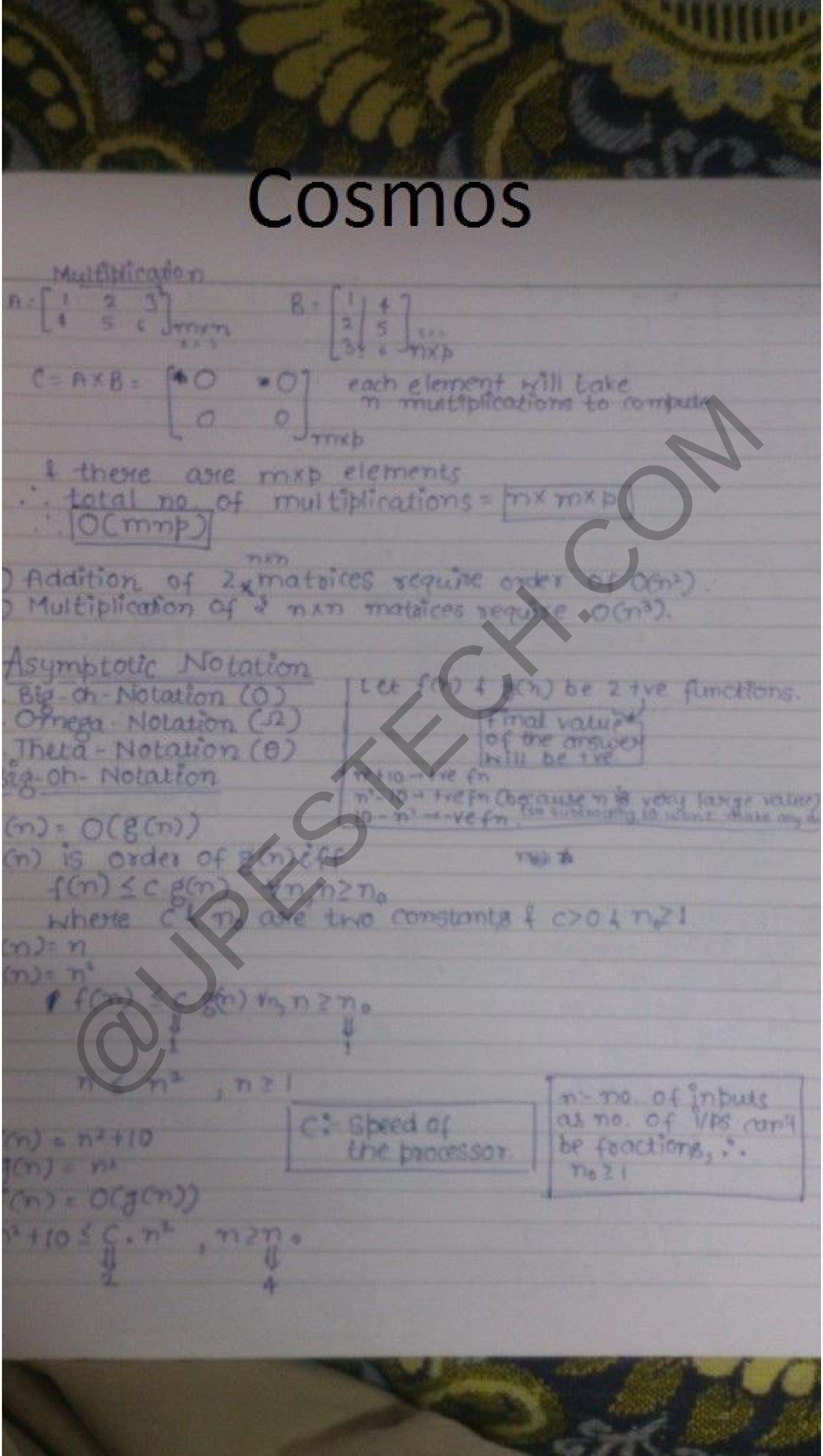
```
② for (i = 1 to n)
     flag[i] = 0                 ──→  n
```

Space complexity increases in this case

```
③ for (i = 1 to n)               ──→  n
   { if (flag[a[i]] == 0)
        flag[a[i]] = 1
     else
        printf [a[i]];
   }
```

2n ⇒ $\boxed{O(n)}$

Cosmos



Multiplication

$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{m \times n}$ $\quad B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{n \times p}$

$C = A \times B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}_{m \times p}$ each element will take n multiplications to compute

& there are $m \times p$ elements

∴ total no. of multiplications = $\boxed{n \times m \times p}$

∴ $\boxed{O(mnp)}$

) Addition of 2 $n \times n$ matrices require order of $O(n^2)$.

) Multiplication of 2 $n \times n$ matrices require $O(n^3)$.

Asymptotic Notation

. Big-oh-Notation (O)

. Omega Notation (Ω)

. Theta-Notation (θ)

Let f(n) & g(n) be 2 +ve functions.

Final value of the answer will be +ve

$n^2 + 10$ → +ve fn

$n^2 - 10$ → +ve fn (because n is very large value)

$10 - n^2$ → -ve fn. (on subtracting 10 won't make any d

Big-oh- Notation

$(n) = O(g(n))$

(n) is order of $g(n)$ iff

$f(n) \leq c\, g(n)\ \forall n, n \geq n_0$

where c & n_0 are two constants & $c > 0$ & $n_0 \geq 1$

$(n) = n$

$(n) = n^2$

$f(n) \leq c\, g(n)\ \forall n, n \geq n_0$

$\quad\quad\ \Downarrow \quad\quad\quad\quad \Downarrow$

$\quad\quad\ 1 \quad\quad\quad\quad\ 1$

$n \leq n^2 \quad , n \geq 1$

C: Speed of the processor.

n: no. of inputs as no. of I/Ps can't be fractions, ∴ $n_0 \geq 1$

$(n) = n^2 + 10$

$g(n) = n^2$

$(n) = O(g(n))$

$n^2 + 10 \leq \underset{\Downarrow\ 2}{C} \cdot n^2 \quad , n \geq \underset{\Downarrow\ 4}{n_0}$

Cosmos

- $f(n) = n^2$
- $g(n) = n$

$f(n) \leq c.g(n)$

. there is no c for these functions

$\therefore f(n) \neq O(g(n))$

☆ $\boxed{g(n) \text{ is greater by factor 'c' than } f(n)}$

Omega Notation(Ω)

$f(n) = \Omega(g(n))$

or

f(n) is omega of g(n) iff

$f(n) \geq c\,g(n) \quad \forall n, n \geq n_0$

where c & n_0 are 2 constants, $c>0$ & $n_0 \geq 1$

- $f(n) = n^2 - 10$

$g(n) = n^2$

$f(n) = \Omega(g(n))$

$n^2 - 10 \geq \underset{\overset{\Downarrow}{1/2}}{c}.n^2 \quad \boxed{n_0 \geq 5}$

$\boxed{g(n) \text{ is smaller than } f(n) \text{ by 'c' times.}}$

- $f(n) = n^2$

$g(n) = n$

$n^2 \geq \underset{\overset{\Downarrow}{1}}{c}.n \;, n_0 \geq 1$

- $f(n) = n$

$g(n) = n^2$

$n \geq c.n^2$

there is no such c.

$\therefore f(n) \neq \Omega(n^2)$

Theta Notation

$f(n) = \theta(g(n))$

or

f(n) is theta of g(n) iff

① f(n) is order of g(n)

$f(n) \leq c_1 g(n)$ &

② $f(n) = \Omega(g(n)$

ie $f(n) \geq c_2 g(n)$

such that there exist 2 constants c_1 & c_2 & n_0 where $c_1>0, c_2>0$ & $n_0 \geq 1$.

☆ [the value of n_0 sho
be same for both O
& Ω(n) ?

e.g.

$f(n) = n^2 + 10$

$g(n) = n^2$

① $n^2 + 10 \leq c_1 n^2$

$c_1 = 2$

② $n^2 + 10 \geq c_2 n^2$

$c_2 = 1$

$\therefore \boxed{n^2 + 10 = \theta(n^2)}$

$f(n) = n^2$

$g(n) = n^2$

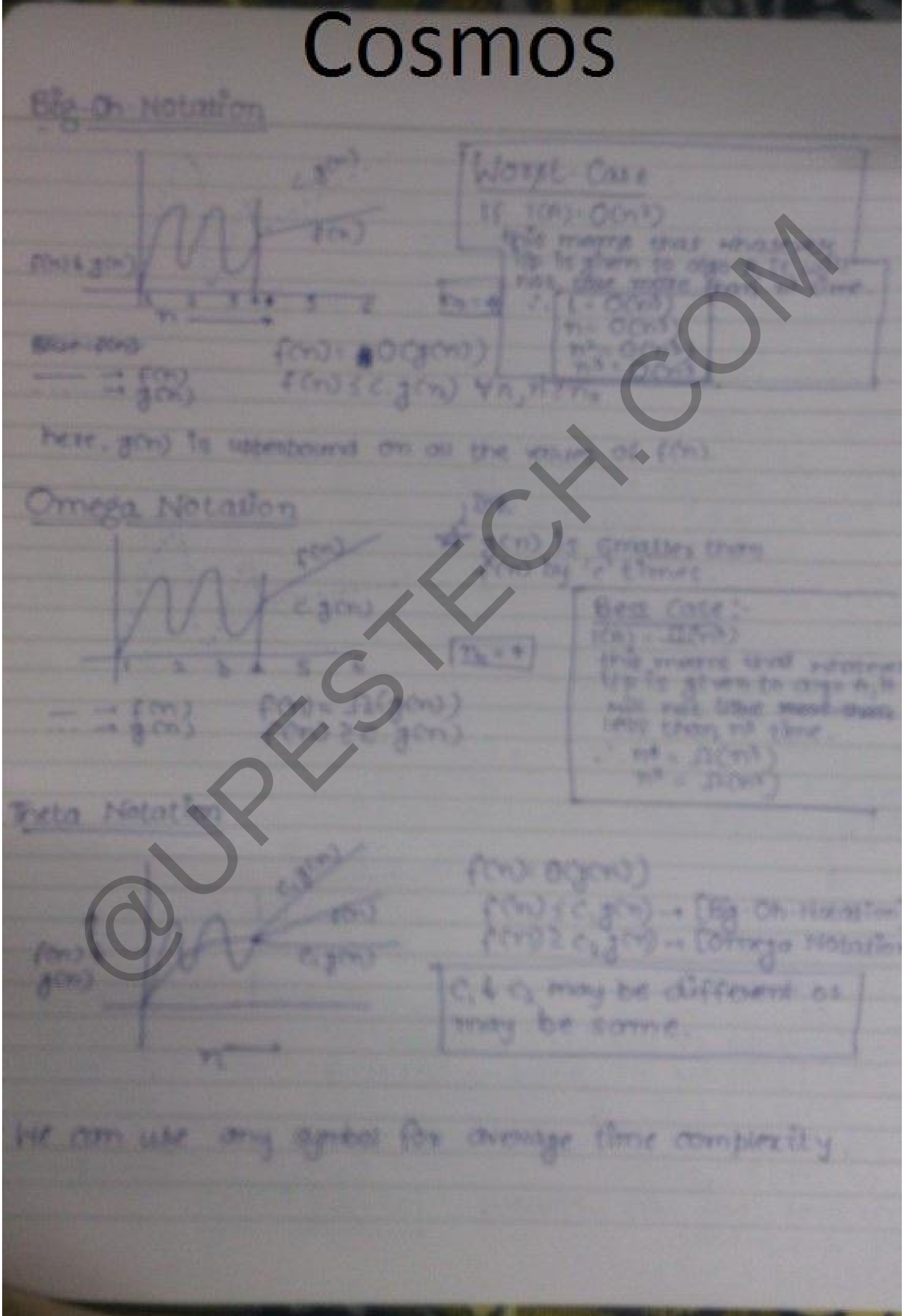
$n^2 = O(n^2$

$n^2 = \Omega(n^2$

$\therefore$

$\boxed{n^2 = \theta(n^2}$

Cosmos



Big-Oh Notation

Worst Case

If $f(n) = O(n^3)$

this means that whatever i/p is given to algo, it will not take more than n^3 time.

∴ $1 = O(n^3)$
$n = O(n^3)$
$n^2 = O(n^3)$
$n^3 = O(n^3)$

→ $f(n)$
→ $g(n)$

$f(n) = O(g(n))$

$f(n) \le c.g(n) \; \forall n, n \ge n_0$

here, g(n) is upperbound on all the values of f(n).

Omega Notation

Ex: c.g(n) is smaller than f(n) by 'c' times

→ $f(n)$
→ $g(n)$

$f(n) = \Omega(g(n))$
$f(n) \ge c.g(n)$

Best Case:

$f(n) = \Omega(n^3)$

this means that whatever i/p is given to algo, it will not take ~~more than~~ less than n^3 time

∴ $n^4 = \Omega(n^3)$
$n^5 = \Omega(n^3)$

Theta Notation

$f(n) = \Theta(g(n))$
$f(n) \le c_1 g(n)$ → [Big Oh Notation]
$f(n) \ge c_2 g(n)$ → [Omega Notation]

c_1 & c_2 may be different or may be same.

We can use any symbol for average time complexity.

★If there are 2 algo A & B.

$T(A) = O(T(B)) \longrightarrow$ A is better

$T(A) = \Omega(T(B)) \longrightarrow$ B is better

$T(A) = \Theta(T(B)) \longrightarrow$ Both are equivalent.

Conclusion :-

Big-oh-Notation (O)

$n^2 = O(n^2) \longrightarrow$ Tightest upper bound [the upper bound which is exactly equal to the f(n).

$n^2 = O(n^3)$

$n^2 \neq O(n)$

$n^2 = O(n^{100})$

A = O(B) here B is upper bound, where B may or may not b tight upper bound.

Small-oh-Notation :-

$f(n) = o(g(n))$

$n^2 = o(n^3)$ $\quad f(n) < c\,g(n)\ \forall n, n \geq n_0$

$n^2 = o(n^{100})$ $\quad$ such that $c > 0$ & $n_0 \geq 1$.

$n^2 \neq o(n^2)$

A = o(B), here B is upper bound, where B is not tighest uppe bound.

Big-Omega Notation (Ω):-

$n^2 = \Omega(n^2) \longrightarrow$ tightest lower bound.

$n^2 = \Omega(n)$

$n^2 = \Omega(1)$

A = Ω(B), here B is lower bound which may or may not tightest lower bound.

Small omega Notation (ω):-

$n^2 \neq \omega(n^2)$ $\quad f(n) = \omega(g(n))$

$n^2 = \omega(n)$ $\quad f(n) > c\,g(n)$

$n^2 = \omega(1)$ $\quad$ such that $c > 0$ & $n \geq n_0$

& $\boxed{n_0 \geq 1}$

A = ω(B), here B is lower bound which is not tightest lower bound.

Theta-Notation (θ)

$n^2 = \theta(n^2)$ ⟶ Both Tightest upper bound & tightest lower bound.
$n^2 \neq \theta(n)$
$n^2 \neq \theta(n^3)$
$A = \theta(B)$ → B is both Tightest upper bound & tightest lower bound.

Types Of Time Complexity :-

(1) Constant time complexity ⇒ $O(1)$ ⟵ $O(\log\log n)$ [in b/w]
(2) Logarithmic time complexity ⇒ $O(\lg n)$ ⟵ $O(\sqrt{\lg n})$ [in b/w]
(3) Linear " " ⇒ $O(n)$ ⟵ [illegible]
(4) Quadratic " " ⇒ $O(n^2)$ ⟶ $O(n \log n)$
(5) Cubic " " ⇒ $O(n^3)$ ⟶ $O(n^{3/2})$
(6) Polynomial " " ⇒ $O(n^k)$ $k \geq 1$
(7) Exponential " " ⇒ $O(k^n)$ $k \geq 2$

$n > (\log n)^2$ $\qquad 2^n < n^n$
$n > (\log n)^{100}$ → value of n is so large that constant powers can't overpower them
$n < (\log n)^n$

[$n > \log n$
& $n > (\log n)^c$
where c is a constant
by $n < (\log n)^n$

---

① $T = \sum_{i=1}^{n} i = 1+2+3+4+ \ldots +n = \frac{n(n+1)}{2} = O(n^2) = \Omega(n^2)$

② $T = \sum_{i=1}^{n} i^2 = 1^2+2^2+3^2+ \ldots +n^2 = \frac{n(n+1)(2n+1)}{6} = \boxed{O(n^3)} = \Omega(n^3)$

★ this means that the sum value can't cross n^3.

③

→ To add the squares of n natural nos., it will take $O(n)$, i.e. only one for loop.

④ $T = n \times n \times n \times \ldots \times n$ [n times]
$T = O(n^n)$ → this means T value can't exceed n^n.
It will take $O(n)$ to compute T.

(iii) $T = n!$

$= n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

$\leq n \times n \times n \times \ldots \times n \times n \times n$

$= O(n^n)$ → n! can't exceed n^n

→ It will take O(n) time to solve it, i.e. to calculate factorial because it will take only one for loop.

(iv) $T = \sum_{i=1}^{n} x^i$

$= x + x^2 + x^3 + \ldots + x^n$

almost equal $= \frac{x(x^n - 1)}{(x-1)}$ $= O(x^n)$ → it will can't exceed x^n with appropriate value of n.

① $f(n) = n^2$

$g(n) = 2^n$

$n^2 \leq c \cdot 2^n$

∴ $f(n) = O(g(n))$

$c = 1$ & $n_0 = 4$

② $f(n) = 2^n$

$g(n) = n^n$

n	2^n	n^n
1	2	1
2	4	4
3	8	27
4	16	256

∴ $f(n) = O(g(n))$

$c = 1$ & $n_0 = 2$

or $2^n = O(n^n)$

or $n^n = \Omega(2^n)$

③ $f(n) = n^2 \log n$

$g(n) = n(\log n)^{10}$

$n^2 \log n$ $n(\log n)^{10}$

$n \cdot n \log n$ $n \log n (\log n)^9$

$n > (\log n)^9$

∴ $f(n) > g(n)$

∴ $f(n) = \Omega(g(n))$

→ for very large values of n

e.g. $n = 2^{1000}$

$(\log n)^9 = (1000)^9$

$2^{1000} >>>> (1000)^9$

④ Which statement is false?

(a) $n^2 2^{3\log_2 n} = O(n^5)$

(b) $\frac{4^n}{2^n} = O(2^n)$

(c) $2^{\log_2 n} = O(n^2)$

(d) None

(a) $n^2 2^{3\log_2 n}$ n^5

$n^2 2^{\log_2 n^3}$ n^5

n^5

$\boxed{n^{\log_b a} = a^{\log_b n}}$

⑤ Which statement is T/F?

(a) $100\, n \log n = O\left(\frac{n \log n}{100}\right)$ → T $100 \cdot n \log n \leq c \cdot \frac{n \cdot \log n}{100}$

★ In such problems remove the constants.

$100\, n \log n \neq O\left(\frac{n \log n}{100}\right)$ → no constant should make f(n) & g(n) equal.

$(n+k)^m \simeq n^m + k^m$

(b) $\sqrt{\log n} = O(\log\log n) \rightarrow F$ ~~$\sqrt{\log n} \le c \log\log n$~~
(c) $0 < x < y$ then ~~$(\log n)^{1/2} \le c \log\log n$~~
$n^x = O(n^y) \rightarrow T$ ~~$\frac{1}{2} \log\log n \le c \log\log n$~~
(d) $2^n \ne O(n^k)$ where k is constant $\rightarrow T$

~~$2^n \le c.n^k$~~

★ exponential is always faster than the polynomial.

⑤ T/F = ?
(a) $(n+k)^m = O(n^m)$ where k is constant $\longrightarrow T$
(b) $2^{n+1} = O(2^n) \longrightarrow T$
(c) $2^{2n} = O(2^n) \longrightarrow F$

(b) $2^{n+1} \le c.2^n$
(c) $2^{2n} \le \overset{3}{c}\, 2^n$
$2.2^n \le 3.2^n$

(a) $(n+k)^m \le c.n^m$
$n^m + {}^mC_1 n^{m-1}k + {}^mC_2 n^{m-2} + \dots + k^m$
$\simeq n^m + k^m$
$\le n^m + n^m$
$= O(n^m)$

★ if k is not constant, then $O(n^m)$ & $O(k^m)$ can both be the answer.
★ constant is always smaller than the function.

Q Consider the following two functions :-
$f(n) = n^3 \quad 0 \le n < 10{,}000$
$\quad\;\; n^2 \quad n \ge 10{,}000$
$g(n) = n \quad 0 \le n < 100$
$\quad\;\; n^3 \quad n \ge 100$

- for n b/w 1 & 100
$\overset{n^3}{} \qquad \overset{n}{}$
$g(n) = O(f(n))$ or $f(n) = \Omega(g(n))$
- for n b/w 101 & 10,000
$\overset{n^3}{} \qquad \overset{n^3}{}$
$f(n) = \Theta(g(n)) \quad n_0 \ge 100$
- for $n \ge 10{,}000$
$\overset{n^2}{} \qquad \overset{n^3}{}$
$f(n) = O(g(n))$

$(n+k)^m \simeq n^m + k^m$

(b) $\sqrt{\log n} = O(\log\log n) \rightarrow F$ ~~$\sqrt{\log n} \le c \log\log n$~~
(c) $0 < x < y$ then ~~$(\log n)^{1/2} \le c \log\log n$~~
$n^x = O(n^y) \rightarrow T$ ~~$\frac{1}{2} \log\log n \le c \log\log n$~~
(d) $2^n \neq O(n^k)$ where k is constant $\rightarrow T$

$2^n \not\le c.n^k$

★ exponential is always faster than the polynomial.

② T/F = ?
(a) $(n+k)^m = O(n^m)$ where k is constant $\rightarrow T$
(b) $2^{n+1} = O(2^n) \rightarrow T$
(c) $2^{2n} = O(2^n) \rightarrow F$

(b) $2^{n+1} \le c.2^n$
$\Downarrow$
(c) $2^{2n} \le \overset{3}{c}\, 2^n$
$2.2^n \le 3.2^n$

(a) $(n+k)^m \le c.n^m$
$n^m + {}^mC_1 n^{m-1}k + {}^mC_2 n^{m-2} + \ldots + k^m$
$\simeq n^m + k^m$
$\le n^m + n^m$
$O(n^m)$

★ if k is not constant then $O(n^m)$ & $O(k^m)$ can both be the answer.
★ constant is always smaller than the function.

Q Consider the following two functions :-
$f(n) = n^3 \quad 0 \le n < 10{,}000$
$\qquad n^2 \quad n \ge 10{,}000$
$g(n) = n \quad 0 \le n < 100$
$\qquad n^3 \quad n \ge 100$

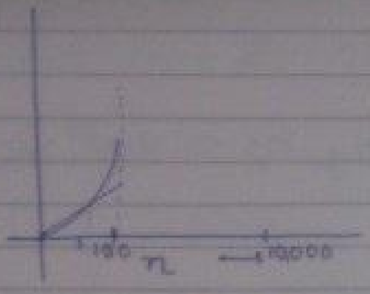
• for n b/w 1 & 100
$n^3 \qquad n$
$g(n) = O(f(n))$ or $f(n) = \Omega(g(n))$

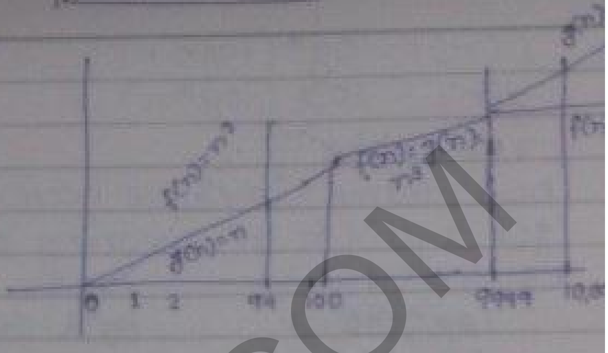
• for $n \ge 10{,}000$
$n^2 \qquad n^3$
$f(n) = O(g(n))$

• for n b/w 101 & 10000
$n^3 \qquad n^3$
$f(n) = \Theta(g(n)) \quad n_0 \ge 100$

$n! = o(n^n)$
$n! \neq \omega(n^n)$

Graph





Q Consider the following fn. :-
$f_1 = 2^n$
$f_2 = n^{3/2}$
$f_3 = n\log n$
$f_4 = n^{\log n}$
arrange above fn. in increasing order.
Ans $n\log n < n^{3/2} < n^{\log n} < 2^n$

1000 ×

$n^{\log n} > n^{3/2}$
because fn. is always greater than constant.
$\cancel{n}\log n \quad \cancel{n}n^{1/2}$
$\sqrt{n} > \log n$
$\frac{1}{2}\log n > \log\log n$

2^n
$n\log 2$

$n^{\log n}$
$\log n \times \log n$

Q $f_1 = (\log n)!$
$f_2 = \log(n!)$
$f_3 = (\log n)^{\log n}$

as $nx > x!$
∴ $f_2 < nx$
∴ $nx > f_2$ & $x! > f$

$f_1 = x!$
$f_2 = \log(n!) \le \log(n^n) = n\log n = nx$
$f_3 = x^x$

$(\log n)! < \log(n!)$

$\log(\log n)!$ $\log\log(n!)$

$(\log n)^{\log n}$
$\log n \log(\log n)$

$\log(n!)$
$\log(n!)$

$f_2 < f_1 < f_3$

Q. $f(n) = \log^*(\log n)$ $16 : 3$
$g(n) = \log(\log^* n)$ $n = 2^k$ $4 : 2$
rem b/w ?

how many times we have to apply log so that we can get 1, this is the meaning of *.

e.g. $\log^*_2 16 = 3$ $\log_2 3 = 1$
~~$\log^*_2 256 = 4$~~ $\log^* 4 = 2$
$\log_2{}^* 2^{16} = 4$

~~[illegible]~~

$f(n) = \log^*(\log 65{,}536) = 4$
$g(n) = \log(\log^* 65{,}536) = \log_2 5 = 2$
$f(n) > g(n)$

Properties Of Asymptotic Notation :-

① ~~Reflexing~~
Reflexive property
(i) $f(n) = O(f(n))$
(ii) $f(n) = \Omega(f(n))$
(iii) $f(n) = \Theta(f(n))$

② Symmetric Property
(i) If $f(n) = O(g(n))$
then $g(n) \neq O(f(n))$
(ii) If $f(n) = \Omega(g(n))$
then $g(n) \neq \Omega(f(n))$
(iii) If $f(n) = \Theta(g(n))$
then $g(n) = \Theta(f(n))$

③ Transitive Property
(i) If $f(n) = O(g(n))$ &
$g(n) = O(h(n))$
then $f(n) = O(h(n))$
(ii) If $f(n) = \Omega(g(n))$ &
$g(n) = \Omega(h(n))$
then $f(n) = \Omega(h(n))$
(iii) If $f(n) = \Theta(g(n))$ &
$g(n) = \Theta(h(n))$ then
$f(n) = \Theta(h(n))$

④ If $f(n) = O(g(n))$ then
$2^{f(n)} \neq O(2^{g(n)})$
because if $f(n) = 2n$
& $g(n) = n$
$2n = O(n)$
but $2^{2n} \neq O(2^n)$

⑤ $f(n) \neq O(f(\frac{n}{2}))$
because
take $f(n) = 2^{2n}$

Cosmos

⑥ $f(n) \neq O((f(n))^2)$

because take $f(n) = \frac{1}{n}$

$\frac{1}{n} > O\left(\frac{1}{n^2}\right)$

⑦ If $f(n) = O(g(n))$

&

$d(n) = O(h(n))$

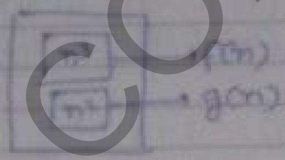
then

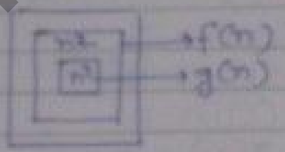
(i) $f(n) + d(n) = \max\{g(n), h(n))$ → $O(g(n)+h(n))$

(ii) $f(n) . d(n) = O(g(n) . h(n))$

→ both are executed individually

→ nested for loops





⑧ If $f(n) = O(g(n))$

then

$h(n) . f(n) = O(h(n) . g(n))$

e.g. ★ h(n) should be +ve function.

$3 < 5$

$6 \times 3 < 6 \times 5$

Q Consider following 4 fn :-

$f(n) = O(g(n))$ $g(n) = O(h(n))$ f(n)

& $g(n) \neq O(f(n))$ & $h(n) = O(g(n))$

True?

(a) $f(n) + g(n) = O(g(n))$ → T

(b) $f(n) = O(h(n))$ → T (by transitivity)

(c) $h(n) = O(f(n))$ → F

(d) $h(n) . f(n) = O(h(n) . h(n))$ → T

(d) $f(n) = O(h(n))$

$h(n) f(n) = O(h(n) . h(n))$

(c) $h(n) = O(g(n))$

but $g(n) \neq O(f(n))$

∴ $h(n) \neq O(f(n))$

(a) By $f(n) = O(g(n))$

∴ $f(n) < g(n)$

∴ f(n) + g(n) = max

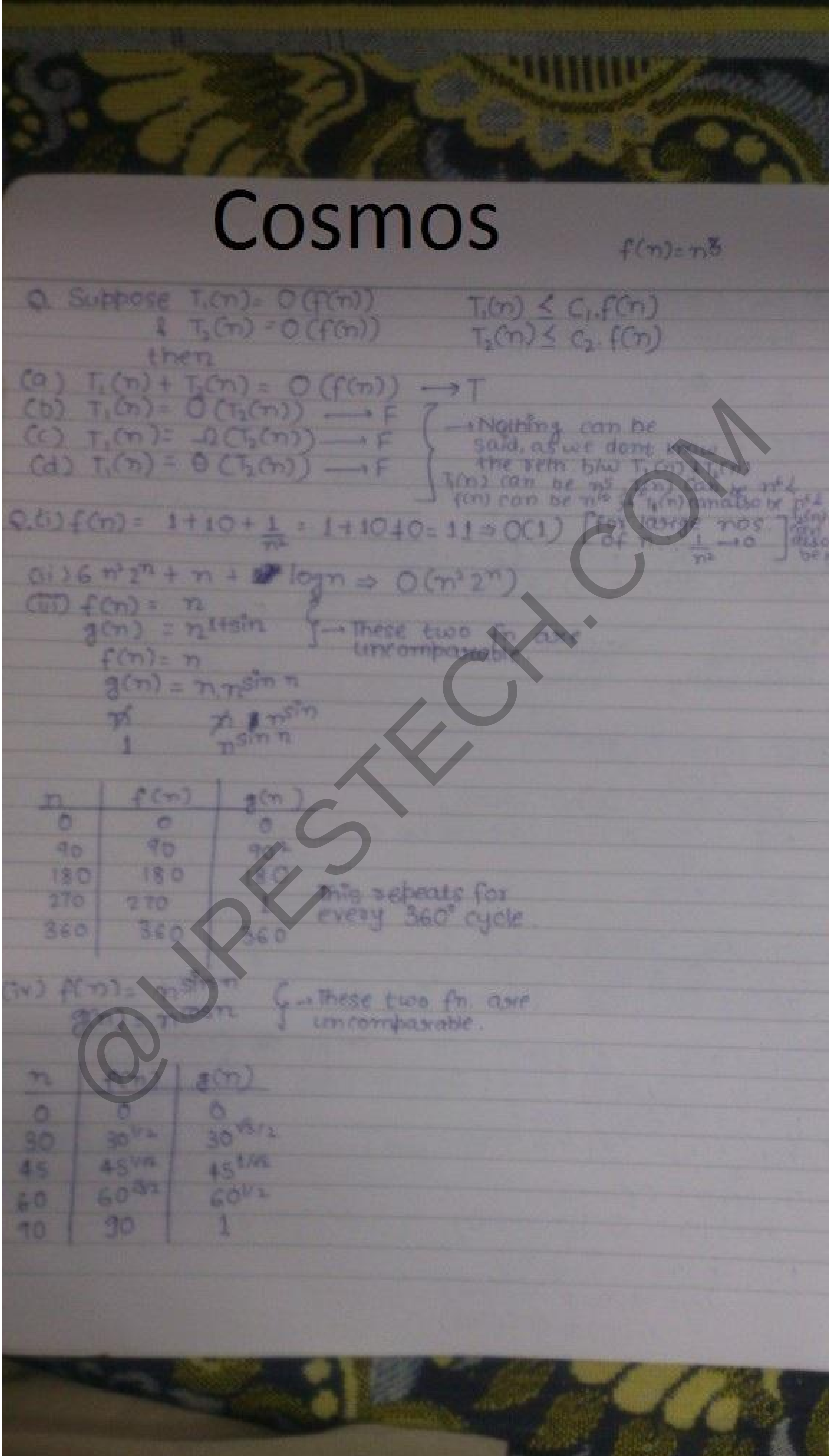
{f(n), g(n)}

but g(n) ≠ O(f(

∴ g(n) ≯ f(

∴ f(n) < g(n

∴ f(n) + g(n) = O



Cosmos

$f(n) = n^3$

Q. Suppose $T_1(n) = O(f(n))$ & $T_2(n) = O(f(n))$ then

$T_1(n) \leq C_1 . f(n)$
$T_2(n) \leq C_2 . f(n)$

(a) $T_1(n) + T_2(n) = O(f(n)) \longrightarrow T$
(b) $T_1(n) = O(T_2(n)) \longrightarrow F$
(c) $T_1(n) = \Omega(T_2(n)) \longrightarrow F$
(d) $T_1(n) = \Theta(T_2(n)) \longrightarrow F$

→ Nothing can be said, as we dont know the rel^n b/w $T_1(n)$ & $T_2(n)$. $T_1(n)$ can be n^5, $T_2(n)$ can be n^4 & $f(n)$ can be n^{10} & $T_1(n)$ can also be n^4 &

Q.(i) $f(n) = 1 + 10 + \frac{1}{n^2} = 1 + 10 + 0 = 11 \Rightarrow O(1)$ [for large nos of n … $\frac{1}{n^2} \to 0$]

(ii) $6n^2 2^n + n + \log n \Rightarrow O(n^2 2^n)$

(iii) $f(n) = n$
$g(n) = n^{1+\sin}$
→ These two fn are uncomparable

$f(n) = n$
$g(n) = n . n^{\sin n}$
$\cancel{n}$ $\cancel{n} \cdot n^{\sin}$
1 $n^{\sin n}$

n	f(n)	g(n)
0	0	0
90	90	90^2
180	180	180
270	270	1
360	360	360

this repeats for every 360° cycle

(iv) $f(n) = n^{\sin n}$
$g(n) = n^{\cos n}$
→ These two fn are uncomparable.

n	f(n)	g(n)
0	0	0
30	$30^{1/2}$	$30^{\sqrt{3}/2}$
45	$45^{1/\sqrt{2}}$	$45^{1/\sqrt{2}}$
60	$60^{\sqrt{3}/2}$	$60^{1/2}$
90	90	1

Date

09.06.12

★ Problem is small when no. of element is only 1.

Binary Search

T(n);

a[i] =	10	20	30	40	50	60	70
i =	1	2	3	4	5	6	7

a → array

x → no. to search

j → last element

i → 1st element

```
BS(a, i, j, x)
    if (i==j)
      { if (a[i] == x)          ⟶ O(1)
            return i;
         else
            return (-1);
      }
    else
      { mid= (i+j)/2;           ⟶ O(1)
        if (a[mid]==x) return(mid);  ⟶ O(1)
        else
         ~~if (a[mid BS~~
         if (a[mid]<x)          ⟶ O(1)
            BS(a, mid+1, j, x);
         else
            BS(a, i, mid-1, x);
      }
}
```

$$T(n) = \begin{cases} O(1) & ,\text{if } n=1 \\ O(1)+O(1)+O(1)+T(\frac{n}{2}) & \text{otherwise} \\ = T(\frac{n}{2})+c \end{cases}$$

$$T(n) = T(\tfrac{n}{2})+c$$

$$= T(\tfrac{n}{4})+c+c$$

$$= T(\tfrac{n}{8})+c+c+c$$

$$= T(\tfrac{n}{2^k})+kO(1)$$

$$= O(1) + O(\log n)$$

$n = 2^k$

$\log_2 n = k$.

Q1. I/p - A sorted array of n elements
o/p - Find 2 elements a & b such that a+b>1000

{int(a,i,j)
{ }

a[i] →	1000	2000	3000	4000	5000	6000	7000	8000	9000
i →	1	2	3	4	5	6	7	8	9

Array:-

- this will take O(1) to search the element after which we want to add
- this will take O(n) to add the element after the given element.

Link list

- this will take O(n) to search the address of the node after which we want to add the element
- this will take O(1) to add the element

& → reference operator
* → dereference operator

Solution:- take the last two elements & add them, if their sum >1000, then it is the answer, if they dont give then no other combination will give as it is the biggest combination.

```
algo :- ① pick last 2-element (a,b) → O(1)[as it an array
        ② if (a+b>1000) return (a,b); → O(1) [one comparison
           else                               to check greater
             return (-1);                     than 1000]
```

∴ only one comparison
∴ complexity = O(1) + O(1) = O(1)

Cosmos

Q2. similar to Q.1.
but ~~only~~ a+b==1000.

```
for(i=1; i≤n; i++)
  { for(j=i+1; j≤n; j++)
      { if(a[i]+a[j]==1000])
            return(a[i],a[j]);
        ~~else if (a[i]+a[j]>1000)~~
          ~~return(-1);~~
      }
  }
return(-1);
```

Time complexity:- $\boxed{O(n^2)}$

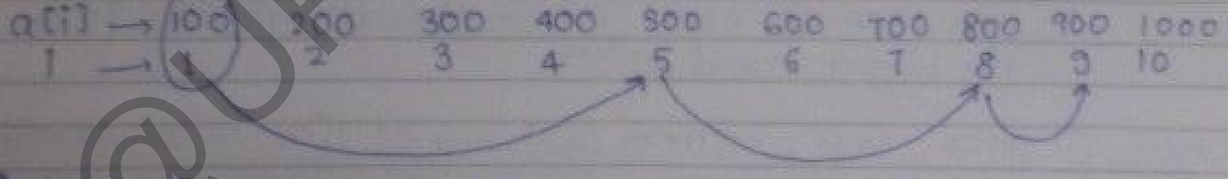
Algo:-

→ ① For the element a apply L.S. to find another element b such that a+b==1000. ⟶ $O(n)$

② continue for all elements until a+b==1000. ⟶ $O(n^2)$

$O(n^2)$

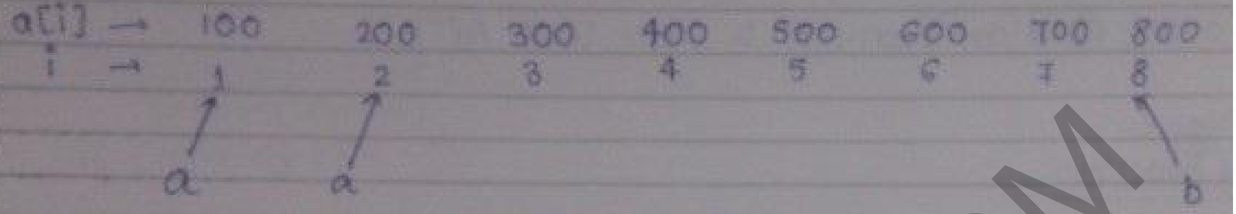
Improvement using Binary Search:-



① take a
② search b for every a which will take $O(lg n)$
there are n a's.
∴ Complexity = $n \times \log_2 n$ ⟶ $\boxed{O(n\log_2 n)}$

3rd Improvement :-

a+b = 1300



a[1] + a[8] = 900 < 1300
increase 1 to 2
∴ a[2] + a[8] = 1000 < 1300
increase 2 to 3
a[3] + a[8] = 1100 < 1300
increase 3 to 4
a[4] + a[8] = 1200 < 1300
increase 4 to 5
a[5] + a[8] = 1300 ⟶ a = 500 , b = 800

} ⟵ Greedy Technique.

∴ it will take only one scan.
∴ complexity (time) = $\boxed{O(n)}$

Algo :-

```
① Take 2-variables A & B .
      A → 1st element
      B → last element.
② If (A+B == C)
      return (A,B);
   else
      if (A+B > C)
        B = B-1;
      else
        A = A+1;
③ continue 2nd step until A+B == C.
```

But if there are no elements with A+B == C, then stop when i == j, (i.e. when the position of A & B are equal.

Q. i/p :- A sorted array of n-distinct elements both +ve o
-ve.
o/p :- find an element b such that A[i]==i.

Algo:-
My practice:-

① Go to the middle element & check whether it is +ve or -ve.

② If

```
    func(a,i,j){
for~~(i=1; i≤n; i++)~~
  { mid=$\frac{i+j}{2}$ ;  if (a[mid]==mid) return (mid);
  if (a[mid] ≥0 )
     ~~mid~~ i=mid+1;
  else
       j=mid-1;

   func(a,i,j);
}
```

Sir's Soln.:-

1) Using linear search.

```
   ~~Check~~
   for(i=1; i≤n; i++)
     {if (a[i]==i)
        return (i);
     }
```

$\longrightarrow \boxed{O(n)}$

) Using Binary search.

Cosmos

```
mid = (i+j)/2
if (a[mid]==mid)
    return (mid);
if (a[mid+1]
```

a[i] →	-100	-50	-35	4	10	12	13	14	17	110	125	135	140
i →	1	2	3	4	5	6	7	8	9	10	11	12	13

example:-

↱mid
a[i] → 14
i → 8

```
algo:-
if (position > value)
    go right
else
    go left
```

$\boxed{O(\log n)}$

as we can see
8 < 14
∴ the 7th element can be 7,
∴ go left.
we can't go right because
9th element will be > 14
[∵ since distinct elements,
in worst case we will have
9th element as 15
10th " " 16
11th " " 17
∴ position can never be equal to value.]

Q. I/P :- A sorted array of n-elements contain only 0 &
O/P :- which is greater, i.e whether no. of zeroes are greater or no. of 1's are greater.

Algo:-

```
        n0=0; n1=0;
for (i=1; i≤n; i++)
   { if (a[i]==0)
        n0++;
      else
        n1++;
   }                              → O(n)
if (n0>n1)
    return 0;
else
    return 1;
```

a[i] →	0	0	0	0	0	0	1	1	1	1
i →	1	2	3	4	5	6	7	8	9	10

① go to mid element,
② check mid element, if it is 0, then check

- if n is even.

if $(\frac{n}{2})$ is 0 && $(\frac{n}{2}+1)$ is 0,
then no. of 0's greater
else if $(\frac{n}{2})$ is 0 && $(\frac{n}{2}+1)$ is 1
then no. of 0's = no. of 1's
else if $(\frac{n}{2})$ is 1 && $(\frac{n}{2}-1)$ is 1 then no. of 1's greater.   } → O(1)

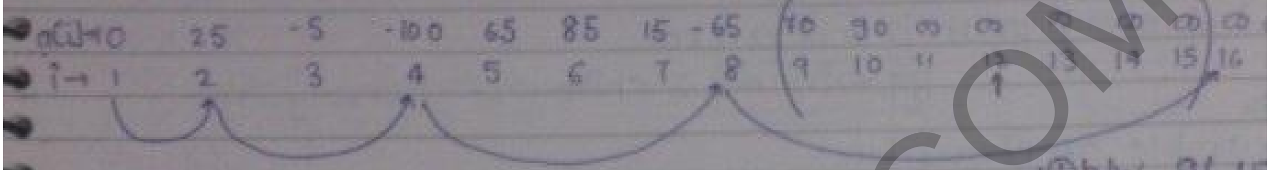
- if n is odd

if $\frac{n+1}{2}$ is 0, then no. of 0's are greater.
else no. of 1's are greater.   } → O(1)

Q. i/p :- An array of n elements unknown size, which contain both +ve & -ve no. until some place & afterwards all are ∞.
o/p :- find the 1st infinite place in the array.

a[i] →	0	-15	25	65	-5	100	∞	∞	∞	∞
i →	1	2	3	4	5	6	7	8	9	10

① Using linear search → O(n).



a[i] →	0	25	-5	-100	65	85	15	-65	40	90	∞	∞	∞	∞	∞	∞
i →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Algo:-

① Start with 1st element, if a[i] == ∞ then stop else multiply the position by 2 continue till a[i] == ∞.

② After reaching the ∞, then check whether the position & i before that ∞ is also ∞, then apply binary search b/w $\frac{i}{2}+1$ & $i-1$; & check whether middle element is ∞
if it is go left otherwise go right

① b/w 9 & 15
check 12th p
if is ∞
② check 11th is
then apply B
b/w 9 & 11.
③ check 10th is
its not then
[illegible]

* ④ we find
is ∞, which
the answ

~~We~~ Complexity:- $\boxed{O(\log_2 n)}$

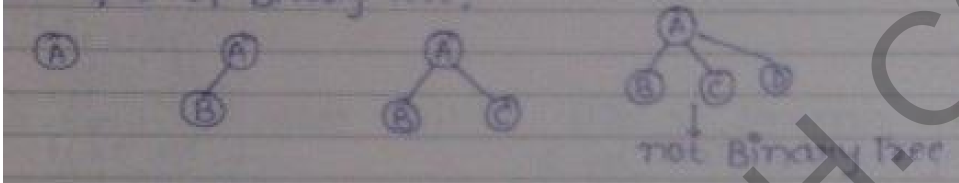
Graph	Tree
① No root element.	① root element is present.
② If there is a path b/w any two vertices then it is a connected graph.  If there is no path b/w any two vertices then it is non-connected graph.  May or may not be connected	② Tree is always connected.
③ May or may not be a directed graph.	③ It is always directed. (By default the directions are from top to bottom.)

May be cyclic or acyclic.	④ It is always acyclic.

Every tree is a graph,
but every graph is not a tree.

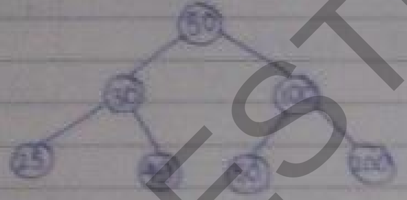
$\boxed{\text{Maximum}}$ 2-children → Binary Tree

xample of Binary Tree:-

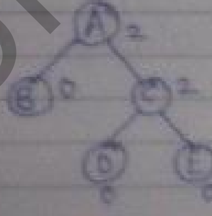


inary Search Tree:- In a given binary tree, at every
ode, root is ≥ all nodes present in left subtree &
all nodes present in right subtree.

e.g.

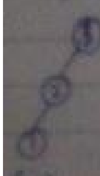


trict Binary Tree:



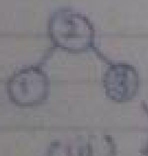
Every node can have 0 children or 2 children. [not a single child]

how many Binary Search Tree possible with 3 nodes.
1, 2, 3



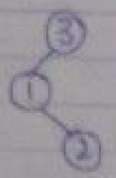
(i)

(ii)



(iii) this is the binary search tree made by binary search algo.

(iv)



(v)

★ No. of levels created by binary ~~bs~~ search algorithm = $\log_2 n$ [n - no. of elements]

★ height of tree = no. of levels - 1

- For Binary Tree: (n=5)

→ Max. levels = 5

→ Max. height = 5-1 = 4

- For Binary Search Tree: (n=5)

→ no. of levels (max.) = 5

→ Max. height = 4

- For Balanced Binary Search Tr (n=5)

→ no. of levels = $\log_2 5 = 3$

→ ~~max. of~~ height = 3-1 = 2.

★ No. of binary search trees possible with n-elements:-

$\boxed{\cancel{{}^{2n}C_{n/2}}}$ $\frac{{}^{2n}C_n}{n+1}$

→ if n=3

$${}^6C_3 = \frac{6!}{3!\,3!} = \frac{\cancel{6}\times5\times4}{\cancel{3}\times\cancel{2}} = 5\times4$$

$$\frac{{}^6C_3}{4} = \frac{5\times4}{4} = \boxed{5}$$

★ No. of Binary Trees with 3 nodes:-

~~[illegible]~~ 3×4 = 12 2×3 = 6 3×4 = 12

12+12+6 = $\boxed{30}$

★ : no. of binary trees with n-nodes:-

$$\boxed{n! \times \left(\frac{{}^{2n}C_n}{n+1}\right)}$$

Cosmos

★ No. of unlabeled Binary Trees = no. of binary search trees = $\frac{^{2n}C_n}{n+1}$.

★ No. of labeled Binary Trees = no. of binary trees = $n! \times \frac{^{2n}C_n}{n+1}$

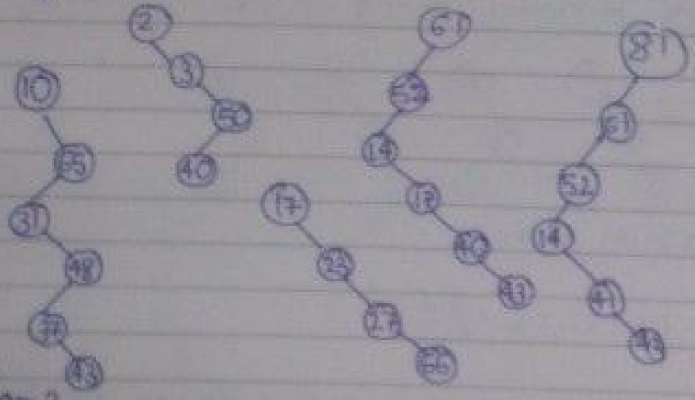
Q. A set of n-distinct elements & an unlabeled binary tree with n-nodes.
In how many ways we can populate the tree with the given set ~~set~~ of elements so that it becomes BST.

~~(A)~~ 1
(B) 0
(C) n!
(D) $\frac{^{(2n)}C_n}{(n+1)}$

→ My answer:- because only 1 tree is given. ∴ with 1 tree, we can arrange the nodes in only 1 way so that it becomes a BST

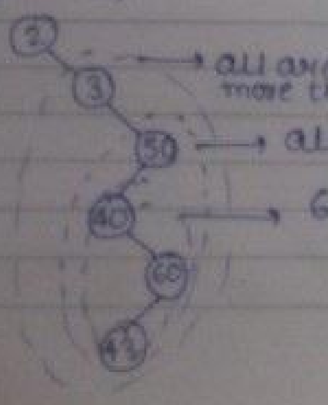
Q. A binary search tree is used to locate no. 43, then which of the following sequence is not possible?

(A) 61, 52, 14, 17, 40, 43)
(B) 2, 3, 50, 40, 60, 43
(C) 10, 65, 31, 48, 37, 43
(D) 81, 61, 52, 14, 41, 43
(E) 17, 22, 27, 66, 18, 43



To check :-

(B)



→ all are more than 2

→ all are more than 3

→ 60 is greater than 50.

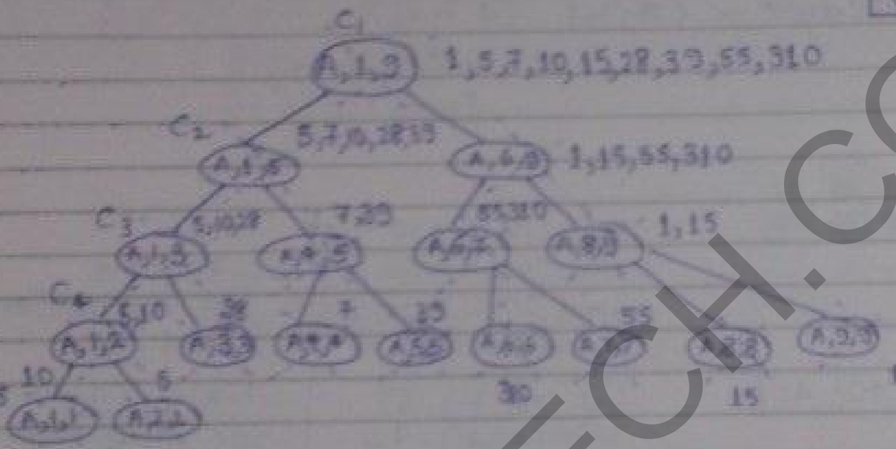
Merge Sort

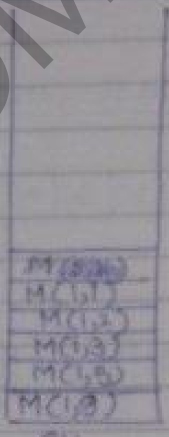
Note: Merging 2 sorted subarrays.

ex.:-

A [10 5 28 7 39 310 55 15 1]

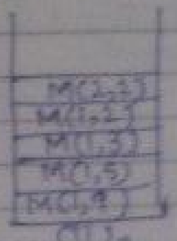
* Max. stack size = no. of levels in th
tree.
So the levels & he
are calculated t
measure the stac
size





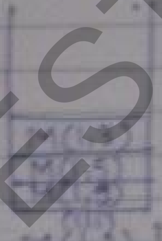
(i)

- M(1,1) is successfully completed.



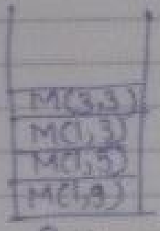
(ii)

Now M(2,2) is successfully completed.

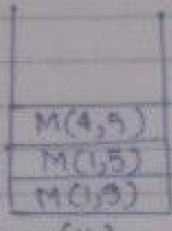


(iii)

M(1,2) is completed because its both children are completed.



(iv)



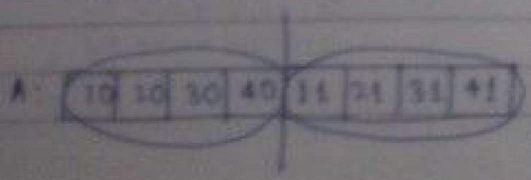
(v)

M(3,3) is complet
& ∴ M(1,3) is com
because its both
children are suc
completed.

* Space Complexity = i/p size + stack size. → i/p size + 5n
no. o
le
(n - each
space

★ Time complexity = sum of all function calls.
Time complexity = 17 * (function call).
↳ no. of function calls.

Merge procedure



- sorting ~~with~~ using the same array :- inplace sorting.
- Sorting using the diff. array :- Outplace sorting.

i/p to merge-procedure → M (a, i → mid, mid+1 → j)

↓ sorted array from i to mid

↓ sorted array from mid+1 to j.

o/p of merge-procedure → ~~[illegible]~~ (a, i → j)

↓ sorted array from i to j

Worst-Case:-

					11	21	31	41
A:	10	20	30	90	[illegible]	[illegible]	[illegible]	[illegible]

B:	10	11	20	21	30	31	40	41

min(10,11) = 10
min(20,11) = 11
min(20,21) = 20
min(30,21) = 21
min(30,31) = 30
min(40,31) = 31
min(40,41) = 40
41

→ Total no. of comparisions
4 + 4 − 1 = 7

Worst case :-
Time complexity:-
$m+n-1$ { → m: da... of ... } n: ...
$\boxed{O(m+n)}$
When both the arra... size is equal -
$\frac{n}{2}+\frac{n}{2}-1 \Rightarrow n-1 \rightarrow O$

Best Case :-

A:	10	20	30	40	1	5	7	9
	1	2	3	4	5	6	7	8

p_1 p_2

B:	1	5	7	9	10	20	30	40

min.(10,1) = 1
min(10,5) = 5
min(10,7) = 7
min(10,9) = 9
10
20
30
40

→ **Best Case**
Time complexity:-
no. of comparisons = ...
if 1st part of array has size m
& 2nd part of array has size n
∴ Best case complexity
O[min(m,n)].
If both are of $\frac{n}{2}$ size
then:- $\frac{n}{2}+\frac{n}{2}$
$\boxed{\Omega(n)}$
↳ it can't go less than this
(as we have to atleast read all the n-elements.)

$\frac{n}{2} \geq cn$
$\frac{n}{2} \leq cn$

If we ~~use~~ don't use the second array B, then we have to re-sort the two array parts of A which increases the time complexity.

Merging Algorithm :-

```
Merge (a, i, mid, mid+1, j)
{ k = mid+1 ; P=1
   while (i ≤ mid && k ≤ j)
   { if (a[i] < a[k])
        { B[P] = a[i];
          i++;
          P++;
        }
        else
        { B[P] = a[k];
          k++;
          P++;
        }
   }
   for ( ; i < mid; i++)
     { B[P] = a[i];
       P++;
     }
   for ( ; k ≤ j; k++)
     { B[P] = a[k];
       P++;
     }
   for (k=1; k<j; k++)
       a[k] = b[k];
   }
}
```

Worst Case :-

$T(n) = n-1 = O(n)$

Best Case :-

$T(n) = \frac{n}{2} = \Omega(n)$

Average Case :-

$\Theta(n)$

$$T(n) = \begin{cases} O(1), & \text{if } n=1 \\ T(\frac{n}{2}) + T(\frac{n}{2}) + O(n) & \text{oth} \end{cases}$$

$$= 2T(\frac{n}{2}) + O(n)$$

$$= 2T(\frac{n}{2}) + n$$

Merge - Sort

```
MergeSort (a, i, j)
{ if (i==j)
     return (a[i]);        ⟶ O(1)
  else
  { mid = (i+j)/2;         ⟶ O(1)
T(n/2) ⟶ MergeSort (a, i, mid);
T(n/2) ⟶ MergeSort (a, mid+1, j);
     Merge (a, i, mid, mid+1, j);  ⟶ return (a);
  }
}
```

Cosmos

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

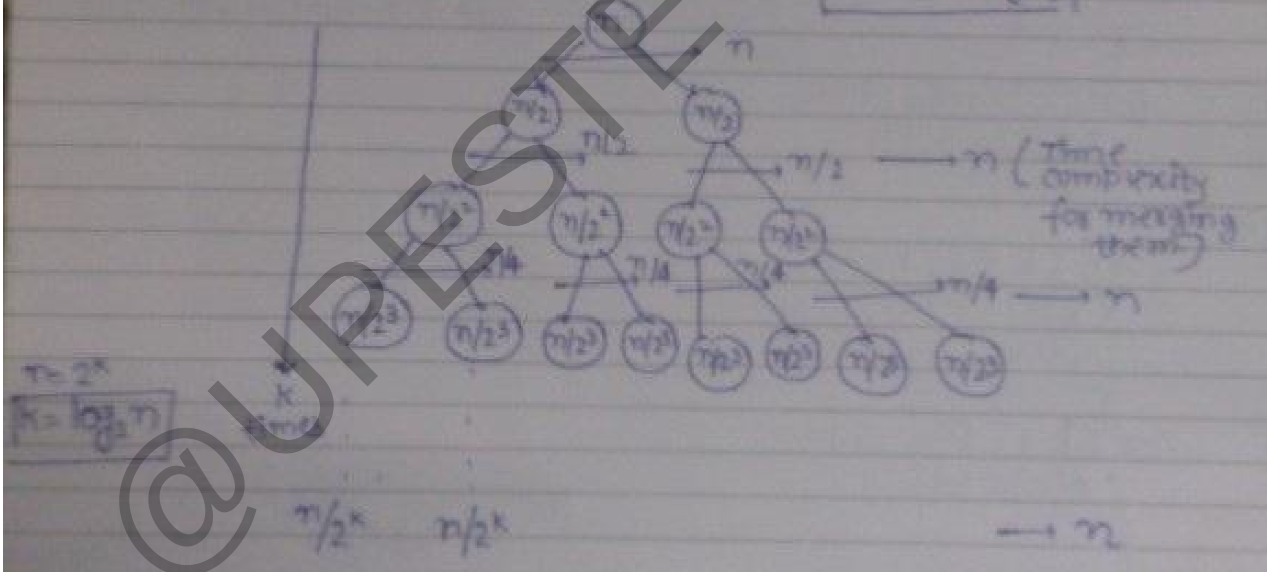
$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$= 2\left[2\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + \frac{n}{2}\right] + n$$

$$= 2\left[2^2T\left(\frac{n}{8}\right) + \frac{n}{2} + \frac{n}{2}\right] + n$$

$$= 2^3T\left(\frac{n}{8}\right) + 2n + n \qquad 2^kT\left(\frac{n}{2^k}\right) + kn$$

$$\therefore \boxed{T(n) = \Theta(n\log_2 n)} \text{ put } k = \log_2 n$$



$\therefore$ no. of levels = $k = \log_2 n$

& at each level the time complexity = n.

$\therefore$ Time Complexity = $n\log_2 n$

★ Merge-Sort is not inplace sorting technique because in the merging procedure we are taking an extra array of size n.

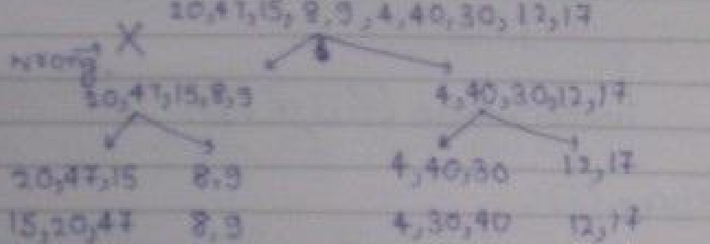
no. of variable size in each function

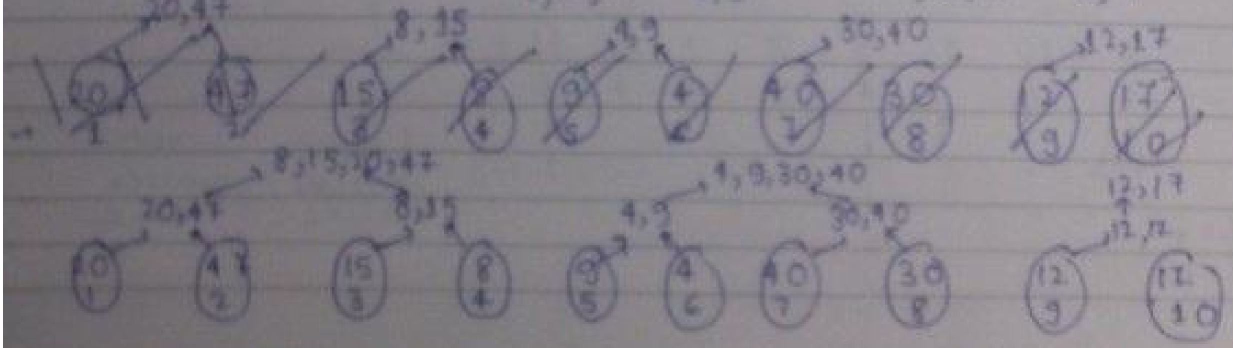
$$\text{Space complexity} = 2n + |c\log_2 n| + 2n \Rightarrow \boxed{O(n)}$$

array size of n+n
A
[n-elements, each element of 2B, ∴ total size = 2n B]
array size of A
array size of B
Stack size
no. of function calls (from the tree of prev. page.)
array size of B

Note:- If the array size is very small, merge sort is not advisable (insertion sort is advisable), so for larger size arrays merge sort is advisable.

Q. If one uses straight 2-way merge sort algo to sort following elements.
20, 47, 15, 8, 9, 4, 40, 30, 12, 17.
then the order of these elements after 2nd pass of the algo. is
(A) 8, 9, 15, 20, 47, 4, 12, 17, 30, 40
✓(B) 8, 15, 20, 47, 4, 9, 30, 40, 12, 17
(C) 15, 20, 47, 4, 8, 9, 12, 30, 40, 17
(D) None.





★ Quick sort is used to find out the n-th smallest element in the array.

★ In straight 2-way merge-sort

- We combine & sort 1^{st} & 2^{nd} element, then 3^{rd} & 4^{th}, then 5^{th} & 6^{th}, & so on. [in 1st pass.]
- In 2nd pass we merge 1^{st} 2^{nd} & 3^{rd} 4^{th} element & sort them, then 5^{th} 6^{th} 7^{th} 8^{th}, & so on.

→ In above example, no. of levels required to sort 1 to 4

→ ∴ no. of levels = $\log_2 n$

& each level take n time

∴ complexity is $n\log_2 n$.

Quicksort :- [Tony Hoare]

< Randomized Quicksort (when pivot element is choosen randomly).

< Normal Quicksort (when pivot element is choosen from same position.)

① It uses Divide & Conquer Algorithm.
② In-place sorting algo.
③ Not stable.

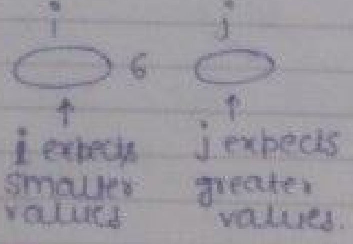
```
Partition(a,p,q)
{  x = a[p];  i = p;
   for(j = p+1 ; j ≤ q ; j++)
     {  if(a[j] ≤ x)
          { i = i+1;
            interchange(a[i],a[j]);
          }
     }
     Interchange (a[i],a[p]);
}
return (i);
}
```

[i] → 6 10 13 5 8 3 2 11

→ 1 2 3 4 5 6 7 8

= 6

= p;



	6	5		10				
a[i] →	6	~~10~~	13	~~5~~	8	3	2	11
i →	1	2	3	4	5	6	7	8
		↑ i		↑ j	↑ j	↑ j		

			3			13		
a[i] →	6	5	~~13~~	10	8	~~3~~	2	11
i →	1	2	3	4	5	6	7	8
			↑ i			↑ j	↑ j	

				2			10	
a[i] →	6	5	3	~~10~~	8	13	~~2~~	11
i →	1	2	3	4	5	6	7	8
				↑ i			↑ j	↑ j

	2			6				
a[i] →	~~6~~	5	3	~~2~~	8	13	10	11
i →	1	2	3	4	5	6	7	8
	↑ p			↑ i				

exchange (a[i],a[

→ smaller than 6 → greater than 6

(2 5 3) 6 (8 13 10 11)

↳ pivot element

• Apply partition algo in the following elements:-

a[i] →	65	70	75	80	85	60	55	50	45
i →	1	2	3	4	5	6	7	8	9
	↑ i	↑ j	↑ j	↑ j	↑ j	↑ j			

x = 65

a[i] →	65	60	75	80	85	70	55	50	45
i →	1	2	3	4	5	6	7	8	9
		↑ i					↑ j		

a[i] →	65	60	55	80	85	70	75	50	45
i →	1	2	3	4	5	6	7	8	9
			↑ i					↑ j	

a[i] →	65	60	55	50	85	70	75	80	45
i →	1	2	3	4	5	6	7	8	9
				↑					↑
				i					j

a[i] →	65	60	55	50	45	70	75	80	85
i →	↑				↑				
	p				j				

a[i] →	45	60	55	50	65	70	75	80	85
i →	1	2	3	4	5	6	7	8	9

★ As there is no combination, it is not divide & conquer, it is partial divide & conquer.

a[i] →	25	57	48	38	11	90	89	29
i →	1	2	3	4	5	6	7	8
	↑ ↑	↑	↑	↑	↑			
	i p	j	i	j	j			

	⑪	25						
a[i] →	~~25~~	~~11~~	48	38	57	90	89	29
i →	1	2	3	4	5	6	7	8
	↑	↑				↑	↑	↑
	p	i				j	j	j

(40 70 10 20 60 50 30)

```
Quicksort(a,p,q)
{if (p=q)
    return(a[p]);
 else
  {
   m=partition(a,p,q);   ⟶ O(n) ⟶ n
   Quicksort (a,p,m-1);  ⟶ ~~T(n/2)~~  T(n/2)
   Quicksort (a,m+1,q);  ⟶ ~~T(n/4)~~  T(n/2 -1) ✓
   return(a);
  }
}
```

- ★ To partition in Merge sort → mid = $\frac{l+r}{2}$ → $\boxed{O(1)}$
- ★ To partition in Quick sort → m = partition(a, p, q) → $\boxed{O(n)}$

Merge sort → no reln. b/w 1 & 2

Quick sort → there is reln. b/w 1 & 2
all values in 2 > all values in 1.

★ Let T(n) be the amount of time req. to sort n elemen using Quicksort, then T(n)

$$T(n) = \begin{cases} O(1), & n=1 \\ \text{[illegible]} \; n + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}-1\right) & \text{if } n>1. \end{cases}$$

T(n) (Best case)

$$T(n) = \begin{cases} O(1), & n=1 \\ 2T\left(\frac{n}{2}\right) + \underset{\text{partition time}}{n}, & n>1 \end{cases} \longrightarrow \Omega(n\log_2 n)$$

occurs when everytime the partition algo divides the elements into equal groups (having same no. of element)

partition time (this is merge time in merge sort)

<u>Worst Case:</u>

In worst case the partition will create 0 elements on one side & (n-1) elements on other side.

∴

$$T(n) = \begin{cases} O(1), & n=1 \\ n + T(0) + T(n-1) & \text{if } n>1 \\ = n + T(n-1) \\ = T(n-1) + n & \longrightarrow O(n^2) \end{cases}$$

n + n-1 + n-2 + ...

$\leq n + n + \ldots +$

$\leq n \cdot n$

$\leq n^2$

$O(n^2)$

★ When array is not at all sorted, then Quicksort is the best sort.

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}-1) + 2n$

$= 2T(\frac{n}{2}) + n$

$= n\log_2 n$

Average Case:-

$T(n) = 2T(\frac{n}{2}) + n \longrightarrow$ Lucky Case

$T(n) = T(n-1) + n \longrightarrow$ Unlucky Case

$T(n) = 2T(\frac{n}{2}) + n$

$= 2[T(\frac{n}{2}-1) + \frac{n}{2}] + n$ [Lucky followed by unlucky]

$= 2T(\frac{n}{2}-1) + 2n$

$\approx 2T(\frac{n}{2}) + n \longrightarrow \theta(n\log_2 n)$

↳ which is similar to Best Case

$T(n) = T(n-1) + n$

$= 2T(\frac{n-1}{2}) + n-1 + n$

$= 2T(\frac{n-1}{2}) + 2n - 1$ → constants can't change the complexity.

$= 2T(\frac{n}{2}) + n$

) Quicksort is run on 2 i/ps shown below to sort in ascending order

) $1, 2, 3, 4, \ldots, n$

ii) $n, n-1, n-2, \ldots, 1$

et C_1 & C_2 be the no of comparisons made for /p's (i) & (ii) respec., then

) $C_1 < C_2$ (B) $C_1 > C_2$ [✓ (C)] $C_1 = C_2$ (D) Not comparable

★ If array is almost/already sorted then insertion sort is best sort [O(n)].

(i) 10 20 30 40 50 60 ⟹ 6 comparisons ~~(actually 5, but we take~~
1 2 3 4 5 6
↑ ↑ ↑ ↑ ↑ ↑
i j j j j j

(ii) 10 20 30 40 50 60 ⟹ 5 comparisons (actually 4, but we 5).
↑ ↑ ↑ ↑ ↑
i j j j j

6+5+4+3+2+1 = $\frac{6\times7}{2}$ = 21

T(6) = T(5) + 6 → 6 comparisons in 1st partition
= T(4)+6+5
= T(3)+6+5+4
= T(2)+6+5+4+3
= T(1)+6+5+4+3+2
= 1+2+3+4+5+6
→ depth of stack is 'n'
→ worst case.

n / n-1 / n-2

2nd 60 50 40 30 20 10
↑ ↑ ↑ ↑
i j j i

(10) ¦ 50 40 30 20 60 60 → n swaps in worst case (1 swap in best case)
↑ ↑ ↑
i j j

10 ¦ 50 20 30 40 60
↑ ↑ ↑
p i

10 p (20) (50 30 40 60)

★ 1st time partition → 6 comparisons
2nd " → 5 "
3rd " → 4 "
⋮
1 "

∴ T(6) = T(5) + 6 = T(4)+6+5 = $\boxed{O(n^2)}$

★ In descendin
order, the par
ns will contai
(n-1) & 0 elem
which given O(
time complexi

★ Quicksort gives worst case when array is sorted (whether in descending or ascending).

★ Quicksort is best sort even though its worst case is $O(n^2)$ because the worst case i.e. (when the array is already sorted) never happens since we never sort an already sorted array.

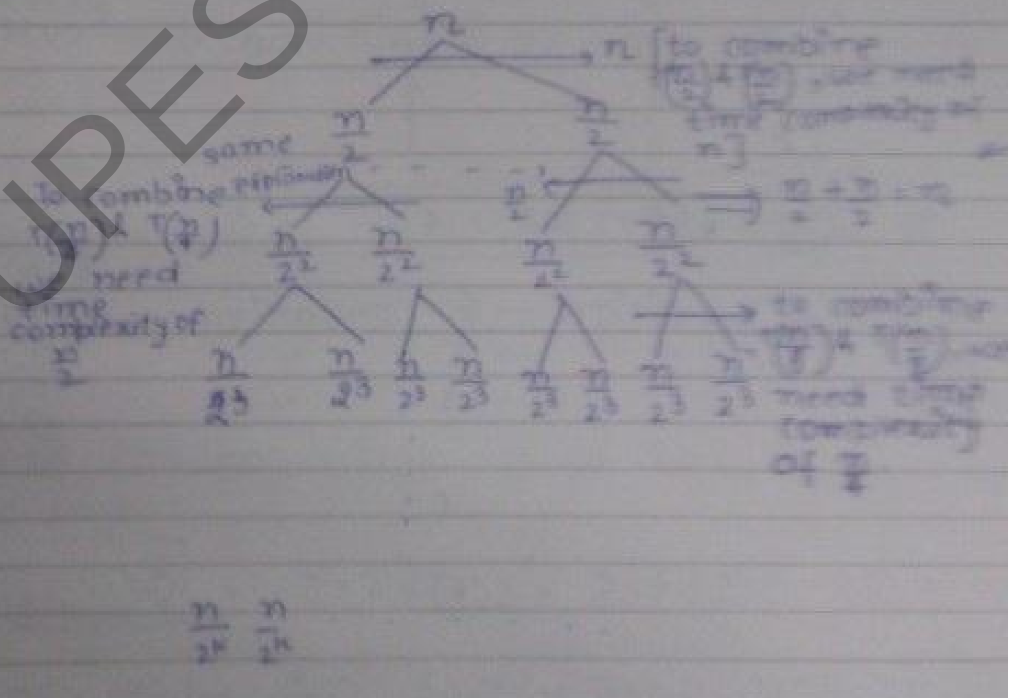
★ After applying few passes of quicksort ~~[illegible]~~ on the given array we got following o/p.

1 10 5 8 25 44 55 30 70

then how many pivot elements are there in above o/p

Ans. ⑤

maximum 3 times the partition algo was applied. it may be 1,2 or 0 because [0 → when array is like that only.



Date

10.06.12

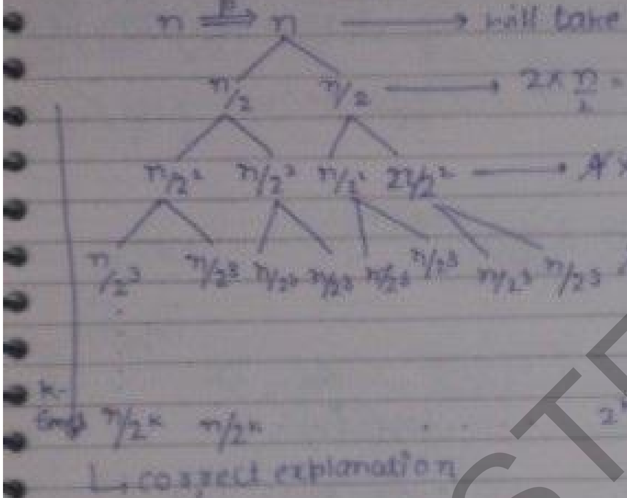
Recursive Tree Method :-

$T(n) = 2T(\frac{n}{2}) + n$

$= T(\frac{n}{2}) + T(\frac{n}{2}) + n$ → combination ~~[illegible]~~ algo

[method is useful when there ~~[illegible]~~ are 3 ~~methods~~ elements]

$n \xrightarrow{k} n$ → will take n times



$\rightarrow 2 \times \frac{n}{2} = n$

$T(\frac{n}{2})$ makes $T(\frac{n}{4}) + T(\frac{n}{4}) + (\frac{n}{2})$

Other $T(\frac{n}{2})$ makes $T(\frac{n}{4}) + T(\frac{n}{4}) + (\frac{n}{2})$

these $\frac{n}{2}$'s combine to give n.

$\rightarrow \cancel{A}4 \times \frac{n}{2^2} = n$ [4 subproblems]

$8 \times \frac{n}{2^3} = n$ [8 subproblems]

$2^k \rightarrow$ n should be equal to 2^k

$n = 2^k \rightarrow k = \log_2 n$

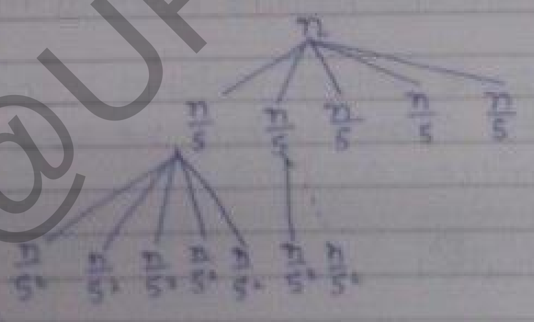
[because in the end each ~~[illegible]~~ subproblem should contain only one element]

~~[illegible]~~ $n \times k = \boxed{n \times \log_2 n}$

$O(n \log_2 n)$

↳ correct explanation

Q $T(n) = 5T(\frac{n}{5}) + n$



$\rightarrow 5 \times \frac{n}{5}$

$\rightarrow 26 \times \frac{n}{5^2}$

$5^k = n$

$k = \log_5 n$

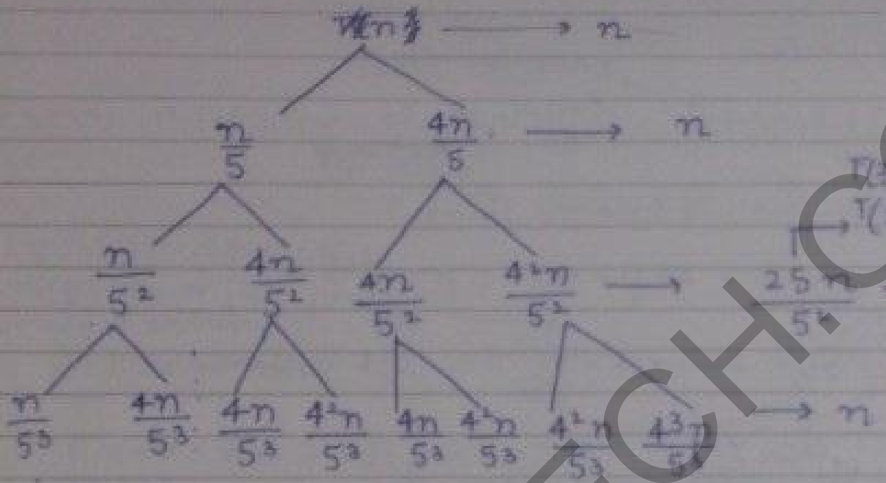
$T(n) = O(n \log_5 n) + O(n)$

$= O(n \log_5 n)$ [max. of $O(n \log$ & $O(n)$]

because both sides have same height $\longrightarrow = \Theta(n \log_5 n)$

Cosmos

Q. $T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + n$



$T(\frac{n}{5}) = T(\frac{n}{5^2}) + T(\frac{4n}{5^2}) + \frac{n}{5}$

$T(\frac{4n}{5}) = T(\frac{4n}{5^2}) + T(\frac{4^2n}{5^2}) + \frac{4n}{5}$

$\frac{n}{5} + \frac{4n}{5} = n$

$n = 5^3$

$n = 125$

$n = 5^k$

$\log_5 n = k$

~~$4n = 5^k$~~

~~$\log_5(4n) = k$~~

~~$O(n \log_5 (4n))$~~

$4^k n = 5^k$

$n = (\frac{5}{4})^k$

$(\log_{5/4} n) = k$

$\frac{4^3 n}{5^3} = \frac{n}{(5/4)^3}$

dividing by 5/4 will create more levels, so $n/(5/4)^k$ creates most no. of levels, ∴ we for max. no. of levels we equate n with $(\frac{5}{4})^k$ $\boxed{\text{My answer: } O(n \log_{5/4} n)}$

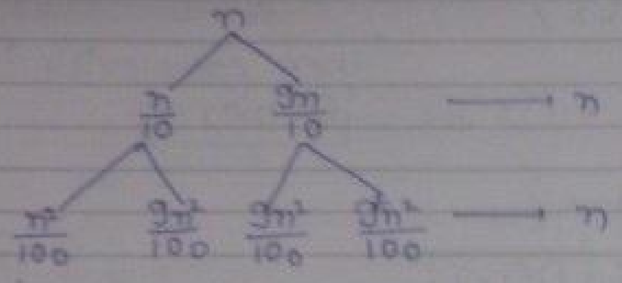
★ $n/5^3$ will stop somewhere in the middle, whereas $n/(5/4)^k$ will go till the end.

★ after some no. of levels, some values vanishes & the partition cost will be less than n, but we take O(n), as less than n can be written as O(n).

Sir's answer → $\boxed{O(n \log_{5/4} n)}$

Q. $T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + n$

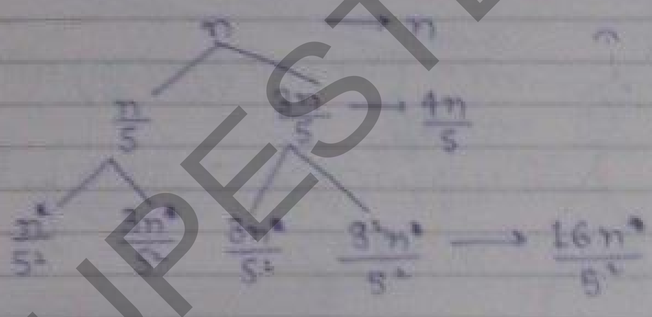
Similar to previous one

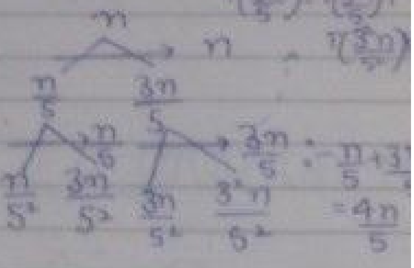


Complexity :- $\boxed{O(n \log_{(10/9)} n)}$

Q. $T(n) = T(\frac{n}{5}) + T(\frac{3n}{5}) + n$ $T(\frac{n}{5}) = T(\frac{n}{5^2}) + T(\frac{3n}{5^2}) + \frac{n}{5}$

$T(\frac{3n}{5}) = T(\frac{3n}{5}) + T(\frac{3n}{5^2})$





$\frac{4^3 n}{5^3}$

$n + \frac{4n}{5} + \frac{4^2 n}{5^2} + \dots + \frac{4^k n}{5^k}$ $\log_{5/3} n = k$

$\frac{n[1((\frac{4}{5})^{\log_{5/3} n} - 1)]}{\frac{4}{5} - 1} = 5n[1 - (\frac{4}{5})^{\log_{5/3} n}]$

→ this is less than 1

∴ Complexity :- $5n[1 - (\frac{4}{5})^{\log_{5/3} n}]$ m

$\le 5n \longrightarrow \boxed{O(n)}$

Cosmos

Note:-

If $T(n) = T\left(\frac{n}{a}\right) + T\left(\frac{n}{b}\right) + n$ if $\frac{n}{a} + \frac{n}{b} = n$

then complexity is $\boxed{O(n\log n)}$

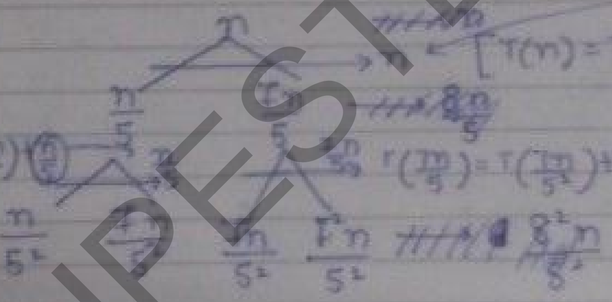
but if $\frac{n}{a} + \frac{n}{b} < n$

then complexity is $\boxed{O(n)}$

Q. $T(n) = 8T\left(\frac{n}{2}\right) + n^2$ — $O(n^2\log n)$

Q $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{5}\right) + n$

this means problem is increasing stepwise



$\rightarrow n \leftarrow \left[T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{5}\right) + \boxed{n}\right]$

$\rightarrow \frac{8n}{5}$

$T\left(\frac{n}{5}\right) = T\left(\frac{n}{5^2}\right) + T\left(\frac{7n}{5^2}\right) + \boxed{\frac{n}{5}}$

$T\left(\frac{7n}{5}\right) = T\left(\frac{7n}{5^2}\right) + T\left(\frac{7^2 n}{5^2}\right) + \frac{7n}{5}$ $\rightarrow \frac{n}{5} + \frac{7n}{5} = \frac{8n}{5}$

$\rightarrow \frac{8^2 n}{5^2}$

$2^{\log_2 n}$

$n^{\log_2 2}$

$= n$

$n + \frac{8n}{5} + \frac{8^2 n}{5^2} + \ldots + \frac{8^k n}{5^k}$

$\frac{5^k}{8^k} = n$

$k = \log_{5/8} n$

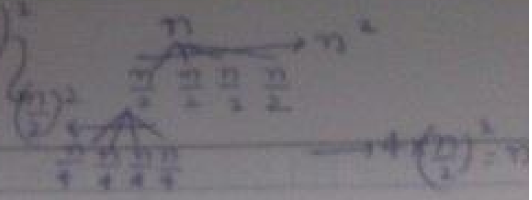
$= \frac{n\left[\left(\frac{8}{5}\right)^{\log_{5/8} n} - 1\right]}{\frac{3}{5}} = \frac{5n}{3}\left[\left(\frac{8}{5}\right)^{\log_{5/8} n}\right]$

$\approx n \times \left(\frac{8}{5}\right)^{\log_{5/7} n} \approx n \times n^{\log_{5/7} 8/5}$

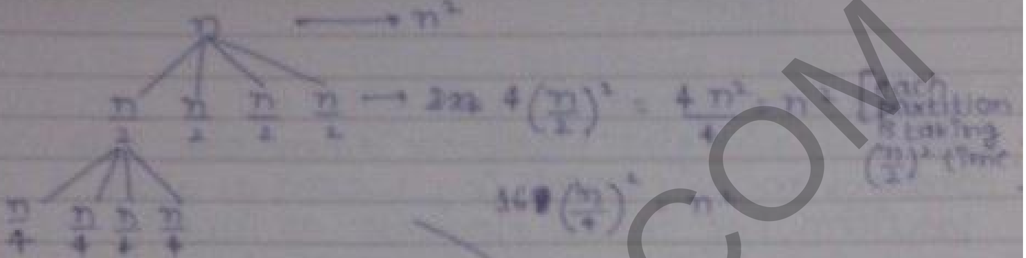
$\therefore$ complexity is $\boxed{O(n \times n^{\log_{5/7} 8/5})}$

$T(\frac{n}{2}) = 4T(\frac{n}{4}) + (\frac{n}{2})^2$

$T(n) \sim$ ~~[illegible]~~



Q. $T(n) = 4T(\frac{n}{2}) + n^2$



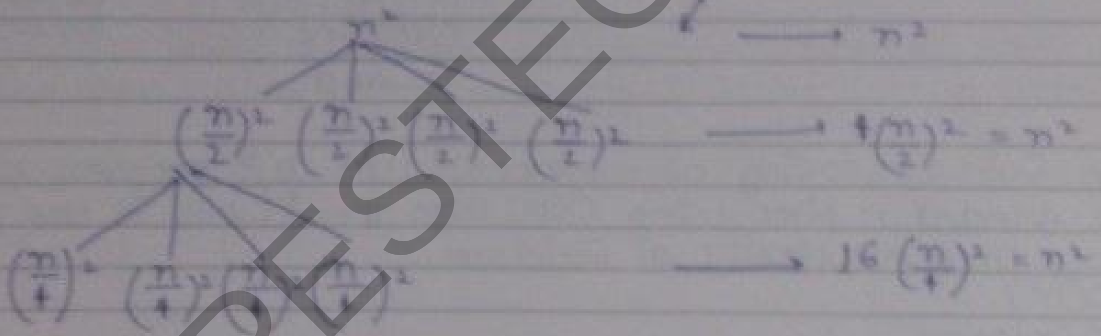
$\longrightarrow n^2$

$\longrightarrow 2n\ 4(\frac{n}{2})^2 = \frac{4n^2}{4} = n^2$ [each partition is taking $(\frac{n}{2})^2$ time]

$16(\frac{n}{4})^2 = n^2$

no. of levels $\rightarrow n = 2^k \rightarrow k = \log_2 n$

$\therefore$ Complexity $\boxed{O(n^2 \log_2 n)}$



$\longrightarrow n^2$

$\longrightarrow 4(\frac{n}{2})^2 = n^2$

$\longrightarrow 16(\frac{n}{4})^2 = n^2$

Quicksort:-

★ Quicksort will give the best case even when a single element is on one side of partition & even crores of element on other side

$T(n) = T(n\alpha) + T(n - n\alpha) + n$ [Best Case]

$0 < \alpha < 1$ [not equal, because if $\alpha = 0$ or 1 then it will give the worst case]

★ $T(n) = T(10) + T(n-10) + n$

↓

$O(n^2) \rightarrow$ worst case [as n is very large & hence (n-10) won't make any effect]

↓

no. of levels $\rightarrow \frac{n}{10}$ & each level complexity is n

$\frac{n^2}{10} \rightarrow O(n^2)$

$T(n) = T(1{,}00{,}000) + T(n - 1{,}00{,}000) + n$

$\boxed{O(n^2)}$

★ **Worst-Case:-**
Quicksort will give worstcase when one partition has constant no. of elements (e.g. 1,00,000 as above) & other partition has no. of elements as function of n.

★ **Best Case** will happen when both partition will have no. of elements as function of n.

Worst-Case:- $\boxed{T(n) = T(c) + T(n-(c+1)) + n}$

c:- constant
c+1:-1 because of pivot element (it wont be taken in the partition)

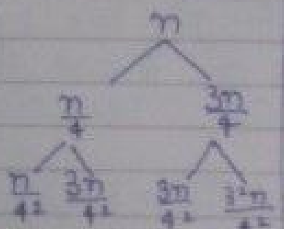
but as +1 in (c+1) won't affect the answer, ∴ we can write it as T(n-c).

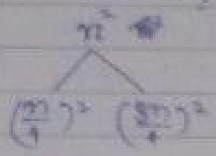
Q. In Quicksort, sorting of n-elements, the $\left(\frac{n}{4}\right)^{th}$ smallest element is selected as pivot using O(n) algo, then what is the worst case time complexity of Quicksort.

(a) $O(n^2)$
(b) $O(n\log n)$
(c) $O(n)$
(d) $O(n^3)$

Ans.

$\boxed{O(n\log_{4/3} n)}$





$O(n^2)$ $\frac{n^2}{16} + \frac{9n^2}{16} = \frac{10n^2}{16}$

$n^2 + \frac{10n^2}{16}$

$5n^2 + n^2 + n^2$

$(n^2 \log n)$

$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \underbrace{O(n)}_{\text{place pivot at correct position}} + \underbrace{O(n)}_{\text{selecting pivot}}$

acc. to the ques.

[when

$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2n$

If we have choosen n^{th} smallest element :-
then it is worst case.

Q. The median of n-elements can be found using O(n)
algo which one of the following is correct about
complexity of quicksort if median is selected as pivot

(A) $O(n^2)$
(B) $O(n\log n)$
(C) $O(n)$
(D) $O(n^3)$

Median :- The middle element of a sorted array.
this means $\left(\frac{n}{2}\right)^{th}$ smallest element.

Stable Sorting Technique :-

The relative order of repeated elements not changed after sorting, then sorting technique is called Stable Sorting Techni

e.g.

$a[i] \rightarrow 10_a, 11, 20, 10_b, 40, 50, 60$
$i \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7$
$\uparrow i \quad \uparrow j \quad \uparrow j \quad \uparrow j$

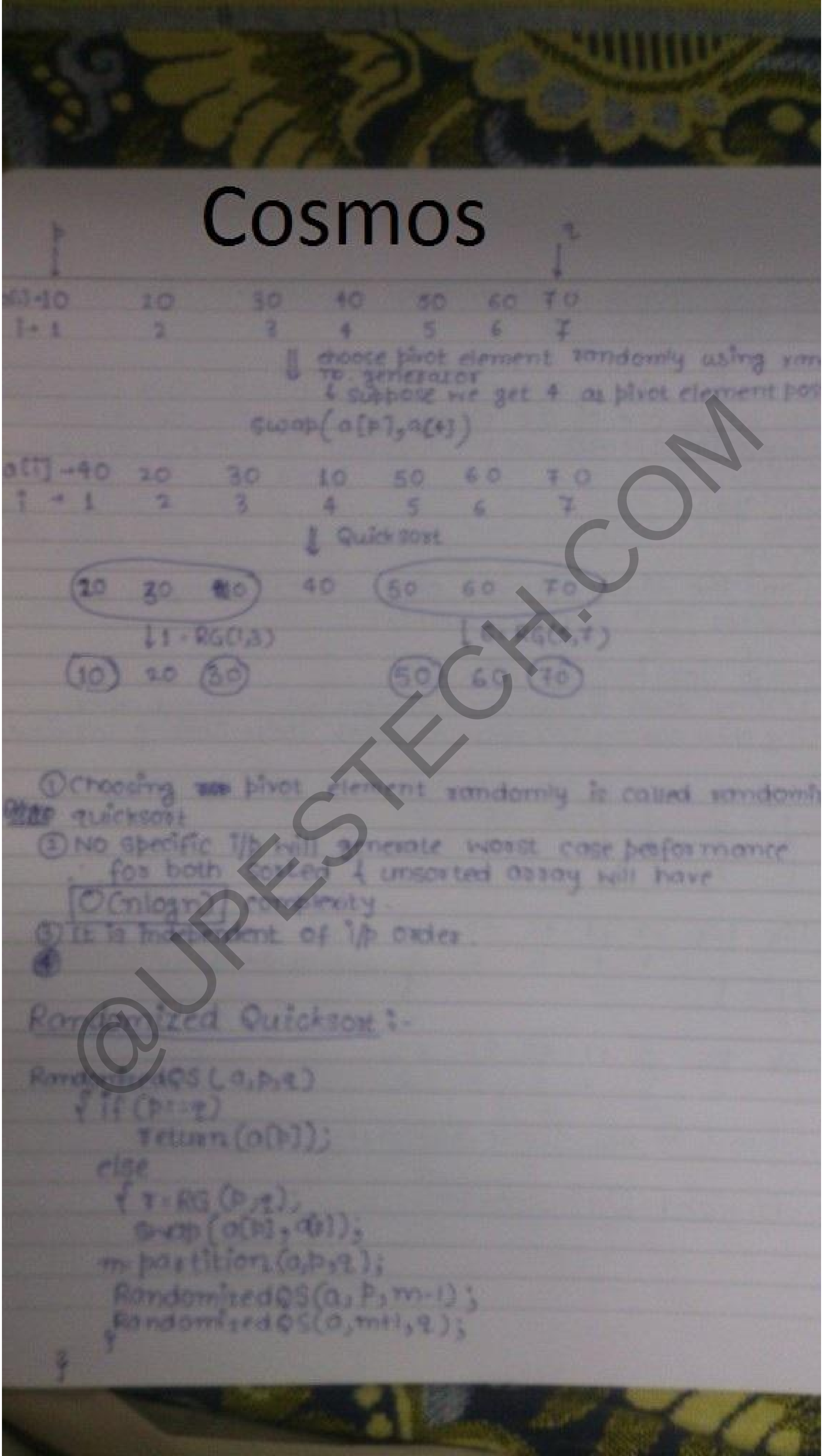
$a[i] \rightarrow 10_a \quad 10_b \quad 20 \quad 11 \quad 40 \quad 50 \quad 60$ now, interchange
$i \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7$ a[i] & a[p]
$\uparrow i \quad \uparrow \quad \uparrow j$

$a[i] \rightarrow 10_b \quad 10_a \quad 20 \quad 11 \quad 40 \quad 50 \quad 60$
$i \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7$

★ Quicksort is not stable, but Mergesort is stable.

Randomized Quicksort :-

Cosmos



p ↓ q ↓

a[i] → 10 20 30 40 50 60 70
i → 1 2 3 4 5 6 7

⇓ choose pivot element randomly using ran
To. generator
& suppose we get 4 as pivot element pos

swap(a[p], a[4])

a[i] → 40 20 30 10 50 60 70
i → 1 2 3 4 5 6 7

↓ Quick sort

(20 30 10) 40 (50 60 70)

↓ 1 - RG(1,3) ↓ 6 - RG(5,7)

(10) 20 (30) (50) 60 (70)

① Choosing ~~the~~ pivot element randomly is called randomi
~~the~~ quicksort
② No specific i/p will generate worst case performance
for both sorted & unsorted array will have
$\boxed{O(n\log n)}$ complexity.
③ It is independent of i/p order.
④

Randomized Quicksort :-

```
RandomizedQS(a,p,q)
 { if (p==q)
     return(a[p]);
   else
    { r=RG(p,q);
      swap(a[p], a[r]);
   m=partition(a,p,q);
    RandomizedQS(a, p, m-1);
    RandomizedQS(a, m+1, q);
   }
 }
```

	Quicksort	Randomized Quicksort
Best:	nlogn	nlogn
Avg.:	nlogn	nlogn
Worst:	n^2 (almost sorted) 99% its not going to happen	n^2 (when all the elements a same) 99.99999% its not going t happen.

Selection Procedure:-

i/p:- An array of n-element & integer k.

o/p:- Selecting k^{th} smallest element

e.g.	40	70	30	60	10	50	30	k=4
	1	2	3	4	5	6	7	

Algo-1

① Find 1st element & delete.
② " 2^{nd} " " "
③ " 3^{rd} " " "
④ " k^{th} " & return it

$\Rightarrow O(kn)$ when k=1
when k=n O(n)→best
$O(n^2)$ → worst case

Algo-2

① Sort the array ⟶ nlogn
② Return a[k] ⟶ O(1)

Best & worst case → O(nlogn

Algo-3

40	70	20	60	10	50	30
1	2	3	4	5	6	7

```
Selectionprocedure (a,i,j,k)   → kth smallest element
{ if (i==j)
      { if (i==k)
            return (a[i]);
        else
            return (-1);
      }
  else{ m=partition (a,i,j);
        if (m==k)
            return (a[m]);
        else
        { if (m<k)
              Selectionprocedure(a,
          else
              Selectionprocedure(a,i,
        }
      }
}
```

Let T(n) be the amount of time req., then

$$T(n) = \begin{cases} \Omega(n) \rightarrow \text{best case} \\ O(n\log n) \rightarrow \text{worst case} \end{cases} \longrightarrow \text{My answer}$$

Sir's answer:-

$$T(n) = \begin{cases} O(1), & n = 1 \\ n + 2 + T(\frac{n}{2}) & \text{if } n > 1 \end{cases}$$

time for partition

$$= T(\frac{n}{2}) + n$$

$$= \Omega(n)$$

Best case (when the partition algo creates "almost" equal no. of elements on both sides.)

Worst Case:-

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ n + 2 + T(n-1) & \text{if } n > 1 \end{cases}$$

time for partition

$$= T(n-1) + n$$

$$= O(n^2)$$

→ Worst case (when the partition algo creates partition when one partition contain "constant" no. elements.)

Strassen's Matrix Multiplication :-

① Without Divide & Conquer :-

$A_{n \times n}$, $B_{n \times n}$

1. $C_1 = A + B \longrightarrow O(n^2)$
2. $C_2 = AB \longrightarrow O(n^3)$

C_1 :- To select one element it will take $O(n^2)$ & to add two elements it will take $O(1)$

∴ complexity :- $O(n^2)$

C_2 :- To get the element in final matrix → $O(mnp)$

$\downarrow$ $m \times n$ $n \times p$

② Using Divide & Conquer :-

(i) If the given two matrices are $\leq 2 \times 2$, then the problem is small.

(ii) Only for square matrices.

(iii) The size of square matrices should be powers of 2.

endian

$$A = \begin{array}{c} \begin{array}{cc} A_{11} & A_{12} \end{array} \\ \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 17 \end{array}\right]_{4\times4} \\ \begin{array}{cc} A_{21} & A_{22} \end{array} \end{array} \qquad B = \begin{array}{c} \begin{array}{cc} B_{11} & B_{12} \end{array} \\ \left[\begin{array}{cc|cc} a & b & c & d \\ e & f & g & h \\ \hline i & j & k & l \\ m & n & o & p \end{array}\right]_{4\times4} \\ \begin{array}{cc} B_{21} & B_{22} \end{array} \end{array}$$

① divide the size of matrices by 2 till we reach the matrix size of 2×

(e.g. the above is divided till we get 2×2)

e.g. $T(16) = T(8)$
$= T(4)$
$= T(2)$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}_{4\times4}$$

~~C_{11} $\begin{bmatrix} 1\times a+2\times e & 1\times b+3\times f \\ 5\times a+6\times e & 5\times b+6\times f \end{bmatrix}$~~

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$
$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$
$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$
$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

8 multiplications of matrices 2×2

$$T(4) = \underbrace{8\,T\left(\frac{4}{2}\right)}_{} + \underbrace{4}_{} \times \cancel{(2\times2)} \left(4/2\right)^2$$

2×2 mat

4 additions of matrices of sizes 2×2

$$T(8) = 8\,T\left(\frac{8}{2}\right) + 4\times\left(\frac{8}{2}\right)^2$$

$$T(n) = \begin{cases} O(1), & \text{if } n \le 2\times2 \\ 8\,T\left(\frac{n}{2}\right) + 4\times\left(\frac{n}{2}\right)^2 \end{cases}$$

$$T(16) = 8\,T\left(\frac{16}{2}\right) + 4 \times \left(\frac{16}{2}\right)^2$$

8 matrix multiplications of size 8×8

4 matrix additions of size 8×8

each addition takes $O(n^2)$ complexity ∴ a 8×8 matrix will take $(8)^2$

$$T(n) = 8\,T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = 8\,T\left(\frac{n}{2}\right) + n^2 = 64\,T\left(\frac{n}{4}\right) + 8\frac{n^2}{4} + n^2$$

$$= 8^3\,T\left(\frac{n}{2^3}\right) + 64\frac{n^2}{8} + 8\frac{n^2}{4} + n^2$$

Cosmos

$= 2^k T\left(\frac{n}{2^k}\right) + n^2[1 + 2^1 + 2^2 + 2^3 + \dots + 2^{k-1}]$

$= n_2 + n^3 - n^2$ $\qquad\qquad$ $\frac{n^2[2^k-1]}{1}$

$\leq n^3 + n^3 - n^3$

$\boxed{O(n^3)}$ $\qquad\qquad$ $2^k = n$

Note:

① Using divide & conquer & without using it, it will take same time $O(n^3)$ or $\theta(n^3)$.

② Without using divide & conquer is better because of stack space.

★ Strassen Method:-

$$T(n) = \begin{cases} O(1), \text{ if } n \leq 2\times2 \\ 7T\left(\frac{n}{2}\right) + 16 \times \left(\frac{n}{2}\right)^2 \end{cases}$$

$aT\left(\frac{n}{b}\right) + f(n)$

$n^{\log_b a} = n^{\log_2 7}$

$f(n) = n^2$ (as constants dont affect complexity)

$\boxed{n^{\log_2 7} > n^2}$ $\therefore$ complexity:- $\boxed{O(n$

7 multiplications of size $\frac{n}{2} \times \frac{n}{2}$

16 additions of size $\frac{n}{2} \times \frac{n}{2}$ [each will take $O\left(\left(\frac{n}{2}\right)^2\right)$ time $[\underset{\text{for } n\times n}{O(n^2)}]$]

$\boxed{O(n^{2.81})} \rightarrow$ Strassen method.

But the best case is $\rightarrow \boxed{O(n^{2.32})}$

Conclusion

① Max-Min. $\rightarrow 2T\left(\frac{n}{2}\right) + 2 \rightarrow O(n)$

② Power of an element $\rightarrow T\left(\frac{n}{2}\right) + 1 \rightarrow O(\log n)$

③ Binary Search $\rightarrow T(n/2) + c \rightarrow O(\log n)$, $\Omega(1)$

④ Merge Sort $\rightarrow 2T\left(\frac{n}{2}\right) + n \rightarrow \theta(n \log n)$

⑤ Quick Sort $\rightarrow$ $2T\left(\frac{n}{2}\right) + n \rightarrow \Omega(n \log n)$; $T(n-1) + n \rightarrow O(n^2)$

⑥ Selection procedure

- $T(\frac{n}{2}) + n \longrightarrow \Omega(n)$ Avg.
- $T(n-1)+n \longrightarrow O(n^2)$

⑦ Matrix Multiplication $\longrightarrow T(n) = 8T(\frac{n}{2}) + n^2 \longrightarrow O(n^3)$

$\hookrightarrow T(n) = 7T(\frac{n}{2}) + n^2 \longrightarrow O(n^{2.81})$

Q.

```
A(n)
{ if(n≤1) return;
    else
      return(A(√n));
}
```

(A) $O(n)$
(B) $O(\log n)$
✓(C) $O(\log\log n)$
(D) $O(n^2)$

rec. relation $\rightarrow T(n) = \begin{cases} O(1), & n\le 1 \\ T(\sqrt{n}) + O(1), & \text{otherwi} \end{cases}$

to calculat $\sqrt{n}$

$$T(n) = T(\sqrt{n}) + O(1)$$
$$= T(n^{1/4}) + O(1)$$
$$= T(n^{1/8}) + O(1)$$
$$\vdots$$
$$= T(n^{1/2^k}) + O(1)$$

$= T(1) + kc$
$= O(1) +$
$c\log\log n$
$= O(\log\log n)$

~~$n^{1/2^k} = y$~~
~~$\frac{\log_2 n}{2^k}$~~

$n^{1/2^k} = c$

$\frac{1}{2^k}\log_2 n = \log$

$\frac{1}{2^k}\log_2 n = C_1$

$\frac{1}{C_1}\log_2 n = 2^k$

$\boxed{\log_2\log_2 n = k}$

Q.

```
DAA(n)
{ if(n≤1) return
    else
    {
  return(DAA(n/2) + DAA(n/2)+100);
    }
}
```

$T(n) = \begin{cases} O(1), & n\le 1 \\ 2T(\frac{n}{2}) + c, & \text{otherwise} \end{cases}$

$= 2\left[2T(\frac{n}{4}) + c\right] + c$

$= 2^2T(\frac{n}{4}) + 2c + c$

$= 2^2\left[2T(\frac{n}{8}) + c\right] + 2c + c$

$= 2^3T(\frac{n}{2^3}) + 4c + 2c + c$

$= 2^kT(\frac{n}{2^k}) + 2^{k-1}c + 2^{k-2}c + \dots + c$

$T(n) = nT(1) + C_1 \quad [n = 2^k]$

$= n + C_1$

$= \boxed{O(n)}$

(?!)

2. A (n)

```
A (n)
{ if (n ≤ 1)
    return;
  else
    return(A(n/2) + A(n/2) + n);
}
```

Similar to:-

```
b = A(n/2);
c = A(n/2);
d = b+c+n;
return(d);
```

This is just addition & it will take O(1) time for addition. [not a function call.]

$$T(n) = \begin{cases} O(1) & , n \le 1 \\ T(\frac{n}{2}) + T(\frac{n}{2}) + c & \text{otherwise} \end{cases}$$

$= 2T(\frac{n}{2}) + \cancel{n}\, c$

$= 2\left[2T(\frac{n}{4}) + \frac{n}{2}\right] + \cancel{n}\, c$

$= 2^2 T(\frac{n}{2^2}) + n + n$

$= 2^2\left[2T(\frac{n}{2^3}) + \frac{n}{4}\right] + n + n$

$= 2^3 T(\frac{n}{2^3}) + n + n + n$

$= 2^k T(\frac{n}{2^k}) + kn$

but, $2^k = n$

$\therefore nT(1) + n\log_2 n\, c$

$\cancel{O(n\log_2 n)} \quad = \boxed{O(n)}$

My mistake. [taken n as function call.]

If return(A(n/2) * A(n/2));

$T(n) = 2T(\frac{n}{2}) + c$ → for multiplication.

☆ Master theorem is applicable when f(n) is a [illegible] polyno
function.

Master Theorem:-

It is applicable only in the form of divide & conquer, e.g.

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ $\quad a \geq 1$
$\quad b > 1$

Case 1:- $f(n) = O(n^{\log_b a - \epsilon})$ where $\epsilon > 0$
then
$\boxed{T(n) = \theta(n^{\log_b a})}$

Case 2:- $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$
then $\boxed{T(n) = \theta(f(n))}$

Case 3:- $f(n) = \theta(n^{\log_b a})$
then $\boxed{T(n) = \theta(n^{\log_b a} * \log n)}$

e.g. ① $T(n) = 8T\left(\frac{n}{2}\right) + n^2 \rightarrow a = 8, b = 2$ $\quad n^{\log_b a} = n^3$
$n^2 = O(n^3)$
$f(n) < n^{\log_b a}$ $\quad \boxed{\theta(n^3)} \longrightarrow$ Case 1

② $T(n) = 2T\left(\frac{n}{2}\right) + n \longrightarrow a = 2, b = 2$ $\quad n^{\log_b a} = n$
$f(n) = n^{\log_b a}$
$\therefore \boxed{\theta(n \log n)} \longrightarrow$ Case 3

③ $T(n) = 2T\left(\frac{n}{2}\right) + 1$
$n^{\log_b a} = n$ $\quad , f(n) = 1$ $\quad f(n) < n^{\log_b a} \longrightarrow \boxed{\theta(n)} \rightarrow$ Cas

④ $T(n) = T\left(\frac{n}{2}\right) + c$
$f(n) = c$
$n^{\log_2 1} = n^0 = 1$ $\quad n^{\log_b a} = f(n)$ $\quad \therefore \theta(n^0 \log n) = \theta(\log n$

⑤ $T(n) = 4T\left(\frac{n}{2}\right) + n^3$ $\quad$ Case 2:- $\boxed{\theta(n^3)}$
$n^{\log_2 4} = n^2$

⑥ $T(n) = 8T\left(\frac{n}{2}\right) + n^2$ $\quad \theta(n^3)$
$n^{\log_2 8} = n^3$
$n^{\log_b a} = n^3$ $\quad n^2 = O(n^3)$
$f(n) = n^2$

Imp.
☆ ϵ tells that
n^3 is how many
times greater than
n^2, $\boxed{\epsilon = 1}$
$n^2 = O(n^{3-1})$

★ Master theorem is applicable when $f(n)$ is a polynomi
function.

Master Theorem:-

It is applicable only in the form of divide & conquer, e.g.

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ $\quad a \geq 1$

$\quad b > 1$

Case 1:- $f(n) = O(n^{\log_b a - \epsilon})$ where $\epsilon > 0$

then

$\boxed{T(n) = \Theta(n^{\log_b a})}$

Case 2:- $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$

then $\boxed{T(n) = \Theta(f(n))}$

Case 3:- $f(n) = \Theta(n^{\log_b a})$

then $\boxed{T(n) = \Theta(n^{\log_b a} * \log n)}$

e.g. ① $T(n) = 8T\left(\frac{n}{2}\right) + n^2$ $\rightarrow a = 8, b = 2$ $\quad n^{\log_b a} = n^3$

$n^2 = O(n^3)$

$f(n) < n^{\log_b a}$ $\quad \boxed{\Theta(n^3)} \rightarrow$ Case 1

② $T(n) = 2T\left(\frac{n}{2}\right) + n$ $\rightarrow a = 2, b = 2$ $\quad n^{\log_b a} = n$

$f(n) = n^{\log_b a}$

$\boxed{\Theta(n \log n)} \rightarrow$ Case 3

③ $T(n) = 2T\left(\frac{n}{2}\right) + 1$

$n^{\log_b a} = n$ $\quad , f(n) = 1$ $\quad f(n) < n^{\log_b a} \rightarrow \boxed{\Theta(n)} \rightarrow$ Case

④ $T(n) = T\left(\frac{n}{2}\right) + c$

$f(n) = c$

$n^{\log_2 1} = n^0 = 1$ $\quad n^{\log_b a} = f(n)$ $\quad \therefore \Theta(n^0 \log n) = \Theta(\log n$

⑤ $T(n) = 4T\left(\frac{n}{2}\right) + n^3$ $\quad$ Case 2:- $\boxed{\Theta(n^3)}$

$n^{\log_2 4} = n^2$

Imp.

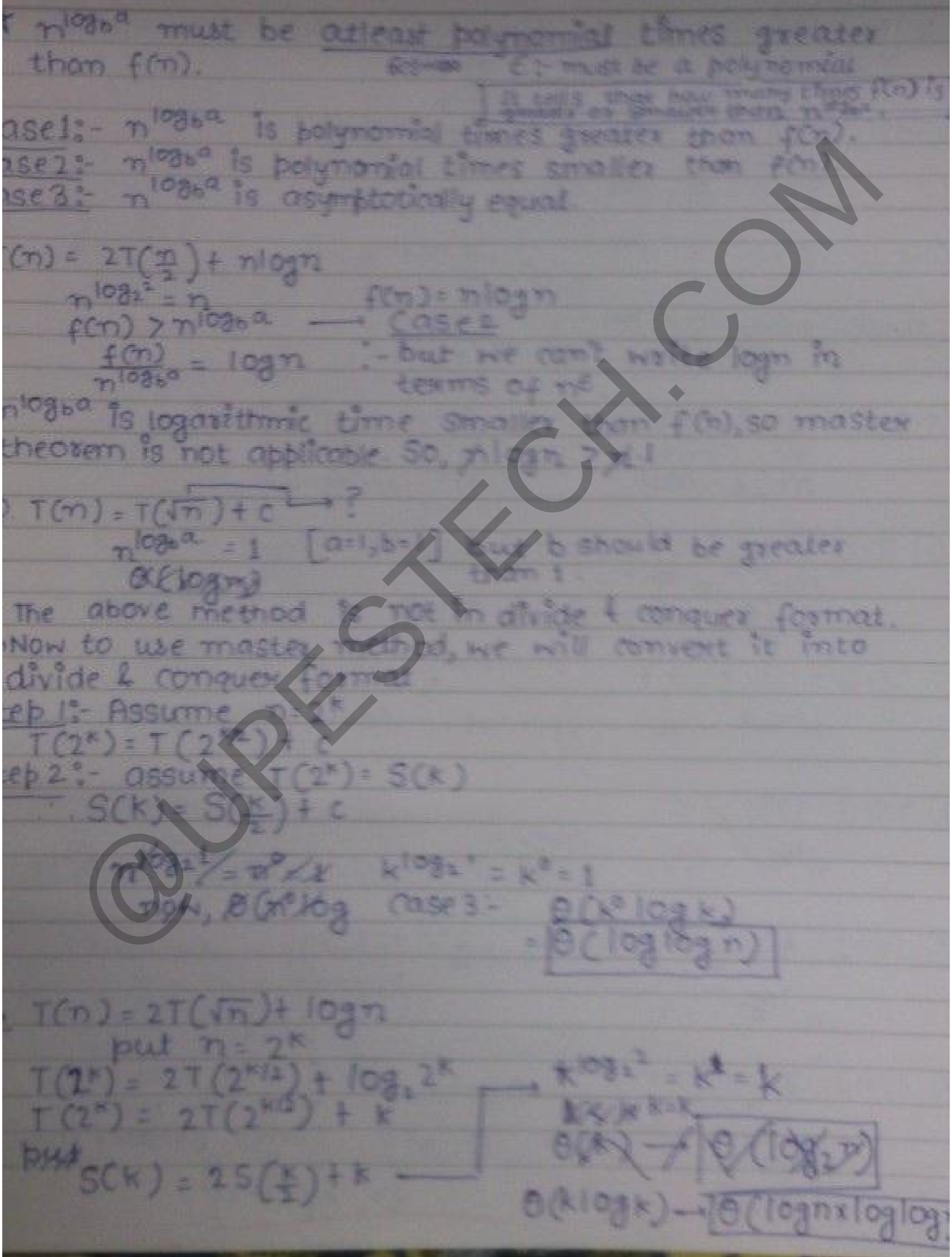
⑥ $T(n) = 8T\left(\frac{n}{2}\right) + n^2$ $\quad \Theta(n^3)$ $\quad$ ★ ϵ tells that n^3 is how many times greater than n^2, $\boxed{\epsilon = 1}$

$n^{\log_2 8} = n^3$

$n^{\log_b a} = n^3$ $\quad n^2 = O(n^3)$

$f(n) = n^2$ $\quad n^2 = O(n^{3-1})$

Cosmos



$n^{\log_b a}$ must be atleast polynomial times greater than f(n). ~~[illegible]~~ E:- must be a polynomial

[it tells that how many times f(n) is greater or smaller than $n^{\log_b a}$.

ase1:- $n^{\log_b a}$ is polynomial times greater than f(n).

ase2:- $n^{\log_b a}$ is polynomial times smaller than f(n).

se3:- $n^{\log_b a}$ is asymptotically equal.

$T(n) = 2T(\frac{n}{2}) + n\log n$

$n^{\log_2 2} = n$ $f(n) = n\log n$

$f(n) > n^{\log_b a} \longrightarrow$ Case2

$\frac{f(n)}{n^{\log_b a}} = \log n$:- but we can't write logn in terms of n^{ϵ}

$n^{\log_b a}$ is logarithmic time smaller than f(n), so master theorem is not applicable. So, $n\log n > n$!

$T(n) = T(\sqrt{n}) + c \longrightarrow$?

$n^{\log_b a} = 1$ [a=1, b=1] but b should be greater than 1.

$\Theta(\log n)$

The above method is not in divide & conquer format. Now to use master method, we will convert it into divide & conquer format.

ep 1:- Assume $n = 2^k$

$T(2^k) = T(2^{k/2}) + c$

ep 2:- assume $T(2^k) = S(k)$

$\therefore S(k) = S(\frac{k}{2}) + c$

$n^{\log_2 1} = n^0 < k$ $k^{\log_2 1} = k^0 = 1$

$\Theta(n^0 \log$ case 3:- $\Theta(k^0 \log k)$

$= \boxed{\Theta(\log\log n)}$

$T(n) = 2T(\sqrt{n}) + \log n$

put $n = 2^k$

$T(2^k) = 2T(2^{k/2}) + \log_2 2^k$

$T(2^k) = 2T(2^{k/2}) + k$

put $S(k) = 2S(\frac{k}{2}) + k$ ——

$k^{\log_2 2} = k^1 = k$

$k < k^{k=k}$

~~$\Theta(k) \rightarrow \boxed{\Theta(\log_2 n)}$~~

$\Theta(k\log k) \rightarrow \boxed{\Theta(\log n \times \log\log$

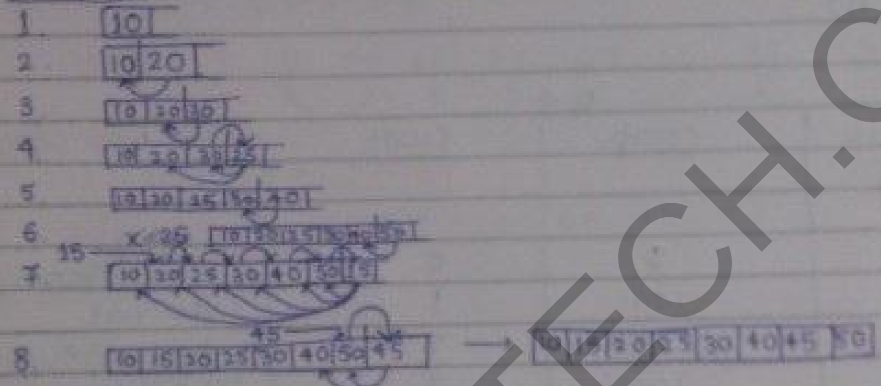
Insertion Sort:-

Note:- Insert & Sort.

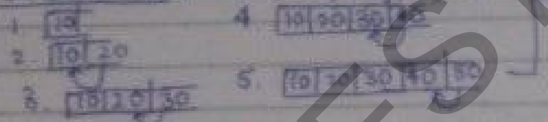
example:- A[10 20 30 25 40 50 15 45]
1 2 3 4 5 6 7 8

★It is inplace sorting technique.

1st pass:-



Best Case:- 10,20,30,40,50



→ n-1 comparisons

∴ $\boxed{\Omega(n)} \longrightarrow$ Best case
(when it is already sorted.)

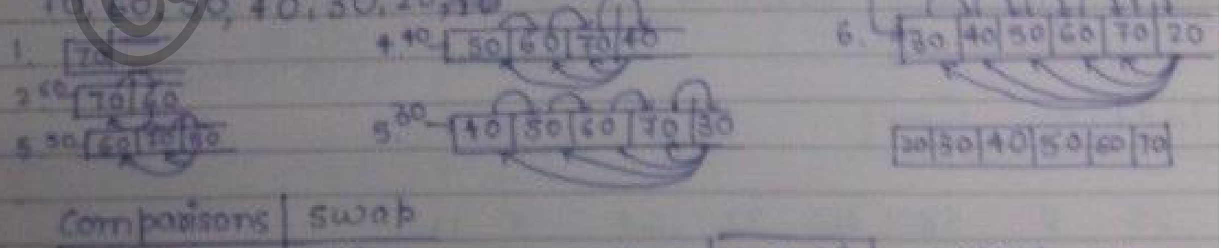
Note①:- To sort n elements using insertion sort, it will take (n-1) comparisons in the best case.

② If array is almost sorted, then insertion sort is the best

③ If array size is small, then insertion sort is best sort (merge sort is worst in this case.)

Worst Case:-

70, 60, 50, 40, 30, 20, 10



Comparisons	swap
0	0
1	1
2	2
3	3
4	4
5	5
n-1	n-1
n^2	n^2

$2n^2 \rightarrow \boxed{O(n^2)} \rightarrow$ worst case

Average case:-

When half of the elements are to be sorted, like
10, 20, 30, 70, 60, 50

$\frac{n}{2} * \frac{n}{2} \rightarrow \frac{n^2}{4} \rightarrow \boxed{O(n^2)}$

Cosmos

While applying insertion sort, to insert a particular element to its correct place, for that we are using linear search, then what is the worst case time complexity of insertion sort if we replace by binary search?

) $O(n^2)$ (C) $O(\log n)$
B) $O(n\log n)$ (D) $O(n)$

My answer :- (B)

Sir's Answer :-

Worst case

LS comparisons	swap	BS comp.	swap
•	•		
0	0	0	0
1	1	log1	1
2	2	log2	2
3	3	log3	3
n-1	n-1	log(n-1)	n-1
n^2	n^2	nlogn	n^2

} → swap operations do not decrease in binary search

LS: $2n^2$ → $O(n^2)$

BS: $\underset{\text{n}}{n\log n} + \underset{\pi}{n^2} \longrightarrow \boxed{O(n^2)}$ → ans.

:How to decrease no. of swaps.

If i/p is present in Linked List.

∵ In linked list we can only perform Linear Search which will take ~~$O(n^2)$~~ n^2 comparisons & 0 swaps.

∴ complexity :- $\boxed{O(n^2)}$

↓ 0+1+2+3+ +n ≤ n(n+1) +n ≤ n^2

Selection Sort

:-	25	5	58	45	32	64	3	11
	1	2	3	4	5	6	7	8

s1 :- i = 1

min: ~~1~~ ~~2~~ 7 (minimum's position) [compare (i) Select position 1 to be the 1st minimum.

(ii) compare the minimum with positions 2, 3, ...

if any element < minimum,

then minimum = that element.

& minimum's position = that element's position.

Straight Selection Sort :-

(iii) swap(a[i], a[min])

a[i] → 3 5 58 45 32 64 25 11
1 2 3 4 5 6 7 8

pass 2 :- now, i = 2
min = 2

pass 3 :- i = 3
min = ~~3~~ ~~4~~ ~~5~~ ~~7~~ 8

a[i] → 3 5 11 45 32 64 25 58
1 2 3 4 5 6 7 8

pass 4 :- i = 4
min = ~~4~~ ~~8~~ 7

a[i] → 3 5 11 25 32 64 45 58

pass 5 :- i = 5
min = 5

pass 6 :- i = 6
min = ~~6~~ 7

a[i] → 3 5 11 25 45 64 58

pass 7 :- i = 7
min = ~~7~~ 8

a[i] → 3 5 11 25 45 58 64

Complexity :-

1st pass :- n-1 comparisons
2nd pass :- n-2 comparisons
3rd " :- n-3 "
4th " :- n-4 "

$\therefore (n-1)+(n-2)+(n-3)+ \dots +1 \leq n+n+n+ \dots$ n times =

Best Case :- $\Omega(n^2)$ $\boxed{\Theta(n^2)}$
Worst case :- $O(n^2)$

Bubble Sort :- ↵ $\boxed{O(n^2)}$

i/p :- 75 58 10 84 5
1 2 3 4 5

pass 1 :- 75 58 10 84 5
58 75 10 84 5
58 10 75 84 5
58 10 75 84 5
58 10 75 5 84

pass 2 :- 58 10 75 5 84
10 58 75 5 84
10 58 75 5 84
10 58 5 75 84

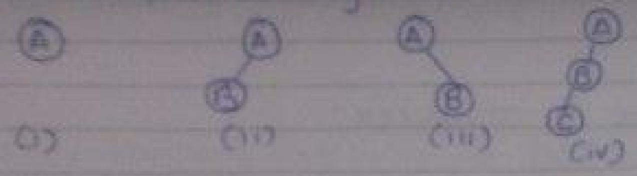
pass 3 :- 10 58 5 75 84
10 5 58 75 84

pass 4 :- 10 5 58 75 84
5 10 58 75 84

Cosmos

Heapsort

Almost Complete Binary Tree :



(i),(ii),(iii),(iv)
↓
Binary Tree

(i),(ii),(iv) :- almost complete binary tree.

(iii),(v) :- not an almost complete binary tree.

In almost complete Binary Tree :-

(i) Without completing left, we can't go to the right.

(ii) Without completing a level, we can't go to the next level.

A binary tree is said to almost complete binary tree iff :-

① At every node, after completing left child, only then go to right child

② At every node, after completing the present level, only then go to next level

* In the given almost complete binary tree, if last level is also filled completely, then it is called complete binary tree.

* Another name for complete binary tree :- maximal binary tree.

$1+2+4+\ldots+2^n$

$\frac{1(2^{n+1}-1)}{2-1}$

$= 2^{n+1}-1$

No. of levels = $n+1$

Note:- A complete binary tree with K-levels, the no of nodes = $\boxed{2^k-1}$

A binary tree with k-levels, the max no. of nodes = $\boxed{2^k-1}$.

Note:- The no of nodes in a complete binary tree at level i = $\boxed{2^{i-1}}$

Q. If there are n nodes in a binary tree:-

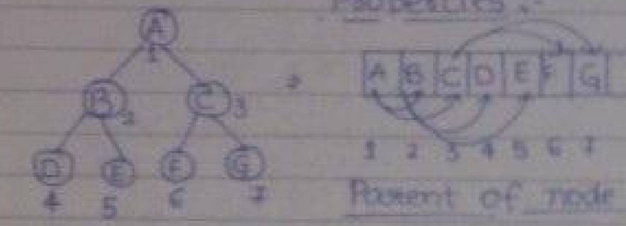
max no of levels = $\boxed{n}$

min no. of levels (for complete binary tree) = $\boxed{\log_2(n+1}$

★ If there are binary tree & complete binary tree, then the complete binary tree will take less time for any arbitrary operation.

How to represent binary tree in computer :-

Properties :-



A	B	C	D	E	F	G
1	2	3	4	5	6	7

Parent of node at ith place :- If i is even = $\frac{i}{2}$ th po

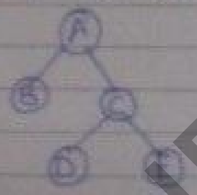
if i is odd = $\lfloor \frac{i}{2} \rfloor$ th p

(floor v

Child :- Left Child of node at ith position : $\boxed{2i^{th} \text{ position}}$

Right Child of node at ith position : $\boxed{(2i+1)^{th} \text{ position}}$

Q. Store the following binary tree using array representation.



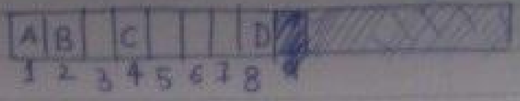
A	B	C			D	E
1	2	3	4	5	6	7

Q. Store the following Binary Tree in form of array :-

A		B				C								D
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

★ Array Representation of Binary Tree is favourable when tree is almost complete binary tree. Linked List Representation is favoured when tree is not almost complete (like the above question).

Q.



★ After reaching last node, we ~~stabb~~ stop.

Heapsort :-

- Heapsort require Heap trees

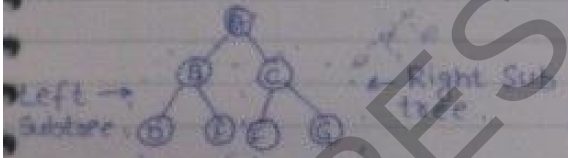
★ Heap Trees are almost complete binary trees.

- Heatrees / Min heap Tree
  \ Max Heap Tree

Min Heap Tree

~~• An almost comp~~

• In a given almost complete binary tree, at every node root is minimum (or equal) in comparision to its children, then the tree is called min heap tree



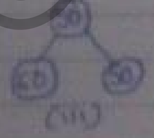
★ we dont have to compare Sub trees, but only children.

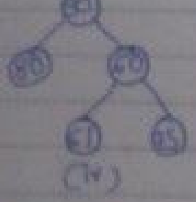
e.g.

(i),(ii), (iii)- min heap trees

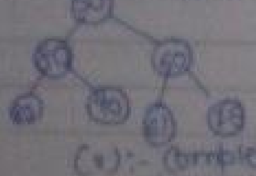

(i)


(ii)


(iii)


(iv)

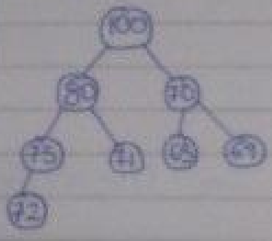
(iv) - not a min heap tree (because it is not almost complete binary tree)



(v):- Complete binary tree & min heap tree.

Max Heap Tree

• In a given almost complete binary tree, at every node is maximum (or equal) in comparison its children, then it is max heap tree.



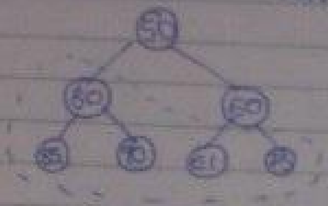
★ If there are n no. of leaf nodes, then no. of internal nodes are n-1
So total no. of nodes = $\boxed{2n-1}$

[Only for Complete binary trees.]

★ The root of min heap tree is the minimum element.

★ The 2nd min. is from 2nd level

★ The 3rd min. can be :-



The 3rd min. can be in dotted places.

Strict Binary Tree :-

- Every node must have either 0 children or 2 children

★ The max. element in the min heap tree is at the last level

Insertion in Min heap tree :-

(i) Store tree in form of array :-

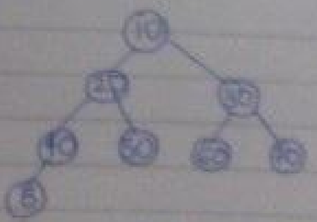
⇒

10	20	30	40	50	60	70
1	2	3	4	5	6	7

(ii) Now, insert '5'

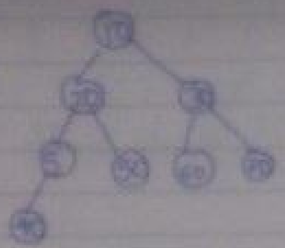
10	20	30	40	50	60	70	5
1	2	3	4	5	6	7	8

5 is the child of '40'.

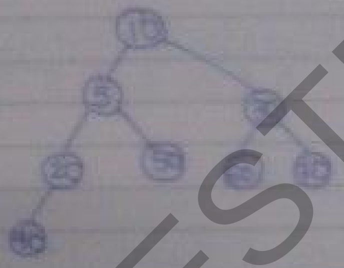


We can only insert at inserted position because we have to make it almost complete binary tree.

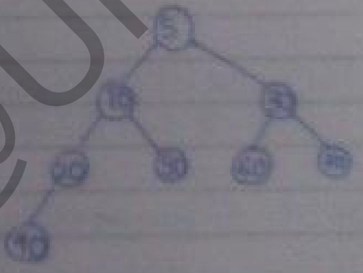
(iii) Now 5 will be compared with 40, where 5 is min, so it will be swapped



(iv) Now 5 will be compared with its parent (20), & hence swapped.



(v) Now 5 will be compared with its parent (10), & hence swapped.

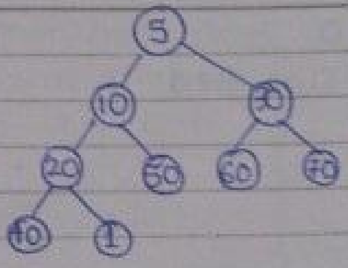


Now, in array:-

10	[illegible]

Now, Insert '1'

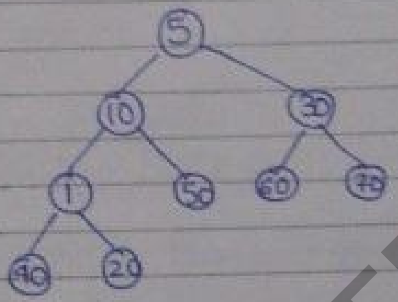
Cosmos



5	10	30	20	50	60	70	40	1
1	2	3	4	5	6	7	8	9

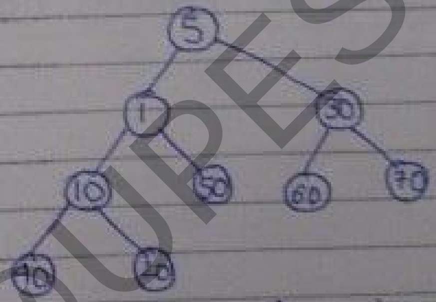
1 is right child of 20 which is at 4, ∴ 1 is at 4×2+1=9.

Now, compare '1' with its parent & swap it:-



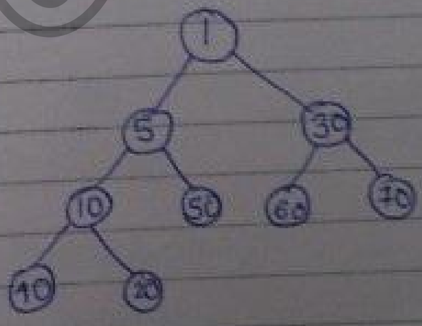
5	10	30	1	50	60	70	40	20
1	2	3	4	5	6	7	8	9

Now, compare '1' with its parent & swap it:-



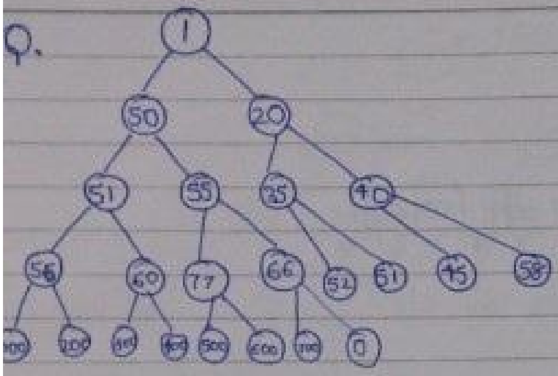
5	1	30	10	50	60	70	40	20
1	2	3	4	5	6	7	8	9

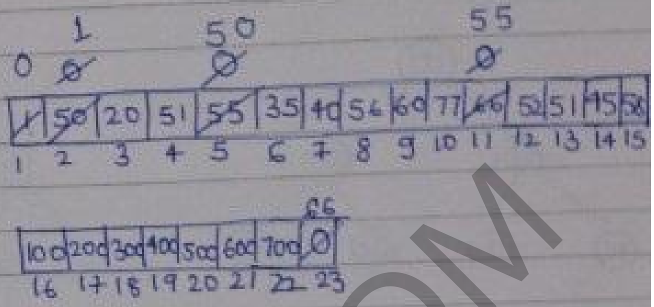
Now, compare '1' with its parent & swap it:-



1	5	30	10	50	60	70	40	20
1	2	3	4	5	6	7	8	9

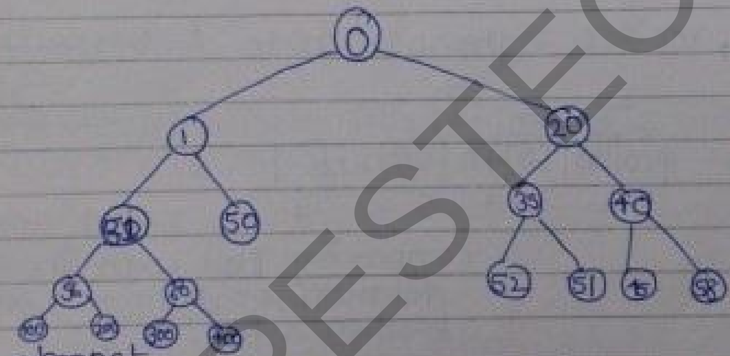
Q.





Now, insert '0'.

. Compare 0 with element at 11th position



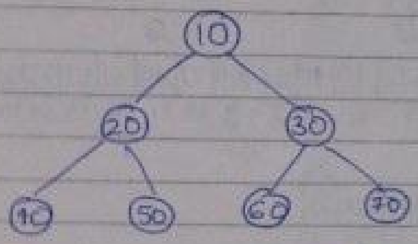
* In almost complete binary tree with min heap tree properties, there are $\lceil \log_2 n \rceil$ levels, so ~~the~~ at max. there will be $\{\log_2 n\}$ comparisons & $\log_2 n$ swaps. ∴ $\log_2 n + \log_2 n = 2\log_2 n$

- Worst Case :- $O(\log_2 n)$
- Best Case :- $\Omega(1)$
- Average Case :- $\Theta(\log_2 n)$

* In order to insert an element into min heap o max heap which already contains n elements requ $\boxed{O(\log_2 n)}$ (Worst Case & Average Case).
& $\Omega(1)$ (Best Case).

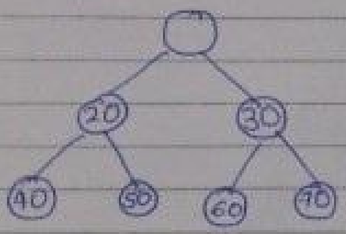
☆ We always delete the root.

Deletion



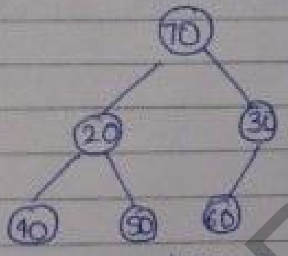
10	20	30	40	50	60	70
1	2	3	4	5	6	7

· Now, delete '10'.



	20	30	40	50	60	70
1	2	3	4	5	6	7

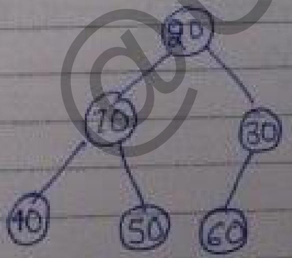
Now, go to last level, replace rightmost node & place it t root.



70	20	30	40	50	60	
1	2	3	4	5	6	7

Comparing root with its children, i.e. position 1 with position 2 & 3.

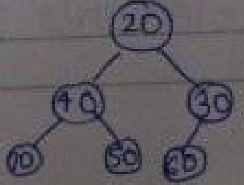
Now, compare 70 with 20 & 30, 20 is min, ∴ Swap ~~replace~~ 70 & 20



20	70	30	40	50	60	
1	2	3	4	5	6	7

Comparing positio
2 with its childre
(4 & 5)

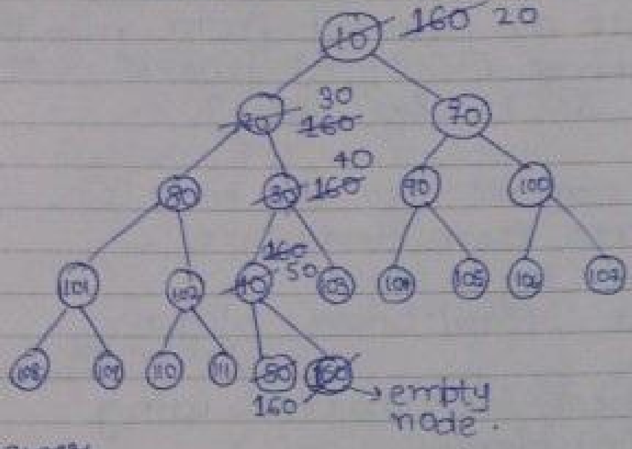
Now compare 70 with 40 & 50, 40 is min, ∴ Swap 70 & 40.



20	40	30	70	50	60	
1	2	3	4	5	6	7

Cosmos

Q.



Answer

20	30	70	80	40	90	100	101	102	50	103	104	105	106	107	108	109	110	111	160
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

20	30			40					50										160	
~~160~~	~~160~~			~~160~~					~~160~~											
~~10~~	~~20~~	30	80	~~30~~	90	100	101	102	~~40~~	103	104	105	106	107	108	109	110	111	~~50~~	~~160~~
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

em
no

~~1. Swap 1 & 2~~

1. Delete 1.
2. Place 21 at 1
3. Compare 1 with 2 & 3, swap 1 & 2.
4. Compare 2 with 4 & 5, swap 2 & 5
5. Compare 5 with 10 & 11, swap 5 & 10.
6. Compare 10 with 20, ~~& 21~~ swap 10 & 20.

In deletion:- (or level)

At every step, there are 2 comparisons & 1 swap,
so no of levels = $\log_2 n$.
∴ Time complexity = $3\log_2 n$ = $\boxed{O(\log_2 n)}$ → Worst cas

Average Case:- $\boxed{\theta(\log_2 n)}$

Best Case :- $\boxed{\Omega(1)}$

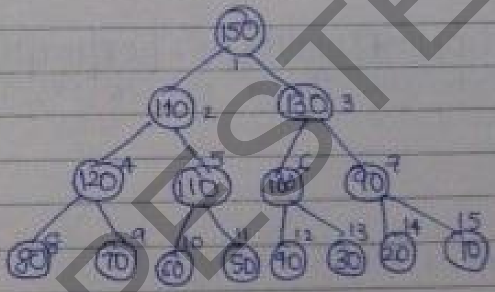
Note:- In order to delete an element from min hea
or max heap requires $O(\log_2 n)$ [worst case
& average case] & $\Omega(1)$ [Best Case.]

★ If we have to insert n elements into the heap (i.e.
constructing min heap tree from empty tree):-
each insertion will take $\log_2 n$ time,
as there are n nodes, ∴ total time = $n\log_2 n$
(Brute Force Method)

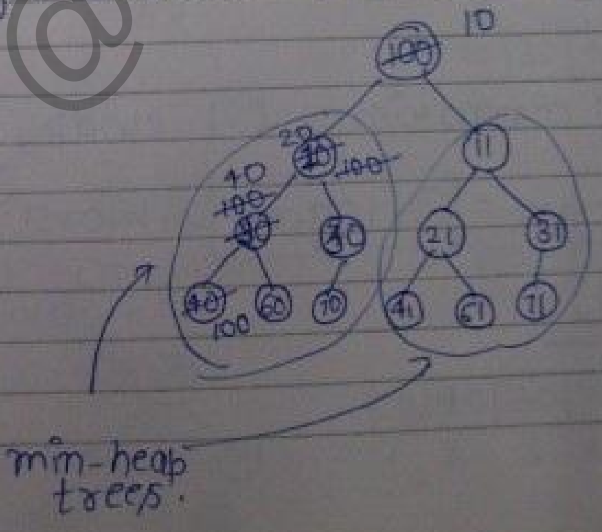
★ But Constructing min heap (or) max heap using
Build heap will take $O(n)$ time.

i/p:- 150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10
create min-heap tree.

Step 1:- Construct almost complete binary tree.

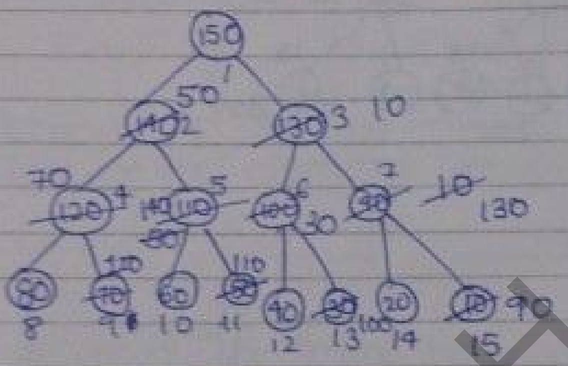


★ We can use min heapify method only when left &
right subtrees of root are themselves minheap tree.



~~No~~

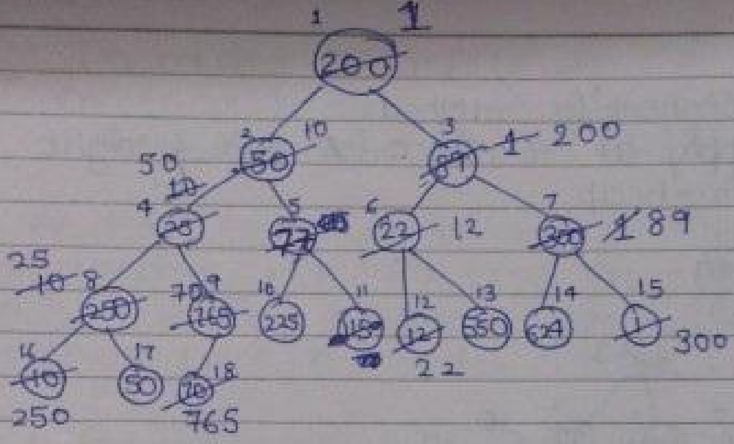
Step 2:- Now, apply Minheapify method
we can only apply at 7, since its left & right subtree are min-heap.

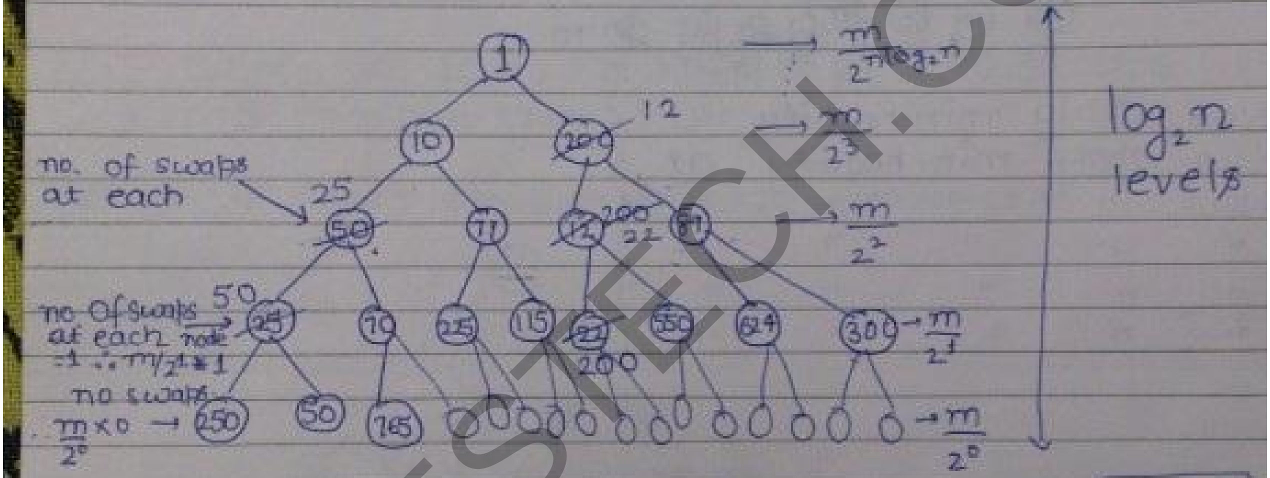


- Now, apply min-heapify at 6.
- Now, apply min-heapify at 5.
- Now, " " " " 4.
- " , " " " " 3.
- " , " " " " 2.
- " , " " " " 1

Q. Construct min heap tree for following methods :-
200, 50, 89, 25, 77, 42, 300, 250, 765, 225, 115, 12, 550, 624, 1, 1
70.

P.T.O.





m:- no. of nodes at last level
n:- total no. of nodes in tree.
→ $\boxed{m = \frac{n}{2}}$

Time complexity of build heap method =

$$\frac{m}{2^0}\times 0 + \frac{m}{2^1}\times 1 + \frac{m}{2^2}\times 2 + \dots + \frac{m}{2^{(\log n)-1}}\times[(\log n)-1]$$

$$= m\left[\underbrace{\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \dots\dots + \frac{[(\log_2 n)-1]}{2^{(\log n)-1}}}_{O(1)}\right]$$

$= m * O(1)$

$= \frac{n}{2} * O(1)$

$= \boxed{O(n)}$

Cosmos

- ~~Finding~~ Deleting 1st min element → $\log_2 n$
- ~~Finding~~ Deleting 2nd min. element → $2\log_2 n$
- ~~Finding~~ Deleting 3rd min. element → $3\log_2 n$ — V.Imp
- Finding nth min. element → O(n)

because nth min. element = max. element

& max. element is in last level,
& so to find max. element at last level, it will take O(n) time

---

Heapsort

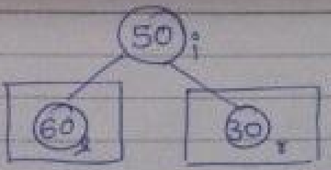
1) O(n) time to make the min. heap
2) Deleting the ith min. element from min. heap :- $O(\log_2 n)$
this ith element is

Heapsort:-

1) Create min heap Using Build heap :- O(n)
2) Delete the elements one by one & store from right hand side of original array.
each deletion takes $O(\log_2 n)$ & storing at right hand side takes O(1) time,
So to delete 'n' elements & storing them at right hand side :- $O(n\log_2 n)$

Complexity :- $O(n) + O(n\log_2 n)$
$= \boxed{O(n\log_2 n)}$

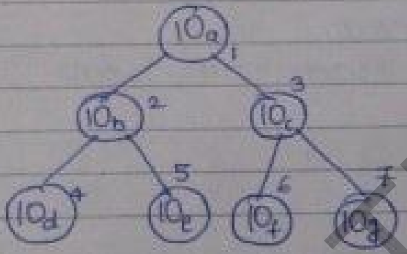
- Min heap will give descending order. } In-place sorting
- Max heap " " ascending " .



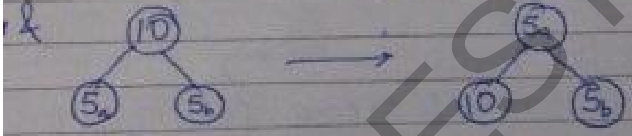
```
Minheapify(i)
{  min=i;
   l=2*i;
   r=(2*i)+1;
   if (a[min]>a(l))
       min=l;
   if(a[min]>a(r))
       min=r;
   swap(a[i],a[min]);
}
```

10_a 10_b 10_c 10_d 10_e 10_f 10_g

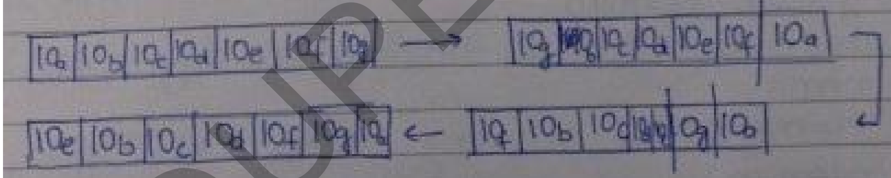
nstruct min-heap:-



using min-heapify:-
there will be no swaps.



[from min-heapify]



Heapsort is not stable sorting technique as the rdering of similar elements changes.
Heapsort & Quicksort are not stable, rest others re stable.
ll sorting techniques are in-place except Merge-sort.

Conclusion :-

① Min heapify :- $O(\log_2 n)$, $\Omega(1)$
② Build-heap :- $\Theta(n)$
③ Finding min :- $O(1)$
④ Deleting min :- $O(\log_2 n)$
⑤ Deleting 5th min. :- $5\log_2 n$:- $O(\log_2 n)$
⑥ Finding 5th min. :- Deleting upto 4th min. :- $4\log_2 n$:- $O(\log$
⑦ Deleting nth min. :- Deleting max. element :- $O(n)$

max. element is in last level, & hence taking only last level & finding max. element from it & deleting it will take $\frac{n}{2}$ which is $O(n)$.

Greedy Technique (V.V.Imp.)

i/p :- Most of the problems in Greedy contain n-i/p's & our goal is finding subset which will optimize our objective.

Basics :-
① Solution Space :- all possible solutions [For given n-i/p all possible soln (may or may not be correct) is called soln space.
② Feasible Solution
③ Optimal Solution

- Feasible :- It is one of the soln from soln. space which w satisfy our condition.
- Optimal :- It is one of the feasible soln. which optimizes our goal.

★ e.g. if we have a class of 200 students & we want to find out top 10 students.

- Solution Space :- $^{200}C_{10}$:- all combinations of 10 students.
- Feasible Solution :- that combinations of 10 studen which satisfy the objective, i.e. whose avg. is greater than a specified value.
- Optimal soln. :- that combination of 10 students from feasible soln. whose marks are greater than rest other combinations.

★ To find top 10 students from 200 students then create ~~min~~ heap which will take O(n) time & delete first 10 max elements which will take $10\log_2 n$ time ∴ $O(n)+O(\log_2 n) = \boxed{O(n)}$.

Control abstraction of Greedy

```
                    ┌──→ array of n-elements
Greedy Technique(a,n)
{ solution = Φ              // no soln. in the beginning
  for(i=1; i≤n; i++)
   { x = select(n);          // finding the feasible solutions
     if (Feasible(x))        // checking for feasible soln.
        add(x, solution);
   }
}
```

Time complexity depends upon :-

- Select(n)
- Feasible(x)
- add(x, solution)

Best Case :- $\Omega(n)$ [when all three fn. take O(1) time

- example :- Select(n) → n
  feasible(x) → n^2
  add(x, solution) → n^4
  ∴ complexity :- $\boxed{O(n^5)}$ (because of for loop)

★★ Q. The time complexity of Greedy Technique :-
(a) O(n) (b) $O(n^2)$ (c) $O(n^3)$ (d) $O(n^4)$

→ though we can't say anything about the complexity of the functions, so it may even be $O(n^{100})$ but from the available option (d) is the most appropriate option.

Applications Of Greedy :-

① Job Sequencing with Deadline
② Knapsack problem
③ Huffman Coding
④ Optimal Merge pattern
⑤ Minimum Cost Spanning Tree
  (i) Prim's
  (ii) Krushkal's

⑥ Single Source Shortest Path
(i) Dijkstra's
(ii) Bellman-Ford

Job Sequencing with Deadline

① Single-CPU :- at a time only single job is executing
② Interleaving is not allowed (Round-Robin is not possible).
③ Arrival times of all jobs are same. (FCFS is not possible)
④ 1-unit of running time for every job. (SJF is not possible).

	J_1	J_2	J_3	J_4
Deadline	2	1	2	1
Profit	50	75	45	80

$(J_4 \rightarrow J_1) \rightarrow$ Optimal Soln.

$\{J_2, J_1\} \rightarrow 125$
$\{J_2, J_3\} \rightarrow 120$
$\{J_4, J_1\} \rightarrow \boxed{130}$
$\{J_4, J_3\} \rightarrow 125$
$\{J_1, J_3\} \rightarrow 95$
$\{J_3, J_1\} \rightarrow 95$
$\{J_1\} \rightarrow 50$
$\{J_2\} \rightarrow 75$
$\{J_3\} \rightarrow 45$
$\{J_4\} \rightarrow 80$
$\{\ \} \rightarrow 0$

→ Feasible Solution

Soln Space:-

$J_1 J_2$ $J_2 J_1$ $J_3 J_1$ $J_4 J_1$ J_1 $\{\ \}$
$J_1 J_3$ $J_2 J_3$ $J_3 J_2$ $J_4 J_2$ J_2 no job at all.
$J_1 J_4$ $J_2 J_4$ $J_3 J_4$ $J_4 J_3$ J_3
J_4

$J_1 J_2 J_3$ 3 jobs $\rightarrow {}^4P_3 = \frac{4!}{1!} = 4!$

$J_1 J_2 J_3 J_4$ 4 jobs $\rightarrow {}^4P_4 = \frac{4!}{0!} = 4!$

Soln Space :- $17 + 2 \times 4!$
$= 17 + 48$
$= \boxed{65}$

Optimal Soln. :-

J_2, J_1
J_2, J_3
J_4, J_1
J_4, J_3
J_1, J_3
J_3, J_1

Objectives:-
complete as many jobs as possible in the deadline.

To calculate Optimal Soln. directly :-

1	2
J_4	J_1

Step(i):-
jobs that can be done in 2nd month.
(J_1 & J_2), we take J_1 for max. profit

Step(ii):-
Jobs that can be done in 1st month
(all jobs can be done in 1st month, but J_1 is done in slot 2),
So, only J_2, J_3, J_4 are in competition,
J_4 is giving max profit

Jobs	✓ J_1	✓ J_2	✓ J_3	J_4	✓ J_5	✓ J_6
Profits	24	18	22	10	12	30
Deadlines	5	3	3	2	4	2

n = 6

J_2	J_6	J_3	J_5	J_1
1	2	3	4	5

↑ start from this side.

n=5

obs	✓ J_1	✓ J_2	✓ J_3	J_4	✓ J_5
rofits:	10	20	~~30~~ 15	5	80
eadlines:	3	3	3	4	4

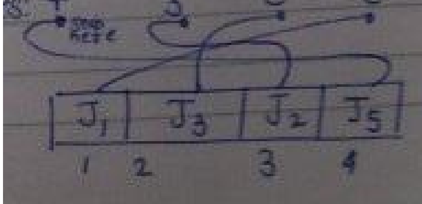
J_1	J_2	J_3	J_5
1	2	3	4

↑ start from this side

Profit = 125

(i) :- Sort all the jobs in decreasing order of profit.

	J_5	J_2	J_3	J_1	J_4
	80	20	15	10	5
s:	4	3	3	3	4



Q. $n=9$

Jobs	✓ J_1	✓ J_2	✓ J_3	J_4	✓ J_5	J_6	✓ J_7	✓ J_8	✓ J_9
Profits:	15	20	30	18	18	10	~~20~~ 23	16	25
Deadlines	7	2	5	3	4	5	2	7	3

Sort acc. to profits

Jobs	J_3	J_9	J_7	J_2	J_4	J_5	J_8	J_1	J_6
Profits	30	25	23	20	18	18	16	15	10
Deadlines	5	3	2	2	3	4	7	7	5

J_2	J_7	J_9	J_5	J_3	J_1	J_8
1	2	3	4	5	6	7

Profit : $\boxed{147}$

J_4 & J_6 are left o

Algorithm

① Sort all jobs in decreasing order of profits → O(nlog

② Find maximum deadline in the array of n-deadlin & take array of that size, & start from RHS. → $\boxed{O(n}$

③ For every slot no. 'i', apply linear search to find job which contain deadline ≥ 'i' → $O(n^2)$

Knapsack Problem :-

problem:- $\sum_{i=1}^{n} w_i > M$

M :- weight that knapsack can hold

w_i :- weight of i^{th} object.

Feasible Soln. :- $\sum_{i=1}^{n} x_i w_i \le M$

Optimal Soln. :- $\sum_{i=1}^{n} x_i P_i$ is maximum.

★ (Fractions are allow in Greedy)

★ When fractions are n allowed, it is shifted i Dynamic prog ing.

now,

n = 3   M = 20

objects	Ob_1	Ob_2	Ob_3
profits	25	24	15
weights	18	15	10

My answer

Optimal Soln. :- $\frac{18}{18} \times 25 + \frac{2}{15} \times$

$= 25 + 3.2$

$= 28.2$

★ All 3 are feasible solutions.

Greedy About Profit:-

$1 \times 25 + \frac{2}{15} \times 24 + 0 \times 10$

$= 25 + 3.2 = \boxed{28.2}$

Greedy About weight:-

Take less weight object 1st:-

$(\underline{0} , \underline{10/15} , \underline{1})$

- Take 3rd object 1st
- Now, out of 1st & 2nd object, 2nd object has less weight.

$\therefore$ Profit :- $0 \times 25 + \frac{\cancel{10}^{2}}{\cancel{15}_{3}} \times \cancel{24}^{8} + 1 \times 15$

$= 16 + 15 = \boxed{31}$

Greedy About Both Profit & weight:- (Sir's answer)

bj 1 :- 18 weight → 25 profit

1 " → $\frac{25}{18} = 1.3$

j 2 :- 15 weight → 24 profit

1 " → $\frac{24}{15} = 1.6$

- take obj 2 first, then obj 3 & then obj 1.

j3 :- 10 weight → 15 profit

1 " → 1.5 "

$(\_ , \underline{1} , \underline{5/10})$

profit = $0 \times 25 + 24 \times 1 + 15 \times \frac{1}{2} = 24 + 7.5 = \boxed{31.5}$

reedy Knapsack will give Optimal soln. always by
iving priority to both profit & weight.

Q. n=5 ✓

Objects	Ob1	Ob2	Ob3	Ob4	Ob5
profits	5	2	2	4	5
weights	5	4	6	2	1

M = 12

Ob1 :- 5 weight → 5 profit
1 " → 1 "

Ob2 :- 4 weight → 2 "
1 " → 0.5 "

Ob3 :- 6 " → 2 "
1 " → 0.33 "

Ob4 :- 2 " → 4 "
1 " → [illegible] "

Ob 5 :- 1 " → 5 "

profits
Ob5 → 5
Ob4 → 4
Ob1 → 5
Ob2 → 2
16

~~12~~ 11
9
4
0

Objects → (1, 1, 0, 1, 1)
Profit = $\boxed{16}$

Q. n=7 M=15

Objects	ob1	ob2	ob3	ob4	ob5	ob6	ob7
profits	10	5	15	7	6	18	3
weights	2	3	5	7	1	4	1
profit:weights	5	1.6	3	1	6	4.5	3
	✓	✓	✓		✓	✓	✓

~~15~~
~~14~~
~~12~~
~~8~~
~~7~~
2

objects: (1, 2/3, 1, 0, 1, 1, 1)

$$\text{Profit} = 10 + \frac{2}{3} \times 5 + 15 + 6 + 18 + 3$$
$$= 52 + 3.2$$
$$= 55.2$$

$$\frac{1 \cdot \times}{3 \cdot}$$

Algorithm

```
for(i=1 to n)
    job[i] = pi/wi ;  → O(n)
Sort job array in decreasing order of pi/wi. → O(nlog
```

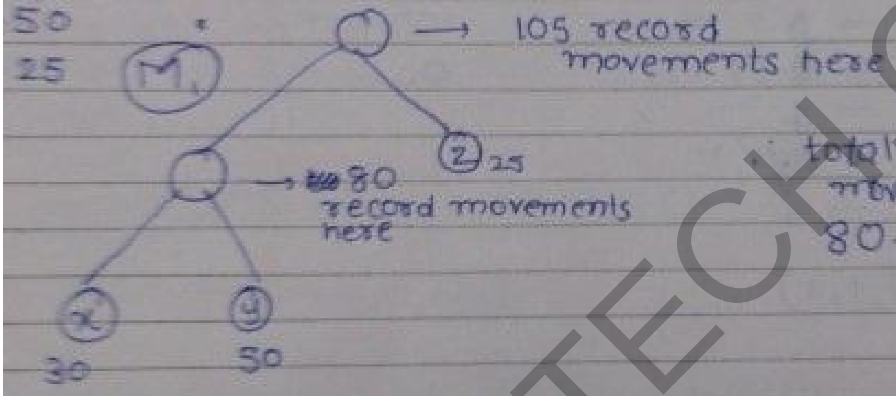
p: profits
w: weights

Take one by one object untill capacity of knapsack beco
s zero ⟶ $\boxed{O(n)}$

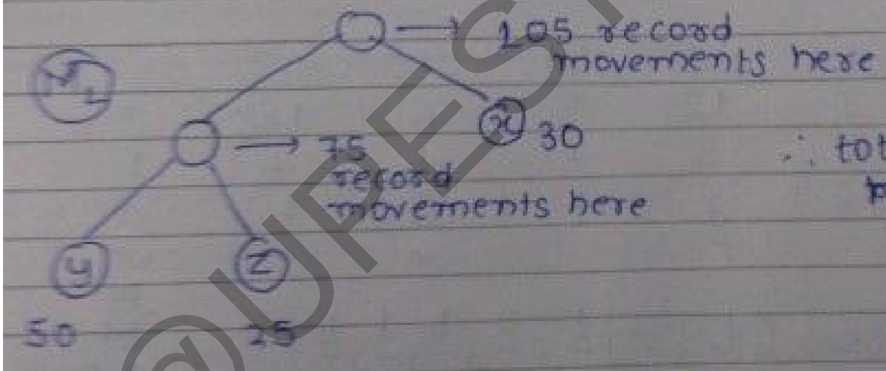
∴ Time complexity $= O(n) + O(n\log_2 n) + O(n)$
$= \boxed{O(n\log_2 n)}$

ptimal Merge Pattern

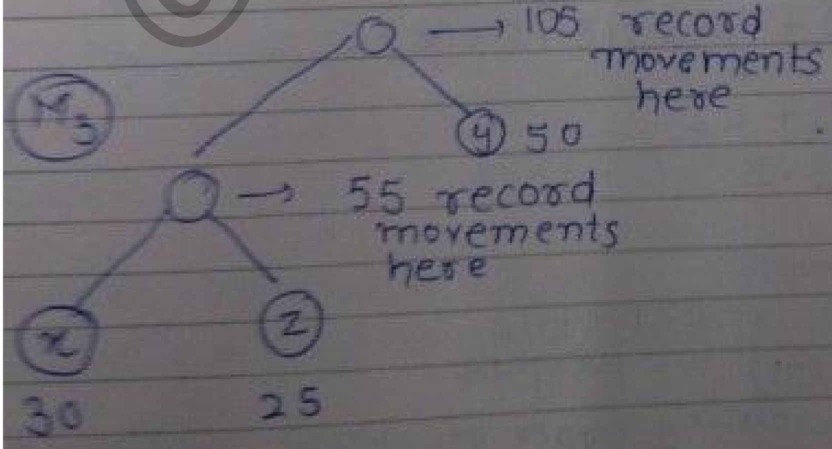
5 files
30
50
25



∴ total record movements = 80+105 = 185 record movemen



∴ total record movements = 75+105
= 180
record
movemen



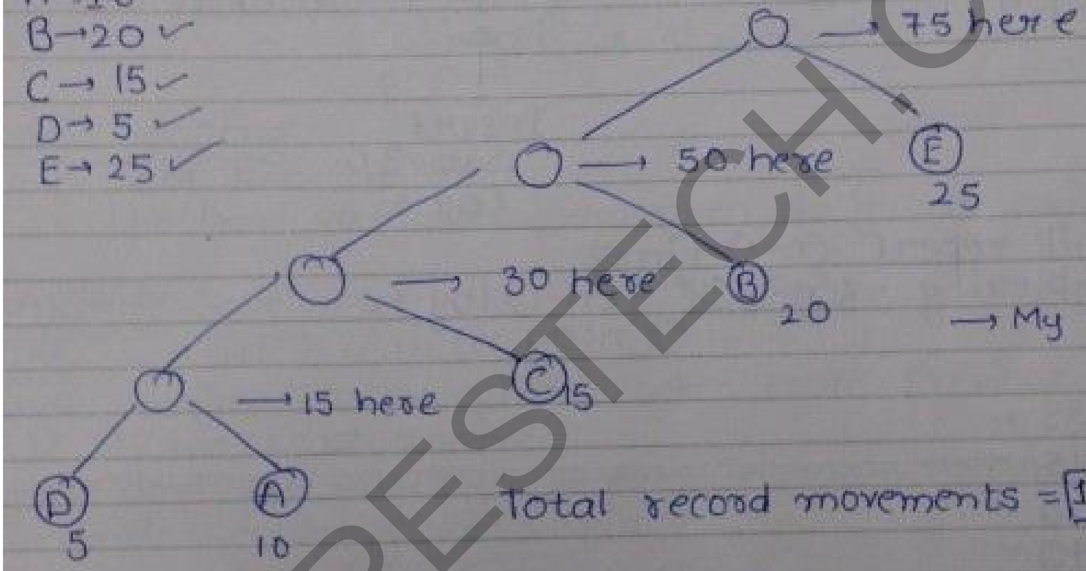
∴ total record movements = 55+1
= 160
record
movem

Cosmos

For the given problem, there are 3 merge patterns, M_1, M_2, M_3, in all of them M_3 is obtimal merge pattern because it is taking least record movements.

e Min. no. of record movements required to merge 5 files

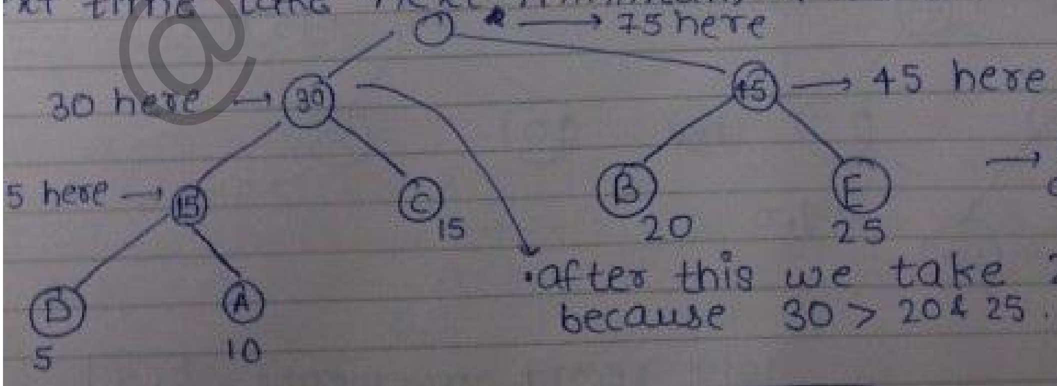
A→10 ✓
B→20 ✓
C→15 ✓
D→5 ✓
E→25 ✓



→ My answer

Total record movements = $\boxed{170}$

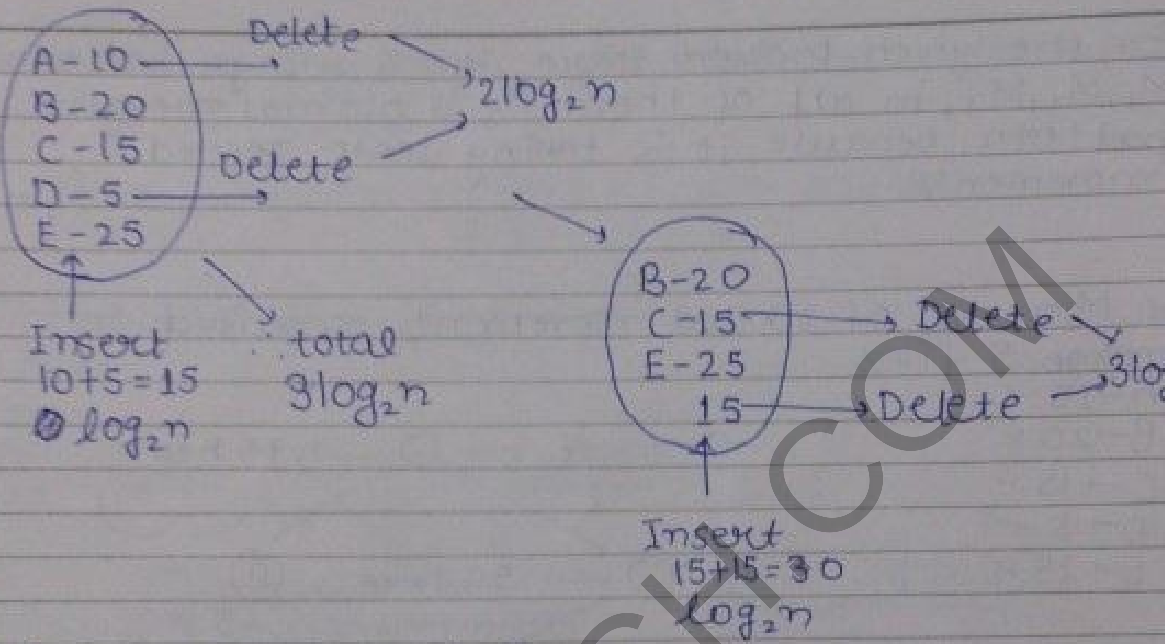
★
r these problems, take ~~1st~~ 1st two minimum first time.
xt time take next minimum & so on.



→ Sir's answer.

after this we take 20 & 25 because 30 > 20 & 25.

75+45+ 30+15 = $\boxed{165}$



This will repeat (n-1) times

∴ Complexity :- 3~~(n-1)~~ 3(n-1) $\log_2 n$

$= O(n\log_2 n)$

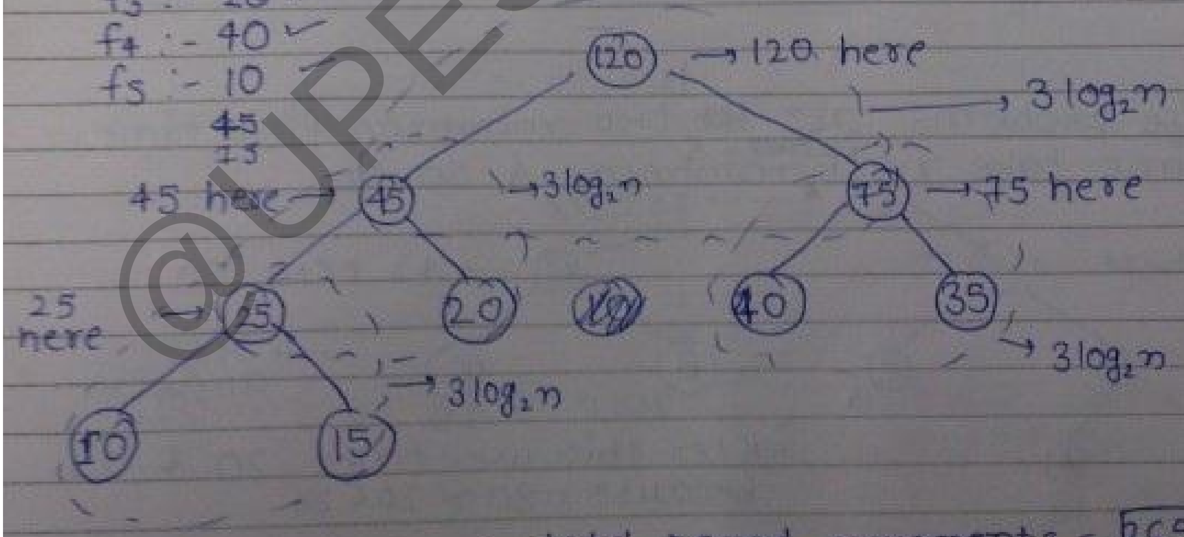
Q. f_1 :- 35 ✓

f_2 :- 15 ✓

f_3 :- 20 ✓

f_4 :- 40 ✓

f_5 :- 10 ✓



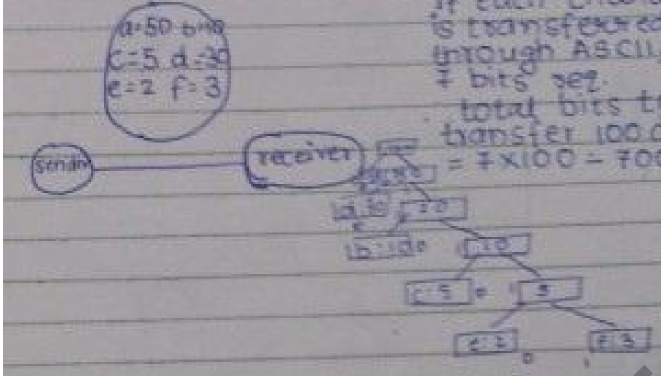
∴ total record movements = $\boxed{265}$

Huffman Coding (~~Huffman coding is best technique because it~~ is application of greedy & hence it gives ~~best~~ optimal solution).

1) Data Encoding

2) Data Compression

Message contains 100 characters



If each character is transferred through ASCII, then 7 bits req.
total bits to transfer 100 characters = 7×100 = 700 bits

compressing

We need to transfer 6 characters, then 3 bits req.,
total bits to transfer 100 characters = 3×100 = 300bits

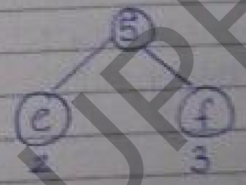
a→0
d→10
b→110
c→1110
e→11110
f→11111

1×50 → 50
2×30 → 60
3×10 → 30
4×5 → 20
5×2 → 10
5×3 → 15
185

Huffman coding

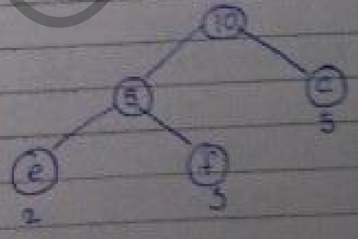
Note:-
More freq ⇒ less bits
Less freq ⇒ more bits

Step 1:-



a=50 b=10
C=5 d=30
5

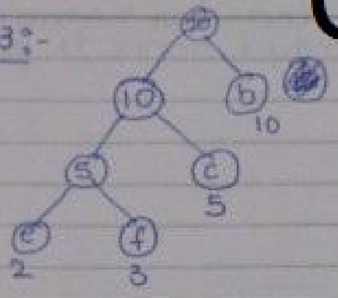
Step 2:- Now 2 minimums are c=5 & 5 from summation of e & f.



a=50 b=10
d=30 10

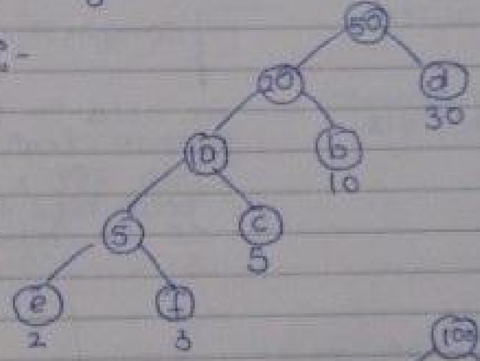
Cosmos

Step3 :-



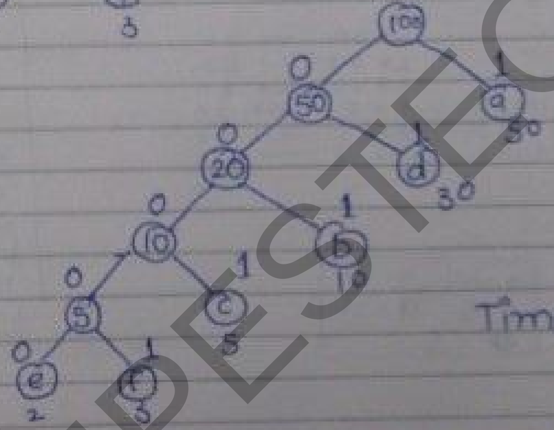
a = 50
d = 30
20

Step4 :-



a = 50
50

Steps :-



← empty

Time Complexity :-
$O(n \log_2 n)$

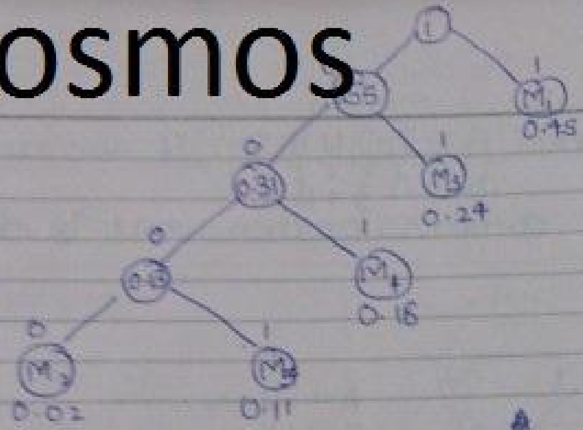
Huffman Coded Tree

a → 1
b → 001
c → 0001
d → 01
e → 00000
f → 00001

→ 185 bits
Average no. of bits = $\frac{185}{100}$ = 1.85

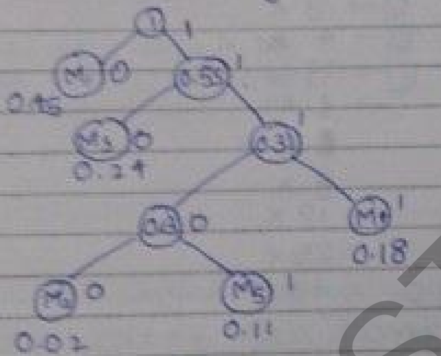
Q. Message M contains
$M = (M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5)$

M_1	M_2	M_3	M_4	M_5
0.45	0.02	0.24	0.18	0.11



M1 → 1 → 1 × 0.45 = 0.4
M2 → 0000 → 4 × 0.02 = 0.0
M3 → 01 → 2 × 0.24 = 0.9
M4 → 001 → 3 × 0.18 = 0.5
M5 → 0001 → 4 × 0.11 = 0.

↑My answer



↑
Sir's answer

M1 → 0
M2 → 1100
M3 → 10
M4 → 111
M5 → 1101

★ Put smaller no. on left hand side & bigger no. on right hand side.

Total no. of bits:-
0.45×1 + 0.02×4 + 2×0.24 + 3×0.18 +
0.11×4 = 1.99 bits
Avg. no. of bits = 1.99 bits.

In Huffman Coding
★ One message will contain 1 bit (but not always)
★ There are two messages with same no of bits.

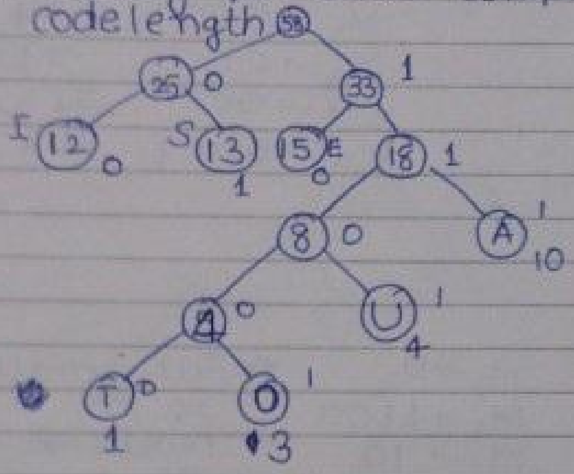
Q. Decode the ~~mes~~ encoded message:-
101100110111111001100101111101

Step 1: See the tree & start from root.
1 → go right from root
10 → go left from 0.55 & we reach leaf,
∴ M3 is detected.
1 → go right from root

Cosmos

A file contains characters A, E, I, O, U, S, T ,if we use Huffman coding for data compression, then what is the avg codelength

A↓10, E↓15, I↓12, O↓3, U↓4, S↓13, T↓1



A → 10 ×
E → 15 ×
I → 12 ×
O → 3 ×
U → 4 ×
S → 13 ×
T → 1 ×
4 ×
8 ×
18 ×
25 ×
33 ×

A → 111 → 3 × 10 = 30
E → 10 → 2 × 15 = 30
I → 00 → 2 × 12 = 24
O → 11001 → 3 × 5 = 15
U → 1101 → 4 × 4 = 16
S → 01 → 2 × 13 = 26
T → 11000 → 5 × 1 = 5

146 → total bits

$$\text{Avg codelength} = \boxed{\frac{146}{58}}$$

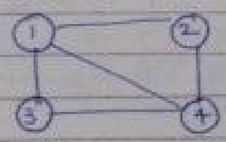
Q. If $f(n) = O(g(n))$
T/F ?
a. $\log(f(n)) = O(\log(g(n)))$ ✓
b. $2^{f(n)} = O(2^{g(n)})$ × → take $f(n) = 2n$
c. $f(n) = O(f(n/2))$ × → take $f(n) = 2^{2n}$
d. $f(n) = O(f(n))^2$ × → take $f(n) = \frac{1}{n}$

$f(n) = O(g(n))$
$n^2 < n^3$
$n^2 = O(n^3)$
~~$2\log n = \Theta(3\log n)$~~
(applying log on both sides)
$n! = O(n^n)$
$\log n! = O(n\log n)$
$f\ n^3 = \Omega(n^2)$
$3\log n = \Theta(2\log n)$
★ Sometimes they becomes equal on application of log on both sides.

Minimum Cost Spanning Tree

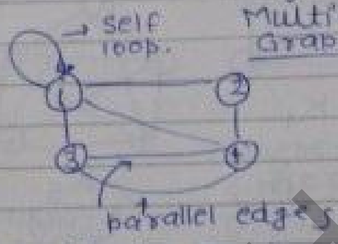
G(V,E)

Simple Graph



① No self loop.
② No parallel edges

Multi-Graph



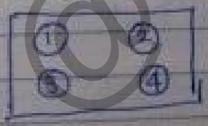
① Self loops or parallel edges

★ In a simple graph with 'n' vertices, then at max. each vertex can have (n-1) degee.
★ In multigraph with 'n' vertices, then at max. each vertex can have infinite degree.

Types Of Simple Graphs :-

★ All null graphs are regular graphs but all regular graphs are not null graphs.

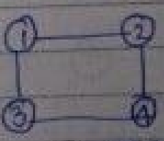
1. Null Graphs



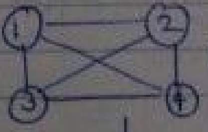
if the degree of every vertex is zero, then it is null graph.

2. Regular Graph → no. of edges = $\frac{n \times m}{2}$ (n → no. of vertices, m → degree of each vertex)

if the degree of every vertex is same, then it is regular graph.



2-regular graph.
↓
$n = 4$
$m = 2$
∴ no. of edges $= \frac{4 \times \cancel{2}}{\cancel{2}} = 4$

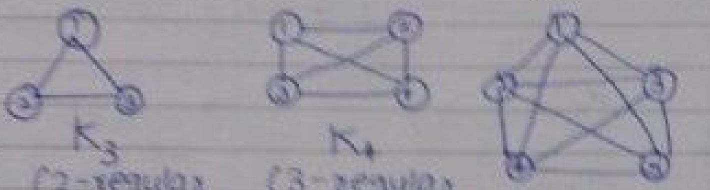


3-regular graph.
$n = 4$
$m = 3$
∴ no. of edges $= \frac{\cancel{4}^{2} \times 3}{\cancel{2}} = 6$

All complete graphs are regular but all regular graphs are not complete

Complete Graph:-

If each vertex is connected to all other vertices, then it is called complete graph.



K_3 (2-regular graph) $\frac{3\times2}{2}=3$

K_4 (3-regular graph) $\frac{4\times3}{2}=6$

K_5 (4-regular graph) $\frac{5\times4}{2}=10$

Maximal Graph

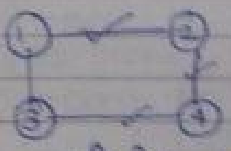
degree of each edge

$$\boxed{\frac{(n-1)\times n}{2}}$$

→ total no. of vertices

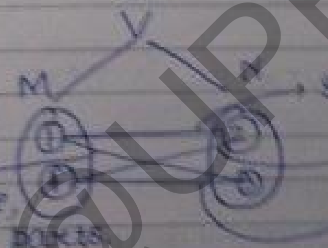
→ total no. of edges

Bipartite Graph



$V=\{1,2,3,4\}$

divide the vertices into M & N two parts & all the edges must be b/w M & N only & not within M only or N only.

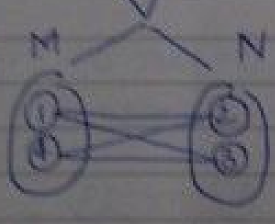


p(iv):- & 5 ve edge, diff. parts

→ Step (i):- ① & ② have edge ∴ put them in M & N resp.

→ Step (ii):- ② & ④ edge, ∴ different parts

→ Step (iii):- ③ & ④ edge, ∴ " "

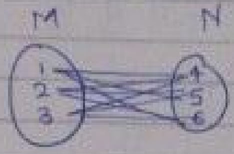


But ①—④ not possible, ∴ it is not bipartite graph.

★ Connected Graph :-

In a graph each ~~vert~~ there is a path b/w every pair of vertices. Min. no. of edges = V-1 [V:- no. of vertices]

Complete Bipartite



every element in M is adjacent ~~to~~ each element in N, so it is complete bipartite graph

G(V,E)

$|E| \leq \frac{V(V-1)}{2}$ [in simple graph] → but possible for multigraphs too

★ $|E| = O(V^2)$

apply log on both sides:-

$\log E = O(2 \log V)$

$\boxed{\log E = O(\log V)}$

∴ $E \log E \Rightarrow E \log V$ (from this)

No. of graps

n=2

∴ no. of graphs = 2

n=3

~~no~~ no. of graphs = $2^{\text{max degr}}$
$= 2^{n(n-1)/}$

simple

★ $\boxed{\text{no. of graphs with 'n' vertices} = 2^{\frac{n(n-1)}{2}}}$

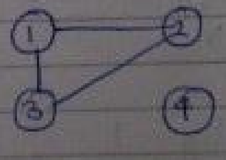
Spanning Tree :-

A connected subgraph H of a given Graph G(V,E) is said to be spanning tree iff

① It should contain all vertices of G.

② Edge H should contain (V-1) edges if G contains V-vertices.

e.g.

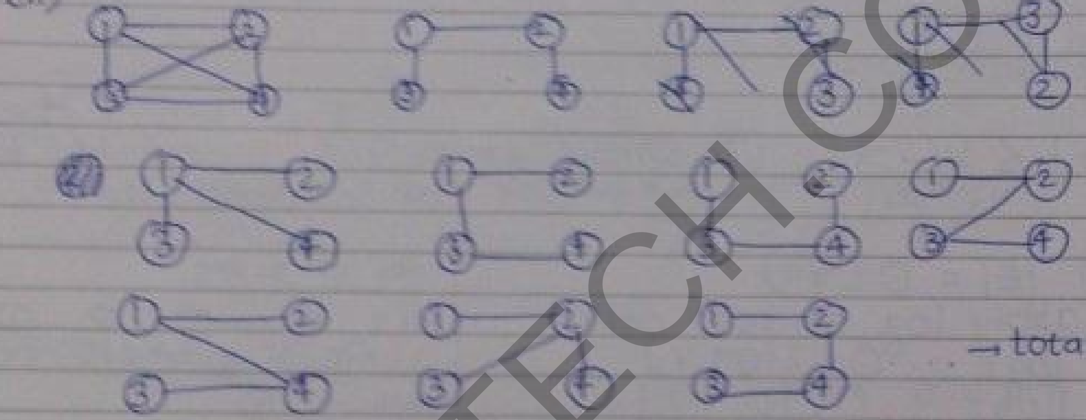


→ it contains all vetices & contains (V-1) edges, so it should be spanning tree, but it is not connected subgraph.

Q. Find no. of spanning trees for the following graph :-

(i)

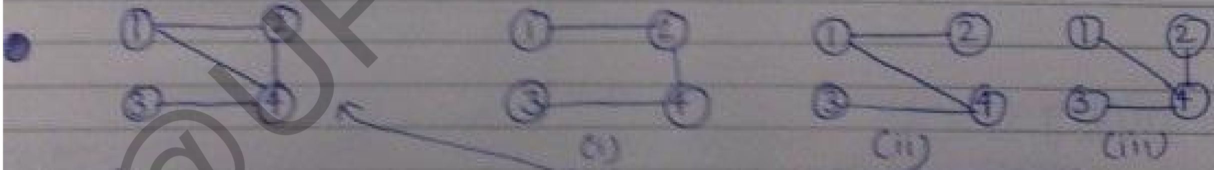
(ii)



→ total:16

☆ The no. of spanning trees in a 'n'-vertex $\boxed{\text{complete}}$ $\boxed{\text{graph}}$ = $\boxed{n^{n-2}}$.

Q Find no. of spanning trees:- [when graph is not complete]



$\boxed{O(n^2)}$

Kirchaff Theorem

① Make Adjacency Matrix

$$M = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}$$

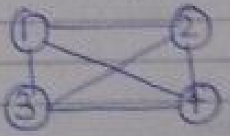
) (i) replace all non-diagonal ~~one~~ 1's by -1.
(ii) replace all diagonall zeros by its equal degree.

$$M = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 2 & -1 & 0 & -1 \\ 2 & -1 & 2 & 0 & -1 \\ 3 & 0 & 0 & 1 & -1 \\ 4 & -1 & -1 & -1 & 5 \end{array}$$

③ Co-factors of any element will give no. of spanni
trees.

$$C_{11} = \begin{vmatrix} 2 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 3 \end{vmatrix} = 2(3-1) - 1(1)$$
$$= 2 \times 2 - 1 = \boxed{3}$$

Q. Find no. of spanning trees for following graph:-



Step (i):- Adjacency Matrix

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

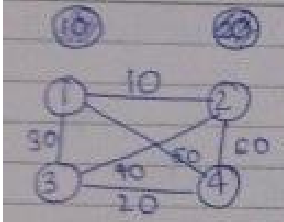
Step (ii):-

$$M = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

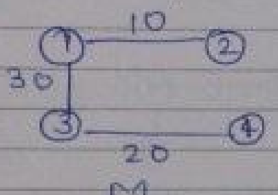
$$C_{11} = \begin{vmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{vmatrix} = 3(9-1) + 1(-3-1) - 1(1+3)$$
$$= 24 - 4 - 4$$
$$= 16$$

Minimum Cost Spanning Tree

① Covering all vertices of the graph with min. cost of the tree edges.



- for the graph 16 spanning trees are possible, in all of them M is the min. cost spanning tree.

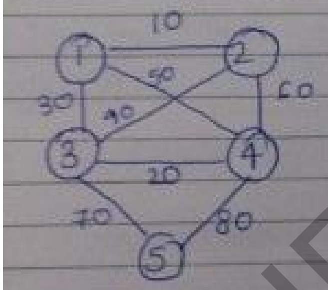


M.

★To find out Min. Cost Spanning tree, we have 2 algorithms:-

) Krushkal
i) Prim's

Krushkal :-



Step(i) :- Take 1st min.

edge weight	starting pt.	ending pt.
★ 10	1	2
20	3	4
30	1	3
40	3	2
50	1	4
60	2	4
70	3	5
80	4	5

Now, make a min heap for all the edges, so no. of edges = E, ∴ time complexity = $\boxed{O(E)}$

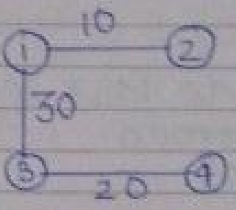
Now take 1st min. (Delete it from heap).

Deletion from heap :- $O(\log_2 n)$
but $n = E$ ∴ $\boxed{O(\log_2 E)}$

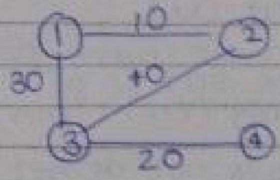
ep (ii) :- Take 2nd next min.

- In best case we will get min. cost spanning tree when 1st (V-1) edges do not make any cycle.
  $\therefore$ (V-1) times $\log_2 E + O(E) = (V-1)\log_2 E$ O(E)
  $= \Omega(V\log$

- take next min. :- (this also takes $\log_2 E$)

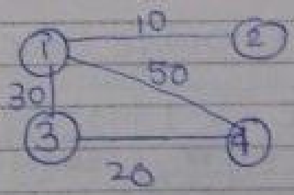


- take next min. :- (this also takes $\log_2 E$)



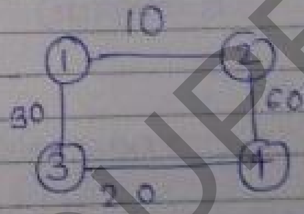
→ cycle → so delete this edge.

- take next min. :- (this also takes $\log_2 E$ from min. heap)



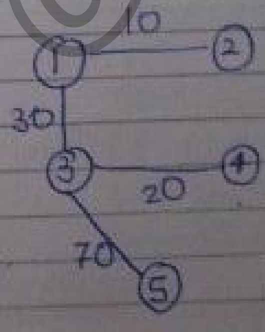
→ cycle, so delete this edge

- take next min. :-



→ cycle, so delete this edge

- take next min. :-



→ no cycle, & all vertices are covered, so this is min. cost spanning tree.

Algorithm:-

① Create Min heap. ⟶ O(E) → $\log_2 E$

② Delete one by one min. & ~~add to MST if no~~ cycle add to MST if no-cycle formed untill (V-1) edges are added to MST

Best Case:- When we get MST first (V-1) times without getting a cycle at all.

(i.e. when we have to execute step ② of algo only (V-1) times)

$E + (V-1)\log E$

$= E + V\log E$

but $\log E = O(\log V)$

$= E + V\log V$

$= \boxed{O(E + V\log V)}$ → Best Case & Average Case

when graph is not dense:- $\Omega(V\log V)$

when graph is dense:- $O(V^2)$

in best case:- E = V-1 → when graph contains (V-1) edges only.

in worst case:- E = $\frac{V(V-1)}{2} = O(V^2)$ (when graph is complete)

Worst Case:-

(when we have to execute step ② of algo E times.)

$E + E\log E$

$= E + E\log V$

$= \boxed{O(E\log V)}$

☆ In Kruskal's algorithm, in middle we may get disconnected graph (Forest), but in case of Prim's algo, we always get connected graph.

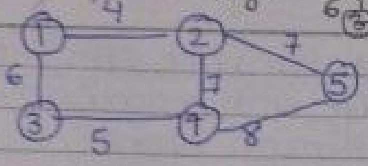
Prim's

Consider the following graph:-

* if there are more than one MST for a given connected graph, then ~~all~~ the graph contain atleast ~~one~~ one edge weight which is the edge weight of more than one edges.

e.g.

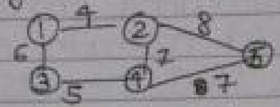
has 2 MST's, so it must contain atleast 2 edges with same weight.



How many diff. min. cost spanning trees are possible?

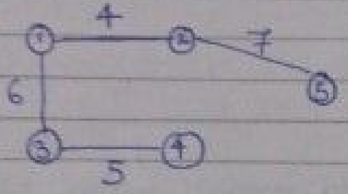
* but converse is not true, i.e. if graph contains edges with same weight, then it might not have more than one MST,

e.g.



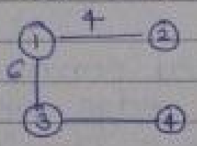
has only one MST.

Ans. Using Krushkal :-



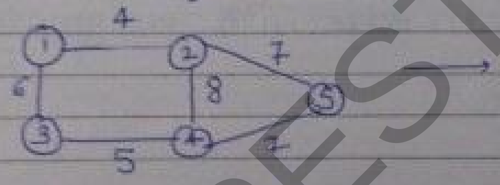
Only 1

at this stage :-

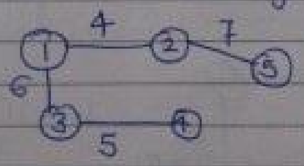


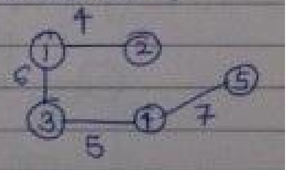
next min. is 7 which is b/w ② & ④ and ② & ⑤.
but ② & ④ makes a cycle, so we can't use it,
∴ we can only use edge b/w ② & ⑤, & we stop because we have 4 edges now.

but if the graph is :-



∴ min spanning trees are :-





1

~~If there are~~

★★

Note: ① For the given graph more than 1 MST may be possible.

implies that * ← ② For the given graph, if there are more than one MST, then atleast one of the edge weight will be repeated.
(but converse is not true)

✓ ③ If the given graph is connected & no edge weight is repeated, then exactly one MST is possible.

④ There will be no MST if graph is disconnected.

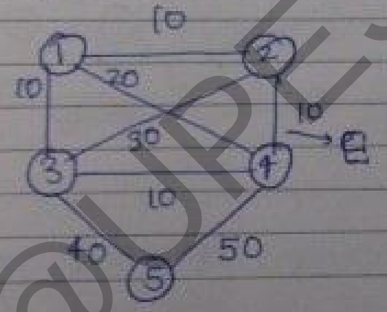
☆☆ If edge weight is not given, then we assume it to be 1.

Q. Let G be an undirected connected graph with distinct edge weights. Let emax be the edge with maximum weight & emin be the edge with minimum weight, then check following statements are T/F?

T (a) Every MST should contain emin.
T (b) If emax is in MST, then its removal must disconnect G.
F (c) No MST contain emax.
T (d) G has unique MST.

Q. Let G be an undirected connected graph with N vertices. If 'W' is the min. edge weight among all edge weights in G & E be a specific edge with weight 'W'. Then

(a) E may be there in MST.
T (b) If E is not in MST, then all edges ~~have~~ are having weight 'W'. in that cycle.
(c) Every MST should contain an edge with weight 'W'.
(d) Every MST should contain E.

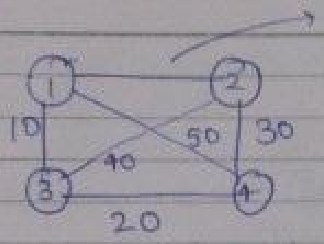


W = 10

edge weight is not given, assume it to be 1.

An undirected graph G has 'n' nodes, the adjacency matrix is given by n×n square matrix. whose
(i) Diagonal elements are zero's.
(ii) Non-Diagonal elements are one's } → connected graph.
then check :-
a) G has no MST.
b) G has multiple MST's with diff. costs. :- multiple MST's with same cost may be possible.
c) G has unique MST of cost n-1.
d) G has multiple MST's each of cost n-1.

✓✓

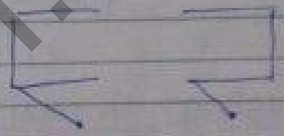


if ~~not~~ there is no edge weight, we assume it to be 1.

→ MST
↓
31

Q. Let T & T' be 2-spanning trees of a connected graph G. Suppose that an edge e is in T but not in T' & edge e' in T' but not in T & remaining edges are assumed to be same, then which is T after perform following opⁿ on T & T'.

(i) $(T - \{e\}) \cup \{e'\}$
(ii) $(T' - \{e'\}) \cup \{e\}$

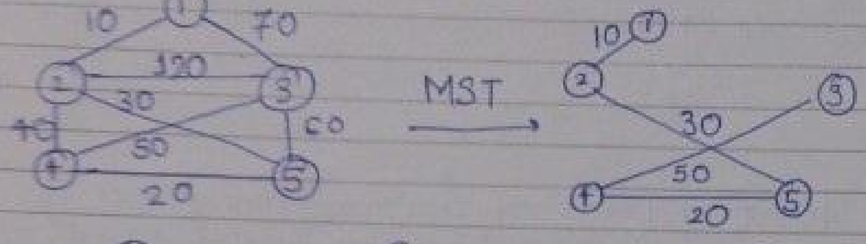


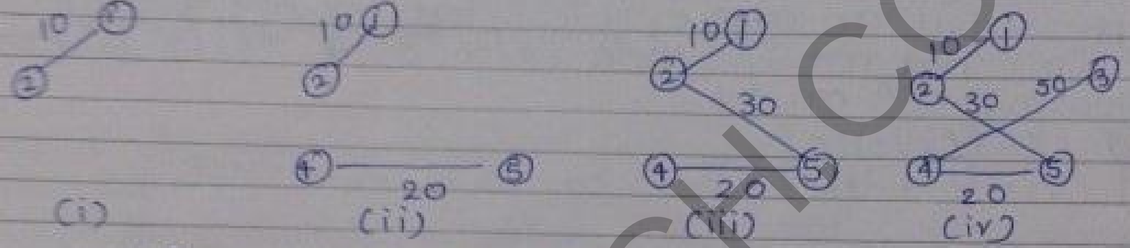
(a) Both T & T' are ST.
(b) T → ST but not T'
(c) T' → ST but not T
(d) both are not ST.

My answer:- removing e from T & adding e' makes it T' &
removing e' from T' & adding e makes it T.

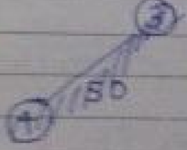
☆ if the remaining edges are not same:-
✓✓

Prim's Algorithm

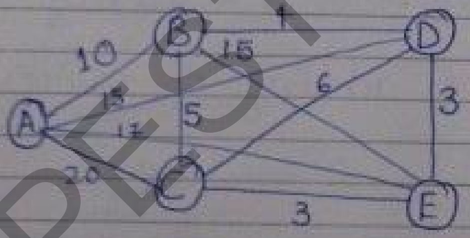




Using Kruskal algo

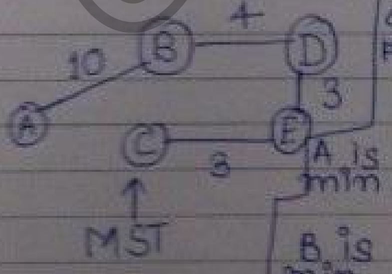


Apply Prim's algorithm :-



★

Decrease Key & Increase Key Opn in min heap & max. heap takes:- Worst Case :- O(log₂ n)
Best Case :- Ω(1)



MST

Build-heap takes $\boxed{O(V)}$ → $\boxed{\text{Decrease Key Operation}}$

	A	B	C	D	E
distance:	∞	∞	∞	∞	∞
parent:	N	N	N	N	N
	0	∞	∞	∞	∞
	N	N	N	N	N
		10	20	15	17
		A	A	A	A
			5	4	6
			B	B	B
			5		3
			3B		B

A is min.

B is min.

delete it from min. heap take O(log₂V)

This d/lo
takes pla
V time
∴ $\boxed{V^2 \log}$

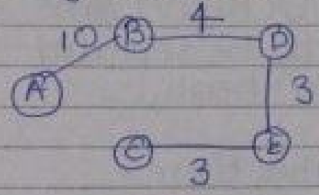
every decre
key opⁿ take
log₂V times
at max whe
to perform
decrease key

D is opⁿ
V-1
min verte
∴ V log V

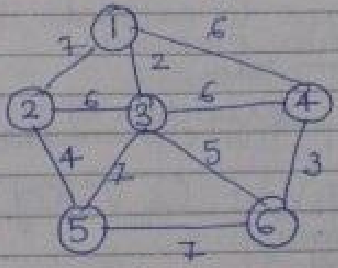
E is mi

Now,

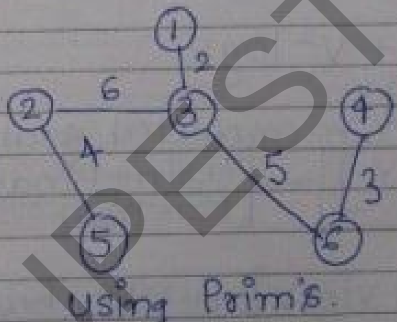
using Kruskal's



Apply Prim's algo on the following graph:-



	0	6 ~~7~~	2	3 ~~6~~	4 ~~7~~	5
distance:-	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	1	2	3	4	5	6
parent:-	N X	~~N~~ ~~1~~ 3 [illegible] X	~~N~~ 1 X	~~N~~ ~~2~~ 6 X	~~N~~ ~~3~~ 2	~~N~~ 3 X



using Prim's.

Note:-

Step (i) Build-heap of all the vertices :- takes O(V) time.

(ii) Take min out of the min-heap :- takes O(1) time, rearranging heap will takes:- $O(\log_2 V)$.

(iii) Now, change the distance value using decrease key opⁿ:- one decrease key opⁿ on one vertex takes $\log_2 V$ time.

(iv) Now, at max. in one stage all the vertices will have to undergo decrease key opⁿ, so time taken :- $O(V \log_2 V)$.

(v) Now, we take out next min. from heap & have to rearrange the heap, it takes :- $O(\log_2 V)$ time.

(vi) Now, the decrease key opⁿ have to be repeated till only one vertex is left in min-heap, ∴ the total no. of stages = V.

∴ ~~decre~~ total decrease key opⁿ takes time :-

$$\underbrace{V}_{\text{V times decrease key op}^n} \times \underbrace{V\log_2 V}_{\text{~~decr~~ time complexity of decrease key op}^n \text{ in one stage.}}$$

My answer :-

we have to extract 'V' min. elements from heap

extraction of next min. element from heap

∴ Time Complexity = $O(V\log_2 V) + O(V^2\log_2 V) + O(V$

$= \boxed{O(V^2\log V)}$

Sir's Answer :-

Time Complexity = $V + V\log V + V^2\log V$

$= (V+V^2)\log V$

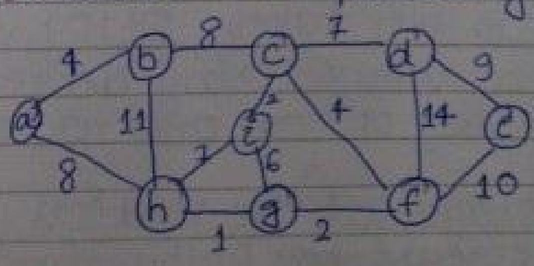
But $V^2 = E$ (in completed graph)

$\boxed{O[(V+E)\log V]}$ → [Using Binary Min. heap]

Time Complexity :- $O[E + V\log V]$ [using Fibonacci min heap]

:- $O(V+E)$ [using Binomial min heap]

Consider the following graph



Cosmos

which one of the following sequence of edges of MST is not true using Prim's algo when the algo. Started from vertex a.

(a) (a,b) (b,c) (c,i) (c,f) (f,g) (g,h) (c,d) (d,e)
(b) (a,b) (b,c) (f,c) (c,i) (f,g) (g,h) (c,d) (d,e)
(c) (a,b) (a,h) (h,g) (g,f) (f,c) (c,i) (c,d) (d,e)
(d) (a,b) (b,c) (h,g) (f,g) (c,i) (c,f) (c,d) (d,e)

(d) is wrong because it creates unconnected graph.

distance	0	~~∞~~ 4	~~∞~~ 8	~~∞~~ 7	∞	~~∞~~ 4	~~∞~~ 6	~~∞~~ ~~8~~ 7	~~∞~~ 2
	~~a~~	~~b~~	~~c~~	d	e	~~f~~	g	h	~~i~~
parents	N	~~N~~ a	~~N~~ b	~~N~~ c	N	~~N~~ c	~~N~~ i	~~N~~ ~~a~~ i	~~N~~ c

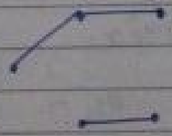
(b) is wrong because using Prim's after (a,b) & (b,c) → (c,i) should come but (f,c) is coming.

(a)

(b)

(c)

(d)



← This is wrong because it created undi graph.

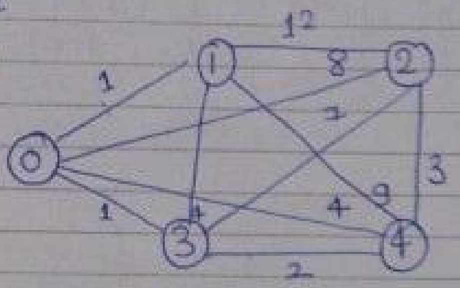
(a) & (c) are correct using Prim's.

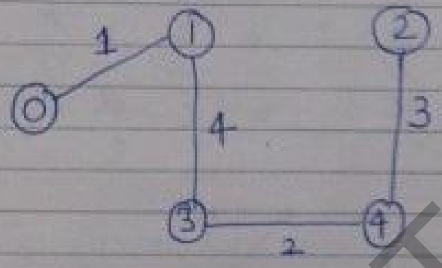
Q. Consider a complete undirected graph with vertex set V = {0, 1, 2, 3, 4}
Entry W_{ij} in matrix W below is the weight of t edge {i, j}.

$$W = \begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 8 & 1 & 4 \\ 1 & 1 & 0 & 12 & 4 & 9 \\ 2 & 8 & 12 & 0 & 7 & 3 \\ 3 & 1 & 4 & 7 & 0 & 2 \\ 4 & 4 & 9 & 3 & 2 & 0 \end{array}$$

What is the cost of ~~the~~ MST for the above graph such that vertex 0 is a leaf node in that spanning tree?

(a) 8
(b) 9
(c) 10 :- My answer
(d) 11

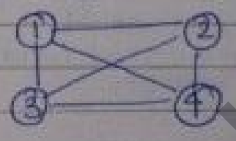




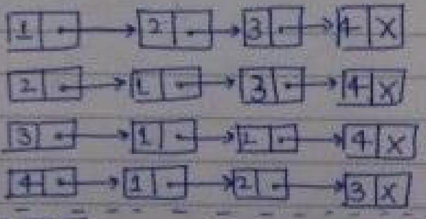
(using Krushkal's)

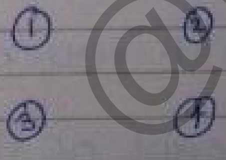
Graph Representations

① Adjacency Matrix

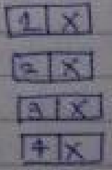


$$\rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}$$





$$\rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \end{array}$$

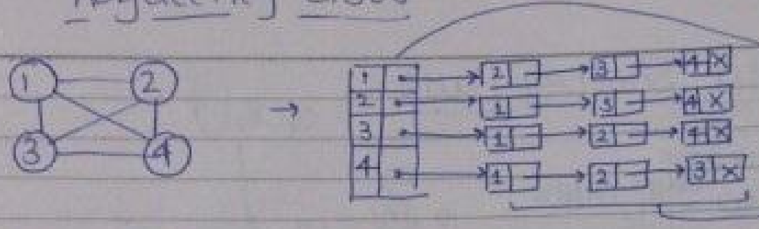


o represent graph Adjacency matrix will take $O(V^2)$ → always

o find out the degree of a vertex, it will take $\Theta(V)$.

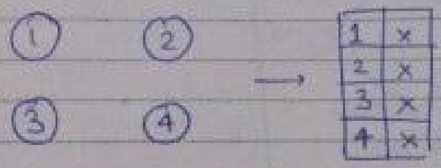
o find out that the graph is connected, it will take $\Theta(V^2)$.

Adjacency List:-



Space Complexity:-
$V + 2E = \boxed{O(V+E)}$
or
$\boxed{\Theta(V+E)}$

whichever is big, it will be the answer.



~~Spa~~ In this case :-

★ To find out the degree of ~~each~~ vertex :-

$\Omega(1)$ → when no adjacent neighbours.

$O(V)$ → when graph is connected.

(Sparse graph)

★ When less no. of edges :- Adjacency List

★ When more no. of edges :- Adjacency Matrix

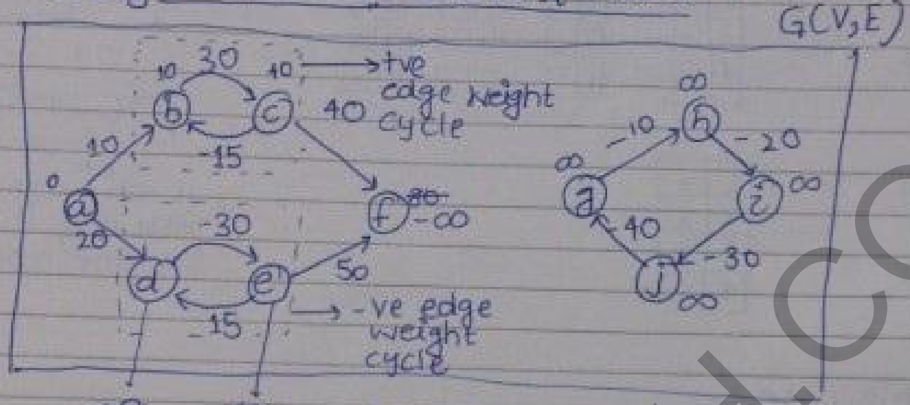
(dense graph)

Single Source Shortest Path

Date

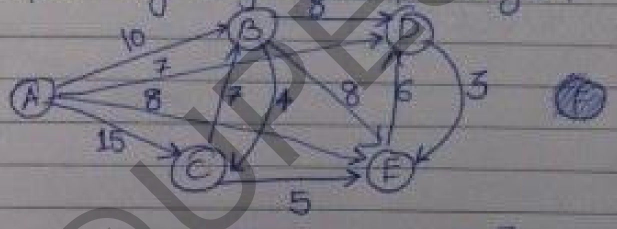
21.07.12

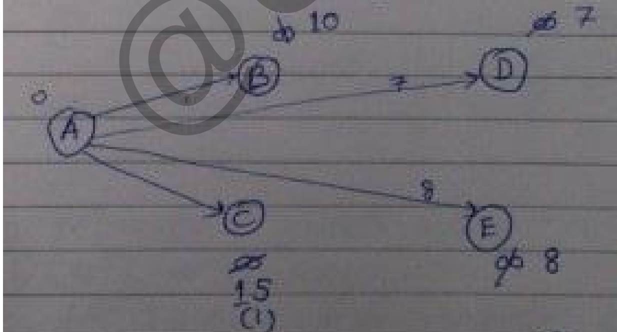
Single Source Shortest Path



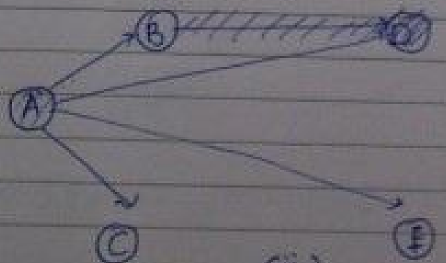
- MC(A,B) = ∞ if there is no path b/w A & B
- MC(A,B) = -∞, B is participant of negative edge weight cycle or dependent on -ve edge weight cycle
    - d, e ⟶ participants
    - f ⟶ dependent

Q. Find shortest path from vertex 'a' for the following graph using dijkstra's algo?

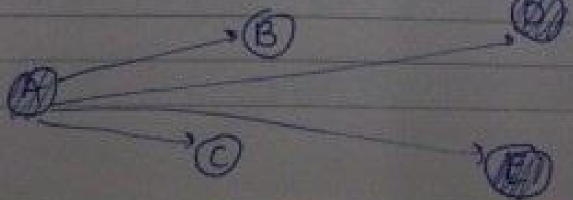




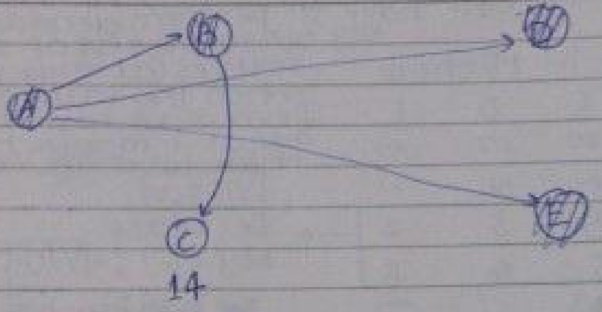
(i)



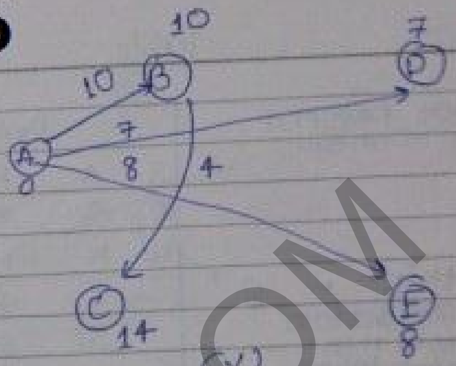
(ii)



(iii)

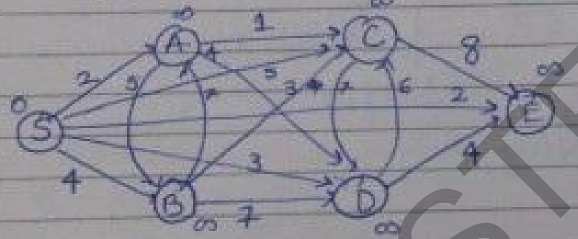


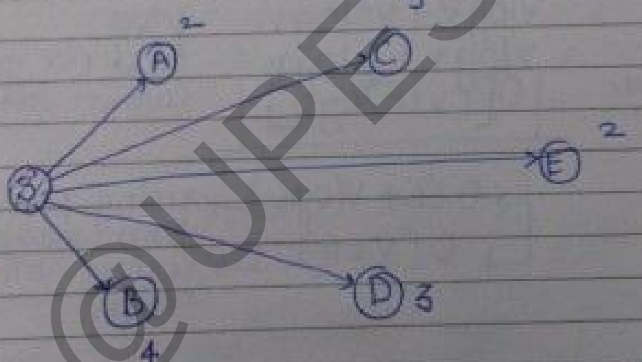
(iv)



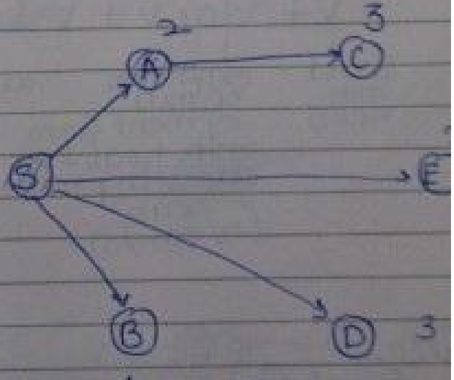
(v)

Q. Apply Dijkstra starting from vertex 'S'.

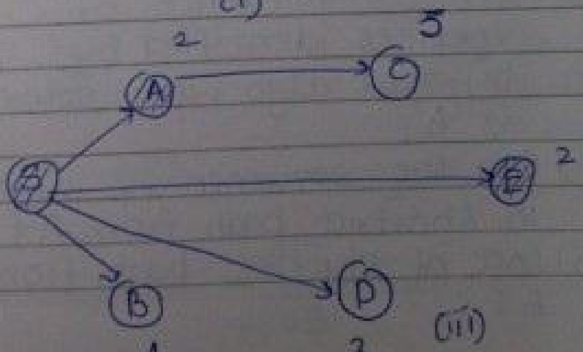




(i)

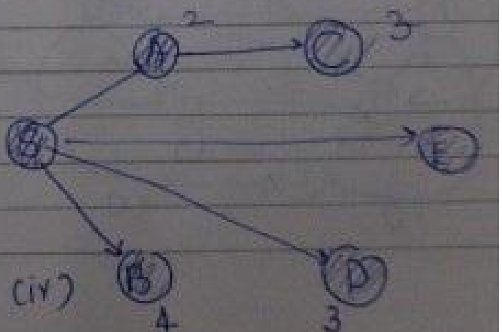


(ii)



(iii)

(iv)



Explanation:

Table

Not present in Minheap	S	A	B	C	D	E
	0	∞	∞	∞	∞	∞
	N	N	N	N	N	N
S		3	4	5	3	2
		S	S	S	S	S
A			4	3	3	2
			S	A	S	S
E			4	3	3	
			S	A	S	
C			4		3	
			S		S	
D			4			
			S			
B						

this extraction from min heap takes $\log_2 V$ time ← S

$\log_2 V$ ← A

$\log_2 V$ ← E

$\log_2 V$ ← C

$\log_2 V$ ← D

$\log_2 V$ ← B

$V\log_2 V$

$V\log_2 V$

$V\log_2 V$

$V\log_2 V$

Decrease Key opn. :-

- Delete the min. from min. heap → O(1)
- Adjust the root :- $\log_2 V$
- We have to perform decrease key opn to the rest of the vertices (at max)
- When the value change from ∞, we have to adjust it in min. heap, which takes $\log_2 V$ time,

$\boxed{V\log_2 V}$

→ Decrease key opn :- $\boxed{V\log V}$

~~Be~~ Deletion on min. → O(1)

at max we have to perform decrease opn. (V-1) times.

(V)$\log_2 V$.

∴ Time Complexity:-

$$\underset{\substack{\text{build}\\\text{min.}\\\text{heap}}}{V} + \underset{\substack{\text{extraction of min.}\\\text{element from min. heap \&}\\\text{adjusting min. heap}\\(V\text{ times})}}{V\log_2 V} + \underset{\substack{\text{all decrease}\\\text{key opn}}}{V^2\log_2 V} = V\log V + V^2\log V$$

$$= V\log V + E\log V$$

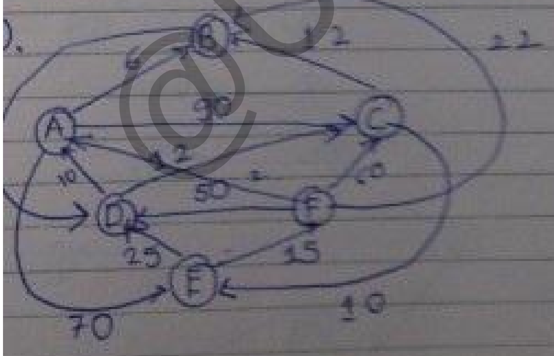
$$= \boxed{O[(V+E)\log V]}$$

$V^2 = E$ in dense graphs (worst case)

↳ $O(E + V\log V)$ → using fibonacci min heap

↳ $O(V+E)$ → using binary heap.

).

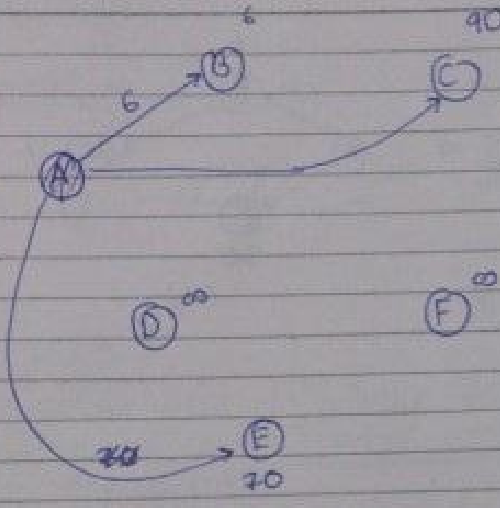


(i) O/P the sequence of vertices identified by Dijkstra algo when source is 'A'.

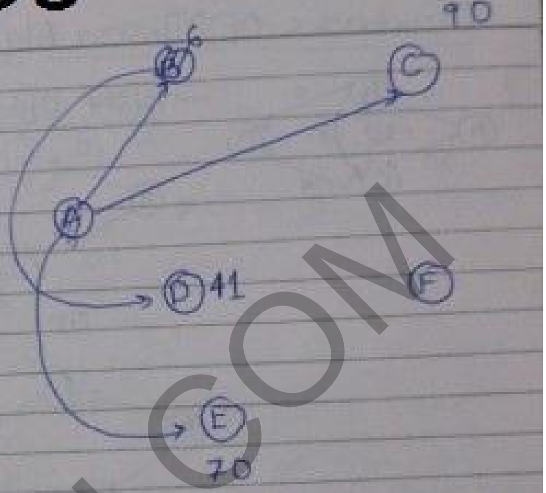
(ii) O/P the sequence of vertices in shortest path from A-E

(iii) Cost of shortest path from A-E?

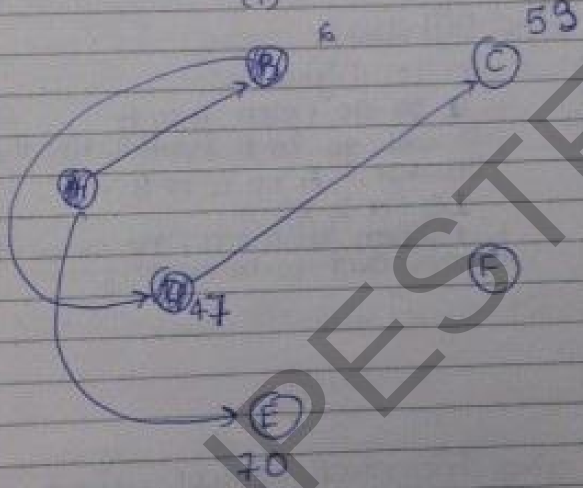
Cosmos



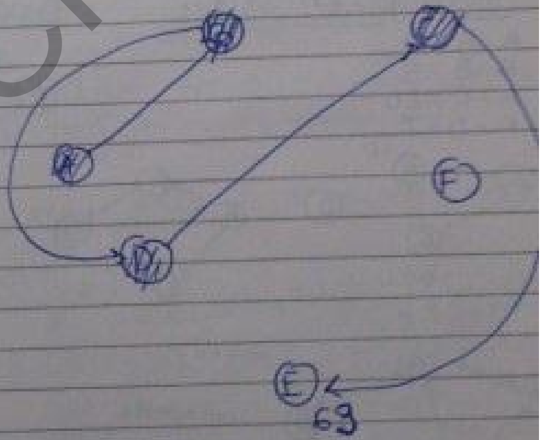
(i)



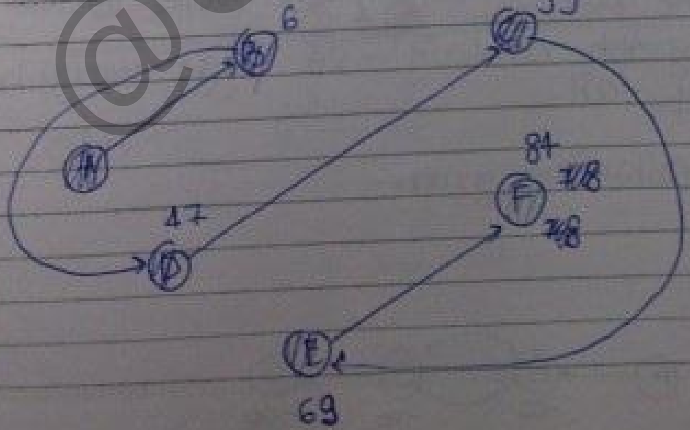
(ii)



(iv)

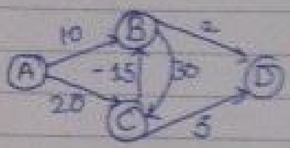


(v)

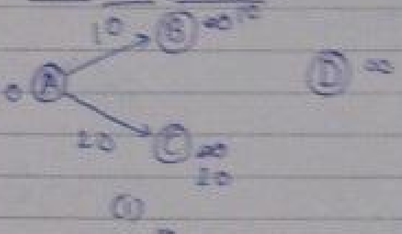


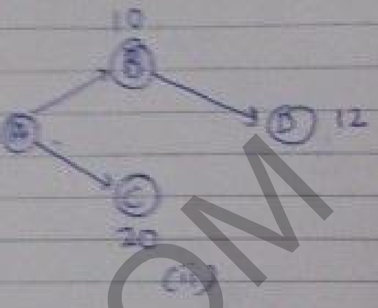
Ans. (i) A - B - D - C - E - F
(ii) A - B - D - C - E
(iii) 69

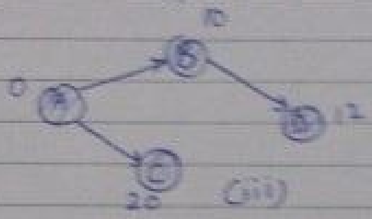
Drawbacks of Dijkstra Algo



→ Apply Dijkstra:-

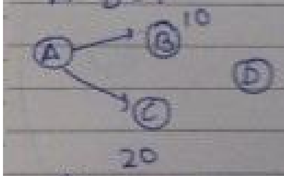




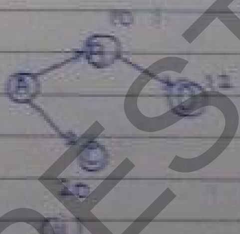


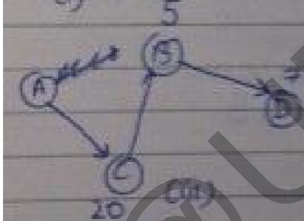
→ Using Manually

A-A = 0
A-B = 5
A-C = 20
A-D = 7

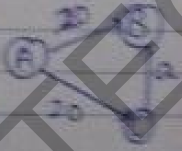


(i)





Drawback of Dijkstra:-



acc to Dijkstra:-

1 to go from A to B

it will go to B from A directl

because:- A to C to B:-

10+a > 20

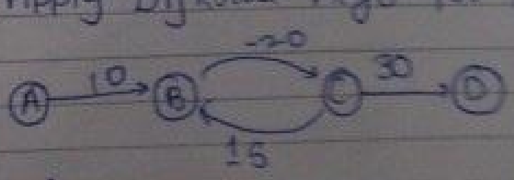
but when a is -ve, the answe comes out to be wrong.

Which vertices gives incorrect result using Dijkstra:- to go to B:- Ⓐ to Ⓒ to Ⓑ = 5

& then vertices reachable from Ⓑ also faces the problem. Ⓑ & Ⓓ

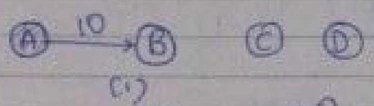
Q. ★ Note:- If the given graph contain -ve edge weights then Dijkstra algo may fail.

Q. Apply Dijkstra Algo for following graph:-



using manually:-

Dijkstra :-


(i)


(ii)


(iii)

~~⇒~~ how many vertices gives wrong answer :-

Note :-

- ★ Dijkstra will defmitely fail when there is a -ve edge weight cycle in the graph.
- ★ Dijkstra will work fine for -ve edge weight when we eliminate a vertex after using the vertex 2 times.
- ★ Dijkstra will work fine for -ve edge weight cycle when we eliminate a vertex after using it (V-1) times.

but we can't take this edge because B is out of the min heap ∴ B is affected, & hence we can go to C via B, ∴ C is also affected & we can go t D via C, ∴ D is also affected.

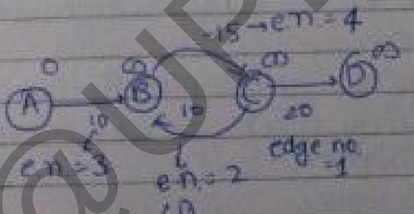
Bellman-Ford Algo :-

Note :- ① Bellman Algo takes O(V.E) time. using bellman ford

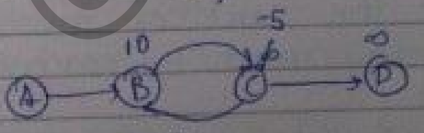
the algo is repeated (V-1) times.

each time all the edges are calculat

Q.

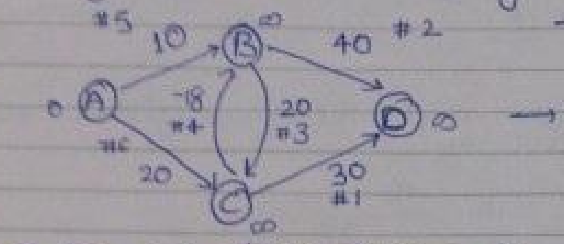

(i)


(ii)

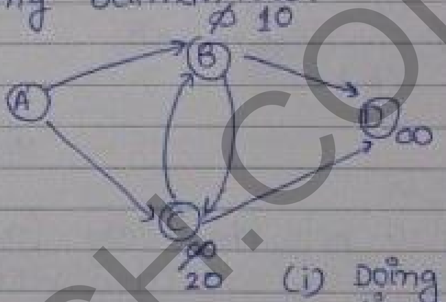

(iii)


(iv)

Apply Bellman-Ford algo:- → V=4 → we have to do (V-1)=3 times.

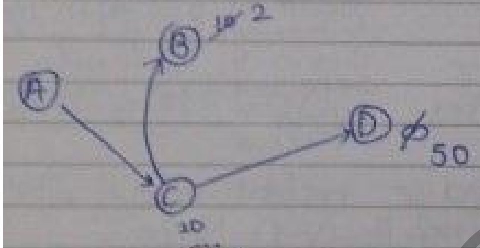


→ using Bellman Ford:-



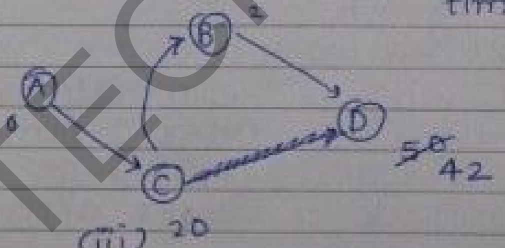
(i) Doing 1st time

~~we~~ (no -ve edge weight cycle, only -ve edge weights, we have to do it only 2 times ~~is not~~ (V-1) times.)



(ii)

doing 2nd time



(iii)

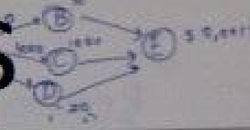
doing 3rd time

[★ after performing V-1 times, do it one more time to check, if the values change then there is -ve edge weight cycle, otherwise not.]

★★ When there is

- If given graph contain both +ve & -ve edge weight, bellman ford will give correct result always.
- If given graph contain -ve edge weight cycle, Bellman ford will inform (or report) saying that graph contain -ve edge weight cycle & we can't compute shortest path.
- The Bellman Ford will also give the vertices involved in -ve edge weight cycle. (but they must be reachable from the source.)

Dynamic Programming



Greedy :- ① Sometimes give wrong answer.
② Takes less time. ③ One decision at a time.

Dynamic : ① Always give correct answer.
② Take More time because it cover all possibilities.
③ Sequence of decisions.

- In greedy to choose shortest distance from A to E, greedy will choose path A to B (shortest path from A), & then choose B to E, which gives wrong answer.
- In dynamic programming, it will search every possibility & hence take more time.

** Greedy guess the answer which may or may not be correct.

Applications of Dynamic Programming - (Recursive)

① Fibbonacci Series
② Longest Common Subsequence
③ Multistage Graph
④ Matrix Chain Multiplication
⑤ Travelling Sales Person
⑥ 0/1 Knapsack
⑦ All pair shortest path
⑧ Sum of subset problem.

★ In divide & conquer → height of tree :- $\boxed{\log n}$
★ In Dynamic Programming → height of tree :- $\boxed{n}$

when we
will consid
all possib
combinatio

Fibonacci - Series :-

n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

Cosmos

Recurrence Reln:

$$T(n) = \begin{cases} 0, & \text{if } n=0 \\ 1, & \text{if } n=1 \\ T(n-1)+T(n-2), & \text{otherwise } (n>1) \end{cases}$$

$\boxed{T(n) :- Fib(n)}$

```
fib(n)
{ if (n==0) ~~or n==1)~~
        return 0;
  if (n==1)
        return 1;
  else
      return (fib(n-1)+fib(n-2)); // or a = fib(n-1);
                                         b = fib(n-2);
                                         c = a+b;
                                        return c;
}
```

```
fib(6)
{
   a= fib(5);
   b= fib(4);
   return(a+b);
}
```

```
fib(5)
{
   a= fib(4);
   b= fib(3);
   return(a+b)
}
```

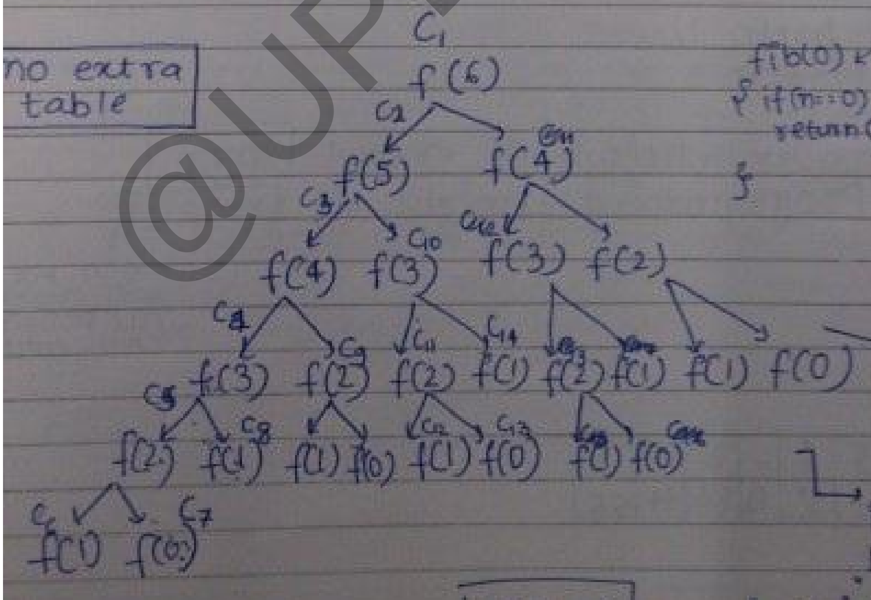
```
fib(4)
{
   a= fib(3);
   b= fib(2);
   return(a+b);
}
```

```
fib(2)
{
   a= fib(1);
   b= fib(0);
   return(a+b);
}
```

```
fib(0)
{ if (n==0)
     return 0;
}
```

```
fib(1)
{
   if (n==1)
     return 1;
}
```

no extra table



→ n level binary tree.

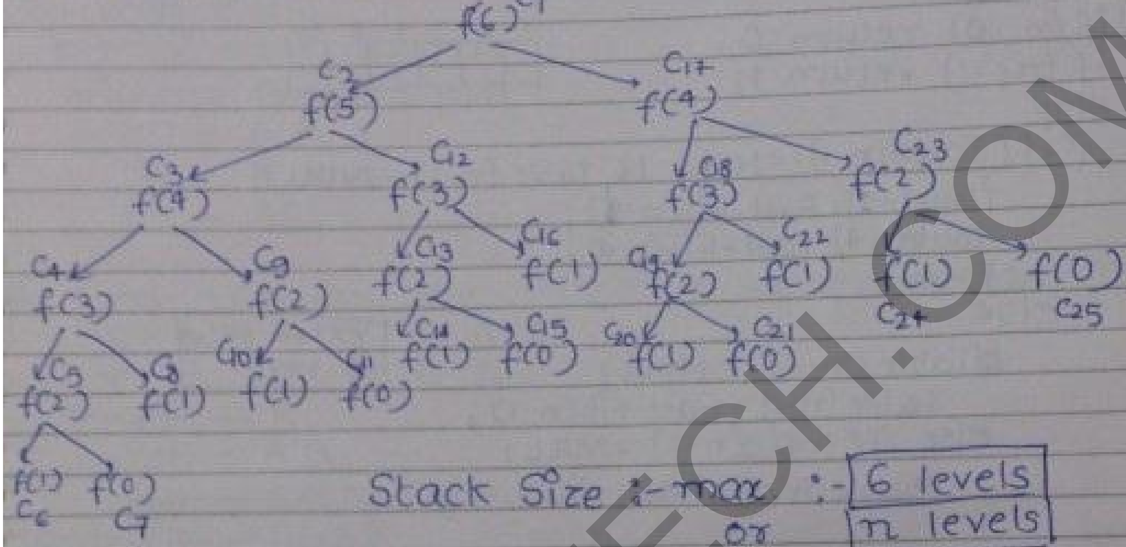
↳ if we assume that it is complete complete binary tree,

∴ no. of nodes $= 2^n - 1$

& each node there is a func. call, ∴ no. of fn. calls $= 2^n - 1$.

$\boxed{O(2^n)}$ ←

$f(6) \rightarrow f(5) \rightarrow f(4) \rightarrow f(3) \rightarrow f(2) \rightarrow f(1)$



Stack Size :- max :- $\boxed{6 \text{ levels}}$
or $\boxed{n \text{ levels}}$

& in fibonacci series fn. :- each fn. requires 6 Bytes.

∴ Space complexity = max. Stack size × 6 Bytes + 2 (i/p size)

= n × 6 Bytes + 2

= $\boxed{\Theta(n)}$

Drawback Of Recursive Soln. :-

- Note :- In the above recursive tree many subproblems are repeating (overlapping subproblems).
- In dynamic programming, we will solve every subproblem only once.
- Dynamic Programming time complexity for fibonacci series $\boxed{O(n)}$ because it will call only distinct fn calls. As soon as a particular fn. is completed, that value is stored in a table so that no need to complete the function again.

Fibonacci Series using Dynamic Programming :-

Time Complexity :- $\boxed{O(n)}$

Space Complexity :- $\underset{\text{i/p}}{2} + \underset{\text{max n level stack size}}{6n} + \underset{\text{Table size for storage of f(1), f(2), f(3), f(4), f(5), f(6}}{n} = \boxed{\Theta(n)}$

(size of 'n' in fib(n) which is of 2 Bytes) ← i/p

$f(3) = f(1) + f(2)$

Fast Fibonacci Series :-

```
Fast-Fib(n)
  {if (n==0) return 0;
   if (n==1) return 1;
   else
    {if(Table[n-1]==NULL && Table [n-2]==NULL)
        {Table[n-1]= Fast-Fib(n-1);
         Table[n-2]= Fast-Fib(n-2);
        }
     else
        {if(Table [n-1] == NULL)
            Table [n-1] = Fast-Fib(n-1);
         else if(Table[n-2]==NULL)
            Table[n-2] = Fast-Fib(n-2);
        }
    Table[n]= Table[n-1] + Table[n-2];
    return(Table[n]);
   }
  }
```

extra table is req.

Max. level of the tree:- $\boxed{n}$

LCS :-

Subsequence:- of a given sequence is just the given sequence only in which 0 or (more) symbols are left out.

e.g. $S = (\overset{1}{A}, \overset{2}{B}, \overset{3}{B}, \overset{4}{A}, \overset{5}{B}, \overset{6}{B})$ → sequence

$S_1 = (\underset{1}{A}, \underset{4}{A}, \underset{5}{B}, \underset{6}{B})$ → subsequence

$S_2 = (\overset{1}{A}, \overset{4}{A})$ → subsequence

S_3 $(\underset{2}{B}, \underset{3}{B}, \underset{5}{B}, \underset{6}{B})$ → "

$S_4 = (B, B, A, A)$ → not the Subsequence

$S_5 = (A, B, B, B, A)$ → " " "

Common Subsequence:-

z is a common subsequence of x & y if z is subsequence to both x & y.

e.g. x = A B B A B B
y = B A A B A A
CS_1 = A B A
CS_2 = B A B
CS_3 = AB

Q Find LCS for the following subsequences:-
x = A B C B D A B
y = B D C A B A

1 length → A
2 length → A B
3 length → A B A
4 length → BDAB
~~5 length~~

Q.
1 2 3 4 5 6
x: B B B A A A
y: B B A A A A

(decrementing 2nd sequence when it doesn't matches.)

LCS(6,6) = 1 + LCS(5,5)
1 + LCS(4,4)
1 + LCS(3,3)
0 + LCS(3,2)
1 + LCS(2,1)
1 + LCS(1,0)
0

Q. x: B B A A B A
y: B B B A B A

(decrementing 1st sequence when it doesn't matches.)

LCS(6,6) = 1 + LCS(5,5)
1 + LCS(4,4)
1 + LCS(3,3)
0 + LCS(2,3)
1 + LCS(1,2)
1 + LCS(0,1)

* In such case, when we have the choice of decrementing either of the sequence,
- greedy says, decrement either of them, but not both (this sometimes give correct result & sometimes incorrect.
- dynamic says do both options, this always give correct result.

LCS(i, j) = Length of ~~Con~~ longest common subsequence possible with the 2 sequences x & y, where sequence x contains 'i' symbols & y contain 'j' symbols

```
~~LCS(i,j)
 { if (i==0  || j==0)
     { print("No common subsequence");
     }
   else if (i=~~
```

$$LCS = \begin{cases} 0 ; \text{ if } i=0 \text{ or } j=0 \\ 1+LCS(i-1,j-1) \quad \text{if } x[i]=y[j] \\ \max(LCS(i,j-1), LCS(i-1,j)) \quad \text{, otherwise} \end{cases}$$

```
LCS (i,j)
 { if (i==0 || j==0)
     {printf(" 0 common sequence");
      return 0;
     }
   else if ( x[i]==y[j])
     {  a= LCS(i-1,j-1);
          return (a+1);
     }
 else
    { if(LCS(i,j-1) > LCS(j-1,j))
          return
```

If we dont use dynamic programming then many fn. calls are repeated, at max. there are (m+n) levels, so for ↑complete binary tree, the max. no. of nodes are 2^{m+n}, & lets suppose each fn call takes O(1) time. ∴ complexity [illegible]

```
    a = LCS(i-1, j);
    b = LCS(i, j-1);
  if (a>b)
    return a;
  else
    return b;
  }
}
```

★★ In the recurssive tree
If we draw recurssive tree for LCS of (m,n), in the worst case we will get (m+n) level complete binary tree. So total no. of nodes in (m+n) level complete binary tree is $2^{m+n}-1$

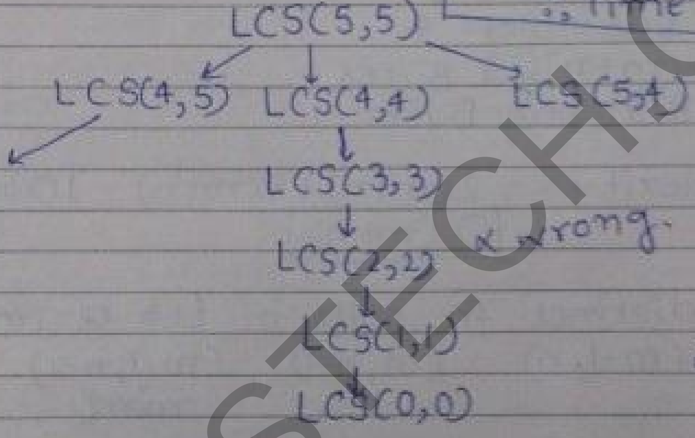
∴ $2^{m+n}-1$ no. of function calls.

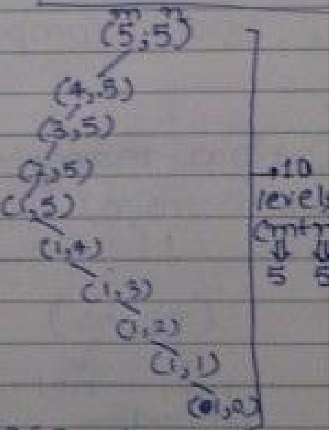
We assume each fn. call takes O(1) time.

∴ Time complexity :-

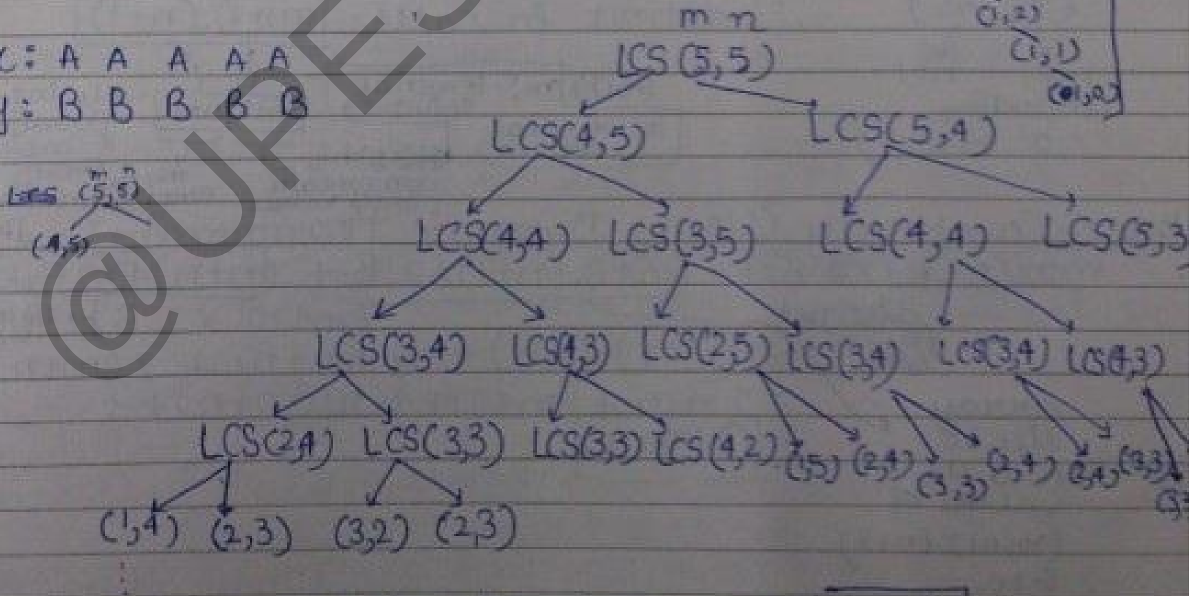
$2^{m+n} * O(1)$

$= \boxed{O(2^{m+n})}$





x : A A A A A
y : B B B B B



★ The height of the tree (at max.) = $\boxed{m+n}$

n above recurssive tree many fn. calls are repeated, so we ave to go to dynamic programming, which will solve very subproblem only once.

pace complexity:-

2 i/p arrays are req. of size m & n.

∴ i/p size = (m+n) × 2 Bytes (each integer/char of 2B).

Stack size (max.) = m+n

no. of local variables = 4

∴ size = 4(m+n) × 2 Bytes

= 8(m+n) Bytes

. Space Complexity = 2(m+n) + 8(m+n) = 10(m+n)

$= \boxed{O(m+n)}$

How many distinct fn. calls in LCS of (m,n).

(m,n) (m-1,n) (m,n-1) (m-1,n-1)

1

(m, n)

↓ m+1 ↓ n+1

(including 0 as well)

∴ distinct fn. calls = $\boxed{(m+1).(n+1)}$

m:- length of sequence 1

n:- length of sequence 2.

$\boxed{m+1}\boxed{n+1}$ → using Permutations & Combinations theory

m+1: this can contain (m+1) values

n+1: this can contain (n+1) value

ace Complexity (Using Dynamic Programming):-

$\max(m,n) \times 4\text{ B} + \underbrace{(m+1).(n+1) \times 2\text{ B}}_{\text{distinct fn calls}} + \underbrace{(m+n)4 \times 2\text{ B}}_{\text{distinct fn calls}} + (m+n) \times 2\text{ B}$ → i/p array size

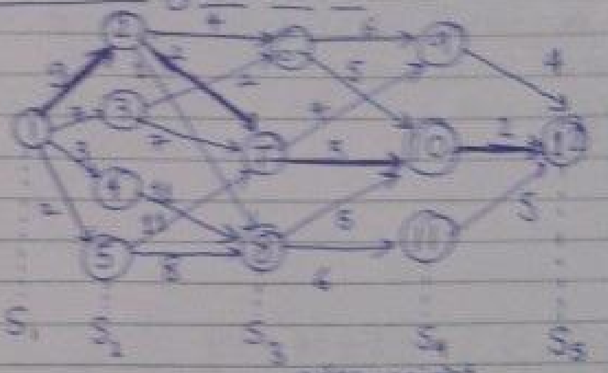
distinct fn calls → array size to hold the values of (m+1).(n+1) size

$= 2(m+1).(n+1) + 8(m+n) + 2(m+n)$

$= 2mn + 2m + 2n + 10(m+n) + 2$

$= \boxed{O(mn)}$

Multistage Graph



using Greedy :-
① - ⑤ - ⑧ - ⑩ - ⑫ ;- 17

MSG(1,1) — from 1st stage; from 1st vertex

MSG(3,8) — from 3rd stage; from 8th vertex

to final destn. which is vertex 12

→ edge weight

MSG(1,1) = { c(1,2) + MSG(2,2) → 9+7 = ⑯ ; c(1,3) + MSG(2,3) → 7+9 = ⑯ ; c(1,4) + MSG(2,4) → 3+18 = 21 ; c(1,5) + MSG(2,5) → 2+15 = 17 } → min of the 4

MSG(2,2) = { c(2,6) + MSG(3,6) → 4+7 = 11 ; c(2,7) + MSG(3,7) → 2+5 = 7 ✓ ; c(2,8) + MSG(3,8) → 1+7 = 8 } → min of the 3

MSG(2,3) = { c(3,6) + MSG(3,6) → 2+7 = 9 ✓ ; c(3,7) + MSG(3,7) → 7+5 = 12 } → min of 2

MSG(2,4) = { c(4,8) + MSG(3,8) → 11 + 7 = 18

MSG(2,5) = { c(5,7) + MSG(3,7) → 11+5 = 16 ; c(5,8) + MSG(3,8) → 8+7 = 15 ✓ } → min. of 2

MSG(3,6) = { c(6,9) + MSG(4,9) → 6+4 = 16 ; c(6,10) + MSG(4,10) → 5+2 = 7 ✓ } → min. 2

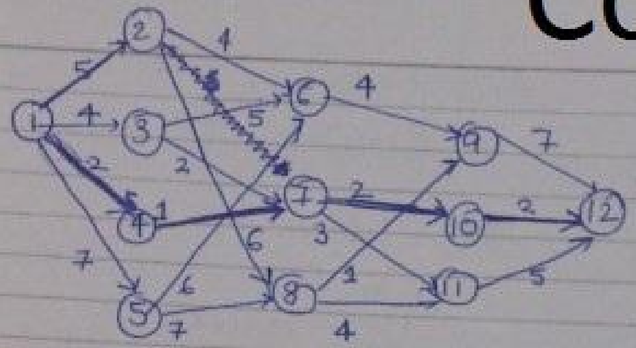
MSG(4,9) = c(9,12) = 4

MSG(4,10) = c(10,12) = 2

MSG(3,7) = { min (c(7,9) + MSG(4,9) → 4 + 4 = 8 ; c(7,10) + MSG(4,10) → 3 + 2 = 5 ✓)

MSG(3,8) = { min (c(8,10) + MSG(4,10) → 5+2 = 7 ✓ ; c(8,11) + MSG(4,11) → 6+5 = 11)

MSG(4,11) = c(11,12) = 5



$$\text{MSG}(1,1) = \left\{ \min \begin{pmatrix} C(1,2)+\text{MSG}(2,2) \\ C(1,3)+\text{MSG}(2,3) \\ C(1,4)+\text{MSG}(2,4) \\ C(1,5)+\text{MSG}(2,5) \end{pmatrix} \right. \begin{matrix} \rightarrow 5+14=19 \\ \rightarrow 4+6=10 \\ \rightarrow 2+5=7 \checkmark \\ \rightarrow 7+15=22 \end{matrix}$$

$$\text{MSG}(2,2) = \left\{ \min \begin{pmatrix} C(2,6)+\text{MSG}(3,6) \\ C(2,8)+\text{MSG}(3,8) \end{pmatrix} \right. \begin{matrix} \rightarrow 4+11=15 \\ \rightarrow 6+8=14 \checkmark \end{matrix}$$

$$\text{MSG}(3,6) = \{ C(6,9)+\text{MSG}(4,9) \rightarrow 4+7=11$$

$$\text{MSG}(4,9) = C(9,12) = 7$$

$$\text{MSG}(3,8) = \left\{ \min \begin{pmatrix} C(8,9)+\text{MSG}(4,9) \\ C(8,11)+\text{MSG}(4,11) \end{pmatrix} \right. \begin{matrix} \rightarrow 1+7=8 \checkmark \\ \rightarrow 4+5=9 \end{matrix}$$

$$\text{MSG}(4,11) = C(11,12) = 5$$

$$\text{MSG}(2,3) = \left\{ \min \begin{pmatrix} C(3,6)+\text{MSG}(3,6) \\ C(3,7)+\text{MSG}(3,7) \end{pmatrix} \right. \begin{matrix} \rightarrow 5+11=16 \\ \rightarrow 2+4=6 \checkmark \end{matrix}$$

$$\text{MSG}(3,7) = \left\{ \min \begin{pmatrix} C(7,10)+\text{MSG}(4,10) \\ C(7,11)+\text{MSG}(4,11) \end{pmatrix} \right. \begin{matrix} \rightarrow 2+2=4 \checkmark \\ \rightarrow 3+5=8 \end{matrix}$$

$$\text{MSG}(4,10) = C(10,12) = 2$$

$$\text{MSG}(2,4) = C(4,7)+\text{MSG}(3,7) = 1+4=5$$

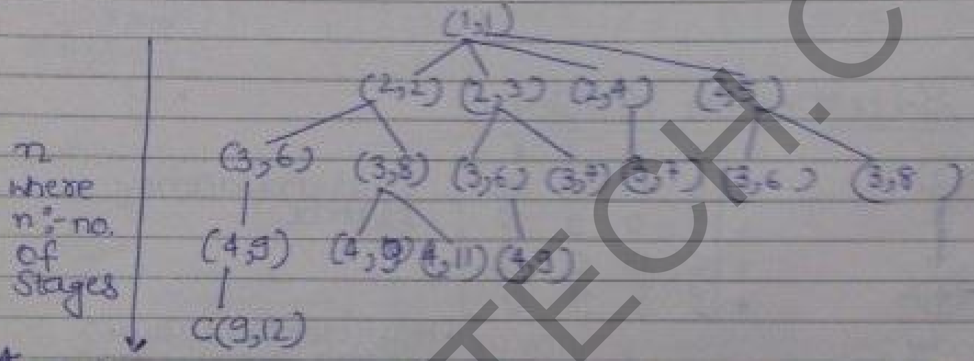
$$\text{MSG}(2,5) = \left\{ \min \begin{pmatrix} C(5,6)+\text{MSG}(3,6) \\ C(5,8)+\text{MSG}(3,8) \end{pmatrix} \right. \begin{matrix} \rightarrow 6+11=17 \\ \rightarrow 7+8=15 \checkmark \end{matrix}$$

nd Method:- from ⑥ :- 11 (6-9-12) ✓
from ⑦ :- 4 (7-10-12) ✓
8 (7-11-12)
from ⑧ :- 8 (8-9-12) ✓
9 (8-11-12)

om ②:- 15 (2-6-min. distance from ⑥)
6+8=14 (2-8-min. distance from⑧) ✓

★ Dynamic programming & brute force becomes same when all problems are distinct.

from ③ :- 5+11=16 (3-6-min dis from ⑥)
2+4 = 6 (3-7-min " " ⑦) ✓
from ④ :- 6+4=10 (4-7- " " " ⑦)



★ Multistage graph will generate in the worst case 'n' level complete binary tree, n:- no of stages so total time complexity is $\boxed{O(2^n)}$ → [no. of nodes in comple binary tree = $2^n - 1$]
But in multistage graph problem many subproblems are repeated so we perform dynamic programming

★ MSG (G(V,E), N) ⟶ V distinct function call

★ All function calls combined take O(E) times.
∴ $\boxed{O(V+E)}$

Space Complexity :- $\underset{\text{i/p in form of adjacency list}}{\underset{\downarrow}{(V+E)}} + \underset{\text{stack-size (no. of stages)}}{\underset{\downarrow}{n}} + \underset{\text{table size (distinct fn. calls)}}{\underset{\downarrow}{V}}$

$= V+E + \underset{\text{max stack size}}{V} + V = 3V+E$
$= \boxed{O(V+E}$

MSG(S_i, V_j) = the min. cost req. from stage S_i & vertex v_j to destination.

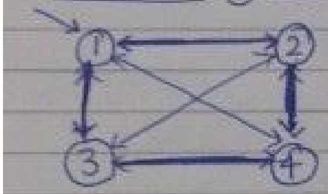
$$MSG(S_i, V_j) = \begin{cases} \left(C(V_j, k) + MSG(S_{i+1}, k)\right) \rightarrow \min. \\ \forall k \in S_{i+1} \;\&\; \underset{\substack{\downarrow \\ \text{edge should be there.}}}{(v_j, k) \in E} \\ C(V_j, D) \quad \text{if } S_i == (F-1) \end{cases}$$

D → Dest^n
F → Final stage

$c(V_j, k)$
~~$v_j \in F$~~
$c(V_j, k) \in E$
$k \in F$

Date
22.07.12

Travelling Salesperson Problem:-



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 9 & 10 \\ 3 & 6 & 13 & 0 & 12 \\ 4 & 8 & 8 & 9 & 0 \end{array}$$

Cost-Adjacency Matrix

Acc. to greedy:-
①-②-③-④-①
㊴

- Hamitoniaun Tour:- Visiting every vertex exactly once.
- Euler Tour:- Visiting every edge exactly once.

Using Dynamic Programming:-

$$\underset{\substack{\downarrow \\ \text{starting from ①}}}{TSP\{1,} \underset{\substack{\downarrow \\ \text{can go to either 2 or 3 or 4}}}{\{2,3,4\}\}} = \min. \begin{cases} C(1,2) + TSP\{2,\{3,4\}\} & \rightarrow 10+25=35 ✓ \\ C(1,3) + TSP\{3,\{2,4\}\} & \rightarrow 15+25=40 \\ C(1,4) + TSP\{4,\{2,3\}\} & \rightarrow 20+23=43 \end{cases}$$

$$TSP\{2,\{3,4\}\} = \min. \begin{cases} C(2,3) + TSP\{3,\{4\}\} & \rightarrow 9+20=29 \\ C(2,4) + TSP\{4,\{3\}\} & \rightarrow 10+15=25 ✓ \end{cases}$$

$$TSP\{3,\{4\}\} = C(3,4) + \underset{\substack{\downarrow \\ \text{go back to 1 from 4}}}{\underline{C(4,\overset{1}{\cancel{4}})}} = 12+8=20$$

$$TSP\{4,\{3\}\} = C(4,3) + C(3,1) = 9+6=15$$

$TSP\{3,\{2,4\}\} = \min. \begin{cases} C(3,2) + TSP\{2,\{4\}\} \rightarrow 13+18=31 \\ C(3,4) + TSP\{4,\{2\}\} \rightarrow 12+13=25 ✓ \end{cases}$

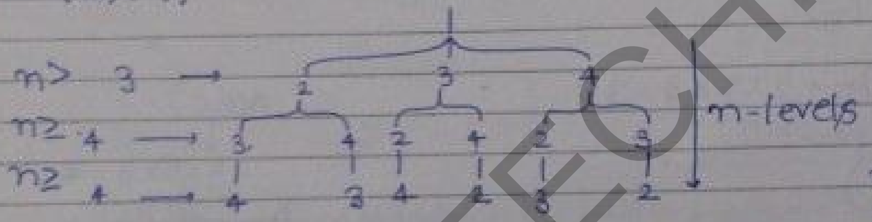
$TSP\{2,\{4\}\} = C(2,4) + C(4,1) = 10 + 8 = 18$

$TSP\{4,\{2\}\} = C(4,2) + C(2,1) = 8 + 5 = 13$

$TSP\{4,\{2,3\}\} = \min. \begin{cases} C(4,2) + TSP\{2,\{3\}\} \rightarrow 8+15=23 \\ C(4,3) + TSP\{3,\{2\}\} \rightarrow 9+18=27 \end{cases}$

$TSP\{2,\{3\}\} = C(2,3) + C(3,1) = 9 + 6 = 15$

$TSP\{3,\{2\}\} = C(3,2) + C(2,1) = 13 + 5 = 18$



at each level we have (at max.) n nod
& there are n levels
n+n+.....+n(n ti

- Time complexity of Travelling Salesperson w/o dynamic programming :- $\Omega(2^n)$ [Brute force method]
- Using Dynamic Programming :- $\boxed{\Omega(2^n)}$ [No overlapping subproblem.]

~~number of nodes in the above tree~~

$n \times (n-1) \times (n-2) \times \ldots \times 1 \leq n \times n \times \ldots \times n$ (ntimes)

$\leq n^n$

$= \boxed{O(n^n)}$

★ Travelling Salesperson is one of the NP complete proble because they can't be solved in polynomial time.

Space Complexity $= \underbrace{V^2}_{\text{adjacency matrix}} + \underbrace{V+E}_{\text{graph}} + \underbrace{V}_{\text{stack size}} + \underbrace{2^V}_{\text{distinct function calls}}$

$= \boxed{O(2^V)} \quad \boxed{\Omega(2^V)}$

TSP(A,R) :- min. cost required to go from vertex A to all other remaining vertices (R) exactly once & coming back to source S

Recurrence Reln. :-

$$TSP(A,R) = \begin{cases} \min\left(c(A,k) + TSP(k,R')\right) & \text{where } \forall k \in R \ \& \ R' = R - \{k\} \\ c(A,S) & \text{if } R = \phi. \end{cases}$$

Source is S.

Matrix Chain Multiplication

e.g. $A_{2\times10}$ $B_{10\times5}$

$C = AB$ ∴ Total no. of multiplications

2×5 $= 2\times5\times10 = 100$

c(k,S) +
c(A,k)

Q. $A_{2\times3}$ $B_{3\times5}$ $C_{5\times3}$

$(AB)C \leftarrow (AB) \rightarrow 2\times5\times3 = 30$

$(AB)_{2\times5}$ $C_{5\times3}$

$2\times3\times5 = 30$

$(AB)C \rightarrow \boxed{60}$ multiplications

A(BC)

$\downarrow$

$3\times5\times3 = 45$

$A_{2\times3}$ $(BC)_{3\times3}$

$2\times3\times3 = 18$

$A(BC) \rightarrow \boxed{63}$ multiplications

No. of multiplications of m×p & p×n matrices = $\boxed{m\times n\times p}$

ABCDE

(A)(BCDE) → this will have 5 because BCDE can be multiplied in 5 ways

(AB)(CDE) → 2

(ABC)(DE) → 2

(ABCD)(E) → 5

14

$A_{1\times5}$ $B_{5\times2}$ $C_{2\times1}$

$(AB)_{1\times2} C_{2\times1} \rightarrow 1\times1\times2 = 2$

$1\times2\times5 = 10$

$10 + 2 = \boxed{12}$ ✓

$A(BC)_{5\times1}$

$\downarrow$

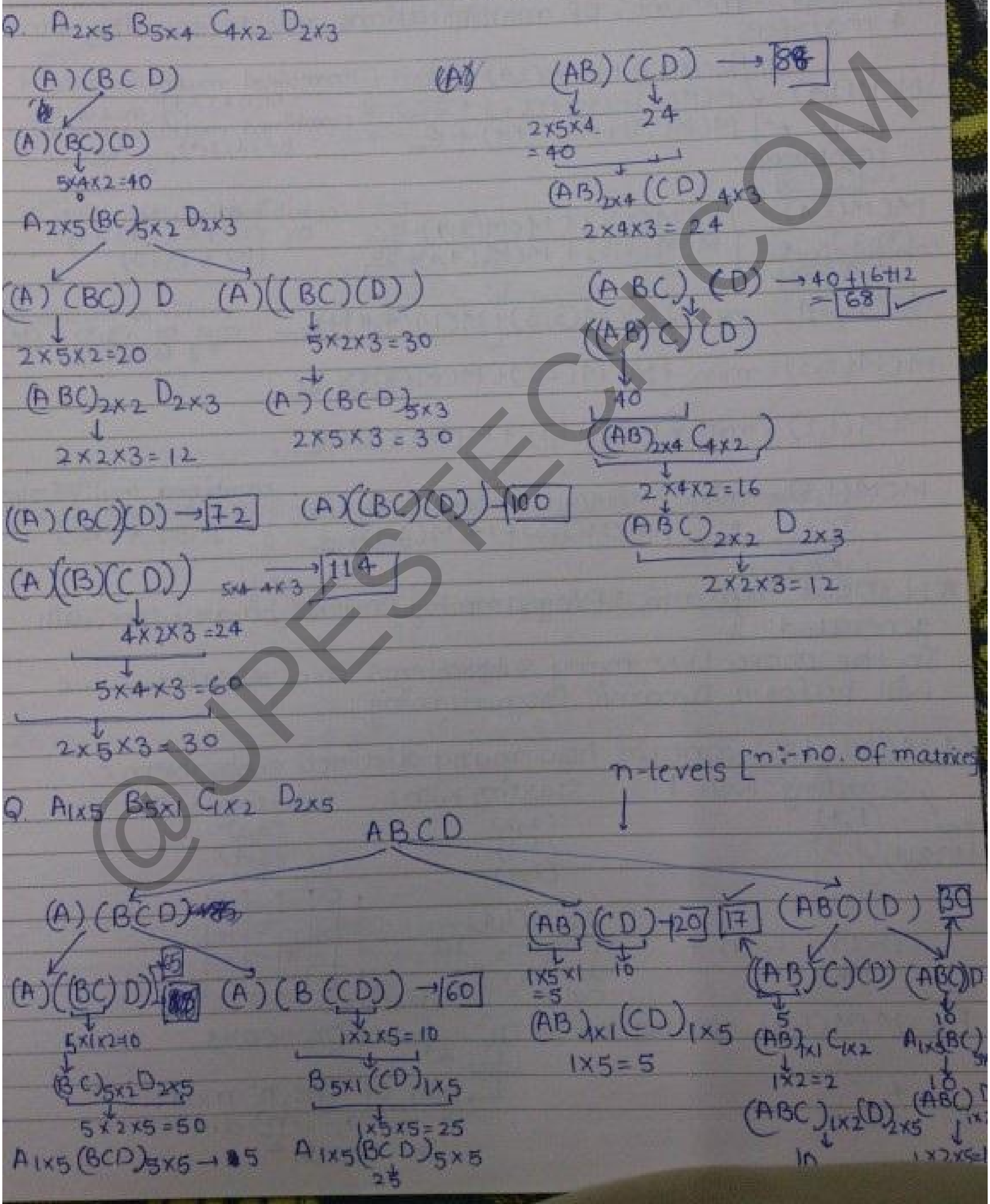
$5\times2\times1 = 10$

$A_{1\times5}$ $(BC)_{5\times1}$

$1\times1\times5 = 5$

$10 + 5 = \boxed{15}$

Cosmos



Q. $A_{2\times5}\ B_{5\times4}\ C_{4\times2}\ D_{2\times3}$

(A)(BCD)

(A)(BC)(D)

$5\times4\times2=40$

$A_{2\times5}(BC)_{5\times2}D_{2\times3}$

((A)(BC)) D

$2\times5\times2=20$

$(ABC)_{2\times2}D_{2\times3}$

$2\times2\times3=12$

(A)((BC)(D))

$5\times2\times3=30$

$(A)(BCD)_{5\times3}$

$2\times5\times3=30$

((A)(BC))(D) → $\boxed{72}$

(A)((BC)(D)) → $\boxed{100}$

(A)((B)(CD)) 5x4 4x3 → $\boxed{114}$

$4\times2\times3=24$

$5\times4\times3=60$

$2\times5\times3=30$

~~(A)~~

(AB)(CD) → $\boxed{88}$

$2\times5\times4=40$ 24

$(AB)_{2\times4}(CD)_{4\times3}$

$2\times4\times3=24$

(ABC)(D) → 40+16+12 = $\boxed{68}$ ✓

((AB)C)(D)

40

$((AB)_{2\times4}C_{4\times2})$

$2\times4\times2=16$

$(ABC)_{2\times2}\ D_{2\times3}$

$2\times2\times3=12$

Q. $A_{1\times5}\ B_{5\times1}\ C_{1\times2}\ D_{2\times5}$

n-levels [n:- no. of matrices

ABCD

(A)(BCD)

(A)((BC)D) $\boxed{35}$

$5\times1\times2=10$

$(BC)_{5\times2}D_{2\times5}$

$5\times2\times5=50$

$A_{1\times5}(BCD)_{5\times5}$ → 5

(A)(B(CD)) → $\boxed{60}$

$1\times2\times5=10$

$B_{5\times1}(CD)_{1\times5}$

$1\times5\times5=25$

$A_{1\times5}(BCD)_{5\times5}$

25

(AB)(CD) → $\boxed{20}$ $\boxed{17}$ ✓

$1\times5\times1=5$ 10

$(AB)_{1\times1}(CD)_{1\times5}$

$1\times5=5$

(ABC)(D) $\boxed{39}$

((AB)C)(D)

5

$(AB)_{1\times1}C_{1\times2}$

$1\times2=2$

$(ABC)_{1\times2}(D)_{2\times5}$

10

(A(BC))D

10

$A_{1\times5}(BC)$

10

(ABC)

each comparison takes $O(1)$ time, as there are $(n-1)$ comparisons, $\therefore$ $\boxed{O(n) \text{ time}}$

- $MCM(1,4)$:- min. no. of multiplications req. to multiply 1 to 4 matrices.

$$MCM(1,4) = \min \begin{cases} MCM(1,1) + MCM(2,4) + \underline{25} \\ MCM(1,2) + MCM(3,4) + \underline{5} \\ MCM(1,3) + MCM(4,4) + \underline{10} \end{cases}$$

combined multiplication cost of (1,1) & (2,4) e.g. (A) & (BCD)

combined multiplication of (1,2) & (3,4) e.g. (AB) & (CD)

$(n-1)$ comparisons $\therefore \boxed{O(n)}$

$$MCM(2,4) = \min \begin{cases} MCM(2,2) + MCM(3,4) + \underline{25} \\ MCM(2,3) + MCM(4,4) + \underline{50} \end{cases}$$

combined multiplication of of (2,2) & (3,4) e.g. (B) & (C,D)

$(n-2)$ comparisons $\therefore \boxed{O(n)}$

$$MCM(3,4) = \min \{ MCM(3,3) + MCM(4,4) + \underline{0}$$

combined multiplication cost of (3,3) & (4,4) e.g. (C) & (D)

$$MCM(2,3) = \min. \{ MCM(2,2) + MCM(3,3) + \underline{10}$$

$$MCM(1,2) = \min. \{ MCM(1,1) + MCM(2,2) + \underline{5}$$

$$MCM(1,3) = \min. \begin{cases} MCM(1,1) + MCM(2,3) + \underline{10} \\ MCM(1,2) + MCM(3,3) + \underline{2} \end{cases}$$

combined multiplication cost of (1,1) & (2,3) e.g. (A) & (BC)

$(n-2)$ comparisons $\therefore \boxed{O(n)}$

★ Without Dynamic Programming n-level binary tree will generate 2^n

In the above tree, many subproblems are repeated, so we will perform Dynamic Programming.

Q. MCM (1,n) contain how many distinct subproblem?

Starting with 1
(1,4)
(1,3)
(1,2)
(1,1)

or MCM(1,4)

Starting with 2
(2,2)
(2,3)
(2,4)

Starting with 3
(3,3)
(3,4)

Starting with 4
(4,4)

$\therefore$ MCM(1,4) contains 10 distinct fn. calls.

	1	2	3	4
1	✓	✓	✓	✓
2	×	✓	✓	✓
3	×	×	✓	✓
4	×	×	×	✓

For MCM(1,n) contains $\frac{n^2}{2}$ distinct function calls.

$\llcorner$ half of the n^2 matrix is filled, $\therefore \frac{n^2}{2}$ distinct fn. calls.

Time Complexity $= \frac{n^2}{2} \times$ time complexity of each fn. call

$= \frac{n^2}{2} \times O(n)$

$= \boxed{O(n^3)}$

Space Complexity :- $\underbrace{4}_{\text{to store size of the matrix (4Bytes for each matrix)}} \underbrace{n}_{\text{no. of matrices}} + \underbrace{n}_{\text{stack size}} + \underbrace{\frac{n^2}{2}}_{\text{table-size (no. of distinct fn. calls)}}$

$= O(n^2)$

Recurrence Reln. :-

MCM(i, j) = { min. no. of multiplications req. to multiply i to j matrices.

$$MCM(i,j) = \begin{cases} \min(MCM(i,k) + MCM(k+1,j)) \ \forall\ k \text{ from } i \text{ to} (j-1 \\ 0 \quad \text{if } i == j. \end{cases}$$

my answer

$$\text{Correct answer} = \begin{cases} \min\limits_{i \le k < j} \left[MCM(i,k) + MCM(k+1,j) + \text{no. of multiplic tions req. to mul ply the matrices } (i, \ (k+1,j) \right] \\ 0 \ , \text{if } i == j \end{cases}$$

0/1 Knapsack Problem

objects	ob_1	ob_2	Ob_3	$n=3$
profits	5	3	4	$M=5$
weights	3	2	1	

Manually:-

case 1 :- ob$_1$ & ob$_3$ $\therefore$ profit = 9
case 2 :- ob$_1$ & ob$_2$ $\therefore$ profit = 8
case 3 :- ob$_2$ & ob$_3$ $\therefore$ profit = 7

Using Greedy:-

ob$_1$ → $\frac{5}{3}$ = 1.66
ob$_2$ → $\frac{3}{2}$ = 1.5
ob$_3$ → $\frac{4}{1}$ = 4

take Ob$_3$ first, capacity left = 4
take Ob$_1$ 2nd capacity left = 1.
but we can't take fraction, $\therefore$ we can't take fraction of ob$_2$

Q n = 3   M = 15

Objects	ob$_1$	ob$_2$	ob$_3$	ob$_4$	ob$_5$
profits	50	29	33		
weight	10	6	7		
profit:weight	5	4.83	4.7		

Using Greedy:-

take ob$_1$ 1st, $\therefore$ capacity left = 5
we can't take any more.
$\therefore$ profit = $\boxed{50}$

Manually :-

ob$_2$ & ob$_3$ → profit = $\boxed{62}$
ob$_1$ → " = 50

* Note :- Greedy algo will not give optimal soln. for 0/1 Knapsack, so we use dynamic programming which will cover all possible combination.

n = 5   M = 10

ojects	ob$_1$	ob$_2$	ob$_3$	ob$_4$	ob$_5$
ofits	7	2	1	6	12
eights	3	1	2	4	6

e.g. 01KP(5,10)
5 objects to choose from (from 1 to 5)
capacity of knapsack

01KP(4,10)
4 objects to choose from (1 to 4) excluded 5.

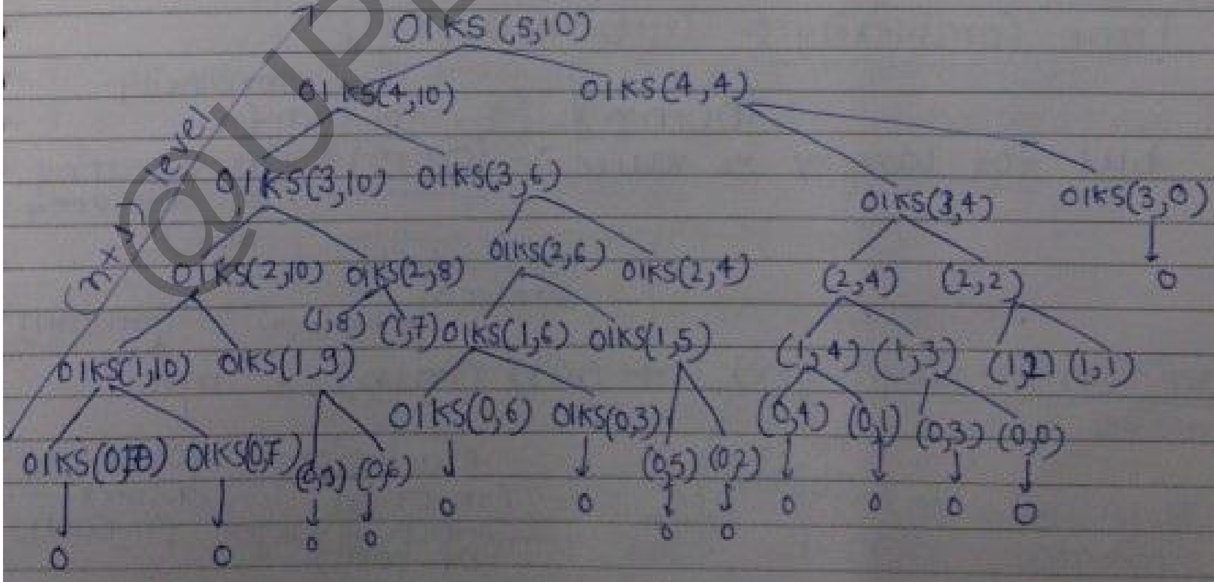
1KP(n,m) = max profit we will get in 0/1 Knapsack proble
here n :- no. of objects & m is the capacity of knapsack.

no. of objects → among which we can select objects to be put into knapsack

m → current capacity.

$$01KS(n,m) = \begin{cases} 0 & \text{if } (n) \text{ or } (m) == 0 \\ ~~\max \begin{cases} 01KP(n-i, m-w_i) + p_i \ \forall i \\ 01KP(n-i, m) \end{cases}~~ & \\ 01KS(n-1, m) & \text{if } W_n > M \\ \max. \begin{cases} 01KS(n-1, m-W_n) + P_n \\ 01KS(n-1, m) \end{cases} & \text{if } W_n \le M \end{cases}$$

```
01KS (n,m){
    if (n==0 or m==0)
      { return 0;
      }
    else if (Wn>M)
      { 01KS(n-1,m);
      }
    else
      {a= 01KS(n-1,m-Wn)+Pn ;
       b = 01KS(n-1,m);
       return (max (a,b));
      }
}
```



0/1 knapsack of (n,m) will generate n-level (approx.) almost complete binary trees.
∴ $\boxed{2^n}$ function calls. & every fn. call takes O(1) time [as we have to make only 1 comparison in max. fn.]
∴ W/O using Dynamic :- $\boxed{O(2^n)}$

★In above rec. tree some fn. calls are repeating, so we perform dynamic programming, which calls distinct fn. calls exactly once.

Q. 01 knapsack contains how many distinct fn. calls

01 KS (5, 10)

it can decrease in 6 ways (5,4,3,2,1,0)

the weight can decrease acc. to the weights of the object, but maximum it can have values from 10 to 0, ∴ 11

∴ 6 × 11 = 66 distinct functions (max.)

∴ 01KS(n, m) ∴ distinct fn. calls = (n+1)(m+1)

(n+1) (m+1)

∴ Time Complexity :- (n+1)(m+1) × O(1)

time complexity of each fn

:- $\boxed{O(mn)}$

but for larger n value :- $\boxed{O(2^n)}$ (as repeating fn. are very less.)

NP Complete problem.

Space Complexity :- n × k + [illegible] (m+n) + mn

max. (m,n)

(m+1)(n+1)

no. of objects

size of each object

Stack size (m or n which is larger of both).

table size (distinct fn. calls)

Sum Of Subsets Problem :-

$S = (70_1, 20_2, 90_3, 50_4, 25_5, 15_6, 40_7, 30_8)$

$\boxed{M = 200}$

Q. To find a subset whose sum is 200.

$S_1 = (90, 90, 50, 40)$

$S_2 = (70, 90, 40)$

$S_3 = (70, 90, 25, 15)$

$S_4 = (70, 20, 25, 15, 40, 30)$

SOS(8, 200)

8 → Set contains 8 elements

200 → the subset sum is 200 (or we can say that the remaining value to make it 200).

e.g.
- SOS(~~8~~0, 200) means set is over but ~~nor~~ no element is choosen.
- SOS(5, 0) means 5 elements are left but we get the subset whose sum is 200.
- SOS(4, 100) means 4 elements are left & ~~100~~ we need 100 more to make the subset sum to be 200.

$$SOS(n,m) = \begin{cases} SOS(n-1, m) \text{ if } m > w_n \\ \text{if } m < w_n \text{ but } m \neq 0 \\ \begin{pmatrix} SOS(n-1, m-w_n), S = S \cup w_n \\ SOS(n-1, m) \end{pmatrix} \\ S \text{ (return subset) if } m = 0. \end{cases}$$

- initial value of $\boxed{S = \phi}$

- initial value of n :- no. of elements in the subset.
- " " " m :- M (e.g. M = 200 in above example.)

$$SOS(n,m) : \begin{cases} -1 \text{ (error message) if } n == 0 \;\&\&\; m \neq 0 \\ S \text{ (return subset) if } m == 0 \\ SOS(n-1, m) \text{ if } \cancel{m > w_n}\ w_n > m \\ \begin{pmatrix} SOS(n-1, m-w_n), S = S \cup w_n \\ SOS(n-1, m) \end{pmatrix} \text{ if } \cancel{m < m \leq} \text{ ; if } w_n \leq \end{cases}$$

SOS(n, m) :- Sum of Subsets of (n, m) is finding a subset from given set of 'n' elements whose sum is 'm'.

★ The rec rel^n makes n-level binary tree which have 2^n nodes.

∴ Time Complexity :- $2^n \times O(1)$
↳ time complexity of each fn.

:- $O(2^n)$.

★ Using Dynamic Programming:-

SOS (n, m)

↓ this can decrease in (n+1) ways
e.g. n = 8
∴ it can be (8, 7, 6, ..., 1, 0)

→ this can decrease into (m+1) ways [maximum]
[e.g. in above example m = 200, it can be (200, 199, 198, ..., 2, 1, 0)
but many of these values are not possible, but m can take max (m+1) values.

∴ Time Complexity :- $\boxed{O(mn)}$
When n is large :- $\boxed{O(2^n)}$

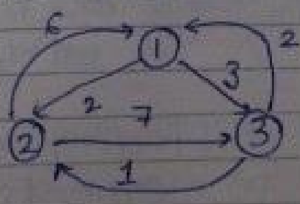
0/1 Knapsack problem can be polynomially reducidle to Sum of Subsets problem, so both are NP Complete

Space Complexity ∴- i/p size, stack size (n-level binary tree), distinct fn. calls.

$$\underset{\text{i/p size}}{(n+1)} + \underset{\text{stack size}}{n} + \underset{\text{distinct fn. calls}}{(m+1)(n+1)}$$

$$= \boxed{O(mn)}$$

ll pairs Shortest Path



① +ve edge weights
|V| * Dijkstra algo ⇒ V.(V+E)logV

② -ve edge weight & -ve edge weight cycle :-
|V| * Belman Ford
|V| * VE
= V(VE)

$A^k(i,j)$ = min. cost required to go from vertex i to vertex j with the intermediate vertices $k \in \{0,1,2,3,\ldots k\}$

A^0	1	2	3
1	0	2	3
2	6	0	7
3	2	1	0

A^1	1	2	3
1	0	2	3
2	6	0	7
3	2	1	0

A^2	1	2	3
1	0	2	3
2	6	0	7
3	2	1	0

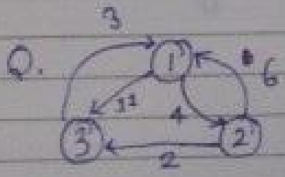
A^3	1	2	3
1	0	2	3
2	6	0	7
3	2	1	0

$$A^1(2,3) = \begin{cases} A^0(2,3) \\ \underbrace{A^0(2,1) + A^0(1,3)}_{\text{via } 1} \end{cases}$$

this means path from 2 to 3 via 1

$A^2(2,3)$ → this means path from 2 to 3 via 1 & 2.

Q.



A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

- $A^2(2,3) = \min\{A^1(2,3), A^1(2,2) + A^1(2,3)\}$

  via 0,1,2 from A^2.
  from $A^2(2,3)$

- $A^3(1,2) = \min.\{A^2(1,2), A^2(1,3) + A^2(3,2)\}$

  via 0,1,2,3 from A^3.
  from $A^2(1,2)$
  [via 0 means direct pat

- $A^0(i,j)$ → V^2 function calls
- $A^1(i,j)$ → V^2 function calls
- $A^2(i,j)$ → " " "
- $A^3(i,j)$ → " " "

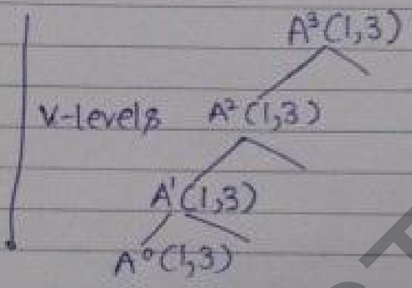
all these are distinct function calls.

$\therefore \underbrace{V^2}_{\text{distinct fn calls (each fn. call of O(1) time)}} \times \underbrace{V}_{\text{V no. of times}}$

```
Allpairsshortestpath(A°,n)
 { for (k=1 to n)
     { for(i=1 to n)
         { for (j=1 to n)
             {
              A^k(i,j)= min.{A^{k-1}(i,j),A^{k-1}(i,k)+A^{k-1}(k,j)};
             }
         }
     }
 }
```

Time Complexity:-

$\boxed{O(V^3)}$



Space Complexity:-

$V^2 + V + V^2 = O(V^2)$

i/p matrix A^0

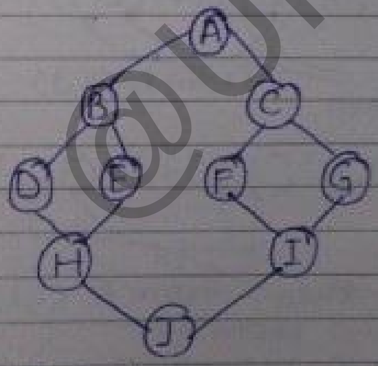
Stack size (V-levels)

table size (next table like $A^0 \& A^1$, $A^1 < A^2$, $A^2 < A^3$)

Graph Traversal

① Breadth First Traversal (BFT)

② Depth First Traversal (DFT)



BFT

A-B-D-H

A-B-C-D-E-F-G-H-I-J

DFT

A-B-D-H-J-E-C-F-I-J-G

```
BFT (v)
{ Visited(v)=1;
  add(v,Q);
  while(Q is not empty)
```

— while loop is executing 'v' times

```
{x=delete(Q) ; printf(x);
 for all w adj x        ⟶ at max. each vertex is
                           adjacent to each other
                           vertex.
   {if (w not visited)
      {visited(w)=1;
       add (w,Q);
      }
   }
 }
}
```

Note1:- In order to implement BFT, we are using queue as the data structure.

Time Complexity :- $\underbrace{V}_{\text{'V' times while loop}}.(\underbrace{1}_{\text{each vertex is connected to only one vertex}}+\underbrace{V}_{\text{each vertex is connected to each other vertex}})$

$= V + V^2$

$= V+E \longrightarrow \boxed{O(V+E)}$ or $\boxed{\Theta(V+E)}$

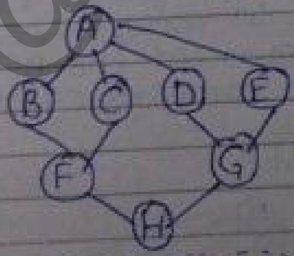
whichever is greater.

- Best case :- $\boxed{\Omega(V)}$
- Worst case :- $\boxed{O(E)}$

★ BFT is also known as Level Order Traversal.

Q. Consider following graph :-

Give 4 diff. BFT's



① Ⓐ-Ⓑ-Ⓒ-Ⓓ-Ⓔ-Ⓕ-Ⓖ-Ⓗ

② Ⓐ-Ⓔ-Ⓑ-Ⓒ-Ⓓ-Ⓖ-Ⓕ-Ⓗ

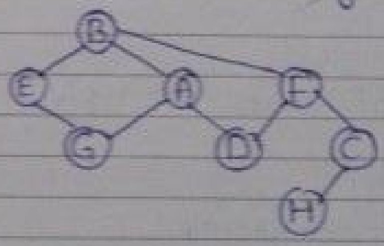
③ Ⓐ-Ⓒ-Ⓑ-Ⓓ-Ⓔ-Ⓕ-Ⓖ-

④ Ⓐ-Ⓓ-Ⓔ-Ⓑ-Ⓒ-Ⓖ-Ⓕ-

Starting from C:-
Ⓒ-Ⓐ-Ⓕ-Ⓑ-Ⓓ-Ⓔ-Ⓗ-Ⓖ

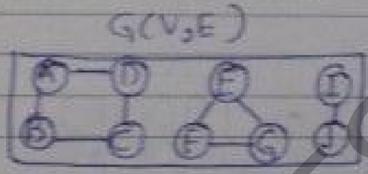
Starting from H:-
Ⓗ-Ⓕ-Ⓖ-Ⓑ-Ⓒ-Ⓓ-Ⓔ-Ⓐ

Q Consider the following Graph:-



Give 4 diff. BFT's starting from H.
① Ⓗ-Ⓒ-Ⓕ-Ⓓ-Ⓑ-Ⓐ-Ⓔ-Ⓖ
② Ⓗ-Ⓒ-Ⓕ-Ⓑ-Ⓓ-Ⓐ-Ⓔ-Ⓖ
③ Ⓗ-Ⓒ-Ⓕ-Ⓑ-Ⓓ-Ⓔ-Ⓐ-Ⓖ
④ Ⓗ only 3.

Applications of BFT:-

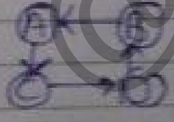


Starting from A:-
Ⓐ-Ⓑ-Ⓓ-Ⓒ Ⓘ-Ⓙ
Ⓔ-Ⓕ-Ⓖ

① Using BFT, we can check whether the graph is connected or not.

) To obtain the no. of connected components, we use BFT.
[connected component:- maximal subgraph which is connected] — O(V+E)

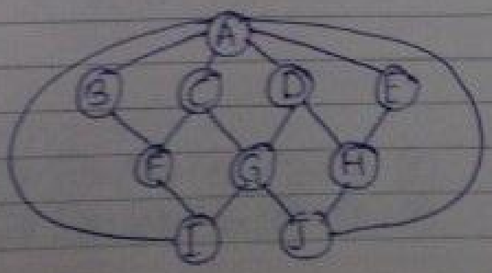
)



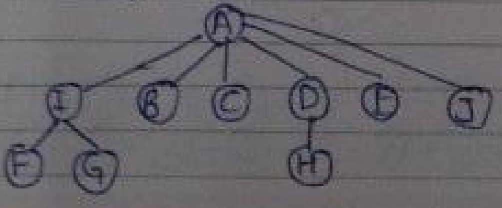
Ⓐ→Ⓑ→Ⓒ←Ⓓ
Ⓐ-Ⓒ-Ⓓ-Ⓑ

In a directed graph, if we come back to vertex where we have started the BFT, then graph contains cycle.

)



Using BFT



★★ Any new problem is given in GATE about Graph, most of the times it is BFT & DFT.

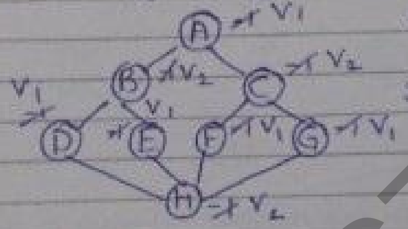
In the given unweighted graph (all edge weights are same), to find out shortest path from the given source, we use BFT algorithm.

★★ If given graph is unweighted, then
Dijkstra ⟶ $O[(V+E)\log V]$
Bellman-Ford → $O(VE)$
BFT ⟶ $O(V+E)$ ✓
BFT takes shortest time.

⑤ To check given graph is Bipartite or not, we use BFT algorithm

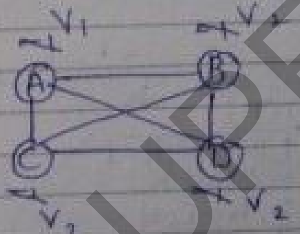
Step 1 :- Visit • graph & mark '-1' to all vertices :- $O(V+E)$

Step 2 :- Place parent in V_1 & children in V_2



V_1	V_2
A	B
D	C
E	H
F	
G	

⟶ Bipartite



V_1	V_2
A	B
	C
	D

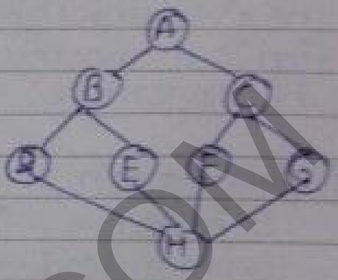
⟶ Not a bipartite. (as C & D are adjacent to B & are in V_2).

Depth First Traversal (DFT) :-

```
DFT(V)
{1. visited(V) = 1;
 2. print(V);
 3. for all w adj to V
      {4. if(w is not visited)
        5.    DFT(w);
      }
}
```

★ DFT will take stack as a data structure.

O/p:- A B D H E F C G



```
DFT(A)
1. A is visited
2. Print A
3. W = B,C
  4. B is not visited        C is now visited
  5. DFT(B)
     1. B is visited
     2. Print B
     3. W = A,D,E
        4. A is visited          E is visited now
           D is not visited
        5. DFT(D)
           1. D is visited
           2. Point D
           3. W=B,H
           4. B is visited
              H is not visited
           5. DFT(H)
              1. H is visited
              2. Point H
              3. W=D,E,F,G
              4. D is visited
                 E is not visited     F is not visited     G is visited no
              5. DFT(E)                5 DFT(F)
                 1. E is visited          1. F is visited
                 2. Point E               2. Point F
                 3. W = B,H               3. W= C,H
                 4. Both B & H visited    4. H is visited
                                             C is not visited
                                          5. DFT(C)
                                             1. C is visited
                                             2. Print C
                                             3. W = A,F,G
                                             4. A & F is visited
                                                G is not visited
                                             5. DFT(G)
                                                1. G is visited
                                                2. Print G
                                                3. W = C,H
                                                4. Both visited
```

★ No. of function calls = no. of vertices
in = $\boxed{V}$

★ In worst case, each function call 'for loop' is repeated (V-1) times in case of complete graph.

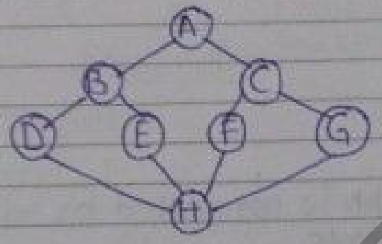
★ In best case, in each fn. call 'for loop' is repeated only once.

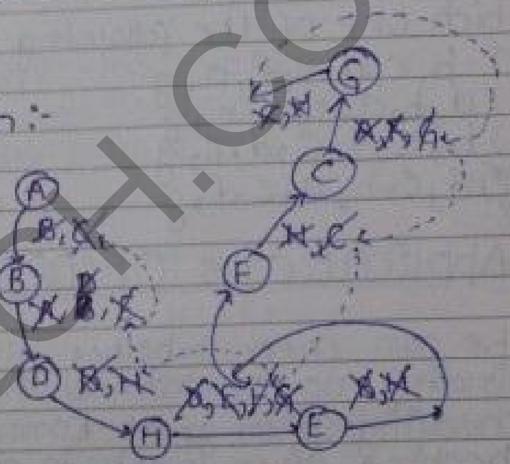
Note1:- In order to implement DFT, we uses stack data structure.

★Time complexity:-

$\theta(V\ \underbrace{V}_{\text{V fn. calls}}.(\overset{\text{Best case}}{1}+\underbrace{(V-1)}_{\text{worst case}})) = \boxed{\theta(V+E)}$ as $\boxed{V^2 \geq E}$

Q. Consider the following graph:-

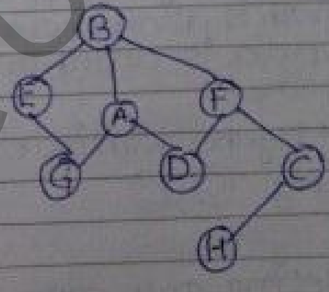




Ⓗ-Ⓔ-Ⓑ-Ⓐ-Ⓒ-Ⓖ-Ⓕ-Ⓓ

Ⓓ-Ⓗ-Ⓕ-Ⓒ-Ⓖ-Ⓐ-Ⓑ-Ⓔ

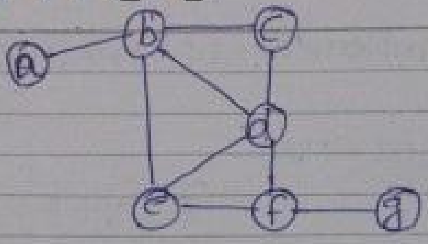
Q. Consider the following graph:-



Give 4 diff. DFT's starting from H.

① Ⓗ-Ⓒ-Ⓕ-Ⓓ-Ⓐ-Ⓑ-Ⓔ
② Ⓗ-Ⓒ-Ⓕ-Ⓑ-Ⓔ-Ⓖ-Ⓓ
③ Ⓗ-Ⓒ-Ⓕ-Ⓓ-Ⓐ-Ⓖ-Ⓔ
④ Ⓗ-Ⓒ-Ⓕ-Ⓑ-Ⓐ-Ⓓ-Ⓖ
⑤ Ⓗ-Ⓒ-Ⓕ-Ⓑ-Ⓐ-Ⓖ-Ⓓ

Q. Consider the following graph:-

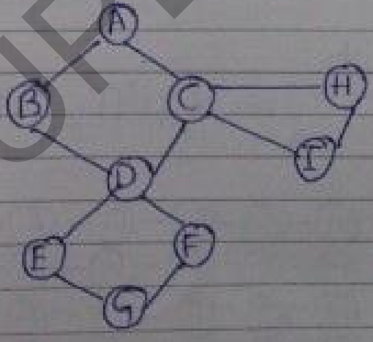


Which one of the following is correct DFT?

[illegible] b, c, d, e, f, a, g
(b) d, b, a, e, f, c, g
(c) b, a, e, c, d, f, g
✓ [illegible] d, e, b, a, c, f, g

Applications Of DFT

① We can check whether given graph is connected or not.
② Finding no. of connected components.
③ Checking given graph contain cycle or not.
④ Checking given graph is strongly connected or not.
⑤ ~~MQ.~~ Finding no. of articulation points or not.
⑥ Finding no. of bridges.



Articulation pts.:-

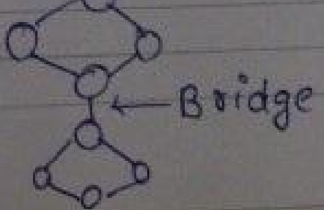
In the given connected graph, by deleting any vertex* if the graph is disconnected, then the vertex is called articulation point.

* together with its edges

e.g. Ⓒ, Ⓓ

Bridges:-

In the given connected graph, by deleting any edge, if the graph is ~~so~~ disconnected, then edge is called bridge.



Date

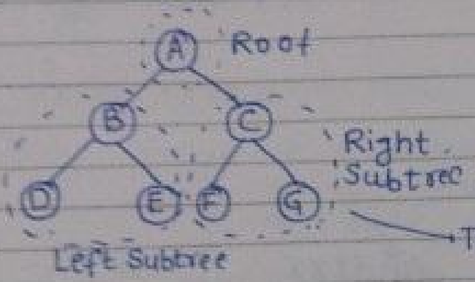
★In order to convert recurssive procedure to iterative procedure (or non-recursive procedure) requires stack data structure.

28.07.12 <u>Tree Traversal</u>

① Inorder (LST Root RST)
② Preorder (Root LST RST)
③ Postorder (LST RST Root)

Time taken for tree traversal:- $O(V+E)$ from graths; no. of edges in tree are (V-1)

$\therefore O(V+V-1) \Rightarrow \boxed{O(V)}$



preorder :- ABDECFG
postorder:- DEBFGCA
inorder :- DBEAFCG

There are $\boxed{V}$ function calls.

```
Preorder(t)
{ ~~print (t->data);~~
  ~~Preorder(t->~~
  if (t!=NULL)
   { printf(t->data);
     Preorder (t->leftchild);
     Preorder (t->rightchild);
   }
}
```

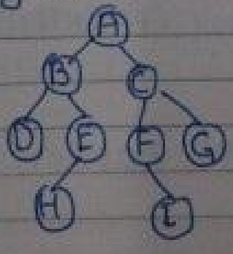
```
inorder (t)
{ if (t!=NULL)
   { inorder(t->leftchild);
     printf(t->data);
     inorder(t->rightchil
   }
}
```

★Note:- If the binary tree contain V nodes, inorder, preorder & postorder traver al takes $\boxed{\theta(V)}$ time.

```
postorder (t)
{ if(t!=NULL)
   { postorder(t->leftchild);
     postorder (t->rightchild);
     printf (t->data);
   }
}
```

Q1. Consider the following C program.

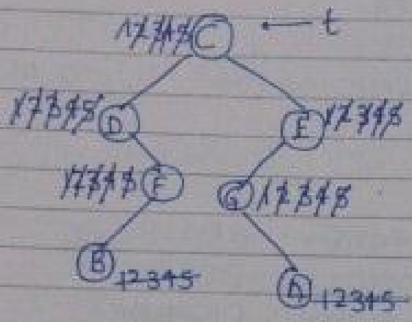
```
-> Aorder(t)
   { if (t!=NULL)
      { printf (t->data);
        Aorder (t->rightchild);
        printf(t->data);
        Aorder (t->leftchild);
      } printf (t->data);
   }
```



<u>O/p:-</u>

AEGGGCFIIIFFC
BEEHHHEBDOD

Q. Apply the above code on the following binary tree:-

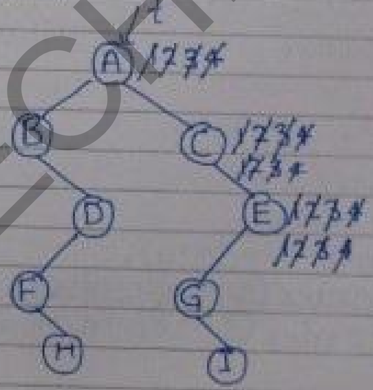


O/p:-

CEEGAAAGGECDFFBBBFDDC

Q. Consider following C program:-

```
B(order)
{if(t!=NULL)
  {1.Border(t→right);
   2.printf(t→data);
   3.Border(t→right);
   4.printf(t→data);
  }
}
```



O/p:-

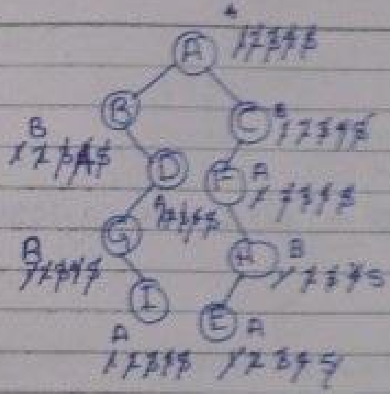
EECEECAEECEECA

Q. Consider following C program:-

```
A(ord Aorder(t)
    { if (t!=NULL)
        {1.printf(t→data);
         2.Border(t→right);
         3.printf(t→data);
         4.Border(t→left);
         5.printf(t→data);
        }
    }
```

```
Border(t)
{if(t!=NULL)
  {1.printf(t→data);
   2.Aorder(t→left);
   3.printf(t→data);
   4.Aorder(t→right);
   5.printf(t→data);
  }
}
```

start from A(order Aorder(t)



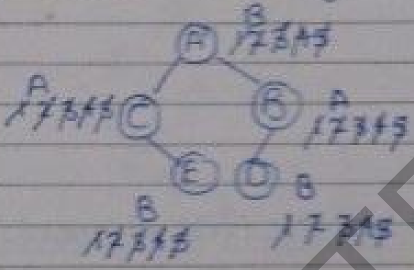
O/P:-

~~ABF/~~

ACFHEEEHHFFCCABBBDDGG [illegible]

BIIIGDBA

Q. Apply above C program starting from Border



O/P:-

~~[illegible]~~

ACEEECCABBDDDBA

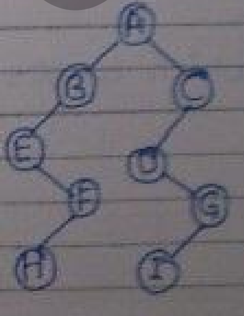
Q. Consider the following binary tree:

preorder :- ABDFHCEGI

inorder :- BFHDAEIGC

postorder :- HFDBIGECA

Q. Consider the following binary tree:-

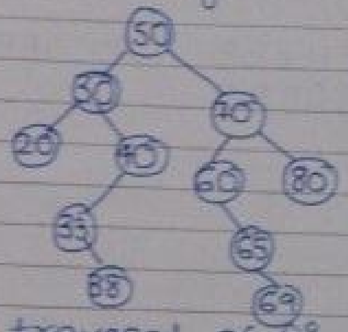


preorder :- ABEFHCDGI

inorder :- EHFBADIGC

postorder :- HFEBIGDCA

Q. Consider following BST :-



Inorder :- ~~2018~~
20,30,33,38,40,50,60,65,69,70,80

★ Inorder travesal of Binary Search tree will always give ascending order.

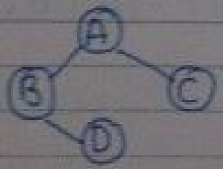
Q.

pre :- A, B, D, F, H, C, E, G, I
in :- B, F, H, D, A, E, I, G, C

Step 1 :- identify the root, it will be the first element of preorder

Step 2 :- now, check A in inorder, left to A ~~a~~ is LST of A & right to A is the RST of A.
∴ LST :- B F H D
RST :- E I G C
from preorder B comes first from ~~B/F/H/D~~ D F H B (LST)
& from preorder C comes first from EICG (RST).

Step 3 :- in order to find LST & RST of B check inorder
see left of B in inorder
LST :- NULL
RST :- F, H, D preorder
from ~~inorder~~, we check from F, H, D which comes 1st.
D comes 1st.



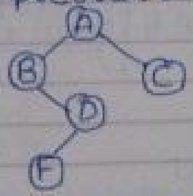
Step4 :- Checking LST & RST of D (from inorder)

LST :- H, F

RST :- Null

from preorder F comes 1st from H, F.

∴

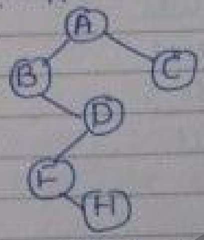


Step5 :- Checking LST & RST of F (from inorder)

LST :- Null

RST :- H

∴

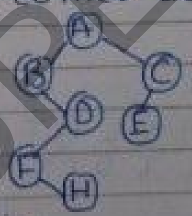


Step6 :- Checking LST & RST of C (from inorder)

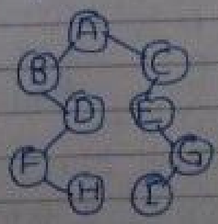
LST :- E, I, G

RST :- Null

Now E comes 1st from E, I, G from preorder



Step7 :- Similarly :-



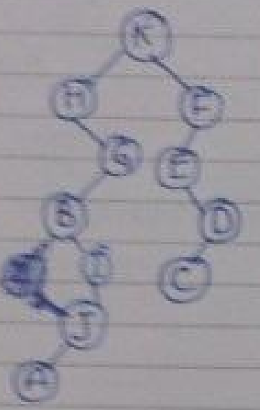
Note :-

If postorder & inorder are given, then procedu is similar, the only diff is that we have to chec the last element from postorder(i.e. the amon the elements of a subtree).

Time complexity

$\boxed{O(n^2)}$ (for constructing the tree from pre/post & inorder) [because for every element, we have to search all n elements & there are total n elements, imp]

Q. pre :- K, H, G, B, I, J, A, F, E, D, C
in :- H, B, I, A, J, G, K, E, C, D, F



K :- LST :- H B I A G
RST :- E C D F

H :- LST :- Null
RST :- B I A J G

G :- LST :- ~~Null~~ B I A J
RST :- Null

B :- LST :- Null
RST :- I, A, J

I :- LST :- Null
RST :- A J

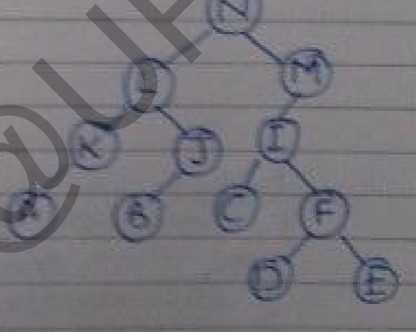
J :- LST :- A
RST :- Null

F :- LST :- ~~Null~~ E C D
RST :- Null

E :- LST :- Null
RST :- C D

D :- LST :- C
RST :- Null

~~post :- L, A, M, K, G, D, B, J, I, F~~
~~in :- L, K, M, A, E, D, C, F, J, I, B~~
post :- A, K, B, J, L, C, D, E, F, I, M, N
in :- A, K, L, B, J, N, C, I, D, F, E, M



N :- LST :- A K L B J
RST :- C I D F E M

L :-
LST :- A K
RST :- B J

J :-
LST :- B
RST :- Null

K :- LST :- A
RST :- Null

M :- LST :- C I D F E
RST :- Null

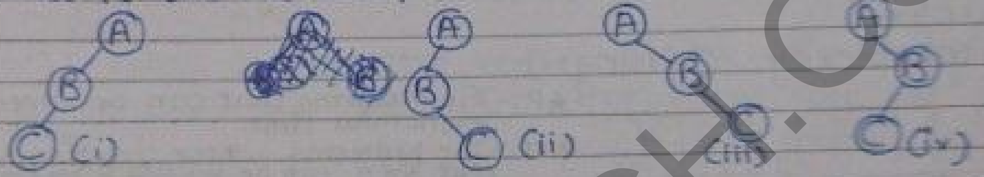
I :- LST :- C
RST :- D F E

F :- LST :- D
RST :- E

~~Bla~~ C :- LST :- Null
RST :- Null

Note:- In order to construct unique Binary Tree fro
the given (preorder & inorder) or (inorder
postorder) requires O(n^2)[worst case & avera
case].

Q. pre:- A, B, C
post:- C, B, A
what is the inorder?



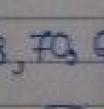
Note:- for the given preorder & postorder, unique binary tree is not possible, ∵ inorder is not available, ∴ inorder is necessary to construct unique binary tree.

Time Complexity:- There is no algo to construct unique Binary Tree for the given preorder & postorder other th
Brute Force, so it will take O(2^n) times.

★ If preorder & postorder is given & we are asked to
find inorder & 4 options

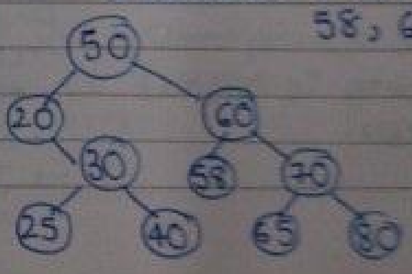
Q. A BST is traversed in preorder & values are printed i
the following order:-
preorder:- 50, 20, 30, 25. 40, 60, 58, 70, 65, 80
what is the postorder:-

50 → root
50:- LST :- 20, 30, 25, 40
RST :- 60, 58, 70, 65, 80



inorder of
BST in so
way
↓
inorder:- 20, 25, 30, 40,
58, 60, 65, 8

postorder:- 25, 40, 30, 20, 58, 65,
80, 70, 60, 50



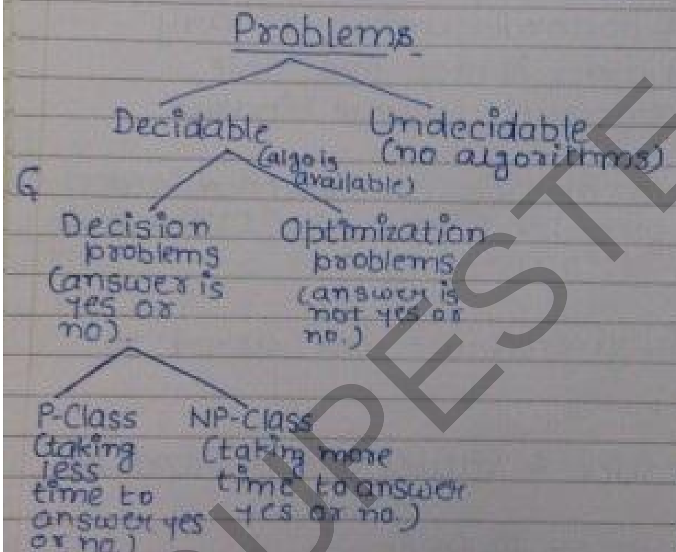
★NPC ~~problems~~ doesn't apply directly to optimization problems, but rather applies to decision problems, in which the answer is simbly 'yes' or 'no' (or, '0' or '1').

Note:- In order to construct unique B.T. from given (inorder & postorder) or (inorder & preorder) requires O(nlogn) time, if inorder is already sorted.

① → To sort inorder :- nlogn
② → To search every element in inorder, we can perform Binary Search :- logn
∴ & every element (n) will perform binary search,
∴ nlogn
∴ time complexity :- O(nlogn)

P, NP, NPH, NPC

★P :- The problems that can be solved in polynomial time.
★NP :- The problems whose solution when once given can be verified in polynomial time.



[time to solve the problem takes max. polynomial time.] $\log_2 n$ falls into this category, though it is not polynomial, but $\log_2 n$ < polynomial

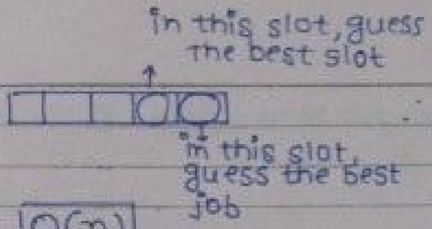
★Every P problem is NP, but not vice-verga.

★A problem 'A' is in P-Class if there exist a deterministic polynomial time algo to solve that problem.

★ P-Class problems :-
1. Binary Search :- $O(\log_2 n)$
2. Lineary Search :- $O(n^1)$
3. Job sequencing with deadline :- $O(n^2)$
4. Quicksort :- $O(n^2)$
5. LCS (m,n) :- O(mn)
6. MST- prim's :- O((V+E)logV)
× 7. Travelling Salesperson problem :- $O(2^n)$:- X → so it is not a p-class.
× 8. 0/1 knapsack :- $O(2^n)$ → so it is not in p-class.
× 9. SoS (m,n) :- $O(2^m)$:- so it is not in p-class.
10. matrix multiplication (strassen) :- $O(n^{2.81})$

- **NP-Class** :- A problem 'A' is in NP-class if there exist a non-deterministic polynomial time algo to solve that problem.
  → Linear Search → O(1) in case of NP, so without
  → Binary Search → O(1)
  → Job sequencing with deadline → O(n)

★ ★ if we can solve a problem w/o guess in polynomial time, then we can solve it with guess in polynomial tim



∴ n- job slots

∴ $\boxed{O(n)}$

④ Quicksort :-

take 1st pivot & guess its position → $O(1)$
similary other $(n-1)$ elements ∴ $nO(1) = \boxed{O(n)}$

⑤ LCS(n,n) :- $O(n)$

⑥ MST-prim's :- take one vertex & pick the smallest edge (guess) which will take $O(1)$ time, there are 'V' vertices, ∴ $\boxed{O(V)}$

⑦ Travelling Salesperson Problem :- $\boxed{O(V)}$

⑧ 0/1 Knapsack problem :-
for every object we have two choices
$(\_ \ \_ \ \_ \ \_ \ \_ \ \_)$
n objects (we can either take it or leave it.)

∴ guessing to take it or leave it, ∴ $O(1)$ time.
n-objects :- $\boxed{O(n)}$

⑨ SOS(n,m) :- $O(n)$

⑥ Strassen Matrix Multiplication :- $O(n^2)$

• Non-determinism :- Takes $\overset{max}{,}$ polynomial time, but only guesses & no proofs.

★ Every P-class problem is NP class problem, because P-class (with proof) are taking polynomial time then when done without proof it will take less than or equal polyn al time (like that in NP class).

★ ∴ every P-problem is also a NP-problem, but every NP-problem need not be a P-problem (e.g. Travelling Salesperson is NP but not P.)

~~$\boxed{P \subset NP}$~~ $\boxed{P \subset NP}$ } acc. to the above Statement.

★If we solve the Travelling Salesperson problem, ~~then~~ & other 2^n complex problems, then

$P = NP$

So, by now

$\boxed{P \subseteq NP}$

we can't say $\underline{P = NP}$ & $\underline{P \subset NP}$ alone

$\therefore \boxed{P \subseteq NP}$

have to be proved, area of open research.

- $P \subset NP \rightarrow$ it is true when someone proofs that Travelling Salesperson problem can't ~~take less than 2^n time~~ be solved in less than or equal to polynomial time.
- $P = NP \rightarrow$ it is true when someone proofs that Travelling Salesperson problem can be solved in polynomial time.

★~~Any pro~~ <u>P-Class</u> :- Any problem is solved in polynomial time without any guess.

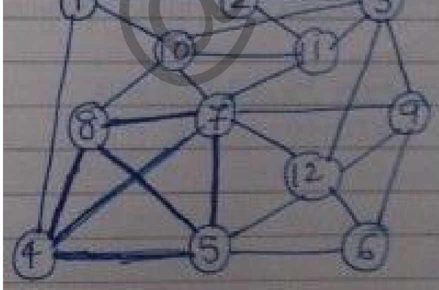
★ <u>NP-Class</u> :- Any problem is solved in polynomial time with guess.

★An algo ~~with~~ without guess gives time complexity of $O(2^n)$, then it is not P, but we can't say anything about whether it is NP ~~or~~ or not.

that's why $\boxed{P \subseteq NP}$

I/p :- Graph $G(V,E)$, integer K

Ques. Does G contain K-clique or not ?



Q. Does the given graph contain subgraph ~~with~~ which is k-vertex complete graph

↑

this problem is not P. because it takes $O(2^n)$ time complexity.

*With guess (NP-class) we choose a subgraph with k-vertices in O(1) time & then verify each vertic will take O(V) time [where V:- no. of vertices in original graph.]

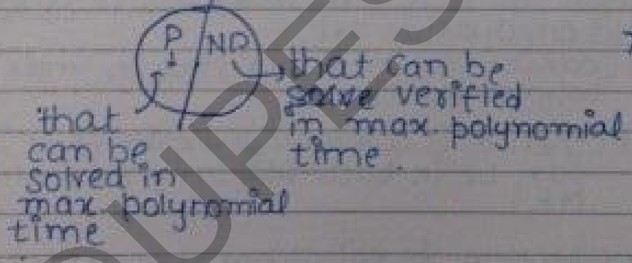
★ NP means :- if we are given correct soln. to a proble then we can verify the soln. in max. polynomia time (we dont solve the problem.)

★ P means :- we can solve the problem in polynomial time.

e.g. Linear Search :- Solution O(n) Verification (whether the element search is what we wanted or not ∴ verification will take O(1) time.

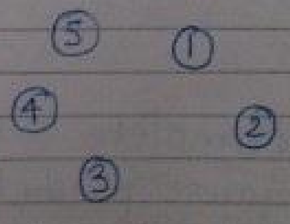
① P-Class :- any problem is solved (for all i/p's) ~~we can say~~ ~~yes or~~ no in polynomial time, then the problem is called as P-class.

② NP-Class :- any problem is verified (correct input only) in polynomial time is called NP-class.



★★ Any problem that can be easily solved (means problem can be solved in max. polynomial time) t it can also be verified easily (verification can be done in max. polynom time). ∴ every P probler is a NP problem but no vice-versa.

Polynomial Time reduction :-



- 3-SAT was the 1st NP problem discovered by Stephen Cook. It took 3 years.
- Next 3000 NP problems were discovered in next 10 years.

- A is an unknown problem & B is known problem, if A is polynomially reducible to B, then

  if B ~~is~~ ~~NP~~ takes 2^n time,

  then A will take $O(n^c)+O(2^n)=\boxed{O(2^n)}$ c :- constant.

★ conversion must be done in polynomial time.

- if B is NP, then A is NP.
- if B is P, then A is P.

★ If convesion takes more than polynomial time, then A & B dont have similar properties.

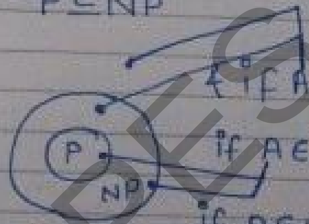
NP-Hard :-

A problem L is said to be NP-Hard if & only if L' is

- polynomially reducible to L for all L' ∈ NP.

$L' \leq_p L, \forall L' \in NP$

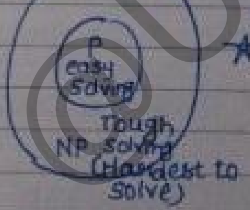
$P \subseteq NP$



{ if A ∈ NPH, then it can ~~only~~ be in only NP or outside NP.

if A ∈ P (can be solved & verified in max. polynomial time.)

if A ∈ NP (then it is dilemna, ~~whether~~ that it can be in both ~~P &~~ only P & only NP.)



every

★ To prove our problem is NP-Hard, then all problems in NP should be reducible to our problem in max. polynomial time.

★ all hardest problem in NP is reducible to L, then L is also Hardest problem.

NP-Complete :-

A problem L is said to be NP-Complete

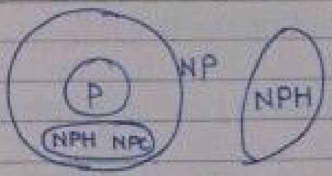
L is NP-Hard. (or)

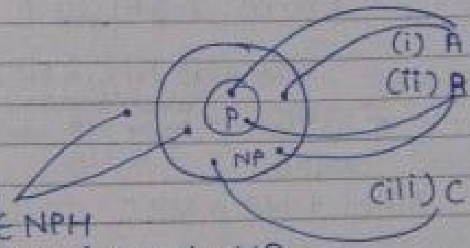
L ∈ NP.

A is NPC when $A \notin P$ $A \in NP$

(Hardest problem inside NP (but not in P) is NPC)

- $A \in NP \rightarrow$ means A is in NP but may or may not be in P.
- $A \in NPC \rightarrow$ means $A \in NP$ but not in P.
- $A \in NPH \rightarrow A \in NPC$ or $A \notin NP$





(i) $A \in P$ (In both)

(ii) $B \in NP$ (either in P & NP both or only NP)

(iii) $C \in NPC$ (only in NP, not in P)

(iv) $D \in NPH$ (either in only NP or outside NP).

Note1:- If L is NPC & $L \leq L'$ then L' is NPH. (Hardest)

(L is polynomially reducible to L')

NPC, NPH, undecidable $\longrightarrow$

★ Hardest Language (L) is reducible to L', then L' is hardest langua

Note2:- If L is NP & $L' \leq_p L$ then L' is NP. (Easier)

(right to left)

unknown on left.

P, NP, decidabl $\longleftarrow$

★ If languages are reducible to our easy language, then those languages are easy.

Q1. A, B & C are 3-decisions problems Let A be a NPC, then which are true/false?

(a) $A \in NP$ (T)

(b) $\forall L \in NP$, $L \leq_p A$ (T)

(c) If $C \in NP$ & $A \leq_p C$ then C is NPC (T)

(d) If $D \in NP$ and $D \leq_p A$ then D is NPC. (F)

(c) $\rightarrow$ C is in NP & A is polynomially reducible to our problem C, then our problem is hardest, $\therefore$ C is NPC.

(d) $\rightarrow$ D is in NP & our problem D is polynomially reducible to NPC problem, but NPC doesn't work from right to left, but D is NP.

Cosmos

. (d)→ if there is:- if $D \in NP$ & some $B \leq_p D$, then $B \in NP$.

Q. Two people 'A' & 'B' have been asked to show that a problem π is NPC.
A shows polynomial time reduction from π to 3-SAT
B shows " " " from 3-SAT to π.
then check for T/F?
(a) π - P (F)
(b) π - NP (T)
(c) π - NPH (T)
(d) π - NPC (T)

NP ← $\pi \leq_p$ 3-SAT → NP — This side NPC, & NPH doesn't work, ∴ only NP works. ∴ π is NP

3-SAT $\leq_p \pi$ → NPH

This side both NPH & NPC works but NPC can be converted into NPH.

∴ NPH + NP = NPC

Q. Let S be a NPC & (Q & R) be two other problems known to be in NP, if $S \leq_p R$ & $Q \leq_p S$, then T/F?
(a) R ∈ NPC (T)
(b) R ∈ NPH (F)
(c) Q ∈ NPC (F)
(d) Q ∈ NPH (F).

• $S \leq_p R$ → R is NPH
but R is in NP
∴ R is ~~in~~ NPC.

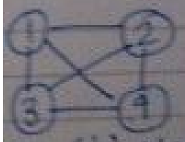
• $Q \leq_p S$ → ~~So~~ if our problem is polynomially reducible to Harder problem,
∴ Q is NP.

Euler Graph:- (If both euler path & euler cycle are possible).
Path:- sequence of zero or more edges. (not disconnected)

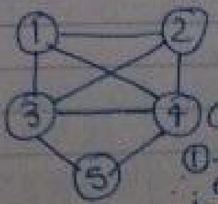
Euler Path:- It is a path which covers every edge of the given graph exactly once.

Euler ~~Cycle~~ Circuit:- It is a closed Euler Path (starting veter is also the ending vertex).

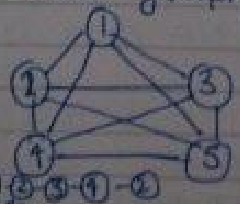
Check the following graphs are euler graphs or not.



(i) No euler path, & as no euler ~~path~~ cycle, there wont be euler cycle.



(ii)
①-②-⑤-④-①-②-③-④-②
euler path
but no euler circuit.



(iii)
①-②-④-⑤-③-
②-⑤
Euler ~~Cy~~ Path & Euler Circuit

★ If we have two vertices with odd degree & rest others with even degree, then (i) if we start from either of the odd degree vertices we will end up completing a euler ~~cycle~~ path on the other odd degree vertex (we can't get a euler circuit). (ii) & if we start with an even degree vertex, we can't even get a euler path.

Note1:- In the given graph, every vertex degree is even, then to that graph both euler path & euler circuit is possible, so given graph is euler graph.

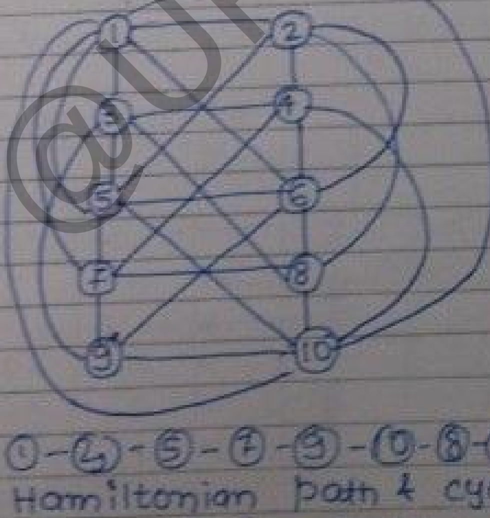
★ Checking given graph is Euler Graph or not, there is $O(V^2)$ time complexity algo, so it is a P-class problem. (checking). or $O(E)$ (for dense graphs) [checking degree of one vertex using adjacency matrix/list will take $O(V)$ time & we have to repeat this V times, ∴ complexity is $O(V^2)$.]

Note2:- If the given graph contain exactly 2 vertices with odd degree then to that graph euler path is possible but euler circuit is not possible, ∴ given graph is not euler graph.

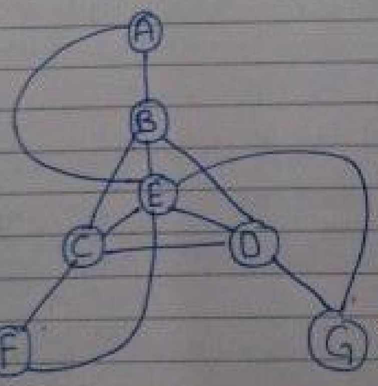
Note 3:- If the given graph contain more than 2 vertices with odd degree, euler path & euler circuit both are not possible.

Hamiltonian Graph

- Hamiltonian path:- It is a path in which every vertex of a given graph is covered exactly once.
- Hamiltonian cycle:- It is a closed Hamiltonian path (starting & ending vertex is same)
- Graph containing both Hamiltonian cycle & path is a Hamiltonian graph.



①-③-⑤-⑦-⑨-⑩-⑧-⑥-④-②-①
Hamiltonian path & cycle possible.
∴ Hamiltonian Graph.



Ⓐ-Ⓑ-Ⓒ-Ⓕ-Ⓔ-Ⓓ-Ⓖ
- Hamiltonian path
- but no hamiltonic cycle.

*There is no algo to check given graph is Hamiltonian graph or not other than Brute force, so it is one of the NPC.

3-SAT (3 variable satisfiable problem)

$(x_1 \vee x_2 \vee x_5) \wedge (x_6' \vee x_{10}' \vee x_2') \wedge (x_{1000} \vee x_8 \vee x_9) \wedge (x_{50} \vee x_{99} \vee x_{26})$

Note:- 3-SAT is NPC, but 2-SAT problem is P.

P-Class	NP-Class
① Shortest path	① Longest path.
② Euler	② Hamiltonian
③ 2-SAT	③ 3-SAT
④ Edge-cover	④ Vertex-cover