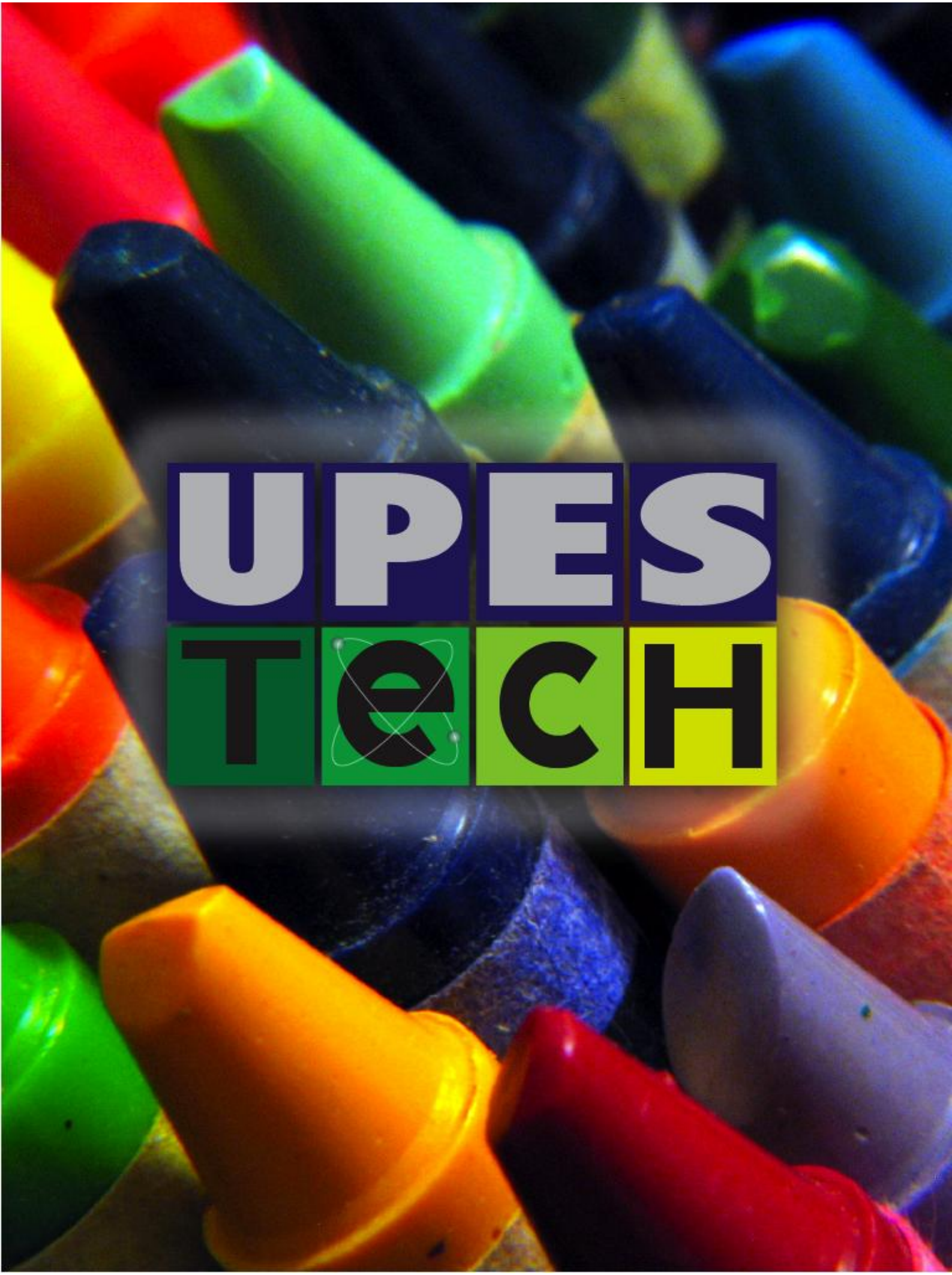

UPES
TECH

Date

Data Structures

Scope Of a Variable

① Static Scoping
② Dynamic Scoping

```
int a=500, b=600;
Main()
{int a,b;
 a=10; b=20;
 swap();
 printf("%d%d",a,b);
}
```

```
swap()
{int t;
 t=a;
 a=b;
 b=t;
 printf("%d%d",a,b);
}
```

free variables (means variables are not in this fn.)

O/P:
600 500
10 20

Static Scoping :-

O/P:- 600 500 10 20

when the free variables are taken from, as global variables.

Dynamic Scoping :-

O/P:- 20 10 20 10

when the free variables are taken from previous function call till we reach main().

If we dont get variables in even in main, then we refer dynamic scoping global variable.

* C language follow Static Scoping, older languages follow dynamic scoping.

O/P in C:
600 500 10 20

```
* int a=10, b=20;
  main()
  {int a=1, b=1;
   A();
  }
```

```
A()
{int a=2, b=2;
 B();
}
```

```
B()
{int a=3, b=3;
 printf("%d%d",a,b);
}
```

Date

04.08.12

- boolean b=true;
  procedure p
  begin
  pf(b);
  end
  begin
  boolean b=false
  p;
  end

· static scoping: true
· dynamic scoping: false

b [true] 1000

b [false] 2000

→ no name for this fn. (i: it is the main fn.)

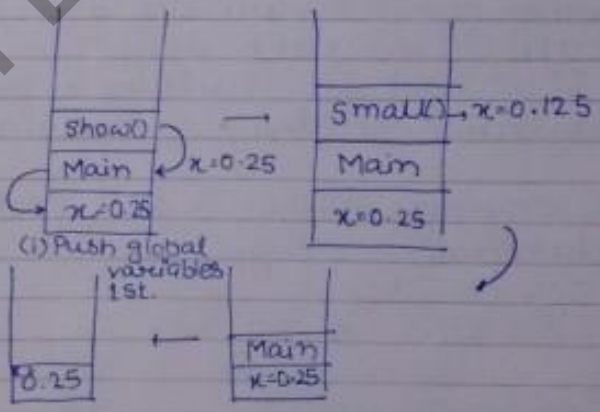
- var x:real
  procedure show()
  begin
  pf(x);
  end
  procedure small()
  begin
  var x:real
  x=0.125;
  show();
  end
  begin
  x=0.25
  show();
  small();
  end

→ no name for this fn, hence main

→ no concept of stack.
since at compile time, we know only global variable.
(at compile time)

· Static scoping: 0.25 0.25
· dynamic scoping: 0.25 0.125
(at run time)

For dynamic:-

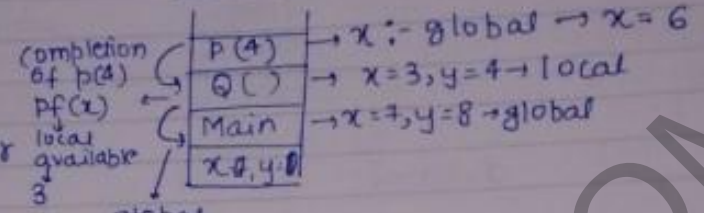


- var x,y:integer
  procedure p (var n:integer)
  begin
  x=(n+2)/(n-3);
  end
  begin
  x=7; y=8;

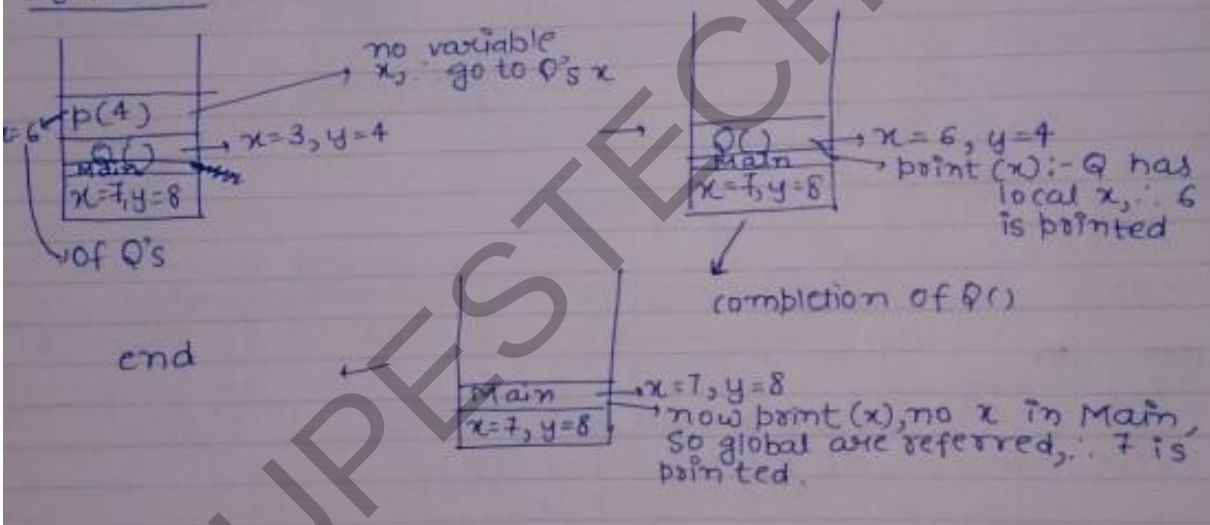
· Static: 3 6
· dynamic: 6 7

```
    Q();
    pf(x);
  end
procedure Q
  begin
    var x,y:integer
      x=3,y=4;
    p(y);
    pf(x);
  end
```

static:



dynamic:-



end

Q. var r:integer

```
    procedure two
    begin
      pf(r);
    end
    procedure one
     begin
      var r:integer
        r=5;
      two;
     end
```

changed by main → 2

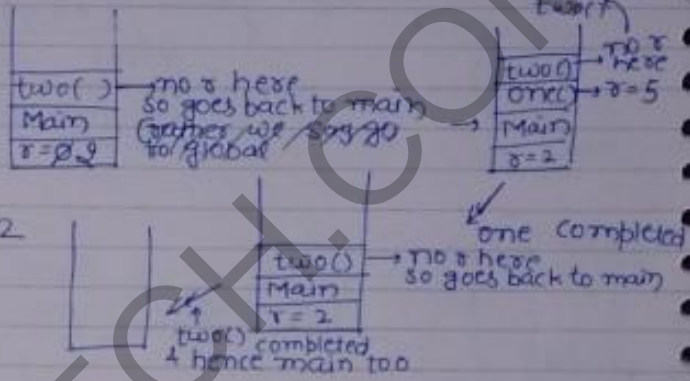
x (global) = 0, 1000

```
begin
  x=2;
  two;
  one;
  two;
end
```

Static: 2 2 2

- 1st 2: pointed by pf(x) of two (global variable)
- 2nd 2: printed by pf(x) of two (when called by one)

dynamic:



∴ o/p :- 2 5 2

- int a; → a, 1000
- a=55; → 0000 0000 0011 0111 (1000 1001) — big endian

1000 1001: 0011 0111 | 0000 0000 — little endian

7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1

55 → nearest power of 2 (smaller is 2^5)

$2^5 \rightarrow 32$ → 55 − 32 = 23

23 → nearest power of 2 (smaller is 2^4)

$2^4 \rightarrow 16$ → 23 − 16 = 7

7 → nearest power of 2 (smaller is 2^2)

$2^2 \rightarrow 4$ → 7 − 4 = 3

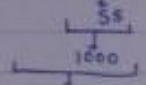
3 → nearest power of 2 (smaller is 2^1)

$2^1 \rightarrow 2$ → 3 − 2 = 1

1 → nearest power of 2 (smaller or equal)

$2^0 \rightarrow 1$ → 1 − 1 = 0

pf(a) → 55
pf(&a) → 1000
pf(*(&a)) → 55



value at address 1000

pf("%d", a) → 55
pf("%u", &a) → 1000

unsigned (for addresses)

pf("%d", *(&a)); → 55

- int a, → a | garbage | 1000
- a = 55; → a | 55 | 1000

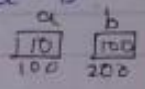
- a → 55
- b → 1000
- *b → 55
- *a → error (compilation error)
- &b → 2000
- *(&b) → 1000

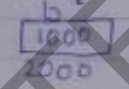
& → referencing operator
* → dereferencing "

```
int a=10, b=-1;
printf("%u", a); → 10
printf("%u", b); → 2's complement of -1 (if range of int is upto 4294967295) then the o/p is 4294967295)
printf("%d", b);
-1
```

int b = &a; // we can't write like this, // because writing int b, // we committed that we will // store a value at 'b'

we write :- int *b = &a;





printf("%u", b) → 1000

we use 'u' because we declared b as pointer

printf("%d", *b) → 55
printf("%u", &b) → 2000

```
int a=10;
int b=(int)&a;
pf("%d", a); → 10
pf("%d", &a); → 100
pf("%d", b); → 100
pf("%d", *b); → compile time error
```

Q1 int a = 10; → a | 10 | 1000

int *b = &a; → b | 1000 | 2000

- Single pointer :- storing address of a value.
- double pointer :- storing address of a pointer.

int **c = &b; → c | 2000 | 3000

because b is itself a pointer

int ***d = &c; → d | 3000 | 4000

d is storing an address which is a pointer to a pointer.

★If variable is holding a value, then to get the value → no '*' req
★" " " " single pointer, then to get the value → 1 '*' req
★" " " " double pointer, " " " " → 2 '*' req

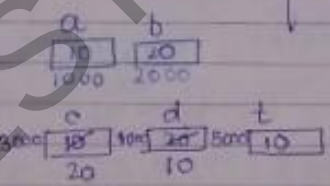
Parameter Passing Techniques

1. Call by value / Call by Copy
2. Call by reference
3. Call by copy-restore
4. Call by Name
5. Call by Need
6. Call by Text

Call by Value:-

```
main()
{ int a=10,b=20;
  printf("%d %d",a,b);
  swap(a,b);
  printf("%d %d",a,b);
}
swap(int c,int d)
{ int t;
  t=c;  // c=20
  c=d;  // d=10
  d=t;
}
```

O/p:- 10 20 | 10 20



★ In C compiler, by default call by value is done,
i.e. swap(a,b) → call by value

★ If any compiler supports call by reference by default
then swap(a,b) → call by reference
(it will pass addresses of a & b, & not values of a & b)

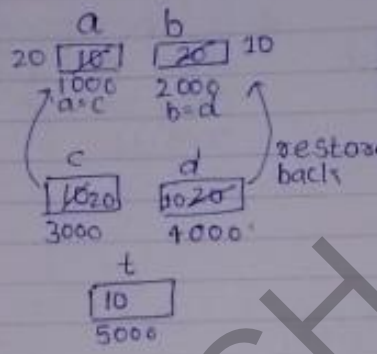
- Without using temp. variable:

```
a=a+b;
b=a-b;
a=a-b;
```

Call by Copy-Restore

```
main()
{int a=10,b=20;
 printf(a,b);
 swap(a,b);
 printf(a,b);
}
swap(int *c, int *d)
{int t;
  t=c;
  c=d;
  d=t;
}
```

★ executing the program in a compiles having call by copy-restore by default.



o/p:-

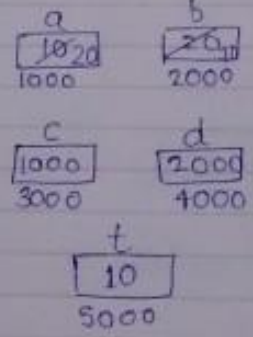
10 20 | 20 10

★ Changes to actual parameter takes place at the end using call by copy restore.

★ Changes to actual parameter takes place on the fly using call by reference.

Call by reference

```
main()
{int a=10,b=20;
 printf("%d %d",a,b);
 swap(&a,&b);
 printf("%d %d",a,b);
}
swap(int *c, int *d)
{int *t;
  t=*c;
 *c=*d;
 *d=t;
}
```



o/p:-

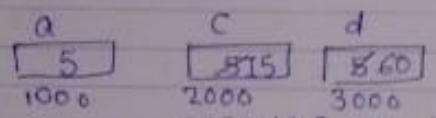
10 20 | 20 10

Q. Consider the following program:-

```
main()
{ int a=5;
  Test(a,a);
  printf(a);
}
Test(int c,int d)
{ c=d*2;
  d*=c-3;
}
```

* Call by reference & call by copy restore gives diff. answer then program contain aliasing.
* If the program contains aliasing then call by reference & call by copy restore may or may not generate same result

Call by value:



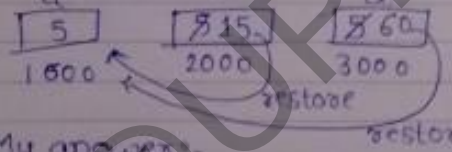
c=c+d*2 → c=c+d*2 → c=5+10 → 15

d*=c-3 → d=d*(c-3) → d=d*(12) → 5*12 → 60

in this case, always R.H.S. is calculated first irrespective of the priorities.

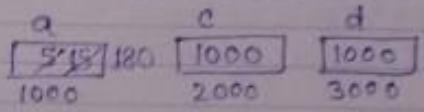
O/p:-
5

Call by Copy-Restore:



My answer:-
O/p:- 60 15/60
whichever is restored last is the output.

Call by reference:-



(*c)=(*c)+(*d)*2;
(*c)= 5 + (5*2)
= 15
(*d)=(*d)*((*c)-3);
= 15 *(15-3)
= 15*12
= 180

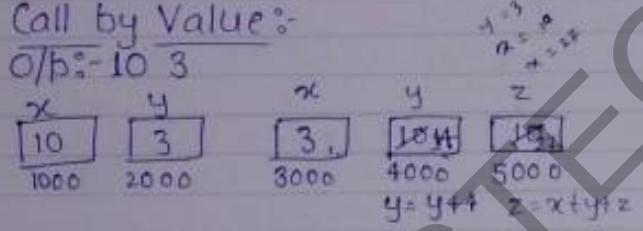
O/p:-
180

Q. Consider the following programs:-

```
main()
{ int x=10,y=3;
  fun(y,x,x);
  printf(x,y);
}
fun(int x,int y,int z)
{  y=y+4;
   z=x+y+z;
}
```

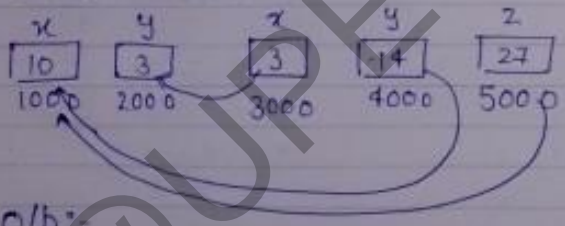
Call by Value:-

O/P:- 10 3



Call by Copy restore:-

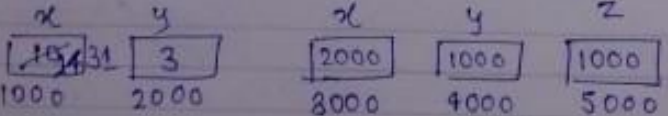
same as above



O/P:-

14/27 3

Call by reference



*y = *y+4
= 10+4
= 14

*z = (*x)+(*y)+(*z)
= 3+14+14
*z = 31

O/P:-

31 3

Aliasing :- x/y

- int x; → [ ] 1000

int &y = x; → means y address is same as x address.

*C doesn't support aliasing.

Q. Imp

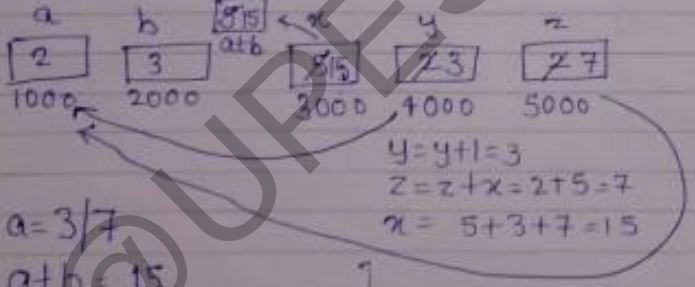
```
main()
{ int a,b;
  a=2; b=3;
  P(a+b,a,a);
  printf(a,b);
}
P(int x, int y, int z)
{ y=y+1;
  z=z+x;
  x=x+y+z;
}
```

this is done in temporary variable & its address is passed.

Call by Value :-

O/p :- 2, 3

Call by Copy restore :-



y = y+1 = 3

z = z+x = 2+5 = 7

x = 5+3+7 = 15

a = 3/7

a+b = 15

if a = 3, then b = 12

if a = 7, then b = 8 → My answer

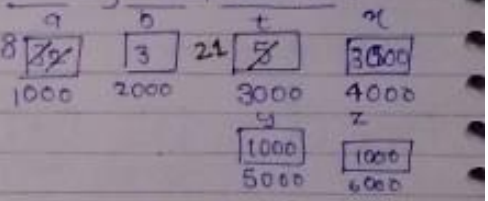
∴ O/p :-

3/7 12/8 → My answer

but answer :-

3/7 3

Call by reference :-



*y = *y+1

= 2+1

= 3

*z = (*z)+(*x)

= 3+5 = 8

*x = (*x)+(*y)+(*z)

= 5+8+8

= 21

O/p :-

8 3

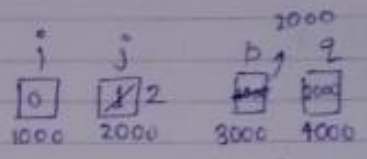
Q. main()
{ int a=10, b=20;
swap(&a, &b);
printf(a,b);
}
swap(int *c, int *d)
{ interchange(&c, &d);
printf(c,d);
}
interchange(int **e, int **f)
{ A(&e, &f);
}
A(int ***g, int ***h)
{ int t;
t=g; // t = ***g;
g=h; // ***g = ***h;
h=t; // ***h = t;
}

Call by reference:-

O/P:-
20 10

Q. Consider following C-program:

```
#include <stdio.h>
void f(int *p, int *q)
  { p = q;
    *p = 2;
  }
int i = 0, j = 1;
main()
{ f(&i, &j);
  printf("%d %d", i, j);
  return(0);
}
```



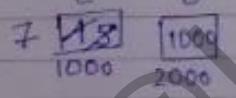
*p = 2

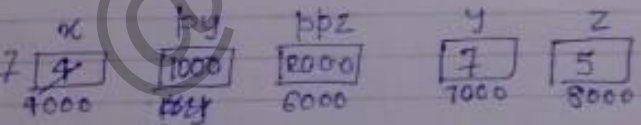
Output:- 0 2

Q. What is the o/p printed by the following C-program:

```
main()
{ int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  printf("%d", f(c, b, a));
}
```

```
f(int x, int *py, int **ppz)
{ int y, z;
  **ppz += 1;
  z = **ppz;
  *py += 2;
  y = *py;
  x += 3;
  return(x+y+z);
}
```





7 + 7 + 5 = 19

O/P:-

19

**ppz = **ppz + 1;
= 4 + 1 = 5

z = **ppz
= 5

*py = *py + 2
= 5 + 2 = 7

y = *py;
y = 7

x = x + 3
= 4 + 3 = 7

Q. Consider the following C program:- (Static scoping, by default as it's a C program)

```
#include<stdio.h>
int x;
void Q(int z)
{ z+=x;
  printf(z);
}
void P(int *y)
{ int x= *y+2;
  Q(x);
  *y=x-1;
  printf(x);
}
main()
{ x=5;
  P(&x);
  printf(x);
}
```

x	y	x(of P)	z
G 5	1000	7	7
1000	2000	3000	4000

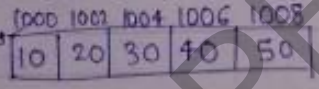
$x = *y + 2 = 5 + 2 = 7$

$*y = 7 - 1 = 6$

$z = z + x = 7 + 5$ (global) $= 12$

O/P:
12, 7, 6

Q. int a[5] = {10, 20, 30, 40, 50};



- array name indicates starting element address
- variable name indicate inside value.

a = 1000

*a = 10

a+1 = 1000 + 1*2 = 1002 [as integer is of 2 bytes]

*(a+1) = *(1002) = 20 = a[1]

*(a+4) = 50 = a[4]

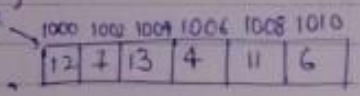
☆ at time of preprocessing a[4] is converted to *(a+4).

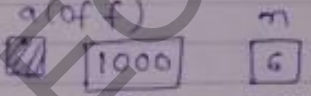
a[50] = *(a+50) = *(base address + 50×2)

↑ random access is possible due to pointers.

- a+0 → skipping 0 elements → a[0]
- a+1 → " 1 " → a[1]
- a+2 → " 2 " → a[2]

Q.
```
#include<stdio.h>
int f(int *a,int n)
 { if(n<=0)
    return 0;
   else
    if(*a%2==0)
     return *a+f(a+1,n-1);
    else
     return *a-f(a+1,n-1);
 }
main()
 {int a[]={12,7,13,4,11,6};
  printf("%d",f(a,6));
  return 0;
 }
```





O/P:-

15

$12+(7-(13-(4+(11-(6+0)))))$
$= 12+(7-(13-(4+5)))$
$= 12+(7-(13-9))$
$= 12+(7-4)$
$= 12+3=15$

1000 1002 1004 1006 1008 1010
12 7 13 4 11 6

3000
$*a\%2 = 12\%2 = 0$
$\therefore *a + f(a+1,n-1)$ → $12+3=15$

$12+f(1002,5)$
$7-4=3$
$7+f(1004,4)$
$13-9=4$
$13+f(1006,3)$
$4+5=9$
$4+f(1008,2)$
$11-6=5$
$11-f(1010,1)$
$6+f(1012,0)$
0

Q. What is the O/p of the following C-program?

```
#include<stdio.h>
int f(int n, int k)
 {if (n==0)
    return 0;
  else
   if (n%2)
    return f(n/2, 2*k)+k;
   else
    return f(n/2, 2*k)-k;
 }

main()
 {printf("%d", f(20,1));
 }
```

O/p :-

9

f(20,1) ← 9

f(10,2) - 1 ← 10-1=9

f(5,4) - 2 = 12-2=10

f(2,8)+4 = 8+4=12

f(1,16)-8 = 16-8=8

f(0,32)+16 = 16

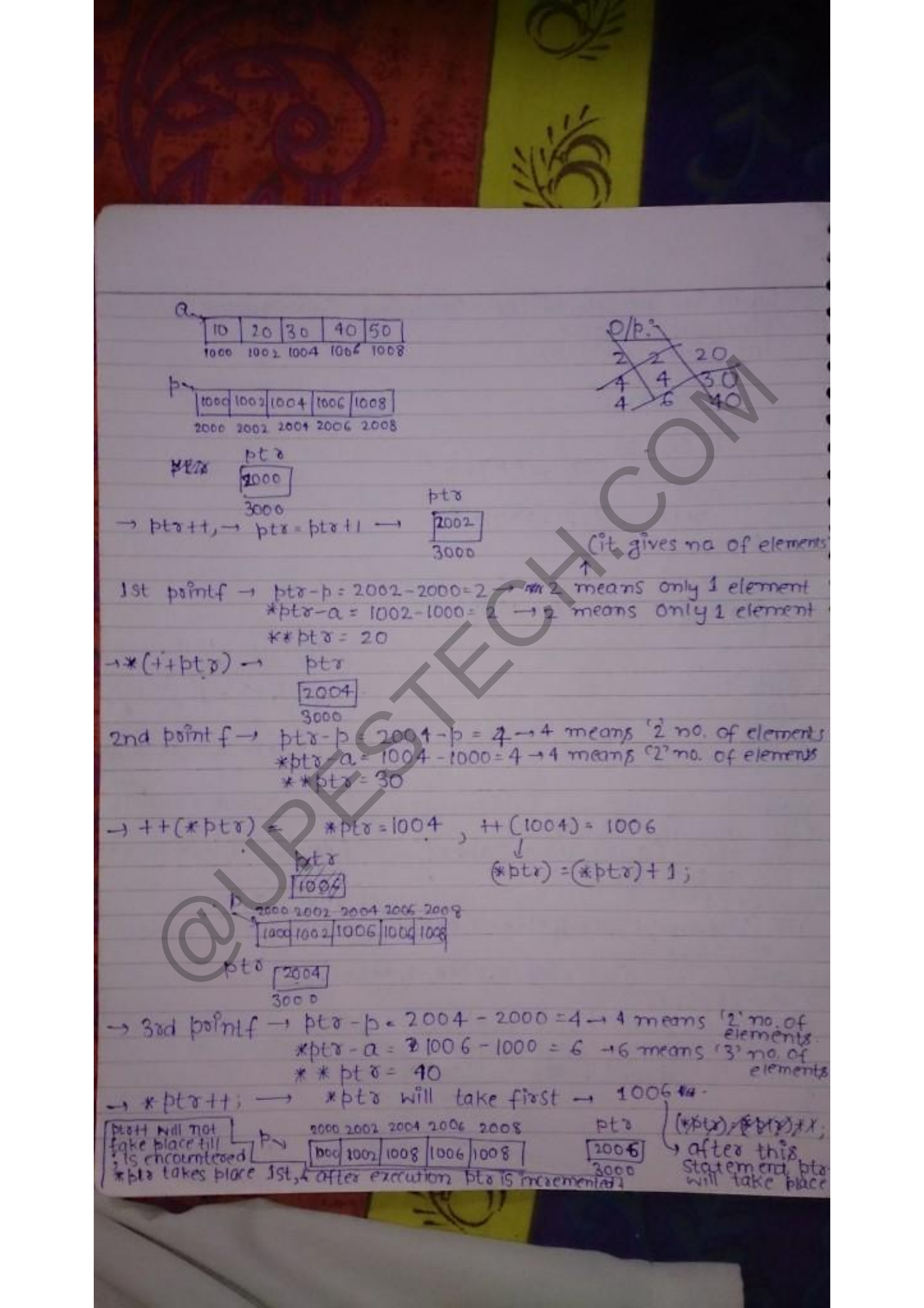
↓

0

Q.

```
Main()
{int a[]={10,20,30,40,50};
 int *p[]={a,a+1,a+2,a+3,a+4};
 int **ptr=p;
 ptr++;
printf("%d %d %d",ptr-p,*ptr-a,**ptr);
  *(++ptr);
 printf("%d %d %d",ptr-p,*ptr-a,**ptr);
  ++(*ptr);
 printf("%d %d %d",ptr-p,*ptr-a,**ptr);
  *ptr++; ← unary operators, execution from right to left
 printf("%d %d %d",ptr-p,*ptr-a,**ptr);
}
```

ptr++ = ptr & after semicolon is encountered then only ptr is incremented



★Note:- '*' and '++' having same priority, so we will go to associativity, & associativity is right to left.

→ 4th printf statement → ptr - p = 2006 - 2000 = 6 → 6 means 3 no of elements
*ptr - a = 1006 - 1000 = 6 → 3 means '4' no. of elements
**ptr = 50 40

★ If two addresses undergoes arithmetic opn.

O/P

1 1 20
2 2 30
2 3 40
2A//50 3 3 40

Q. #include<stdio.h>
main()
{ int a[] = {0,1,2,3,4};
int *p[] = {a, a+1, a+2, a+3, a+4};
int **ptr = p;
**ptr++;
printf("%d %d %d", ptr-p, *ptr-a, **ptr);
*++*ptr;
printf("%d %d %d", ptr-p, *ptr-a, **ptr);
++**ptr;
printf("%d %d %d", ptr-p, *ptr-a, **ptr);
}

a

0	1	2	3	4
1000	1002	1004	1006	1008

p

1000	1002	1004	1006	1008
2000	2002	2004	2006	2008

ptr
2000
3000

→ **ptr++;
↳ from right to left
ptr++ will take place 1st
∴ ptr++, ⇒ ptr = 2000
& *ptr = 1000
**ptr = 0
& after ; , ptr++, ∴ ptr = 2002

O/P:- 1 1 1
1 2 2
1 2 3

1st printf:- ptr - p = 2002 - 2000 = 2 → 2 means '1' element
*ptr - a = 1002 - 1000 = 2 → " " " "
**ptr = 1

→ *+++*ptr → from right to left
*ptr → 1002
++*ptr → (*ptr) = (*ptr) + 1

∴ p →

1000	1004	1004	1006	1008
2000	2002	2004	2006	2008

*1004 = 2

2nd printf:- ptr - p = 2002 - 2000 = 2 → 2 means '1' element
*ptr - a = 1004 - 1000 = 4 → 4 " '2' elements
**ptr = 2

→ ++**ptr → from right to left
*ptr = 1004
**ptr = 2
++ **ptr → **ptr = **ptr + 1

∴ a

0	1	3	3	4
1000	1002	1004	1006	1008

∴ 3rd printf → ptr - p = 2002 - 2000 = 2 → 2 means '1' element
*ptr - a = 1004 - 1000 = 4 → 4 " '2' element
**ptr = 3 [*ptr = 1004 → **ptr = 3]

★ Subtracting two pointers means gives the total size of the elements that can reside in b/w them.
if the elements are integers, then
total no. of int = total size b/w them / 2

*(*ptr)
*(*ptr++);

Note1:- For the pointer variable, we can add integer constants.
e.g. $p = p+5$ $p_1 + i$ (skipping i elements in forward direction).

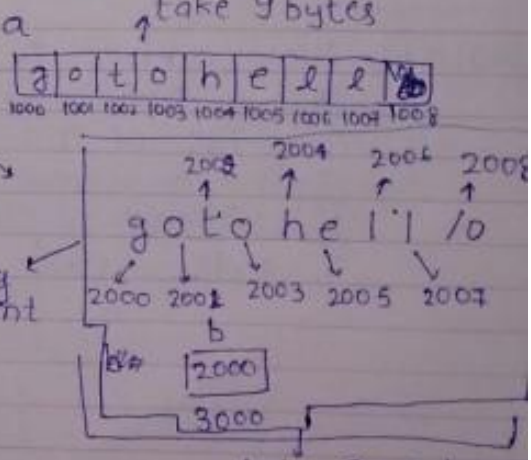
Note2:- We can subtract integer constant from pointer variable
e.g. $p_1 = 1014$
$p_1 = p_1 - 1 = 1014 - 2*1 = 1012$
$p_1 - i$ (skipping i elements in backward direction)
e.g. p:1000
q - 1008
q-p = given no. of elements in

Note3:- Subtraction b/w 2 pointers is possible (p,q) iff
p-q [$p \geq q$ / $p < q$ should be pointers of the same array]
where 'p-q' indicates no of elements b/w two pointers including 1st one & excluding last one.

Note4:- Multiplication, division & addition is not possible b/w two pointers because it has no meaning.

Q.
```
main()
{ char a[] = "gotohell";
  char *b = "gotohell";
  a[3] = printf("%s", a);
         printf("%s", b);
  a[3] = 'w';
  b[3] = 'w';
  printf("%s", a);
  printf("%s", b);
}
```

Array of characters



* a=a+5 → error (we can't change the base address of the array.)
* but b=b+3 → no error because b is character pointer & can be changed.

* printf("%s", 1000) → means it will print characters starting from 1000 address till null character (null character is not printed.)

* printf("%s", *1000) → it will print only one characters at '1000' address.

∴ printf("%s", a) → gotohell
printf("%s", a+1) → hell
printf("%s", *(a+1)) → l
printf("%s", b) → gotohell
printf("%s", b+5) → ell
printf("%s", *(b+5)) → e
printf("%s", *(a+3)) → o

• a[3] = *(a+3)
*(a+3) = 'w'

a	g	o	t	w	h	e	l	l	\0

• b[3] = *(b+3)

printf("%s", a); → gotohell
printf("%s", b); → gotohell

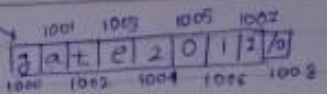
Note 1:- Base address of the array can't be changed but string constant can be changed.
Contents of the array can be changed, but string constant content can't be changed, if contents of string constant are changed, result is undefined.

Date

05.08.12

```
main()
{ char p[]="gate2012";
  printf("%s",p+p[3]-p[1]);
}
```

O/P:
te2012 2012



1003+1003-1001
= 1006+1
= 1002

p[3] = *(p+3) = e
e-a 1004 4
p[1] = *(p+1) = a

★Warning in compilation takes place (program) When there is no loss of data but it doesn't follows semanti rules.

eg. int a,
char b;
a=b; //only warning, implicitly it is converting a=(int) b
//coercion (implicit type conversion)
// b=a; → this generate error message, because loss of dat
// we are storing 2 byte data into 1 byte storage, & hence
//error.
b=(char) a; //correction, explicit conversion.

Semantic rule
Lower bit variable using lower memory can be stored in a variable using more memory but not vice versa.

★ printf("%c",'a'); → a
★ printf("%d",'a'); → 97

★ Lower Address → Higher Byte
Higher " → Lower Byte
→ Big endian

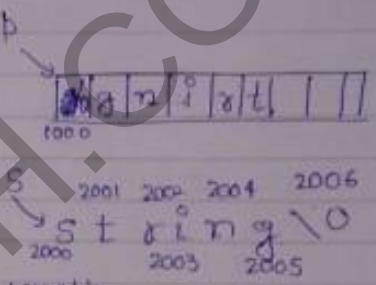
★ Lower Address → Lower Byte
Higher " → Higher "
→ Little endian

★ char s='a';
s=s+1; // s now contains 'b'
★ char s=255;
s=s+1; // s will be 0.
★ char s;
s='a'+'b'; → s=97+98=195
s='b'-'a'; → s=98-97=1

★ *(p+i) = p[i] } These two are same.
*(i+p) = i[p]

★ printf("%s", p); ⟶ gate2012
★ printf("%s", p+5); ⟶ 012
★ printf("%c", *(p+5)); ⟶ 0
★ printf("%c", p); ⟶ error message, because we are using address with %c
★ printf("%s", *p); ⟶ error message, as *p means g
★ printf("%c", *p); ⟶ g

Q. main()
{ char p[20];
char *s = "string"; //string constant
int length = strlen(s);
for(int i=0; i<length; i++)
{ p[i] = s[length-i]; }
printf("%s", p);
}

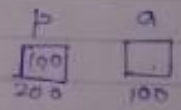


O/P :- Nothing (because first character is \0 itself).

★ sizeof("abcd"); ⟶ 5 (including \0) sizeof means the size of "abcd" in the memory, ∴ 5 Byte.
★ strlen("abcd"); ⟶ 4

Q main()
{ char *p;
printf("%d %d", sizeof(p), sizeof(*p));
}

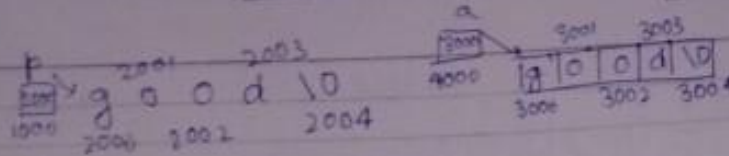
→ let Address to be of 4 Bytes



O/P:
4 1 → *p means p is pointing to a character, which takes 1 B.
→ sizeof(p) means p store address, address takes 4 B.

Q main()
{ char *p = "good";
char a[] = "good";
printf("%d %d %d", sizeof(p), sizeof(*p), strlen(p));
printf("%d %d", sizeof(a), strlen(a));
}

Note:- void a=10; → sizeof(a) → error
void *b=&a; → sizeof(b) → 'b' is a pointer which holds an address & hence takes 4B.



O/P:-
4 1 8 4
4 8 5 4 8 4
sizeof(*p) → p is pointing to a character, which is of 1B.
sizeof(p) → p stores an address
} → when we take address to be taking 4 Bytes

void a;
int b;
float c;
char d;
a=b;
a=c;
a=d;
} → NO ERROR, without typecasting this is possible advantage of void type.

* Any datatype can be stored in void variable & void can be stored in any variable (with loss of data).
* (without loss of data)
b=(int)a; } have to do explicit typecasting
c=(float)a;
d=(char)a;

if:- int a[5];
sizeof(a); → size of the array :- 10B
char a[]="good";
sizeof(a); → size of the array :- 5B (including null character)

Q
```
main()
{ int i=-1, j=-1, k=0, l=2, m;
  m=((i++ && j++ && k++); l++);
  printf("%d %d %d %d %d", i, j, k, l, m);
}
```
m = 1

O/P:
0 0 1

My answer:- -1 -1 1 2 1
But answer:- → this whole statement is true, no need to do l++
m = ((i++ && j++ && k++); l++);
→ left to right because && is binary operator
-1 && -1 && -1 = true = 1
O/P:- 0 0 0 2 1, l++ wasn't done.

* if (0.10)
printf("hi");
else
printf("bye");
O/P:- hi

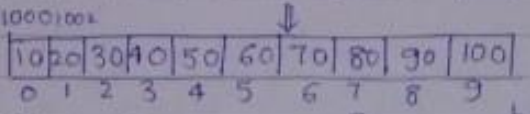
~~* else~~ A=10
B=0
if (A=B)
printf("hi");
else
printf("bye");
O/P:- bye

★ if k=0
then o/p:- 0 0 1 0 3 1 ← (i++ && j++ && k++) is false
∴ we need to execute l++

★ if i=0, j=2, k=3 & l=2
∴ o/p:- 1 2 3 3 1

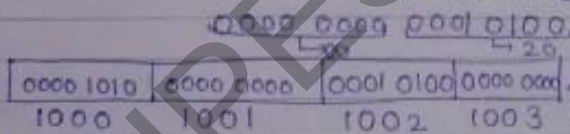
Imp

★ int a[10] = {10,20,30,40,50,60,70,80,90,100}



int *b; = (int *) a;
char *c; // c waiting for an address of character.
float *d;
void *e;
c = (char *)a; → 1000
d = (float *)a; → 1000

printf(d) → 1000
printf(*d) → 10 20 (because d is a float pointer & will print 4 Bytes which is the size of float)

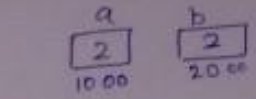


it is in little endian & hence has to be reversed
∴ o/p:- 20 10
If it is in big endian, then o/p will be:- 10 20

printf(c) → 1000
printf(*c) → only 1 Byte will be retrieved, ∴ only 1000 address value is pointed, ∴ o/p:- 10
printf(e) → 1000
printf(*e) → error.

★ if c++, then c = 1001
★ if d++, then d = 1004
★ if b++, then b = 1002
★ if e++, then error.

* No compilation error, but windows stopped the program execution.

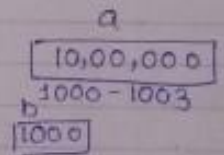


Q.
int a=2;
int *b=(int *)a; → b [2] 2000 → 2 is an integer pointer now, which is integer value before
*if(b) printf(b) → 2
printf(*b) → garbage (value at address 2,3)
++b → 4 (because 2 is now integer pointer & hence ++b will increment it to 4)
++a → 3 (because value in a is integer value.)

Q int a=2;
float *b=(float *)a; → b [2] 2000 → integer value '2' is converted to floating pointer
printf(b) → 2
* printf(*b) → garbage (value at address 2,3,4,5)
++b → 6 (because floating pointer increment it by 4).
++a → 3

Q int a=2;
char *b=(char *)a; → b [2] 2000 → integer value '2' is converted to character pointer
printf(b) → 2
* printf(*b) → garbage (value at address 2)
++b → 3 (increment by 1 only because character pointer)
++a → 3

Q float a=10,00,000; → takes 4 Bytes
int *b=(int *)&a;
printf(b) → 1000
*printf(*b) → since b is an integer pointer & hence will print 2 Bytes out of 4 Bytes (hence data loss)



* if address lines are of 2 Bytes
then float a=10,00,000;
int *b=(int *)a
we are storing a (4 Bytes data) into 2 Bytes integer pointer & hence the 4 Byte a (converted to integer pointer) is stored in 2 Byte 'b' & hence address loss.

Static Variable

```
main()
{ static int var = 5;
  printf("%d", var--);
  if (var)
    main();
}
```

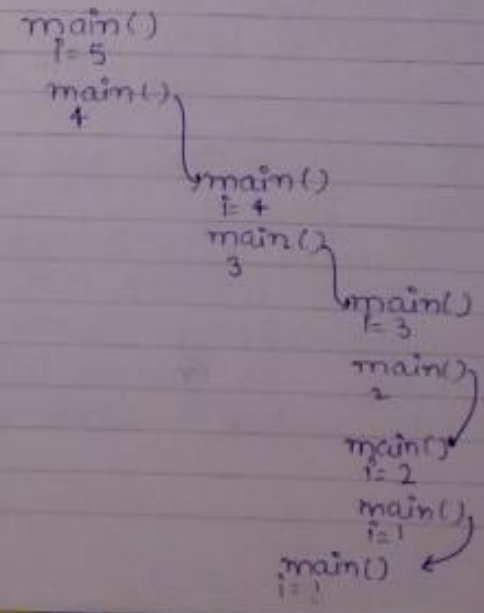
O/P:-
54321

* Writing 'static' means at the time of compilation time memory is allotted. (Static binding) [compile time memory is allocated]
* if 'static' isn't there then the memory is allotted at run time. (dynamic binding) [memory is allocated at run time]

① For the static variable memory will be created only once during its life time (compile time only).
(Recompiling the program will allocate different memory location to the static variable.)
② For the static variable initialization will be done once.
(By default)
③ By default Static & Global variable are initialized to zero.

Q.
```
void main()
{ static int i=5;
  if(--i)
  { main();
    printf("%d",i);
  }
}
```

O/P:
0000



2) main() int f(int n)
{ { static int r=0;
if(n<=0)
return 1;
if(n>3)
{r=n;
return f(n-2)+2;
}
return f(n-1)+r;
}

What is the value of f(5)?

f(5)
(n>3)
return f(5-3)+2 = 16+2=18

f(3)
return f(2)+5; = 11+5=16

f(2)
return f(1)+5 = 6+5=11

f(1)
return f(0)+5 = 6

f(0)
return 1

r
5
1000

O/P:-
18

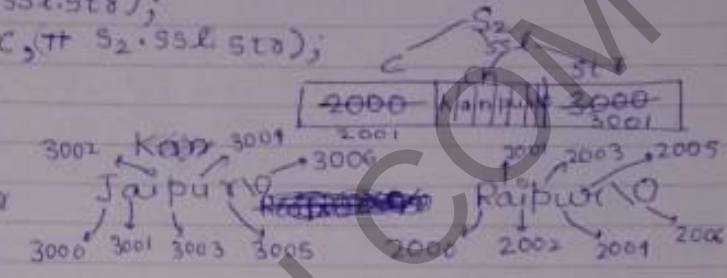
Note:- If we use register instead of static, then r is stored in register for faster access

---

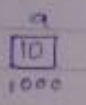
```
#include<stdio.h>
main()
{ struct a
    {char c[7];
     char *str;
    };
  Struct b
```

```
{ char *c;
  Struct a SS1;
};
Struct b S2 = {"Raipur", "Kanpur", "Jaipur"};
printf("%s%s", S2.C, S2.SS1.str);
printf("%s%s", ++S2.C, ++S2.SS1.str);
}
```

O/P:-
Raipur Jaipur
aipur aipur

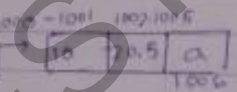


* int a=10;
  int *b=a; //no error



Stoucture & Union

```
Struct *S1
{ int a;
  float b;
  char c;
};
Struct *S1 x;
x = {10, 20.5, 'a'};
```



Note:- Whenever we reference a struct, many variables are referenced, hence selection is req. to select the intended variable using '.'

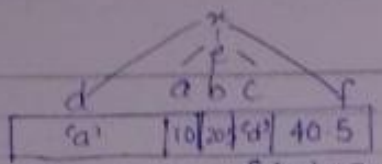
* printf(x) → Garbage. Nothing first element of the struct will be printed.
* printf(x.a) → 10
* printf(x.b) → 20.5
* printf(x.c) → a
* if Stoucl S2
  { int a;
  }

then S2.a not required S1 will also point a.

Both are possible

```
Struct S1
{ int a;
  float b;
  char c;
};
```
→ 7B

```
Struct S2
{ char d,
  Struct S1 e; ← we can declare struct within struct.
  float f
};
```
→ 12B



struct s_2 = {'a', 10, 20.5, 'd', 40.5}

struct x = {'a', 10, 20.5, 'd', 40.5}

x.d = 'a'

x.e.a = 10

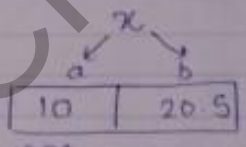
x.e.b = 20.5

x.e.c = 'd'

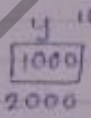
x.f = 40.5

Q
```
struct S
{int a;
 float b;
};
```

1. struct s, x = {10, 20.5};



2. struct s, *y = &x;



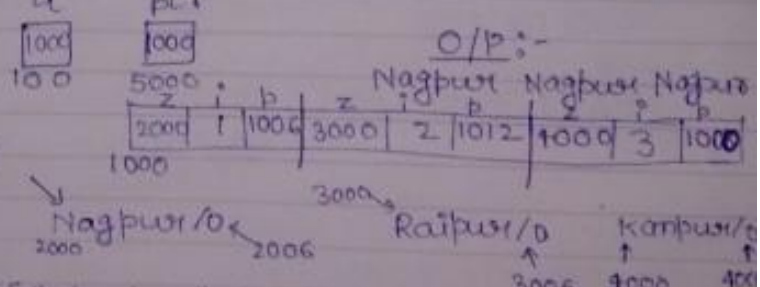
3. printf(y) → 1000

printf((*y).a); → 10

(*y): 1000

or printf(y->a); → 10

Q
```
main()
{struct S
 {char *z;
  int i;
  struct S, *b;
 };
struct S, a[] = {{"Nagpur", 1, a+1},
                 {"Raipur", 2, a+2},
                 {"Kanpur", 3, a}};
struct S, *ptr = a;
printf("%s%s%s%s", a[0].z, ptr->z, a[2].b->z);
```



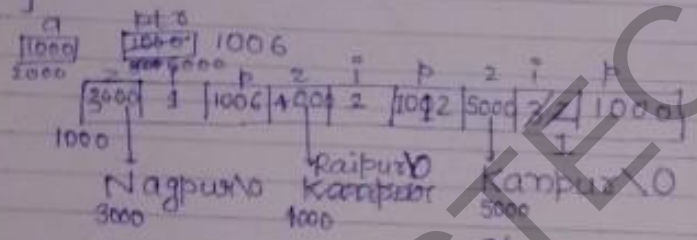
O/P:- Nagpur Nagpur Nagpur

(*ptr).z a[2].(*b).z

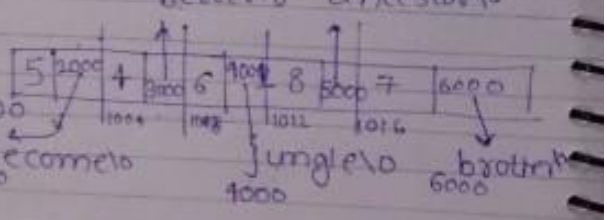
A preprocessor will change p->c into (*p).c

```
main()
{ struct s1
  { char* z;
    int i;
    struct s1 *p;
  };
struct s1 a[] = { {"Nagpur", 1, a+1}, {"Raipur", 2, a+2},
                  {"kanpur", 3, a} };
struct s1 * ptr = a;
printf("%s", ++(ptr->z));
printf("%s", a[(++ptr)->i].z);
printf("%s", a[--(ptr->p->i)].z);
printf("%d", -a[2].i);
}
```

O/P:-
agpur
~~Kanpur~~
kanpur
kanpur
-2 1



Q. #include<stdio.h>
```
struct Test
{ int i;
  char *c;
};
struct Test st[] = {5, "become", 4, "better", 6, "jungle",
                    8, "ancestor", 7, "brother"};
main()
{ struct Test *p = st;
  p = p+1;
  printf("%s", ++(p++->c));
  printf("%c", *++p->c);
  printf("%d", p[0].i);
  printf("%s", p->c);
}
```



*(*p).c ++

O/P:-
etter, u, 6, ungle

p[0] = *(p+0)

Q. void main()
{ char S[] = "hello";
  char R[] = "hello";
  if (S == R)
    printf("equal");
  else
    printf("not equal");
}

O/P:-
Not equal
because S & R both stores diff. base addresses.

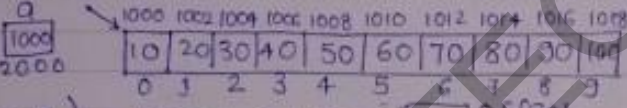
* if *S == *R ('h' == 'h')
  O/P:- equal

*

Arrays

1-D (1-Dimensional)

int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};



a+5 = 1000 + 2*5
= 1010

L(a[5]) = 1000 + 2*(5-0) → offset
= 1010

L(a[9]) = 1000 + 2*(9-0)
= 1018

in C compiler it is like 1000 + 2*5 (so C compiler saves '1' subtraction)

This means before 5th element, how many elements are there.

if array starts with index '1'
L(a[5]) = 1000 + 2*(5-1) → offset
= 1008
L(a[9]) = 1000 + 2*(9-1)
= 1016

Q Array A is defined as follows:-
- A[-5 ... +5]
- Base address (BA) = 999
- Size of element = 100B

Total no. of elements:-
Last index - 1st index + 1

Location of a[-1], a[+5]
L(a[-1]) = 999 + 100 (-1 - (-5)) = 999 + 100 × 4 = 1399
L(a[5]) = 999 + 100 × (5 - (-5)) = 999 + 100 × 10 = 1999

In General :-

- A[lb .. ub]
- BA → Base Address
- C → Size of 1 element

then Location of i^{th} element :-

$$Loc(a[i]) = B.A + C \times (i - lb)$$

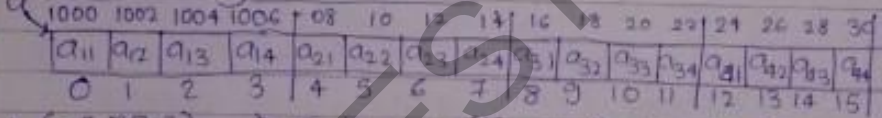
Two-Dimensional Arrays

int a[4][4];

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

* Every multidimensional array is stored in the form of multidimensional or 1-dimensional array.

Row-Major Order :-



$$L(a[i][j]) = B.A + C * ((i - lb_2) * (i - lb_1))$$

$$L(a[1][3]) = 1000 + (4(4-1) + (3-1)) * 2$$
$$= 1000 + 28$$
$$= 1028$$

$$Loc(a[2][4]) = 1000 + (4(2-1) + (4-1)) * 2$$
$$= 1000 + 7 * 2$$
$$= 1014$$

Total Rows
Last Row No. − 1st Row No. + 1

Total Columns
Last Column No. − 1st Column No. + 1

→ array starting from A[1][1]

Q Consider the following 2-D array :-

a[5 25, 50 75]
B.A = 0
C = 5B

$L(a[20][70]) = 0 + ((70-50)(20-5)\times(75-50+1) + (70-50))\times 5$
$= (15\times 26 + 20)\times 5$
$= (390+20)\times 5$
$= 410\times 5 = 2050$

$Loc(a[25][75]) = 0 + ((25-5)\times(75-50+1) + (75-50))\times 5$
$= (20\times 26 + 25)\times 5$
$= 545\times 5$
$= 2725$

In General (Row Major Order)

A[lb_1 ub_1, lb_2 ub_2]
BA, C

$$Loc(a[i][j]) = BA + ((i - lb_1)\times(ub_2 - lb_2 + 1) + (j - lb_2))\times C$$

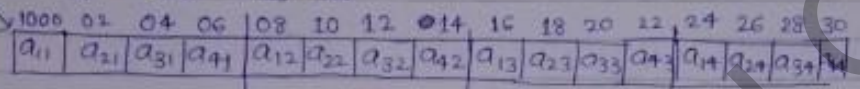
Date

11.08.12

- int a[4][4];

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$$

a



a[lb1 ub1][lb2 ub2]

$$Loc(a[i][j]) = B.A. + \left((j - lb_2) \times (ub_1 - lb_1 + 1) + (i - lb_1)\right) \times C$$

- $Loc(a[3][4]) = 1000 + ((4-1) \times 4 + (3-1)) \times 2$
  $= 1000 + (12 + 2) \times 2$
  $= 1000 + 28$
  $= 1028$

- $Loc(a[4][3]) = 1000 + ((3-1) \times 4 + (4-1)) \times 2$
  $= 1000 + (8 + 3) \times 2$
  $= 1000 + 22$
  $= 1022$

- $Loc(a[4][4]) = 1000 + ((4-1) \times 4 + (4-1)) \times 2$
  $= 1000 + 6 = 1006$

Q. Consider the array (using Column Major Order)
- A[-5 ... +30][25 75]
- BA = 999
- C = 10B

$$Loc(A[0][74]) = 999 + ((74-25) \times (30-(-5)+1) + (0-(-5))) \times 10$$
$$= 999 + (49 \times 26 + 5) \times 10$$
$$= 999 + 12790$$
$$= 13789$$

$$Loc(A[-2][30]) = 999 + ((30-25) \times 26 + (-2+5)) \times 10$$
$$= 999 + (130 + 3) \times 10$$
$$= 999 + 1330 = 2329$$

Lower Triangular Matrix

int a[4][4]

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

* it is valid only for Square matrices.

A[i][j] = 0 if i < j

Storing using Row Major :-

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	a_{41}	a_{42}	a_{43}	a_{44}
1st	2nd		3rd			4th			

LOC(a[4][3]) = 1000 + ((1+2+3) + (4-1) + (3-1)) × 2

= 1000 + 18

= 1000 + ((4-1)(1+2+3) + (3-1)) × 2

= 1000 + (18) × 2

= 1016

$\frac{(i-lb_1)(i-lb_1+1)}{2}$

$$LOC(a[i][j]) = B.A. + \left((i - lb_1) NNS + (j - lb_2) \right) \times C$$

$(i-lb_1)$ → Natural No Sums upto from 1 to $(i-lb_1)$

(4-1) sum of natural nos. (i.e. sum of natural nos. upto 3.)

LOC(a[4][2]) = 1000 + ((1+2+3) + (2-1)) × 2

= 1000 + (6+1) × 2 = 1014

Q Consider matrix defined as follows:

A[-25...+25, -25...+25]

BA = 0

$\frac{5 \times 3}{2} = 15$

Row Major Order, lower Triangular Matrix

C = 100 B

• LOC(A[-20][-21]) = 0 + ((-20+25) NNS + (-21+25)) × 100

= 0 + ((1+2+3+4+5) + 4) × 100

= 0 + (19) × 100

= 1900

• Loc(A[0][0]) = 0 + ((0+25)NNS + (0+25)) × 100
= 0 + ($\frac{25 \times 26}{2}$ + 25) × 100 = 35,000

Column Major Order :-

00	02	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{31}	a_{41}	a_{22}	a_{32}	a_{42}	a_{33}	a_{43}	a_{44}

My answer
↳ Loc(a[i][j]) = B.A + ((j − lb$_2$) reverse order sum + (i − lb$_1$)) × C

Loc (a[4][3]) = 1000 + (((4+3) + (4−3)) × 2
= 1000 + (7 + 1) × 2
= 1030 ~~1020~~ 1016

★
Or Loc(a[4][3]) = 1000 + ($\frac{4 \times 5}{2}$ − (4−3+1)NNS + (4−3)) × 2

skip all columns → we actually want to go to 3rd column

a_{11}	a_{12}	0	0
a_{21}	a_{22}	0	0
a_{31}	a_{32}	a_{33}	0
a_{41}	a_{42}	a_{43}	a_{44}

we are at here (at the end)

we are at here (actually after a_{42}) [after subtracting]

= 1000 + (10 − (2+1) + 1) × 2 $\quad \frac{4 \times 5}{2}$
= (1000 + 8) × 2
= 1016

Loc (a[4][1]) = 1000 + ($\frac{4 \times 5}{2}$ − (4−1+1)NNS + (4−1)) × 2
= 1000 + (10 − 10 + 3) × 2
= 1000 + 6 = 1006

int * a = 2;
Pf(*a);

A[lb1, ub1, lb2, ub2]

skip all columns ↓ go to the starting of the desired column ↓

$$LOC(A[i][j]) = B.A + \left((ub_2 - lb_2)^{NNS} - (ub_2 - j + 1)NNS + (i - j) \right) \times C$$

↑ go to the desired row

Q. A[-5 ... +5, -5 ... +5]
BA = 1000, C = 10
Lower Triangular Matrix
Column Major Order

66 - 21 = 45

$\frac{7 \times 8}{2} = 28$

- $LOC(A[0][0]) = 1000 + \left(11^{NNS} - (5 - 0 + 1)NNS + 1 \right)$
$= 1000 + \left(\frac{11 \times 12}{2} - \frac{6 \times 7}{2} + (0 - 0) \right) \times 10$
$= 1000 + 45 \times 10$
$= 1450$

- $LOC(a[2][-1]) = 1000 + \left(11NNS - (5 - (-1) + 1)NNS + (2 + 1) \right) \times 10$
$= 1000 + (66 - 28 + 3) \times 10$
$= 1000 + (41) \times 10$
$= 1410$

Tri-diagonal Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{42} & a_{44} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{42} & a_{44} \end{bmatrix}$$

N-1 + N + N-1 = 3N-2 memory space.

★ In Tridiagonal Matrix, apart from 1st & last row (which has 2 elements), rest all other rows have 3 elements.

Using Row Major :-

a

a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	a_{32}	a_{33}	a_{34}	a_{43}	a_{44}
1000	02	04	06	08	10	12	14	16	18

$Loc(a[3][4]) = 1000 + ((3(2+3) + (4-2)) \times 2$
$= 1000 + (5+2) \times 2$
$= 1014$

(assuming each row contains 3 elements; skipping desired no of rows; but 1st row contains only 2 elements, subtracting 1 element; going to the desired column.)

$$Loc(a[i][j]) = B.A + (3 \times (j - lb_1) - 1 + (j - i + 1)) \times C$$

$Loc(a[4][4]) = 1000 + (3 \times (4-1) - 1 + (4-4+1)) \times 2$
$= 1000 + 9 \times 2 = 1018$

Using Column Major

$$Loc(A[i][j]) = B.A. + (3 \times (j - lb_2) - 1 + (i - j + 1)) \times C$$

$Loc(A[3][4]) = 1000 + (3 \times (4-1) - 1 + (3-4+1)) \times 2$
$= 1000 + (8) \times 2$
$= 1016$

Using Diagonal by Diagonal :- (Principal diagonal, Lower, Upper)

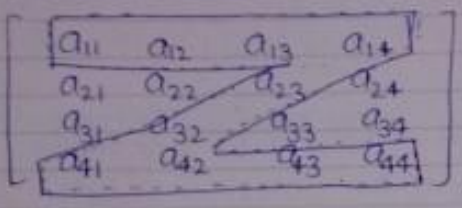
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \rightarrow$$

principal				Lower			Upper		
a_{11}	a_{22}	a_{33}	a_{44}	a_{21}	a_{32}	a_{43}	a_{12}	a_{23}	a_{34}

$Loc(A[i][j]) \rightarrow$ if $(i == j)$ (Principal diagonal)
$= B.A + (j - lb_1) \times C$ [if $i == j$]
$= B.A + ((\underbrace{ub_2 - lb_2 + 1}_{N(ub_2 - lb_2 + 1 = N)}) + j - lb_2) \times C$ [if $i > j$]

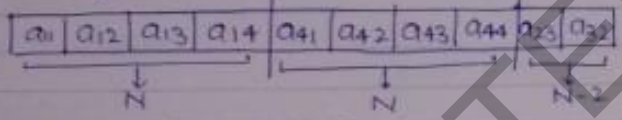
$$= \boxed{BA + (N + (N-1) + (i - lb_1)) \times C \quad (\text{if } i<j)}$$

Z-Matrix



Storing :-
1. Row Major
2. Column Major
3. Store 1st row, last row & remaining elements in diagonal

Row-Maj
Using 3rd approach -

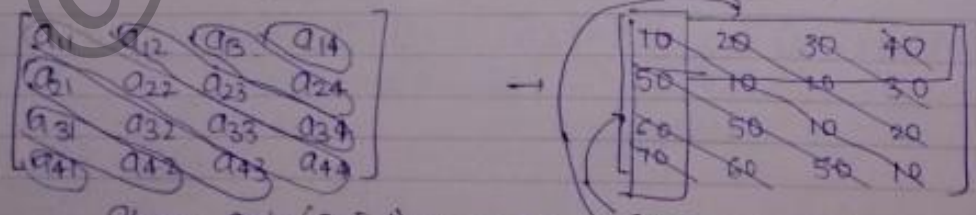


∴ total space = 3N-2

$$Loc(A[i][j]) = \begin{cases} BA + ((j - lb_2)) \times C & [\text{if } i == lb_1] \\ BA + (N + (j - lb_2)) \times C & [\text{if } i == ub_1] \\ BA + (2N + (i - lb_1 - 1)) \times C & \left[\begin{array}{l}\text{if } j \neq lb_1 \text{ \&} \\ j \neq ub_1 \text{ \&} \\ (i+j) = N+1\end{array}\right] \end{cases}$$

Toeplitz Matrix

if $A[i,j] = A[i-1, j-1]$ $\forall i, j$ where $\boxed{i>1 \text{ \& } j>1}$



Store only (2N-1) elements Store only these

Storing:-

① Row - Order
② Column - Order
③ Store 1st row, remaining element in 1st column

$$Loc(a[i][j]) = \begin{cases} B.A + (j - lb_2) \times C_o & (\text{if } i \le j) \\ B.A + (N + (i - lb_1 - 1)) \times C & (\text{if } i > j) \end{cases}$$

[Elements belongs to upper part.]

$N = (ub_2 - lb_2 + 1)$

(elements belongs to lower part)

Three-Dimensional Matrix

int a[2][3][4] → 2 arrays of size 3×4

BA = 1000

Row Major Order

$Loc(A[2][3][4]) = 1000 + ((2-1) \times 12 + (3-1) \times 4 + (4-1)) \times 2$

$= 1000 + (12 + 8 + 3) \times 2$

$= 1000 + 23 \times 10 = 1230$

Loc (A)

int a[-5 .. +5][-10 .. +50][-25 .. +25]

- BA = 0
- Column Major Order
- C = 5

$Loc(A[0][0][0]) = 0 + (5 \times 61 \times 51 + (0+25) \times (61) + (0+10)) \times 5$

$= (3111 + 1525 + 10) \times 5$

$= 85450$

3111 × 5 = 15555 → 1535 → 17090

61 × 25 = 1525

61 × 51 = 3111

17090 × 5 = 85450

$A[lb_1 \ldots ub_1][lb_2 \ldots ub_2][lb_3 \ldots ub_3]$

Row Major Order

$$LOC(a[i][j][k]) = B.A + \left(\left((i - lb_1) \times (ub_2 - lb_2 + 1)(ub_3 - lb_3 + 1) + (j - lb_2) \times (ub_3 - lb_3 + 1) + (k - lb_3)\right) \times C\right)$$

Column Major Order

$$Loc(a[i][j][k]) = B.A. + \left(\left((i - lb_1) \times (ub_2 - lb_2 + 1)(ub_3 - lb_3 + 1) + (k - lb_3) \times (ub_2 - lb_2 + 1) + (j - lb_2)\right) \times C\right)$$

Q. `int a[5] = {10,20,30,40,50};` (specifying this is optional)

`int a[][3] = {10,20,30,40,50,60};` (mandatory)

`int a[][2][3]` (optional, mandatory)

`int a[5];` → garbage values

`int a[5] = {10,20,30};` → 4th & 5th elements are 0.

`int a[5] = {};` → all are 0.

`int a[5] = {10,20,30,40,50,60};` → no compilation error, but a[5] is not accessible

Note:-
In multidimensional Array, M-1 indices are mandatory.

Q.
```
main()
{ int a[3][3];
  printf("%d", ((a==*a) && (*a == a[0])));
}
```
O/P:- 1

a = 1000, *a = 1000, a[0] = *(a+0) = 1000

1000	02	04
10	20	30
06	08	10
40	50	60
12	14	16
70	80	90

a = 1000
a+0 = 1000 ⇒ *(a+0) = *1000 = 1000
a+1 = 1006
a+2 = 1012
*(a+1) = *1006 = 1006
*(a+2) = *1012 = 1012

row is not selected

row is selected. *a+1 = a[1]

row is not selected (a+1) = 1006
& hence we can change the row. (a+1)+1 = 1012

a = 1000
a+1 = 1006
*(a+1) = 1006 → row is selected (we can't change the row now.)
*(a+1)+1 = 1008

```
a[i][j]
↓
b
↓
b[j]
↓
*(b+j)
↓
*(a[i]+j)
↓
*(*(a+i)+j)
```

- a = 1000
- a+1 = 1006 → row is not selected
- *(a+1) = 1006 → row is selected
- *(a+1)+1 = 1008
- *(*(a+1)+1) = 50
- *(*(a+1)+1)+1 = 51

Q. main()
{ int a[2][3][2] = {2,4,7,8,3,4,2,2,2,3,3,4};
printf("%u %u %u %d", a, *a, **a, ***a);
printf("%u %u %u %d", a+1, *a+1, **a+1, ***a+1);
printf("%u %u %u %d", *(*(a+2)+3), *(**a[1]+2), *(a+2), *(*(a[1]+1)));
printf("%u %u %u %d", a+1, a[1]+1, a[1][2]+1, a[1][2][3]+1);
}

*(a+1)+1 *(*(a+1)+2)+1 *(*(*a+1)+2)+1

a[2][3][2] :

a = 1000

2	4	7	8	3	4	2	2	2	3	3	4
00	02	04	06	08	10	12	14	16	18	20	22

*a → array selected
**a → array & row selected
***a → array, row & column selected

O/P :-

1000	1000	1000	2
1012	1004	1002	3
1036	2	1024	2
1012	1016	1022	garbage

3rd printf :-

*(*a[1]+2)
*(**(a+1)+2)

*a → 0th array base address
**a → 0th array 0th row base address

1012 → array selected

((a+2)+3)
1024 → array selected
row → 1024 + 3×2 = 1036 (3rd row)
row selected

**(a+1)+2
1012 + 2×2 = 1016
*(1016) → column selected

row selected

((a+2))
1024 → array selected

((a[1]+1))
((*(a+1)+1))
1012 → array selected
skip 1 row = 1012 + 4 = 1016 → row selected, column selected

12 × 7
= 84

Q *((*((a+5)+2)+2)+1)

1000

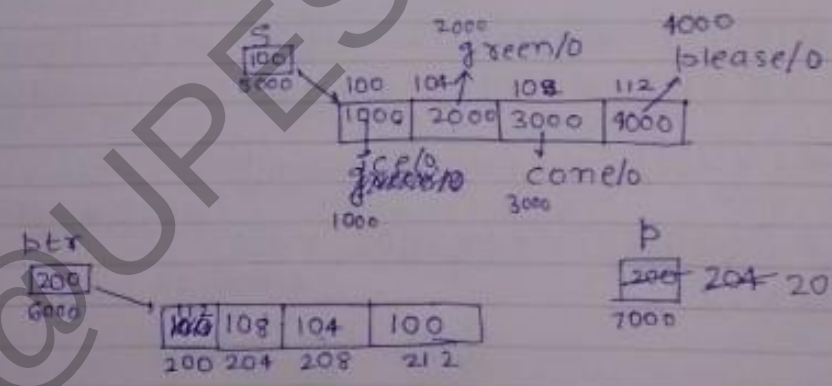
1084

array selected

skip 2 rows: 1084 + 2 × 4
= 1084 + 8
= 1092

skip 1 more row: 1092 + 1 × 4
= 1096

```
#include<stdio.h>
Q main()
{ char *s[] = {"ice","green","cone","please"};
  char **ptr[] = {s+3,s+2,s+1,s};
  char ***p = ptr;
  printf("%s", **++p);
  printf("%s", *--*++p+1);
}
```



O/p:-

p = 200
++p = 204
*++p = 3000
**++p = cone

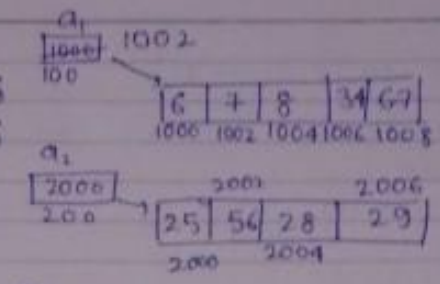
--++p + 1

204
208
104
100
1000
1001

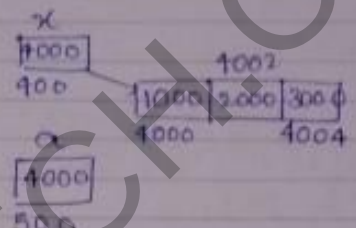
∴ O/p:- cone

@UPESTECH.COM

Q
```
#include<stdio.h>
int a1[] = {6,7,8,34,67};
int a2[] = {25,56,28,29};
int a3[] = {-12,27,-31};
int *x[] = {a1,a2,a3};

main()
{ B(x);
}
B(int **a)
{ printf("%d",a[0][2]);
  printf("%d",*a[2]);
  printf("%d",***++a);
  printf("%d",*(++a)[0]);
}
```



O/P :-
8
-12
7
25



O/P :-

1st printf :- a[0][2] = *(*(a+0)+2)

1st printf :- a[0][2] = *(*(a+0)+2) = *(1000+2×2) = *(1004)

2nd printf :- *a[2] = **(a+2) = **(1000+2×2) = **(4004) = *(3000)

3rd printf :- ***++a[0] = ***++*(a+0) = ***(4000) = **++(1000) = *(1002)

4th printf :- *(++a)[0] = *(++a)[0] = *a[0] = **(a+0) = **(4002) = *(2000) = 25

(++a)[0] ⇓ *(++a+0)

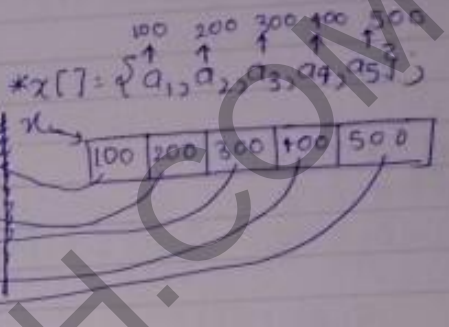
a: 4002, 500

even though
1-D pointer array
x[1][2] :- *(*(x+1)+2) = 28

★ using one dimensional array of pointer, we can get two-dimensional array.

int a[ ]={6

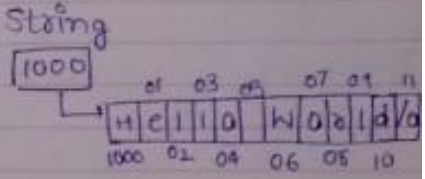
* int a[][5] → 2-D array, where every raw contain uniform no. of columns.
* int *x[ ] → 2-D array, but it will create variable columns in every row.



* int a1[]={6,7,8,34,67};
  int a2[]={25,56,28,29};
  int a3[]={-12,27,-31};
  int a4[]={10,20};
  int a5[]={20};

Q. consider following C-declarations:-
(1) int A[10][10];
(2) int *B[10];
Which is true?
(a) A[5][6]=10; T
(b) A[5]=1000 F (we cant change the address of 5th row of an array.)
(c) B[5][6]=10; T
(d) B[5]=1000 T

Q.
```
main()
{ char string[]="Hello World";
  display(string);
}
void display(char *string)
{ printf("%s", string);
}
```



* In the above program, we are using display fn. W/o declaring so compiler assumes display fn by default integer value, but above program is returning void, so above program will give error of type mismatch.

Q. double foo(double); /* line 1*/
int main()
{ double da, db;
db = foo(da);
return 0;
}
double foo(double a)
{ return a;
}

* if we delete line 1 then conflicting datatype for fn. foo. type mismatch

* In switch statement, we can write default wherever we want, but it will be executed whenever there is no match.

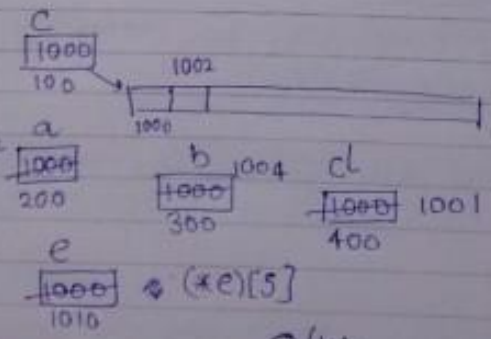
Q. void main()
{ int i=0;
for(int i=0; i<20; i++)
{
switch(i)
{ case 0: i+=5; // we can
case 1: i+=2; put default
case 5: i+=5; anywhere, it
default: i+=4; will have
break; same result.
}
printf("%d", i); // default is at *
}
}

O/P:-

i = i+5

5
16
27
21

Q. main()
{ int c[100];
int *a = (int *)c;
float *b = (float *)c;
char *d = (char *)c;
int (*e)[5] = (int *)c;
for(i=0 to 5)
{ printf(c+i, a, b, d, e);
a++;
b++;
d++;
c++;
}
}



O/P:-

i					
0	1000	1000	1000	1000	1000
1	1002	1002	1004	1001	1010
2	1004	1004	1008	1002	1020
3	1006	1006	1012	1003	1030
4	1008	1008	1016	1004	1040
5	1010	1010	1020	1005	1050

★ int (*a)[5] ⟶ a is a pointer to array which contains 5 elements,
∴ when we do a++, then a is incremented to 10B

Function returning pointer :-

```
int fun();
main()
{int x=10;
 int *b;
 b=fun();
 printf("%d",x+(*b));
}
```

x: 10 (1000) b: 3009 (2000)

```
int *fun()
{ int y=20;
  return (&y);
}
```

y: 20 (3000)

★ function will get the memory when the fn is called, & will be deallocated when the fn. returns

O/P :-
10+20=30

★ fun is a function which returns pointer.

Functional Pointer

```
main()
{ int display(); //creating memory for display fn (base address is allocated)
  int (*funptr)();
  funptr=display;
  (*funptr)();
}
int display()
{ printf("Hi");
  return(0);
}
```

1000 → value at 1000 (at 1000, it is the fn. display)

funptr is a pointer, which internally contains function's base address.

funptr	display
1000	1000
2000	3000

O/P :-
Hi

★ function name indicates base address
∴ display indicates its base address

★ int (*p)() → p is pointer pointing to a function.
int *p[5] → p is an array of pointers pointing to 5 addresses of int variables.
int *p[5]() → p is an array of pointers pointing to 5 fn which are returning integer

Q. main()
{ int (*ptr[3])();
ptr[0] = aaa;
ptr[1] = bbb;
ptr[2] = ccc;
ptr[2](); or (*ptr[2])()
}

1000 int aaa()
{ pf("aaa");
return 0;
}

2000 int bbb()
{ pf("bbb");
}

3000 int ccc()
{ pf("ccc");
}

ptr
4000 4002 4004
| 1000 | 2000 | 3000 |
0 1 2

*ptr[0] = **(ptr+0) ptr[0] = *(ptr+0)
= **(1000) = *(4000)
• value at address 4000

(*ptr[2])()
↓
(**(ptr+2))()
↓
(**(4004))()
↓
(*(3000))()
↓
at ccc fn.

Q. What does the following C-statement indicate
int (*f)(int, int);
★ f is a pointer contains address of a function which is taking two int variables as argument & returning an integer

• int *(*f)(int *, int *);
f is a pointer containing address of a function which takes 2 integer pointers as arguments & returns integer pointer.

② void (*abc)(int, void (*def)());
abc is a pointer containing address of a function which takes 2 argument as i/p, out of which ① one is integer & the other is ② def is a pointer pointing to a fn which returns void & returns void.

③ char * (*(*a[N])())()

- a[N] → array of N pointers
- pointing to function(s) [doesn't take any input]
- these function(s) are returning pointer(s) (pointer type is not specified)
- these returned pointers are pointing to a function
- this function is returning character pointer.

*Array of N pointers pointing to functions which will return a pointer which is pointing to a function which is returning character pointer.

④ (char * (*)()) (*ptr[N])();

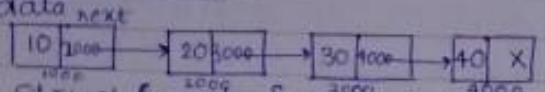
(i) a pointer is pointing to a function
(ii) the function returns a character pointer.
(iii) character pointer is pointing to an array of pointers which are pointing to functions which are returning a pointer.

brackets are having associativity from left to right

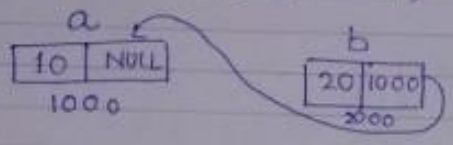
Date

12.08.12

Linked List :-




```
struct node {
  int data;
  struct node *next;
};
struct node a={10, NULL};
```




```
struct node b={20,&a};
struct node *ptr=&b;
```

Self referential data structure

- float type pointer is pointing to float.
- int type pointer is pointing to int.

ptr = 2000

*ptr :- inside 2000 (i.e. selected the address 2000).

(*ptr).next = 1000

*(*ptr.next) :- selected the address 1000

(*(*ptr)next)).data = 10

(*(ptr->next)).data

(ptr->next)->data = 10

Q. Write a C program to print all data fields parts in every node of the linked list.

Ans.

```
#include <stdio.h>
        struct node {int data; struct node *next;};
struct
main()
{ struct node *start= //starting address of linked list.
  while(
  ptr = start;
  while (ptr != NULL)
    {printf("%d", ptr->data);
     ptr = (*ptr).next;
    }
}
```

```
int a
int *ptr = &a
ptr = (int) a;
```

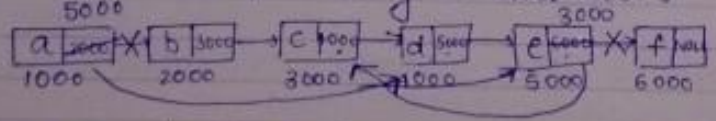
*if S is NULL, then *s is not allowed (it gives segmentation error)

Q. Write a C program to return data of the last node of linked list

Ans.
```
ptr = start;
while (ptr != NULL)                    // Linked list contain atleast 1 node.
  { if ((*ptr).next == NULL)
      { printf("%d", (*ptr).data);
      }
    ptr = (*ptr).next;
  }
```

*
```
struct node {
  int data;
  struct node *next;
};
int Display(struct node *s)
  { struct node *p = s;
    if (p == NULL)
      { printf("Linked list is empty");
        exit(0);
      }
    while (p->next != NULL)    // Linked list contain atleast 2 elements
      { p = p->next;
      }
    printf("%d", p->data);
  }
```

Q. Consider the following Linked List :-



What would be the O/P after executing following sequence of statements :-

int *b, a;
p=&a;

p
3000

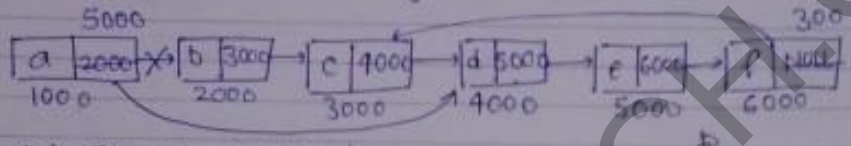
first
1000

```
struct node *p;
p = first->next->next;
first->next = p->next->next;
p->next->next->next = p;
printf(first->next->next->next->data);
```

O/P:-

a

2. Consider the following linked list -



(i) struct node *p

(ii) p=first->next->next

(iii) first->next=p->next->next;

(iv) p->next->next->next->next=p;

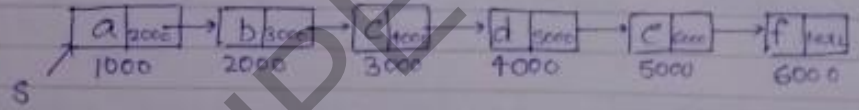
(v) printf(first->next->next->next->data);

p
3000
7000

O/P:-

c

Adding Node in Linked List



(i) Starting of the linked list - Start

```
Add at start (struct node *s, int x)
{ struct node *ptr = (node *)malloc(sizeof(node));
  (*ptr).next = s;
  (*ptr).data = 'g';
  s = ptr;
}
```

(ii) Write a C program to add a node with data x at the end of the linked list.

```
Addatend (node * start)
{ node * ptr = (node *) malloc (sizeof (node));
 (*ptr).data = x;
  node * temp = start;
  While (ptr->ne                if (start == NULL)
                                { start = ptr; ptr->next = NULL; exit(0);}
  While (temp->next != NULL)
    { temp = temp->next;
    }
  temp->next = ptr;
  ptr->next = NULL;
}
```

(iii) Write a C program to add a node with data x, after a node with data y.

→
```
Addinbtw(node *start)
{node *ptr = (node *)malloc(sizeof(node));
 (*ptr).data = x;
  node * temp = start;
    While (temp->data != y)
      { temp = temp->next; if (temp==NULL) if (temp == NULL)
      }                                      {exit(0);}
  temp if (temp==NULL) {exit(0);}
   ptr->next = temp->next;
    temp->next = ptr;
}
```

(iv) Write a C program to add a node with data x before a node with data y.

```
Addbefore (node * start)
{node *ptr = (node *)malloc (size of (node));
 (*ptr).data = x;
  node * temp = start;
  node * tempbef = start;
  While (temp -> data != y)
    { tempbef = start; temp;
      temp = temp->next;
    } if (temp == NULL) exit(0);
```

```
if (tempbef==start)
   { ptr->next=start;
     start=ptr; exit(0);
   }
temp->next=te
ptr->next = tempbef->next;
tempbef->next=ptr;
}
Or
Addbef (struct node *start)
{ node *p= (node*) malloc(sizeof(node));
  p->data=x;
  node * q=start;
  if (q==NULL)
    {return(1);
    }
  if (q->data=='y')
    {p->next=q;
     start=p;
    }
  while(q->next!=NULL && (q->next)->data!='y')
    { q=q->next;
    }
  if(q->next==NULL)
    {exit(1);
    }
  p->next=q->next;
     q->next=p;
}
```

Q Which one of the following statement, adding an element x after current position

a. current = Newnode(x, current)
b. current = Newnode(x, current->next);
c. current->next = Newnode(x, current);
d. current->next = Newnode(x, current->next);

Q Consider the fn. f defined below:-

```
struct item
{int data;
 struct item *next;
};
int f(struct item *p)
 {
return((p==NULL)||(p->next==NULL)||((p->data<=p->next->data)
                 && f(p->next)));
}
```

for a given LL, the fn. f returns 1 iff (indirectly means if LL has 0 or 1 element then it is also in non-increasing order.)

(a) if LL is empty or exactly one element
(b) elements in the LL are sorted in non-decreasing order.
(c) " " " " " " " " -increasing "
(d) none of the above.

Q Write a C program to delete a node at the begining of the linked list.

```
del(node *start)
  { node * ptr = start;
    start = (*start).next;
    free(ptr); // the memory is deallocated pointed by
  }            // ptr, now *ptr is dangling pointer, so to avoid
    ptr=NULL; // any segmentation errors, we perform ptr=NULL.
  }
```

* if we write int a, then we can't deallocate the memory using free()

Q Write a C program to delete a node at the end of LL.

```
delatEnd(node *start)
 {node * ptr = start; prev = start;
  if(ptr==NULL)
    {exit(1);
    }
```

```
while(ptr->next!=NULL)
  {prev=ptr;
   ptr=ptr->next;
  }
if(prev==start)
  {free(prev);
   start=prev=ptr=NULL;
  }
else
  {prev->next=NULL;
   free(ptr);
   ptr=NULL;
  }
}
```

2. Write a C program to delete a node before which contains data x.

```
del(struct node * start)
 {node *ptr=start, *prev=start;
  if(ptr==NULL)
    {exit(1);
    }
  while(ptr->data!='x'&& ptr->next!=NULL)
    {prev=ptr;
     ptr=ptr->next;
    }  if(ptr==NULL){exit(1);}
  if(prev==ptr==start)
   {start=start->next;
    free(ptr);
    ptr=NULL;
   }
  else
   {prev->next=ptr->next;
    free(ptr);
   }ptr=NULL;
}
```

Q Write a C program to delete a node before x.

Using 3 pointers

```
del(node * start)
{node *ptr=start,*prev=start,prevtwobef=start;
 if (ptr==NULL)
   {exit(1);
   }
 while(
 if (start->data=='x')
  {exit(0);
  }
 if (start->next->data=='x')
  { start=start->next;
    free(ptr);
    ptr=prev=prevtwobef=NULL;
  }
 while(
 ptr=(start->next)->next;
 prev=start->next;
 prevtwobef=start;
 while (ptr->data!='x' && ptr->next!=NULL)
   { prevtwobef=prev;
     prev=ptr;
     ptr=ptr->next;
   }
 if (ptr->next==NULL && ptr->data!='x')
   {exit(1);
   }
 prevtwobef->next=prev->next;
 free(prev);
 prevtwobef=prev=ptr=NULL;
}
```

Or

Using 2 pointers

```
del(node * start)
 {node * ptr=start,*prev=start;
   if (start==NULL)
     {exit(1);
     }
   if (start->data=='x')
     {exit(1);
     }
   if (start->next->data=='x')
     {start = start->next;
      prev=ptr=NULL;
      exit(0);
     }
   while( ptr->next
   ptr= start->next->next;
   prev= start->next;
   while((ptr->next)->data!='x' && (ptr->next)!=NULL)
     {prev=ptr;
      ptr=ptr->next;
     }
   if((ptr->next)==NULL && ptr->data!='x')
     {exit(1);
     }
   if(prev->next=ptr->next;
      free(ptr);
      ptr=prev=NULL;
 }
```

Using 1 pointer

```
del(node * start)
 {node *ptr= start;
   if (start==NULL)
     {exit(1);
     }
   if (start->data=='x')
     {exit(1);
     }
```

Note :- In order to delete any node in the given linked list apart from last node two pointers (minimum 2 pointers) are required.

Q Write a program to find middle of the linked list

Using 2 loops

```
find(node *start)
{ node * ptr= start;
  int i=0;
  while (ptr->next!=NULL)
    {itt;
     ptr=ptr->next;
    }
  if (ptr==NULL)
    {exit(1);
    }
  if (ptr== start)
    {return (ptr);
    }
  if(i%2==0)
    {i=i/2;
    }
  else
    {i= i/2+1;
    }   ptr = start;
  for (int j=1; j<=i;j++)
    {ptr=ptr->next;
    }
}
```

O(n)

Using 1 loop

```
struct node
find(node * start)
 { node * ptr= start, * ptrmid = start;
   While (ptr->next != NULL) && (ptr->next)->next != NULL)
     { ptrmid = ptrmid->next;
       ptr = (ptr->next)->next;
     } return (ptrmid);
 }
```

Θ(n)

$\frac{n}{2}$ times

*Binary Search can be applied on Linked List but it is not efficient because it will take O(n) times & $\log_2 n$ extra space.

Q. Write a C program to perform Binary search on LL.

→

```
node * BS(node * start, int x)
{ node * ptr = start;
  node * ptrmid = start, node * m;
  while(ptrmid->data
  if (start->next == NULL)
    {if (start->data == x)
       return (start);
     else
       return (NULL);
    }
  else
    {m = findmid(start);  ——→ O(n)
     if (m->data == x)    ——→ O(1)
       { return (m);      ——→
       }
     else if (m->data < x)  ——→ O(1)
       {m->next = NULL;
        BS(start);
       }
     else
       {start = m->next;
        BS(start);
       }
  }
```

Let T(n) be the time complexity of Binary Search algo on a linked list which contains n elements, then

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

$\Downarrow$

$O(n)$

$\left[\log_2 n \text{ stack space is req}\right]$

$n^{\log_b a} \quad n^{\log_2 1} = n^0 = 1$

Q Write a C program to sort the given linked list.

```
BubbleSort (node * start)
{ node *ptr = start;
  node * prev = start;

  for(int i
  int i = 1;
  while(ptr->next!=NULL)
    { ptr = ptr->next;
      i++;                    // length of linked list
    } ptr = start->next;
  for (int j=0; j<i; j++)
     { while (ptr->next!=NULL)
          { if (prev->data > ptr->data)
               { prev int temp = prev->data;
                    prev->data = ptr->data;
                    ptr->data = temp;
               }
            prev = ptr;
            ptr = ptr->next;
          } ptr = st start;
     }
}
Or
Sort (node *start)
{ node * ptr = start;
  int i = 1;
  while (ptr->next!=NULL)
    { ptr = ptr->next;
      i++;
    }
ptr = start;
for (int j=0; j<i; j++)
{
```

```
while(ptr->next!=NULL)
{
}
```

Or

```
Sort(struct node *S)
{ struct node *p,*q,
  for(p=S; p!=NULL;p=p->next)
    { if(p->data
      for(q=p->next; q!=NULL;q=q->next)
        { if(p->data > q->data)
          { swap(p->data, q->data);
          }
        }
    }
}
```

∴ $O(n^2)$

Q Write a C-program to reverse the given LL.

```
Reverse(node *start)
{ node *ptr=start; node *temp1,node *temp2;
  int i=1;          int    temp1=start;
  while(ptr)N       temp2=start->next;
  while(ptr!=NULL)
  { temp1=ptr->next;
    temp2=(ptr->next)->next;
    ptr->next=temp2;
    (ptr->next)->next=temp1;
    temp1=temp1->next;
    temp2=temp2->next;
  }
}
```

```
Struct node1
 { Struct node * addr; Struct node1 * next;
 };

Reverse (Struct node * start)
 { node * ptr = start; node1 * ptr1;
   While (ptr != NULL)
      { ptr1 = (node1 *) malloc (sizeof (node1));
        ptr1 -> addr = ptr;
        ptr = ptr -> next;
      }
   ptr = start;
```

Or

```
Reverse (Struct node *s)
   { Struct node *p, *q, *r;
       r = S; q = p = NULL;
     While (r != NULL)
        { p = q;
          q = r;
          r = r -> next;
          q -> next = p;
        } S = q;
   }
```

→ 3 pointers compulsory.

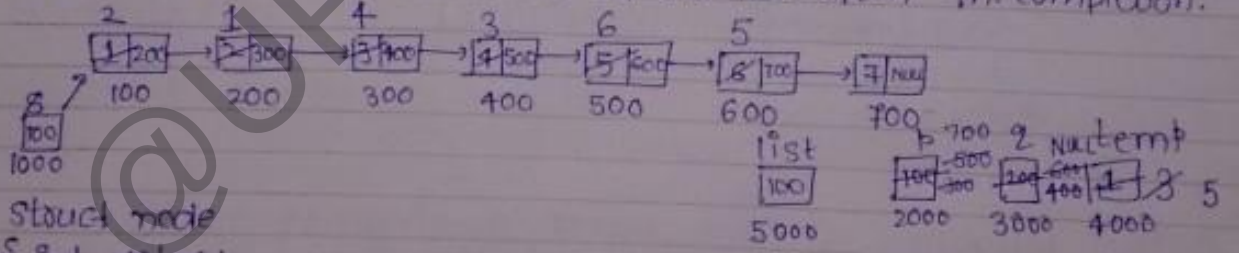
O(n)

Q The following C func. takes a single LL as i/p argument. It modified the list by moving the last node to the front of the list & returns the list. Some part of the code is left blank.

```
Typedef Struct node{
   int value;
   Struct node *next;
   }node,

node* modifiedlist(node *head)
  {node *p,*q;
   if ((head == NULL) || (head->next == NULL))
     return (head);
    q = NULL;
    p = head;
    While(p->next != NULL)
    {q = p;
     p = p->next;
    }
    p->next = head;
    q->next = NULL;        ]→ answer
    head = p;
   return (head);
  }
```

Q. The following C fn. takes a singly linked list of integers as parameter & rearranges the elements of list. The fn is called with the list containing the integers -
1,2,3,4,5,6,7

What will be the contents of the list after fn. completion.




```
Struct node
{ int value;
  struct node *next;
};
```

```
void arrange (struct node *list)
   {node *p, *q;
    int temp;
    if (!list || !list->next)
       return;
    p=list;
    q=list->next;
    while(q)
    {temp=p->value; p->value= q->value; q->value=temp;
      p=q->next;
      q=p->next;
     }
   }
```

list now :- 2, 1, 4, 3, 6, 5, 7

Q. Write a C program to find a cycle (loop) in the given linked list.

```
Struct node1
 {node * addr;
  node1 * next;
 };

findcycle (node * start)
 {node1 * start1= (node1 *)malloc (sizeof(node1));
  start1->addr= start;
  start1->next = NULL;
  node1 * prev= start1;
  node * ptr = start;
  while (ptr->next!= NULL)
    {node1 * ptr1 = (node1 *)malloc (sizeof(node1));
     prev->next = ptr1;
     prev = ptr1;
     ptr1->addr = ptr = ptr->next;
    }  ptr1->addr = ptr;
```

Or

Algorithm

① Take two pointers p_1 & p_2.
② increment p_1 by 1 & p_2 by 2.
③ Max. n loops p_1 & p_2 are same comes out to be same. When p_1 & p_2 comes out to be equal, then there's a cycle.

Algorithm :-
Store every address in Array if that is not found in array continue until match is found.

* If we allocate memory using Malloc, then elements inside struct, values have garbage value.
* If we allocate memory using Calloc, then elements inside struct will be initialized to 0

Q Write a C program to implement Stack using LL.
Q Write a C " " " Queue " "
Q " " " " " LL " arrays.
Q " " " " perform following opn on LL:-
(i) Union of 2 LL.
(ii) Intersection of 2 LL.
(iii) Polynomial addition of 2 LL
(iv) " multiplication of 2 LL

Date

18.08.12

Stacks

```
Struct stack {int data;
              Struct stack *next;
             }; Struct stack *top=NULL;

Push(struct stack *top, int x)
 {
  Struct stack *p=(stack*)malloc(sizeof(stack));
    if(p==NULL)
     { printf("Memory is full");
       exit(1);
     }
    else
     {p->data=x;                    ---> O(1)
      p->next=top;
      top=p;
     }
 }

Pop(int x, Struct stack *top)
 {Struct stack *temp=top;
   if (top==NULL)
     { printf("Stack is empty");
       exit(1);
     }
  top temp=top;                     ---> O(1)
     top=top->next;
     x=temp->data;
     free(temp);
     temp=NULL;
 }
```

Queue

*If To implement Queue property, we need atleast 2 poi
(Front & Rear)

Add

```
Struct queue{
 int data;
 queue * next;
};
```

```
struct queue *front=NULL,*rear=NULL;

Add(int x)
{queue *temp=(queue *)malloc(sizeof(queue));
 temp->data=x;
 if (front==NULL && rear==NULL)
   {front=temp;
    rear=front;                    ->O(1)
    front->next=NULL;
   }
 else if(front==rear)
   {temp->next=front=NULL; rear->next=temp;
 rear front=temp;
   }
}

Del(int x)
{ queue *temp=front;
 if (front==rear==NULL)
   {printf("Queue is empty");
    exit(1);              // no element
   }
 if (front==rear)                  ->O(1)
   {front=front=NULL;
    rear=NULL;            //only one-element
    free(temp);
    temp=NULL;
   }
 else
   {front=front->next;
    free(temp);
    temp=NULL;
   }
}
```

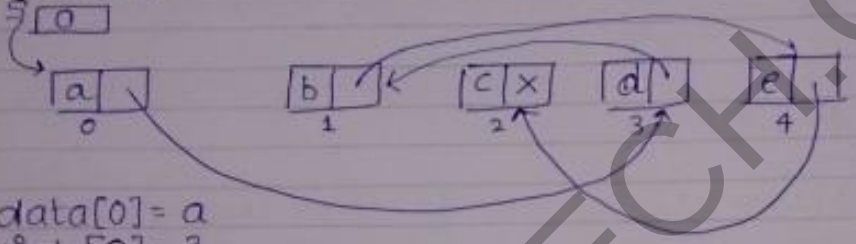
Implementing Linked List Using Arrays

0	a
1	b
2	c
3	d
4	e

data

0	3
1	4
2	-1
3	1
4	2

link



data[0] = a
link[0] = 3

Q. Write a C program to display data part of the linked list using array representation.

```
display(int *d, int *l, int * start)
  { int *temp = start;
    while (*temp!=-1)
      { printf("%d", data[temp]);

display(data, link, s)
  { int temp = s;
    while (temp!=-1)
      { printf("%d", data[temp]);
        temp = link[temp];
      }
  }
```

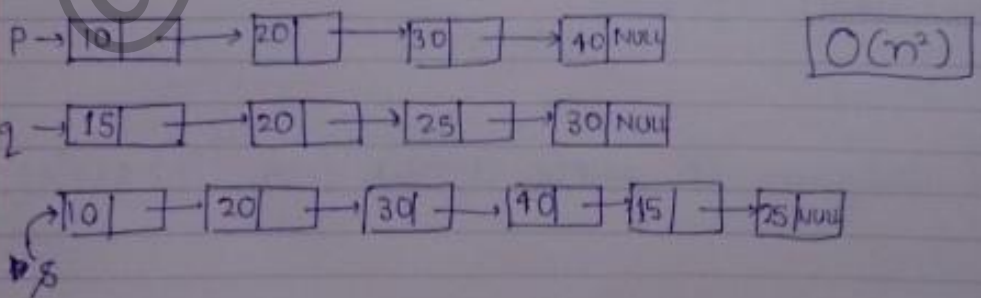
Q. Write a C program to delete a last node from linked lis

```
del(data, link, s)
 { temp = s;
   prev = s;
   while (link[temp] != -1)
     { prev = temp;
       temp = link[temp];
     }
   link[prev] = -1;
 }
```

or

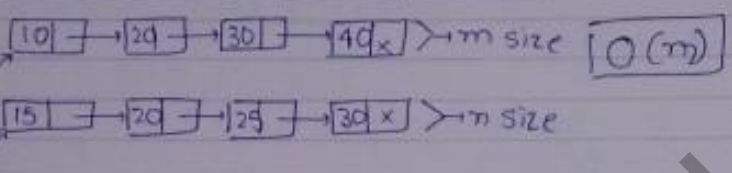
```
deletelast(data, link, s)
 { temp = prev = s;
   if(link[s] == -1)
    {s = -1;
     exit(1);
    }
   else
    { while(link[temp] != -1)
       { prev = temp;
         temp = link[temp];
       }
     link[prev] = -1;
    }
 }
```

Union of 2 Linked List :-



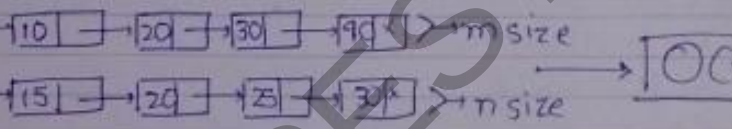
Algo:-
1 Add first linked list to the result. → O(n)
2 Take every element of 2nd LL & check if it is available in first Linked List or not using Linear Search. → O(n²)
- if it is available, dont add it in LL.
- otherwise add it to LL.

Concatenation of 2 LL



Algo:-
1 Go to the last node of the 1st LL. → O(n)
2 & link that node with the starting node of the 2nd LL. O(1).

Intersection of 2 LL:



→ O(n²) or O(mn)

1 Take the 1st node of 1st Linked List & check whether the node's data (of 1st LL) matches with any node's data of 2nd Linked List.
- if it matches with it then add it to result LL.
- otherwise dont add
2 Repeat step 1 with all the other nodes of 1st LL.

Polynomial Addition

p: $10x^3 + 5x^2 + 2x + 5$
q: $20x^3 + 15x^2 + 5x + 20$
$30x^3 + 20x^2 + 7x + 25$

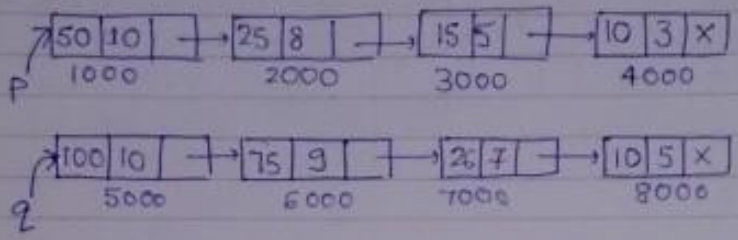
p: $10x^{15} + 20x^5 + 3x^2 + 16$
q: $5x^5 + 7x + 10$
$10x^{15} + 25x^5 + 3x^2 + 7x + 26$

```
struct poly{
  int power;
  int coe;
  poly * next;};
```

Polynomialaddition (

Time Complexity:-

O(m+n)



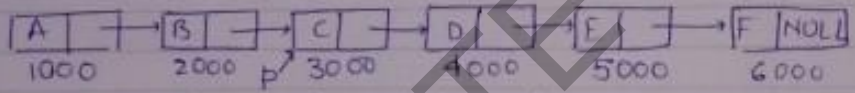

```
While(p!=NULL && q!=NULL)
  {
   if(p->power == q->power)
    {r->power = q->power;
     r->coe = p->coe + q->coe;
     p=p->next;
     q=q->next;
     r=r->next;
    }
   else
    {if(p->power>q->power)
      {r->power = p->power;
       r->coe = p->coe;
       r=r->next;
       q=q->next;
       p=p->next;
      }
     else
      {r->power=q->power;
       r->coe = q->coe;
       r=r->next;
       q=q->next;
      }
    }
  if(p==NULL && q==NULL)
    return (r);
  else
    {if(p==NULL)
      {add q to r;}
     else
      {add p to r;}
}
```

polynomial Multiplication :-

```
while (p != NULL)
  { while(q != NULL)
      { r->power = p->power + q->power;
        r->coe   = p->coe * q->coe;
        r = r->next;
        q = q->next;
      }
    p = p->next;
    q = s; //starting of q
  }
```

→ $O(m^2)$

2. Consider the following LL:-



from the above linked list delete a node pointed by p if only p is given. (starting is not given.)

Ans.

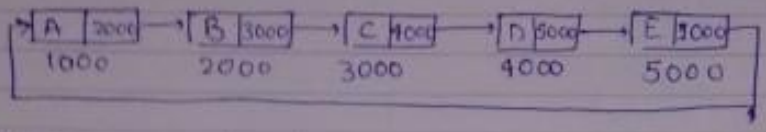
```
del (node *p)
{ node *temp = start;
  node *prev = start;
  while (temp != p)
  {
copy the content
p->data = (p->next)->data;  // but if p is last node,
p->next = (p->next)->next;  // then this algo fails
free (p->next);
p
```

Drawbacks of Singly LL :-

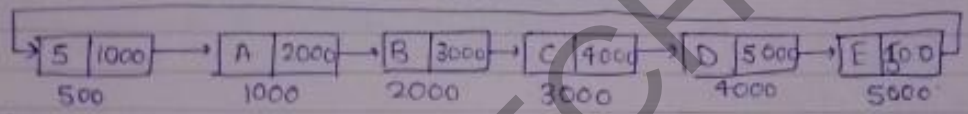
① We can move only in one direction. (we can't go back)

② In singly LL last node's next part is always NULL w/o any use (we can keep starting node address, so that we can go back)

Circular SLL :-



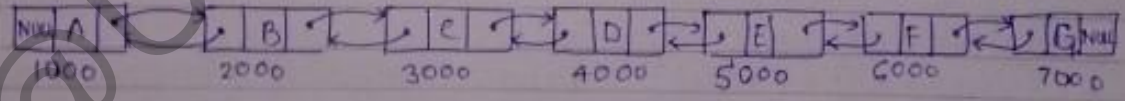
1. We can go back
2. The drawback with circular SLL it may enter into the infinite loop
3. In order to eliminate the infinite loop in circular SLL, we are going to head node.
4. Head node data part will contain valuable info. like total no. of nodes in circular SLL.



if

Doubly Linked List

```
struct DLL
{ struct DLL *left;
  int data;
  struct DLL *right;
};
```



Q. Write a C program to delete a node which contain data as x in the given double linked list.

Ans.
```
del(int x)
{ struct DLL * ptr = start;
  if (start -> data == x)
  { start = start -> right;
    start -> left = NULL;
    free(ptr);
    ptr = NULL;
  }
```

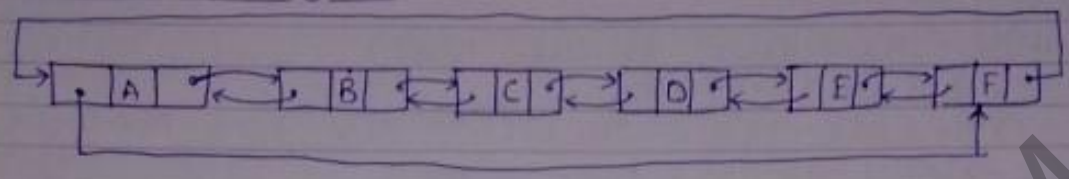
```
    * if (p!=NULL)
        {exit(1);
        }

  else
    { while (ptr->data != x && ptr != NULL)
        { ptr = ptr->right;
        }* if (ptr->right == NULL){(ptr->left)->right = ptr->right;
                                   free(ptr); ptr=NULL;}
  else{(ptr->left)->right = ptr->right;
      (ptr->right)->left = ptr->left;
       free(ptr);
    }ptr = NULL;
  }
}
```

Q Write a C program to insert a node with data x before a node with data y.

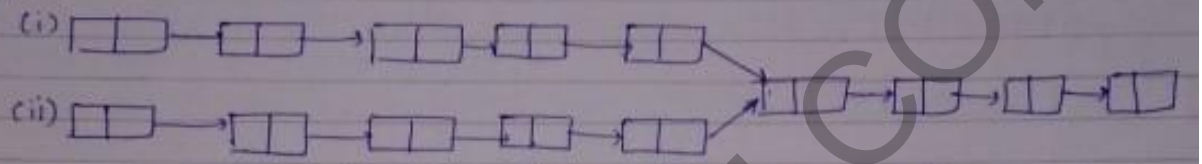
Ans.

```
add (int y, dll * start)
{dll * ptr = start;  dll * temp = (dll *)malloc(sizeof(dll));
 while(ptr->data != y && ptr != NULL)
    { ptr = ptr->right;
    }  if (ptr == NULL) {exit(1);}
 if (ptr == start)
   {temp->data = x;
    temp->left = NULL;
    start->left = temp;
    temp->right = temp->start;
    start = temp;
   }
 else if (start == NULL)
   { temp->data
     exit(1);
   }
 else
   { temp->data = x;
     temp->left = ptr->left;
     temp->right = ptr;
     (ptr->left)->right = temp;
     ptr->left = temp;
   }
}
```

Circular Doubly LL



Q



mark all nodes with flag 0, & make the flag as 1 as the node is visited (do this for 1st LL & then for (ii)nd & check if the flag is 0 or 1, if it is already 1, from that node onwards, count the no. of nodes till NULL) → O(n)

2n

★★ Application of Doubly LL :- Binary Tree

Binary Tree

Q. Write a C program to find no. of leaf nodes in the given binary tree

Ans.

```
totleafnodes (tree * p)
{ if (p->left == NULL && p->right == NULL)
    { itt; return ;
    }
  else{
  totleafnodes (p->left);
  totleafnodes (p->right);
  }
}
void main()
{ tree *start ; // assume it is assigned to root
  int t = 0;    // address
  if (start != NULL)
  {totleafnodes (start); }
}
```

```
struct tree
{ tree * lc;
  int data;
  tree *rc;
};
```

Or

```
NLN (struct tree *s)
{if (s==NULL) return 0;
 else{if (s->lc==NULL && s->rc==NULL)
       return(1);
       else{
          xL = NLN(s->lc); → T(n/2)
          xR = NLN(s->rc); → T(n/2)
          return(xL+xR); → c
       }
 }
```

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

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

$\therefore$ **$O(n)$**

or $T(n) = T(a) + T(n-(a+1)) + c$

if $a = 0$

$T(n) = $ **$O(n^2)$**

Q. Write a C program to find no. of non-leaf nodes in a given binary tree

```
int i=0;
totnonleafnodes (tree *b)
{/*if (b->left!=NULL || b->right!=NULL)
   {i++;
    return;
   }
 else
   {totnonleafnodes (b->left);
    totnonleafnodes (b->right);
   }
}
void main()
{if (root!=NULL)
   {totnonleafnodes(root);
   }
}
```

```
if (p==NULL)
  return;
* if (p->left!=NULL &&
     p->right==NULL)
  {return;
  }
```

Or

```
NLN (tree *s)
 {if (s==NULL) return 0;
  else
    if (s->lc==NULL && s->rc==NULL)
       return(0);
```

```
else
  {xL = NLN(s->lc);
   xR = NLN(s->rc);
   return(1+xL+xR);
  }
}
```

Q Write a C program to find total no. of nodes in the given binary tree.

```
int i=0;
totnodes(tree *p)
{/*if(p->left==NULL&&p->right==NULL)
    {itt;
     return;
    }
  else
    {itt;
     totnodes(p->left);
     totnodes(p->right);
    }
}
```

```
*if(p==NULL)
  return;
```

Or

```
NLN(tree *s)
{if (s==NULL) return 0;
 else
  {if(s->lc ==NULL && s->rc==NULL)
     return (1);
   else
     {xL = NLN (s->lc);
      xR = NLN (s->rc);
      return (1+xL+xR);
     }
  }
}
```

Q Write a C program to find height of a given binary tree.

Ans height (tree * s)
{if (s==NULL) return 0;
else
{if (s->lc==NULL && s->rc==NULL)
return(0);
else
{xL=height(s->lc);
xR=height(s->rc);
return(1+max(xL,xR));
}
}
}

Or
height (tree *p)
fn= no. of nodes

Q. Write a C program to check given binary tree is strict binary tree or not.

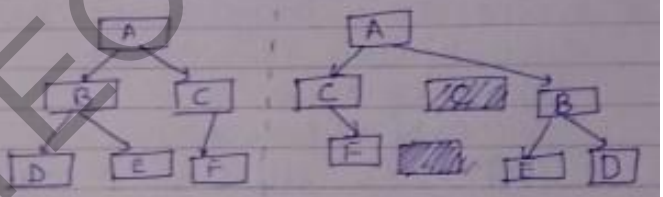
bool flag=true;
StrictOrNot (tree * p)
{ if (p->left==NULL && p->right==NULL)
{return;
}
elseif((p->left==NULL && p->right!=NULL) ||
(p->left!=NULL && p->right==NULL))
{ flag=false;
return;
}
else
{StrictOrNot (p->left);
StrictOrNot (p->right);
}
}
Or
StrictBT (tree * s)
{if (s==NULL) return (1);
else
{if (s->lc==NULL && s->rc==NULL)
return 1;
else

```
if(s->lc!=NULL && s->rc!=NULL)
   return (StrictBT(s->lc) && StrictBT(s->rc));
else
   return 0;
}
```

Q Consider the following C program:-

```
Tree (Struct Btnode *s)
{if (s!=NULL)
   {t=(Struct Btnode *)malloc(sizeof(Struct Btnode));
    t->data = s->data;
    t->lc = s->rc; tree(s->rc);
    t->rc = s->lc; tree(s->lc);
    return(t);
   }
}
```

Mirror Image



Q Write a C program to find min. element in the given binary search tree

Ans.

```
findmin(tree *p)
{ if (p->left==NULL)
    return(p->data);
  else
    findmin(p->left);
}
```

or

```
find min (tree *s)
{ if (s==NULL) return;
  else
  { while (s->lc!=NULL)
      s = s->lc;
    return(s->data);
  }
}
```

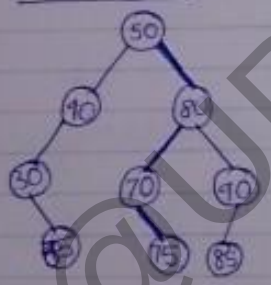
Q. A data structure comprises of nodes, each of which has exactly 2 pointers to other nodes with no Null pointers, the following C program is to be used to count the no. of nodes in given binary tree. It uses a mark field assumed to be '0' initially for all nodes. Find the missing statement in the code :-

```
→ struct test
  {int info,mark;
   struct test *p,*q;
  };
int nodecount(struct test *a)
  {if (a→mark)
      return(0);
   else
   {return(nodecount(a→p)+nodecount(a→q)+1);
   }
   a→mark=1; (my answer)
  }
```

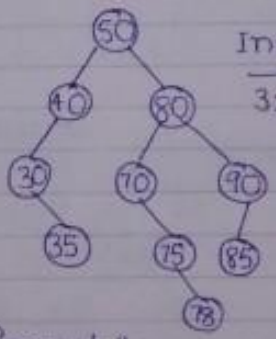
search

Binary Tree

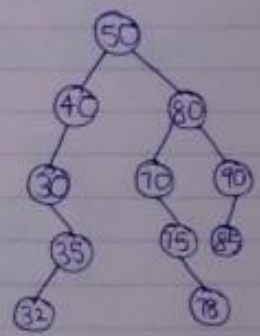
Insertion



Insert 78



Insert 32



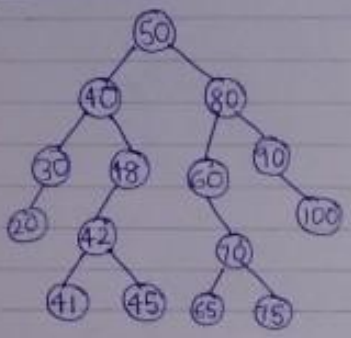
Finding the position to insert :-

- Best Case :- $\Omega(1)$ (root itself)
- Worst Case :- $O(n)$
- Avg Case :- $O(\lg n)$

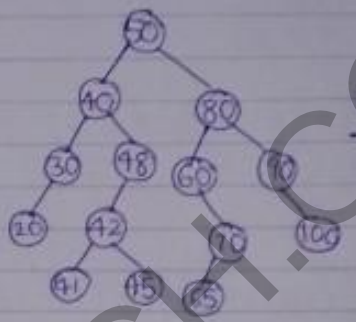
Algo

① Find the place of x → O(n)
② Insert element x → O(1)
O(n)

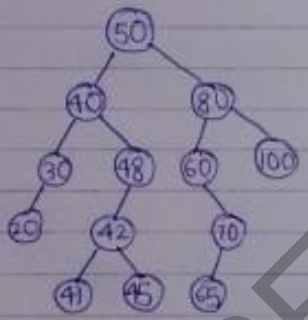
Deletion



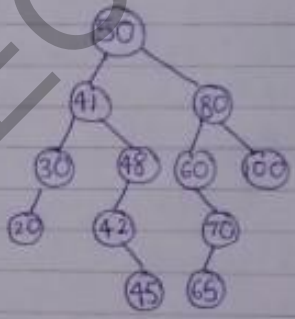
Delete 75 →

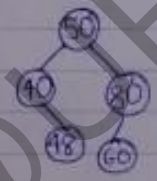


Delete 100 →



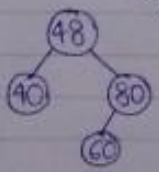
Delete 40 →



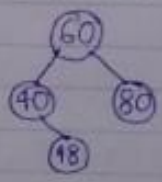


delete 50 →

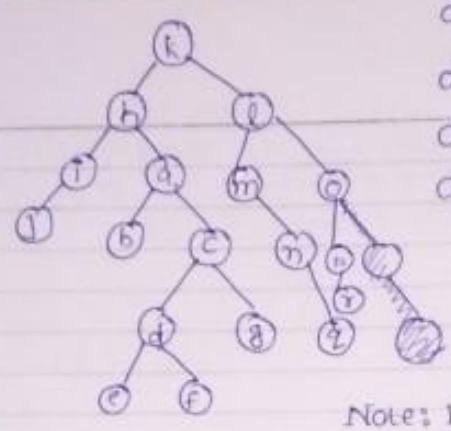
either choose the Left Subtree largest child or
Right Subtree smallest child.



or



Q.



delete (a) = inorder predecessor or inorder successor → O(1)

delete (c) = e (or) g

delete (d) = e (or) f

delete (o) = connect l & p

delete (q) = no need to do any

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

n → finding place; O(1) → for adjustment

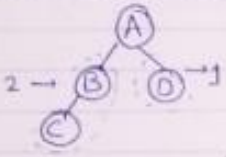
Note: If we delete any node from BST replacement will be done by inorder successor or inorder predecessor.

AVL Tree

(Adelson, Valski', Landies)

Or Balanced Binary Search tree [Size → O(log n)]

Balanced Factor = H(LST) - H(RST) or H(RST) - H(LST)



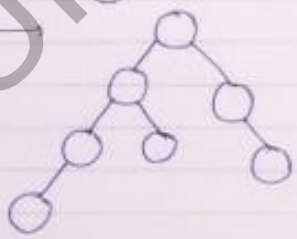
Balanced Factor (of A) = 2 - 1 = 1

Note - A BST in which every node's Balance factor is -1, 0 or 1 (max. diff.) is called AVL tree.

Height = 2 →

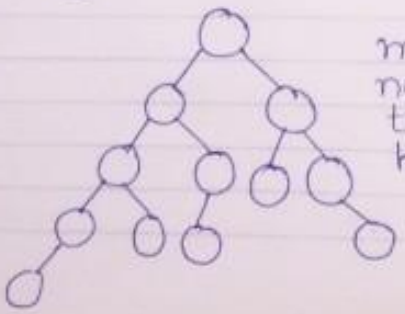
min no. of nodes for a tree to be AVL of height = 2 = 4

Height = 3 →



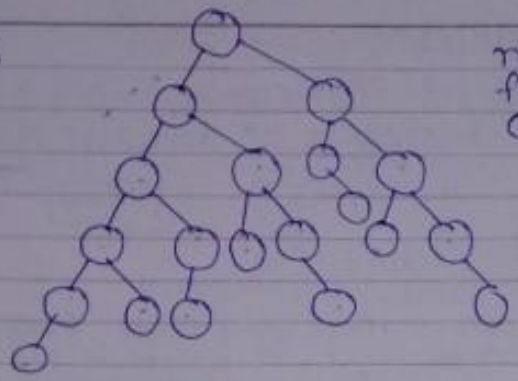
min no. of nodes for a tree to be AVL of height 3 = 7

Height = 4



min no. of nodes for a tree to be AVL of height 4 = 12

height = 5



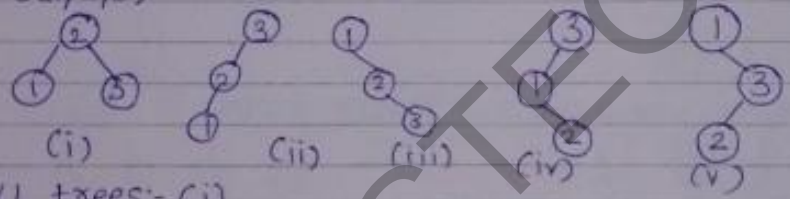
min. no of nodes for a tree to be AVL of height 5 = 20

Min. no. of nodes in a height H AVL tree = MNN(H-1) + MNN(H-2) + 1

(MNN)

$$MNN(H) = MNN(H-1) + MNN(H-2) + 1$$

Qn: 3(1,2,3) → BST



AVL trees:- (i)

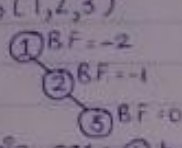
BST:- (i), (ii), (iii), (iv), (v)

(ii), (iii), (iv), (v) are converted to (i) using rotations, & this rotation should take constant time.

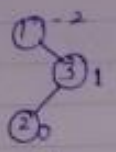
Date

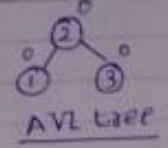
19.08.12 AVL Trees

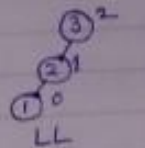
n=3 ⇒ (1,2,3)



In this case ① has problem from R (because of ②) & ② has problem from its right ∴ RR problem.

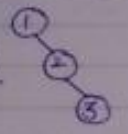




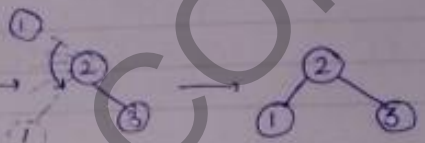


Rotations :-

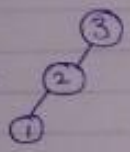
RR



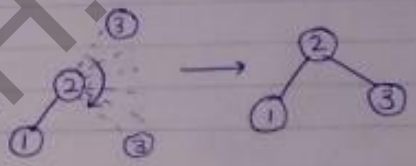
→ 1 keep the nodes as it is which has B.F = 0,1,-1 & rotate the node which has B.F ≠ 0,1,-1 to the anticlockwise (one left rotation)



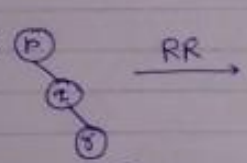
LL



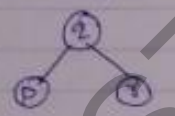
→ 1 keep the nodes as it is which has B.F = 0,1,-1 & rotate the node which has B.F ≠ 0,1,-1 clockwise (one right rotation)



RR



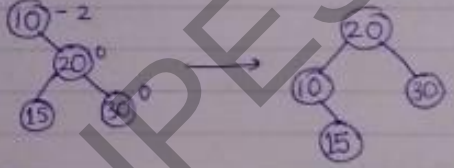
RR →



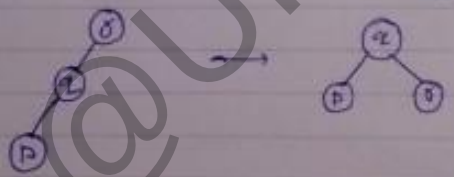
q → lc = p

p → rc = NULL

e.g.



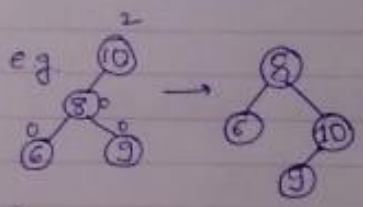
LL



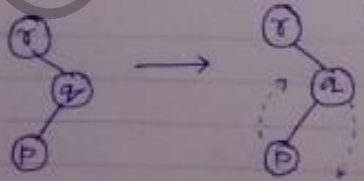
q → rc = r

r → lc = NULL

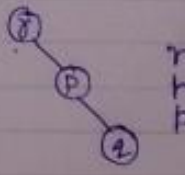
e.g.



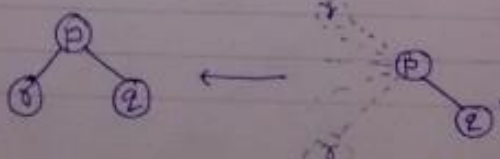
RL

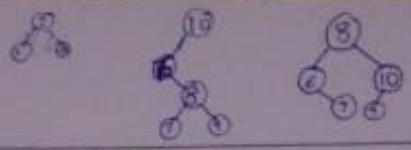


→ 1 do one clockwise rotation →

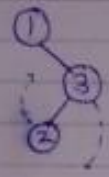


now this has RR problem.

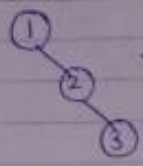




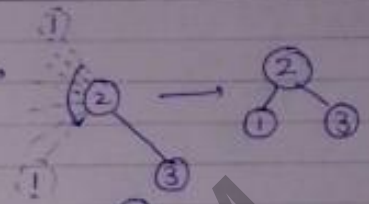
e.g.



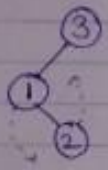
1 clockwise rotation



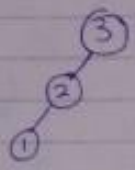
now its RR →



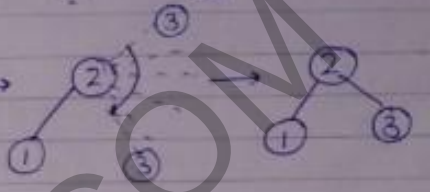
LR



1 anti-clockwise rotation

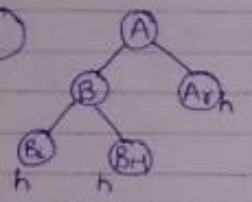


now its LL →



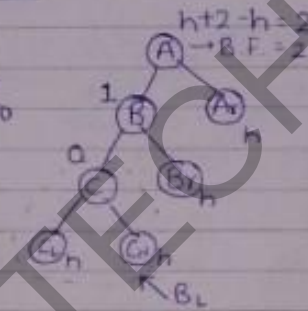
Insertion into AVL tree :-

(1)



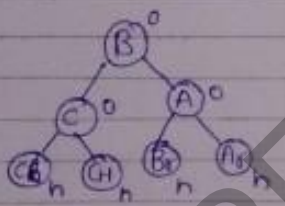
AVL tree

Insert an element into B_L

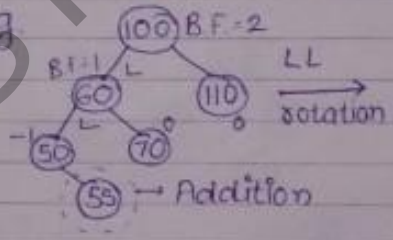


LL rotation

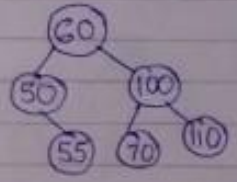
(Problem arises because of insertion in B_L which is Left to B (immediate Left of A))



e.g.

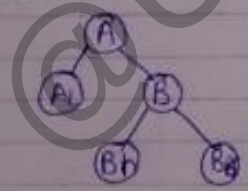


LL rotation →

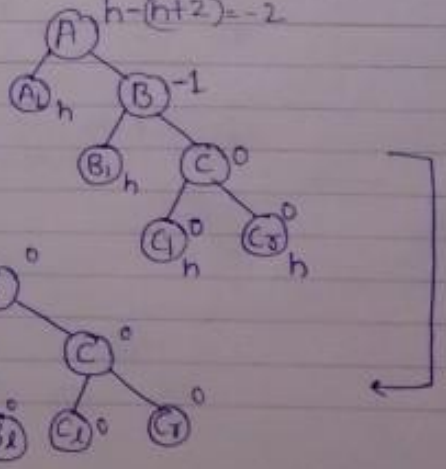


Addition

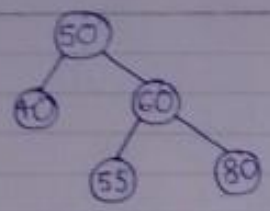
(2) RR problem



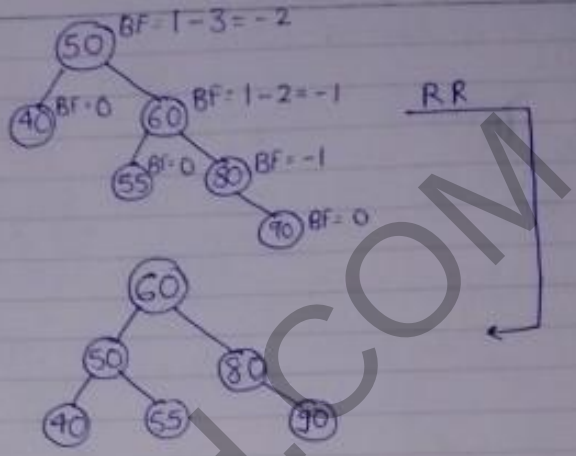
Insertion into B_R →



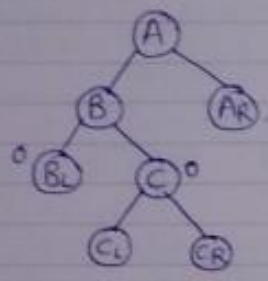
e.g



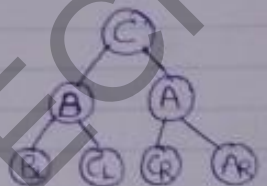
Add 90 →



LR



LR →

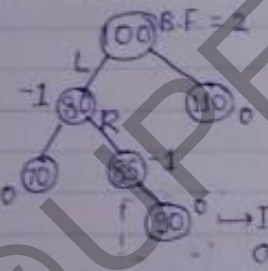


C→R = B
C→L = A
B→R = temp
A→C = temp1

C→L = temp
C→R = temp1

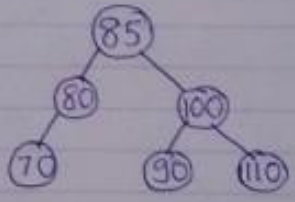
★ In LR problem 2 rotations are compulsory.
★ After doing one rotation we will get LL.
★ after getting LL problem, apply LL rotation.

e.g

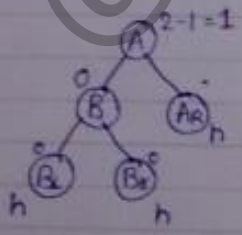


→ Insertion of 90.

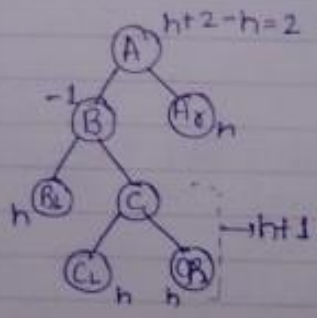
LR →



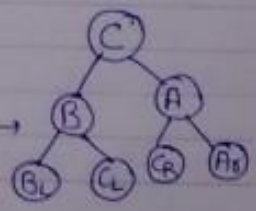
Sir's explanation :-



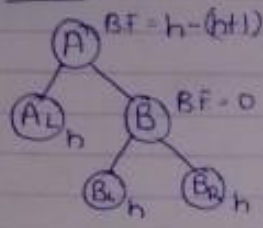
Insert element in B_R →



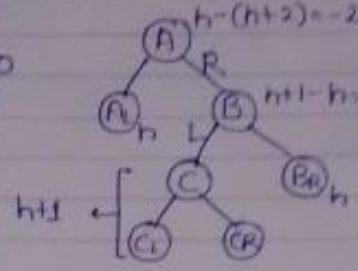
LR →

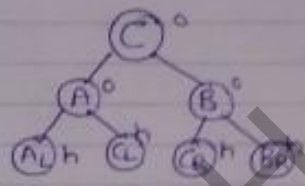


RL



Insert into B_L →

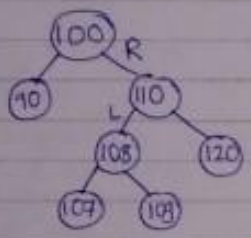




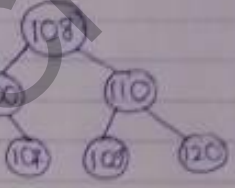
RL rotation

* In RL rotation 2 reverse rotations are compulsory
* After 1 rotation we will get RR problem.
* Apply 1 more RR rotation to get AVL tree.

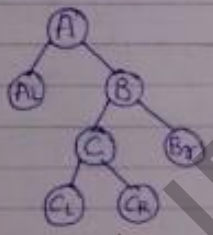
eg

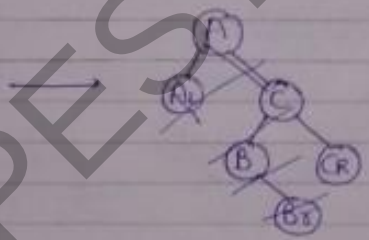


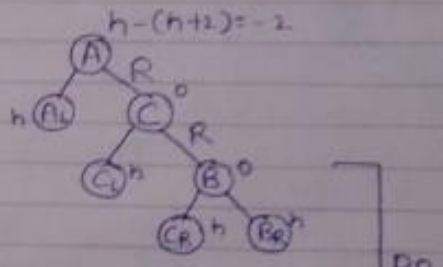
RL problem →



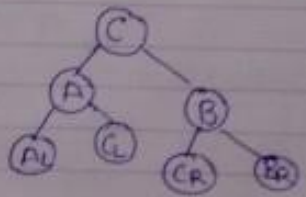
Or



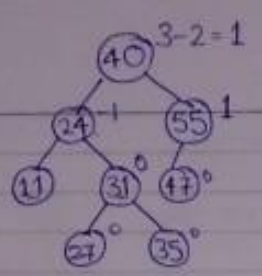


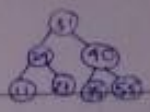


RR

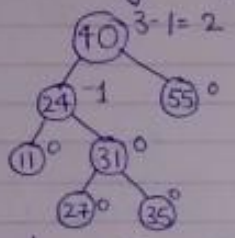


Q. Consider the following AVL tree :-



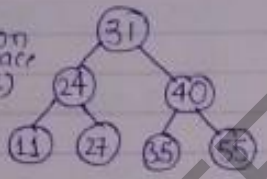


Aftes deleting 47, show resultant AVL tree →



★ To delete an element, search that element, Searching complexity : O(n) [BST & not AVL]
O(logn) [AVL & BST]
& then perform the deletion algorithm.

LR (Deletion takes place Left from RL) rotation




```
Struct AVL
{ int data;
  int height;
  struct AVL *lc, *rc;
};
```

Complexity :-

① Finding place of deletion → lgn

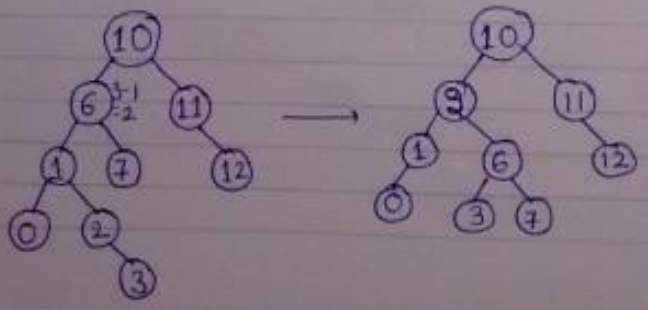
② Deletion of that element → constant

③ Traversing the complete path from site of deletion to the root to check the balancing factors = logn

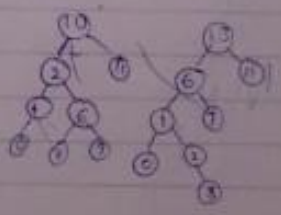
2logn = O(logn)

Q. What will be the resulting AVL tree if we insert following elements into AVL tree :-
10, 6, 11, 12, 1, 7, 0, 2, 3

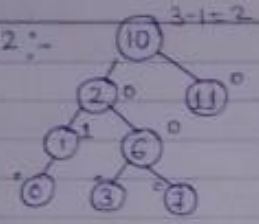
Addition → O(logn)



AVL Tree



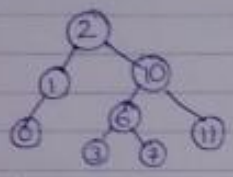
delete 12 :-



Now problem is because of Left subtree of 10, L, now, we are at 2, we can 2 has BF=0, ∴ we can go both left & right, going left results in LL
" right " " LR

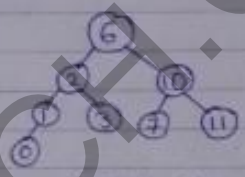
What will be the resulting AVL tree if we delete 12

LL →



1 rotation

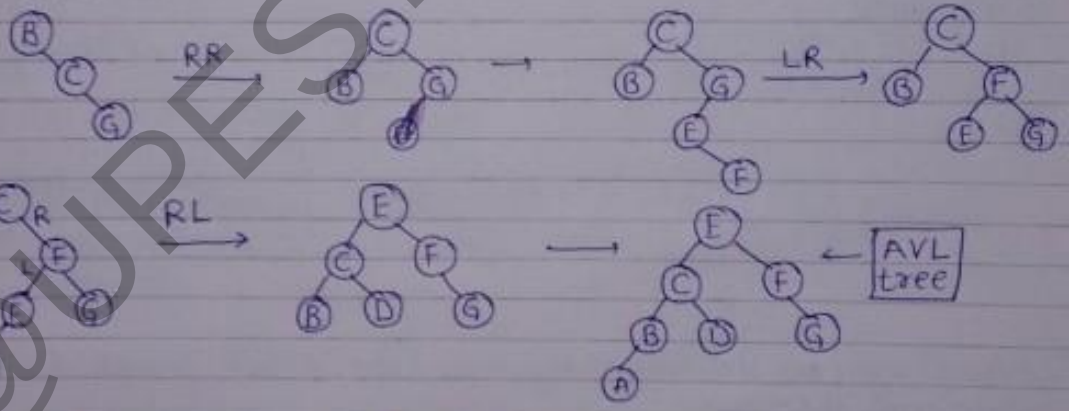
LR →



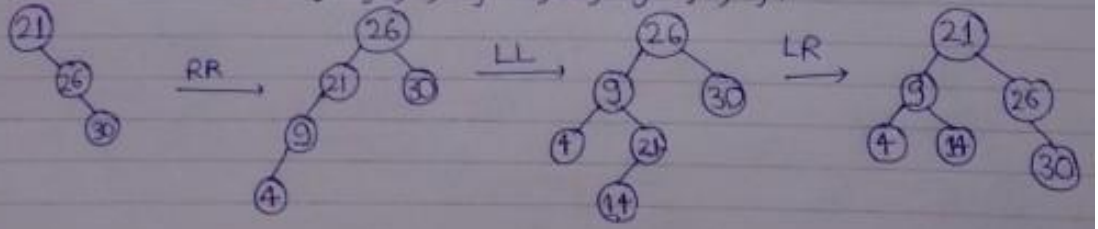
2 rotations

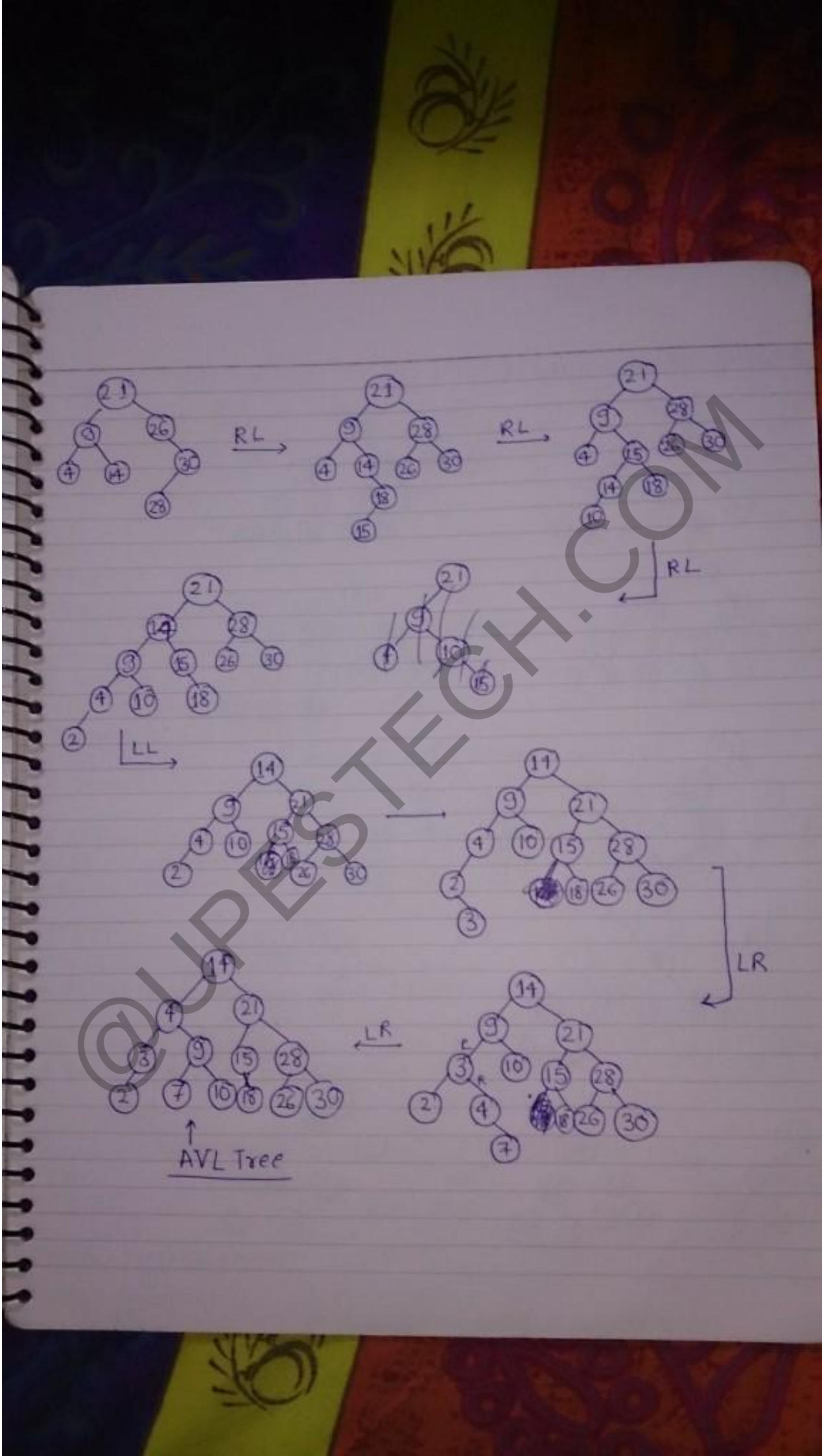
We could apply any of these. (either LL or LR)

Q. Construct AVL trees from following keys
B, C, G, E, F, D, A

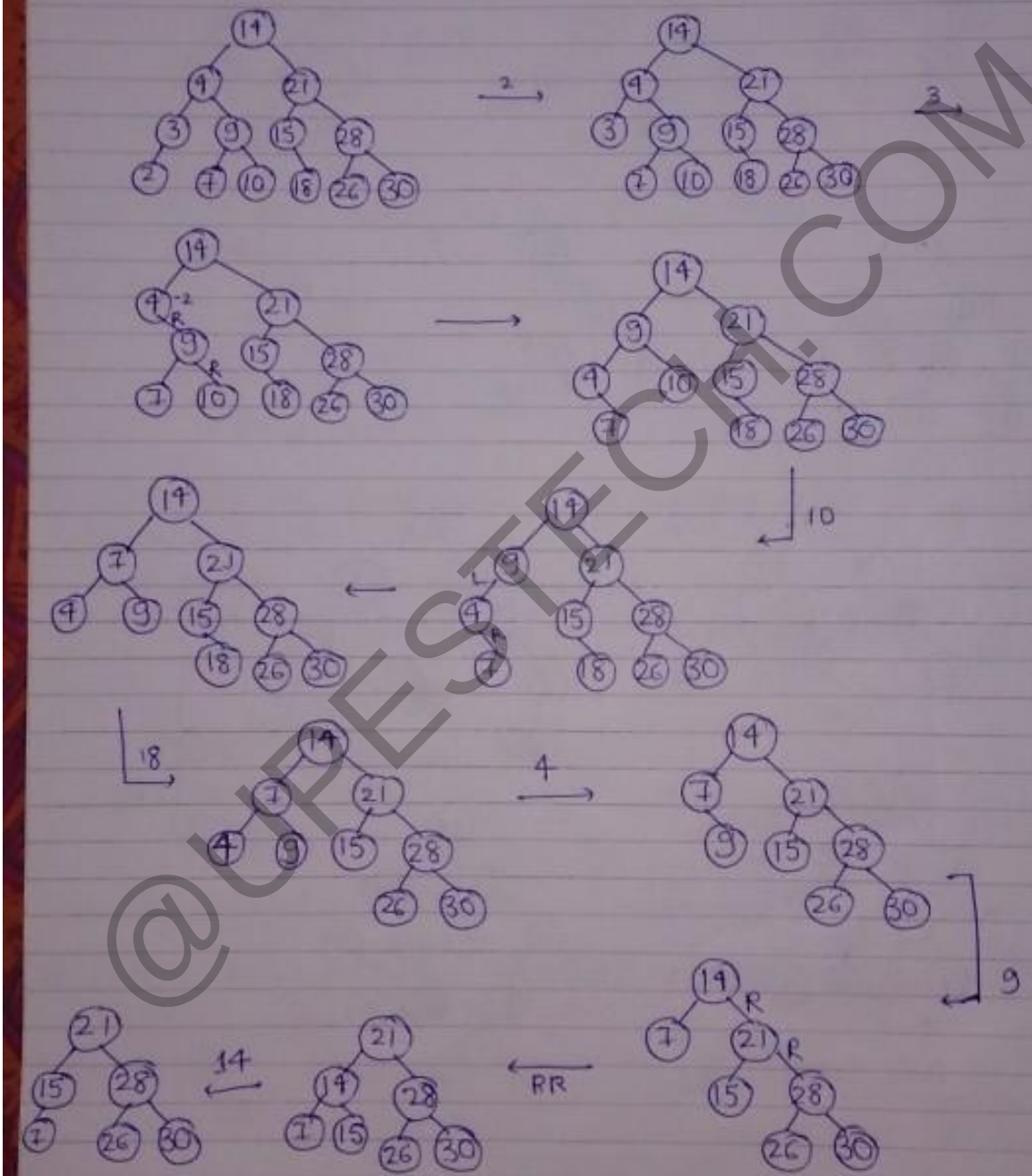


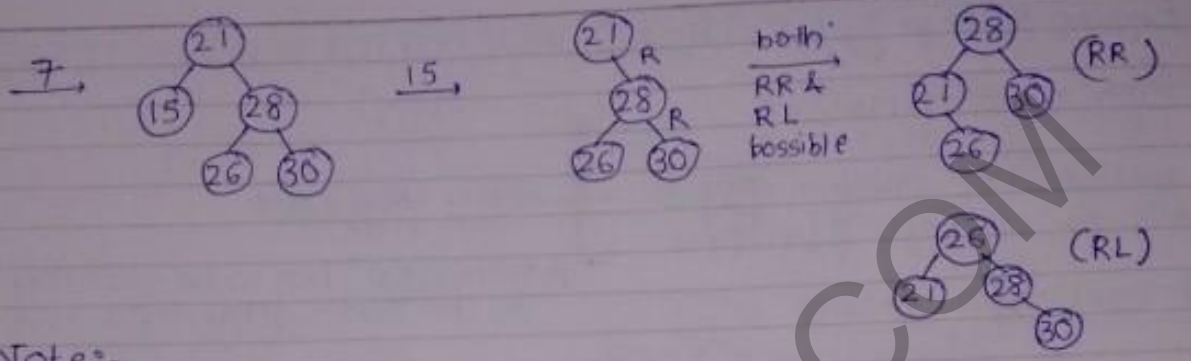
Q Create AVL:- 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7





Q. Delete 2,3,10,18,4,9,14,7,15





Note:-

Algorithm for insertion

1. Find the place of the element → O(log n)
2. Insert that element → O(1)
3. Traverse the same path to find BF → O(log n)
4. If not balanced at some node, perform rotation → O(1)

Total complexity:- O(log

Algorithm for deletion

1. Find the place of the element to be deleted → O(log n)
2. Deletion of element → O(log n) [when there are 2 children of a node to be deleted]
   → O(1) [when children are 0 or 1.]
3. Check the balancing factors → O(log n) by traversing
4. If not balanced at some node, perform rotation → O(1)

Total Complexity:- O(log n)

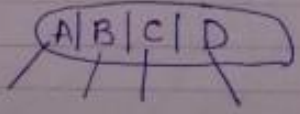
Multiway Search Tree

1. Search Tree (Ordered Tree)
2. At every node minimum 2 children allowed (order of multiw search tree (m)).
3. Height of Multiway search tree is $O(\log_m n)$. [Balanced M-wa search tree.]
4. Deletion & Insertion will take $O(\log_m n)$.

B-Tree:-

Order of B-Tree = m (assume) [means, each node can have max. 'm' children]

∴ maximum (m-1) keys
maximum m-children

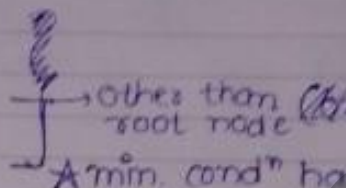


min. $\lceil \frac{m}{2} \rceil$ children

min. $\lceil \frac{m}{2} \rceil - 1$ keys.

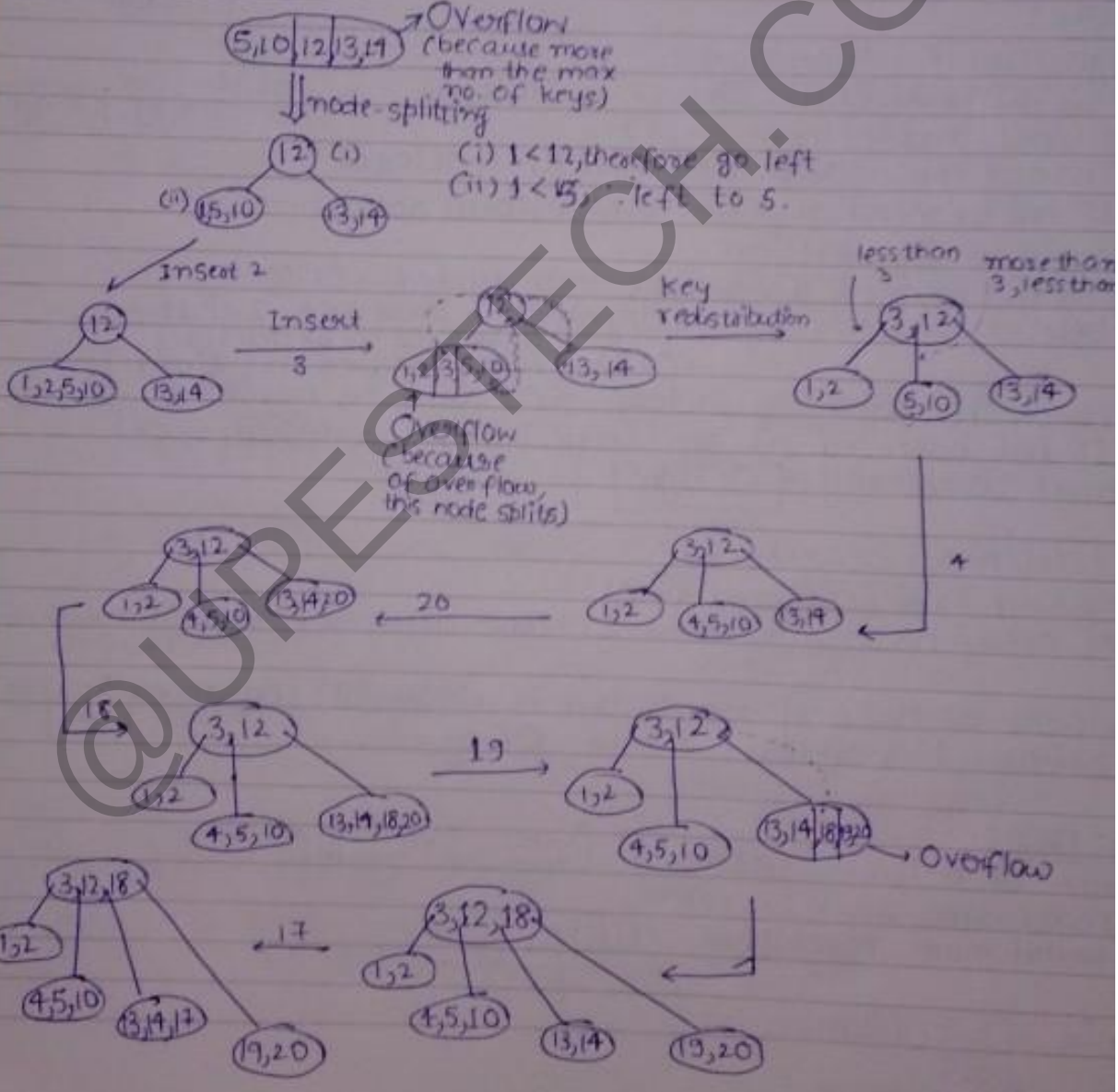
Degree = 5 | Keys: 5, 10, 12, 13, 19, 1, 2, 3, 4, 20, 18, 18, 17, 16, 15, 25, 23, 24, 22, 21, 30, 28, 29

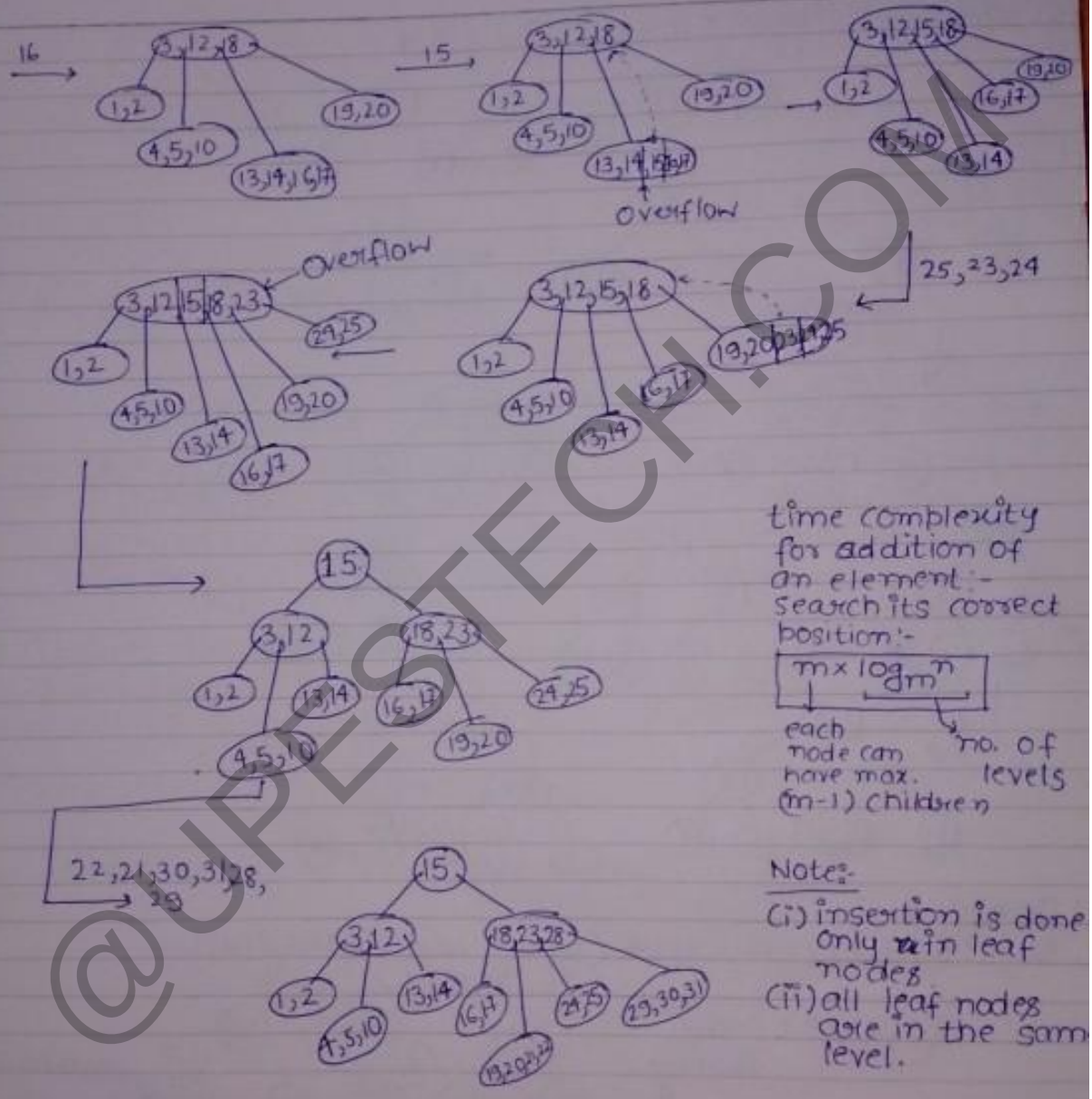
max. no. of children = 5
" " " keys = 4
min no of children = $\lceil \frac{5}{2} \rceil = 3$
min. no. of keys = $\lceil \frac{5}{2} \rceil - 1 = 3 - 1 = 2$



A "min. cond" has to be satisfied by the nodes other than root

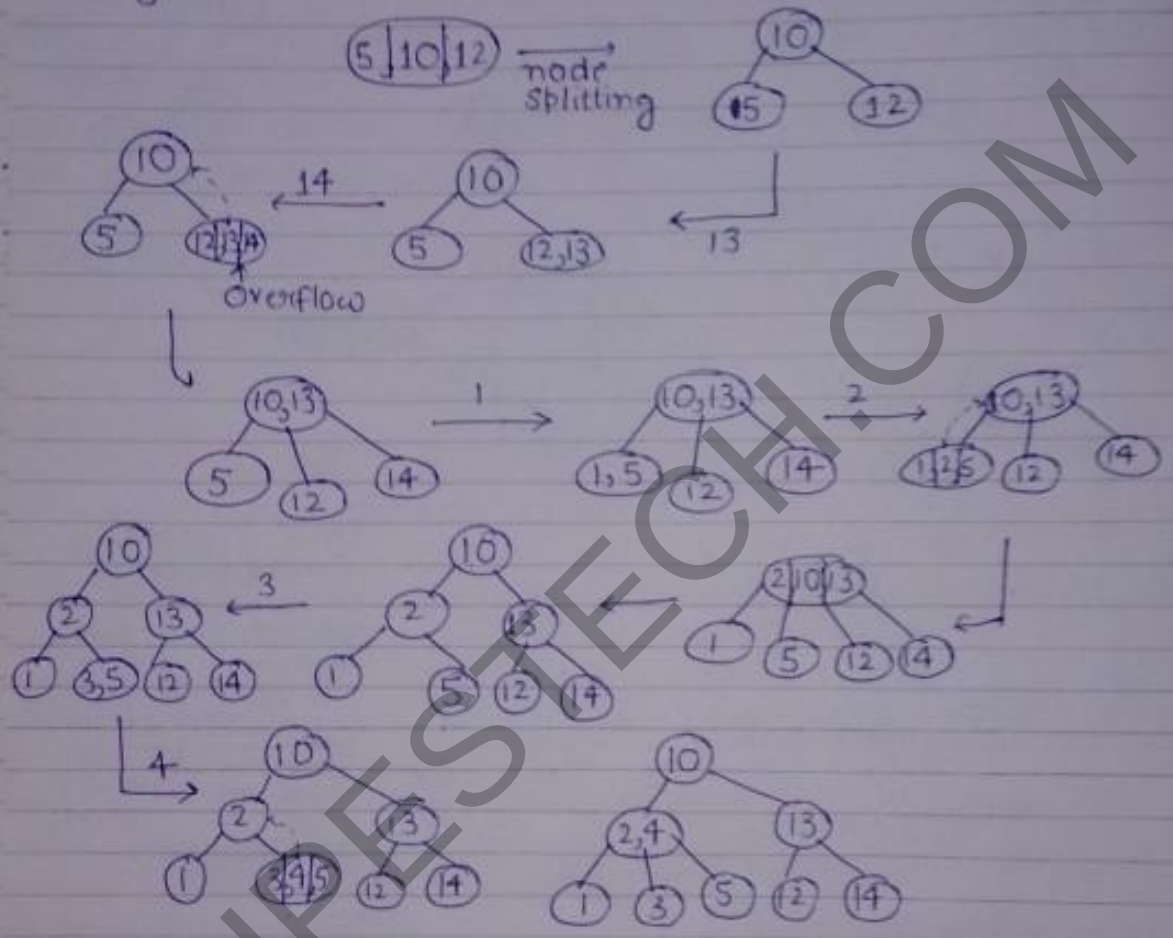
Insertion in B-Trees



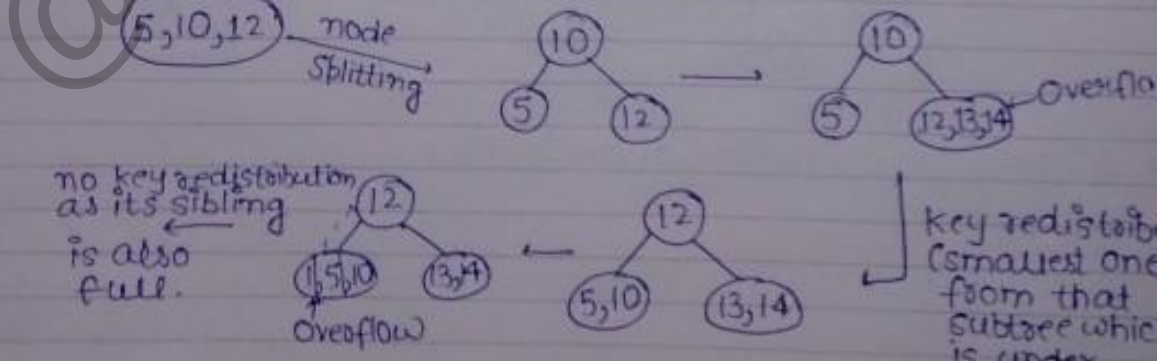


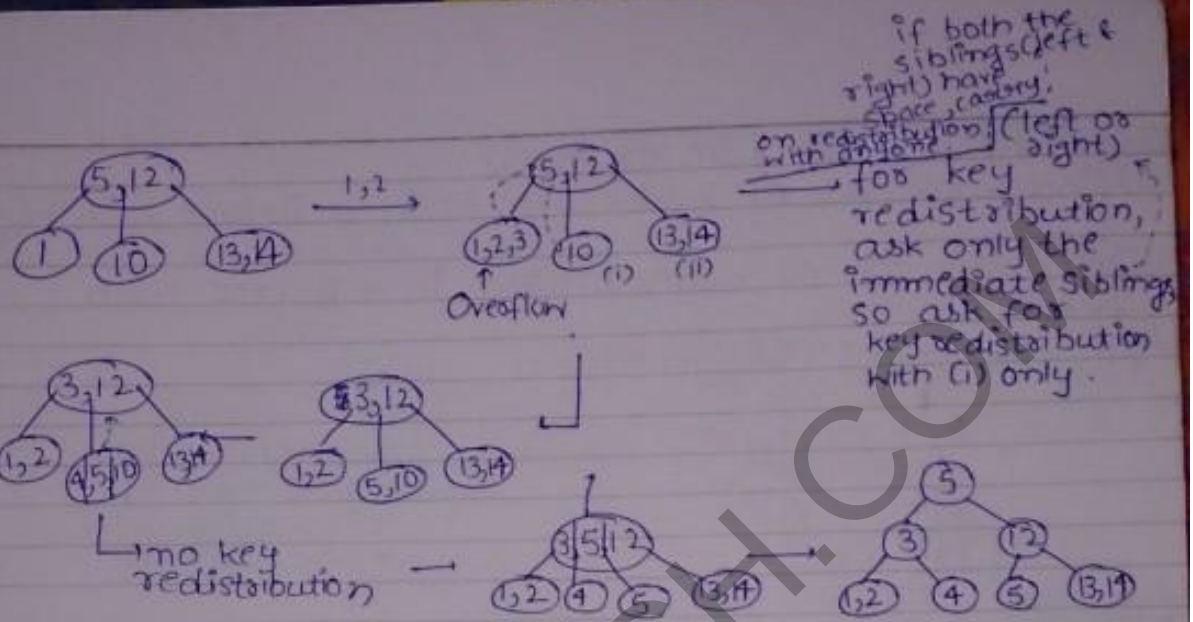
** A 2-3 tree (means max 3 child)

Q degree 3 | keys: 5,10,12,13,14,1,2,3,4



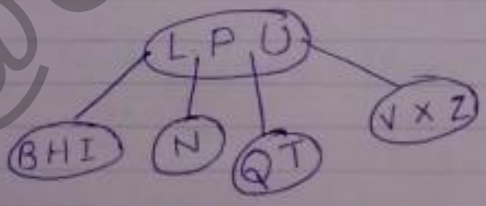
Q. Degree = 3
Keys: 5,10,12,13,14,1,2,3,4 (using key redistribution if possible)



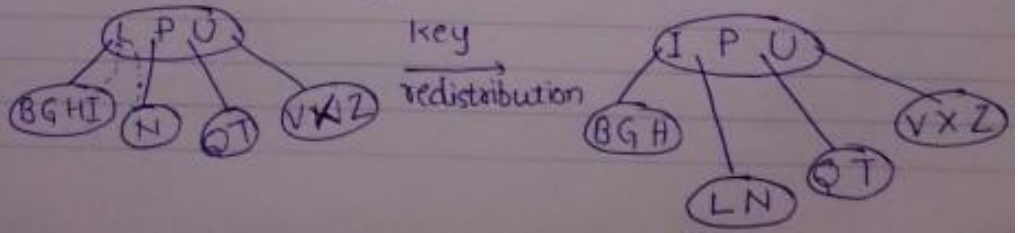


Note:- While constructing B-Tree for the given tree & keys, key-redistribution is always better than node splitting, because we are not wasting space & height of B-Tree also decreases, so time complexity also decreases

Q. Consider the following B-Tree:-
2-3-4 Tree (A B-Tree with min. degree of 2, i.e. max
min. children = 2
min. keys = 1
max. children = 4
max. keys = 3)
in which each data item is a letter



What is the result of inserting G in the above tree:-



Q. Consider a B-Tree with order = 32
(i) find max. no. of nodes present in a B-Tree with height 3.
(ii) find min. no. of nodes present in a B-Tree with height 3.

Ans. (i) height = $\log_{32} n$

$3+1 = \log_{32} n$

$4 = \log_{32} n$

$32^4 = n$

max. children = 32
max. keys = 31
min. children = 16
min. keys = 15

for max.

Level	nodes	keys	children
1	1	31	32
2	32	32*31	32*32
3	32*32	32*32*31	32*32*32
4	32*32*32	32*32*32*31	32*32*32*32

Total nodes = $32^3 + 32^2 + 32^1 + 32^0$

$= \frac{32^4 - 1[32^4 - 1]}{31}$

$\approx 32^3$

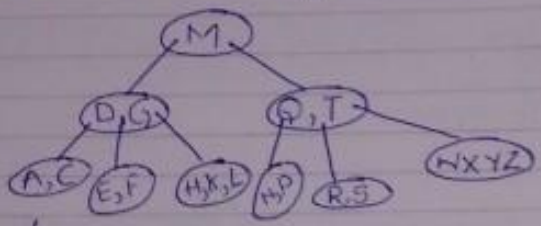
max. no. of nodes in a height h, order m B-Tree:-

$= m^0 + m^1 + \ldots + m^h$

$= \frac{1(m^{h+1} - 1)}{m-1} = \frac{m^{h+1} - 1}{m-1}$

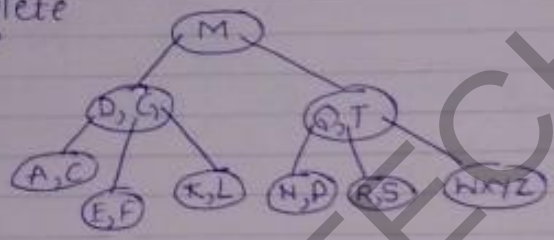
max. keys :- $(m-1)[m^0 + m^1 + \ldots + m^h] = \frac{(m-1)(m^{h+1} - 1)}{(m-1)} = (m^{h+1} - 1)$

Q. Consider the following B-Tree



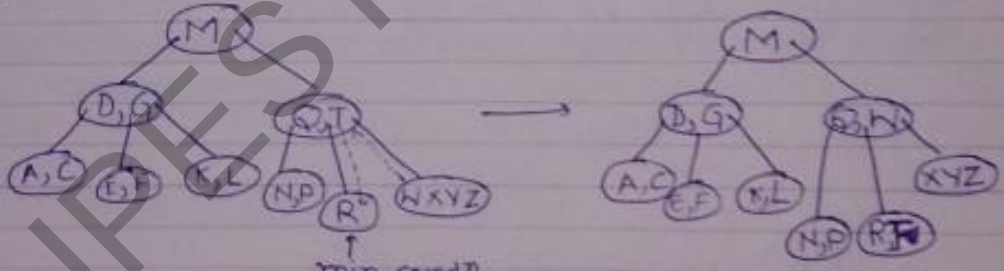
max. no. of keys = 4
order = 5
min no. of children = $\lceil \frac{5}{2} \rceil = 3$
min no. of keys = 3-1 = 2

Delete H



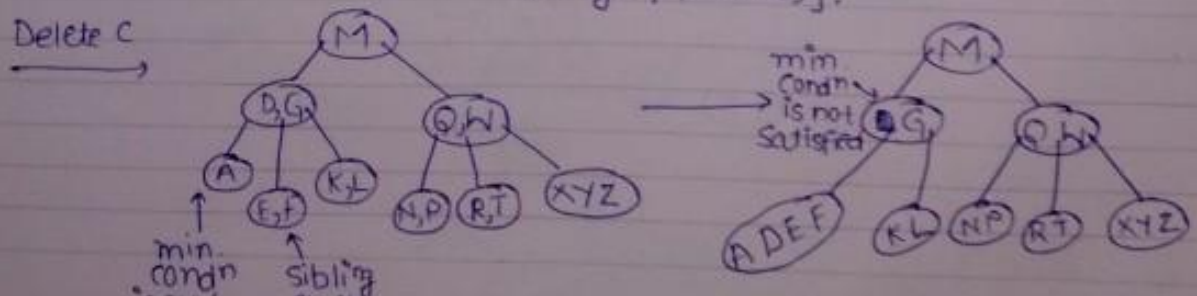
* If the element H is present in leaf node, delete that element & check for the min. cond^n, If it is satisfied, then do nothing, otherwise

Delete 5

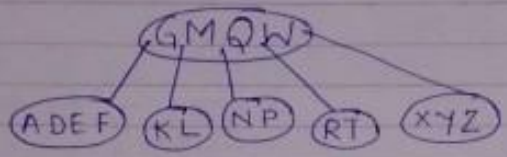


min. cond^n not satisfied
[ask left sibling → can't give
ask right " → can "]
[if min. cond^n is not satisfied, distribute the keys from immediate sibling (if possible)].

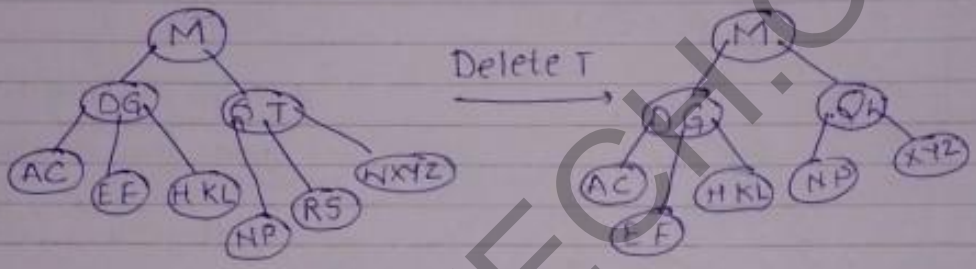
Delete C



G asks the sibling, but it can't help... node merging

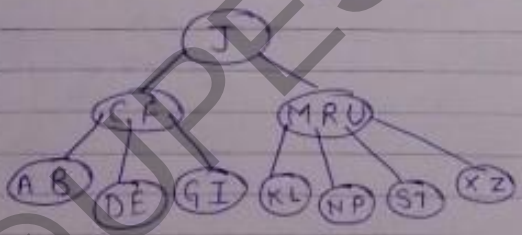


Q. Consider the following B-Tree.

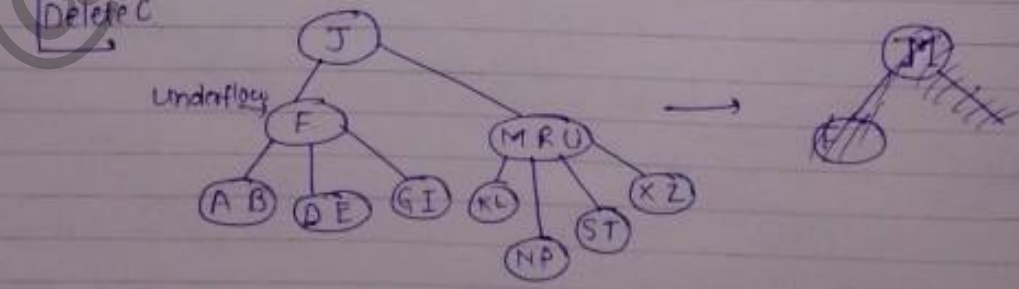


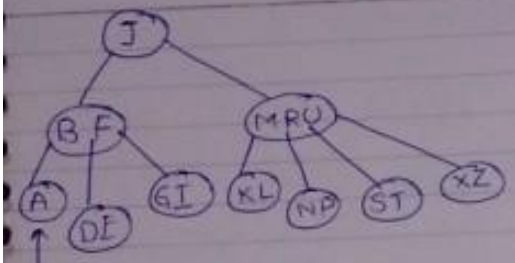
★ If we delete any element which is present in non-leaf node, replacement will be done by children, then if not by children, then it is done by sibling, if not satisfied then merge with parent.

Q.



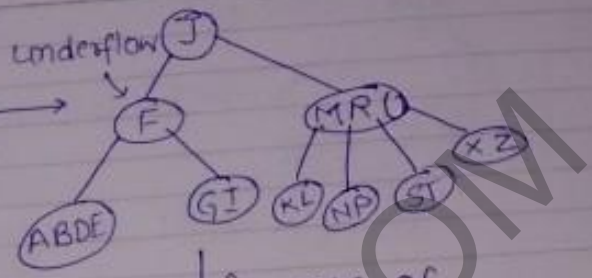
min. keys = 2
min. children = 3
max. keys = 4
max. children = 5



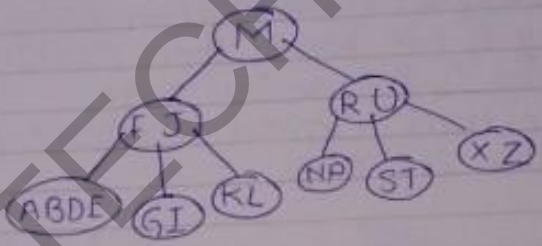


underflow
(now A will ask its sibling, if not satisfied by its sibling, then merge with parent)

[in case of underflow, ask the parent, in case of deletion, replace blindly with children & then perform other op^n]

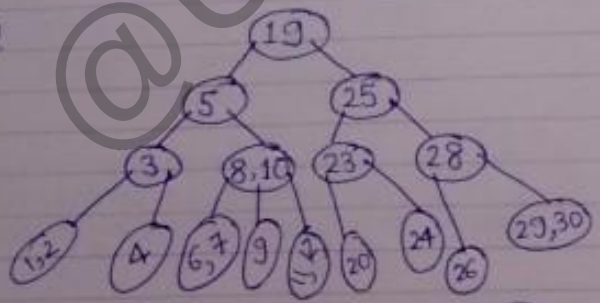


in case of underflow, ask the sibling 1st, (MRU) can give



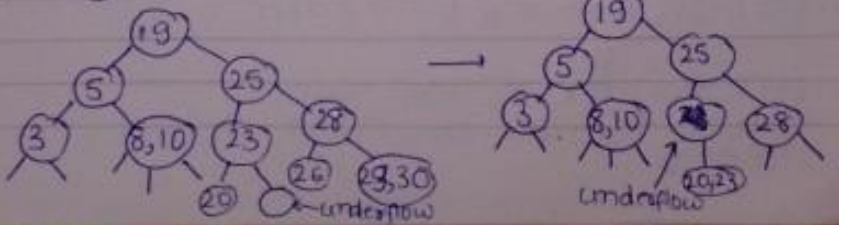
**In case of element deleted from non-leaf node, replace the element (or take the element) from children & then perform op^n of underflow (if there is some underflow in children).
In case of underflow, ask help from siblings, if not then merge with parents.

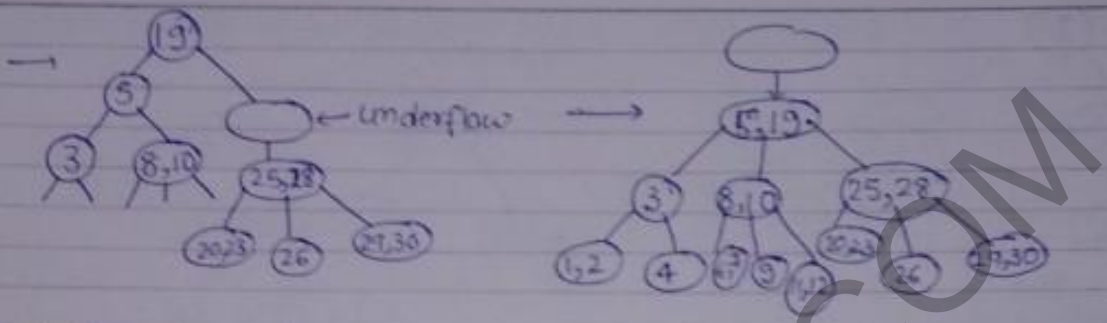
Q.



min-keys = 1
min-children = 2
max.-children = 4
max. keys = 3

Delete 24

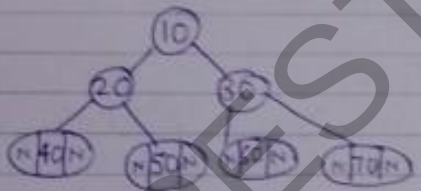




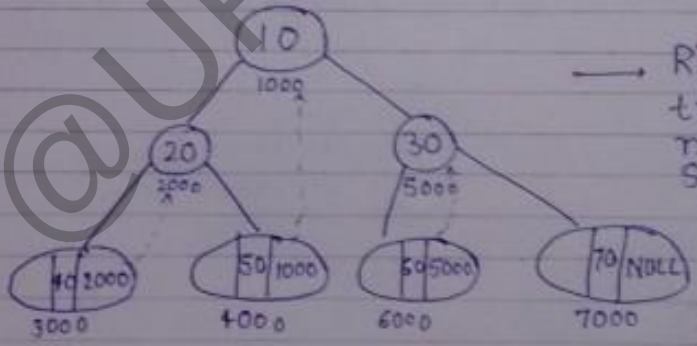
Deletion-Algorithm:-

1. Element is in leaf node
   delete element & check for underflow
   if underflow happened then borrow keys from siblings. If sibling is not ready then combine with the parent.
2. Element is in non-leaf node
   delete that element & replace by inorder predecessor or inorder successor.

Q.



1. In Binary Tree, all leafnodes left nodes Left pointer & right pointer are null.



→ Right In threaded binary tree (right pt. of leaf node contain inorder successor address)

⇓

To find inorder no. need of recursive procedure (no stack req.) using threaded BT.

→ Left-In Threaded Binary Tree

⇓

Left pointer of leaf nodes contain inorder successor address.

32 → children

Q In order 32 B-Tree with height 3, find min. keys & min. nodes

[min]

level	nodes	keys	children
1	1	1	2
2	2	15*2	2*16
3	32(2*16)	32*15	2*16*16
4	2*16*16	2*16*16*15	0

Order m B-Tree with height h contain min. no. of nodes:-

$$1 + 2\times\left(\frac{m}{2}\right)^0 + 2\times\left(\frac{m}{2}\right)^1 + 2\times\left(\frac{m}{2}\right)^2 + \cdots + 2\times\left(\frac{m}{2}\right)^{h-1}$$

$$= 1 + 2\left\{ \frac{1\left(\left(\frac{m}{2}\right)^h - 1\right)}{\left(\frac{m}{2} - 1\right)} \right\} = 1 + 2\left\{ \frac{\left(\frac{m}{2}\right)^h - 1}{} \right\}$$

Stack

Side is open another side is closed.

LIFO or FILO :
Top is variable which contain position of top most elem

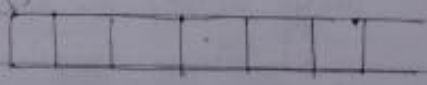
ADT of Stack (what op' perform on data structure)
1. push (stack, e) = stack (return)
2. pop (stack) = element of top.
3. IS empty (Stack) = Boolen.
4. Full (Stack) = Boolen

Implementing stack using array

```
int a[5] = { };
a[10] = 0;
pf(a[10]); [ a[10] is not allocated to you it give garbage ]
```

Segmentation error = Item is not available.

Insertion (push)



push(a, N, TOP)

a: Name of stack or array
Top: of stack
N: Size of stack, array

DS

```
# push(S,N,Top,x)
{
  if(Top=N-1)
  {
  printf("Stack is overflow");        => O(1)
   exit(1) => (abnormal Termination)
  }
   else
  {
     Top++;
     S[Top]=x
  }
}
```

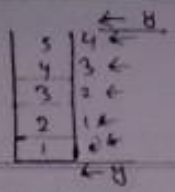
a	b	c	x
0	1	2	x

Top: 3 (insertion)

deletion Top:
--Top = 2
--Top = 1
--Top = 0
--Top = -1

S[Top++]=x [ready to data loss]
S[++Top]=x (✓)

```
pop(S,nTop)
{
  if(Top==-1)
  {
    Pf("Stack is underflow");  => O(1)
    exit(1)
  }
  else
  {
  int y= S[Top]
    pf(y);
    Top--;
  }
}
```

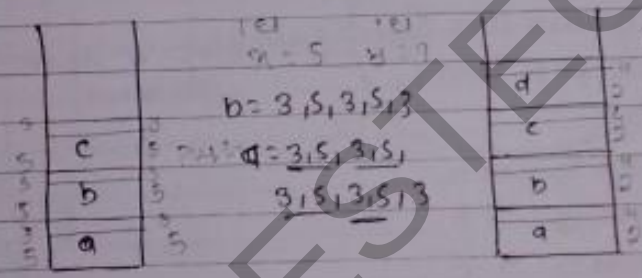


Let S be a empty stack, storing from empty stack n natural no are puted into stack. When they then perform 'n' pop opn. Assume push and pop opn will take 'x' second each and y second elapsed b/w endof such stack and befor the start of next opn.

⇒ Stack life of element 'M' defined as the time elapsed from push(M) to befor the pop(m)

Find average stack life of an element in stack

a) nx(x+y) b) 3y+2x) c) (x+y)-x d) None



life time of d = 4 = 4 (x=2, y=4)

" " c = 0, 2,4, 4,2,4 = 16

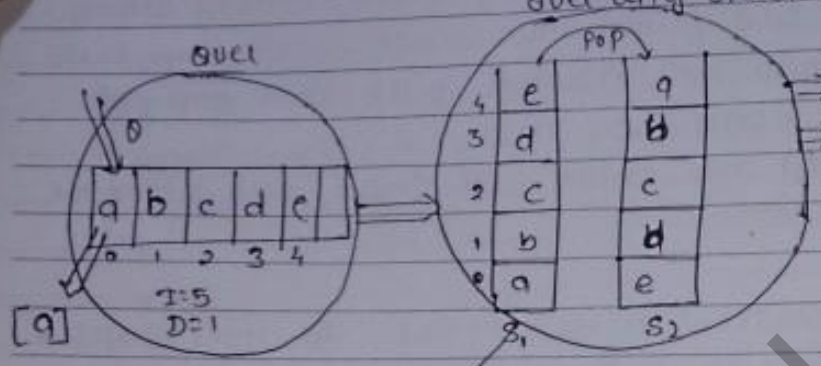
" " b = 0, 2,4, 2,4, 4,2, 4,2,4 = 28

" " a = 0, 2,4, 2,4, 2,4, 4,2, 4,2, 4,2,4 = 40

$$\frac{4+12+28+40}{4}$$

Q) write a c program to implement queue using stack

queue using stack

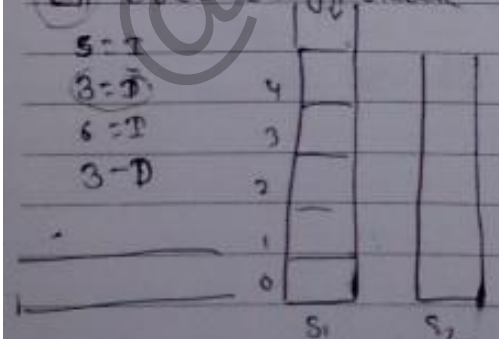


I : 5-push in S_1

D : 5 pop from S_1
5 push into S_2
1 pop from S_2 ⇒ 9 (give 'd' as deleted element)

1 pop from S_2 ⇒ a
2 pop from S_2 ⇒ b
3 pop from S_2 ⇒ c
4 pop from S_2 ⇒ d
5 pop from S_2 ⇒ e

Q) queue using stack



5 : push - S_1
5 : pop : S_1
5 : push - S_2
3 : pop - S_2
6 : push : S_1
2 : pop : S_2
6 : pop : S_1
1 : push : S_2
1 : pop : S_2

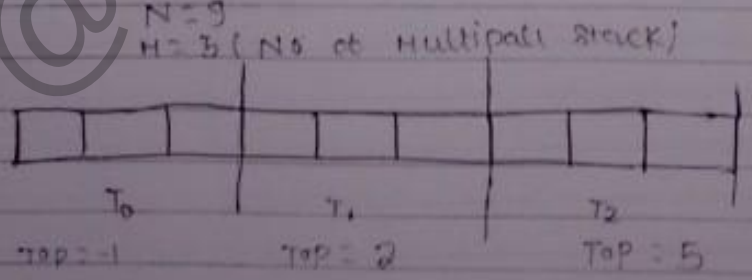
pop : 3 + 3 + 2 + 6 + 3
= 15 pop

22 push
17 pop

program

```
Dequeue()
{
  if (S2 is empty)
  {
    if (S1 is empty)
      print("Queue is underflow")
    else
    {
      while (S1 is not empty)
      {
        x = pop(S1);
        push(S2, x);
      }
    }
  }
}
```

Implementing multiple Stack in single



Ting top: i^{th} Stack: (Max may stack) $\frac{N}{M} - 1$

$\therefore C \times \frac{9}{3} - 1 = -1$

1st stack = $(1 \times \frac{9}{3}) - 1 = 3 - 1 = 2$

2nd stack = $2(\frac{9}{3}) - 1 = 2 \times 3 - 1 = 5$

:

50th stack = $50(\frac{9}{3}) - 1 =$

:

i^{th} stack = $\boxed{i(M/H) - 1}$

Top

* When we say the To is full.

Ans) When top of stack or Top is equal to Top of 1st stack.

When 50th stack is full

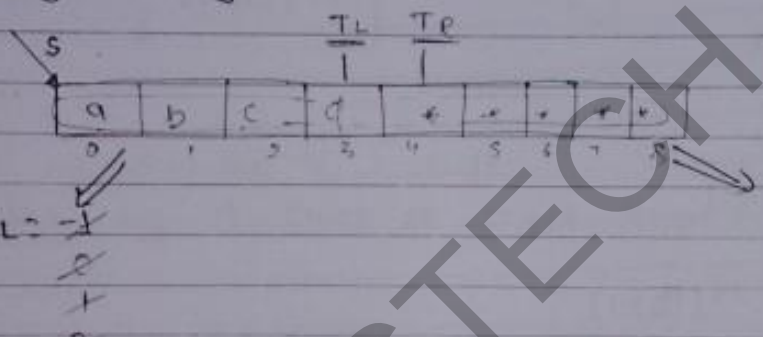
Ans) Top 50th = 51 $(\frac{9}{3}) - 1$ (next stack top)

push opn

```
push (S, N, M, Ti, x)      (Ti : Top of ith stack)
{
    if (Ti == (i+1)(N/M) - 1)
    {
        pf( stack is full);
        exit(1);
    }
    else
    {
        Ti++;
        S[Ti] = x;
    }
}

pop (S, Ti, N, M)
{
    if (Ti == i*(N/M) - 1)
    {
        Pf( stack is underflow)
        exit(1);
    }
    else
    {
        y = S[Ti];
        Ti--;
        return y;
    }
}
```

#) using above c program we can implement multiple stack in single array but it is not efficient is - 1 stack is all other remain stack are empty it is giving stack is overflow.

#) efficient way of implementing multiple stack in single array.



if TL and TR are side by side. then stack will give full condition for efficient use.

#) TL = TR-1 both should side by side
TL = 5 TR = 6

TL = TR-1

Application of Stack

1) Recursion
a) total recursion
b)

1) Recursion
   i) Tail recursion
   ii) Non-tail recursion
   iii) Nested recursion
   iv) Indirect recursion

2) Infix to postfix
3) postfix evaluation
4) Tower - Hanoi
5) Fibonacci Series

6) Write a c program to find min^m element in the given array

Ans

```
void main()
{
   int a[10];
   a[0] = min^m;
   Min^m( a, n)
   {
   if (a[0] > a
```

Recursive

```
# min(a,i,j)
{ if (i==j)
    return a[i];
  else
  { if (a[i]>a[j])
      Min(a,i+1);
    else
      Min(a,i,j-1)
  }
```

Time complexity = O(n).
Space complexity = O(n).

Recursion

T = 1+T(n-1)

2) 1+T(n-1) = O(11)

Q) write a c-program to print the given array in the reverse order.

A)

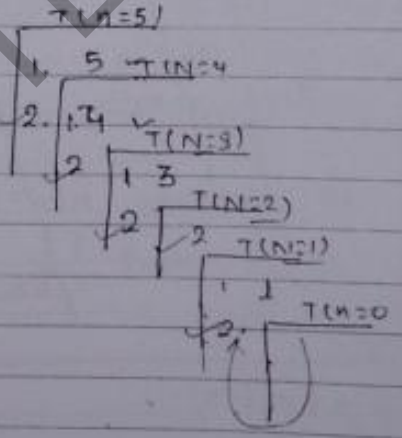
```
pr(a,i,j)
{
  if (i=j)
  return a[i];
  else
  {
    pf("%d",a[j]);
    pr(a,i,j-1)
  }
}
```

T(n) = 1+T(n-1)
= O(n)
Space complexity: O(n);
Time complexity: O(n);

Tail recursion

#

```
TT(n)
{
  if(n<=0) return;
  else
  {
    printf("%d",n);
    TT(n-1);
  }
}
```



1. In the given recursive program, at the recursive function call is there then the function is called tail-recursive program.

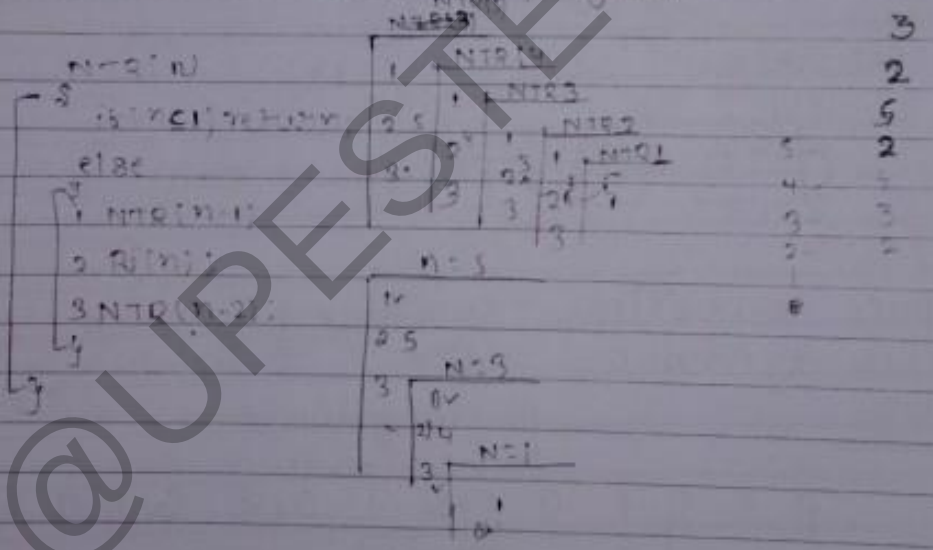
2. The draw back with tail recursive program is we are wasting lot of stack step.

3. The advantage with the tail recursive program is, it is very easier to write equivalent non recursive program with the help of for loop

4. In given recursive program only one recursive program only one recursive is is also at the end then called tail recursive program.

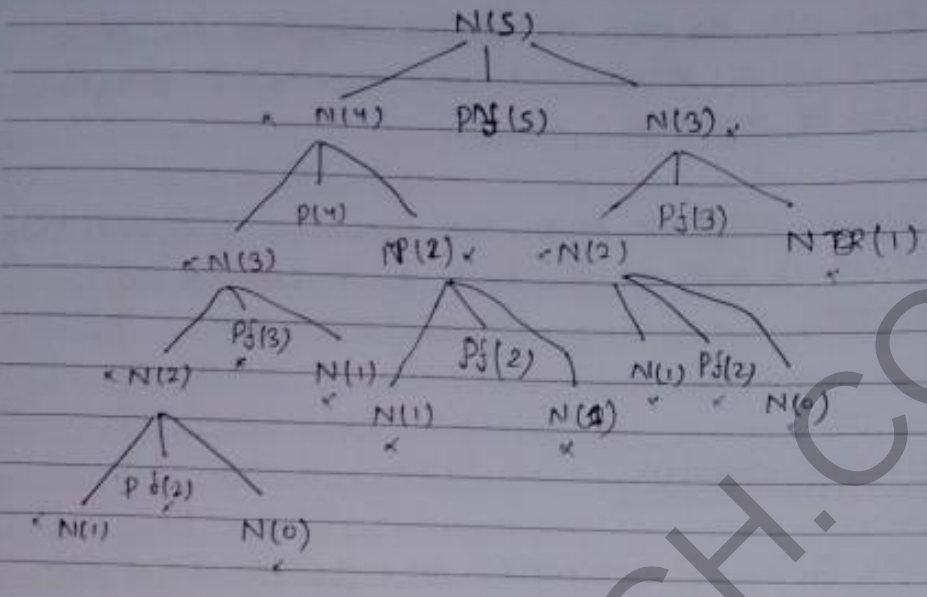
Non tail recursive program

In the given recursive program. After the recursive call there are something to do then recursive program called Non tail recursive program.



2 3 4 2 5 2 3 A

Tree Method



n level binary tree.

2^n

$O(2^n)$

2 3 4 2 5 2 3 ft

[ ] => [with dynamic]

Space complexity: table size + Stack

with dynamic ↓ ↓

prog. How many distinct function call

N + N

$S = O(2N)$

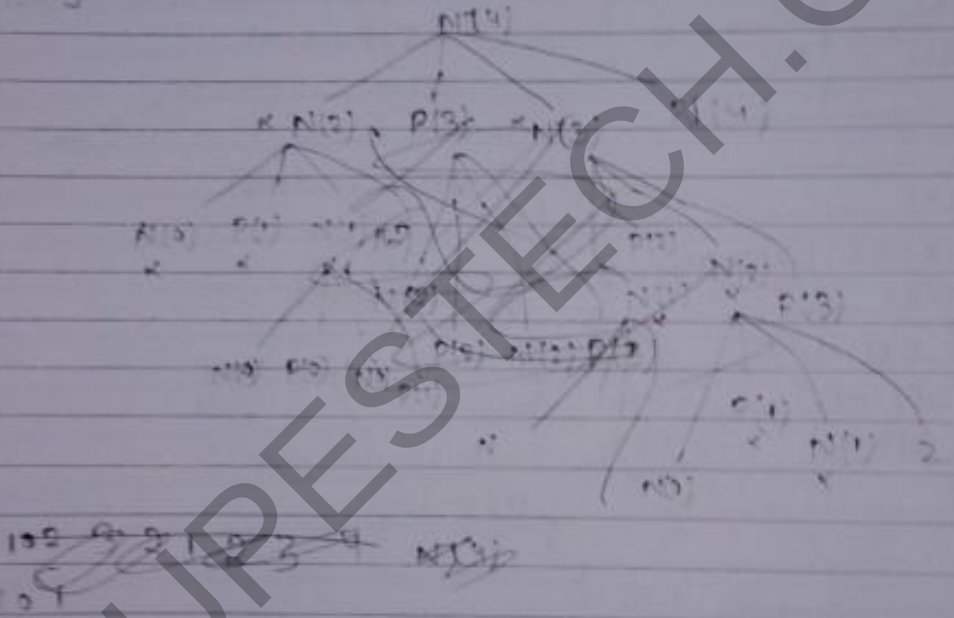
without dynamic $T = O(2^n)$

Space: $O(n)$

Consider the recursive program

```
NTR(n)
{
  if (n<0) return
  else
  {
    NTR(N-2)
    PS(n-1)
    NTR(n-1)
    PS(n)
  }
}
```

NTR(4)



1 0 1 2 3 0 1 2 1 0 1 2 3 4.

Time complexity ⇒ O(n) } dynamic programs
Space complexity ⇒ O(n) }

When we function store in table then it dynamic programing

NOTE: We are not wasting stack space necessarily
⇒ It is very difficult to find non-recursive program with the help of for loop.

⇒

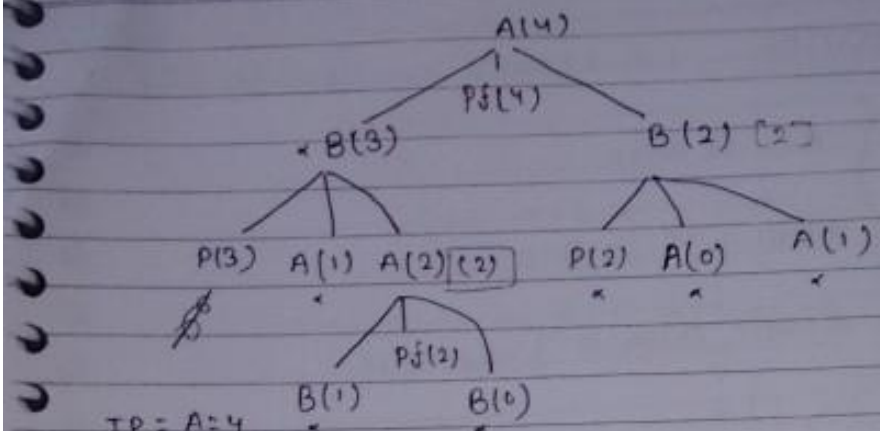
Indirect recursion

```
#1 A()      B()
   {        {
     B()      A()
   }        }
```

```
I/P = A(4)
```

```
A(n)
{
  if(n<=1) return
  else
  {
    B(n-1);
    Pf(n);
    B(n-2);
  }
}
```

```
B(n)
{
  if(n<=1) return
  else
  {
    Pf(n)
    A(n-2);
    A(n-1);
  }
}
```



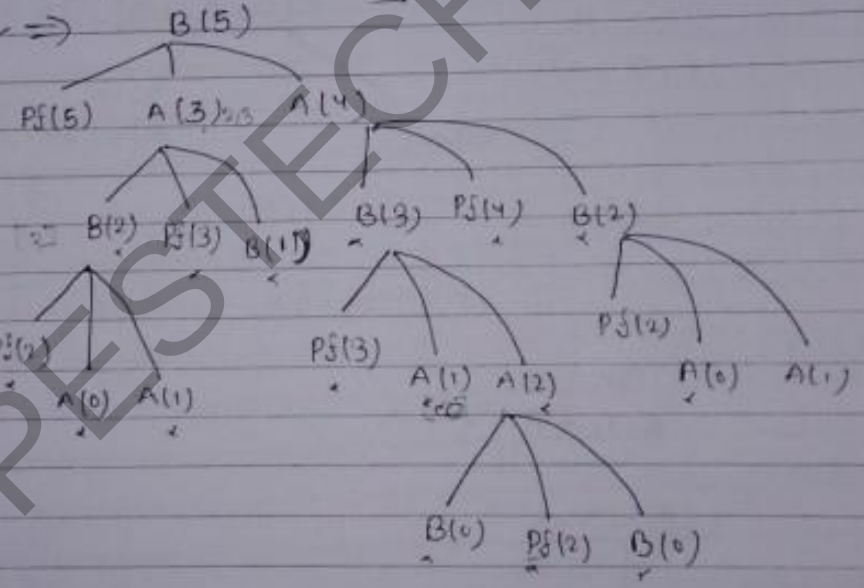
IP = A=4

⇓

3 2 4 2 Ans.

[(IP = B=5) ⇓]

(wrong at work) ⇒



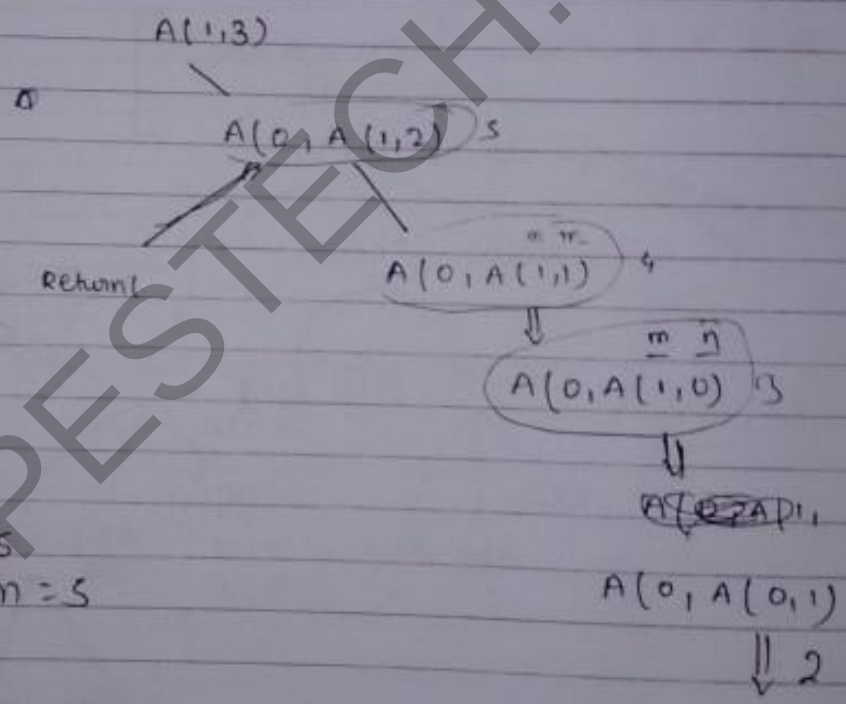
5 2 3 3 2 4 2

If T is A(x,x) : then distance function : n^2 distance

Nested recursion (Ackermann's recursion relation)

$$A(m,n) = \begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } n=0 \\ A(m-1, A(m,n-1)) & \text{other else} \end{cases}$$

A(1,3)



Am = 5
Return = 5

A(2,2)

A(2,2)

⇓

A(1, A(2,1))

A(1,5)

A(1, A(2,0))

A(1,1)

A(0, A(1,0))

A(0,1)

All → A

(1,0)=2
(1,1)=3
(1,2)=4
(1,3)=5
(1,4)=6
(1,5)=7
(1,6)=8

A(2,8)

ISA

A(1, A(2,7))

A(1, A(2,6))

A(1, A(2,5))

A(1, A(2,4))

A(1, A(2,3))

A(1,17) ⇒ 19

(1,15) (1,17)

(1,13) = 15

(1,11) = 13

(1,9) = 11

A(3,3)

⇓

GI All

A(3,3)

⇓

A(2, A(3,2))

⇓

A(2, A(3,1))

⇓

A(2, A(3,0))

A(2,5)

Infix to postfix

Infix: a+b*c
postfix: abc*+
prefix: +a*bc

Note :- In all the three notation operand order can't be change and operator may be change

↑ POS Pribix

Note To evaluate the give postfix expression exactly on scan is enough.
but to evaluate the give infix and pribix many scan are required

Infix: a+b-c (+, -)((priority same))
postfix: abc+c- when same two is in first with
prefix: -+abc but having high priority.

Infix: a+b*c/d↑e↑f-g*h [↑ = higher priority
Associative property
for ↑ it's right to left]

a+bc* def↑↑/+gh*-

pribix -+a/*bc↑d↑ef*gh

Q) infix = e ↑ d - a * b ↑ f / g + h * c / i + J - K

postfix ↑ed ↑bf

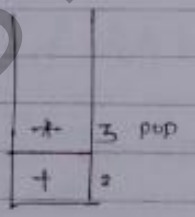
ed↑a - a*bf↑g/ hc*i/+J+K-

prefix: - + -↑ed *a↑bf/g /hc/iJK

POSTfix = ed↑abf↑*g/-hc*i/+J+K-
prefix = -+ -↑ed/*a↑bg/*hc/iJK

Bracket Associativity is ⟵

a + b * c



Post fix

a b c * +

a + b * c ↑ d - e

[when we occur with equal or lower priority then pop higher prior

a b c d ↑ * + e -

a ↑ b ↑ c ↑ d ↑ e

a b c d e ↑ ↑ ↑ ↑ ↑

Lower priority i is coming the ⇒ push.
High to low ⇒ pop.
Left to right ⇒ pop.
Right to left ⇒ push.

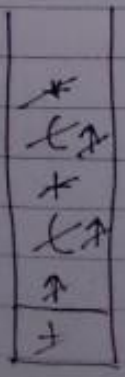
Infix e ↑ d − a * b ↑ f / g + h * c / i + j − k

e d ↑ a b f ↑ * g / − h c * i / @ + j + @ k −

While converting infix to postfix we are using operator stack (only operator are push into stack)

Max^m stack space is = '3'.

a + b * (c + (d * e) ↑ f) ↑ g



a b c d e f ↑ + g ↑ ↑ +

prefix to postfix

	prefix		postfix
1)	a	———	a
2)	+ab	———	ab+
3)	*a+bc	———	abc+*

4) + * / - a b c d ↑ e / - e f g h

a b - c / d * e f - g / h ↑ +

⇒

* a / b - c ↑ ↑ ↑ g f h + e * f, g, (Try at home)

a * b / c - g f ↑ h ↑ e f, g, * + ↑ b g * - / *

Another way start from right.

* D

* a / b - c ↑ ↑ ↑ g f h + e * f, g,

Date

Q) a b + c d - *

find out infix (postfix ⟶ infix)

Ans (a + b) * (c - d) = a + b

7) a * b + c d / +

a * b + c / d Ans.

postfix evaluation

Infix = 2 + 3 * 4 + 5 ↑ 1 / 2 - 3 * 6

postfix

2 3 4 * + 5 1 ↑ 2 / + 3 6

Max^m Stack size = 2.

*
+

* / - *

Total string length = 'N'.

$\lceil \frac{N}{2} \rceil$ = operand $\lfloor \frac{n}{2} \rfloor$ = operator.

Evalution ⇒ 2 3 4 * + 5 1 ↑ 2 / + 3 6 * -

pop

4
3
2

OP₂ pop() = 1st = 4 × = 12

+ OP₁ pop() = 2nd = 3

12 5 5

14

2nd step

12 = 14

2

3rd

1 = 5¹ = 5

5

#

	6
2	3 18
14	14

14 + 2 = 16

* pop: $OP_2 = 6$ $OP_1 = 3$ = 18

- pop: $OP_2 = 18$ $OP_1 = 14$

16 − 18 = −2 Ans

#

```
int i=0;
x = postfix[i]
while ( x != 10)
{
  if ( x is operand)
    { push(x);
      i++
    }
  else
    {
      OP2 = pop();
      OP1 = pop();
      r = OP2 * OP2 ;
      Push(r);
```

O(n) = Time complexity

operand Stack

Max^m size = $\lceil \frac{n}{2} \rceil$

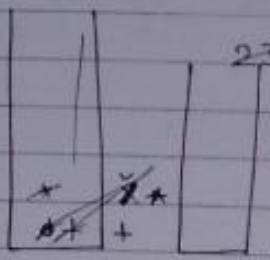
Stack

NOTE while evaluating the given postfix expression we are using operand stack only operand are pushed into stack.

Date

Infix: 2 + 3 * 4 - 5 + 6 / 7 * 4 ↑ 5

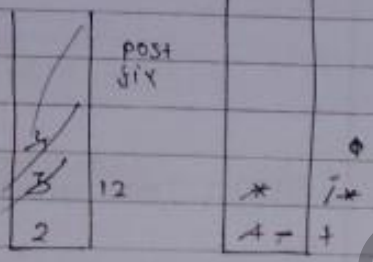
postfix



2 3 4 * - 5 6 7 / 4 * + ↑ 5

postfix: 2 3 4 * 15 - 67 / 45 ↑ * +

evaluation ⇒



[Max^m Stack size: '2']

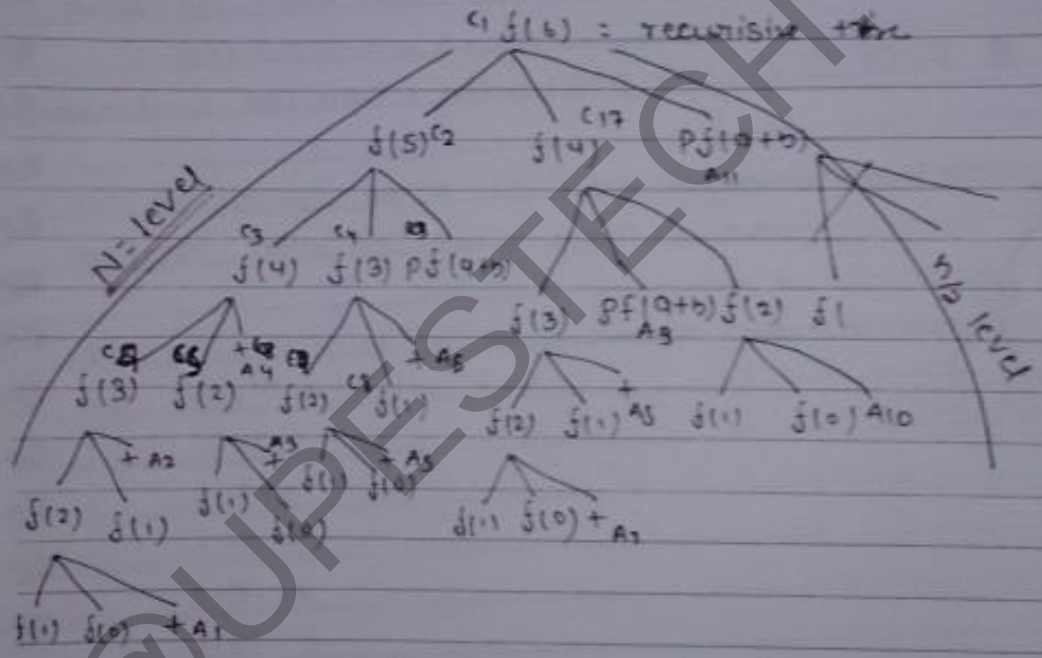
Fibinocci Series

n	0	1	2	3	4	5	6	7	8
fib(n)	0	1	1	2	3	5	8	13	21

$$fib(n) = \begin{cases} 0 \text{ if } & n=0 \\ 1 \text{ if } & n=1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

combinely: n if n=0 and n=1

```
fib(n)
{
 if ( n==0 || n==1)
 return n;
else
   {
     a = fib(n-1);
     b = fib(n-2);
     return a+b)
     }
}
```

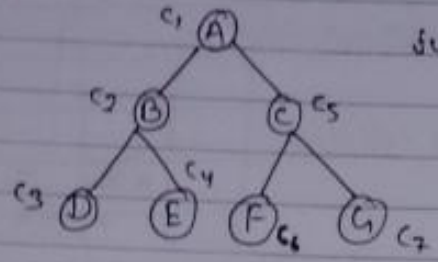


Make tree and give function nos

Max^m Stack Size No of lable.

= D(A) => wait

Function called in programming language = preorder.



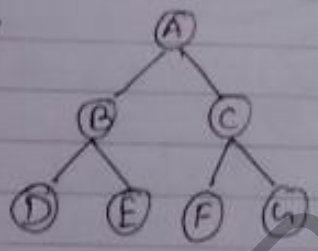
function call sequence.

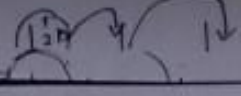
A B D E C F G = function cal sequence => preorder

function Evaluation

D E B F G C A => post oder (Function evaluation)

Q.





Date

Q1) In fibbonici of 'N' after how many function call first addition will perform = N+1 function call.

Ans Draw recursive tree it may be for any recursive program. and give fun^c call order and count.

Q2) In fibinoci of 'n' is there any adition after last function call.

Ans=) yes (and also N/2 addition may be there)

After C25 = A10, A11, A12.

Q3) In fibinoci of 'n' is there any function call after last addition.

Ans=) No after A12 total program is over.

Q4) In fibinoci of 'N' many function call are there.

Ans) Max^m function $O(2^n)$

$f(0) = 1$

$f(1) = 2$

$f(2) = 3$

$f(3) = 5$

$f(4) = 9$

$f(5) = 15$

$f(6) = 25$

$f(7) = 41$

⋮

$f(n) = f(n-1) + f(n-2) + 1$

In fibinaci of 'n' how many addition will be there.

No of addition

=) f(0) = 0;

=) f(1) = 0;

=) f(2) = 1

f(3) = 2

f(4) = 4

f(5) = 7

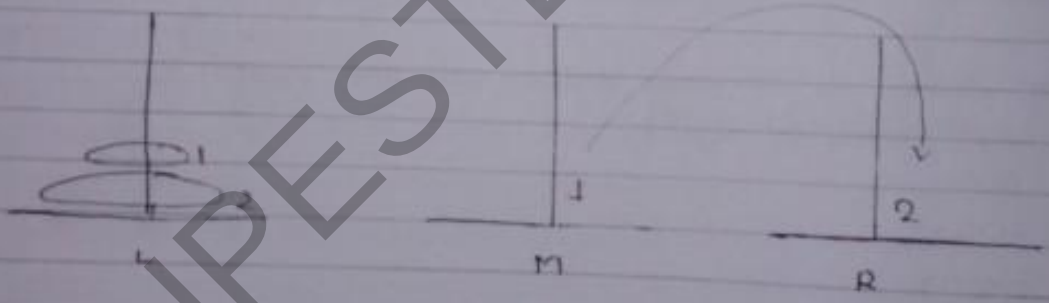
f(6) = 12

f(7) = 20

$T(n) = 1 + T(n-1) + T(n-2)$

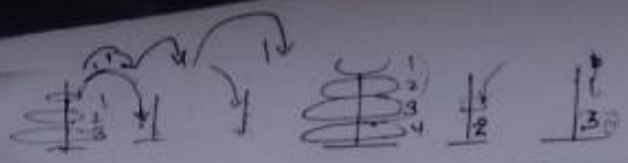
$f(n) = f(n-1) + f(n-2) + 1$.

Tower of Hanoi



MOVE:

1) L - M
2) L - R
3) M - R

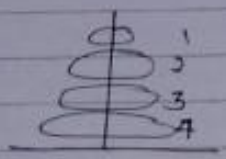


N=3

L - R
L - M
R - M
L - R
M - L
M - R
L - R

N=4

L-R
L-M
R-M



N=4 ⇒

Top 3 have to send Middle = ie take 7 Step.
Ab move L - R (4 No block)
then 3 block at middle take 7 step Max total Step '7'.

1) L-M	6) R-M	11) R-L
2) L-R	7) L-M	12) M-R
3) M-R	8) L-R	13) L-M
4) L-M	9) M-R	14) L-R
5) R-L	10) M-L	15) M R

N=5

1) Move Top 4 from L to M = 15
2) Move Top 1 from L to R = 1
3) Move Top 4 from M to R = 15

} = 31 Step.

TOH(n, L, M, R)

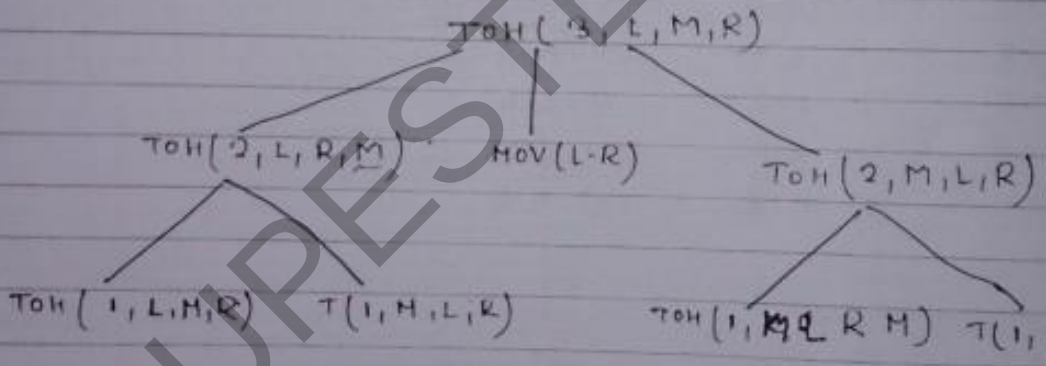
L → Source, R → Destination

```
{
  If (n==0) return
  else
  {
    TOH (n-1, L, R, M)
    Move (L → R)
    TOH (n-1, M, L, R)
  }
}
```

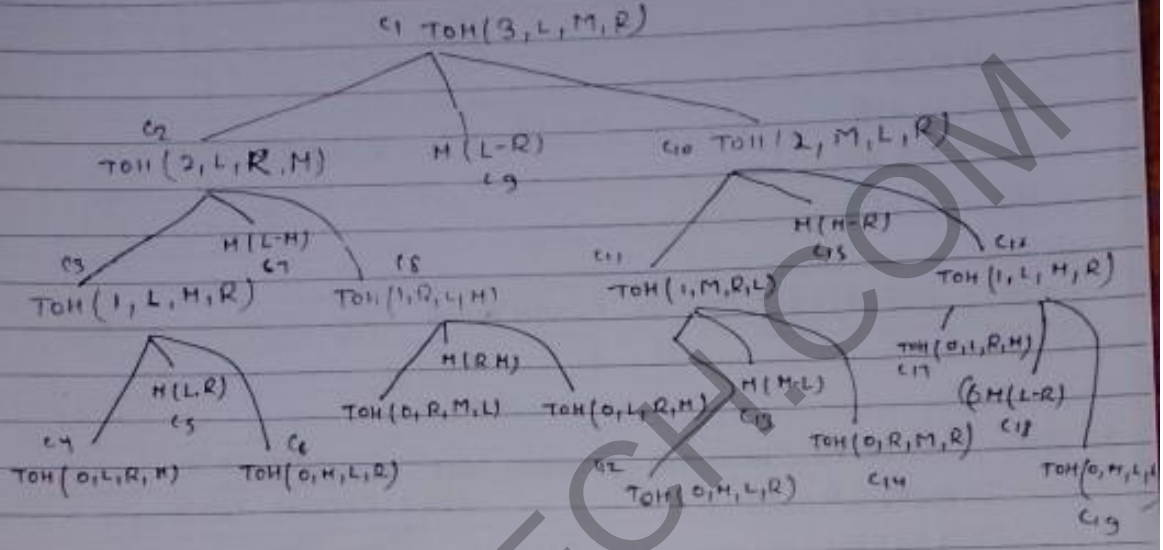
$$T(n) = T(n-1) + 1 + T(n-1)$$

$T(n) = 2T(n-1) + 1$ (No of moves)

$T(n) = 2T(n-1) + C$ = time complexity



Recursive tree for TOH



Q1 In TOH(n) after how many function call move is take place

Ans) N+1 function + [After C4 there is M]

Q2) TOH (n) is there any move after last function call

Ans) No move (last)

Q3) In TOH is there any function call after last move

=) after M1 there is [C15 (last function call)

Q4) In TOH of '3' after how many fuction there is MOV for M1 M → R

Ans) It is 6th move. (befor it C12 fuction call)

Q) In TOH of 'N' how many function call are there

No' of function call

TOH(0) = 1

TOH(1) = 3

TOH(2) = 7

TOH(3) = 15

TOH(4) = 31

TOH(5) = 63

$T(n) = 2T(n-1)+1$

Q1 In TOH of 'N' how many moves are there:

TOH(0) = 0

TOH(1) = 1

TOH(2) = 3

TOH(3) = 7

TOH(4) = 15

TOH(5) = 31

⋮

$T(n) = 2T(n-1)+1$ $\quad$ $T(n) = 2^n - 1$

Time $T(n) = O(2^n)$

Space: No' of level $O(n)$

Dynamic programming

T(n - - -)
|
(n-1) (3) (2) (1) = O(n+1)

T = O(n)

Space complexity = Stacksize + diminat function
= n + 6n
= 7n

Chapter 8

Q5) C — Row-Major Order

char a[100][100]
Base address = '0'
a[40][50] = 0 + (40)(100) + (50-0)
= 4000 + 50
= 4050

Q6) a 7) a 8) c 9) a 10) b 11) a 12) c 13) 14) 15) d

Q8)

A =
a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

B =
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

11) int a=10;
int *b=a

b =
1	5	9	13
0	0	10	14
0	0	11	15
0	0	0	16