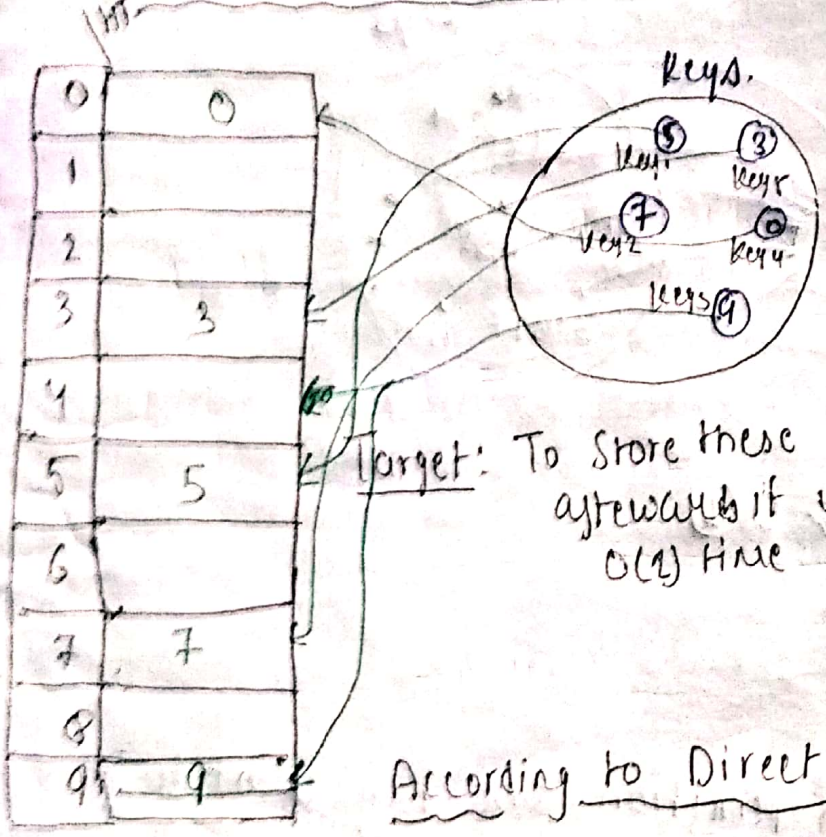
Hashing:

① It is one of the searching Technique.
② Worst Case Searching Time Complexity - O(1)

Direct Address Table:



Host-Table
(Vertical array)

Forget: To store these keys inside the table. afterwards if we search so it will take O(1) time

According to Direct Address Table:

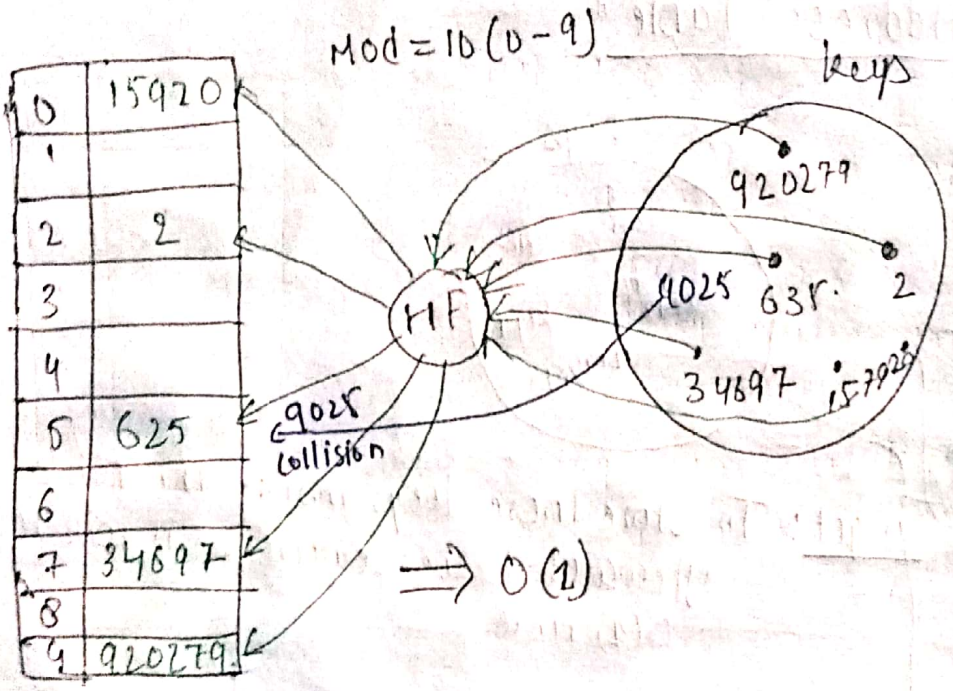
① Key Itself is the address without any calculation or without any modification
② Searching Time = O[1][EC]

② The Drawback with the D.A-T. is even the No. of keys are very less but if one of the key is very large 2^{1000} then there is need of Hash table 2^{1000} of size.

(4) To Eliminate above drawback. we are going to the Hash function.

Hash function:

probe = search = a theoph



Hash Table (HT)

After Applying Hash function. one key point to same hash Index called Collision.

Disadvantage/Drawback: Collision.

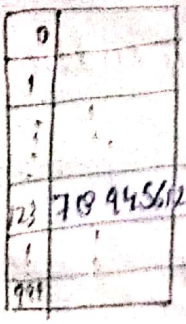
Types of Hash function:

(1) Division Module Method.

Ex①: M = 1000 (0-999)

Key = 789456123

HF(key) = key Mod M. = 123.



0-7

Ex 2: M = 8 (2^3)

Key = 10101010101010

HF(key) = Key Mod M

16 bit LSB

= 1010101010101100 Mod 2^3

= 100 (in 0-7)

1110010010100101010 - - - - - 1010100 Mod 2^3

= 100 ✓ 16 bit hash bit

If bit is 10 or 16 bit it always give LSB 3 bit so the major drawback.

Ex 3: M = 2^k

1000110101011101001011100 Mod 2^k = 1011100

LSB-k bit

Counter possible if M is power of 2

NOTE: Drawback of Division Module Method

① Dont pick M values exactly powers of 2 because $M = 2^k$. then Hash-function of key is = LSB key bit always.

HF(key) = key Mod M.

② Pick M value which is the Prime Number. but it cannot be very close to powers of 2.

Ex: 511 Prime No

which is very close to power of 2 = 512

[5 ✗ Prime 10 ✗ 3 Prime

701 ✓ Prime. It is far from power of 2]

② Mid-Square Method:

① First Square
② Then Mi' take Middle

Dest compare ① ②③

M = 1000 (0-999)

Key = 84526.

HF(key) = $(84526)^2$

= 7144644676 (Even No so take any So we cannot find the middle so take any two

└ but take 3 digit coz Max index slot is in 3 digit. So that we can store. Many digit.

└ If we take only 6. so slot will be 0-9 So Many Collision will be there. (Majority address same)

0	
!	
469	84526
!	
999	

Here every digit will be participated in Division Modulo Method only LSB 3 digit will be participated.

What I compare previous 2 Method.

③ Digit Extraction Method:

M = 1000 (0-999)

Key: 789456123 (9 digits.)

HF(key) = 963

963 — 789456123

3rd 6th & 9th will be extracted

7 8 (9) 4 5 6 1 2 (3)
1 2 (3) 4 1 6 7 8 (9)

In exam. it will be given. which will key will be extracted.

3. Folding Method: Perform Bar as compare with ① 2 ③

① Fold Boundary Method:

M = 1000 (0-999)

key = 789456123

Take 3 digit from first & last boundaries. and the Add

HF(key) = 789
123
912 — If carry come

912 — 789456123

141 2
141
+2
143 store in.

Ex = Where Collision will occure

789603123

789
123
912 = 912 — 789603123
Same slot different values
(Collision)

(ii) Fold Shifting: Here every 3 elements will be fold.

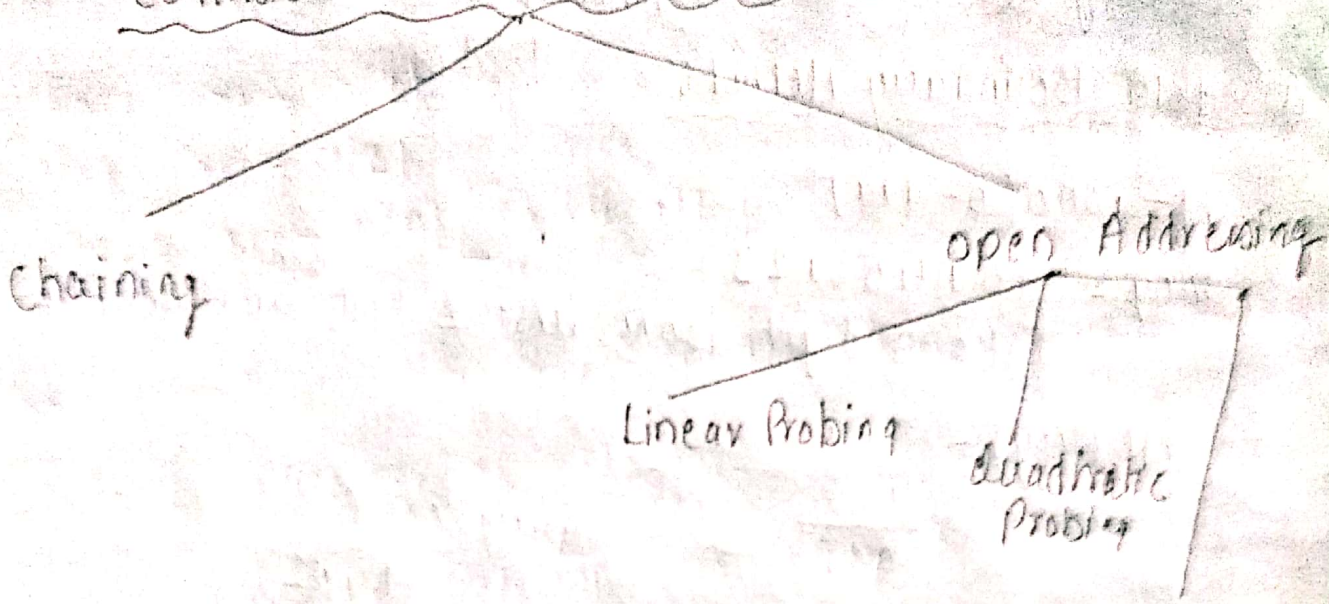
789 456 123

Start from Right side

789
456
123
1368

136
+8
144 ← 789456123

Collision Resolution Techniques



① Chaining:

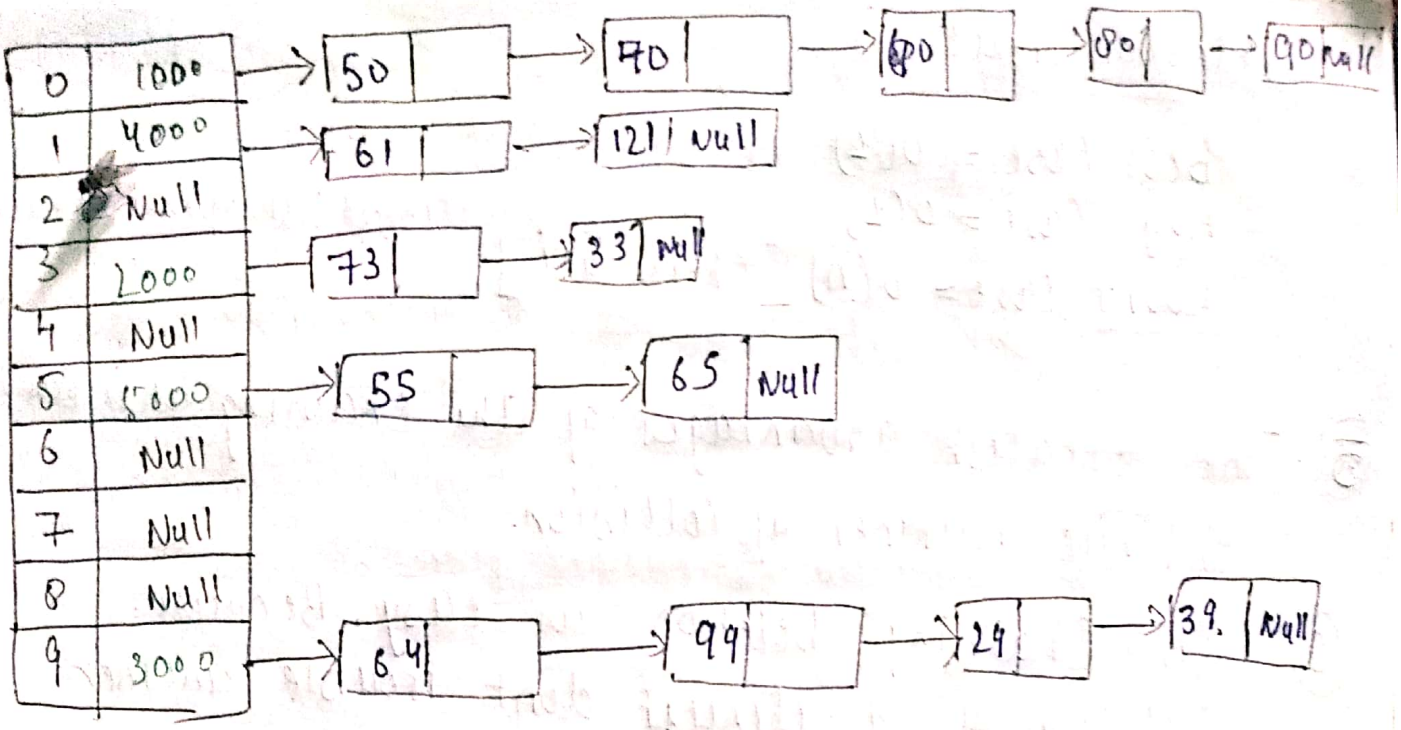
M = 10 (0-9)

HF(K) = K Mod M

Keys = 50 69 73 70 99 40 33 29 61 60 55
65 39 90 121

#1 CRT = Chaining. (Means every key outside

0	50	70 40 60 90
1	61	121
2		
3	73	33
4		
5	55	65
6		
7		
8		
9	69	99 29 39



Max Chain = 5 Min = Null

Array of pointers we will take to implement.

In worst case it will give O(n) but ~~it is ver~~ in very real incident because we use best hash method, so which is not possible. ~~but in~~ very rear chance to

Greatest thing about this: We can store infinite elements (key)

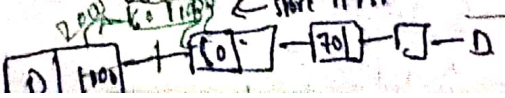
NOTE:

① Lot of space we are wasted. outside in the form of linked list even the space available inside.

② Searching time = O(1) [BC, Avg. Case]
= O(n) [Most probably all keys will go to same slot. but it very rear] [WC]

③ Insertion Time = O(1) [BC, AC, WC]

Worst Case = O[1] If new key come placed at first no need to insert at last



④ Deletion Time:

Best Case = O(1)
Avg. Case = O(1)
Worst Case = O(n) [Check all] in linked list of that address.

⑤. The greatest advantages of the chaining is it can handle Infinite Number of collision..

⑥ In Chaining Deletion is easy.. Because. Deletion of 1 element dont trouble another element.

Open-Addressing:

Linear-Probing: (also called closed Hashing) *

Ex: M = 10(0-9).
HF(K) = K Mod M.
Keys = 55, 99, 65, 36, 49, 40, 85, 39, 88.
CRT = L.P.

$$LP(Key\ i) = (HF(Key) + i) \bmod M.$$
$$\forall \in (0, 1, 2, \ldots M-1)$$

1st i=0
2nd i=1
3rd = 2

M = 10(0-9)

LP(55,0) = (HF(55) + 0) Mod M
= 5 + 0 = 5

LP(99,0) = (9+0) = 9

LP(65,0) = (5+0) = 5 — collision
LP(65+1) = 5+1 = 6 ...

HT	
0	49
1	40
2	39
3	88
4	
5	55
6	65
7	36
8	86
9	99

LP(36,0) = (36+0) mod M = 6+0 = 6

LP(36,1) = (36+1) mod M = 6+1 = 7 - collision

LP(49,0) = 9+0 = 9

LP(49,1) = 9+1 = 10 = 0 - collision

LP(40,0) = (40+0) mod M = 0+0 = 0

LP(40+1) = (0+1) = 1 - collision.

LP(85,0) = 5+0 = 5

LP(85+1) = 5+1 = 6 collision

LP(85+2) = 5+2 = 7 collision

LP(85+3) = 5+3 = 8 = collisio

LP(39,0) = (39+0) mod M => 9+0 = 9

LP(39+1) = 9+1 = 10 = 0.

LP(39+2) = 9+2 = 11 = 1 3-C.

LP(39+3) = 9+3 = 12 = 2

LP(88,0) = 8+0 = 8

LP(88,1) = 8+1 = 9

LP(88 2) = 8+2 = 10 = 0

LP(883) = 8+3 = 11 = 1 5-C

LP(884) = 8+4 = 12 = 2

LP(885) = 8+5 = 13 = 3

Ans: M = 10(0-9)

HF(K) = K mod M

CRT = LP

After inserting 6-keys Table shown -below.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

① Correct Insertion Sequences?

a) 46 42 34 52 23 33

b) 34 42 23 52 33 46

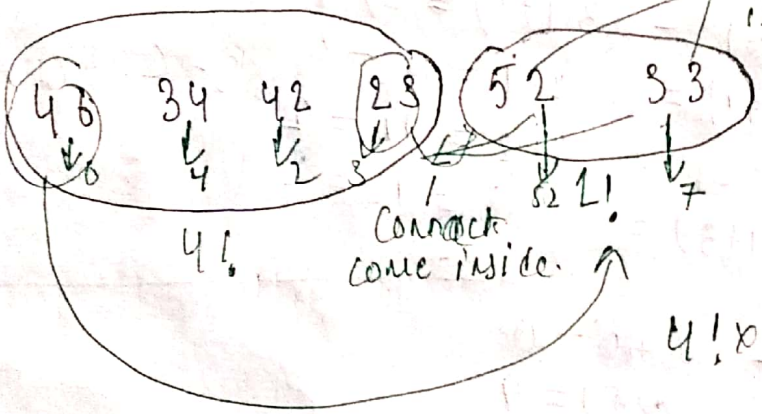
✓ c) (46 34 42 23) 52 33

d) 42 46 33 23 34 5

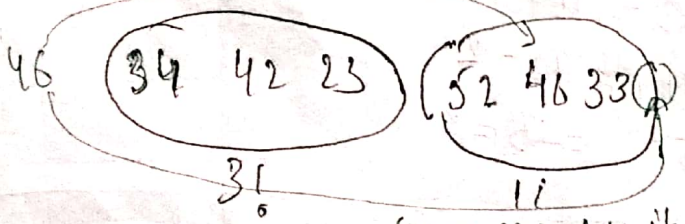
Insertion Sequence possible?

a) 10 (b) 20

(c) 30 (d) 40



4! × 1! = 24



= 6

Total = 24 + 6 = 30

Whenever no collision. many possibility can be found..

② Quadratic Probing:

M = 10 (0–9)

HF(k) = k mod M

Keys = 55, 99, 85, 36, 99, 40, 85, 39, 88

CRT = OP If Given $c_1 = 1, c_2 = 1$

$$OP(key, i) = (HF(key) + c_1 i + c_2 i^2) \bmod M$$

If c_1 & c_2 are not given

then = $(HF(key) + i^2) \bmod M$.

HT

0	40
1	99
2	.
3	.
4	.
5	55
6	36
7	85
8	88
9	99

$OP(55,0) = 5 + 1.0 + 1.0^2 = 5$

$OP(99,0) = 9 + 1.0 + 1.0^2 = 9$

$OP(65,0) = 5 + 1.0 + 1.0^2 = 5$

$65,1 = 5 + 1.1 + 1.1^2 = 7$ — 1-C

$OP(36,0) = 6 + 1.0 + 1.0^2 = 6$

$OP(49,0) = 9 + 1.0 + 1.0^2 = 9$

$OP(49,1) = 9 + 1.1 + 1.1^2 = 11 = 1$ — C

$O(40,0) = 0$

$O(85,0) = 5 + 1.0 + 1.0^2 = 5$

$= 5 + 1.1 + 1.1^2 = 7$

$5 + 1.2 + 1.2^2 = 11 = 1$

$5 + 1.3 + 1.3^2 = 7$

$5 + 1.4 + 1.4^2 = 5$

$5 + 1.5 + 1.5^2 = 5$

$5 + 1.6 + 1.6^2 = 7$

$5 + 1.7 + 1.7^2 = 1$

$5 + 1.8 + 1.8^2 = 7$

$5 + 1.9 + 1.9^2 = 5$ — 10-C

QP(39,0) 39 not assigned.

{10-C

QP(88,0) = 8

Total = 22 collision

③ Double Hashing:

M = 10(0-9)

Hf1(K) = K Mod M. Hf2(K) = 1 + (Key Mod(M-2))

Keys = 55, 99, 65, 36, 49, 40, 85, 39, 88

CRT = DH.

DH(key, i) = (Hf1(key) + i. Hf2(key))

{∀i ∈ 0 1 2 3 ... M-1}

HT	
0	40
1	49
2	
3	85
4	
5	55
6	36
7	65
8	88
9	99

DH(55,0) = 5 + 0.Hf2(55) = 5

DH(99,0) = 9 + 0.Hf2(99) = 9

DH(65,0) = 5 + 0.Hf2(55) = 5 - C

DH(65,1) = 5 + 1.Hf2(65)

65 Mod(8) = 1

5 + 1 + 1 = 5 + 2 = 7

DH(36,0) = 6 + 0 = 6

DH(49,0) = 9 + 0.2 = 9

= 9 + 1 ∈ (1+U)

= 9 + 2 = 11 = 1.

DH(85,0) = 5 + 0·6 = 5
5 + 1·6 = 11 = 1
5 + 2·6 = 17 = 7 ← 3-C
5 + 3·6 = 3

39,0 = 9 + 0·8 = 9
= 9 + 1·8 = 17 = 7
10-C — 9 + 2·8 = 5 ← 14-C
9 + 3·8 = 3
9 + 4·8 = 1

Ex ② M = 10(0-9)
HF(K) = K MOD M
Keys = 53, 61, 70, 92, 90, 50
CRT = L.P.

0	70
1	61
2	92
3	53
4	90
5	50
6	
7	
8	
9	

LP(90,0) = 0 + 0 = 0
0 + 1 = 1
0 + 2 = 2
0 + 3 = 3
0 + 4 = 4

LP(50,0) = 0 + 0 = 0
0 + 1 = 1
0 + 2 = 2
0 + 3 = 3
0 + 4 = 4
6 + 5 = 5

Major Drawbacks also starting from 0 index of 4 afterwards. (Primary clustering problem).

① We are probing 1 by 1 slot or 1 slot after another

② Searching time = $O(1)$ [BC]
= $O(M)$ [WC]

③ Primary Clustering: If two-keys contain same starting hash address then they will follow same path unnecessarily in linear manner. Because of this reason average searching time will increase this problem is known as primary clustering.

'Linear probing' is having the drawback of pt primary clustering, average searching time is equal to (Avg. ~~Access~~ Searching time more)

Drawback..

Avg. Searching Time = $1 + 2 + 3 + \cdots + n$

(1 will take 1 – unnecessary path.
2 have to wait for 1 unnecessary path.)

$= \frac{M(M+1)}{2} = \frac{M+1}{2} = O(M)$

for all $\frac{2}{M}$

④ Insertion Time: = $O(1)$ [BC]
= $O(M)$ [WC, AC]

⑤ Deletion Time: = $O(1)$ [BC]
= $O(M)$ [WC, AC]

⑥ Deletion Difficult because 1-element Deletion trouble other elements But we can manage with special Symbol ↓

If more $ then re-hash again.

Ex(2): M = 10(0-9)

HF(k) = k MOD M.

Keys = 53, 61, 70, 92, 90, 50.

0	70
1	61
2	92
3	53
4	
5	
6	90
7	
8	
9	

90

50

OP(90,0) = 0 + 1.0 + 1.0² = 0
= 0 + 1.1 + 1.1² = 2
= 0.1.2 + 1.2² = 6
3 Comparison

OP(50,0) = 0.1.0 + 1.0² = 0
= 0 + 1.1 + 1.1² = 2
= 0.1.2 + 1.2² = 6
= 0.1.3 + 1.3² = 12 = 2
⋮

It will take more then 3 no Comparision! Min:-3 So redundancy there. unnecessary comp (Secondary Clustering)

① Secondary Clustering: If two keys contain same starting has Address they both will follow same path. unnecessarily in Quadric manner because of this reason. Average searching time will increase. this problem is known as Secondary clustering.

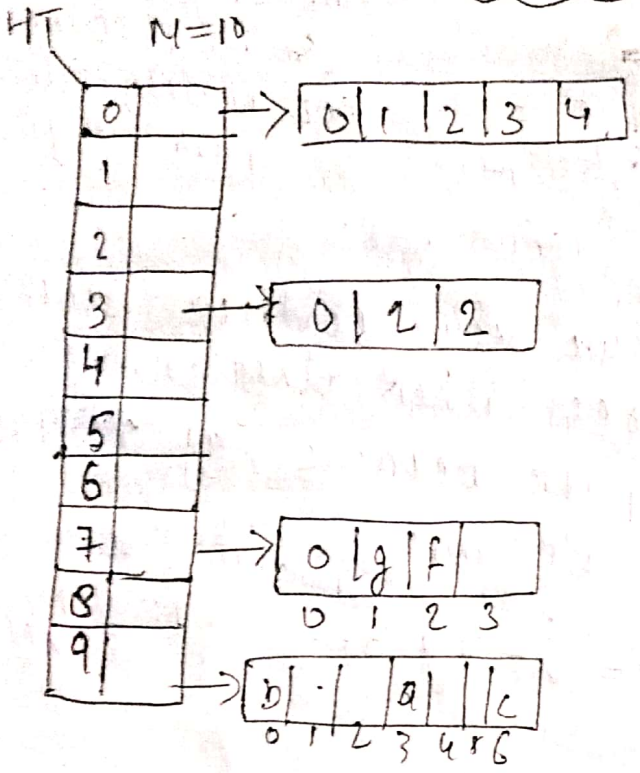
② Quadratic Probing is having the draw-back of Secondary Clustering.

② Searching Time = O(1) [BC]
O(N) [WC]

16 | Sept-2016

Using perfect Hashing you will get worst case time-complexity. O(2).

Perfect Hashing



$h_0(d) \rightarrow 0$
$h_0(e) \rightarrow 0$

$h_1(d) = 4 \quad h_1(e) = 0$

$h_0(a) \rightarrow 9$
$h_0(b) \rightarrow 9$
$h_0(c) \rightarrow 9$

$h_a(a) = 3 \quad h_a(b) = 0 \quad h_a(c) = 6$

$h_0(f) \rightarrow 7$
$h_0(g) \rightarrow 7$

$h_7(f) = 2 \quad h_7(g) = 1$

Time complexity :- [O(1)] [EC]

for every slot more than one hash function & hash table are there.

Universal hash function is independent from key called Universal hash function

Uniform hash function gives priority to every slot equal & can go to any slot, equal probability

M = Number of slots

N = Number of keys

M = slots = N-keys

1 slot = $\frac{N}{M}$ keys/slot (In 1 slot how many)

or

Average

or

Load factor (α)

Number of elements per slot

Load factor (α) for Chaining. $0 \leq \alpha \leq \infty$

Open Addressing

$0 \leq \alpha \leq 1$

① The expected Number of probes in an unsuccessful search in open Addressing Technic $= \frac{1}{1-\alpha}$

② For Sucessful Search $= \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

Programming

① Scope of a Variable

(i) Static Scoping.

(ii) Dynamic Scoping.

Ex ①

```
int a = 5;          // allocate here at compile time (static area)
Main()
{
  int a = 10;       // allocate at run time (stack area)
  B();
}
B()
{
  int a = 20;
  C();
}
C()
{
  int a = 30;
  D()
}
D()
{
  int a = 40;
  Pf(a);
}
```

Local variable memory will create at run-time.
· If there is one function G() No one can call. In that function so many declaration. So what we need of creating memory for them. So that's the reason. Local variable allocate at run-time.

Static are is a small area. If you didn't assigned any value. By default it will zero instead of garbage value. (Small area you can clean. But delhi you can not clean).

Compiler ⇒ HLL to LL lang.
(translator / converter).
Processor only understand Low-Level-lang.

a + b ⟶ ADD R0 R1
HL. LL.
(User Convinience) (CPU Convinience).

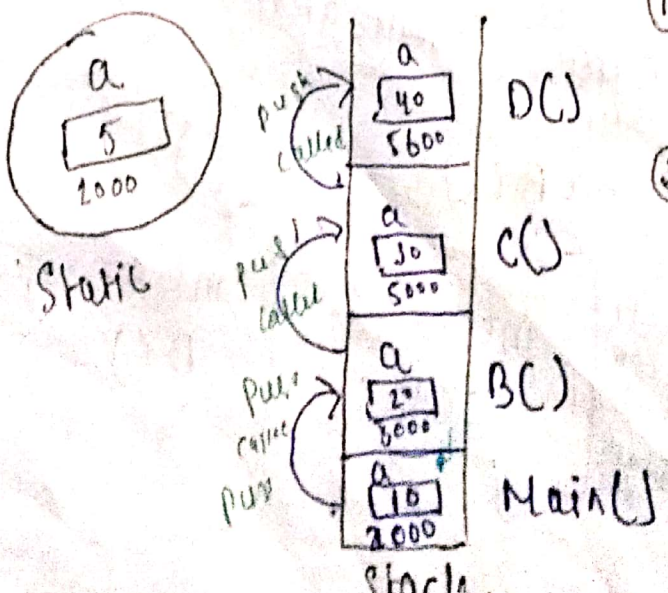
① First he check or verify it is my program or not. It convert HL to LL after checking syntatic & semantic are valid or not.

Logical error : Semantic error
Physical error : Syntax "

Rule :

① Within the function you cannot declare data variable with same name.
② Without declaring a variable you cannot use it.

After Compilation over:



Scope problem comes if there is no decleration of local variable & you are calling a function. Compiler confused which variable value should a print.

How to resolve the scope problem.

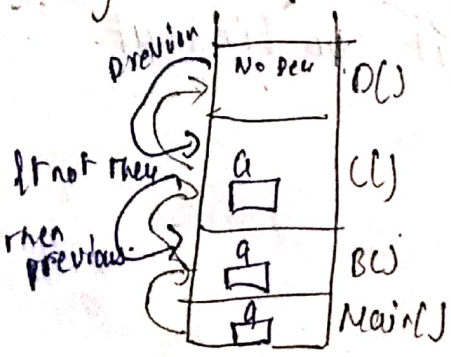
You say compiler to resolve the problem there is two ways

(i) Static Scoping :- During Compile time: Compiler will resolve Static Scopiy using Globle variable static value

(ii) Dynamic Scoping :- Processor will resolve dynamic scoping taking previous value as a Handling Very difficult

Every Programming follow Static Scoping.

```
D()
{
  int a=10 - No decloration.
  Pf(a)   So Static Scoping will take Static valu
}         Dynamic will take previous value.
```



In Perevious first example (2) In Dfunction No Decleration there. This function is in the top of stack. because of No declaration of a. compiler confused which value of 'a' i should print. coz there are many value of a. this problem is called Scooping problem.

To handle the problem two ways are there.

(1) Static Scooping (2) Dynamic Scoping

Compiler solve very easily because very less work. but in Dynamic type checking. it means you all checking at runtime. So very work you have to do. Checking previous one. if a is not then. then previous. and up to so on. So to handle Dynamic Scooping it is very difficult So ~~every~~ ~~pt~~ 'C' lang follows. Static type checking It will take less time. coz it will scan one type. at compilation time.

Dynamic type checking :-

- Type checking done at run-time.
- More time.
- All type related error it cannot find out.

Static Scooping type checking :-

- Type checking done at compile time.
- It will take ~~so~~ less time.
- All type of related error it can find out.

For ex:- a=10 b=20

```
if(a>b)
{
  =
}
else
{
  =Ctype here;
}
```

If condition true it will execute only if part if never know the problem or error there in else part.

But in static before runtime it will scan the program at 1 time. if error it will give error msg.

Memory allocation should be Dynamic only.

(Reason) It cannot be static. Compiler cannot estimated how much function will take how much space coz of recursion. (Drawback: It don't support recursion)

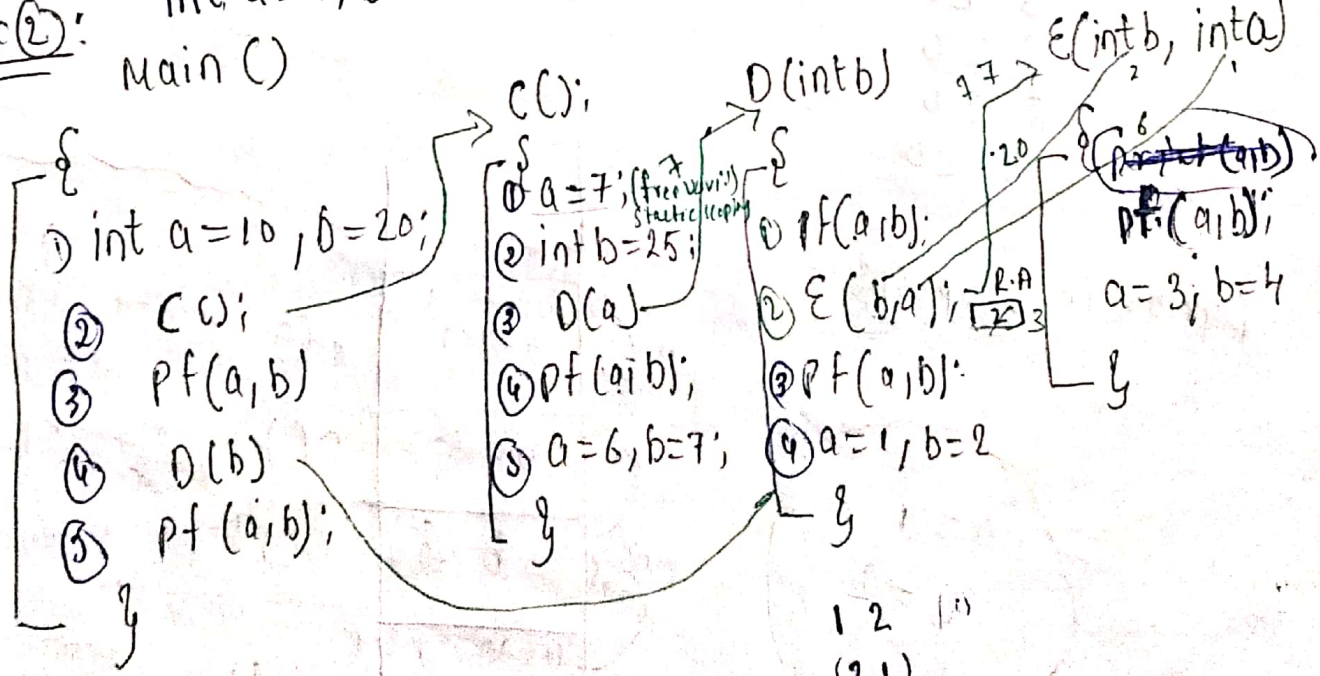
Recursive Program should be Dynamic due to some function & variable. Calculate again & again.

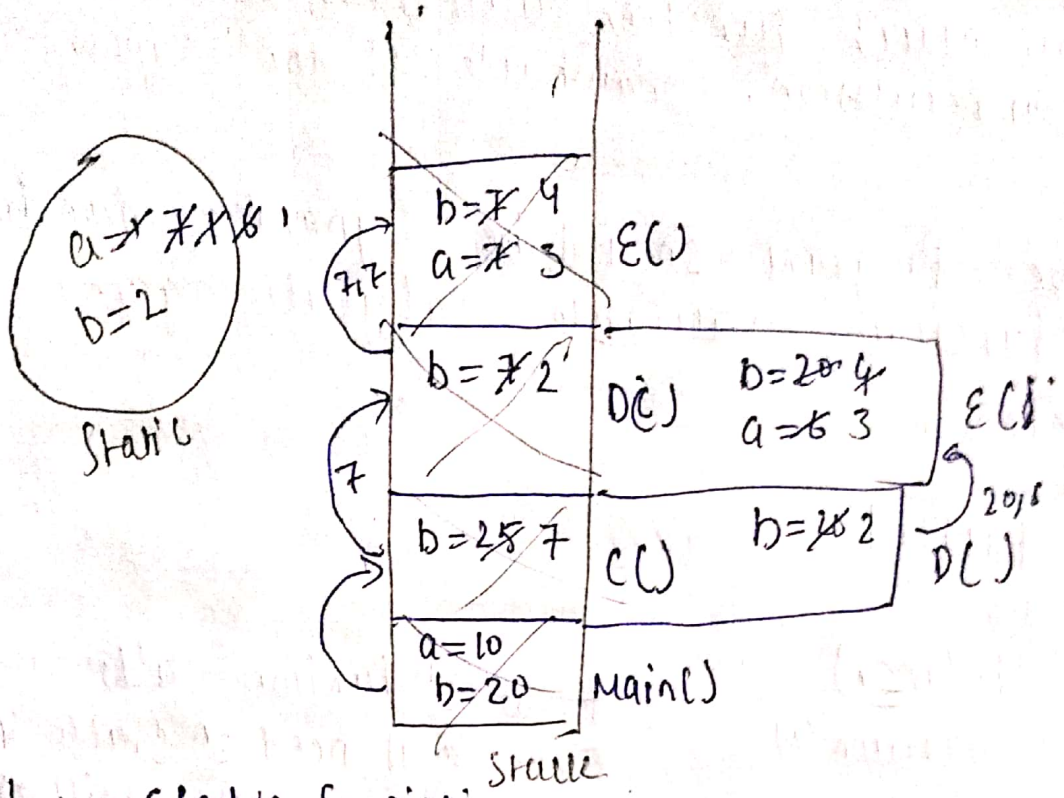
Ex:
```
fact(n)
{
 if(n<=1)
  return(1)
  else
 return(fact(n-1)*n)
}.
```

2^n function calls. So it need 2^n stack space. But compiler will check at once & think that it is one line code it will think it will get only 2 stack space.

① Static Scoping:

Ex(2):
```
int a=1, b=2
Main()
{
 int a=10, b=20;
 C();
 pf(a,b)
 D(b)
 pf(a,b);
}

C();
{
 a=7; (free variable static scoping)
 int b=25;
 D(a)
 pf(a,b);
 a=6, b=7;
}

D(int b)
{
 pf(a,b);
 E(b,a);
 pf(a,b);
 a=1, b=2
}

E(int b, int a)
{
 pf(a,b);
 pf(a,b);
 a=3; b=4
}
```

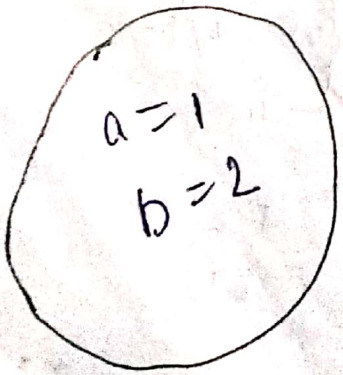


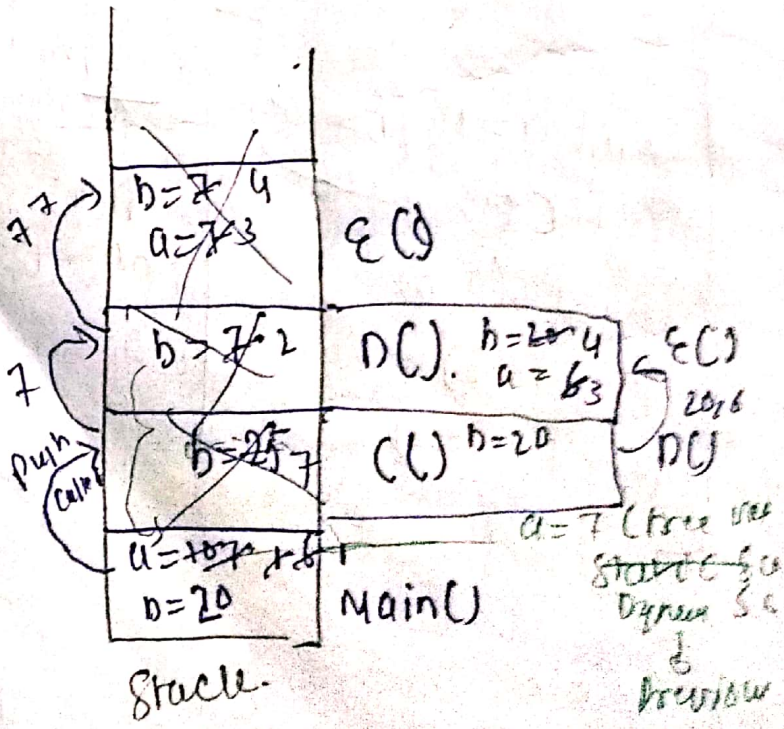


O/P: Static Scoping:

7, 7
7, 7
7, 7
1 25
10 20
6 20
6 20
6 20
10 20

Dynamic Scoping:



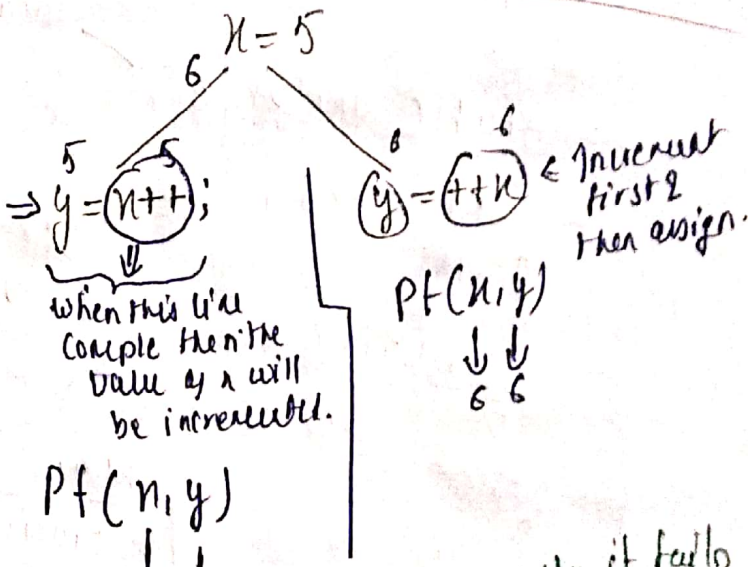


Dynamic scoping:

O/P: 7, 7
7, 7
7, 7
10, 25
6, 20
6, 20
6, 20
6, 20
1, 20

Static Variables:

```
main()
{
  static int var = 5;
  if(--vr)
  {
    main();
    pf(vr)
  }
}
```



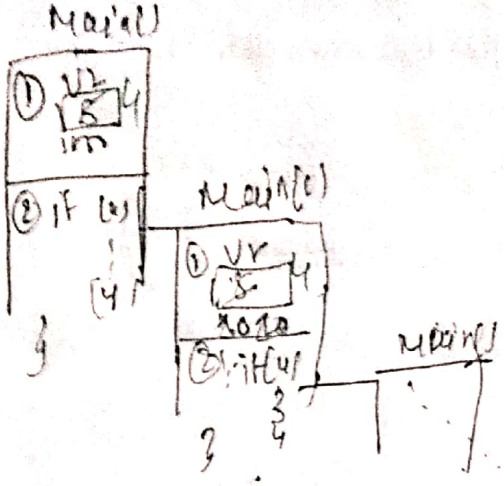

```
if(1)
pf(Hi)
else
pf(Bye)
```

0 — only it fails only if any value float char -ve +ve all will be passed (True)

A thing which can be stored in — 2 byte called char
— 2 " integer.

In comp. No char no int, float, everything will be based on space(size).
char we can store it in integer but integer cannot be stored in char, similarly, float.

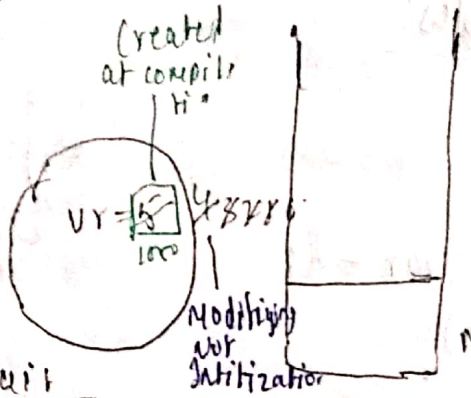
Ex-① If static Not there.



With Static :-

```
Main()
{
1. static int vr=5;
2. if(--vr)
   {
   3 Main()
   4. P f(vr)
   }
}
```

local static variable so created at compile time.

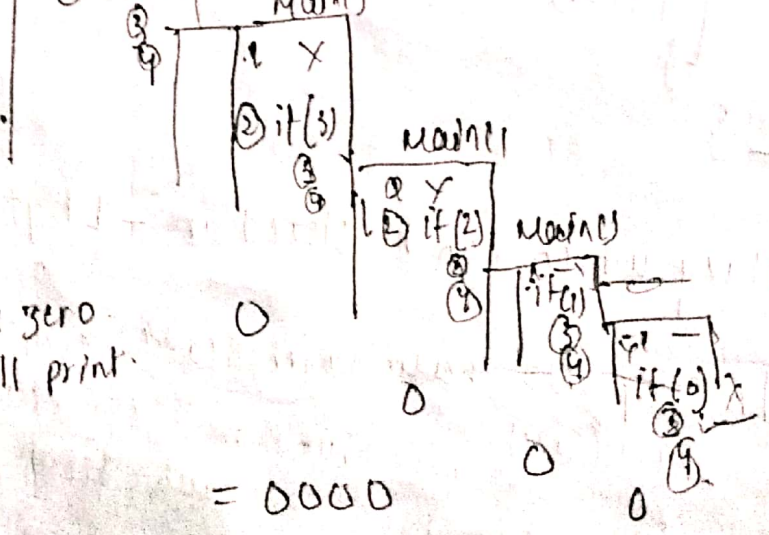


Keyword

Main

① static → While seeing static, processor skip this line. it is not his work it will done with compiler

② if(4)



—5=4
—4=3
—3=2
—2=2
—1=0 Falls

then 4 zero will print.

0 0 0 0

= 0000

NOTE:

1) Static For static variable Memory will allocate Compile time in static area of Main. memory.

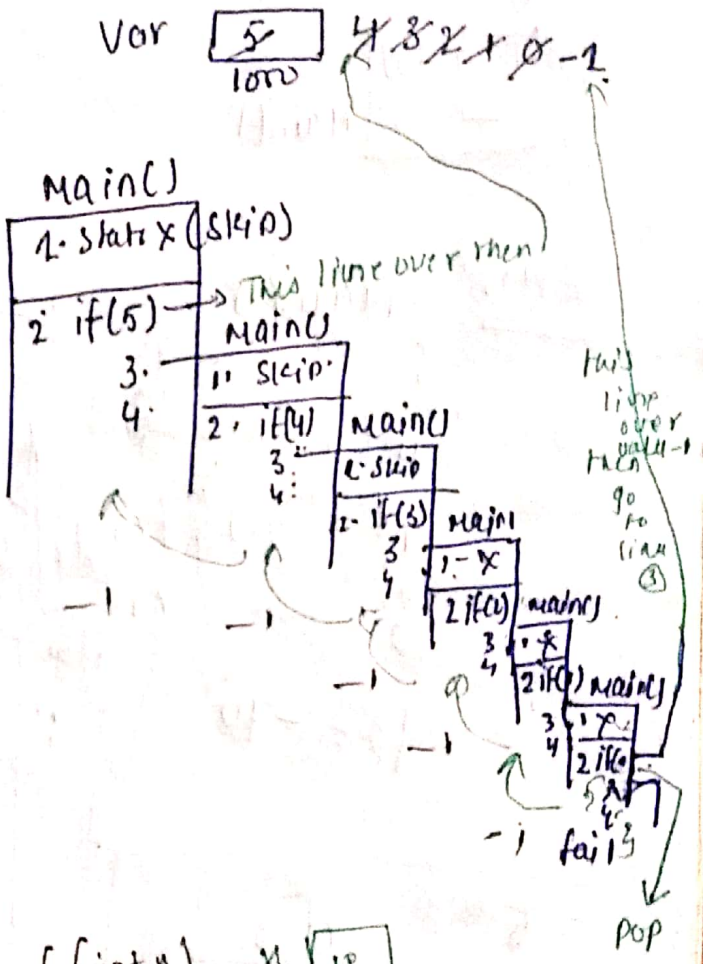
② For static variable Memory will allocate only once.

③. Static variable will be initilized only once.

④ By default static variable will be initilized to '0'

Ex②:

```
main()
{
1. static int var=5;
2. if(var--)
   {
   3. main()..
   4. pf(var)
   }
}
```



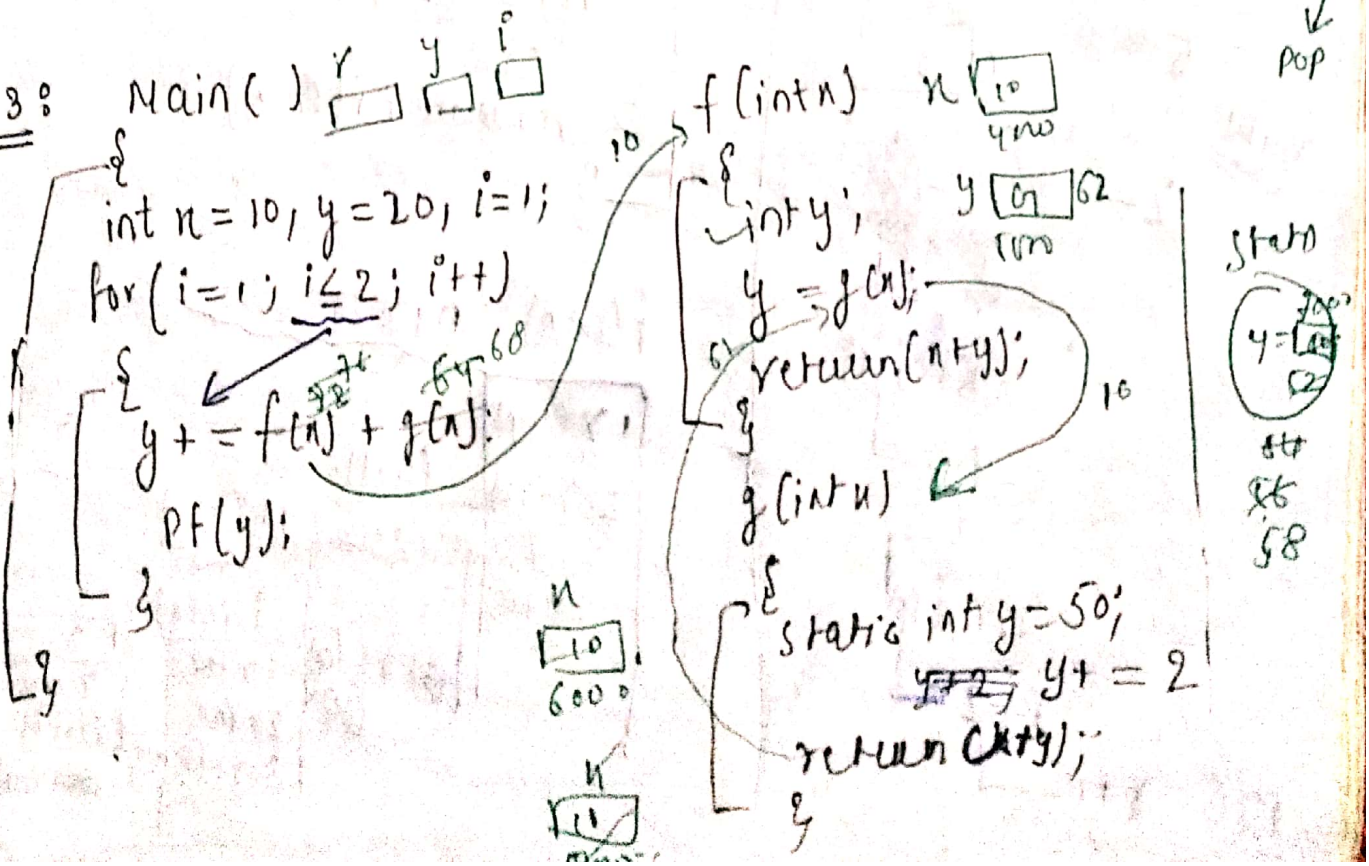
o/p: -1 -1 -1 -1 -1

Ex 3:

```
main()
{
int n=10, y=20, i=1;
for(i=1; i<=2; i++)
  {
  y+=f(n)+g(n);
  pf(y);
  }
}
```

```
f(int n)
{
int y;
y=g(n);
return(n+y);
}
```

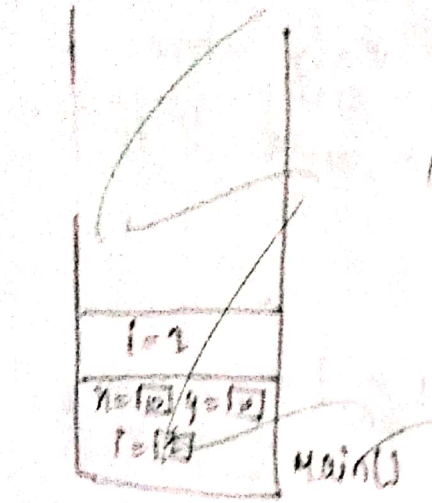
```
g(int n)
{
static int y=50;
y+=2
return(n+y);
}
```



int a = 10 Initialization (But Initialization not done
a = a + 5 — Modification (In static variable Modification can be done)

y: 50 (5000)

O/P: 156 / 300



Static

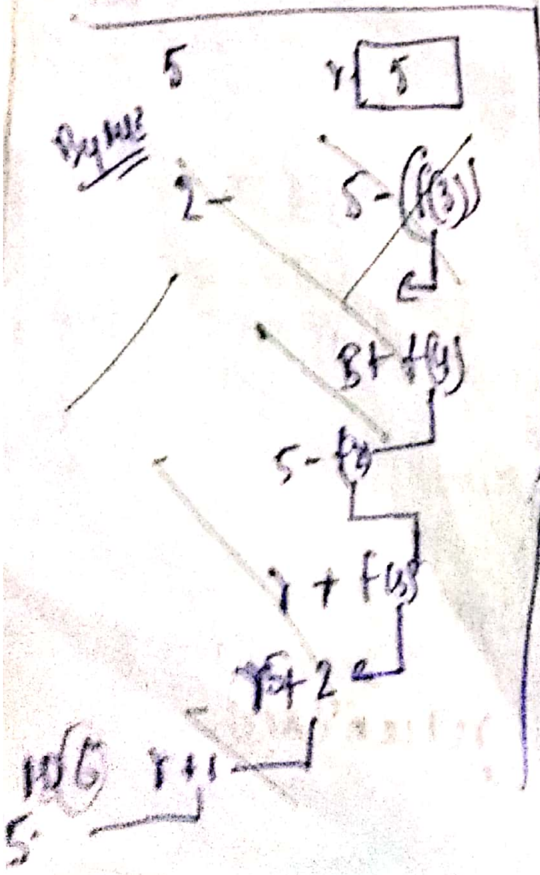
r [5] 10 (1000)

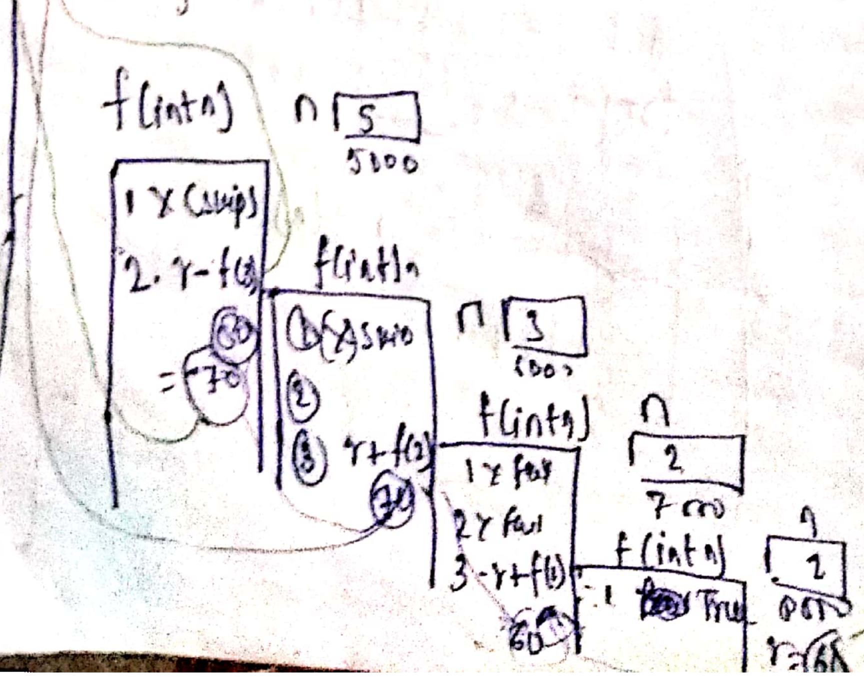
Ex (1):

```
Main()
{
int n=5;
pf("%d", f(n));
}
```

O/P = -70

```
f(int n)
{
static int r=5;
(1) if(n<=1)
{
r=10
return (r+50);
}
(2) if(n>3)
return(r-f(n-2))
else
(3) return(r+f(n-1))
}
```





11/9/2019

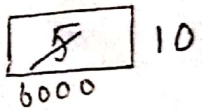
Extern Variable:

int a=5 - Global Var.

Ex①

```
Main()
{
extern int a
  Pf(a);
  a=10;
  Pf(a);
}
```

while using this keyword. Memory never creates at runtime. It will use the Memory of Global variable. If there is variable which is copy of that (local variable but taking memory of global)



O/P: 5
10

Ex②

```
Main()
{
extern int a → No Memory created at runtime coz of extern keyword.
  Pf(a) → No Global variable there. No variable there with name a. So error.
  a=10;
  Pf(a);
}
```

O/P: Error Msg. (undefined)

Ex③

```
Main()
{
extern int a: No relation
  Pf(hi); No need to use the value of a. So Print Hi
}
```

So while checking it checks at global. So Print Hi

O/P: Hi

Ex④

```
extern int a; — If you don't initialize memory not created.
Main()
{
extern int a;
Pf(a)
}
```

O/P: error undefined Symbol.

Ex①

```
extern int a = 10
main()
{
extern int a;
  Pf(a)
  a=5
  Pf(a)
}
```

o/p: 10
5

— If you initialize the value then extern never work. So memory will be created coz extern variable Global.

local variables

```
int a
auto int a
register int a.
↳ Memory will allocate at register. & program will be faster.
```

All are same

```
main()
{
int a=5;
int a=10
  pf(a)
}
```

Multiple declaration error. 2 local with same name.

Within the function with same name. you can create multiple variable. by using the keyword. "extern" otherwise not possible.

```
int a=5
main()
{
extern int a;
extern int a;
extern int a;
extern int a;
  Pf(a)
}
```

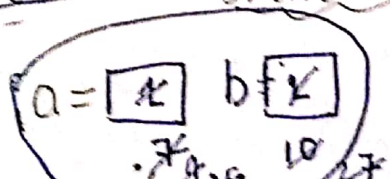
Allow using the Global Memory.

Ex 6: A program which contain all the keyword. what we learn before.

```
int a=1, b=2;
Main()
{
  auto int a=10;
    Pf(a,b);
    C();
  Pf(a,b)
  a=3, b=4
    D(b)
  Pf(a,b);
}

C()
{
  Static int a=7;
  Pf(a,b);
  a=9, b=10;
    D(a);
  a=21, b=22;
}

D(int b)
{
  extern int a;
  Pf(a,b);
  a=25, b=27;
    E()
  a=30, b=31
}

E()
{
  register int a
      =29
  Pf(a,b);
  a=30, b=39
}
```

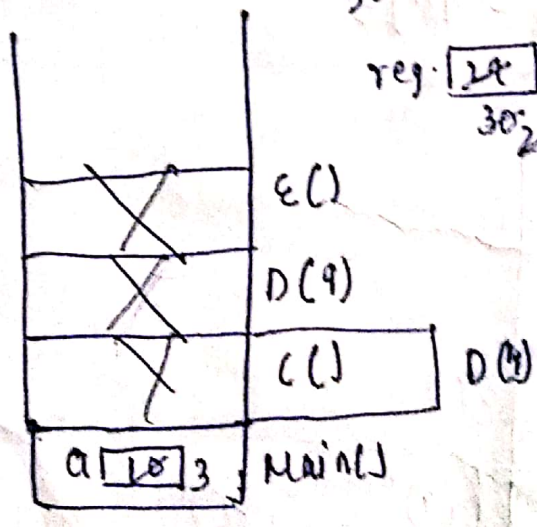
Static Scoping

Sol:



Static 30 25 30 27 31 39 31

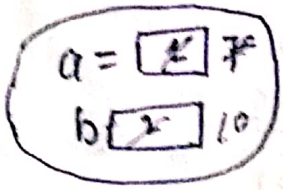
reg 29 30 29 30

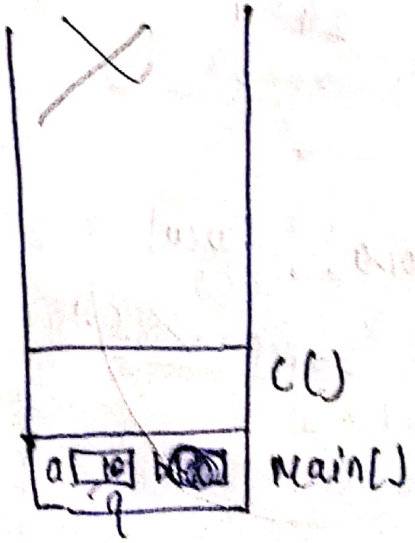


o/p: 10 2
7 2
9 10
29 27
10 22
21 4
29 27
3 31

By me:

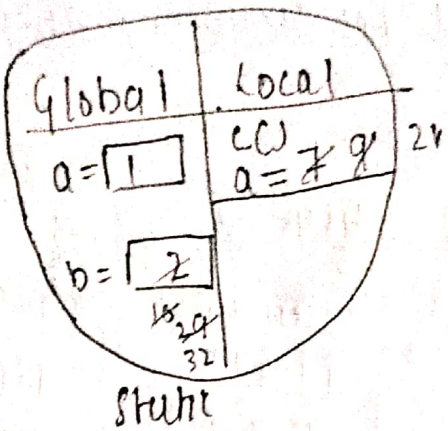
Dynamic Scoping



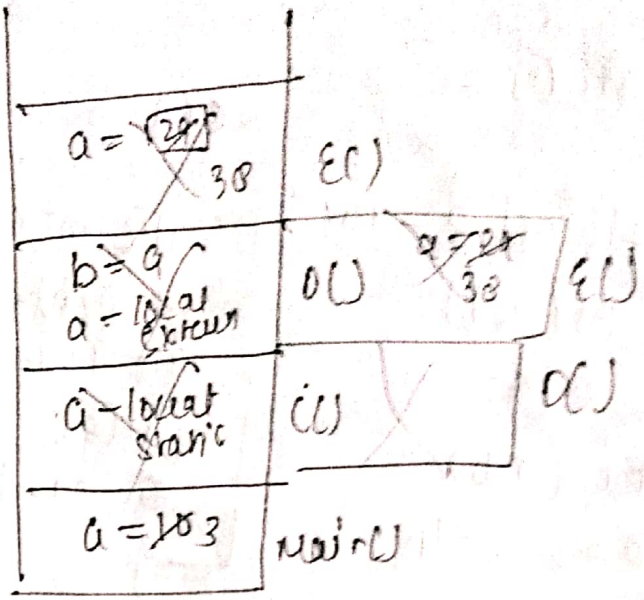


o/p: 10 2
7 2

By Sir



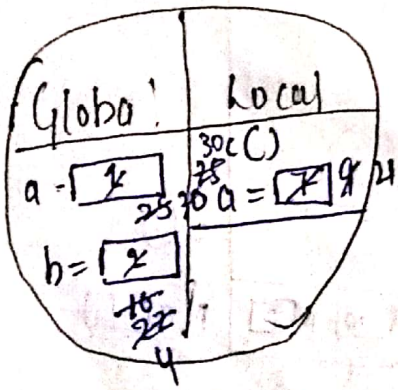
Static



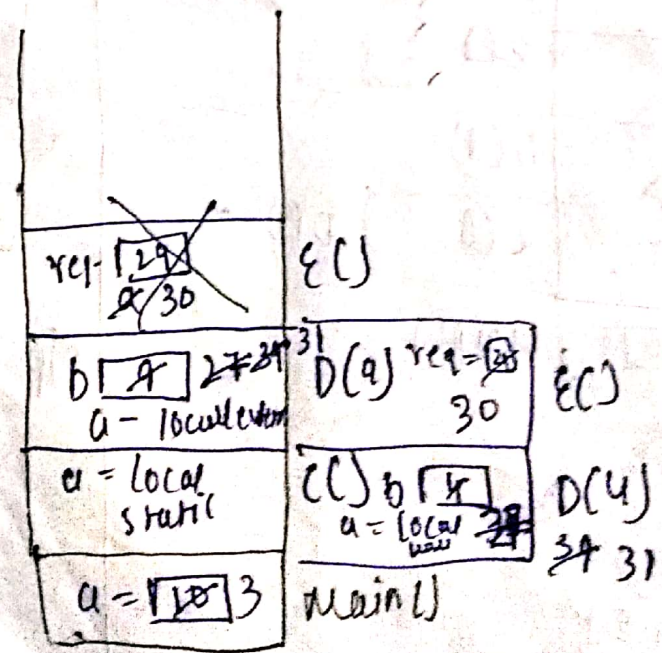
Stack

O/P: 10 2
7 2
1 9
29 10
10 22
30 4
29 4
3 39

Dynamic Scoping:



O/P: 10 2
7 2
1 9
29 27
10 22
30 4
29 27
3 4



ex (7):

```
int i;
Main()
{ int i;
for(i=1; i<=25; i++)
            20<25
{
switch()
      (i=1)
Case 1: i+=3
Case 2: i+=4
Case 3: i+=5
Default : i+=3
} break — No use of break
}           after every case break required
pf(i)
}
```

i:0 static

: 0/1 X 4 8 ~~8~~ ~~13~~ ~~16~~ ~~17~~ ~~20~~ 24 28 29

o/p: 16 ~~20~~ 24 28

NOTE:

- In Switch Sat Statement after case. break is needed if not it will executed remaining cases also.
- After default no need of break anyhow it is a last statement.
- In Switch Statement default can be placed anywhere. But it will execute when no-match is there.

& | && — Boolean and operator And operator
L bitwise

Ex: Main()

```
{
int i=-1, j=-1, k=-1, l=2, m
M=((i++ && j++ && k++) || (l++);
          └ Whoever came first who will be done first
pf(i, j, k, l, m);
}   0  0  0  2  1
```

i = -1, j = -1, k = -1, l = 2, m = 1

() = Left → right
= Right → left

for anything = 1 (so need to require

ex 7:

```
int i;
Main()
{ int i;
for(i=1; i<=25; i++)
{
  switch(i)
  {
    case 2: i+=3;
    case 2: i+=4;
    case 3: i+=5;
    default: i+=3;
    {
      break;
    }
    pf(i);
  }
}
```

(i=1)

20<25

break — No use of break after every case break required

o/p: 16, 20, 24, 28

NOTE:

- In switch stat statement after case. break is needed if not it will executed remaining cases also.
- After default no need of break anyhow it is a last statement.
- In switch statement default can be placed anywhere. But it will execute when no-match is there.

& | && — Boolean and | bitwise AND operator

Ex:

```
Main()
{
int i=-2, j=-2, k=-1, l=2, m;
m=((i++ && j++ && k++) || (l++));
pf(i,j,k,l,m);
}
```

i = -1, j = -1, k = -1, l = 2, m =

i++ (true(1) non-zero), j++ (true(1)), k++ (true(1)), l++ (true(1)), OR

1 || 1 = 1 — whoever came first who will be done first

() = left → right

= right → left

o/p: 0 0 0 2 1

for anything = 2 (so need to required to execute left)

1 or anything = 1
0 && " = 0
} called short circuit

In Previous we get ans 1 so we need to execute further. so (i++) never execute. coz before its execution we get answer coz 1 or anything = 1.

If i=0, j=-1, k=-1, l=2, m

m = (((i++) && j++ && k++) || (i++))
1 0 0 2

PSF i, j, k, l, m.
1 1 1 1 1
0 -1 -1 3 1

└ Never execute
0 & anything = 0 (so No need to execute).

Problem: Side effect.

Pf(++i, ++i, ++i)
6 7 8 → } Two Different answer.
8 7 6 ←

Pf: (i+1, i+2, i+3)
6 7 8 → } no side effect.
6 7 8 ← Do left to right.

Every program will be side effect free program.

Always do from left to right.

Pointers

Ex ①

① int a = 10 — Declaration ⇒ OS ⇒ Inside b-row address 8B

② int *b = &a — Decl ⇒ OS ⇒ a [10] 1000; b [1000] 2000

So OS will always take memory whatever the size of the address

③ Pf(a) ⇒ 10

Pf("%u", b) ⇒ 1000 ⇒ user ⇒ value of b

Pf("%d", *b) ⇒ 10 (*b = 1000)

Wrong Declarations:

int *a = value

int a = address

Ex ② ① inta = 10 — a [10] 1000

② int *b = &a — b [1000] 2000

read ⇒ b is a pointer of type integer (or b is a pointer which containing address of integer)

③ int **c = &b — c [2000] 3000

↳ c is a double pointer of type integer

Pf(a) ⇒ 10

Pf(*b) ⇒ 10 (*b = 1000 → 10)

Pf(**c) ⇒ 10 (*c = 2000 → 1800)

you will get the value of that address

you will get " " " " "

Pf(***c) ✗

*** c — 2000 — the value inside 2000 — 1000 — the value inside 1000 — 10

More than 3 star or 3 pointer not allowed

(OR you declare only 2 pointer (2 star)

So how can we use 3 star. (only 2 or less than 2)

Ex ③ int a = 10

int *b = &a.

pf ((a) * *b)

10 × 10 = 100

It is not multiplication. We will take it is a pointer. coz we declare before b as pointer. Even processor also confused what we will take as pointer or multiplication so he will check the declaration. if declared there then it will take as pointer

This statement we will take as multiplication coz we didn't declare the double pointer. So processor first check the declaration. Double pointer is not there so it will take as multiplication.

every pointer = 8 Byte
size of integer = 2 Byte
" " char = 1 Byte
" " float = 4 "

Ex ④: int a = 10 (a: 10, address 1000)

int *b = &a (b: 1000, address 2000)

int **c = &b (c: 2000, address 3000)

sizeof(c) = 8 Byte

sizeof(*c) = 8 B — 1000 = Addr

sizeof(**c) = 2B — 10 = value

char:
⇒ 8 Byte
⇒ 8 Bytes
⇒ 1 Byte

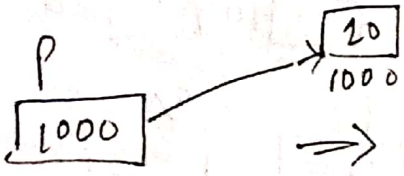
P is a pointer means it contain address.
But which pointer.

⇒ P is a pointer pointing to float
or P is a float pointer.

Here we know P is a pointer which contain adder.

Sizeof(P) = 8B
size of (*P) = 4B.

So How we collecte of pointer is which type. if the pointer is pointing integer then P is a integer pointer like this



⇒ P is a pointer pointing to Integer
or
P is a Integer Pointer

Size of a Pointer(P) = 8B
Sizeof(*P) = 2B

Ex⑥

```
int i=5, j=7;
main()
{
  f(&i, &j);
  pf(i, j);
}
```

```
f(int *j, int *i)
{ i = j;
 *i = *j+5
 *j = (*i -3',
  j = i
 *i = *j;
}
```

Sol by me:

int i: 1000, 5 10
int j: 2000, 7 0

f: int j: 1000, 300
jut e: 1000, 2000, 4000

*i = 10
*j = 5
*i = xi

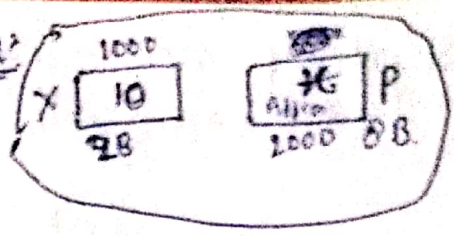
o/p: ~~10 5~~ ~~5 7~~
7 7

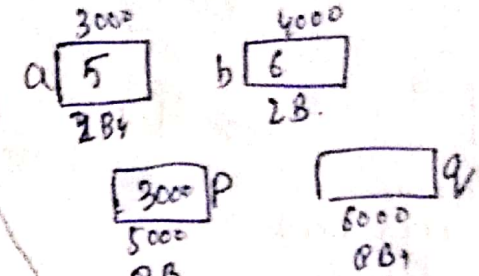
Ex

```
f(int x, int *p)
{
  *p = x;
  x = 10;
}
main()
{
  int a=5, b=6;
  int *p = &a, *q;
  *p = 20   q = &p;
  f(a, &b)
  *q = &b;
  *p = 30
  Pf(a,b)
}
```

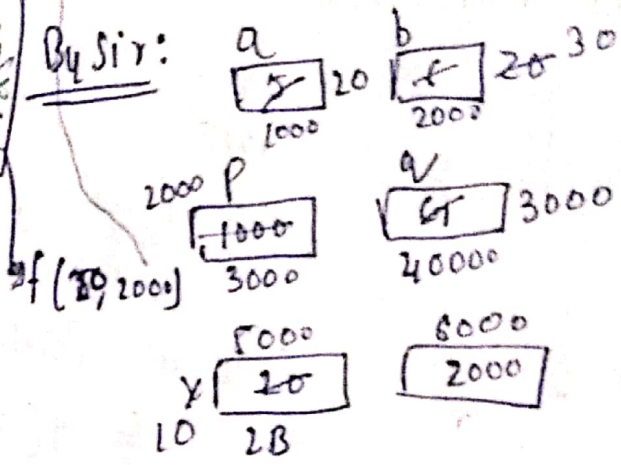
O/P: 20, 30

By Ma'am



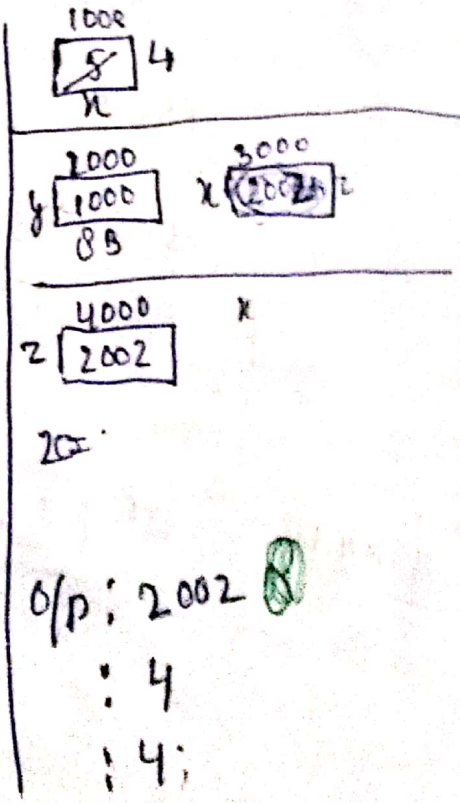


By Sir:



Ex:

```
int n;
void Q(int z)
{
  z += x;
  Pf(z);
}
void P(int *y)
{
  int x = *y + 2;
  Q(x);
  *y = x-1
  Pf(x);
}
main()
{
  n = 5
  P(&n);
  Pf(n);
}
```



share

s [6] 1000

y [1000] 2000 x [7] 3000

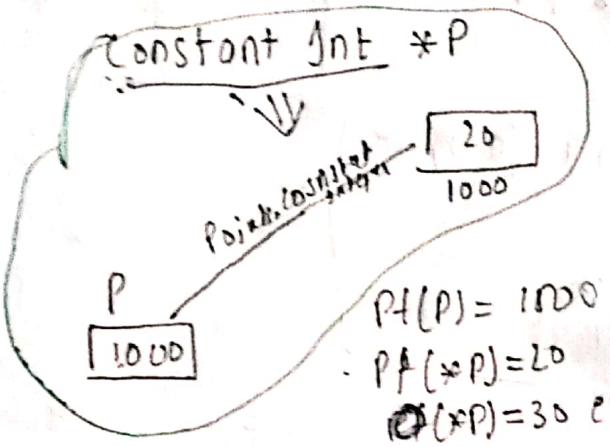
2 [7] 2

O/P = 12, 7, 6

---

int *constant P

Constant Int *P

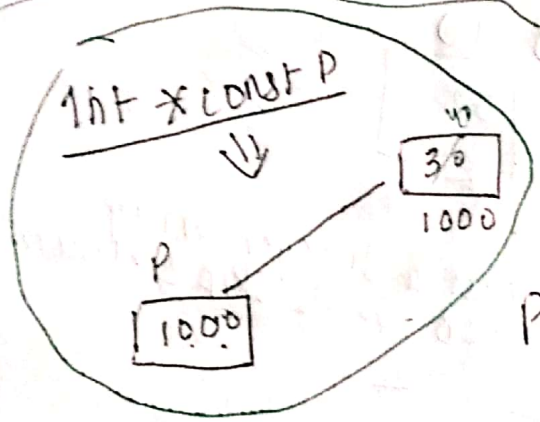


Pf(P) = 1000 ✓

Pf(*P) = 20

(*P) = 30 error we cannot change coz of constant.

But Here you can change the value of P = 2000



Printf(P) = 1000

Pf(*P) = 30

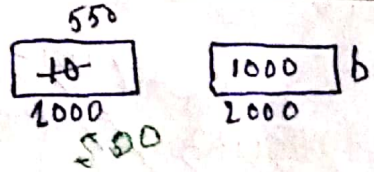
*P = 40

P = 2000

P = 2000 (X Error. Because Here P is constant you cannot change it)

Ex 10:

```
main
{
int a = 10;
int *b = &a;
scanf("%d", b); 500
 Pf("%d", a+50); ⇒550
}
```



O/P : Flp 550

Whatever the I/p you will take

Sept. 12

```
Main()
{
  int a[] = {25, 32, 61, 98, 26, 32};
  Pf("%d", f(a, 6));
}
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);
}
```

Here a+1 No possible coz array

1000 passing only base addre of whole array

Here a+1 possible coz of variable.

or it can be written like this a[] ↑ It doesn't mean array come only base address come

By Nil =

a	25	32	61	98	26	32
1000	2	4	6	8	10	

a 1000 (2000) n 6 (3000)

25 + f(1002, 5)

In array pf(a) = 1000
a = 1000
a+1 = 1002

variable = Inside
Array = outside

By Sir:

a	1000	02	04	06	08	10
25	32	61	98	26	32	
0	1	2	3	4	5	

a ⇒ 1000
a+1 ⇒ 1002 (coz it is a integer array. int = 2B So its skip 2 element)
a+3 ⇒ 1006

---

a ⇒ 1000
a+3 ⇒ 1006
*(a+3) ⇒ 98
or
a[3]

---

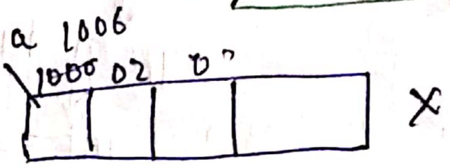
a ⇒ 1000
a+1 ⇒ 1002
*(a+1) ⇒ 32
or a[1] = 32

NOTE:

1D: $a[i] = *(a+i) = *(i+a) = i[a]$
for user convinience — But actually like this

2D: $a[i][j]$ = Using 1D- you can get 2D like this: (a[i])[j]
actually b [j]
→ $*(b+j)$
$*(a[i]+j)$
$*[*(a+i)+j]$

a = (a+3) = 1006 — (a+3: 1000 → 1006)



× L- value can't be changed.. (means base address of address cannot be changed

- Address content you can change. but address cannot be change.

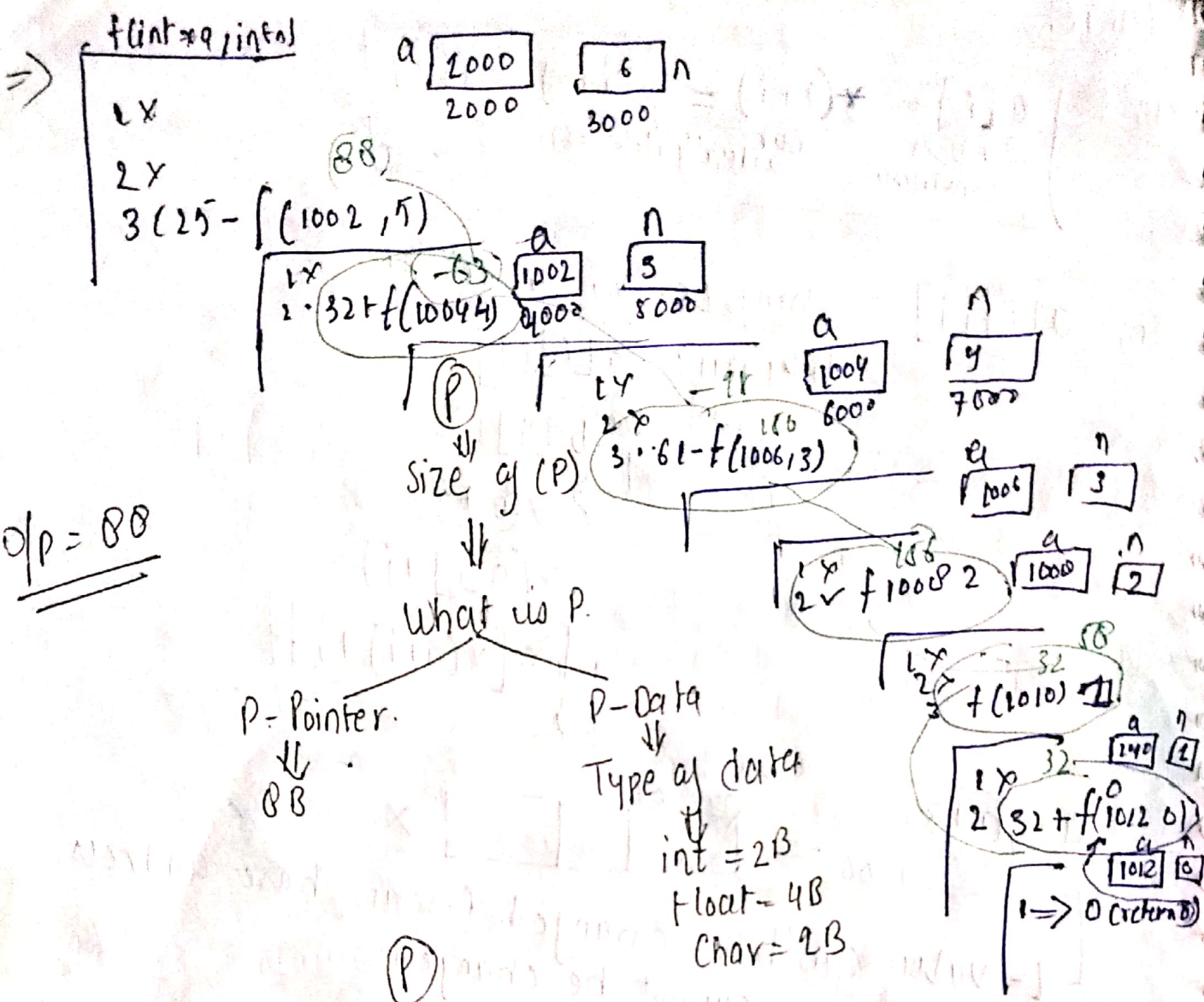
$*(a+3)$ = 99 (content you change..) — (a+3: 1000 → 1006)

- In array +1 is skiping one element. (forward)
- In array a-1 is skiping one element (backward) =998

print a = 1000
print &a = 1000
} Both are same but the difference is

a+1 ⇒ 1002

&a+1 ⇒ 1012



O/P = 88

What is P.

P - Pointer.
⇓
8B

P - Data
⇓
Type of data
⇓
int = 2B
Float = 4B
Char = 2B

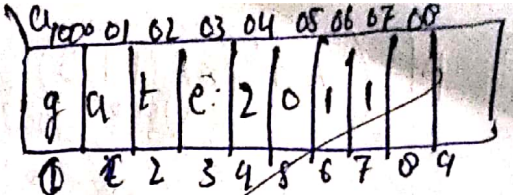
P
⇓
P+1
⇓
What is P

P - pointer
⇓
Type
⇓
int = 2B
float - 4B
char = 1B.

P - data
⇓
always +1

Ex: 2011 GATE

```
Main()
{
char a[] = "gate2011";
pf("%s", a+a[3]-a[1]);
}
```



1003 − 1000 = 3

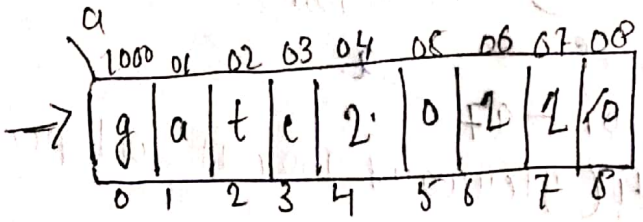
1000 + − 1000

Every char array end with Null

strlen("abcd") = 4

sizeof("abcd") = 5 (include 1 Null character).

4+1 = 5



① pf("%s", "gate2011"); Gate2011

② pf("%s", 2000); = Gate2011 (give base addre and it will print all the char until Null)

③ pf("%s", a+3); e2011 (1000)

④ pf("%c", *(a+3)); c (1000, 1003)

⑤ pf("%s", *(a+3)); error ✗ (1000, 1003, e)

print string e but it will print 1 char 'e' so error garbage value..

⑥ pf("%c", a+3); Error ✗ (1000, 1003)

print char but in will print 1003 string (error) garbage value

⑦ pf("%d", *(a+3)) = 101

(a+3: 1000 → 1003; *(a+3) = e)

└ It will print the ASCII value.

a b c d e.

97 98 99 100 101

⑧ pf("%s" a + a[3] - a[1]);

a → 1000

*(a+3) (1000 → 1003) = e − *(a+1) (1000 → 1001) = a

⇓

1000 + 101 − 97

= 1004 (afterward 1004 it will print)

o/p = 2011

Ex: Main()

{

① char a[10];

② char *b = "hello"; 3000 (3000 onwards all include the Null) = 6 B

③ int length = sizeof(b); 3000 — strlen(b); = (3000 onwards exclude Null) = 5

for(i=0; i<length; i++)

{

a[i] = b[length − i];

└ b[4] − Shortcut ⇓ *(b+4) actual expression (3000 → 3004) ~0

}

pf("%s", a); 1000

}.

→ No Output

int a[10];

sizeof(a) = 20

strlen(a) = X

⇓

only applicable for string functions.

int: 2-5.

char: 1

a-b-c

0 − 256

① ⇒

a	1000	01	02	03	04	05	06	07	08	09
	G (0)	G (o)	G (l)	G (l)	G (e)	G (h)	G	G	G	G.
	0	1	2	3	4	5	6	7	8	9

② b → 3000 (2000) ← Base Address → 3000 01 02 03 04 05

"hello" – stored in memory ← char. array.

Giving string is nothing but base address

③ length → 5 (4000)

pf(b) = 3000
pf(*b) = h
pf*(b+1) = e

```
Main()
{
① int a[5][3] = {10,20,30,40 ... ... 150};
   Pf("%u %u", a+2, *(a+2)+2);   → 1012.
   Pf("%u %u %u", *a+2, **a+2, *(a+3)+5);   → 1028
   Pf("%d", ((a == a[0]) && (*a == a[0][0])));
}
```

① → 1

Row-Major order

a	0	1	2
0	1000 / 10	1002 / 20	1004 / 30
1	1006 / 40	1008 / 50	1010 / 60
2	1012 / 70	1014 / 80	1016 / 90
3	1018 / 100	1020 / 110	1022 / 120
4	1024 / 130	1026 / 140	1028 / 150

Address

a = 1000

a+1 = 1006
└ 2D-Array skipping 1-row.

a+4 = 1024
└ skip 4×3 = 12×2 = 24.
(4 rows, 3 cols; 2 byte)

a = 1000

a+2 = 1012
*(a+2) = 1012
(1st time did not get the value coz of 2D)

a+2 = 1012
(a+2)+1 ⟹ 1018 (1-row)
*(a+2)+1 ⟹ 1018

---

a = a+2 (1000) → 1012 ✗
= a+2 (1012) ✗
a+1 (1000) → 1006 ← row selected base address of next address
row not selected

Address cannot be changed

a+2 = 1012
*(a+2) = 1012
*(a+2)+1 = 1024
i.e (skip one element)
(((a+2)+1)) = 80
a[2][1]

*((a+2)+1)+2 ⇒ 1022
**((a+2)+1)+2) ⇒ 120
a[3][2]

1016
1018
1020
1022

Note:

- In 3D-Array to get the data 3 star needed. 1st star. 2D skip 1 2D, 2nd star 1 row, 3rd get data.
- In 2D-Array to get the data 2 star needed
- In 1-D " to " " " 1 " "

a ⇒ 1000
*a ⇒ 1000
**a ⇒ 10
⇓
((a+0)+0)
or
a[0][0]

a ⇒ 1000
a+1 ⇒ 1006
*a+1 = 1002
**a+1 ⇒ 11
&a ⇒ 1000
&a+1 ⇒ 1030
└ skip total 2D 3×5 = 15×2 = 30

Q:

int a [5][4][7]

1000 1028 1028

((a+2)+3) ⇒ 1034

-2.. 7×2 = 14×2 = 28

*(a[2])+4 ⇒ 1120

work as → *(a+2) 1112 1000 array skipped

1112

1112

two are fighting

Bracket is a winner coz of highest priority. 56×2 = 112

a[i] In 2D array base address of 2-D

a[2] In 3D array base addres of 2-D

a+1 = skip up 1 2-D.

*a+1 = selected 1-2-D.

Ex: Main()

{

int *b[6]

real: b is a array which contain 6 elements. where every one is integer pointer

① int a[] = {10, 20, 30, 40, 50, 60}; ✓

② int *b[6] = {a+2, a+1, a+3, a, a+4, a+5}; ✓

③ int **c = b;

④ (Post Inc have hiest priority).

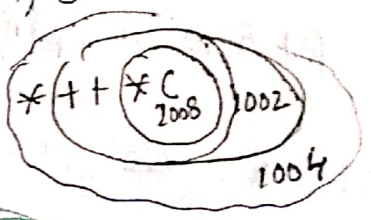
*(c++) 2000 1004 2000 winner

Increment after

pf(c-b, *c-a, **c); ⇒ 1, 1, 20

2008 2000 8/4=2 1002-1000=4/2=2 20

**++c

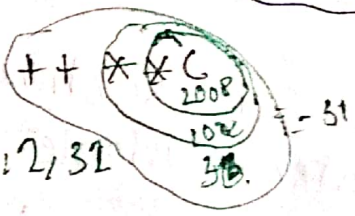


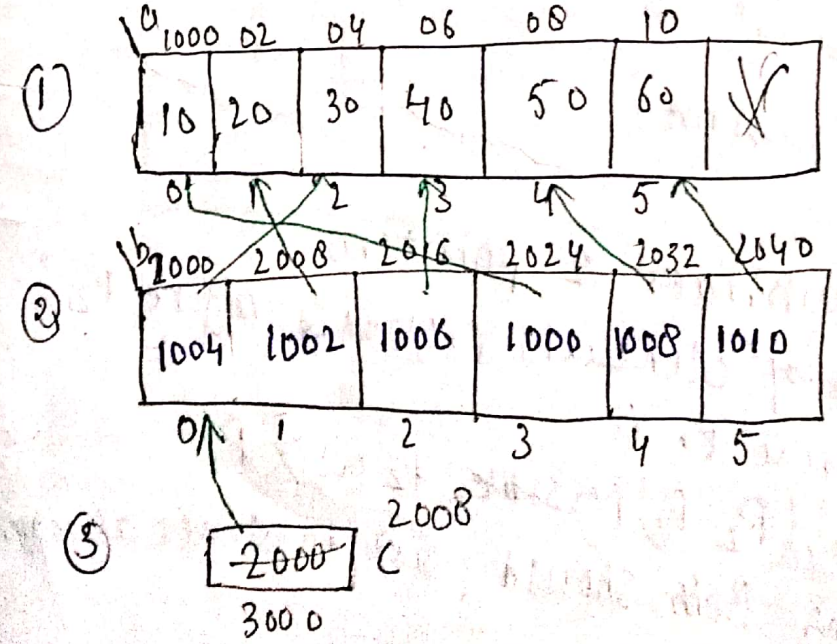
pf(c-b, *c-a, **c); ⇒ 2, 2, 30

2008-2000/8=2 2008 1004-1000/4=2 1004

++**c

pf(c-b, *c-a, **c); ⇒ 1, 2, 32

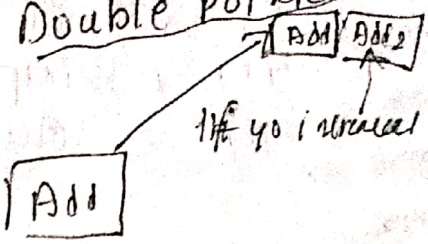




a+2 = 1000

④ * & (++) post inc. both are uniary operators but ++ having higher priority.

c is a integer pointer. so it pointing 8 it will increment by 8 byte = 2008



(4) (-b
2008 2000 = 2008 - 2000 = 8/8 = 1 element in
(Size of 1 element is 8)

If 2040 - 2000 = 40/8 = 5 elements there.

NOTE: (1) We can add integer constant to pointer variable.

Ex: P+5 = skipping 5 elements from P in forward dirn.

P+5.5 = Not possible.

NOTE (2): We can subtract Integer constant from pointer variable.

Ex: P-5 = skipping 5 elements from P in backward direction

P-5.5 = Not possible

NOTE 3: We can subtract 2 pointers

$P_2 - P_1$ = No of elements presents before P_2 from P_1

1 Condition: $(P_2 - P_1)$ possible P_2 is > P_1

2 Condition: Both should point to same array.

NOTE-4: We cannot perform Addition, Multiplication, Division between two pointers coz there is no meaning.

Ex:

```
Main()
{
float b[] = {30, 80, 10, 90, 60};
float *c[] = {b+2, b+3, b+1, b, b+5, b+4};
float **a = c;
. ++*++*a
. pf(a-c, *a-b)
. ++**(a+2)
pf(a-c, *a-c) = 3
pf(**++a+2) = 3
}
```

O/P = 0 1 3
0 3
93

①

b	1000	04	08	012	16	20
	~~80~~	~~80~~ 81	10	~~90~~ 91	60	55
	0	1	2	3	4	

c	2000	08	16	24	32	40
	1008	1012	1004	1000	1020	1016
	0	1	2	3	4	5

a | 2000 | 3000

13/Sept

main()

① char *a[] = {"stupid", "bacho", "chachi", "papa", "pop"};

char **b[] = {a+2, a, a+1, a+3, a+5, a+4};

char ***c = b;

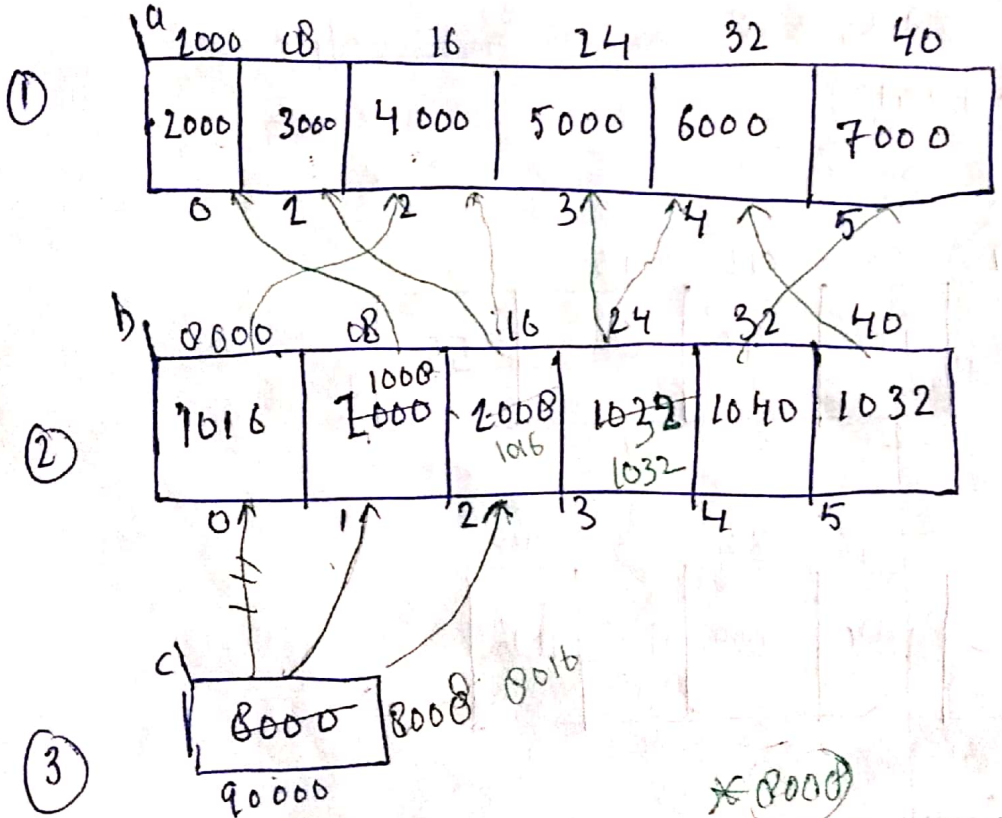
***++c+3;

Pf("%s", *++*(c+2)+1);

Pf("%s", *++*++c+1);

Pf("%c", *++**(c-1)+10);

}



④ *++*++c+3

8008

8000

1008

3000 + 3 = 3003

*8008

++2000 = 2008

Pf: *++*(c+2)+1

8008

8024

1024

1032

6008 afterwards

o/p ⇒ apa.

(5) * ++ * ++ C+1

8008
8016
2008
2026
4000 + 12 = 4001 ⇒ hacha

(6) * ++ * * (C-1) + 10

8016
8008
2008
3000
3002
97 + 10 = 107 ⇒ k

Ex:

```
main()
{
char a[] = "hello";
char b[] = "hello";
if( *a == *b)
pf(hi)
  else
pf(bi)
}
```

Ex:

```
main()
{
char *s = "%d\n";
s++; =2000
(s++); 2001
P(s-2, 300);
```

2000 ⇒ S: 1000 (1000) 2001 2002

2002 - 2 = 2000

pf(2000, 300)

Pf(2000, 300)

⇓

Pf("%d\n", 300)

⇓

300

%u ⇒ To Print Addr
%f ⇒ " " Float
%s ⇒ " " String
%c ⇒ " " Char
%d ⇒ " " Integer

Structures & Union.

Structures: User-Defined Data type.

Pre-Defined Data type d. 20 (1000)

Ex①:

① main()
{
struct Node.
{
int a; ⇒ 2B
float b; ⇒ 4B
char c; ⇒ 1B
} 7B
}

② int d = 10
Pf(d) ⇒ 10
Pf(&d) ⇒ 1000

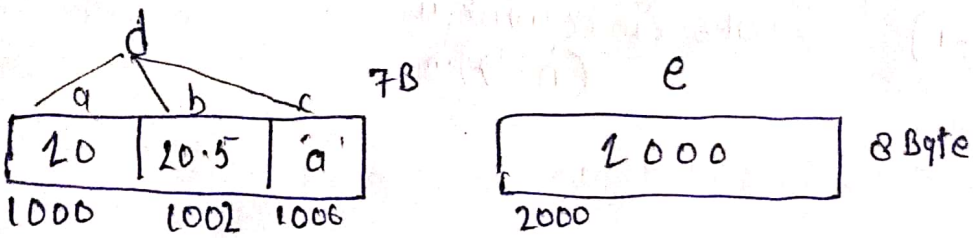
Difference

② struct Node d = {10, 20.5, 'a'}
Pf("%d", d.a) ⇒ 10
Pf("%c", d.c) ⇒ a
Pf("%u", &d) ⇒ a 1000
Pf("%u", &(d.b)) ⇒ 1002

③ int *e = &d
Pf(e) ⇒ 1000
Pf(*e) ⇒ 10

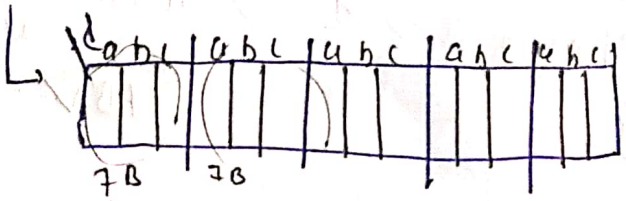
③ struct Node *e = &d
Pf("%u", e) ⇒ 1000
Pf("%d", (*e).a) ⇒ 10

Pf("%.f", (*e).b) ⇒ 20.5
(or e→b)



Array of structure. Normally struct will be global

struct node d[5] = {10, 20.5, 'a'}



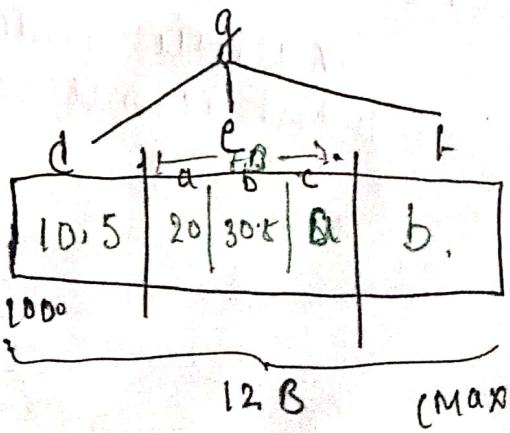
- Inside struct member not need be a pre-defined user-defined also we use.

Ex 2:

① struct S1
{
int a ⇒ 2
float b: ⇒ 4
char c ⇒
} 7B

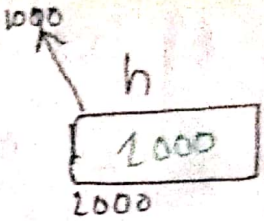
② struct S2
{
float d ⇒ 4B
struct S1 e ⇒ 7B
char f; ⇒ 1B
} 12B

③ struct S2 g = {10.5, {20, 30.5, 'a', 'b'}}



Pf("%.f", g.e.b) = 30.5
Pf("%c", g.f) = 'b'

(4) Struct S2 *h = &g



Pf("%f", (*h).e.b) = 30.5

Pf(h+1) ⟹ 1012 (100.c) can be write as (h→).b

Union:

Ex: Union S2
~~struct~~
{
float d ⇒ 4
struct S, e ⇒ 7
Char f ; ⇒ 1
};

} 12B X instead of taking 12B it take longest byte of the member. 7B ✓

In Structure every member having their own space. but in the case of union all member will share same space but one at a time. (Normally that why we use structure instead of Union. because it is not clear what resource we can taking...

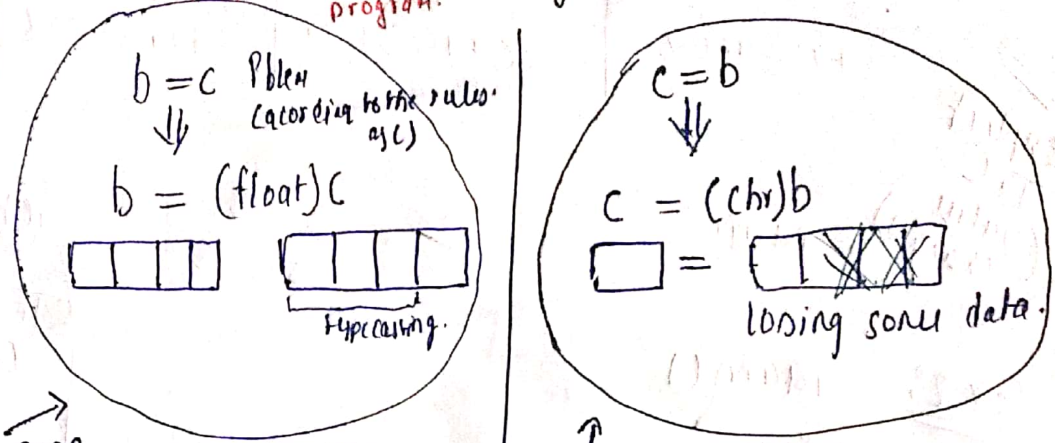
Type Casting:

(in Data

int a;
float b;
Char b;

Both should be same type if not then typecasting

But it doesn't create trouble in program.



Done by the compiler — Implict T.C or coersion.

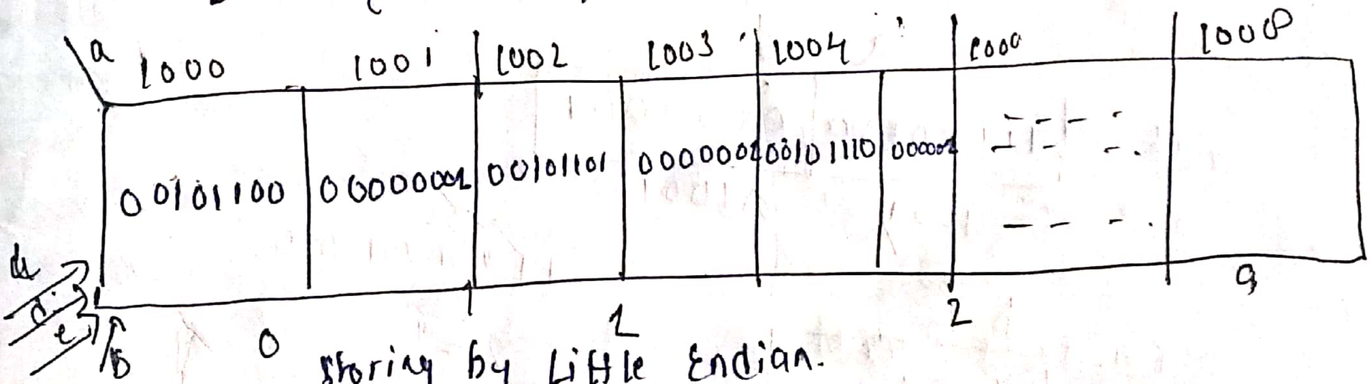
Done by the User (OTPER) — explicit.

Type Casting in painters:

void - storing easy
- retreive difficult.

00101100 = (44)

int a[10] = { 300, 301, 302, 303 .... 309 }.



storing by Little Endian.

int *b = a   b [1000] 2000

Char *c = (Char*)a   c [1000] 3000   Type Castin

float *d = (float *)a   d [1000] 4000   Type casting.

void *e = a   e [1000] 5000   Type Casting (without Type Casting) storing.

— no type cast any address of any data type can store
↳ No Datatype.

storing easy but Retreiving difficult

b ⇒ 1000	c ⇒ 1000	d ⇒ 1000	e ⇒ 1000
*b ⇒ 2B = 300	*c ⇒ 2B = 44	*d ⇒ 4B ⇒ x42 (big data)	*e ⇒ error
b+1 ⇒ 1002	c+1 ⇒ 1001	d+1 = 1004	e+1 ⇒ error. But retrieve

explicit T.C (value changes)

~~Narr~~

(char *c = (char *)e

To verify In Comp Little & Big Endian

int a = 300

char *b = (char *)&a;

1000

pf(*b) = 44

char *c = ?

Ex 2:

```
Main()
{
  int a = 500;
  Char *b = (chr *)&a;
  b++;
  *b = 15;
  pf("%d", a);
}
```

(chr *)&a; → 1000

b++; 1000 → 1000

pf("%d", a); → 4084

a

11110100	00000001

1000 ↑ 1001

b

2000

2000

2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 B

1 1 1 1 0 1 0 0

= 404B

$2^2 = 4$

$2^4 = 16$

$2^{11} = 204B$

Ex 3:

```
Main()
{
  void *vp;
  Char ch = 'g';
  Char *cp = "goofy";
  int j = 20;
```

vp: g — 1000 — 2000 / 5000 / 4000

ch: g — 2000

j: 20 — 3000

cp: 4000 — 3000

vp = &ch;
2000

char 2000

pf("%c", *(char *)vp); → g

vp=&j;
5000

pf("%d", *(int *)vp); → 20

vp=cp; 4000

pf("%s", (char *)vp+3);

}

4000+3= 4003 fy.

o/p: g20fy

ans:

int a[6]={6,7,8,34,67};

int a[4]={23,56,28,29}.

int a[3]={-12,27,-31}

int *n[]={a, a2, a3};

Using array of pointers: Here we have created a 2-D array in which
1st row → contain 6 columns
2nd row → " 4 columns.
3rd row → " 3 columns.

main()
{
f(n)
}

a 4000 4008
[4000]
5000

int *a[]

f(int **a)
{

① pf("%d", a[0][2]) → 8 → *(*(a+0)+2) 8.

② pf("%d", *a[2]); → **(a+2) -12

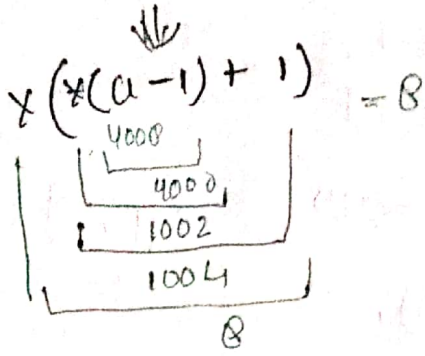
③ pf("%d", *++a[0]);

④ pf("%d", *(++a)[0]); 7

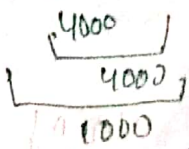
⑤ pf("%d", a[-1][1]); 8

((a-1)+1)
56

(5) = a[-1][1]



(4) ++*(a+0)



```
fun()
{
auto int j=5;
pf("%d ;j++);
}
```

```
main()
{
fun();
fun();
}
```

pf: 5, 5.

An object whose storage class is auto is reinitialized at every function call.

---

(1) Void Pointers cannot be used for de-referencing because each variable type takes diff. amount of memory.

(2) Default value of extern storage class is "0"

int a[5][10] How it to implement 2-D array with 3 uniform row.

2D ⟹ array.
1 row → 6C
2 row → 4C
3 row → 3E

using array pointers.

Ex: int a[10][10];
int *b[10];

T/F — array of pointers it is possible..

T a) a[5][7] = value.
F x b) a[7] = address. (In the middle of the array we cannot change the address)
T c) b[3] = address.
T d) b[3][5] = value.
↳ Because of array of pointers it is possible

Linked List

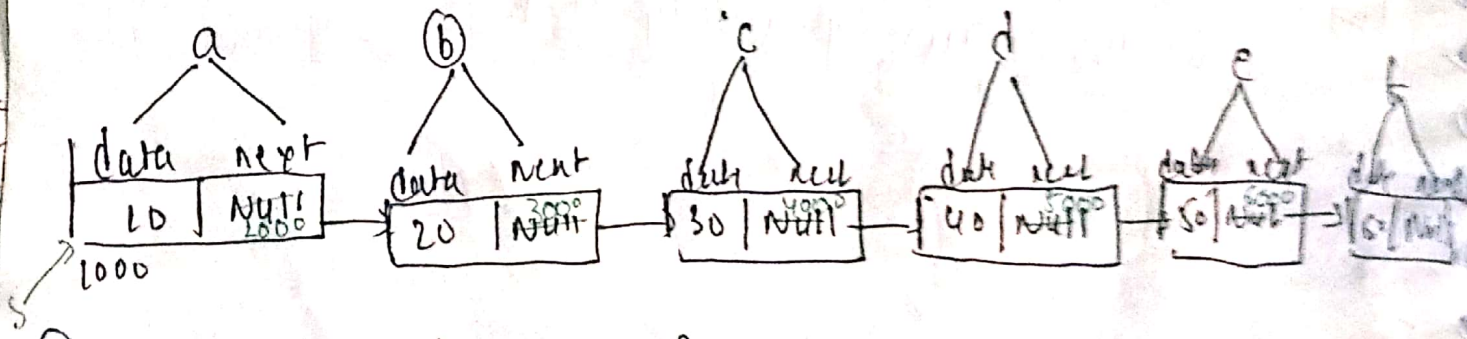
Write a C program to create linked list & returning the linked list.

main()

① struct node
{
int data; ⟹ 2B
struct node *next; ⟹ 8B } 10B
}

② struct node a = {10, Null}
" b = {20, Null}
" c = {30, Null}
" d = {40, Null}
" e = {50, Null}
" f = {60, Null}.

③ a.next = &b
b.next = &c
c.next = &d
d.next = &e
e.next = &f



④ struct node *s = &a

pf(s) ⇒ 1000
pf(*s).data) ⇒ 10
or
s → data.

return (s)
}.

Ex②: Write a C Program to return address of 2nd last node in the given linked list. ∟ Linked List contain min 2 Nodes

④ struct node *s = &a.

Atleast-2ndLastNode( ) { function call.

}

Atleast-2nd-LastNode(struct node *s)

{

① if (s == Null)
  u-empty-stop

② if (s.next == Null)
  u-1-element-stop

③ if while (s->next != Null)
  { p=s
    s = s.next
  }
  return (p);

Best for T.C

Linked list is empty s. contain null

s → or Null

Linked list is 1. s contain. Next null

s → 10 | Null

① ∟

② ∟

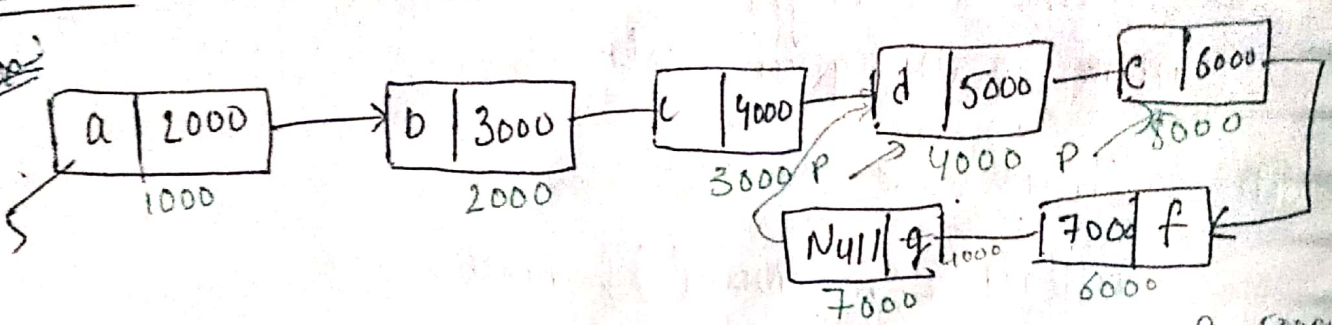
③ while (s->next->next != Null)
  s = s->next
  returns)

Not Null then this will be executed.

Best of Space. Complexity.

14-Sept

Ex:



(i) Struct node *p.

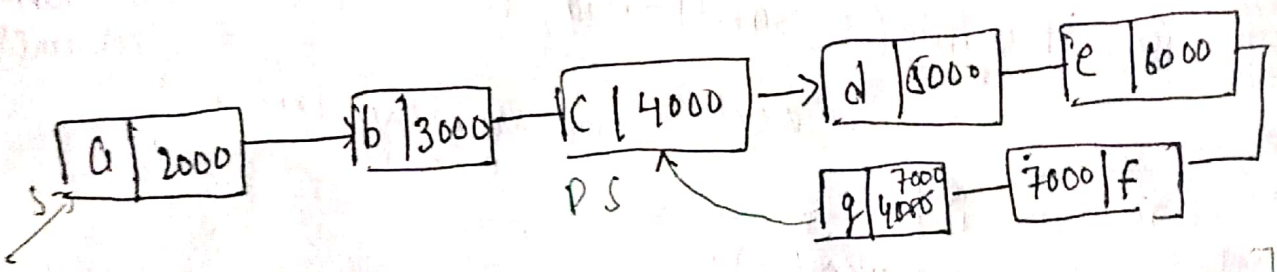
(ii) P = S → next → next → next → next;

(iii) P → next → next → next = S → next → next → next;
= 4000

(iv) S = P → next → next → next → next → next → next → next → next → next;

(v) P = S → next → next → next → next → next → next → next → next → next → data;
= e

(vi) Pf(P → next → next → next → next → next → data);



(i) Struct node *P

(ii) P = S → next → next → next → next → next;

(iii) P → next = S → next → next; = 3000

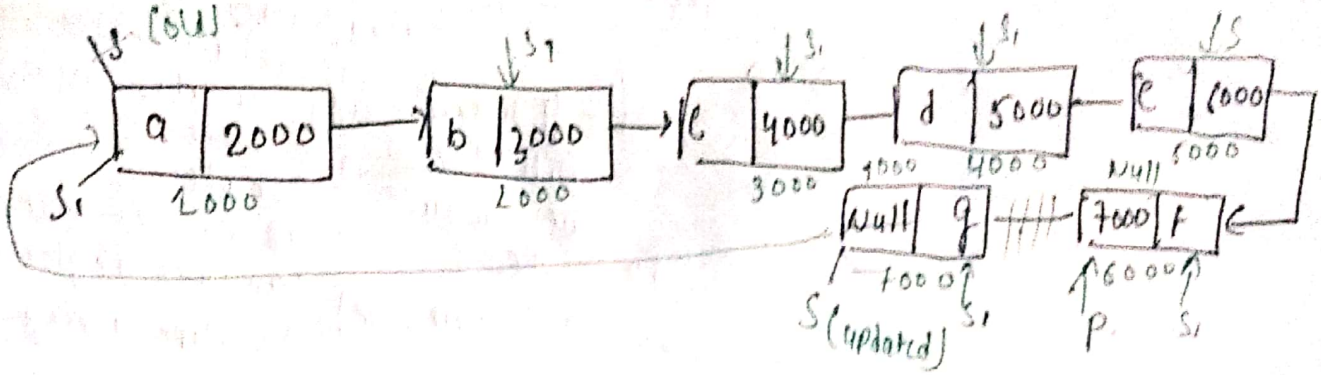
(iv) P = P → next → next → next → next → next → next → next → next → next;
= 3000

(v) S = P → next → next → next → next → next → next.

(vi) P = S → next → next → next → next → next → next

(vii) Pf(P → next → next → next → next → next → next → data);
= e

Ans: Write a C program to move last word of the linked list to the front of the linked list.



Struct node *

function returning a address.

```
Last-2-first(struct node *S)
{
① if (S==Null) || (S->next==Null)
     return(S);
② S1 = S
   while (S1 -> Next != Null).   // or (S1->next->next!=Null)
   {
      P = S1
      S1 = S1 -> Next;
   }
③ P -> next = Null
   S1 -> next = S (old S)
   (updating S) S = S1
   return(S);
```

order is important.

Ans: Write a C-program to Insert a Node with data-x at the end of the given single linked-list

```
Insert-x-at-end(structure node *S, int x)
{
struct node *P;
```

① `p = (struct node)malloc(10B)` → 8000

It will creating the memory of 10B in Heap area as well as returning the address also where the mem. will be creats.

Diff system Diff size

So best way to write `sizeof(struct node)`

Void Pointer - you can store anything. So typecasting.

`p = (struct node)(Type-cast). malloc(sizeof(struct node))` 10B

8000 (address onwards)

8000

```
if (p==NULL) return(p);
p->data = x
p->next = NULL
```

p | G | 8000

By Default Garbage value while using malloc.

If you are using calloc instead of malloc it will clean to zero.

② `if(s==NULL) s=p; return(s);`

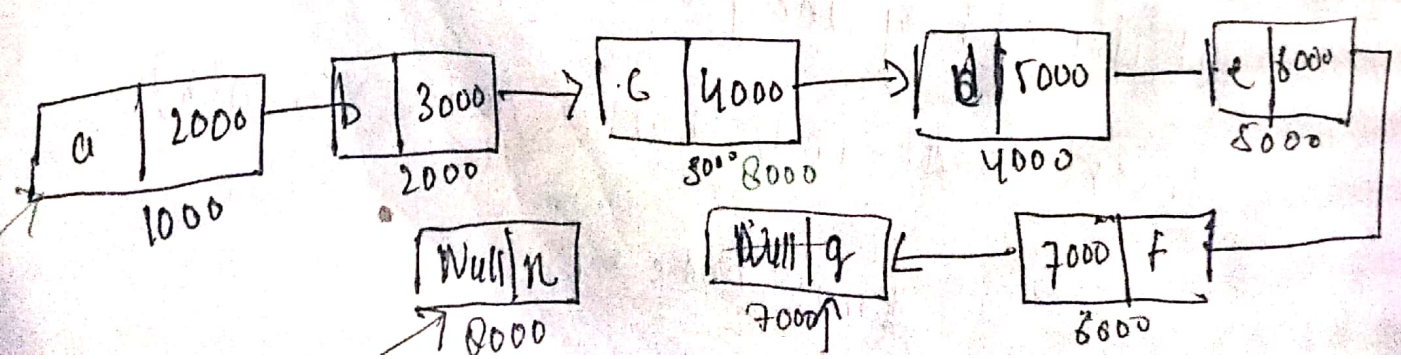
③ `s1 = s`

```
while(s1->next != NULL)
    s1 = s1->next;
s1->next = p
return(s1)
}
```

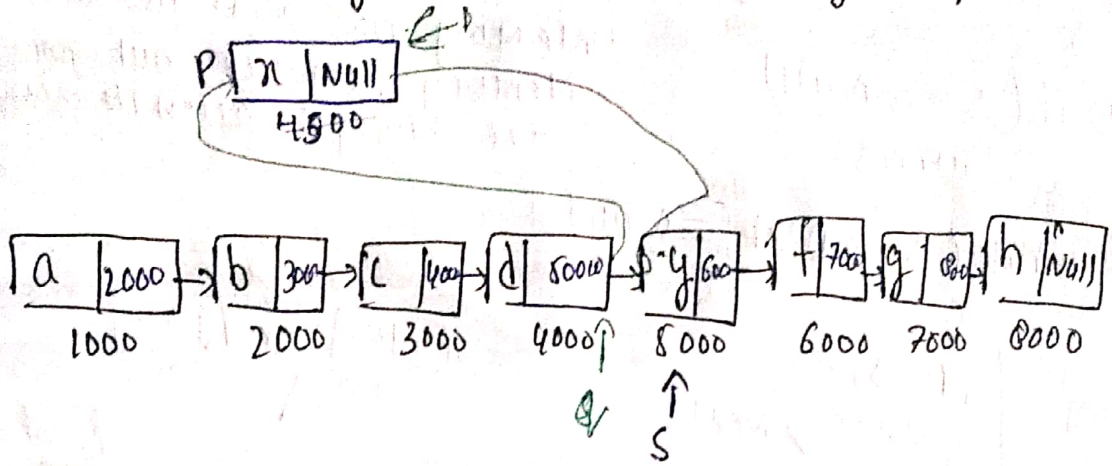
If you want to allocate memory for 5 elements in array.

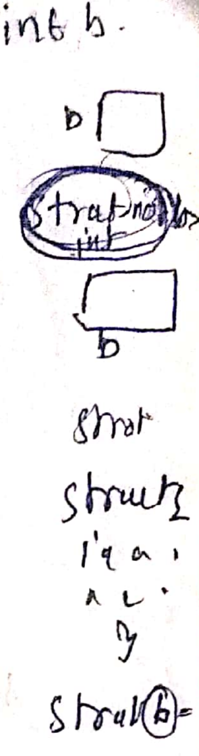
Malloc: `(int*)malloc(5*sizeof(int));`

Calloc: `(int*)calloc(5, sizeof(int));`



Inster Inserting a node data-x before y

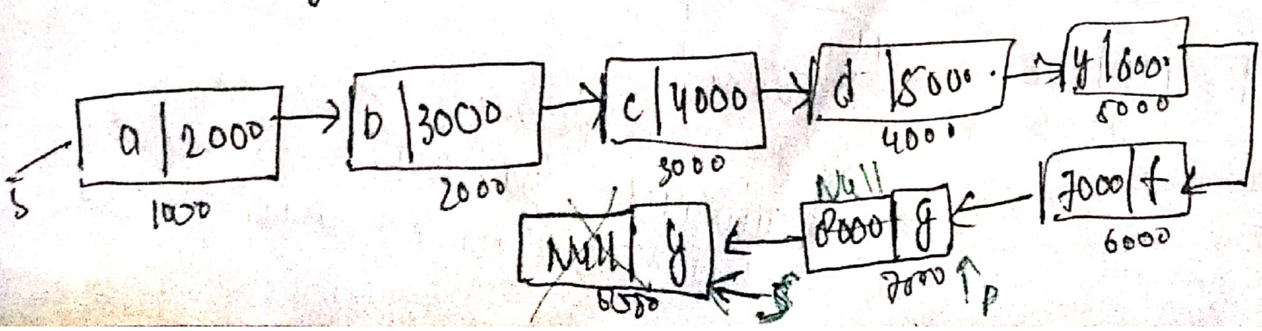





```
while (s -> data != y && s -> next != Null)
{
    q = s
    s = s -> Next;
}
if (s -> data == y)
{
    q -> next = P
    p -> next = s
}
else
{
    Pf(No-y -- so-no-x)
}
}.
```

```
if (s -> data == y)
{
    p -> next = s
    s = P
}
```

ans: Write a C program to delete a node at the end of the given. singlele linked list.



Delete-at-end(S)

{

① if(S==Null)
return(S)

② while(S→Next!=Null)

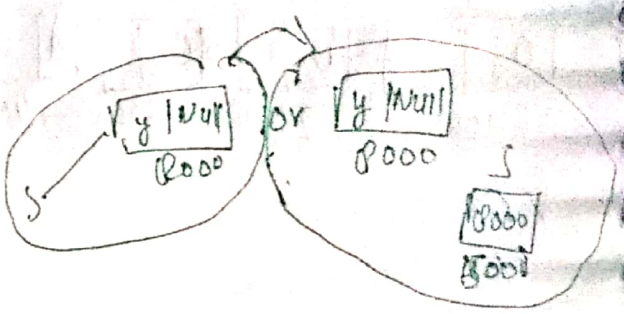
{

P=S;

S=S→Next;

}

Segmentation error:
↳ Memory not available in Memory heirarchy but you are trying to access the memory



② P→next=Null

free(S)

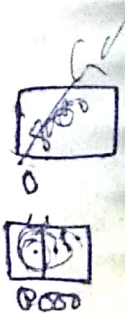
Before ⇒ Pf(S)=8000
Pf(S→data)=y
Pf(S→Next)=Null

after ⇒ Pf(S)=8000
Pf(S→data)=>Garbage
Pf(S→next)=garbage

Pf(s)= ← S=Null

Pf(S)=Gr

Pf(S→data)⇒ Segmentation error.



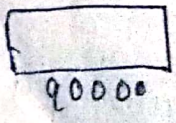
Dangling Pointers: Pointer to that Mem. address which is alreadydeallocated.

P=Malloc
8000

→ Not given or Not using.

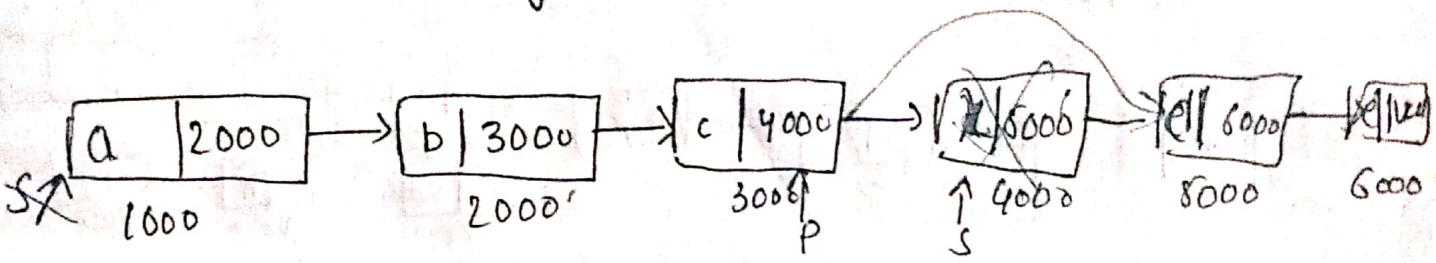
Memory leakage.

P=Malloc
9000



Now P is pointing to 90000
afterwards you want to use 8000.
but there is no way to use.

Ques: Write a C Program to delete a node with data x in a given linked-list.



① → ② while (s → data != x && s → next = Null).

$s_1 = s$

{ p = s
$s_1 = s \rightarrow next;$ }

if ($s_1 \rightarrow data = x$)

{
$p \rightarrow next = s_1 \rightarrow next$
free (s).
$s_1 = NULL$
}

It is working for after one node (coz we don't know the value of p)

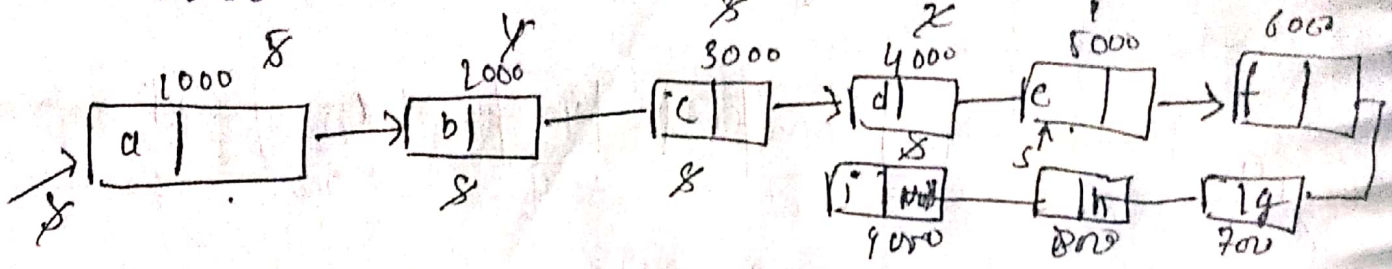
① if (s → data == x) } If the deleting node is the first one.

$s_1 = s$
$s = s \rightarrow next;$
free (s_1)
$s_1 = Null$

① & ② all working for all the nodes.
linked list.

Upto now this example is enough for Gate Purpose.

Ques: Write a C-Program to find Address of Middle Node in the given linked list.



Algo① $C_1 = 0$ $S_1 = S$

① while (S_1 != null)
{
$C = C + 1$
$S_1 = S_1 \rightarrow next;$
}

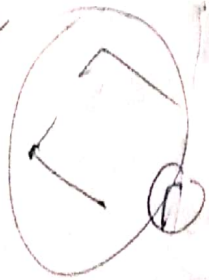
② $C = \lceil C/2 \rceil$

③ while ($C != 1$)
{
$C = C - 1$
$S = S \rightarrow next;$
}

return (C)
}

$n + n/2 = 3n/2 = O(n)$

$\frac{11}{2} = 5$



Algo②:

P	Q
1000	1000
2000	3000
3000	5000
X1	X2

① p = q = s.

② while(q != NULL && q->Next != NULL && q->next->next != NULL)

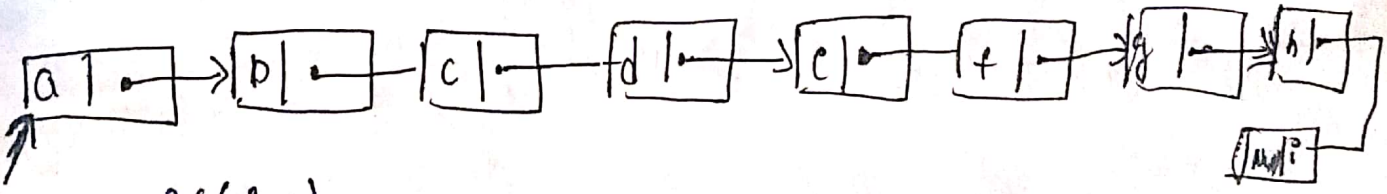
n/2 { p = p->Next; (thinking about Middle). (going by 2 & slowly.) ⇒ O(n)
q = q->next->next. (thinking about size) fast termination condition
Going x2 (Double)
}

return(p)

}

first it will check. after that 'q' 2-node available. then p will be. inc. by 2 & q will be by 2.

Ex: Write a C-program to perform binary-search on a given linked list.




```
BC(S,n)
{ if(s->next==NULL)
  {
    if(s->next==n)
    {
      return(s)
      else
      return(NULL).
    }
  else.
  {
  p = Mid(s)
  if(p->data==n) return(p).
      else.
```

= O(n)

```
else
if(x<p->data)  { p->next=NULL }  => O(1)
else           { BS(p,x)      }  => T(n/2)
BS(p->next, x) ==> T(n/2)
}
}
```

$$T(n) = T(n/2) + n$$

$$\vdots$$

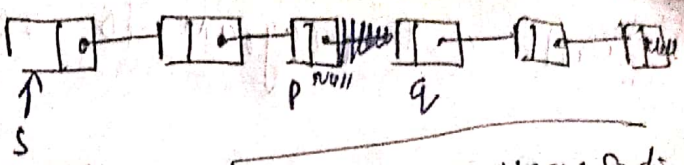
$$O(n)$$

Binary search on linked list is possible but not efficient.

---

ans: Write a C program to perform Merge sort on the linked list.

```
MergeSort()
{
  if(s->next==Null)
    return(s)
  else
  {
    p = Mid(s)        } n
    q = p->next;
    p->next = Null
    s1 = Merge(s)     } 2T(n/2)
    s2 = Merge(q)
    s = Merge(s1, s2) ⟹ n  (Inplace)
    return(s)
  }
}
```



Normal Merge sort

$2T(n/2) + n$

$= O(n \log n)$

space

$n + \log n$

T.C = both are same.

$2T(n/2) + 2n$

$2n \log n$

$= O(n \log n)$

Space

$\log n$

Ques: Let P be the singly linked list. What will be the Complexity of the best known algorithm to check whether the linked list is palindrome or not?

a) $O(n^2)$ b) $O(n)$ ✓

b) $O(n \log n)$ d) $O(\log n)$

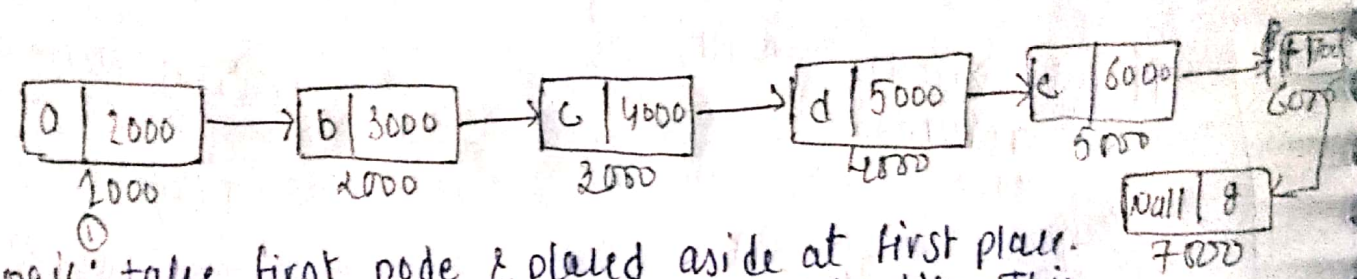
Sol: ① Find the midle element = It will take $O(2)$ time.

② Reverse the element of the list from the middle element till end.

③ Compare the two halves of the list.

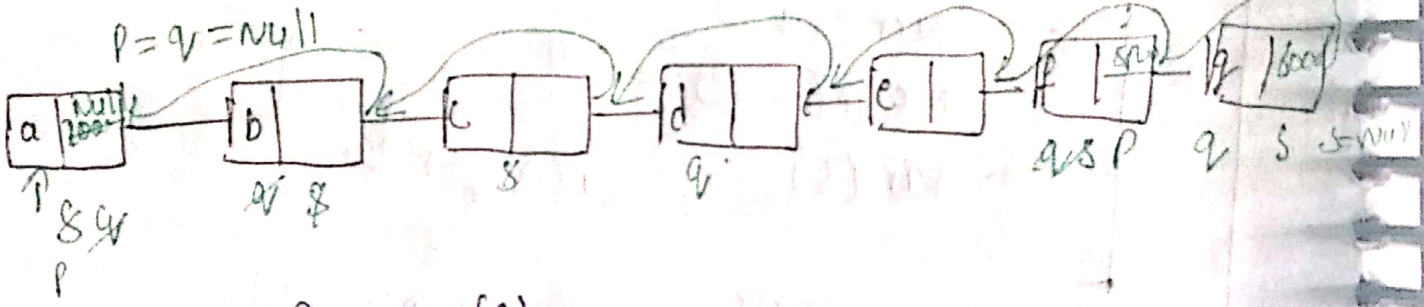
This algo will take $O(n)$ time Complexity

Ans. Write a C-program to reverse the given single list.



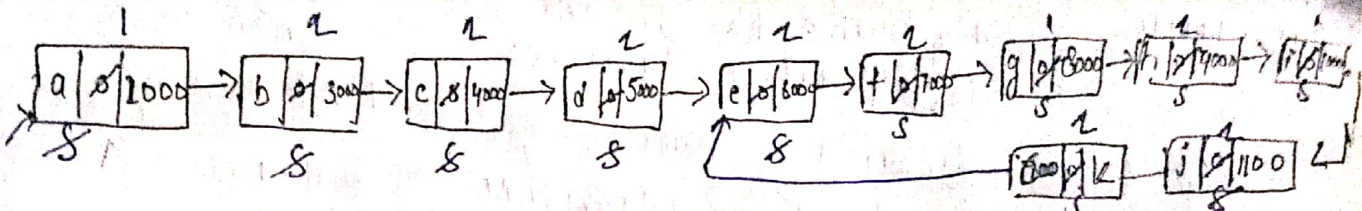
Logic: take first node & placed aside at first place. & then take 2nd placed at first. like this

Logic②: ① 3 node required
② reverse middle node only.




```
Reverse(S)
  P=q=Null
  while (S!==Null)
   {
   P=q
   q=s
   s=s→d.
   q→next=P
     {
      s=q.
     }
    return(s)
```

ans: Write a C-program to find cycle in a given linked-list.



Algorithm ① (BFT or DFT)

① Assume every node contain one more extra field flag. which is zero initially.
using calloc.

② Visit every node & check flag.

```
if(flag==0)
    flag 1
    else
PS(loop is found);
}
```

extra space
⇓
O(n)

Time ⇒ O(n)

Algorithm ②:

Take pointer P & Q starting initially at same place.

① P = Q = S.
② P - by - 2 (×2)
③ Q - by - 1 (×2)
④ while(P != Q)
{
P - by - 1
Q - by - 2
}
Pf(loop found);

O(n)

doubt single node

- Intersection of two linked list of size n & n.

Procedure: ① Sort both the linked list. $O(n \log n)$
② Apply Merge Algorithm. $O(n+n)$
③ Similary union of two linked also, $= O(n \log n)$.
④ Membership Algorithm of linked list.
⑤ Cardinality Algo. For one time.

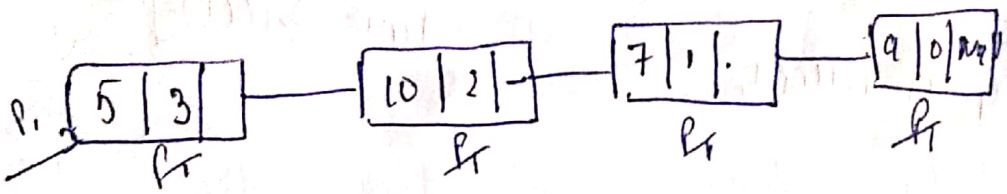
This expression is possible using circular list

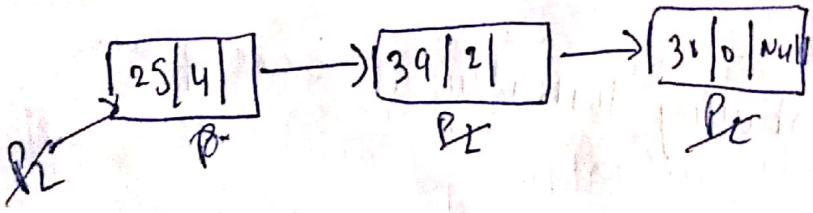
Polynomial Addition using L.L

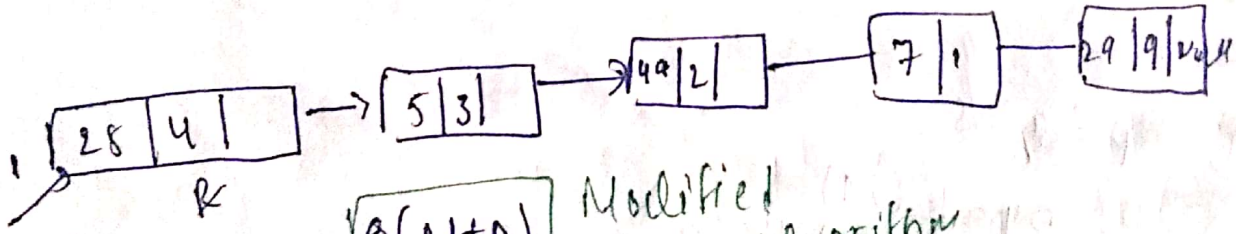
Eg: P_1: $5x^3 + 10x^2 + 7x + 9$
P_2: $25x^4 + 39x^2 + 30$

Comparing which is greater, then take it based on the power

$P_1 + P_2$: $25x^4 + 5x^3 + 49x^2 + 7x + 39$





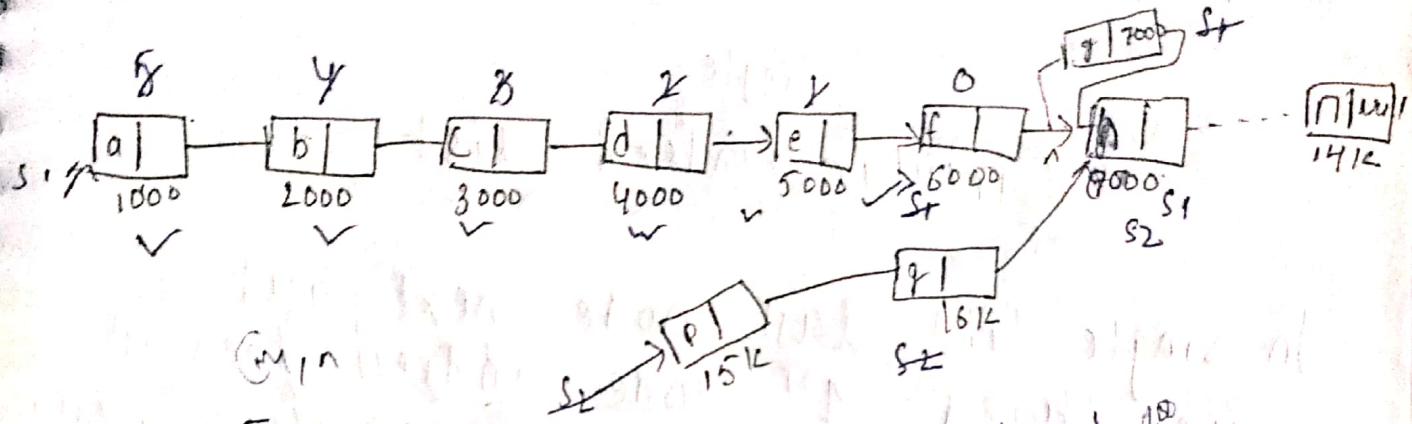


$O(M+N)$ Modified Merge algorithm.

Polynomial Multiplication using linked list will take $O(M \times N)$ times

~~Drawbacks of Single List~~

Ques:- S_1 contain M-elements S_2 contain n-elements to find 1st intersecting node address. to find How much time?



M n
5 14-9

① Difference = M − n (S1 move up to 5 place). M is greater by 5. first element the greater one & then onwards.

(May so it will move.

② while (D != 0)

```
{
S1 = S1 → next
D = D - 1
}
```

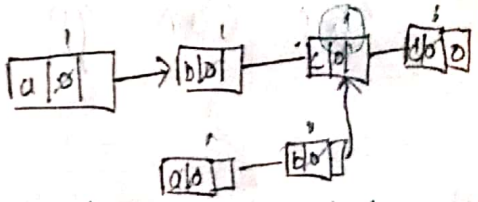
O(m)

while (S1 != S2

```
{
S1 = S1 → next
S2 = S2 → next;
}
```

Max(M,n)

Another Algo:- Take one flag field extra



If flag node 2- found. ~~that node~~ return address of flag node.

Drawbacks of Single Linked List;

① We cannot Go back. Because Every Node contains next node address. So we cannot go back.

② Single Linked List Last Node next part always Null thats means we are.

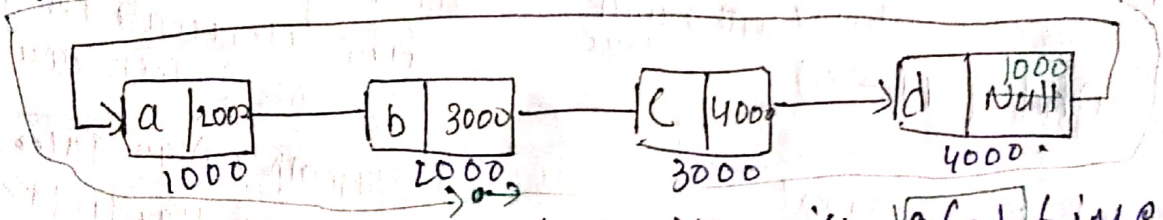
Not utilizing properly.

To Eliminate above Drawback we are going to the Circular Linked List.

Single.

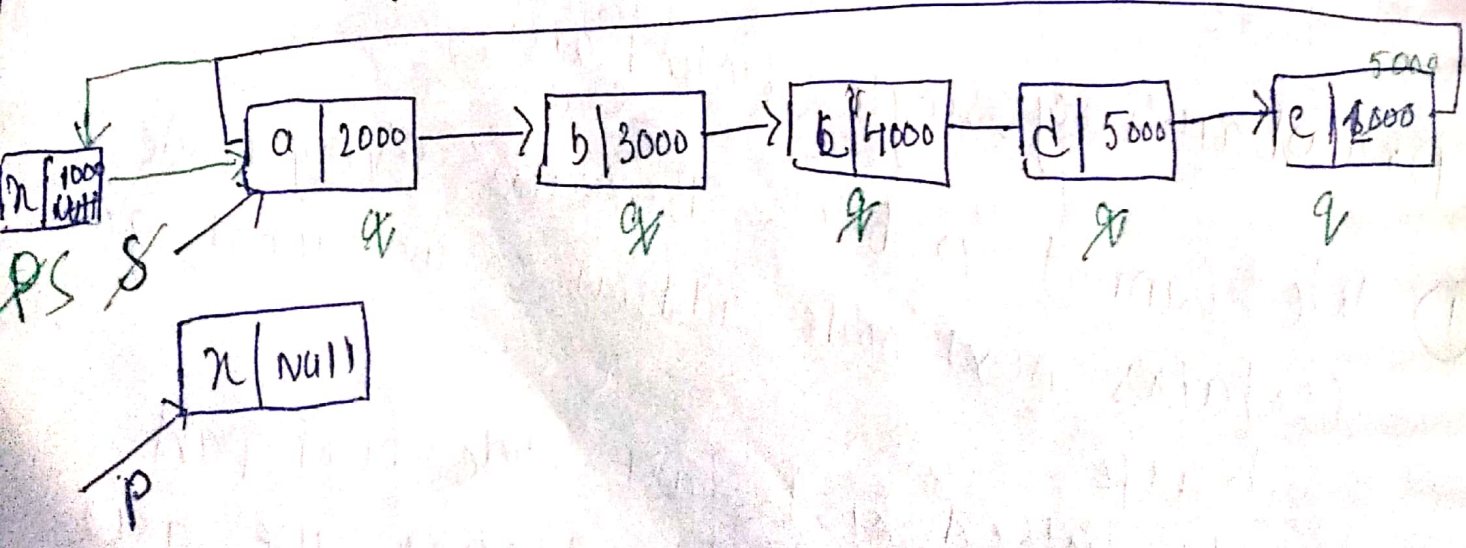
Circular Linked List

In Single list last node next part if we stored 1st node address then it is known as Circular Single linked list.



We can go back but it will O(n) time.
Possible but it will take more time

Qus: Write a C-Program to Insert a Node at the start of the given Circular linked-list.



① P = malloc

② q = s

```
while (q -> next != s)
        q -> q - next
```

③ p -> next = s

q -> d = p

s = p

return(s);

Time Complexity: O(n)

Delete Last Node in Circular Single Linked list:

① P = S

```
while (P -> next != s)
{
    q = P.
P = P -> next
}
q -> next = s
free (P)
P = Null
return(s)
```

Double Linked List:

It will take less time But taking more space

[Any node previous null then it is 1st Node or any node next null is Last Node]

```
Struct DLL
{
struct DLL *prev;
int data;
Struct DLL *next;
}
```

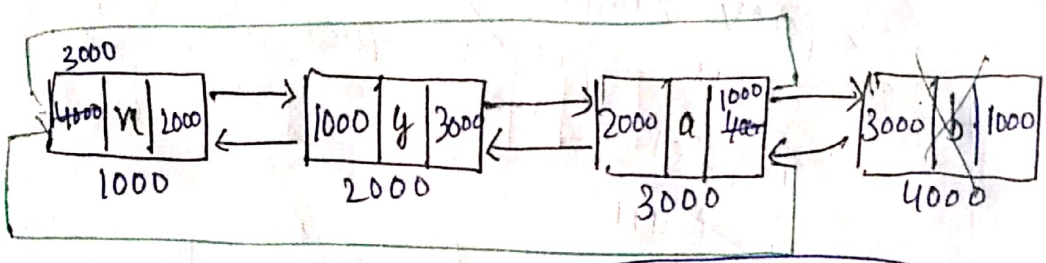
18B.

p → prev → next = p.

add-in-begin.

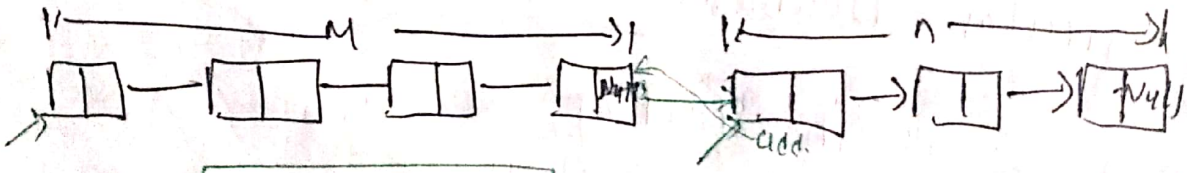
Deleting last node:

```
p = s → prev;
p → prev → next = s
s → prev = p → prev.
free(p)
p = Null
```



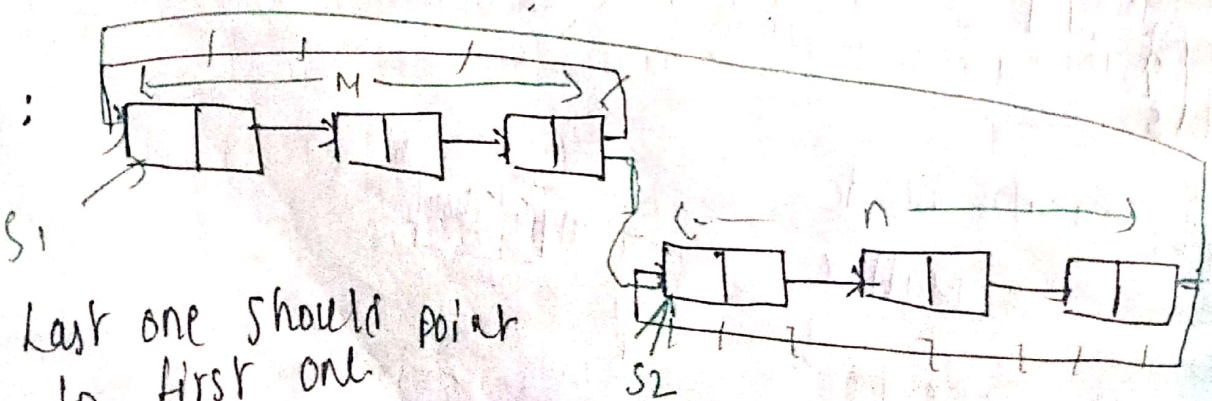
Concatination: $(S_1 \cdot S_2)$: S_1 followed S_2 (End of S_1 should be connected to first node of S_2).

SLL



T.C: O(M) [EC]

C.S.L:

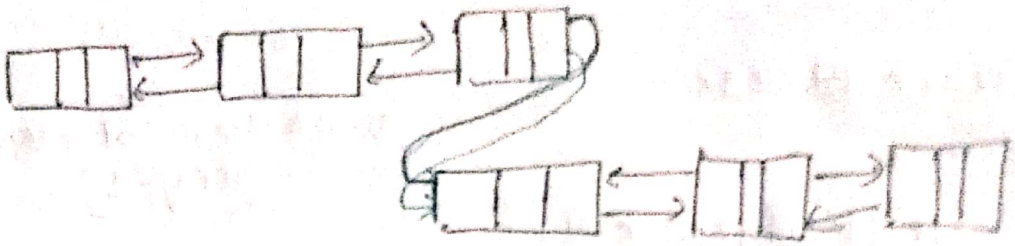


Last one should point to first one.

S_2 should point to S_2

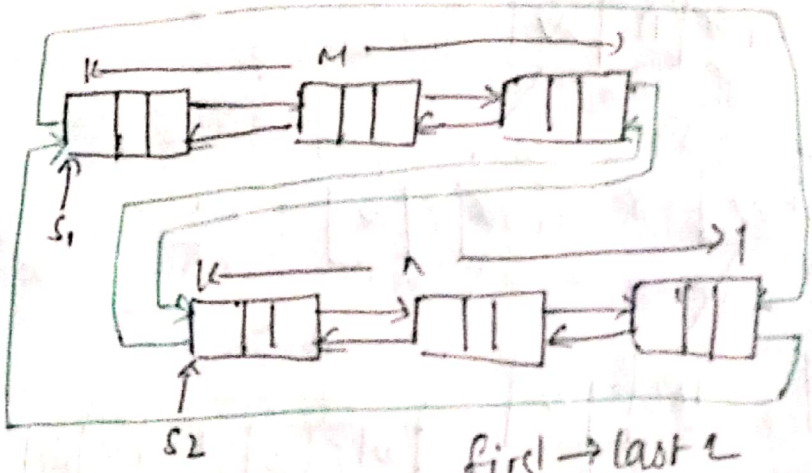
T.C: (O(M+n)) (EC)

DLL:



T.C = O(M) E.C

C-DLL:



T.C: O(2) E.C

first → last 2
Last → first 2
So O(1).

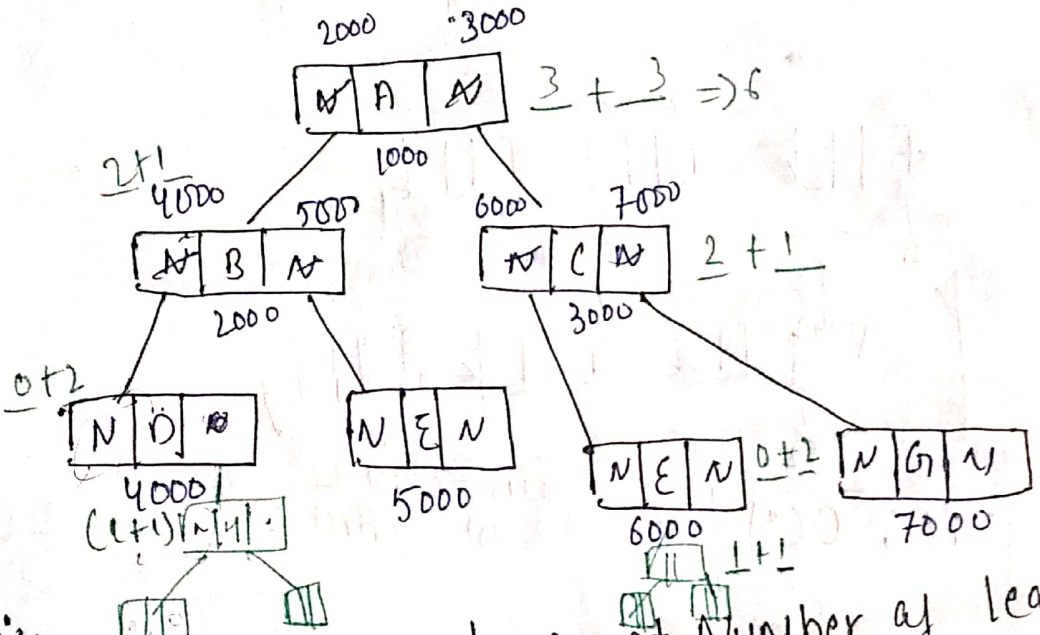
NOTE: The Greatest Advantages with a Double or Circular Double Linked list is if we lost one pointer. then with the help of another pointer we can recover it or we can get it back.

Binary Tree:

Linked List are not suitable for Binary search.

No of nodes in a BT of degree two.
ans = [leaf - 1]

```
Struct BtNode
{
  Struct Btnode *lc;
    int data;
  Struct Btnode *rc;
}
```



ans: Write a C-program to count Number of leaf nodes in the Given binary tree.

```
leaf count
CNLF (struct Btnode BinaryT *r)  ⟹ T(n)
{ ① if(r==null) return(0) {If the tree is empty & No leaf & No root}
② if(r->lc==null && r->rc==null)  (Doubt)
    return(1)
else
a = CNCF(r->lc)  ⟹ T(n/2)
b = CNLf(r->rc)  ⟹ T(n/2)
c = a+b; }
```

Non-leaf nor count

return(0) }

}

$2T(n/2) + c$

$= O(n)$ [EC]

Every node you have to ask (even you leaf node or not atleast 2 times).

Stack space

$\Downarrow$

Min $\Rightarrow \log n$

Max $\Rightarrow n$.

If you are counting Non-leaf Nodes

then if(r->lc==null && r->rc==null) ..

if part (return(0)

else

c = 1

17-Sept-2019

Write a program to Count Number of Internal-Nodes or Non-leaf nodes. in the Given binary tree.

```
CNIN(R)
{
① if(R==Null)
    return(0)
② if(R->left==Null && R->rc==Null)
    return(0).
{
a = CNIN(r->lc)
b = CNIN(r->rc)
```

```
    c=11+a+b
    return(c)
    }
}
```

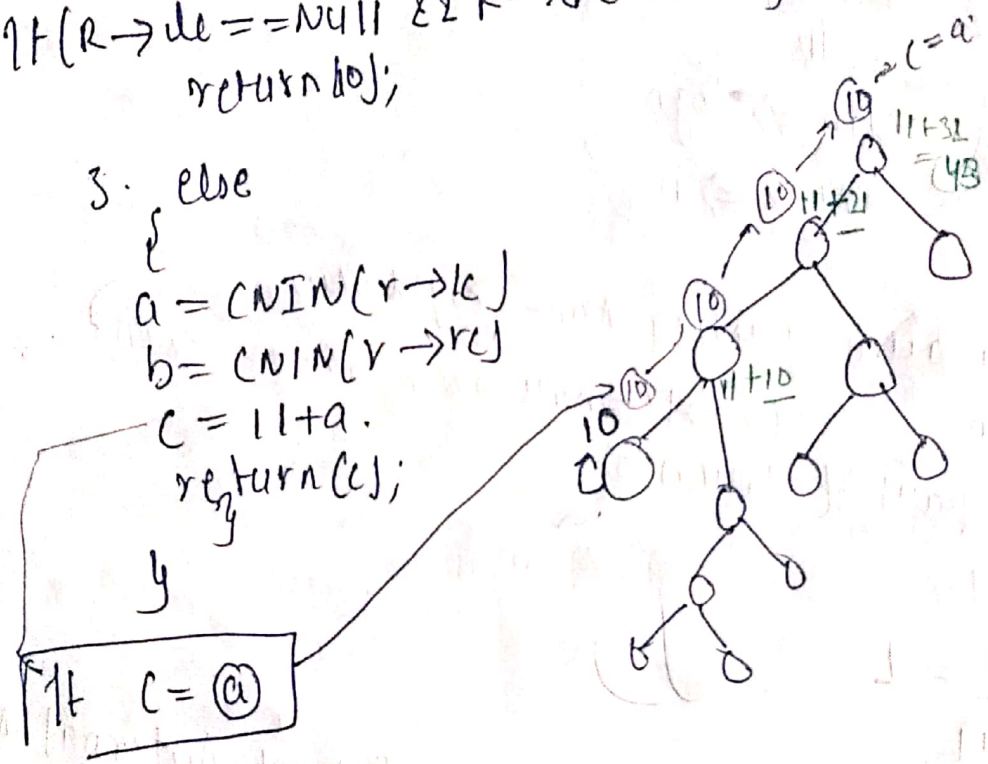
Counting Number of Nodes in the ~~right~~ left most ~~side~~ path.

① if(R==Null)
Return(0)

② if(R->le==Null && R->rc==Null)
return 1;

3. else
{
a = CNIN(r->lc)
b = CNIN(r->rc)
c = 1+a+b.
return(c);
}

}

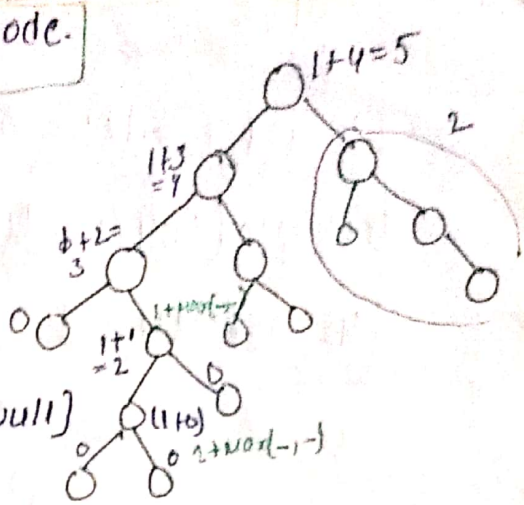


Ques: Write a C Program to find height of a Given Binary tree.

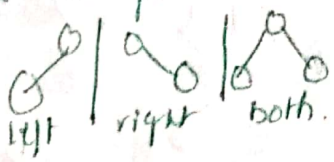
Height of a Binary tree is nothing but height of the root.
The longest distance from root Node to leaf Node.

" Height of a Node = The longest distance from that Node to leaf Node.

Height of BT = Height of root Node.




```
HOBT(R)
{
① if(r==Null)
    return(0);
② if(r->lc==Null && R->rc==Null)
    return(0);
    else.
③ a= HOBT(r->rc)
   b= HOBT(r->rc)
   c= 1 + max(a,b)
      Because of Non-leaf one edge always there.
   return(c)
  }
}.
```



If you want to find No of levels in Binary Tree.

```
② if(r->lc==Null && R->rc==Null)
     return(0+1); or Height of BT+1
```

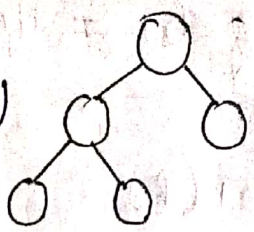
T.C: O(n) [EC]

Program: Write a C-program to verify Given Binary Tree. is strict Binary tree or not.

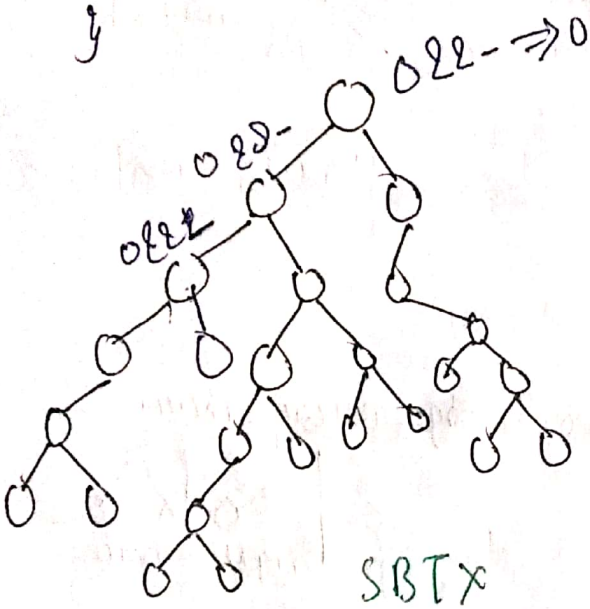
Strict Binary Tree: Every Node having '0' or two children.

```
SBT(r)
{
① if(r==Null)
    return(1);
② if(r->lc==null && R->rc==Null)
    return(1);
```

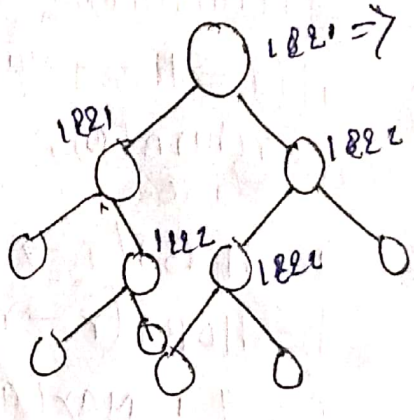
else
{
3. if(r->lc!=NULL && r->rc!=NULL)



return(SBT(r->lc) && SBT(r->rc))
4) return(0). (Non-leaf fail return 0, no need to check further).
}
}



SBTX



SBTV

Write a C-Program to verify Given two-Binary Tree are equal or not.

⇒ V2BT(r1, r2)

① if (r1==Null && r2==Null)
return(1);
~~else~~
~~return(0);~~

~~②~~

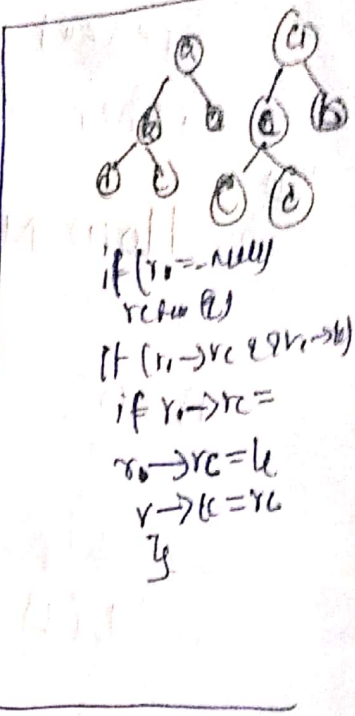
② if((r1==null && r2!=NULL) || (r1!=NULL && r2==NULL)
return(0).

③ if(r1->data == r2->data)

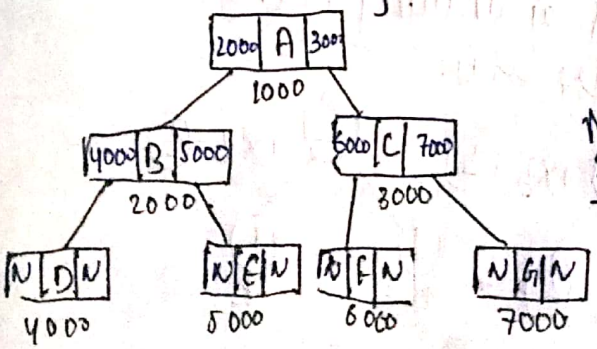
return(V2BT(r1->lc, r2->lc) && V2BT(r1->rc, r2->rc)

④ return(0);
}

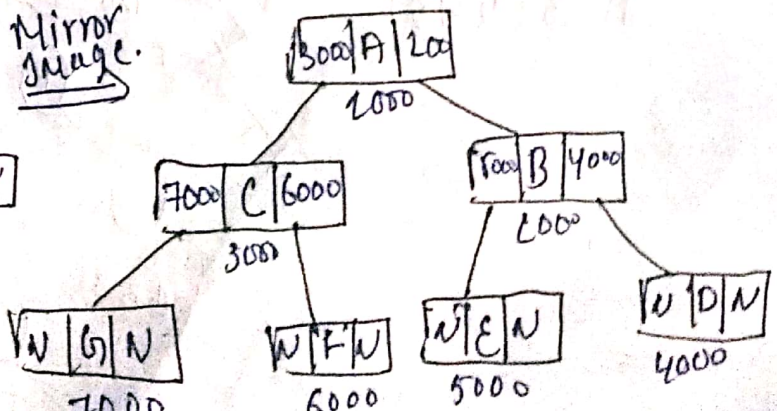
Ques: Write a C Program to convert Binary tree into its Mirror image.



Mirror-Image(r)
{
① if(r==NULL) [Nonode & Empty]
return(r);
② if(r->rc==NULL && r->lc==NULL);
return(r). [only one node]
③ else. [contain non-leaf node]
{ if(r->rc!=NULL && r->lc!=NULL)
r->rc=lc.
r->lc=rc.
t=r->rc
r->rc = Mirror-Image(r->lc)
r->lc = Mirror-Image(t)
return(r)
}
}



Mirror Image.



Binary Search ~~Order~~ Tree's (BST). left = smaller, right = greater.

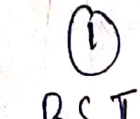
⇒ In the given Binary tree. at every node. Comparing root data all elements present on the left sub tree is smaller. & all elements present on the right its side is greater. then it is known as BST.

⇒ How many BST is possible with n-nodes.

n=0 — 1-BST, Empty set

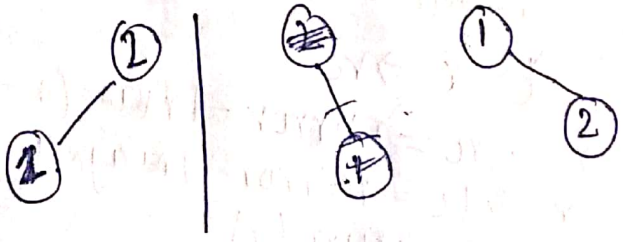
---

n=1 (1) — BST



---

n=2 (1,2)

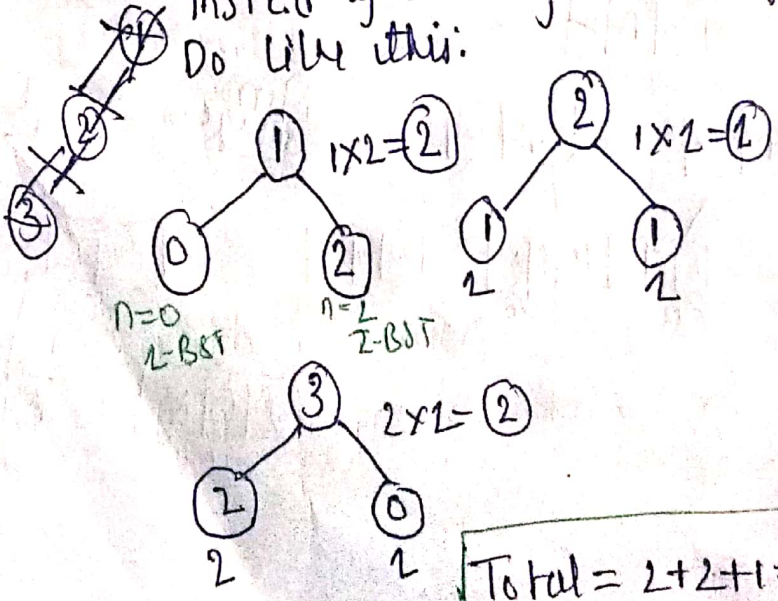


2-BST

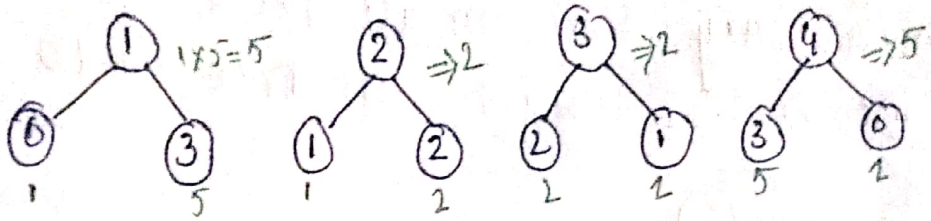
---

n=3 (1,2,3)

Instead of drawing manually 5 BST Do like this:

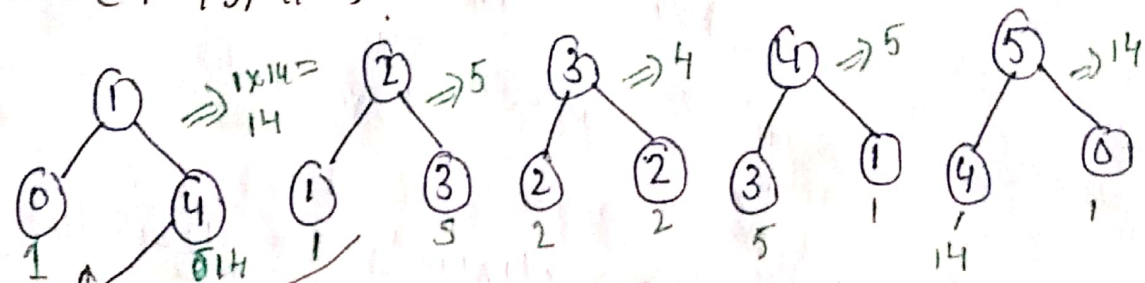


n = 4 (1, 2, 3, 4)



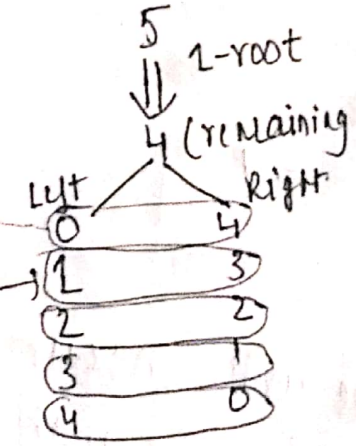
Total = 5 + 2 + 2 + 5 = 14

n = 5 (1, 2, 3, 4, 5)



indicate with 4 node how many BST possible

Total : 14 + 5 + 4 + 5 + 14 = 42



$$BST(N) = \sum_{L=0}^{N-1} BST(L) \cdot BST(R)$$

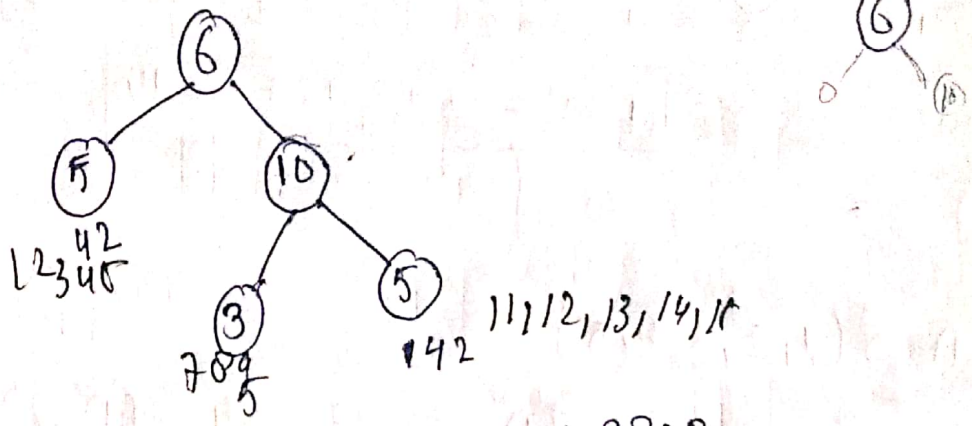
$$L + R = N - 1$$

$$R = N - (L + 1)$$

or You can use = $\frac{2n C_n}{n+1}$ $\frac{10C_5}{6} \Rightarrow$ $\frac{10 \times 9 \times 8 \times 7 \times 6}{1 \times 2 \times 3 \times 4 \times 5 \times 6}$

$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10$

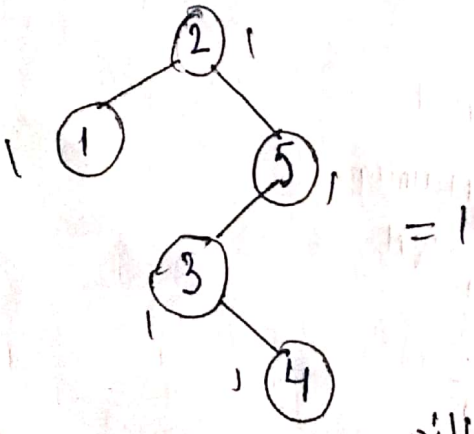
Ques: How many BST possible with 5 nodes (1, 2, ... 5) in such a way that in all those BST 6 will be the root & right sub tree is 10.



$42 \times 5 \times 42 \Rightarrow 8820$

Ques: i/p: Set S with 5-elements $S = \{1, 2, 3, 4, 5\}$
unlabeled Binary-Tree with 5-nodes.

Double



Ques: No. of way we will insert data from S into U.L.B. So that it become BST.

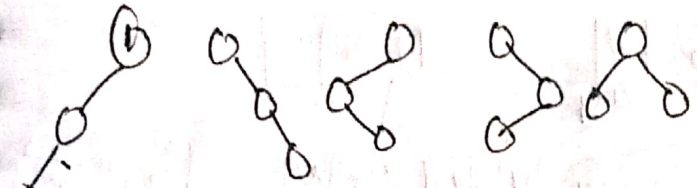
Ans = 1-way

NOTE: No. of BST's with n-nodes = No's of Unlabelled BST's with n-nodes. $= \frac{2^n C_n}{n+1}$

⇒ No of Binary tree = No^n of BST's * n!

n = 3

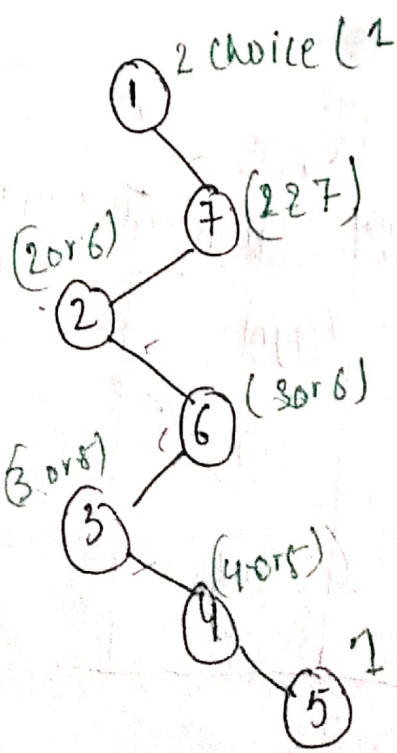
No of unlabelled BST possible (5).



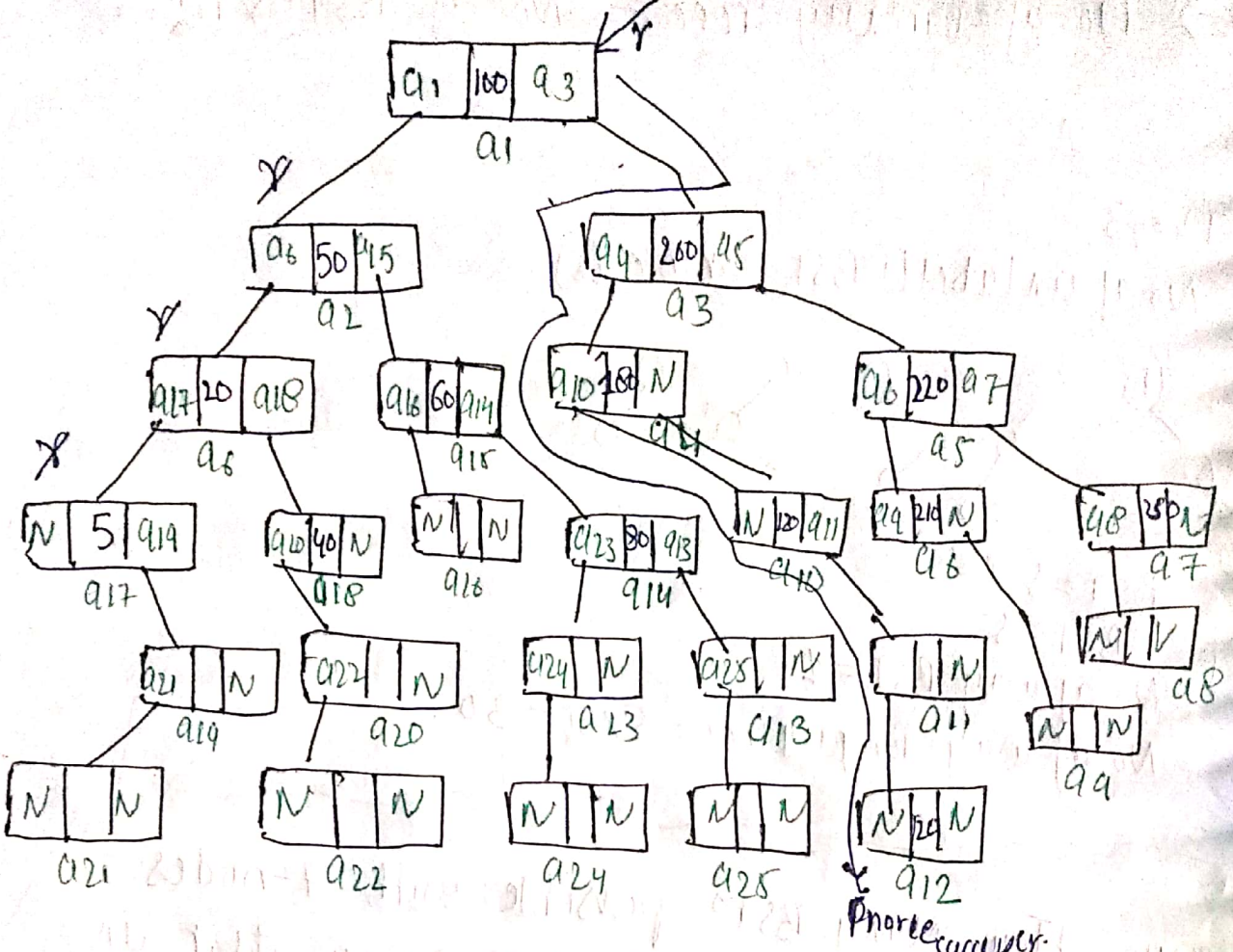
BST = 5
UBST = 5
No of way BST = 2
No of way Binary Tree = 5 × 3! = 30.

---

ans: How many BST's possible with 7-nodes. {1, 2, 3, ... 7} in such a way. that in all those BST's height will be 6



$= 2^6 = \boxed{64}$



ans: How much time it will take to find minimum element.

TC- Find-Min

	BC	WC	AC
BST	O(1) left side no Element	O(n) always at one side	O(logn) Middle Comparing any BST Most of it contain that type of behaviour.
AVL	O(log(n)) Balanced No Element left side not possible	O(log(n))	O(logn)
BT / always had unsorted array	O(n) Min can be every where.	O(n)	O(n)
Max Heap	O(n) you have to search all the node	O(n)	O(n)

Always equal to unsorted array

Coz it is meant for max.

Pblm: How much time it will take to find element "x"

Time Complexity:

	BC	WC	AC
BST	O(1)	O(n)	O(log(n))
AVL	O(1)	O(log(n))	O(logn)
BT	O(1)	O(n)	O(n)
Max Heap	O(1)	O(n)	O(n) – only every path.

Pblm:
How much it will take to you find element x is not existed..

	BC	WC	AC
BST	O(logn) Right is null	O(logn)	O(logn)
AVL	O(logn)	O(logn)	O(logn)
BT	O(n)	O(n)	O(n)
Max Heap	O(1)	O(n)	O(n)

→ root=100 search=120

root=500 (max) search=1000 Not there.

Insertion Element in BST:

Insertion Algo:

① Create node - x ⇒ O(1)

② Find correct position of x ⟶ O(n) [WC]

③ Connect both ⇒ O(1)

O(n) [WC]
O(1) [BC]
O(logn) [AC]

To Create BST for n nodes. Time Complexity is $O(n^2)$ worst Case..

$O(n^2) \Rightarrow$ worst case

$O(n \log n) =$ Best Case & Avg. Case.

Deletion:

In Min-Heap tree Inserting element will take.

BC = $O(1)$

WC & A.C = $O(\log n)$

In Min-Heap tree you have to insert element. n. if n not exist.

T.C = $n + \log n$

$= O(n)$

Inorder: 5, 10, 15, 20, 25, 28, 40, 50, 55, 60, 63, 65, 80, 85, 90, 90, 100, 120, 130, 150, 180, 200, 210, 215, 220, 230, 250.

Inorder predecesser

Inorder successor

Find the Element n. Go 2-time left. & then go to highest right. will null called predecesser. Similary successer.

Deletion Algorithm:

(1) Find - Node - n

(2) (i) 0 - child $\Rightarrow$ Simply delete by adjusting few lines $\Rightarrow O(1)$

(ii) 2 - child $\Rightarrow$ Simply - Delete by connecting Grand-father & grand child.

$\Rightarrow O(1)$

(iii) 2-children ⇒ Replace n- data by predecessor data & successor data & delete.
predecessor & successor
a+b = O(n) [WC]
a+b = O(1) [BC]
a+b = O(logn) [AC]

10-|Sept-|2019

AVL-Tree (or) (Height Balanced BST)

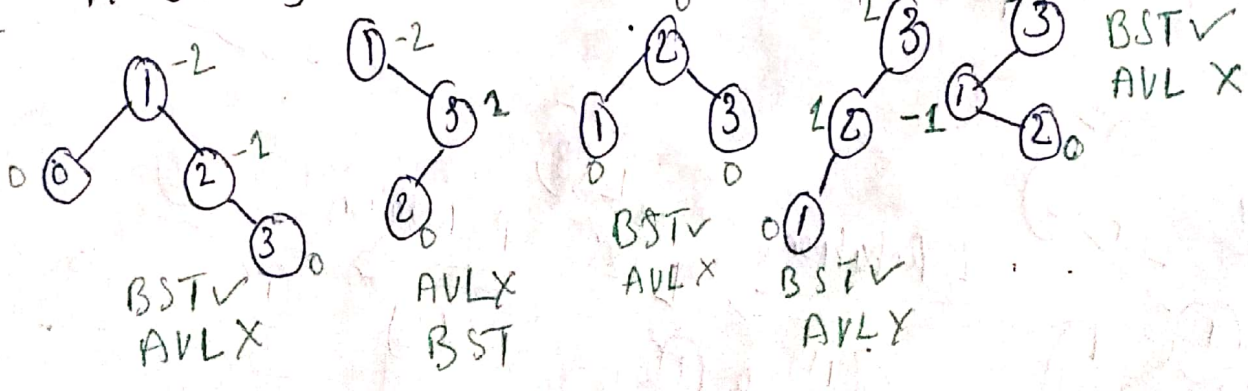
Min = logn
Max = n
logn - Always.

Ques: When can I say a BST is a Height Balanced BST
At Every node balancing factor is -1, 0, +1 then
it can say that it is AVL Tree.

In BST
At Every Node → Balanced factor is -1, 0, +1

H(LST) − H(RST).

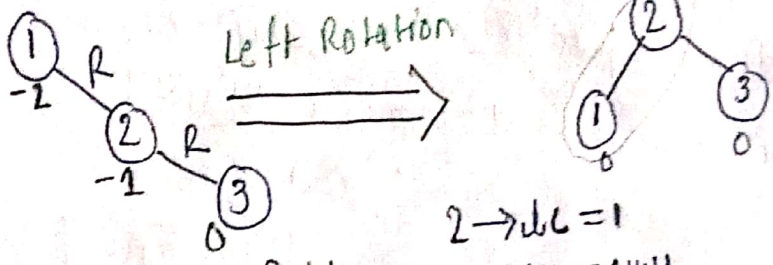
Ex ① n=(1,2,3)



In AVL Tree Max. Difference is 2 Only
Only 4 rotation there
[LL, RR, LR, RL]

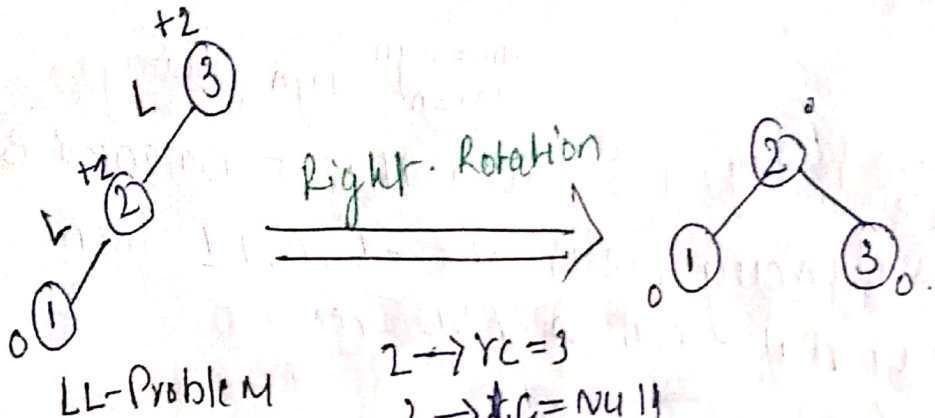
Rotations:

① :



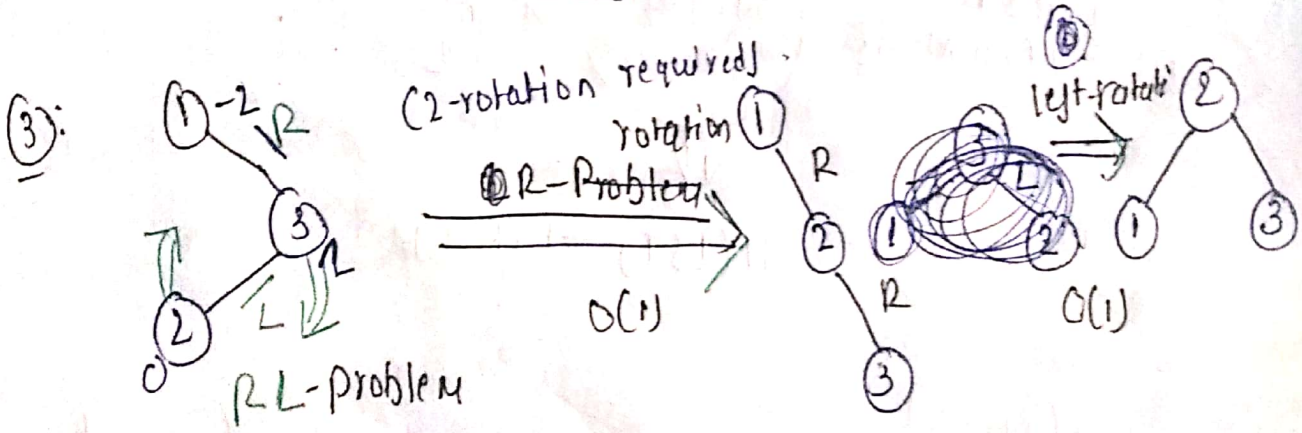
2 → lc = 1
1 → rc = Null
O(1)

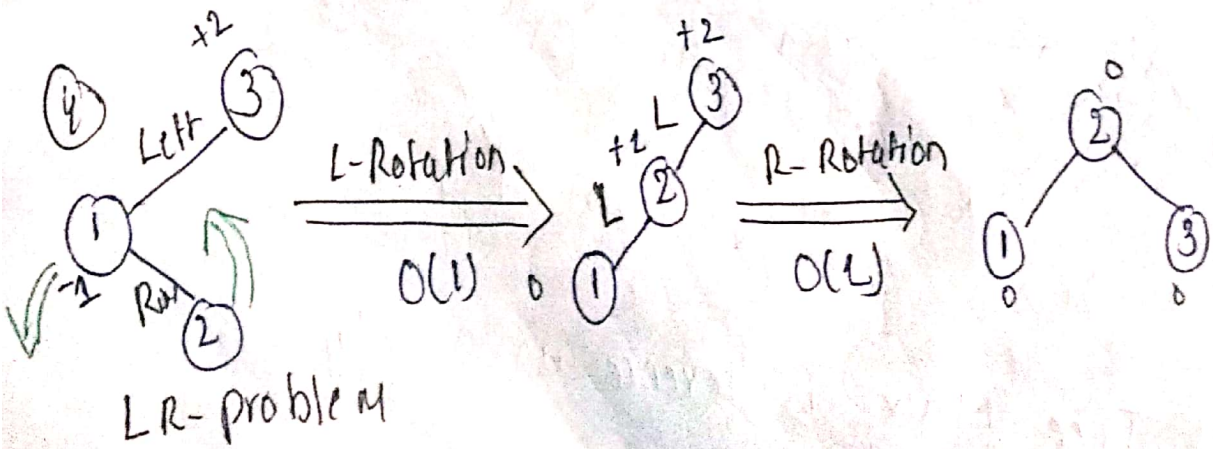
② :



2 → rc = 3
3 → lc = Null
O(1).

③ :

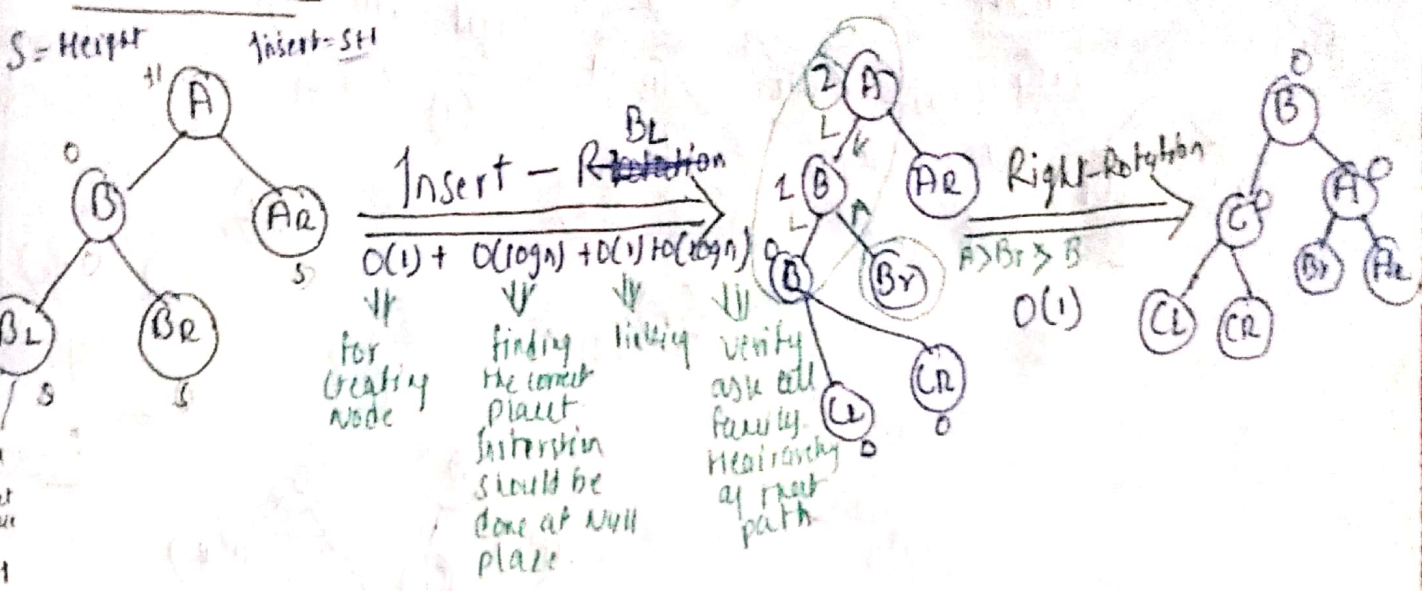




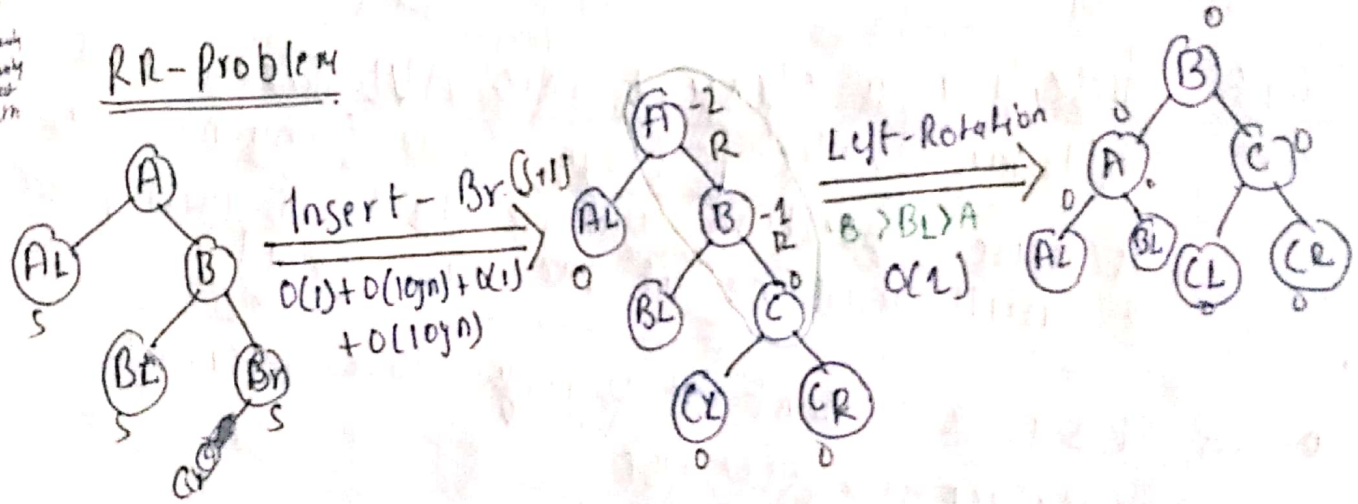
Insertion in AVL Tree:

In AVL BST Insertion should be done at NULL place.

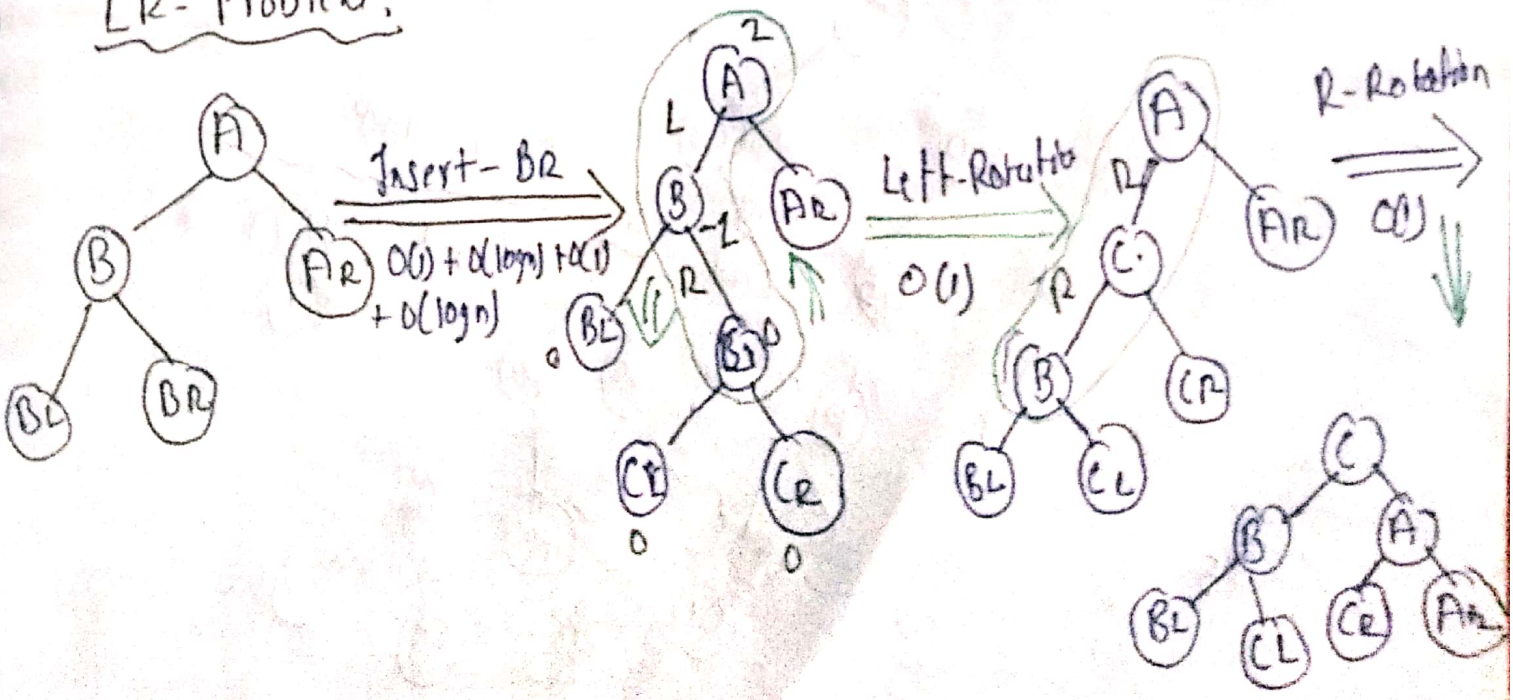
LL-Problem



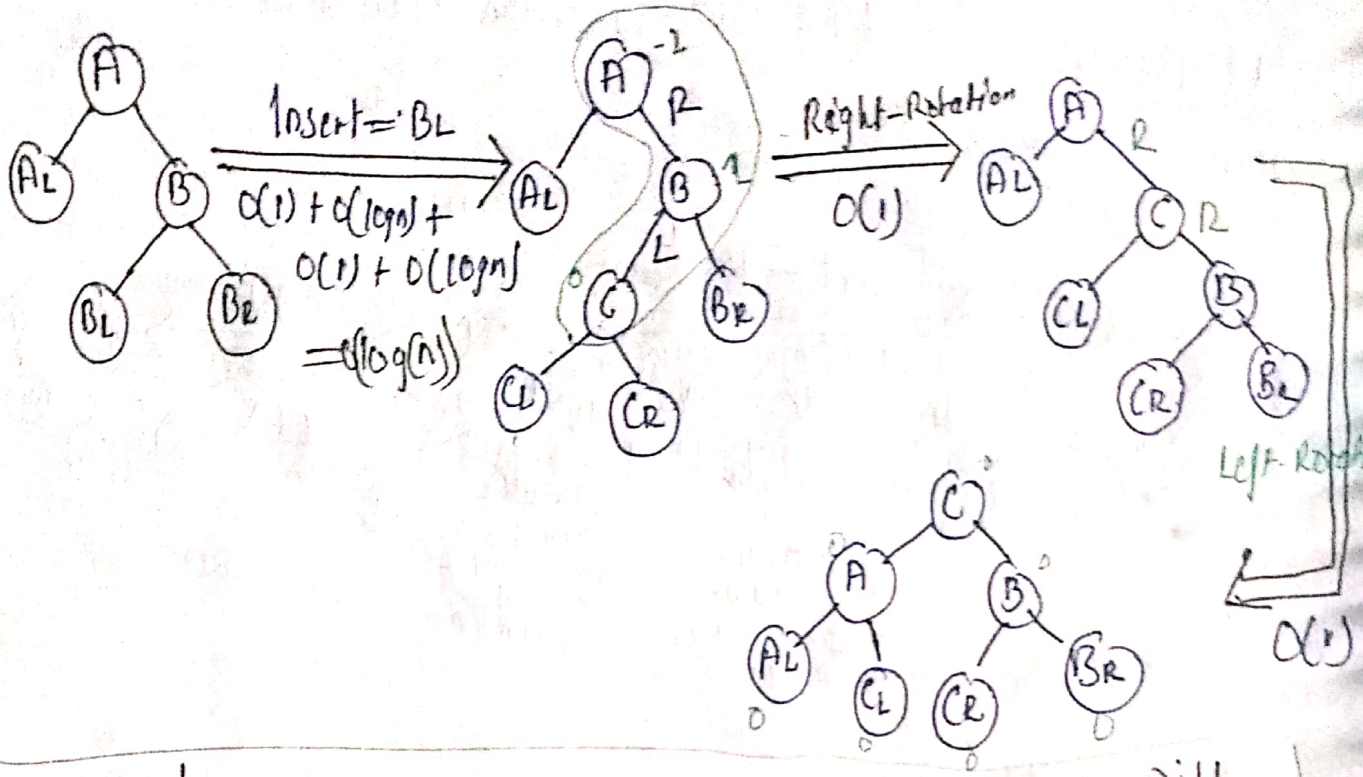
RR-Problem



LR-Problem:



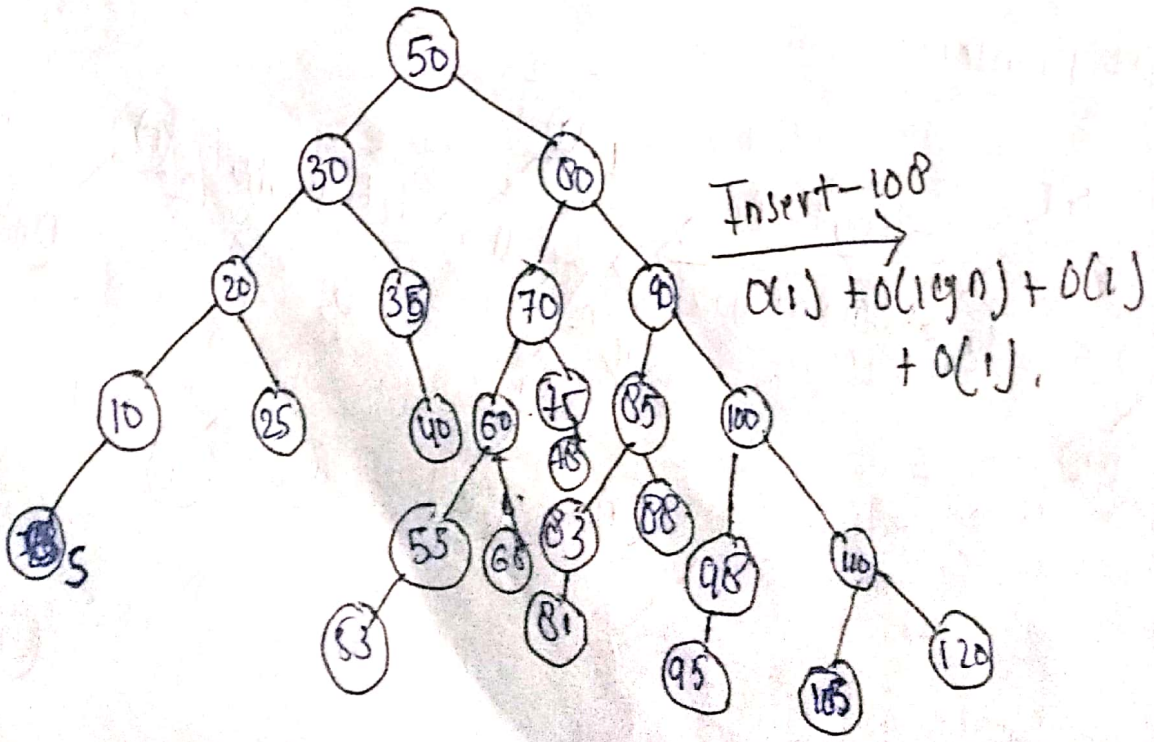
RL Problem:

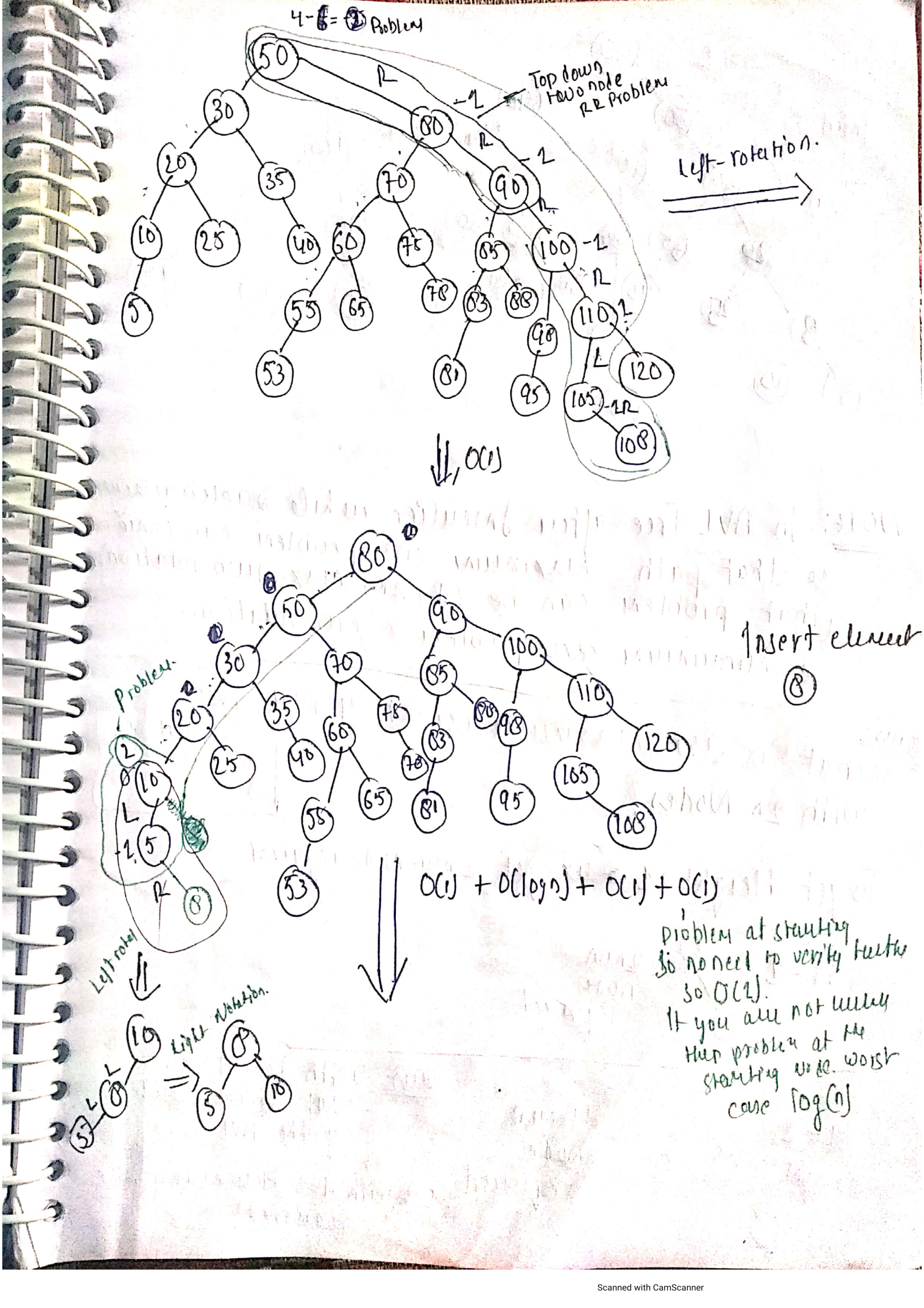


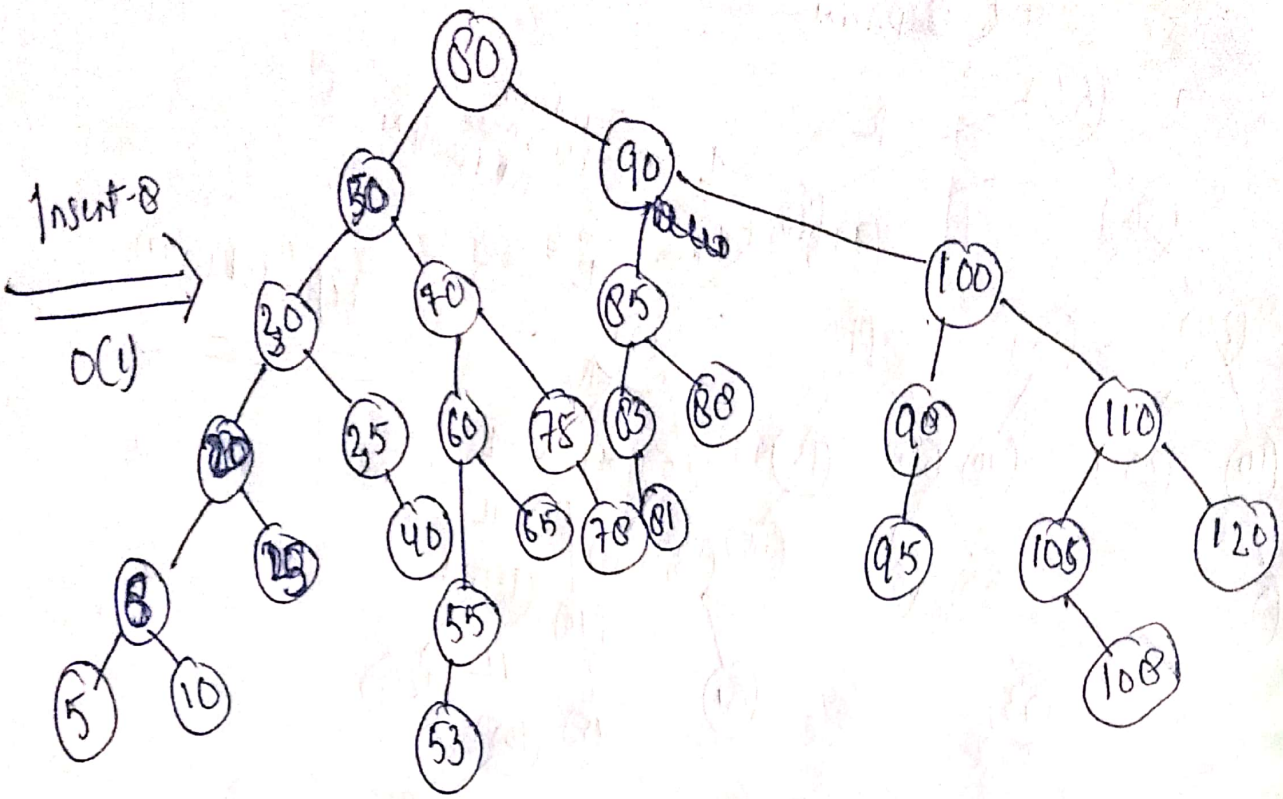
NOTE: Inserting a element into AVL Tree. will allways take $O(\log n)$ time.. (why not O(1) because insertion will be done at null place & null will be there at the last level).

- In BST we dont do verify while Inserting & Deleting the element.

Ex 1)

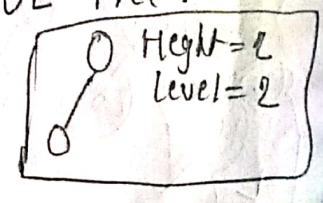






NOTE: In AVL Tree after Insertion while backtracking is that path maximum. one problem can come. (that problem can be LR, RR, max. two rotation) & minimum zero problem. & zero rotation.

Ques: What is the maximum height of AVL Tree with 20 Nodes?

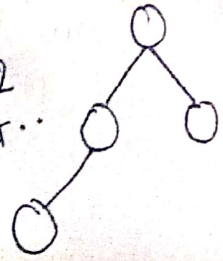


To get Height of 1 ~~we get~~ 2 Min Node required

H=2

2 Min node required

H=2 Height..



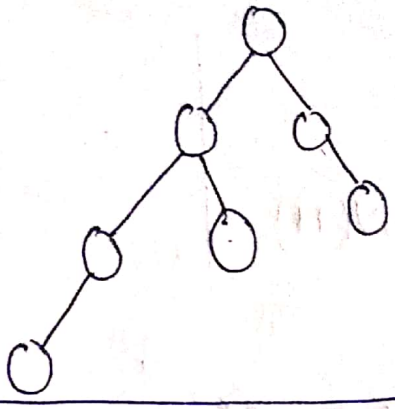
4 Min node required.

Ques: With the 4 Node what is the height of the AVL Tree.

or

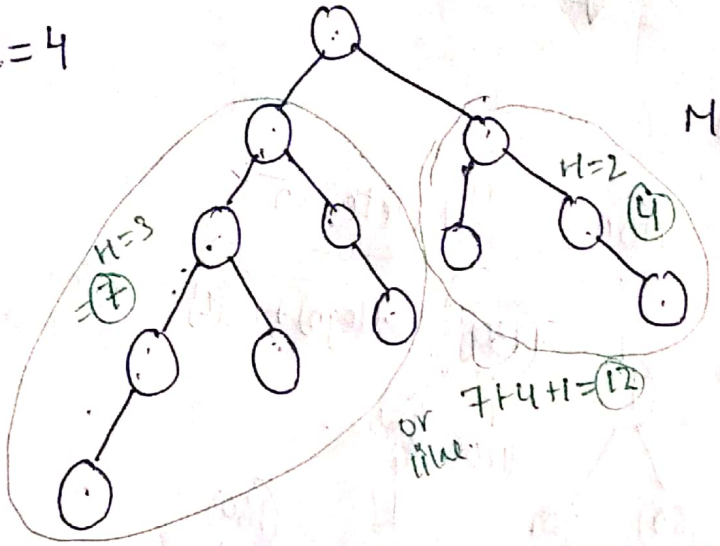
Ques: with H=2 How many min node required!

H=3



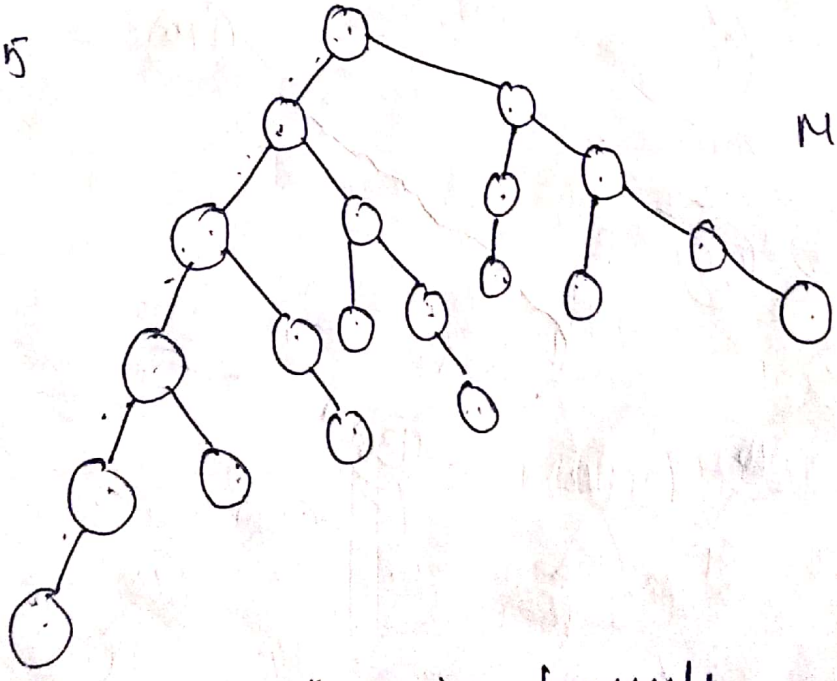
Min- 7 Nodes required

H=4



Min – 12 Nodes required

H=5



Min - 20 Nodes required

Or we can do by using formula.

MNN = Min No of Node in the Height of AVL Tree.

$$MNN(h) = MNN(5) + MNN(4) + 2$$

$$20 + 12$$

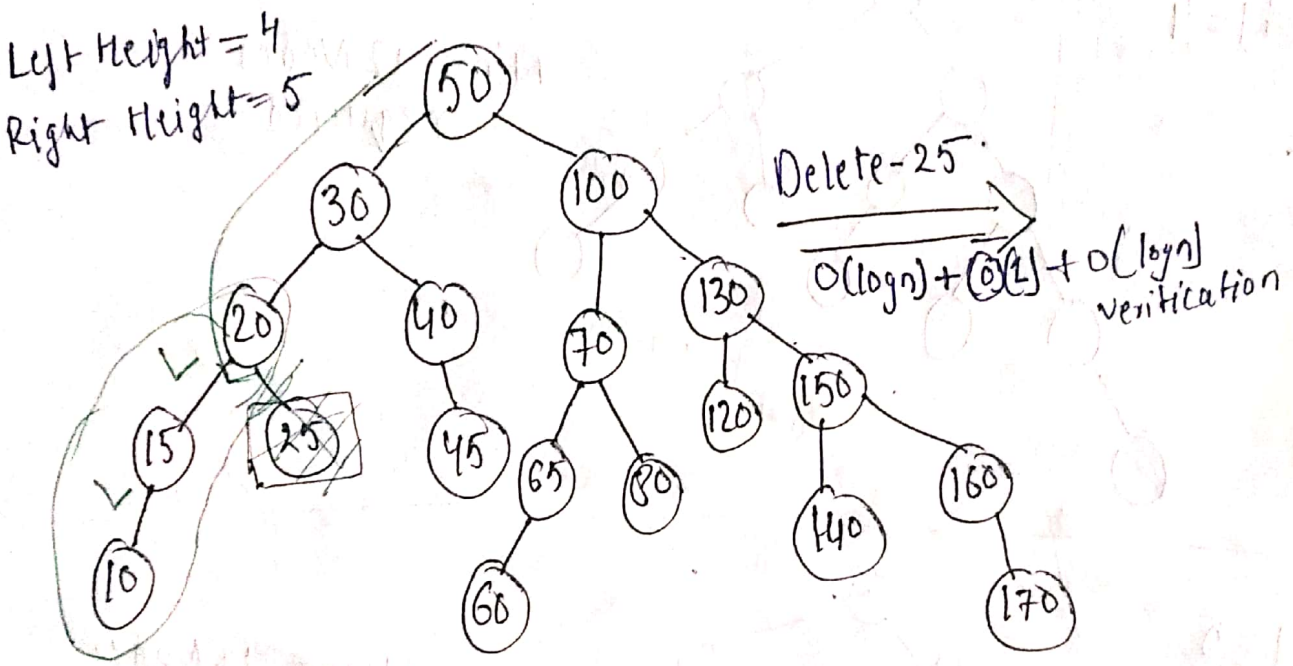
$$= 33$$

Imp formula:

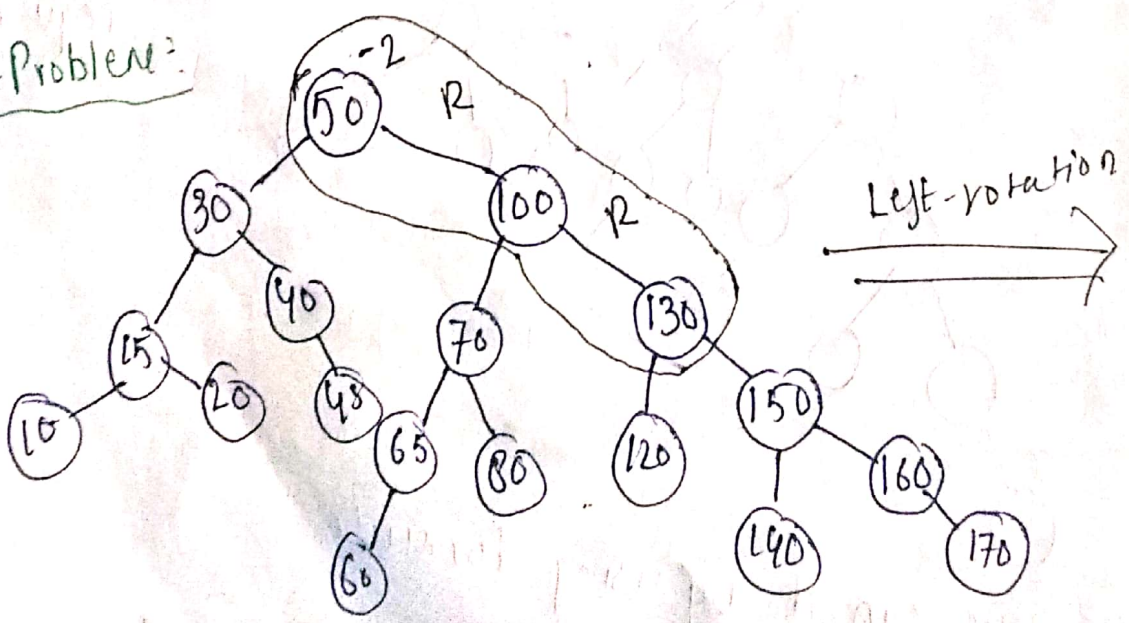
	Nodes
if H=1	= 2
if H=2	= 4

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

Consider the following AVL-Tree:



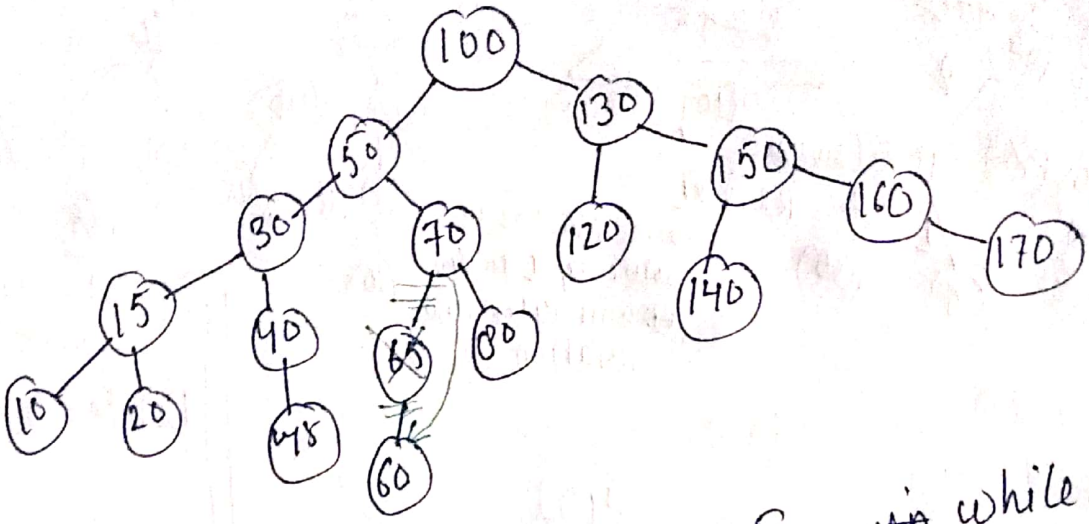
LL-Problem:



If you delete element height will be decrease or same but never increase.

Now LHeight = 3
RHeight = 5 = 5 - 3 = 2 problem.

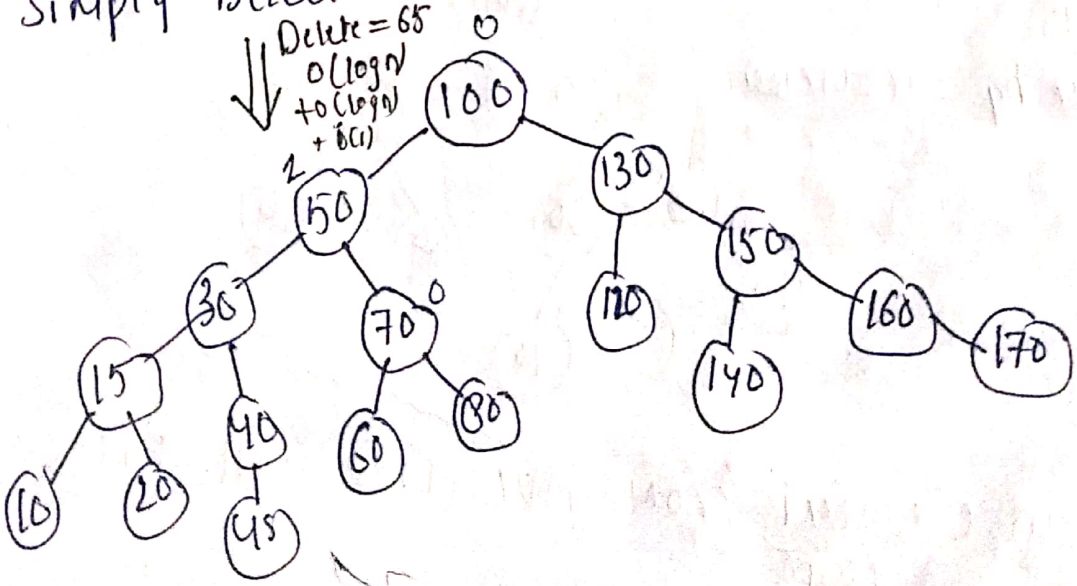
After deleting a node you need to verify balance problem is there or not. while coming

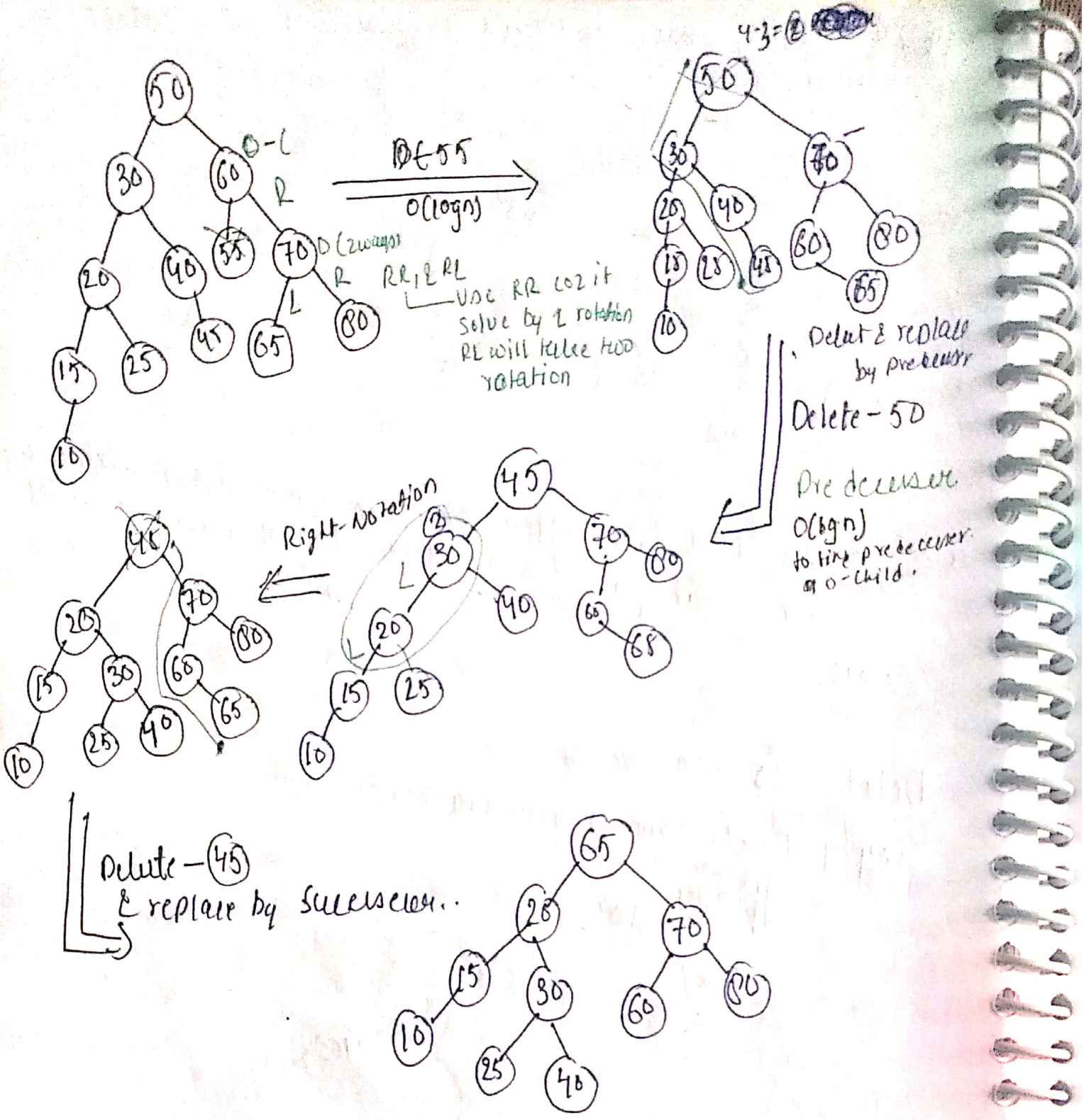


NOTE: In AVL Tree after deletion while backtracking in that path maximum logn problems will come.

Delete - 65 in above Tree:

Simply Delete Connect grandfather to son.





Deleting an element from AVL Tree will take always $\log(n)$ time

ans- In which one of the following data structure following two operation can be done will take Order of $O(\log n)$ time

① Insertion ② Deletion.

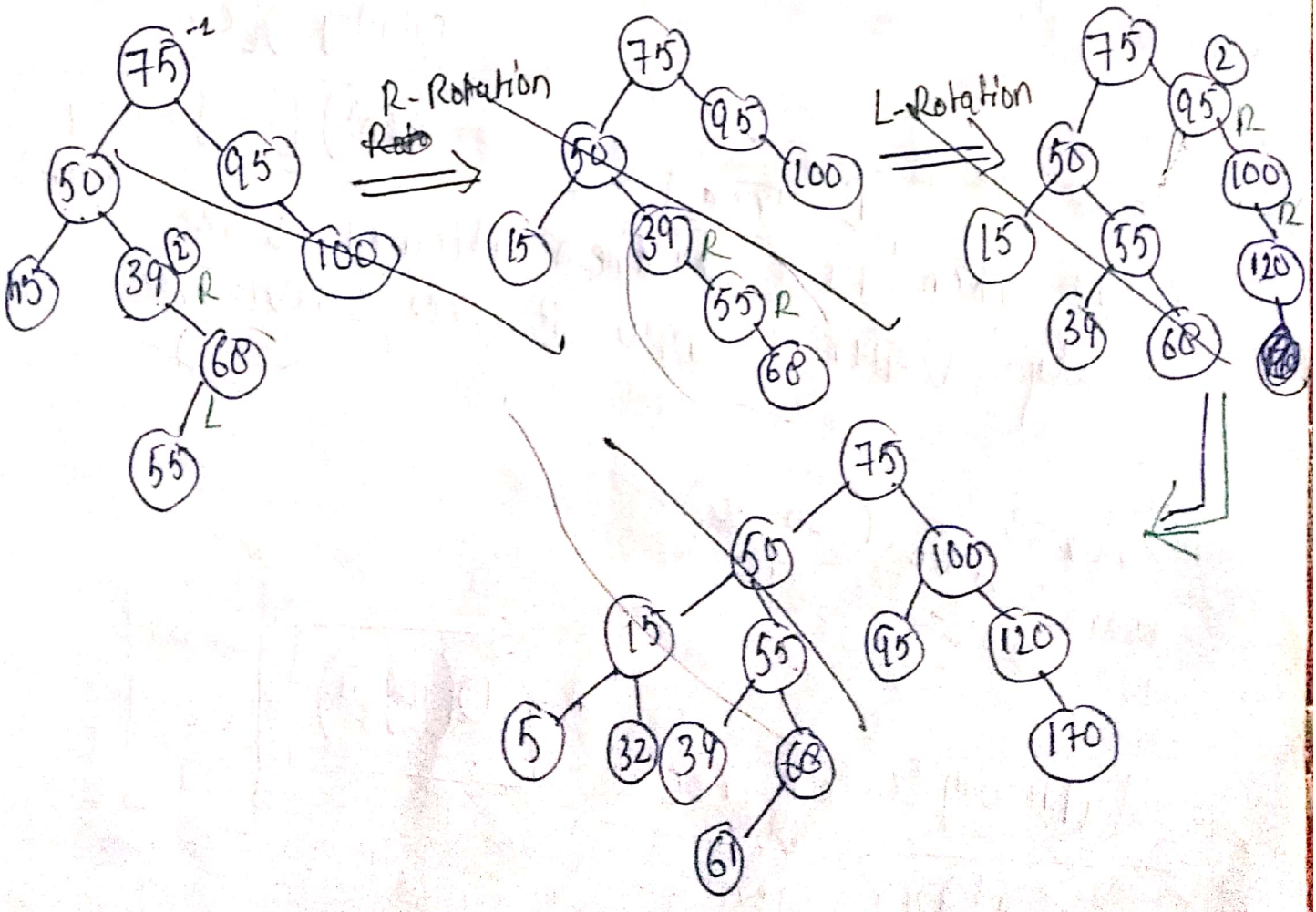
a) AVL
b) Min-Heap. → Deletion → Max - log(n) Min - O(1)
c) Both
d) None of these

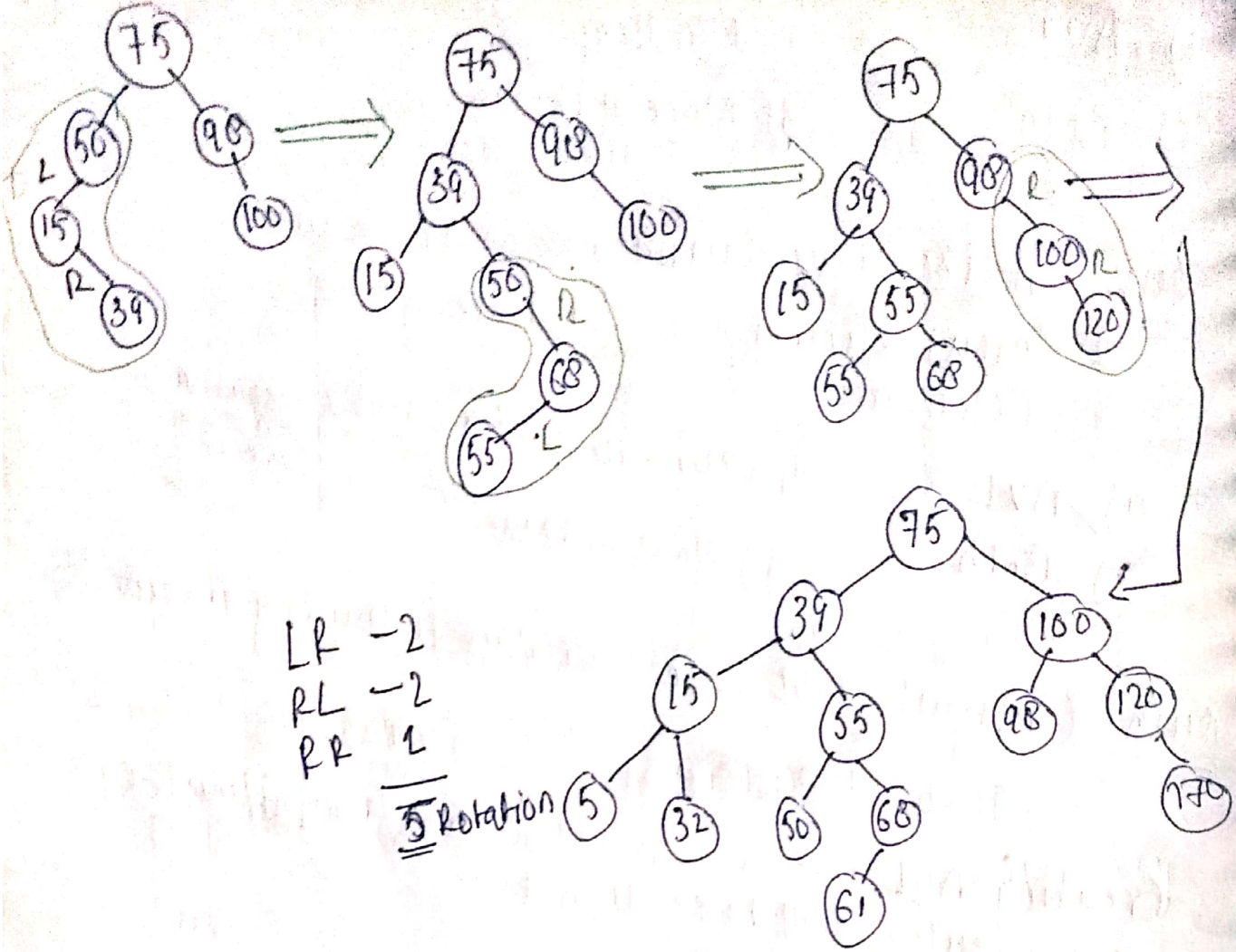
Ques: If the above Question condition is
① Insert element x is not there.
② Deletion

a) AVL
b) Min-Heap - for insert element if which is not there it will take O(n) time to scan whole tree.
c) Both
d) None of these

Ques: Construct AVL Tree. for the following elements.
75, 98, 50, 100, 15, 39, 68, 55, 120, 170, 5, 32, 61

Creating AVL Tree for n-elements, (n log n) time [EC]
(O2 insertion will take log n time.



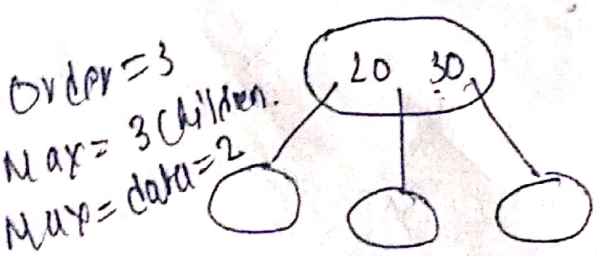


$O(n \log n)$ [EC] = AVL.

$O(n^2)$ [WC] = BST.

B-Tree

More then two children allowed is the only Difference b/w B-Tree & AVL Tree.



Multiway Search Tree.

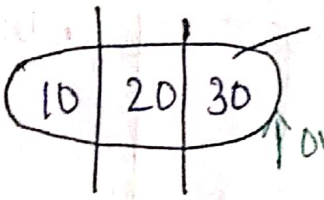
Balanced Tree.

$O(\log_3 n)$

[I.S
D.S.
S T]
Searching

Here No Rotation.

Max = data = 2

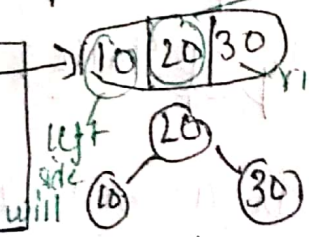


Max data = 2
but you are inserting 3-data. So overflow

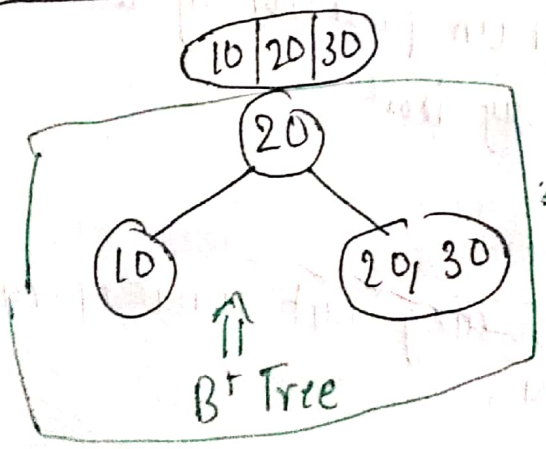
Then Node-splitting.

In B tree - Inserting time. overflow problem arise
Deleting data underflow problem there

Soln:
If overflow ÷ Do Node-Splitting
If underflow ÷ Do Node Combining



B-Tree.



Whenever you are Splitting like this..

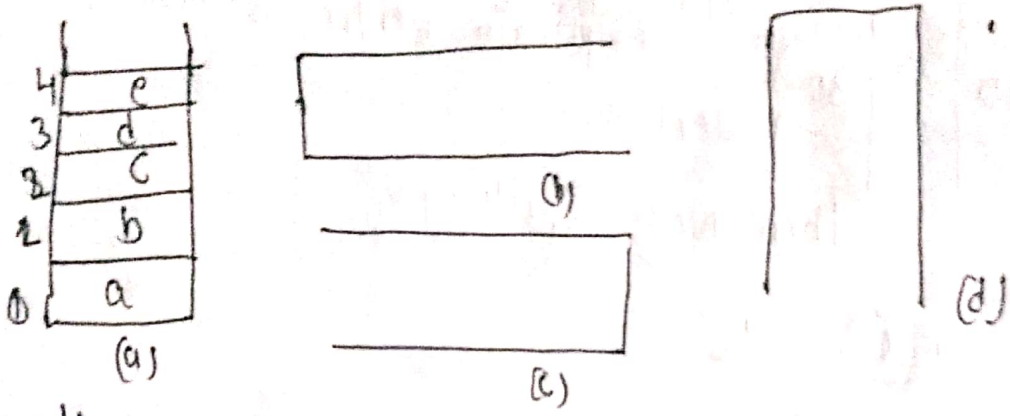
19/Sept/2019

Stack

It is a linear data structure. in which elements are inserted & deleted at one end.

Def: One side open, another side closed.

Representation
• Diff in Different Books



Top = 4

└ Top is a variable which contain position of Top most element.

Property of Stack: LIFO (or) FILO

* Top is a variable which contain position of newly inserted Element - In Queue we say rear.
* Top is a variable which contain position of an element deleted - In Queue we say front.

ADT of Stack.

ADT: Abstract Data Type. It is nothing but what operation you can perform with them..

ADT of stack

⇓

Push()

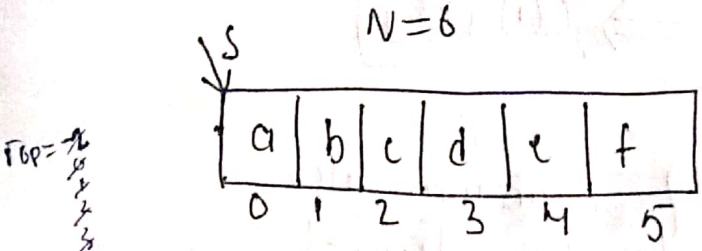
Pop()

Implementing Stacks:

Nothing but Push & Pop operation writing with which data structure you can using like, Queue, array, linked list etc.

```
Push()   Pop()
{        {
}        }
```

Implementing Stacks using array:



Push: Opⁿ

```
Push(int x)
{
  if(Top+1 == N)
  {
    Pf("stack overflow")
  }
  else
  {
    Top++
    S[Top]=x
  }
}
```

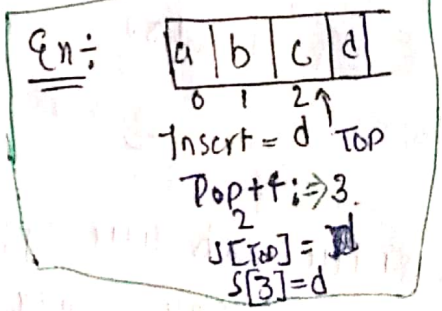
[First inc. then store. if we store without inc. then ~~def~~ overwriting]

= also write

S(Top++)=x ✗

S[++Top]=x ✓

Eg:



⇒ O(1) [EC]

Pop Opⁿ:

~~POP(int)~~ POP()

```
{
  int y
  if(Top == -1)
  {
    Pf("Stack is underflow")
      exit(1);
  }                          ⟹ O(1) [EC]
  else
  {
    y = S[Top]   } or can write also
    Top--        }   y = S[Top--]
    return(y)
  }
}
```

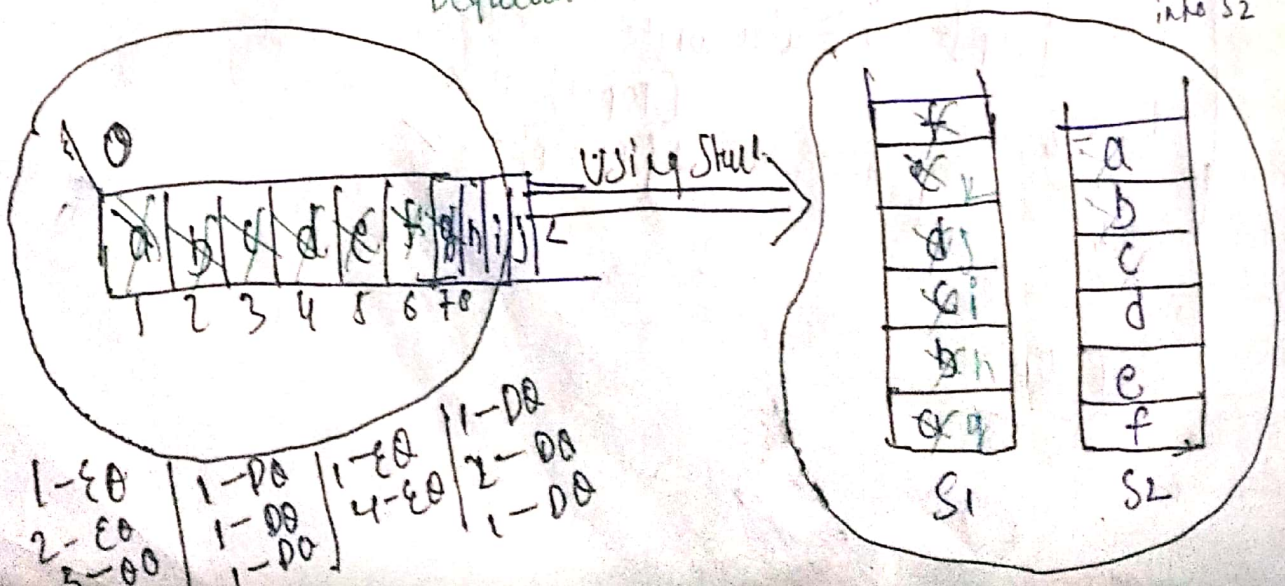
- POP will return but Push not.
- Push take varible element but POP didn't take.
- Push first Incr. & then Store
- POP first Store & the Decrement.

Stack & Queue both are opposite. so there will be chance Implementing stack using Queue or Queue using Stack in ~~of~~ Question in Gate.

Implementing Queue Using Stack;

- Enqueue()
  Dequeue()
- Push()
  POP()

whenever Dequeue happen it will reverse all the element in 2nd stack. Still all elements S1 into S2



1-E0 2-E0 3-00 | 1-P0 1-00 1-D0 | 1-E0 4-E0 | 1-D0 2-00 1-D0

1- EQ = 1-push-S_1

2- EQ = 2-push-S_1

3- EQ = 3-push-S_1

1- DQ = 6-pop-S_1

= 6-push-S_2

= 1-pop-S_2

1- DQ = 1-pop-S_2

1- DQ = 1-popS2

---

1 - EQ = 1-push-S_1

---

4 - EQ = 4-push-S_1 O(n) [WC]

1-DQ = 1-pop-S_2

2-DQ = 2-popS2

1-DQ = 5popS1

push.

= 5 popS2

= 1-pop-S_2

```
void EQ(x)                    Enqueue
{                             ⇒ O(1)
push(S1,x)
}
```

```
int DQ()                      Dequeue
{                             O(2) [BC WC]
if(S2 is not empty)           ⇒ O(n) [WC]
return(Pop(S2))
else
{
while(S1 is not empty)
{
x=pop(S1)
push(S2,x).
}
return(pop(S2))
}
}
```

If the class is like this: Make Dequeue cheaper.

Queue ⇒ Enqueue/Dequeue()

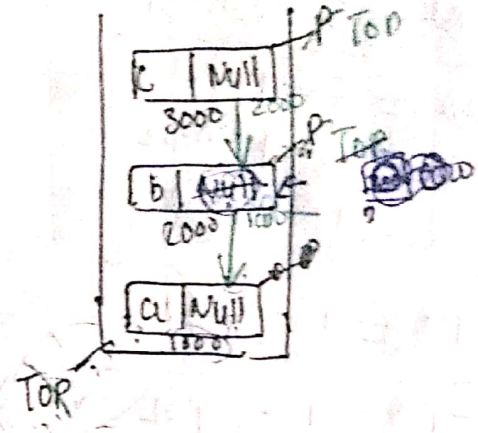
Stack = Push() / Pop() / Reverse() — It will reverse within stack

EQ ⇓ Reverse, push, Reverse

DQ ⇓ Pop()

Implementing Stack Using Linked List:

Push(x)

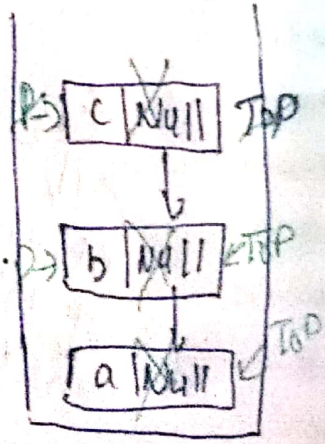



```
Push(x)
{
P = Malloc()
if(P==Null){
  printf("Stack Overflow");
      exit(1);
}
  else
  {
P->Next = Top
    Top = P;
  }
}
```

Pop()

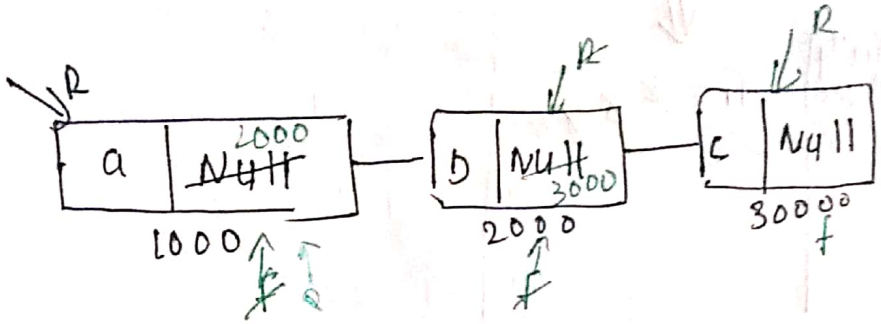
```
Pop()
if (Top==Null){
  Pf("String Under
       Flow")
    }
  else
  {
y = Top->data.
  P = Top
Top -> Top->next;
```

O(1)[EC]



```
free(P)
  P==Null
  return(y)
}
```

Queue Using Linked list:



EnQueue:

```
EQ()
{                          ⟹ O(1)
P=Malloc()
if (P==Null){
 Pt("Queue Overflow");
  }
else.
R→next=P
  R=P
}.
```

DeQueue:

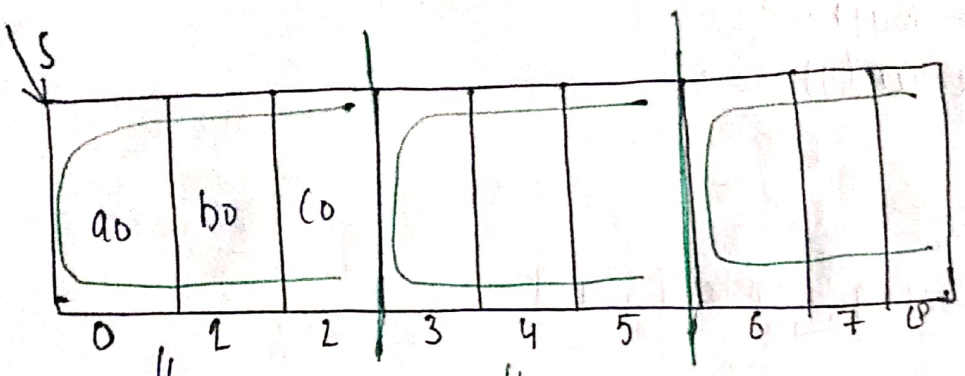
```
DQ()                       ⟹ O(1)
{
if(F==Null)
  print(Queue Empty);
 }
else
P=F
F=f→next
 y=P→data.
   free(P)
     P==Null
       return(y)
   }
}.
```

HW: Implement Stack using Queue:

Implementing Multiple Stack in single array:

Advantage: You can execute multiple programs in a single array like, GCD, Fibonacci, LCS, etc. (recursive program)

N = 9 (Size of array)
M = 3 (No of stacks)
N/M = 9/3 = 3 (Size of 1 stack)



(Stack 0 Top) $T_0 = -1$ (Initially outside) $\quad T_1 = 2 \times 3 - 1 = 5$

```
Push()
{
Top++
S[To] = x
}
```

Push condition for overflow

When can I say Stack No. 55 is full.
Stack No 55 Current Top == Stack No. 56. Initially Top.

2nd stack - Initially Top of Stack.

$2 \times 3 - 1 = 5$

(0 1 2) Before Hoard any stack — Size of stack

0th stack - ITOS

$0 \times 3 - 1 = -1$

25th stack - ITOS.

$25 \times 3 - 1 \Rightarrow 74$

ith stack - ITOS

$i \times \frac{N}{M} - 1$.

Could be POP for underflow

When can I say stack No i is empty.

Stack No i of current TOS T_i == Initially top of the stack Its

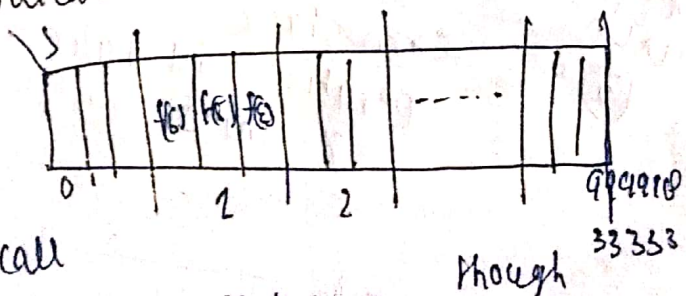
Push:

```
Void Push(x,i)
{
if (Ti==(i+1)N/M -1)
  {
  Pf("stack is overflow");
     exit(1)
  }
  else.
  {
  Ti++
  S[Ti]=x;
  }
}
```

POP:

```
POP(i)
{
if (Ti== i*N/M -2)
{
Pf("stack is underflow"
  exit(0);
}
else
{y=S[Ti]
 Ti--;
 return(y);
}
}.
```

- Using above program we can implement multiple stack in single array but it is not efficient because ~~en~~ if one stack is full all other remaining are empty ~~then~~ we ~~will~~ are getting error msg "stack overflow" ~~Enet~~ ~~Enter~~ Even though space available.

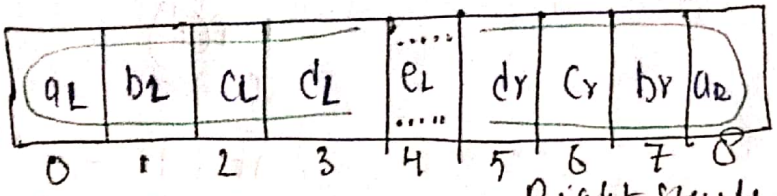


N=99999
M=33333

f(0) call Though space available even we are not able to place function.

Implementing Multiple Stacks in Single Array with Efficiency

n = 9 (Array size)



Left Stack

$T_L = -1$ (initially outside)

Push

{

T_L++

$S[T_L] = x$

Right Stack

$T_R = 9$ (initially outside)

T_R--

$S[T_R] = x$

If this is a Question. What is the Overflow condition

a) $T_L == N-1$

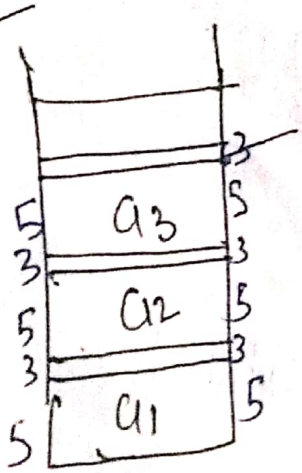
b) $T_R == 0$

c) $T_L == T_R - 1$

d) $T_R == T_L - 1$

Average Lifetime of an element in Stacks:

Ex①: $n = 3 (a_1, a_2, a_3) \Rightarrow$ 3 push cons follow 3 pop con



elapsed time. y-units ③

each stack operation will take n-units. ⑤

a_3 — 3 — 3

a_2 — 3, 5, 3, 5, 3 — 1, 9

a_1 — 3, 5, 3, 5, 3, 5, 3 — 35

Total = 57

Average $= \frac{57}{3} = 19$ //

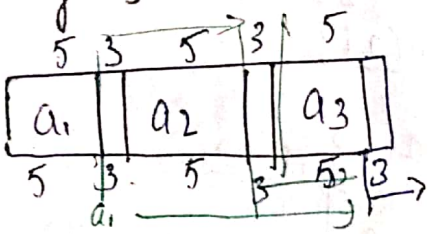
20-/Sept

Queue:

$n = 3 (a_1, a_2, a_3)$

$x = 5$

$y = 3$



$a_1 — 19$

$a_2 — 19$

$a_3 — 19$

$3 \times 19/3 = \boxed{19}$

Application of Stack:

1. Recursion
   (i) Tail Recursion
   (ii) Non-Tail Recursion
   (iii) Nested Recursion.
   (iv) Indirect Recursion.
2. · Infix to postfix
   prefix to postfix.
   postfix Evaluation
3. Towers of Hanoi
4. Fibinocci Series:

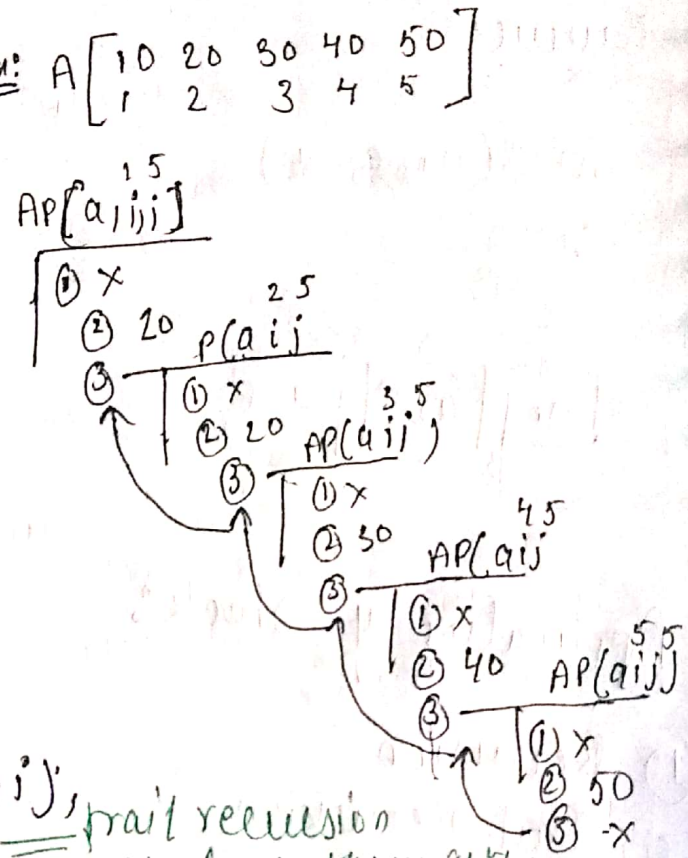
Recursion

recursive

Write a C Program to print the given array of n-elements.

Ex: A[10 20 30 40 50]
1 2 3 4 5

```
Arrayprinting(a,i,j)
{
  if(i==j)
  {
    Pf(a[i]);
    return;
  }
  else
  {
    Pf(a[i])
    Arrayprinting(a, i+1, j);
  }
}
```



Tail recursion
No work there after function call...

Disadvantage: Taking more space

Tail-Recursion?

You are calling recursion func at last after that no work there so misuse of stack space.

(1) In the given recursive program after the function call no work there. Then it is known as Tail recursive program.

Example: Above Program.

Tail recursion ~~taking~~ wasting lot of stack space & time

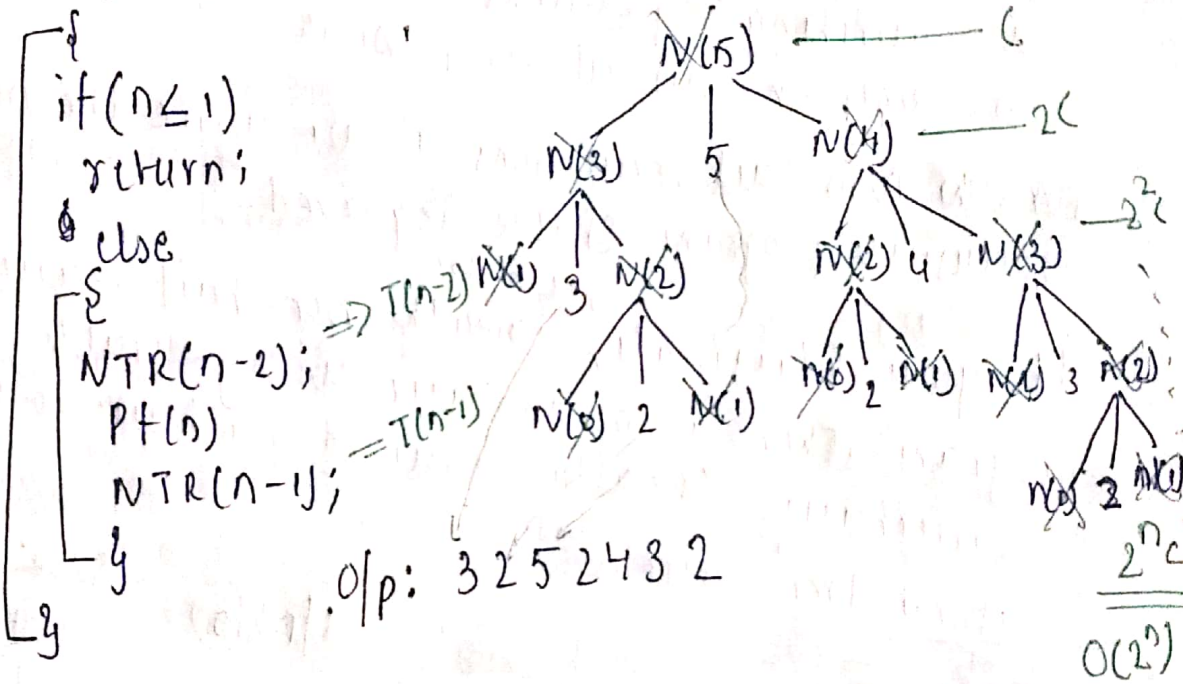
③ The advantage with the tail-recursion is we can write non-recursive program easily with the help of while loop. Take all inside the while loop.

Non-Tail Recursive Program:

i/p = n = 5

Ex:

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



O/p: 3 2 5 2 4 3 2

Time Complexity:

All function calls ~~Do~~ without DP:

Time Complexity: $2^n = O(2^n)$

Space: i/p + extra
2B + nB
O(n)

Distinct function call with DP:

Time Complexity: n = O(n)

Space: i/p + extra.
2B + nB + nB
stack table
2n = O(n)

NOTE:

① In the given recursive program after the function call some work is there then it is known as non-tail recursive program.. above problem is a example for that.

② It is not wasting the stack space.

③ The Difficulty with the Non-Tail recursive program is writing equivalent non-recursive program. is complex but it is possible.

For the Non-tail recursions. if we write the non-recursive program stack required.

④ If you Eliminate recursion form the Non-tail recursive program then also we are using stack data structure.. (Really required you have to use it if not then same it).

```
A(5)
{
Push(5)
Push(3)
Push(1)
Pop()
Push(2)
 :
Pop(3)
Pop(4)
Pop(5)
}
```

Equivalent Non-Recursive Program for above problem.

Here user will be written the push & pop operation.

Here also stack required

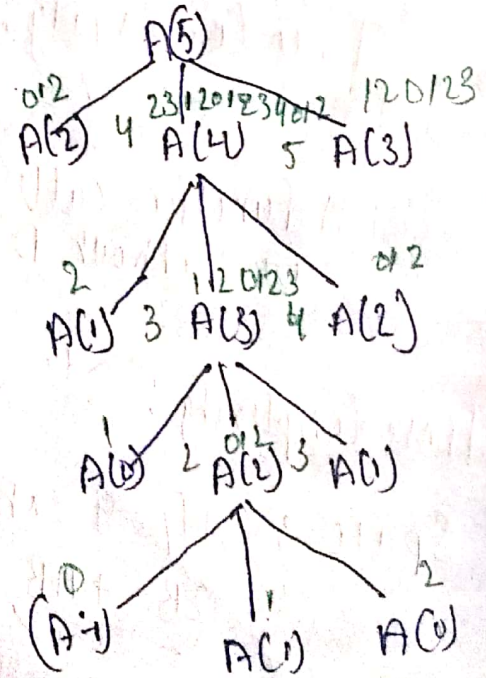
If required it is compulsory re-use it not no required.

Ex①:

```
A(n)
{
if(n<=1)
return;
else
{
Pf(n-2)
A(n-3)
Pf(n-1)
A(n-1)
Pf(n)
A(n-2)
}
}
```

i/p: ⑤

Solve: A(n-1)
⇓
It cover all the possibility



O/p: 3012423120123401251201 23

Without DP

Space Comp.: i/p + extra
2B + nB
O(n)

Time Compl: $O(3^n)$

With DP

Space Complexity: 2B + nB + nB (i/p, stack, extra stack, Table)
O(n)

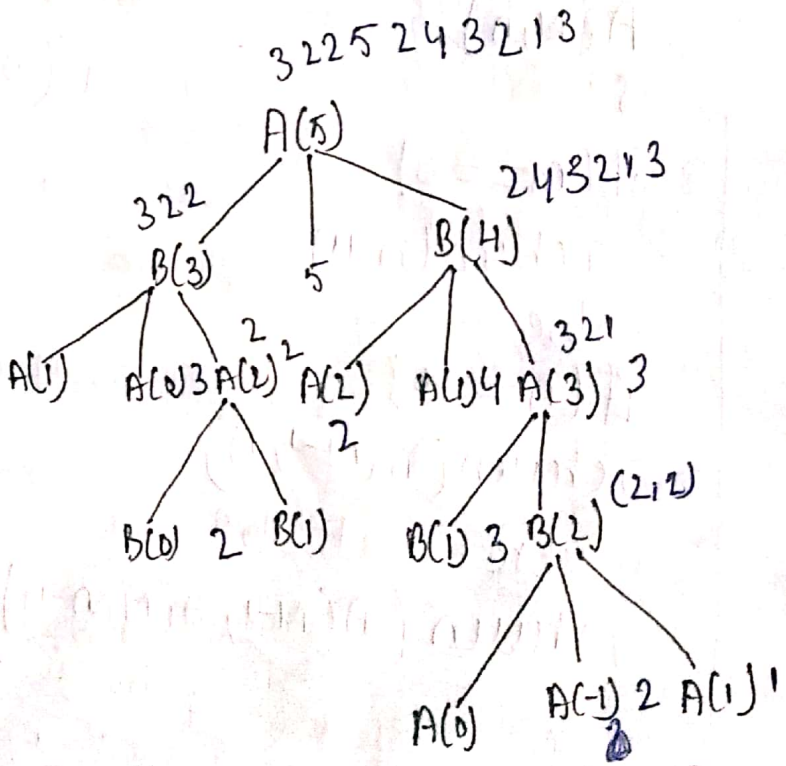
Time Complexity: O(n) & Distinct function calls

INDIRECT RECURSION:

Recursion ⇒ A() calling A()
Indirect Recur ⇒ A() calling B() & B() calling A()

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

```
B(n)
{
if(n<=1)
return;
else
{
A(n-2)
A(n-3)
Pf(n)
A(n-1)
Pf(n)
}
}
```



i/p: A(5)

O/p: 3225243213.

With DP

Dilema

Time: A also n-people
B also n-peoples
Total Distinction

Space = $2n \Rightarrow O(n)$

$4B + nB + 2nB$

$3n \Rightarrow O(n)$.

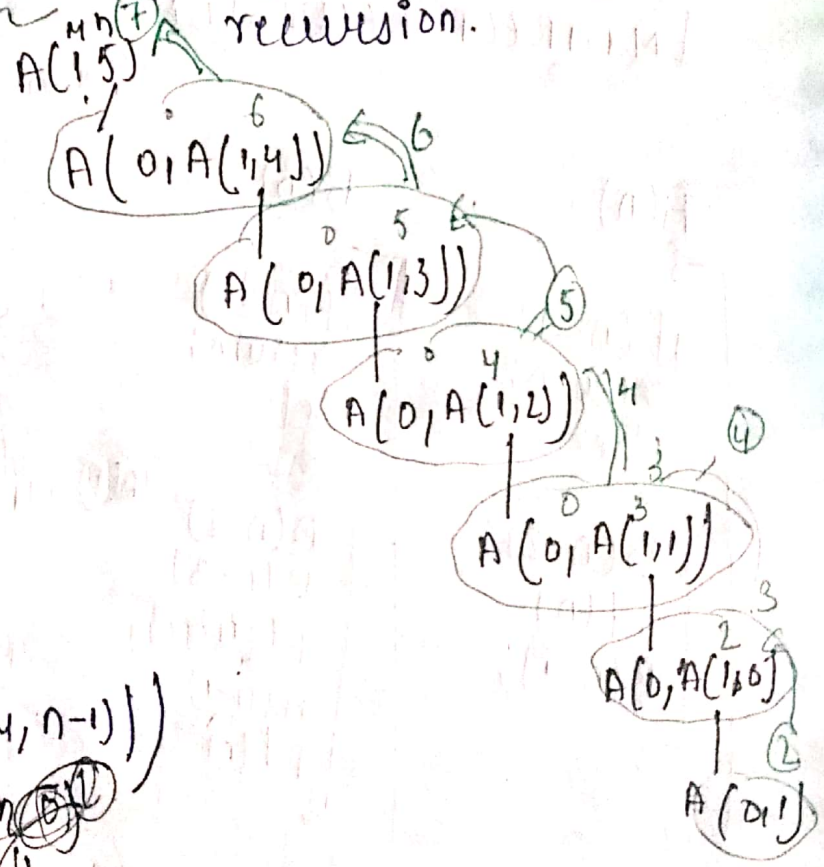
Without DP

Time: 1st ~~function~~ Program 2 time call
2n & 2nd Prog 3 time call
Take upper bound
$= O(3^n)$

Space: i/p + extra.
$4B + nB$
$O(n)$

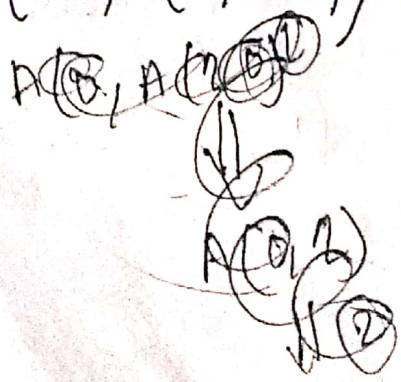
NESTED-RECURSION: Inside recursion again recursion.

```
A(M,n)
{
if(m==0)
 return(n+1)
 else
if(n==0)
return(A(M-1,1))
 else      A(M,n-1)
return(A(M-1,A(M,n-1)))
}
```



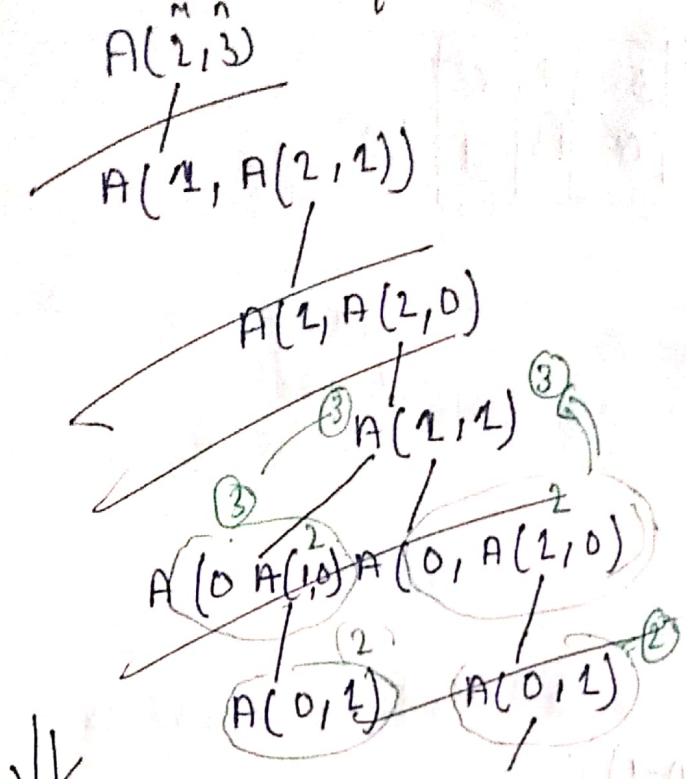
i/p: A(1,5)

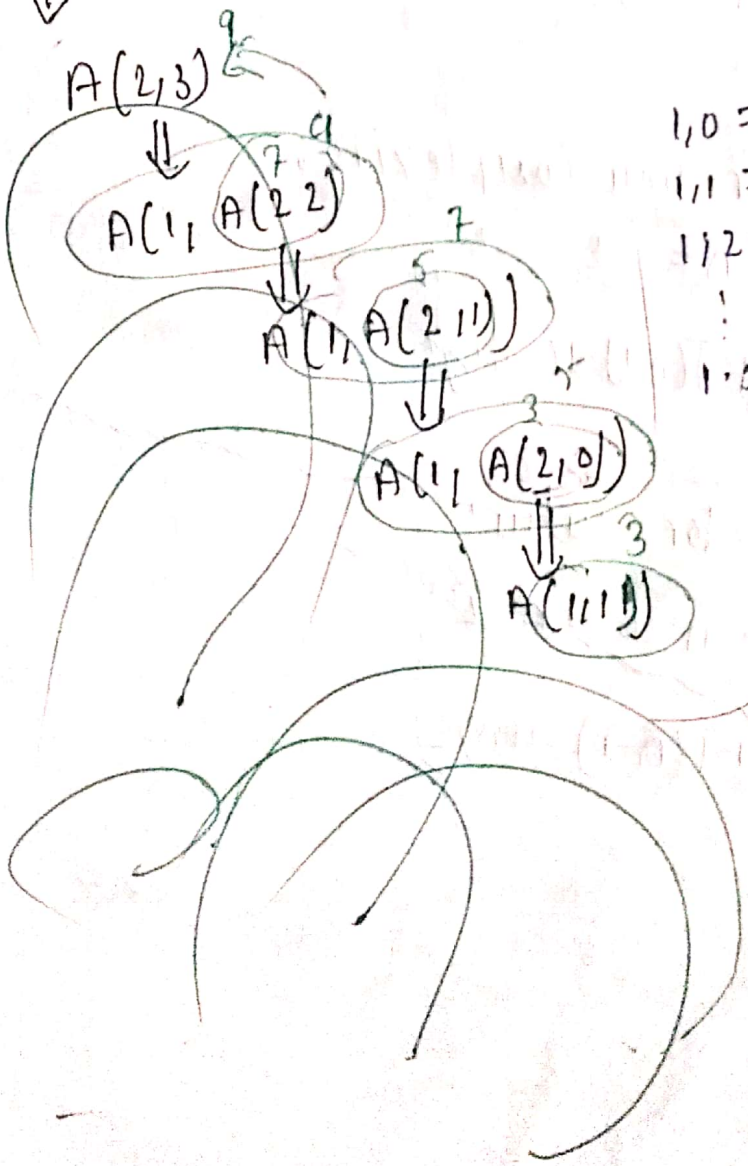
o/p: 7



I/P: A(2,3)

O/P: 9





Observe, for simplicity

1,0 = 2	2,0 = 3
1,1 = 3	2,1 = 5
1,2 = 4	2,2 = 7
⋮	2,3 = 9
1,5 = 7	

FIBONOCCI SERIES:

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

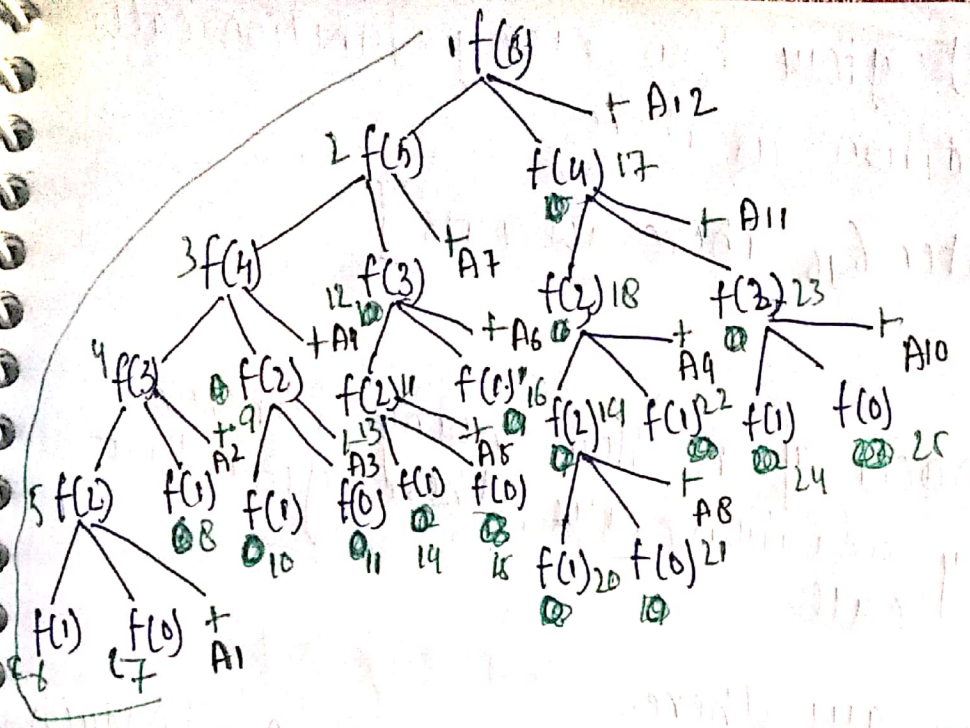
```
f(n)
{
if(n==0 || n==1)
  return(n);
  else
  {
  return( f(n-2)+f(n-1));
   }.
```

Recurrence Relation for Time Complexity:

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

Recurrence Relation for value:

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



Ans(1): In fibinocci of n. After How many function calls first Addition taken place:

Ans: After 7 function call, 2nd Addition taken. Go left most path.

⇒ [n+1] After C7 ther is a A2

↳ Condition: if the program is like this

```
else
{
a = f(n-1) ← Left sid will be n not n/2.
b = f(n-2
c = a+b)
return(c)
}
```

If it is given like this. ⇒

```
else
{
a = f(n-2) ⇒ left side will be n/2
b = f(n-1)
!        = [n/2 + 2]
}
```

Ques2: In fib(6) after how many function calls. 6^{th} addition is there:

Ans: 16 [After 6_{16} there is a A_6]

Ques3: In fibonacci of n ~~How many function~~ How many addition are there after. Last function call.

Ans: 3 Addition are there.

Last function call is f(10) [A_{10}; A_{11}, A_{12}] Additions are there

After last function call while backtracking in every level one addition required.

Ans is $n/2$ [cos left path n if is a fibinocci series) right path $n/2$ high side will be $n/2$]

$n/2 = 6/2 = \boxed{3}$

Ques4: In fibonocci of n How many Addition there:

No. of Additions

$f(0) = 0$
$f(1) = 0$
$f(2) = 1$
$f(3) = 2$
$f(4) = 4$
$f(5) = 7$

$f(100) = f(99) + f(98) + 2$

$f(n) = f(n-1) + f(n-2) + 2$ ⇐ Rec. Rel^n for Addition.

ans: In fibinocci(n) how many function calls are. there.

No of function calls

$f(0) = 1$
$f(1) = 1$
$f(2) = 3$
$f(3) = 5$
$f(4) = 9$
⋮

$f(10) = f(9) + f(8) + 2$

$f(n) = f(n-1) + f(n-2) + 2.$ ∈ Re. Rl^n for function calls

INFIX TO POSTFIX:

Ex:

Infix: a + b * c

Postfix: a b c * +

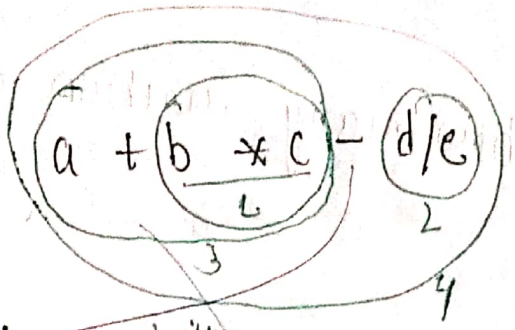
Prefix: + a * b c

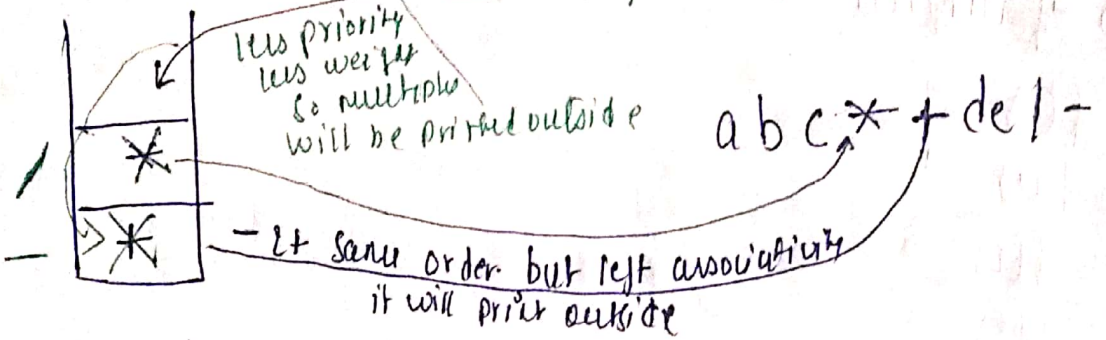
User i/p = Infix
Processor i/p = Postfix

NOTE: In all the above three notations operands order cannot change but operators order may change.

Ex2: Infix: a + b * c - d / e

If two operators having same priority then associativity will come. left to right. (* → same, + → same)





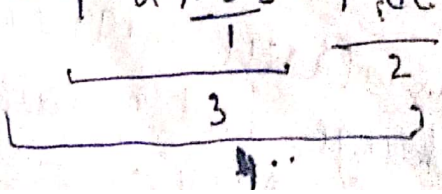
- Infix postfix conversion uses operator stack

Operand ⇒ Print.

Operator

① TOP < NEXT ⇒ PUSH
② TOP > NEXT ⇒ POP
③ TOP = NEXT ⇒ POP (left to right)
④ TOP = NEXT ⇒ PUSH (right to left.)

Prefix: - + a * b c / d e

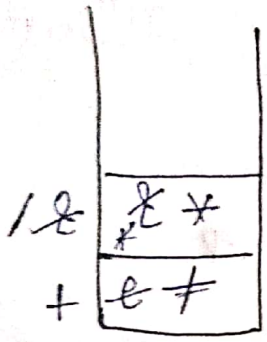


Ex 5:

d/e − a&c * b + g & f/h

3 2 4 6 1 5 7 8

−de/ac&b*−gf&h/+



2- stack space.

Infix - Postfix conversion will take O(n) time

Prefix: +− /de *&acb /&gfh

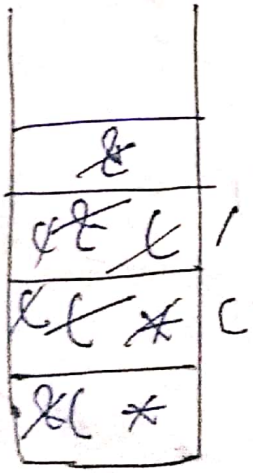
3 2 4 1 5 6 7

Ex (2):

Infix: ((a & b) * (c & d)) & (e − f) * (g / h)

Closed bracket comes i.e pop until open & print

Post & Prefix having No brackets



Post fix:

ab&cd&*ef−&gh/*

Prefix: - have No Algorithm

&&ab &cd −ef /gh

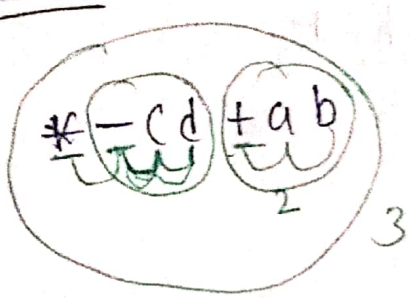
1 2 4 5

3 6 7

Prefix to Postfix:

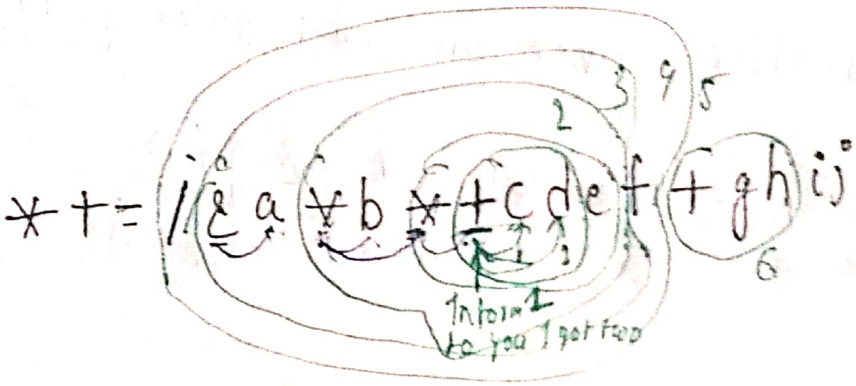
When you will round who 3 people there 1 operator 2 operend.

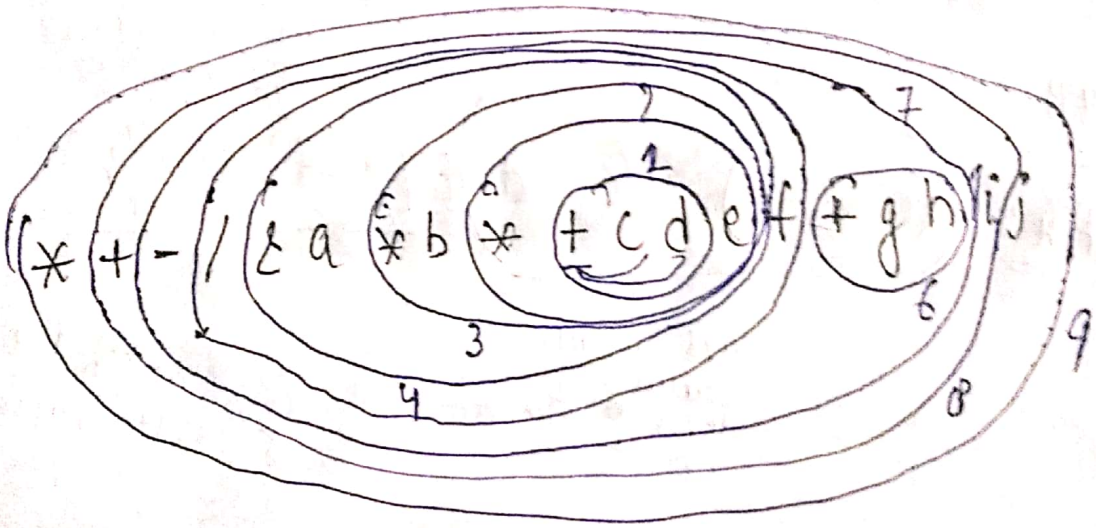
Ex ① : Prefix ——— Postfix



——— c d - a b + *
1 2
3

Prefix:

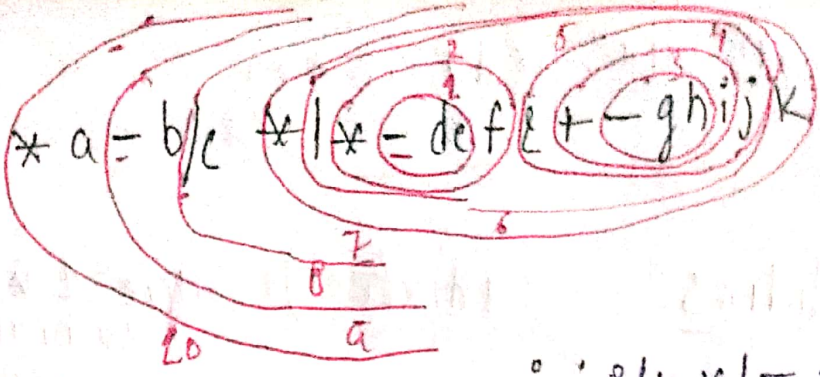




a b c d + e * * & f / g h + - i + j *
1
2
3
4
5
6
7
8

Ex 3:

Prefix: * a - b / c * / * - d e f ^ + - g h i j k



- de-fx
  1
  2

- gh^-i+j^/k*/-*
  3
  4
  5
  6
  7

Postfix Evaluation:

Infix: 2*3-5*3/5*3-7*2+6*1-3

Postfix: Infix to Postfix which will take O(n) time.

⇒ 23*53*5/3*-72*-61*+3-

23*53*5/3*-72*-61*+3-

⇓ Postfix Evaluation.

In this algorithm operands push.

Algorithm:

```
if (operand)
    push
if (operator)
  {
  operator2 = POP()
  operator1 = POP()
    r = OP1 op OP2
    push(r)
  }
```

O(n)

If you saw operator stop. two time pop & perform that opn

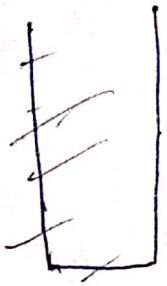
Operand stack.

Problem:

Postfix: (cd+)(ab-)* — 3 (operands: 2, 2)

Infix: (c+d)*(a-b) Here 2 is dxa so brackets may be required

Prefix: -*ab/cd — 3 (operands: 2, 2)



Infix: a*b-c/d

(a*b)-(c/d) [No ambiguity So no need to keep brackets]

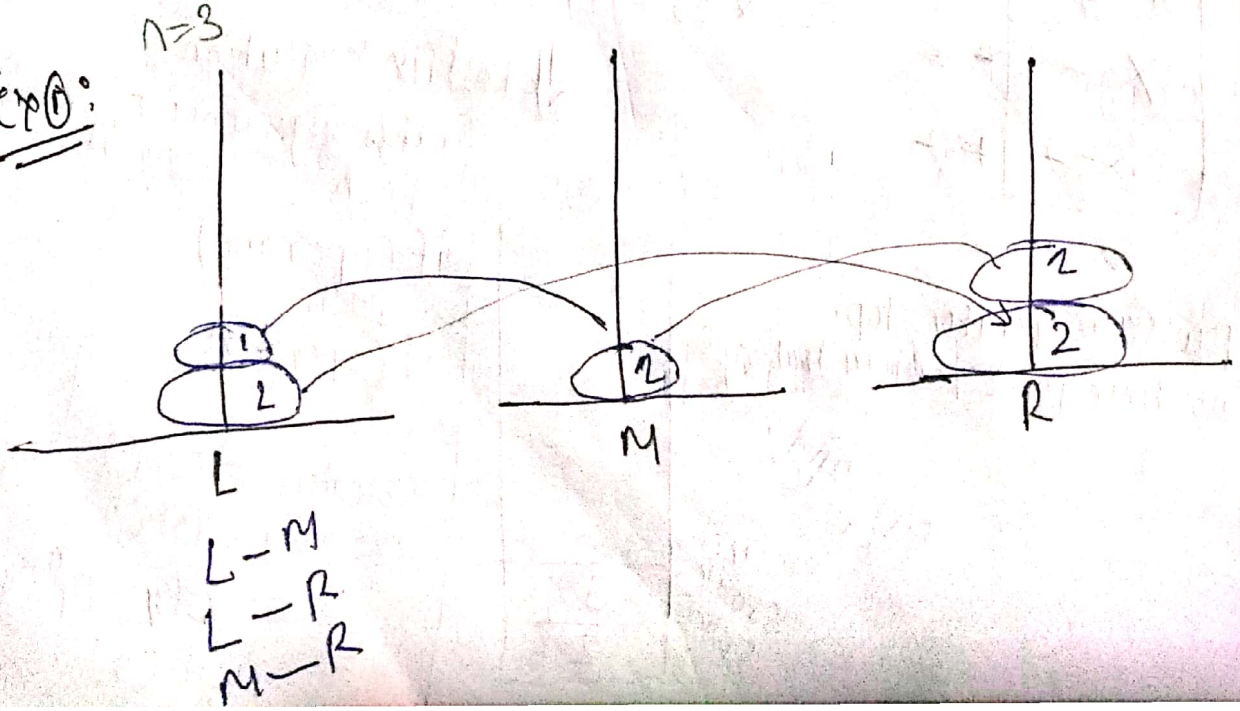
NOTE:
To Evalute postfix Expression one scan is enough but to evalut Infix & Prefix More than 1 ~~set~~ scan required.

Perili removed work pending.

21/Sept

Towers Of Hanoi:

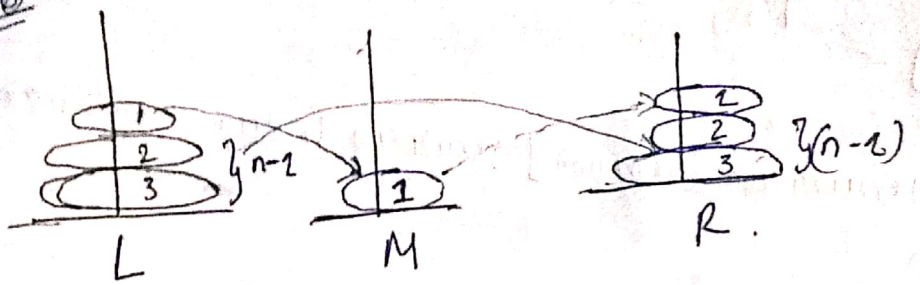
Exp:

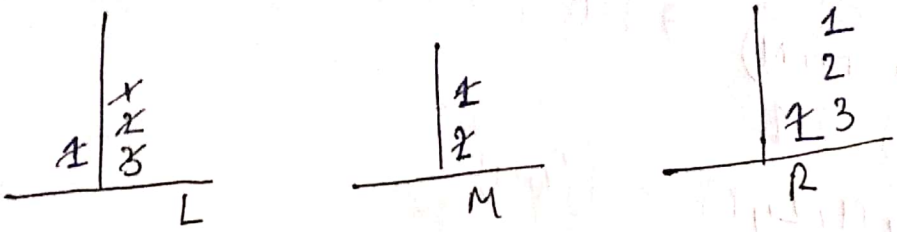


L-M
L-R
M-R

n = 3

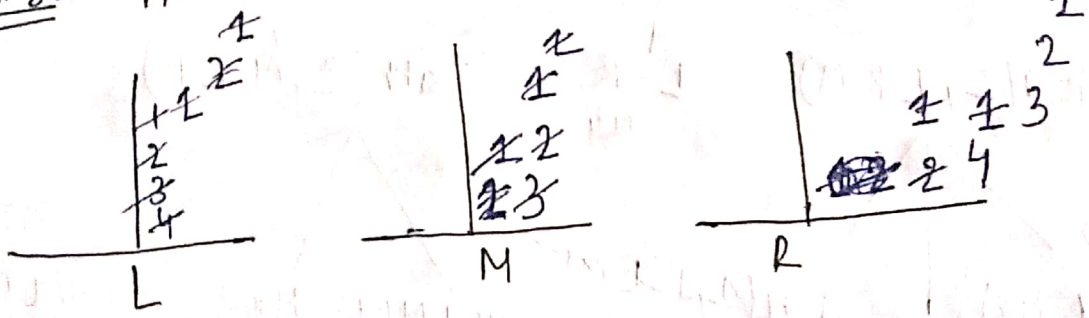
Ex①





L – R	L – R
L – M	M – L
R – M	M – R
	L – R

Ex②: n = 4

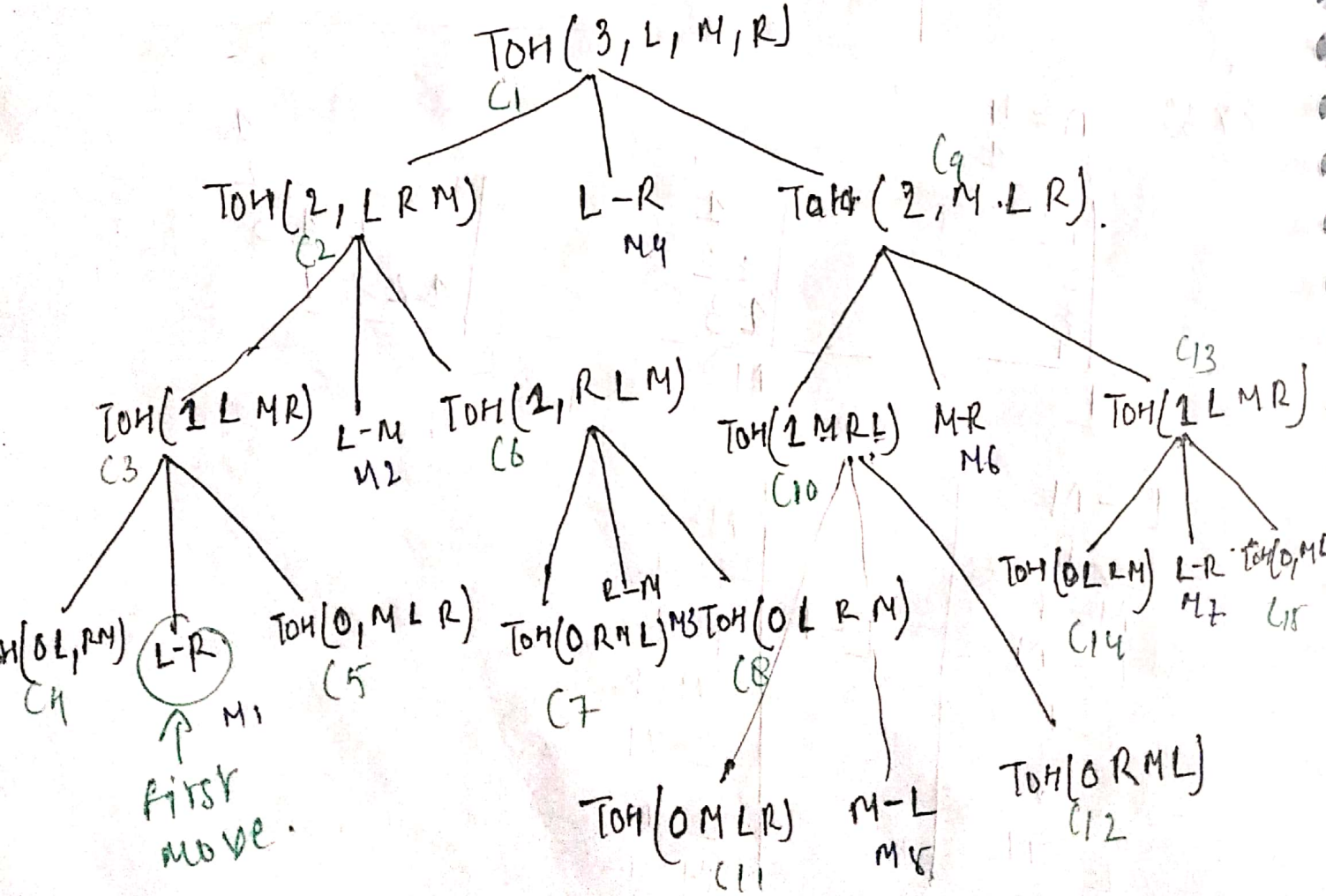


L – M	L – R
L – R	M – R
M – R	M – L
L – M	R – L
R – L	M – R
R – M	L – M
L – M	L – R
	M – R

Algorithm:

TOH(n, L, M, R) ⟹ T(n)

```
{
  if(n == 0) [termination condition] Sometimes In Ques - if will be 2 also
    return;
    else
  {
  TOH(n-1, L, R, M) ⟹ T(n-1)
  MOVE(L-R)   1
  TOH(n-1, M, L, R) ⟹ T(n-1)
  }
}
```



ans① In TOH of n After how many function call. first move will be take place.

Ans: Go to Extreme left. [n+2] [If the termination condition is 0] = ④

After C4 there is a M1.

If the termination condition is 1 then Ans is [n] = ③

ans② In TOH(3) after how many function call there is a move from middle to right.

AN: 12

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

After C12 there is a M6

ans:③ ∴ In TOH of n TOH(n) Is any move after last function call.

Ans: No work In the backtracking. So No move after last function call.

After C15 In Backtracking No work.

ans: In TOH(n) is there is any function call after last move:

AN: 1 function. : C15

ans: In TOH(n) How many Moves there?

Ans:

$T(0) = 0$ Move

$T(1) = 1$ Move

$T(2) = 3$ Move

$T(3) = 7$

$T(4) = 15$

$T(10) = 2T(9) + 1$

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

$= \boxed{2^n - 1}$ $\boxed{2^n - 1}$

Ques. In TOH(n) How many function calls are there.

$TOH(0) = 1$

$TOH(1) = 3$

$TOH(2) = 7$

$TOH(3) = 15$

(3) (15)

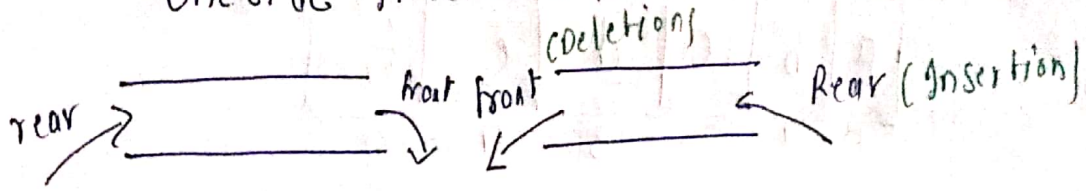
$TOH(10) = 2T(9) + 1$

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

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

Queue

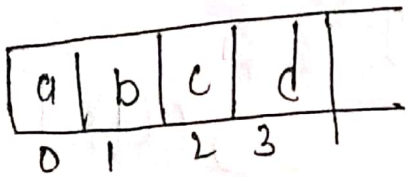
Defination: Both the side open. but the condition is one side Insertion, otherside deletion

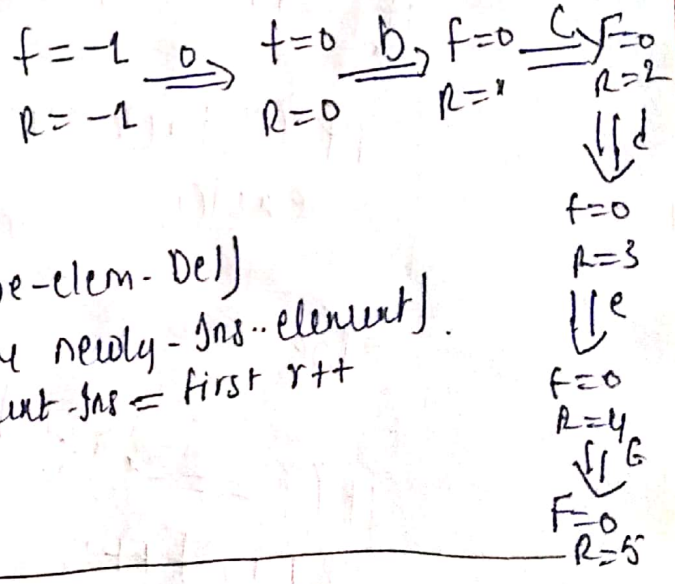


Insertion will be done w.r.t. 'Rear'
Deletion " " " " "Front'

* **Front:** Front is a variable which contain position of the element to be deleted

* **Rear:** Rear is a variable which contain position of the newly inserted element.





Front: 0 (Position of the-elem- Del)
Rear: 3 (Position of the newly-Ins.-element)
Next-Element-Ins = First r++

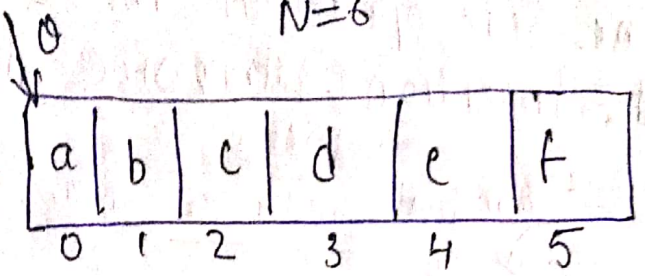
Property: FIFO or LILO

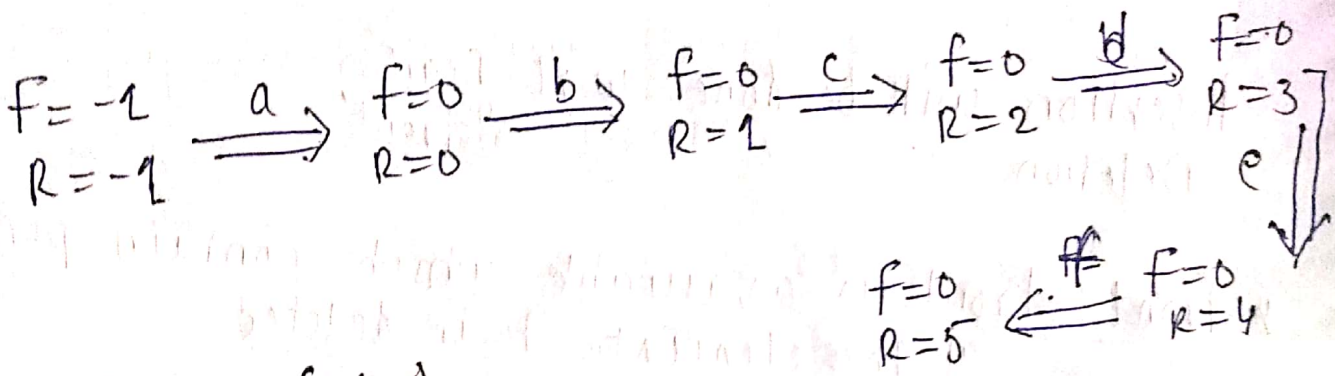
ADT of Queue

↓

1. Enqueue()
2. Dequeue()

Ques: Implementing Queue using array:

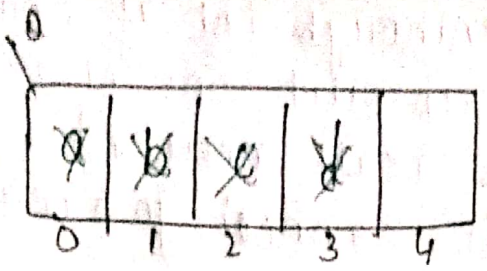





```
Void Enqueue(int x)
  {
  if(R+1==N) or R=N-1
  {
   Pf ("Stack Overflow");
     exit(1)
   }
  else.
  {
   if(R==-1)
    R++, F++;
     else
    R=R+1
    Q[R]=x
   }
}
```

⇒ O(1) [CL]

1st insertion



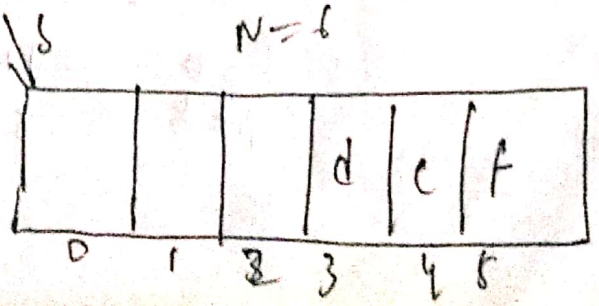
f=0, R=3 —Del→ f=1, R=3 —Del→ f=2, R=3 —Del→ f=3, R=3 —Del→ f=-1, R=-2

f==R (1 element)

```
Dequeue()
{
if (F==-1)
Pf("Queue is underflow").
   exit(1);
   else
{ y = Q[F] {first store the 2nd element)
   if (F==R) (Deleting then (last element)
     F=R=-1
     else
     F=F+1
}
}
```

⟹ O(1) [EC]

Using above program we can Implement Queue. but it is not efficient.

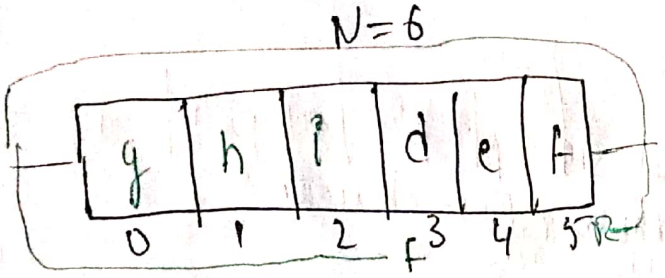


F=3
R=5

In the above queue eventhough lot of space available we cannot insert one more element further. coz rear reached right most ~~pat~~ place & it cannot go back because it is a liner Queue.

To Implement Queue efficiently we are going to the Circular Queue.

Circular Queue



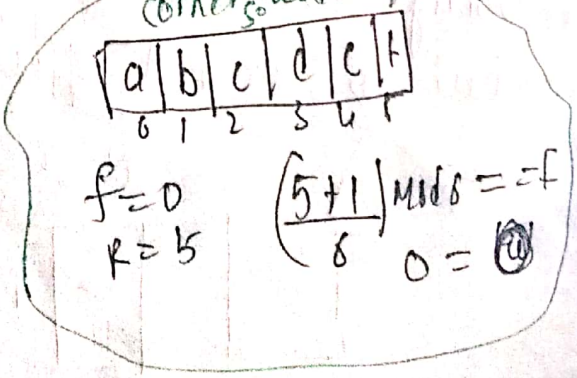
F=3, R=5 → Insert g → f=3, R=0 → h → F=3, R=1 → i → f=3, R=2 → j

(corner problem happen so mod required)

```
void Enqueue(int n)
{
  if((R+1)%N==F)        // change
  {
  Pf("Queue Overflow");
    exit(1)
  }
  else
  {
  if(R==-1)
    R++; F++;
```

if(R+1)%N==F)
C-0. full

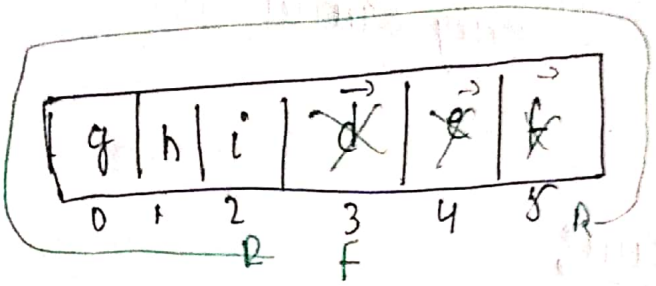
problem
corner so mode required

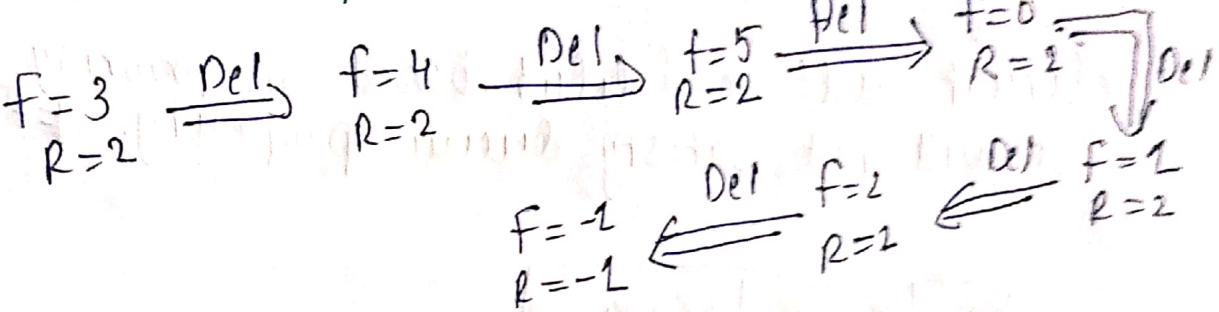


else
R = (R+1) Mod N Only change

Q[R] = x

}






```
int Dequeue()
{
if (F==-1)
{
Pf("Queue is underflow");
exit(1)
}
else
{
y = Q[F]
if (F==R)
F = R = -1
else
F = (F+1) Mod N   only change
return(y);
}
}
```

Priority Queue:

- Ascending Priority Queue
- Descending Priority Queue.

Ascending Priority Queue:

Element will be deleted based on priority.
It will not satisfy Queue property. ~~#~~ FIFO.

~~28~~	80	~~18~~	90	~~36~~	39	48

15 25 26 ..... (whoever the higest priority should be deleted first).

Implementing Ascending Priority Queue:

3 way to repres

① Using unsorted array.

25, 80, 99, 17, 36, 15, 28

a

25	80	99	17	36	18	18
0	1	2	3	4	5	6

i = 0
j = ~~-1~~ 6

Enqueue:

ist - EO
i++
j++
a[i] = x

O(1)

Dequeue
j++
a[j] = x

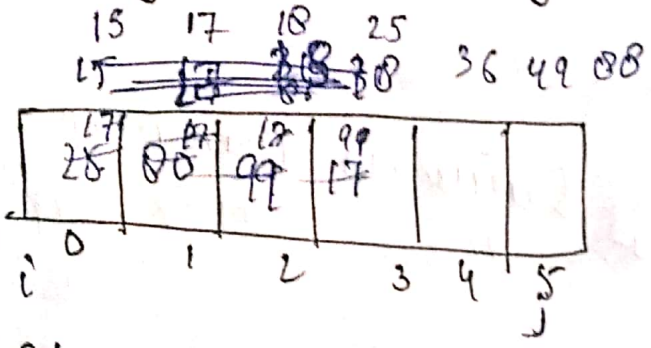
Deletion:

① k = position minelement ⟹ O(n) { compulsory 1 scan to find minimum element

② y = a[k]
a[k] = a[j]
j = j − 1
return(y)
} O(1)

1 − DQ
↓
O(n)

② Using Sorted Array:



EQ:

① Store x in j^{th} place.
② Keep x in correct place } O(n) } O(n)

1 − EQ
↓
O(n)

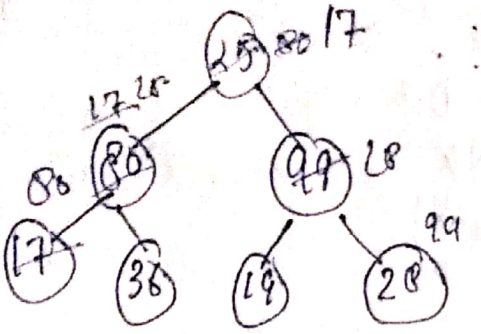
DQ:

y = a[i]
i++
return(y)

1 − DQ
↓
O(n)

③ Using Min heap Best way

25, 80, 99, 17, 36, 19, 28.



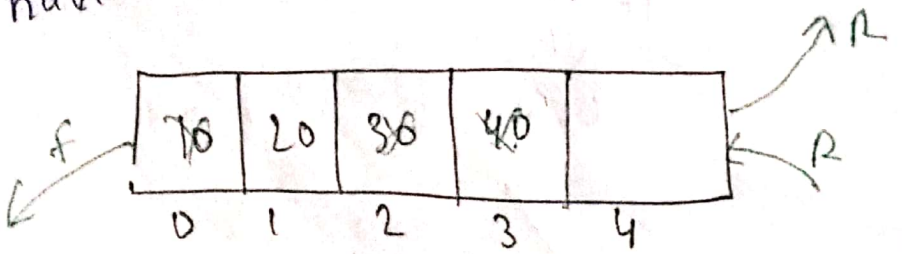
I — EO ⇒ logn
I — DO ⇒ logn

O(logn)

Ascending priority Queue is min heap
" " Max heap.
Descending "

Input- Restricted Queue:

Both the side we can do deletion. only Input have restriction.



f=0 1
R=1 3 4 3 2

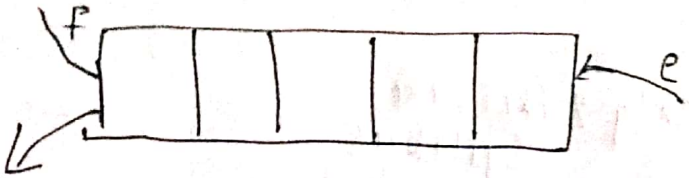
EO
R++

DO-f
f++

DO-R
R--

Not FIFO

Output restricted queue:



DQ
F++

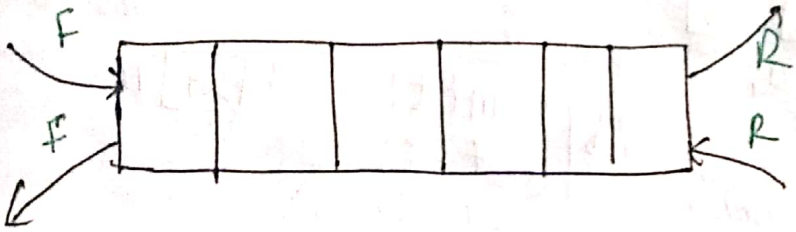
EQ – R
R++

not-FIFO

EQ – f
F--

Double-Ended Queue: (Dequeue)

└ Depending upon context you have to consider



Dequeue Opⁿ : Delete Element from queue.
Dequeue Data Structure: Double Ended Queue

EQ – R
R++

DQ – F
F++

EQ – F
F--

DQ – R
R--

Parameter Passing Techniques:

① Call By Value. } Only 2 Here in the Syllabus.
② Call By Reference }

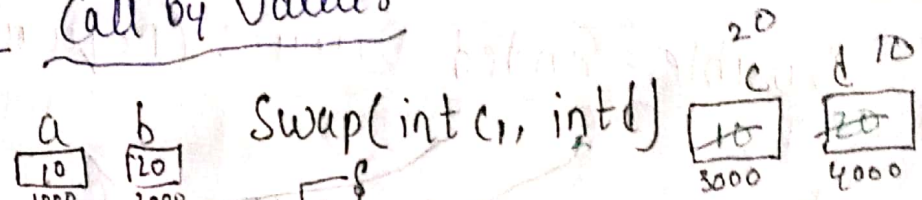
③ Call By Name
④ Call By Need
⑤ Call By Text
⑥ Call By Copy-restore. } Not there in the Syllabus.

Call By Value:

Ex①:

```
Main()
{
① int a=10, b=20;
② Pf(a,b);
③ Swap(a,b);
④ Pf(a,b);
}
```

```
Swap(int c, int d)
{
① int t;
② t=c;
③ c=d;
④ d=t;
}
```



O/P: 20, 20
10, 20

① In Call By Value Original will not affect. Because modification is done in copy.

② If the No's of parameters are very less then go to Call By value.

② Call by Reference:

[Compiler will do

```
Main()
{
① int a=10 b=10
② Pf(a,b)
③ Swap(&a,&b)
④ Pf(a,b);
}
```

a 10 1000 20 b 10 2000

Swap(int *c, int *d).

```
{
① int t;
② t = *c
③ *c = *d
④ *d = t
}
```

c 1000 3000 d 2000 4000

t 10 5000

O/P: 10 20
20 10

① In call by reference original will affected because Modification with done w.r.t address.

② If you want pass more parameter call be reference is better option.

Ques: Consider the following C Program.

```
Main()
{
int a=10 b=20;
Pf(a,b);
f(&a,&b,a);
Pf(a,b);
}
```

a -20 10 1000 b 20 2000

O/p: Value
10 20
10 20

O/p: Reference,
10 20
-20 20

f(int *b, int a, int *c) 1000 *b 2000 a 1000 c

```
{
*b += *c - a
*c -= *b + a
}
```

20 - 20 - 10 -20

Ex(5): Main() Call by Reference.. o/p; Value

```
{
int a=10, b=20
pf(a,b);
f(a*b, a, a-b)
pf(a,b);
}
f(int *c, int a, int b)
{
*c += *a + 10;
*b -= *a + *c
*a *= 10
}
```

100

a [10] 1000 b [20] 2000

o/p: 10 20
10 20

o/p. References.

10 20
100 20

It will be done in temporary variable

a*b [200] 4000 220

4000

Hat place address will be set

-240

[-10] a-b

3000 Address

c [4000] 5000 a [1000] 8000 [3000] b 7000

8/Oct/2019

COMPILER DESIGN

Grammer G(V T P S)

- S → Start Symbol = (C)
- P → Production
- T → Terminal (a, d, b)
- V → Variables {a, B, C, AB, C}

S → ABC
A → BC
B → ac
C → d | b

Types of Grammer:
NOAM

According to Chomsky

1. Type-0 [Unrestricted Grammer]
2. Type-1 [Context-Free Grammer]
3. Type-2 [Context-Sensitive Grammer]
4. Type-3 [Regular Grammer]

① Type 0:

$\alpha \rightarrow \beta$

$\alpha\beta \in (V+T)^*$

② Type 1:

① It should be type 0

② $|\alpha| \leq |\beta|$

Type 2:

① Type 1

② $A \rightarrow B$

A is a single variables.

$\beta \in (V+T)^*$

Type 3:

→ two variable cannot be side by side

S → ABC(X)

① Type 2

② $A \rightarrow BT^*/T^*$ (Either) ↓ Left linear

② or ③ (or)

③ $A \rightarrow TB^*/T^*$ ↓ Right - Linear

Both cannot be there

A, B are variables