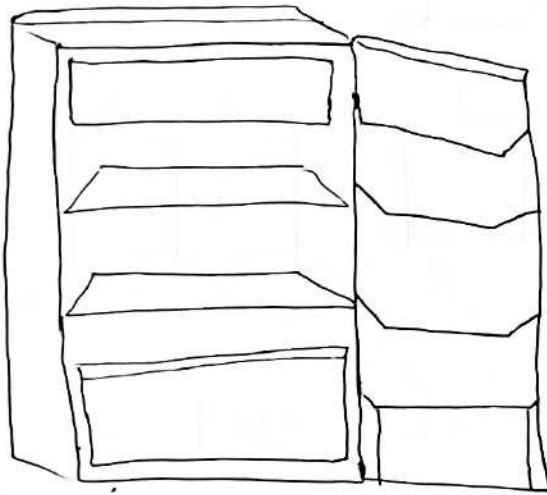


- Introduction
- Array
- Linked List
- Stack
- Queue
- Tree
- Heap
- Graph
- Hashing

① What is Data structure?

→



- ① store
- ② Organize
- ③ Access
- ④ Manipulate

②

LDS

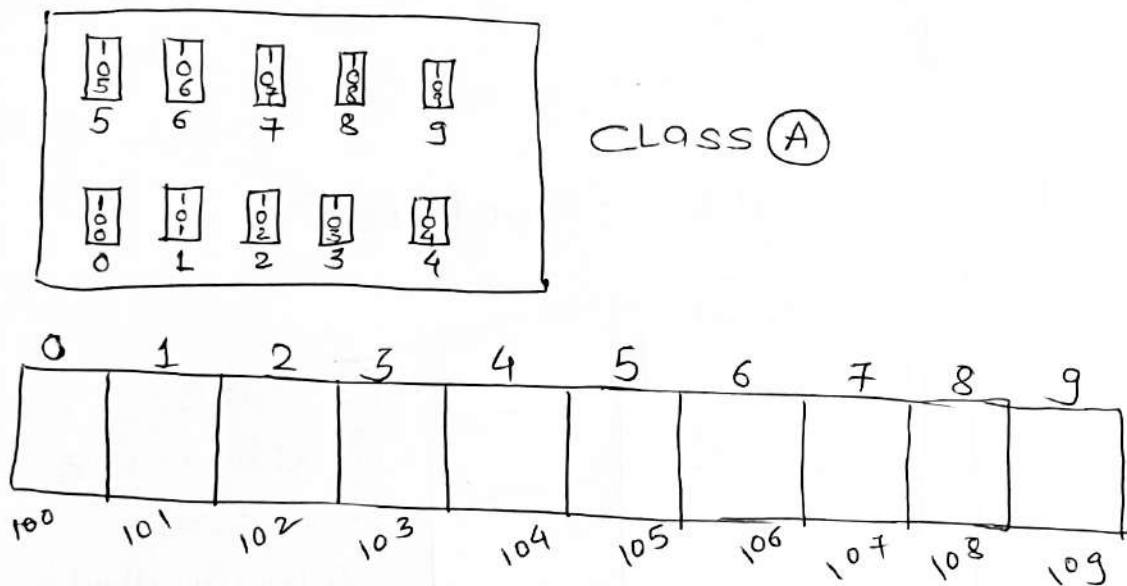
VS

NLDS

- | | |
|---|---------------------------|
| → Adjacently Attached | → Hierarchically Attached |
| → single Level | → Multiple Level |
| → Easy | → Difficult |
| → Single Run | → Multiple Runs |
| → Memory is not Utilized in efficient way | → In efficient way |
| → Array, Stack, LL | → Tree Graph |

③ Array

- store
- Same type
- Contiguous Memory Locations
- Index Accessing



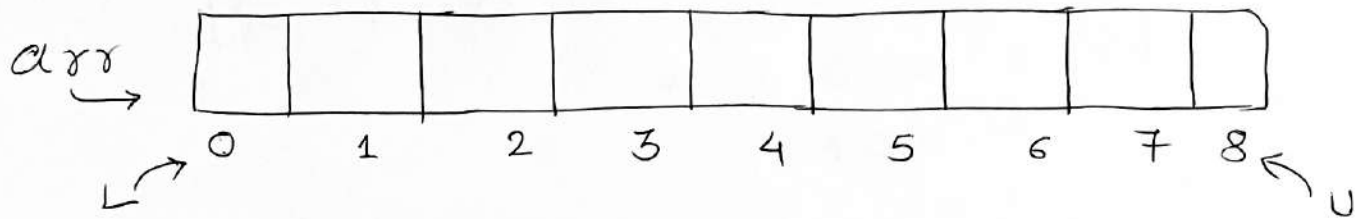
→ Datatype ArrayName [A_size]

↓ ↓ ↓

int Arr [10]

- Random Accessing
- Easy Retrieval
- Easy to use & understand
- Fixed nature
- Insert / Delete Issues

→ 1D Array



$$\text{Size}(\text{arr}) = U - L + 1 = 8 - 0 + 1 = 9$$

$$\rightarrow \text{BA} = 1000$$

$$U = 8$$

$$L = 0$$

$$S = 1$$

$$\text{Addr}(\text{arr}[i]) = \text{BA} + S [i - L]$$

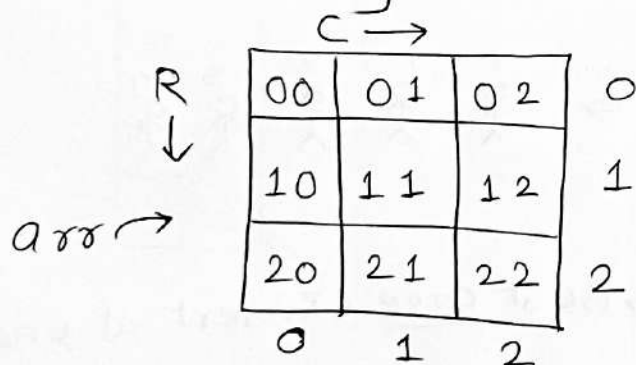
$$\downarrow \quad \downarrow \quad \rightarrow \quad = 1000 + 1 [0 - 0]$$

$$\rightarrow = 1000$$

$$\downarrow \quad \downarrow \quad \rightarrow \quad = 1000 + 1 [1 - 0]$$

$$\rightarrow = 1001$$

→ 2D Array



$$\text{BA} = 1000$$

$$S = 1$$

$$L_R = 0$$

$$U_R = 2$$

$$L_C = 0$$

$$U_C = 2$$

$$\text{Addr}(\text{arr}[i][j]) = \text{BA} + S [(i - L_R)(U_C - L_C + 1) + (j - L_C)]$$

$$\downarrow \quad \downarrow \quad = 1000 + 1 [(1 - 0)(3) + (2 - 0)]$$

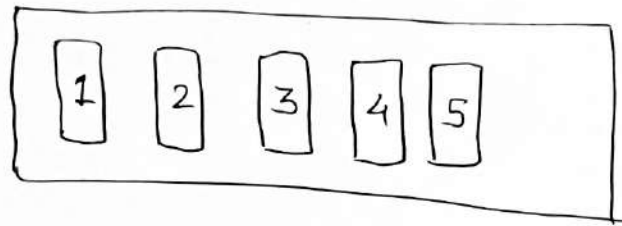
$$\rightarrow = 1000 + 5$$

$$= 1005$$

$$\text{Addr}(\text{arr}[i][j]) = \text{BA} + S[(j - L_C)(U_R - L_R + 1) + (i - L_R)]$$

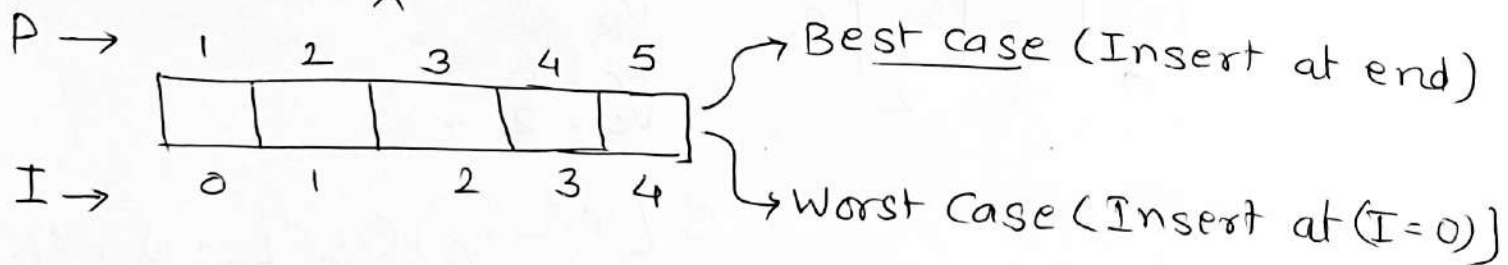
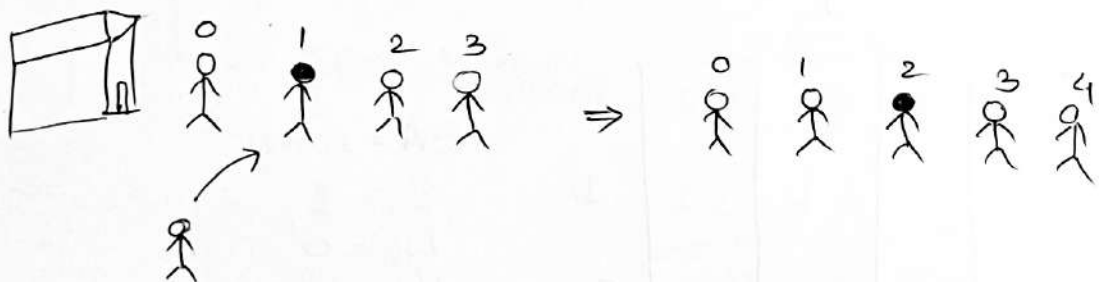
$$\begin{aligned} \downarrow \quad \downarrow \\ 1 \quad 2 &= 1000 + 1[(2 - 0)(3) + (1 - 0)] \\ &= 1000 + 7 \\ &= 1007 \end{aligned}$$

→ Traversal operation :-



```
for (i = 0; i < n; i++)
{
    printf(arr[i]);
}
```

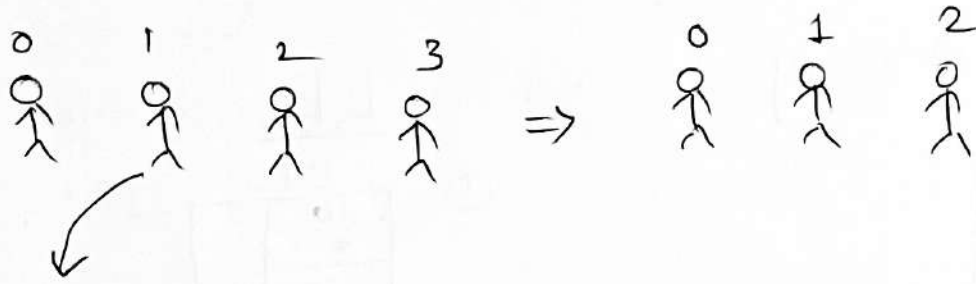
→ Insertion operation :-



$$P = I + 1$$

$$\text{or} \\ I = P - 1$$

→ Deletion Operation



10	20	30	40	50
0	1	2	3	4

↓ delete ($I=1$)

10		30	40	50
----	--	----	----	----

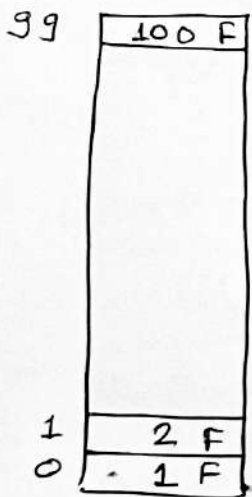
↓

10	30	40	50	
----	----	----	----	--

Best case
(Last element)

Worst case
($I=0$)

→ Searching operation



Best case (1^{st} floor)

Worst case (100th floor)

Array Vs LL

① Contiguous
ML

① Not

1	10	3
2		
3	20	5
4		
5	30	10
	D	L

② Random
access

② Sequential Access

③ I & D takes
More time

③ I & D → fast

④ Fixed, static
Nature

④ Dynamic, changes
are accepted.

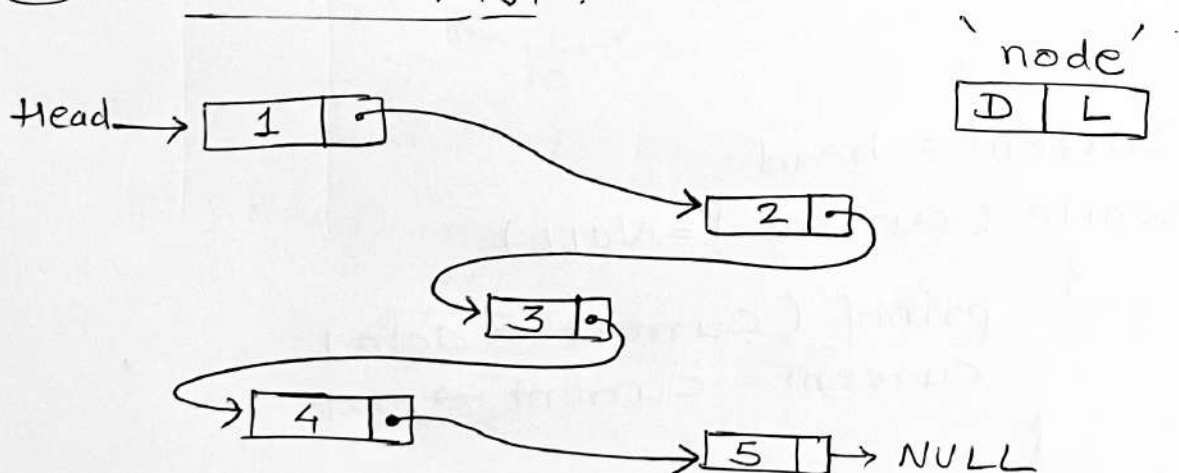
⑤ 1D, 2D, ... MD

⑤ SLL, DLL, CLL

⑥ Sequential Representation

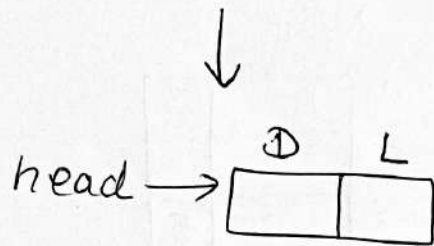
⑥ Linked Representation

④ Linked List:



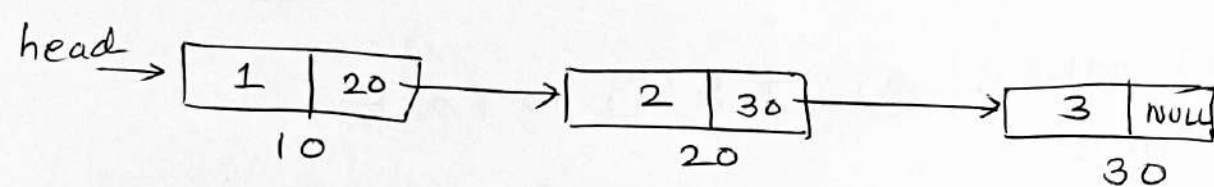
```
struct node
{
    int data;
    struct node *link;
}
```

node* head = malloc(sizeof(node))



head → data = 5

head → link = NULL



head → data = 1

head → link = 20

head → link → data = 2

head → link → link = 30

head → link → link → data = 3

head → link → link → link = NULL

Traversal

⇒ current = head

while (current != NULL)

{

printf (current → data)

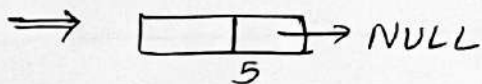
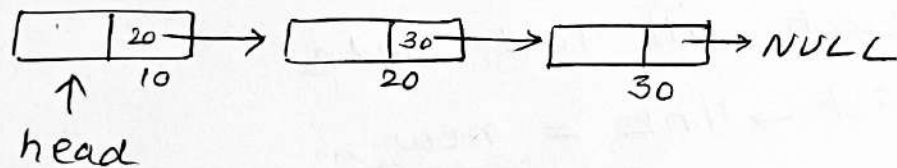
current = current → next

}

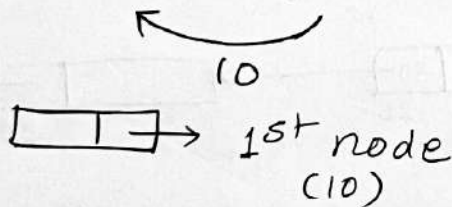
Searching

```
⇒ current = head
while (current != NULL)
{
    if (current → data == key)
    {
        return current
    }
    current = current → link
}
```

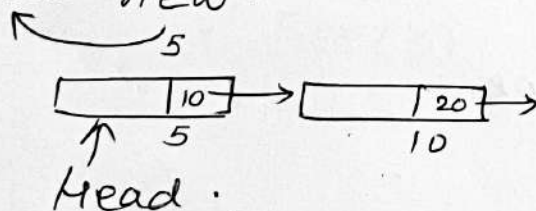
Insert → At Beginning.



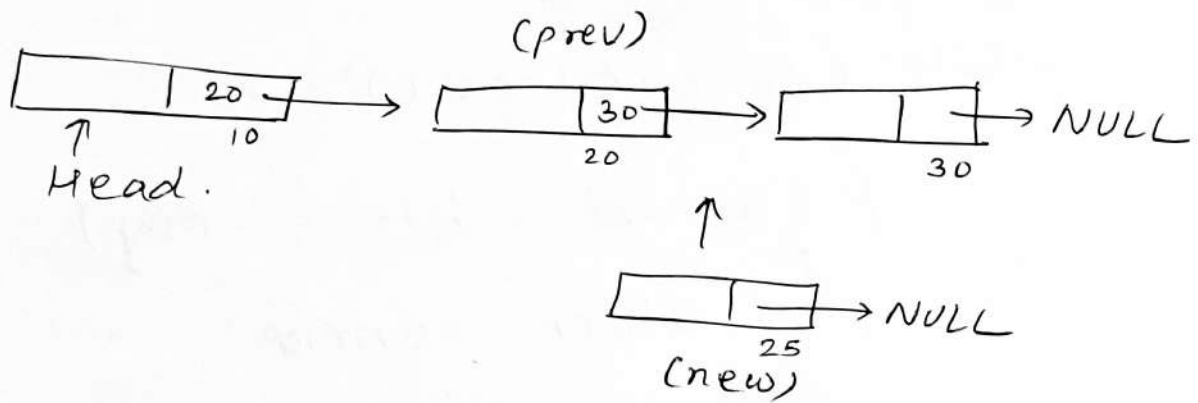
① new → next = head



② head = new



Insert → after a node.

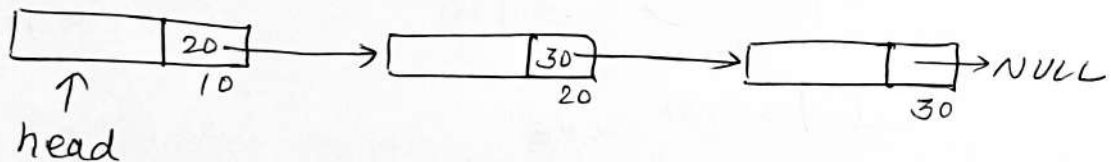


- ① $new \rightarrow next = prev \rightarrow next$
- ② $prev \rightarrow next = new$.

Insert → at End

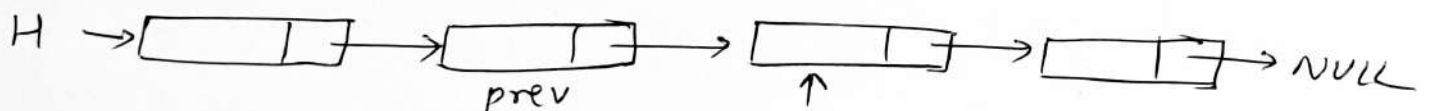
- ① Reach till last node
- ② $last \rightarrow = new$

Delete → At Beginning.



- ① $del = head$
- ② $head = del \rightarrow link$
- ③ $free(del)$

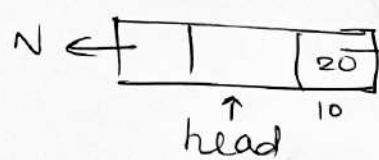
Delete → after a node.



- ① $del = prev \rightarrow link$
- ② $prev \rightarrow link = del \rightarrow link$
- ③ $free(del)$

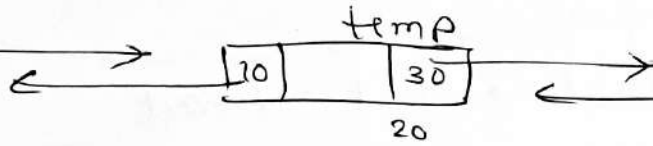
Insert

At Beginning



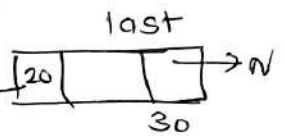
- $\text{new} \rightarrow \text{next} = \text{head}$
- $\text{head} \rightarrow \text{prev} = \text{new}$
- $\text{head} = \text{new}$

After a node



- $\text{new} \rightarrow \text{next} = \text{temp} \rightarrow \text{next}$
- $\text{temp} \rightarrow \text{next} \rightarrow \text{prev} = \text{new}$
- $\text{temp} \rightarrow \text{next} = \text{new}$
- $\text{new} \rightarrow \text{prev} = \text{temp}$

At End



- $\text{last} \rightarrow \text{next} = \text{new}$
- $\text{new} \rightarrow \text{next} = \text{last}$

Delete

At Beginning

- $\text{temp} = \text{head}$
- $\text{head} = \text{head} \rightarrow \text{next}$
- $\text{head} \rightarrow \text{prev} = \text{NULL}$
- $\text{free}(\text{temp})$

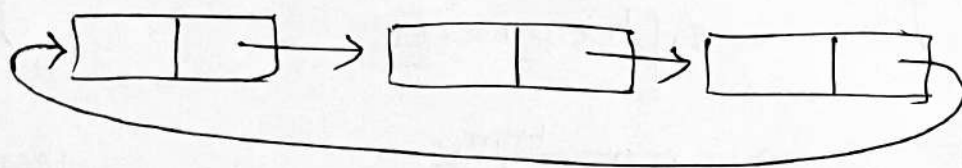
After a node

- $p \rightarrow \text{next} = p \rightarrow \text{prev} \rightarrow \text{next}$
- $p \rightarrow \text{next} \rightarrow \text{prev} = p \rightarrow \text{prev}$
- $\text{free}(p)$

At End

- $\text{temp} = p \rightarrow \text{next}$
- $p \rightarrow \text{next} = \text{NULL}$
- $\text{free}(\text{temp})$

→ Circular Linked List

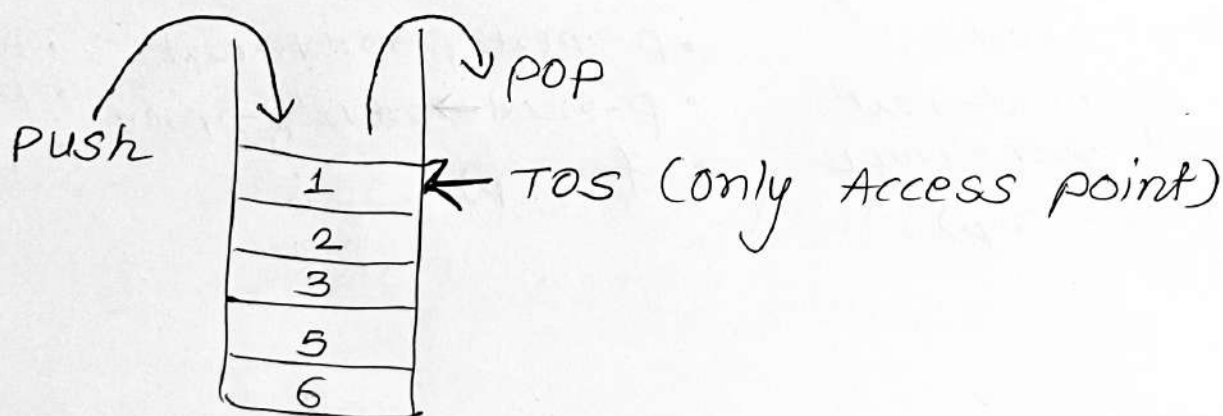


last \rightarrow next = head

- ②. temp = head
- ③. head = head \rightarrow next
- ⑤. free(temp)
- ①. current = head
- while (current \rightarrow next \neq head)
- {
- current \Rightarrow current \rightarrow next
- }
- ④. current \rightarrow next = head

⑤ Stack

- LDS
- I/D \rightarrow from only one End
- FILO/LIFO
- Element at Top (TOS)



- Static Imp: Arrays
- Dynamic Imp: Linked List
- } I/D from "Only One End"

→ Push (arr, n, TOS, x)

{

 If (TOS == n-1)

 { printf("stack is full")

 }

 TOS = TOS + 1

 arr[TOS] = x

}

→ pop (arr, n, TOS)

{

 If (TOS == -1)

 { printf("stack is empty")

 }

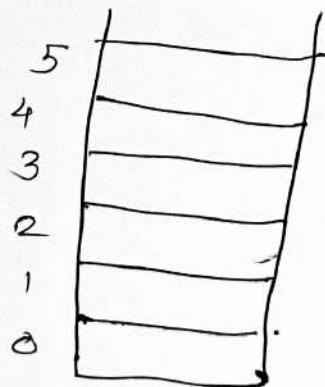
 x = arr[TOS]

 TOS = TOS - 1

 return (x)

}

Q → push(1), push(3), pop(), push(5), push(10)
push(10), pop(), pop(), pop(), pop()



Infix

Prefix

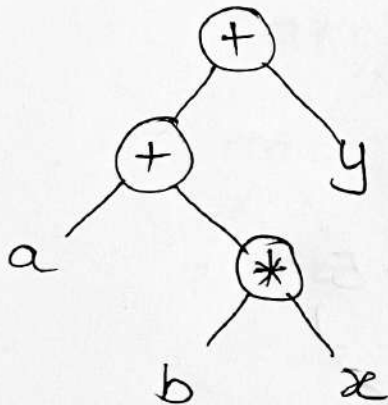
Postfix

$a + b * x + y$
(l-r-x)

$++ a * b x y$
(R-l-x)

$abx * + y +$
(l-x-R)

(Tree)



a

+

b

*

x

+

y

Stack

-

+

+

+

+

+

+

Postfix

a

a

ab

ab

abx

abx*+

abx*+y

↓

abx*+y+

$a + b * x + y$

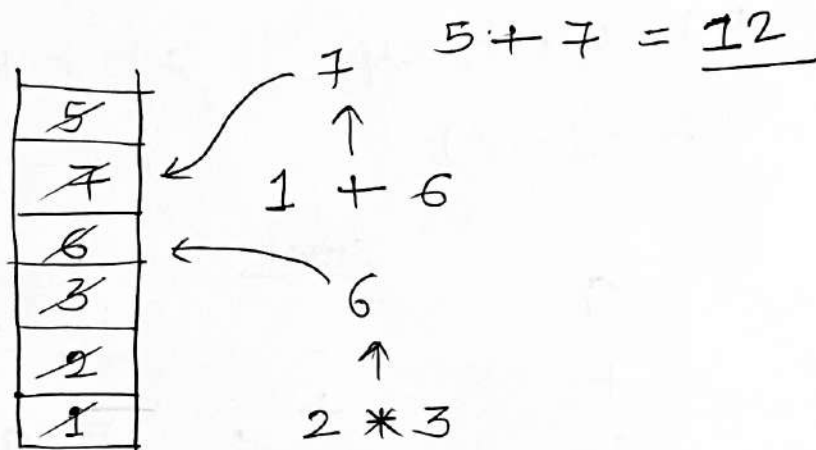
$a + bx * + y$

$abx * + + y$

$\rightarrow abx * + y + \leftarrow$

^ (R→L)
↓ * / (L→R)
+ - (L→R)

$$\Rightarrow 1\ 2\ 3\ * + 5 + (a\ b\ x\ * + y +)$$



$$a + b * x + y = 1 + 2 * 3 + 5$$

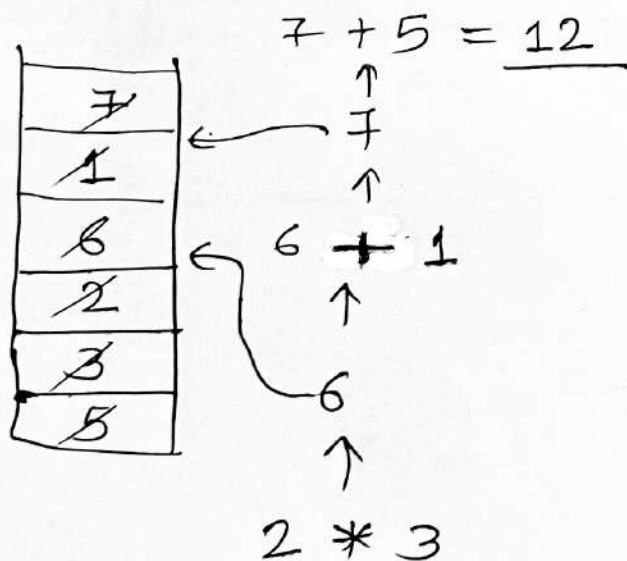
Diagram illustrating the evaluation of the expression $1 + 2 * 3 + 5$ using a stack.

The stack (from top to bottom) contains: 5 , 7 , 6 , 3 , 2 , 1 .

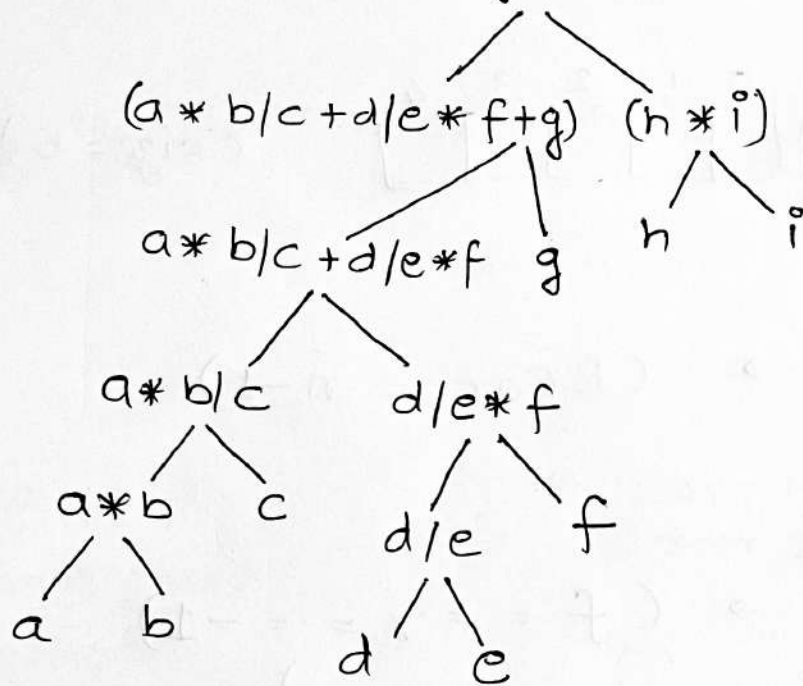
Operations and results shown:

- $2 * 3 = 6$ (indicated by an arrow pointing to the stack element 6)
- $1 + 6 = 7$ (indicated by an arrow pointing to the stack element 7)
- $5 + 7 = 12$ (indicated by an arrow pointing to the stack element 5)

$$++a * b\ x\ y = ++1 * 2\ 3\ 5$$

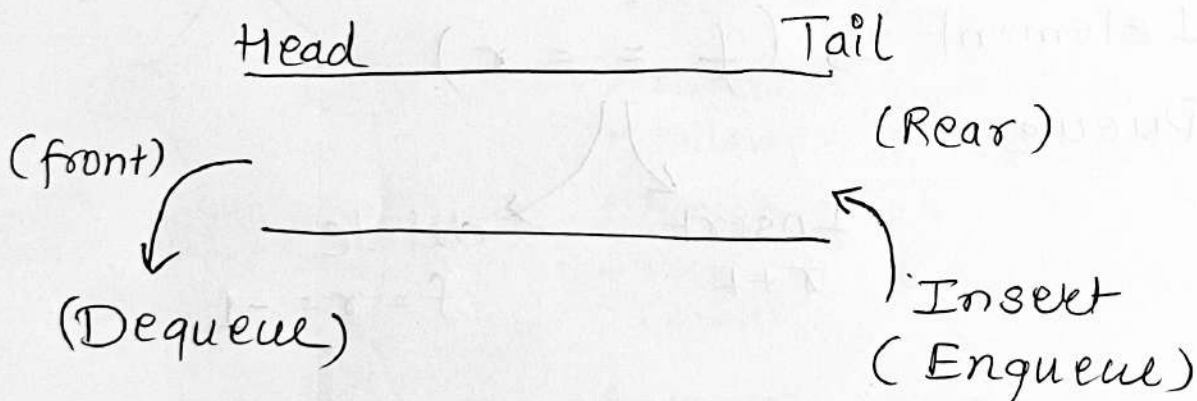


$$\Rightarrow a * b / c + d / e * f + g - h * i$$



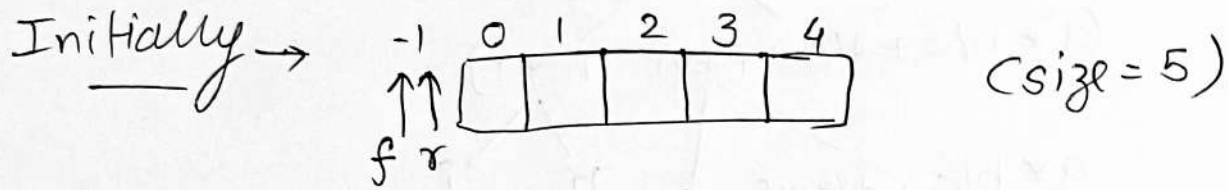
⇒ Queue

→ FIFO or LILO



Initially → front = rear = -1

→ Implementation using Array.



Overflow Condition → (Rear = n-1)

1st element Insertion Condition → (f = r = -1)

f = r = 0 (after insert)

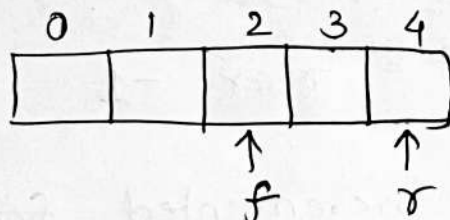
Underflow Condition → (f = r = -1)

Only 1 element In Queue → (f = r)

Insert
r++

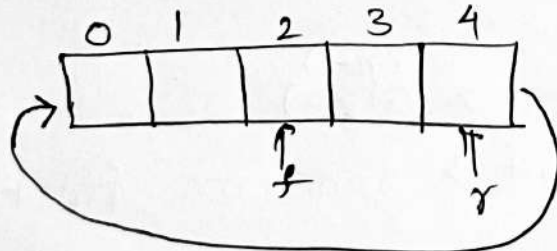
delete

f = r = -1



Now insert won't be possible due to overflow.

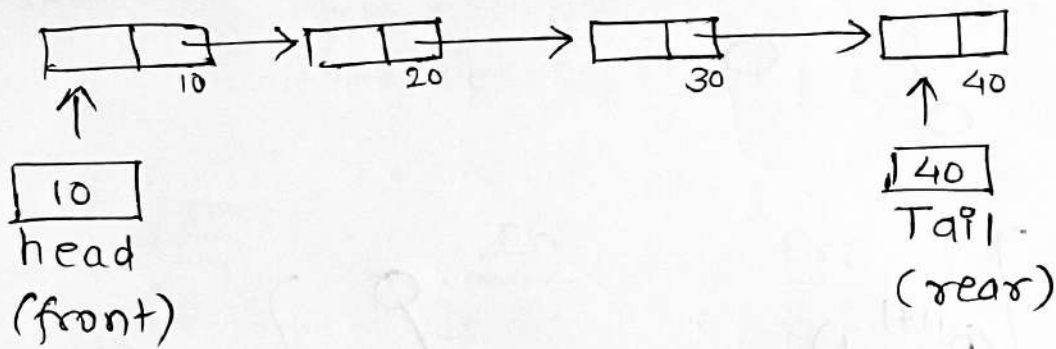
p++
(p+1) % n



So
Circular Queue.

here p → Rear
 → front

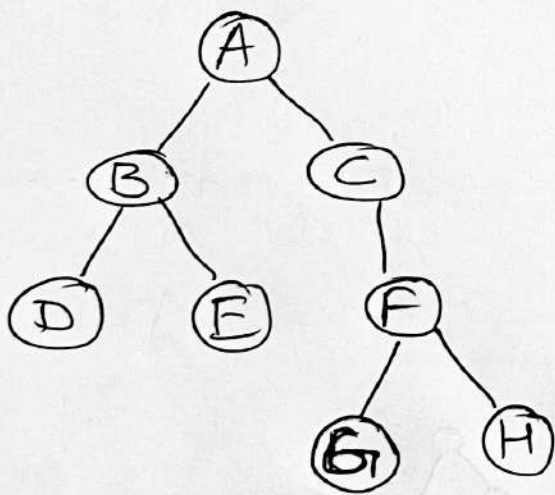
→ Implementation using LL



temp = front
 front = front → next
 free (temp)

rear → next = new
 rear = new

→ Tree



- root = A
- leaf = D, E, G, H
- Parent = A, B, C, F
- children = B, C, D, E, F, G, H
- Non-LN = \overline{LN}
- Path = A → C → F → H (A → H)

- Ancestor & Decendant (Before node) (A) (F, H)

- Sibling → D, E ✓
 B, C ✓
 G, H ✗

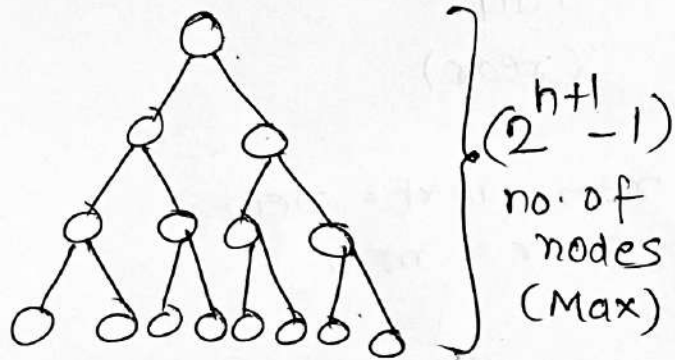
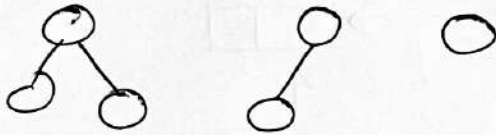
- Depth: Root → node
 (A) = 0 (D) = 2
 (B) = 1 (E) = 2
 (F) = 3 (H) = 3

- Height: node → leaf*
 (A) = 3 (C) = 2
 (F) = 1 (E) = 0

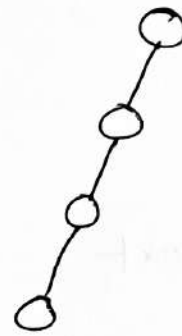
- Degree of node → A = 2
 B = 2
 D = 0

→ Binary Tree

↳ can have at most 2 children



$(2^{h+1} - 1)$
no. of nodes
(Max)

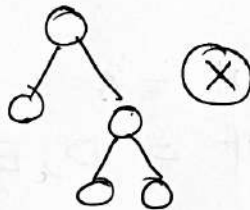
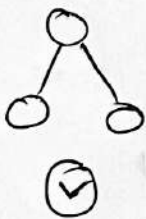


$(h+1)$
no. of nodes
(min)

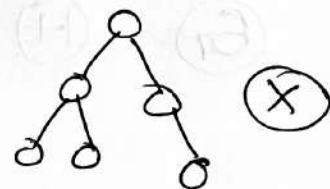
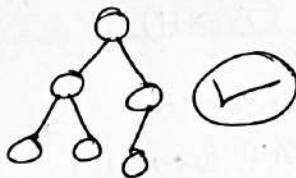
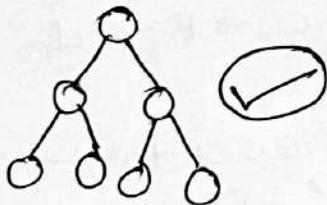
for $h=3$, $2^{3+1} - 1 = 15$

for $h=3$, $3+1 = 4$

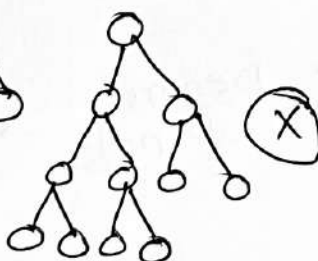
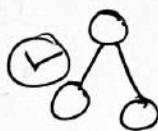
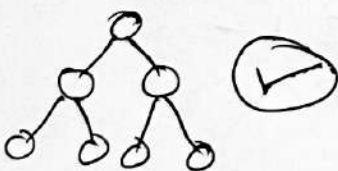
① Full BT (0 or 2)



② Complete BT

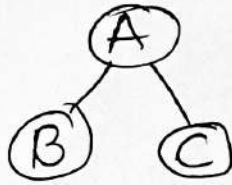


③ Perfect BT



⇒ Tree Traversal

preorder ($R \rightarrow l \rightarrow r$)
Inorder ($l \rightarrow R \rightarrow r$)
Postorder ($l \rightarrow r \rightarrow R$)



pre
ABC

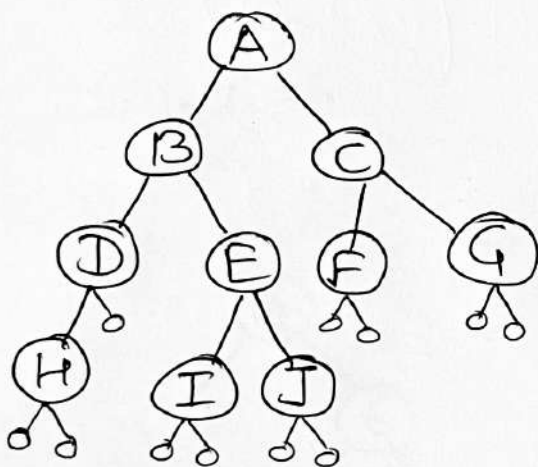
In
BAC

Post
BCA

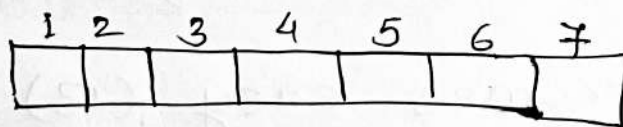
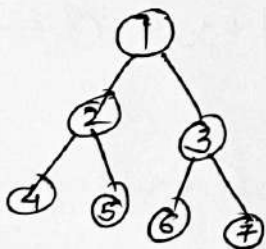
①
In

②
Pre

③
Post



→ Binary Tree (using Arrays)



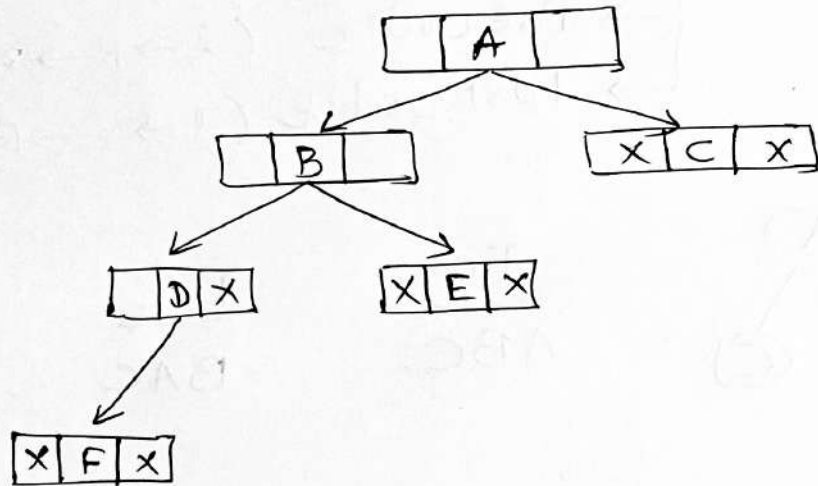
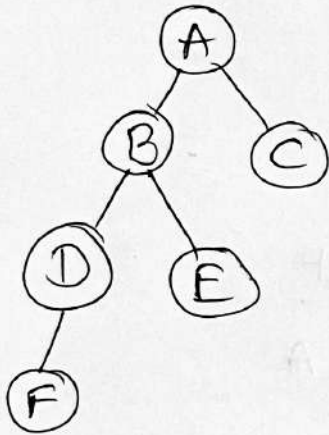
Root at $\rightarrow 1$

Left child $\rightarrow 2 * i$

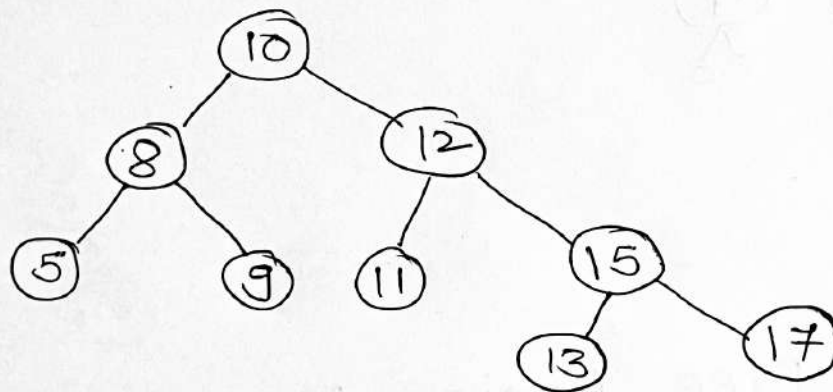
Right child $\rightarrow 2 * i + 1$

for eg:- $i = 3$
 $\rightarrow L = 2 \times 3 = 6$
 $\rightarrow R = 2 \times 3 + 1 = 7$

→ Binary Tree (LL)

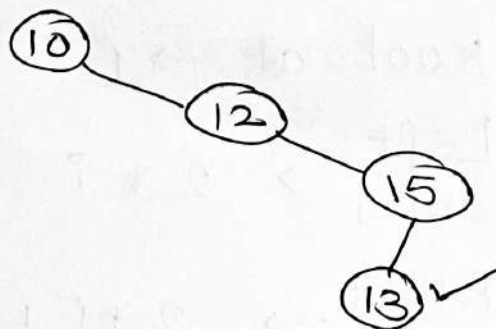


⇒ BST

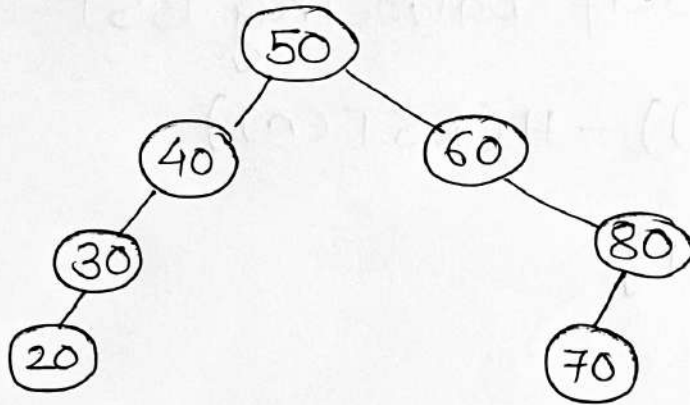


Inorder ⇒ 5, 8, 9, 10, 11, 12, 13, 15, 17

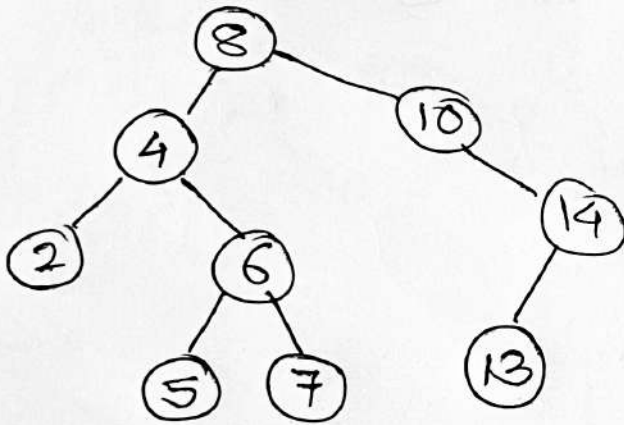
• Searching becomes easy. (13)



- Inserting (50, 40, 30, 60, 80, 20, 70)



- Deletion



- leaf node (Easy) direct (NO Impact)
 - One child node (14) → Replace it with that child (13)
 - 2 child node (4) → Replace it with inorder predecessor. (2).
- 2, 4, 5, 6, 7, 8, 10, 14, 13
 ↑
 for (8) → its (7)

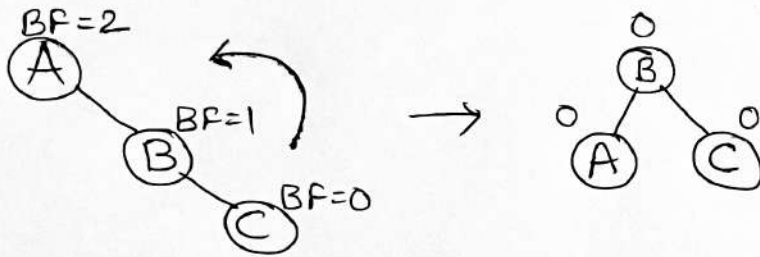
⇒ AVL Tree

↳ It's a self-balancing BST

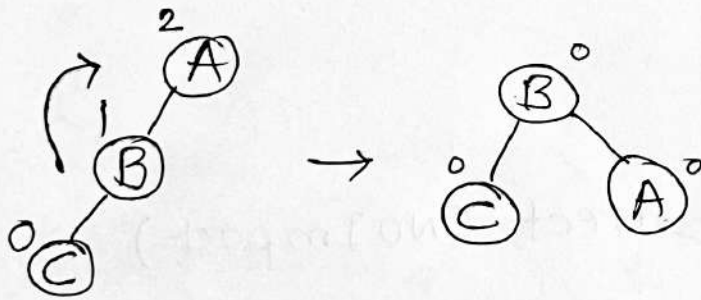
$$B_f = H(LST(n)) - H(RST(n))$$

$$\hookrightarrow \{-1, 0, +1\}$$

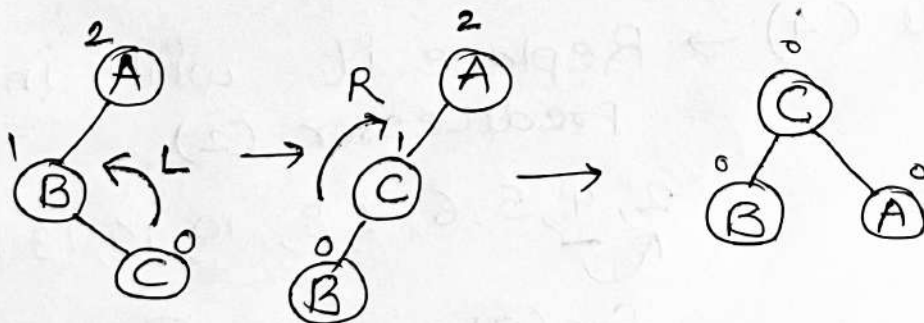
① Left Rotate



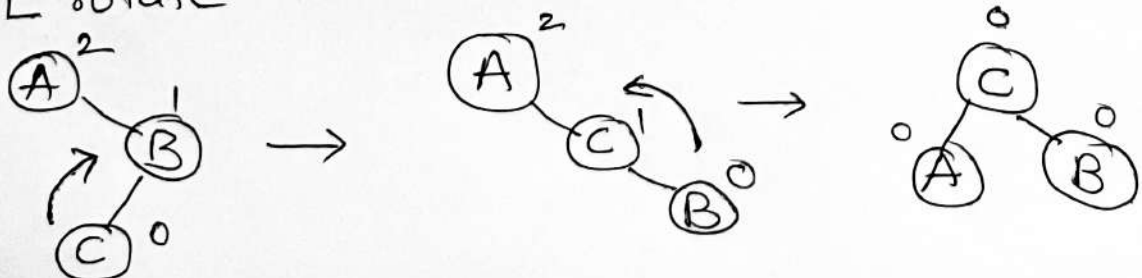
② Right Rotate



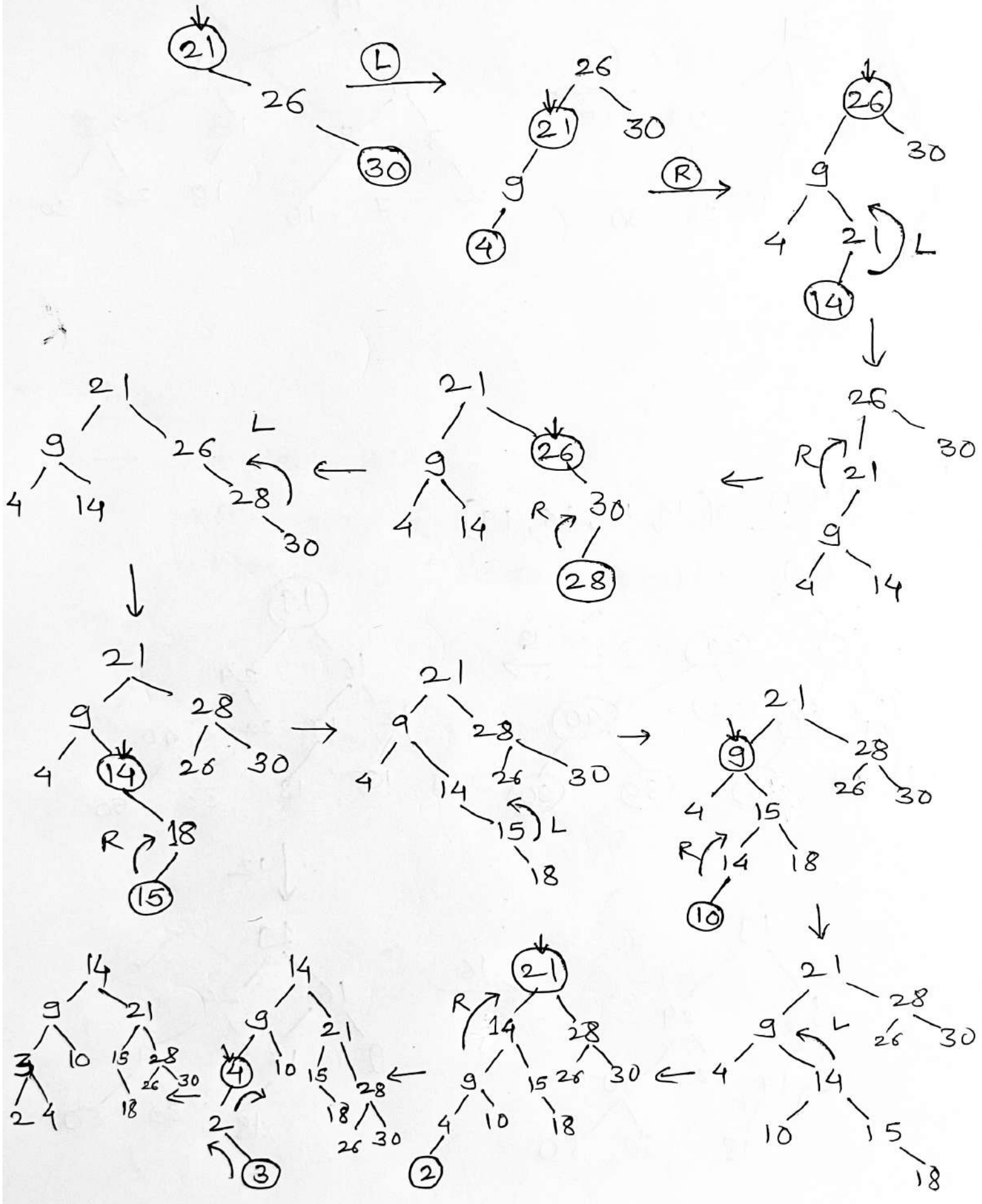
③ LR rotate

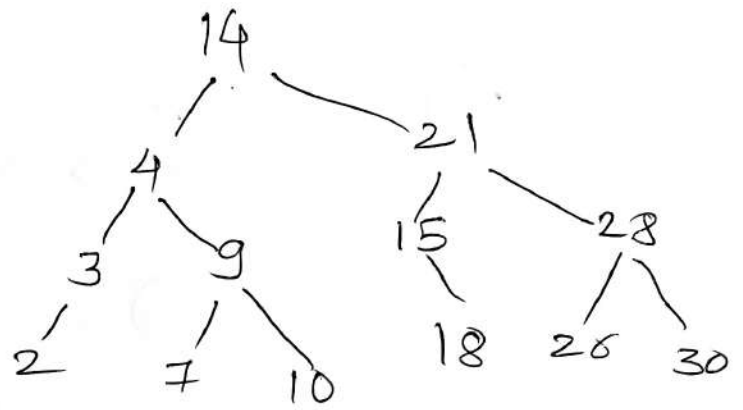
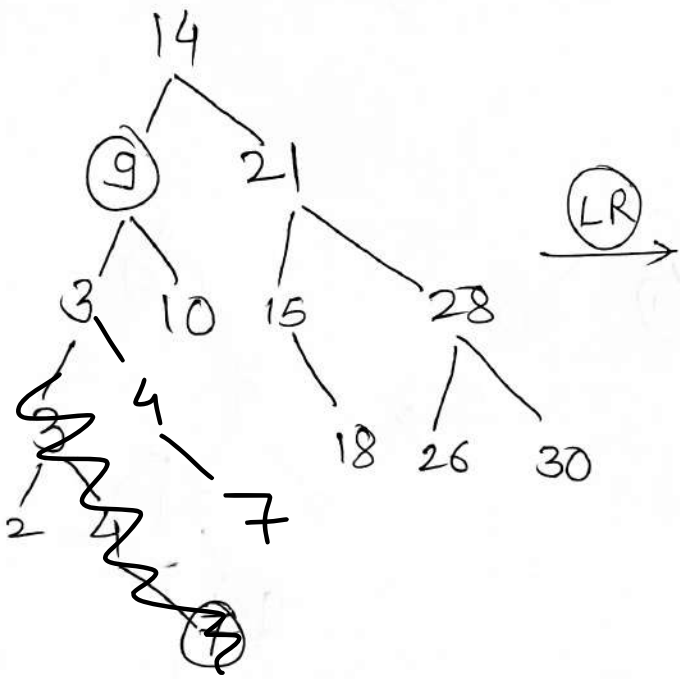


④ RL rotate

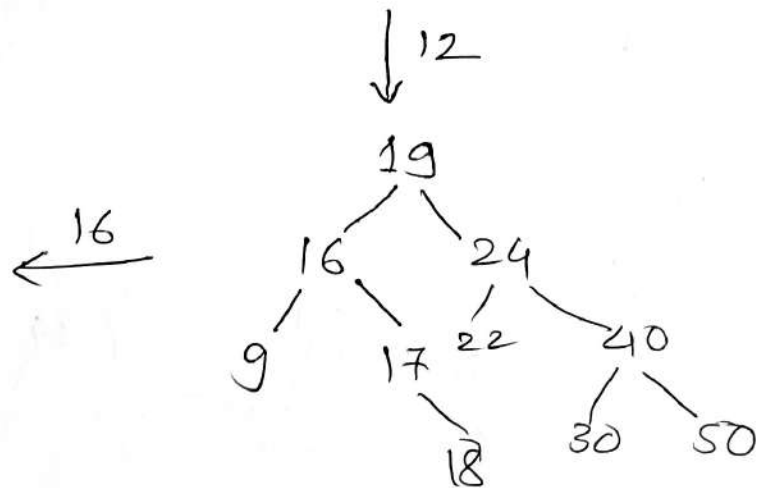
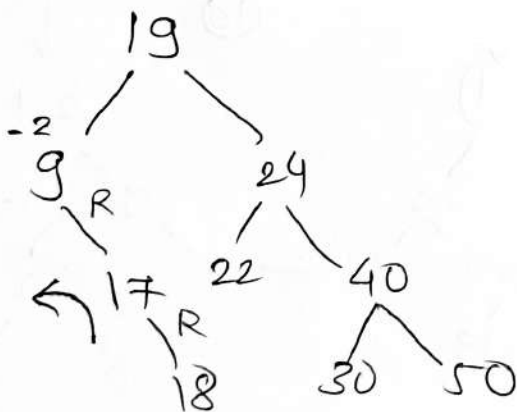
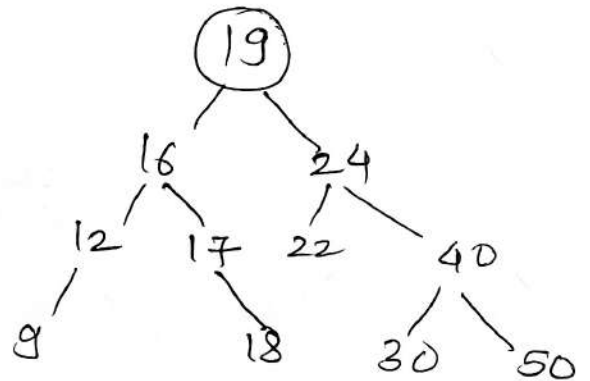
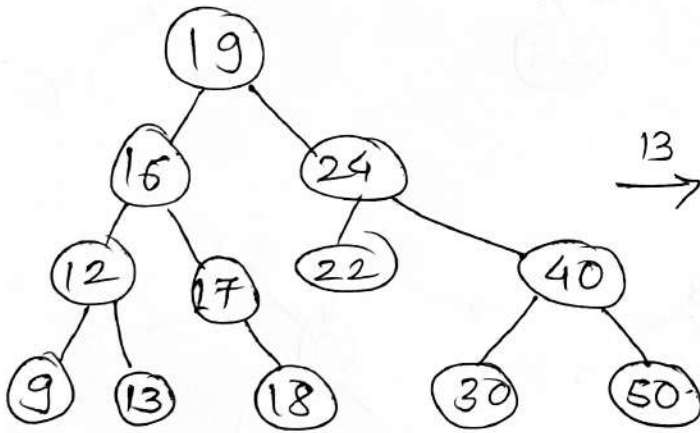


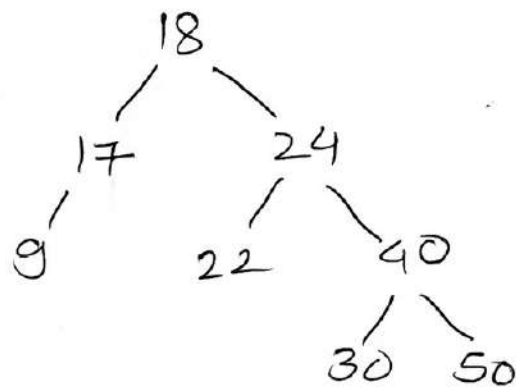
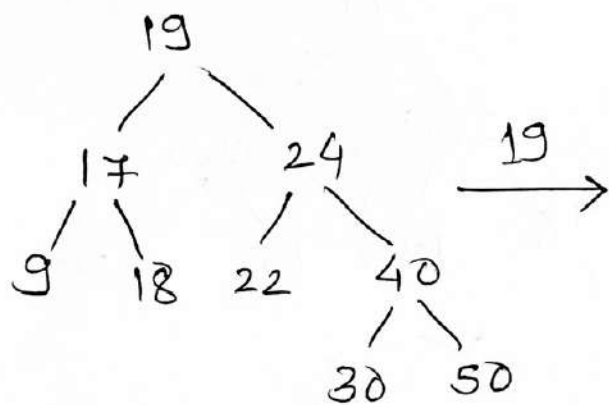
eg:- 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7





Deletion $\Rightarrow (13, 12, 16, 19)$





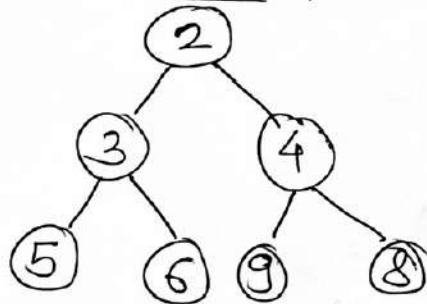
→ Heap

→ Complete Binary Tree

→ Min heap (Parent < child)

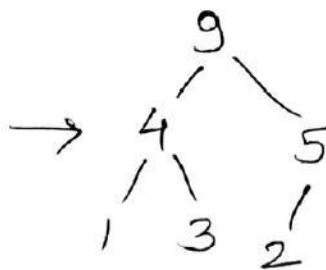
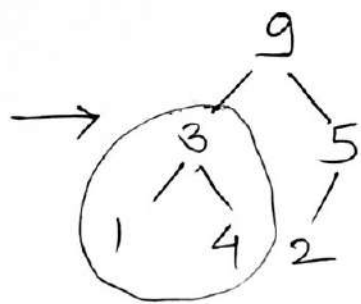
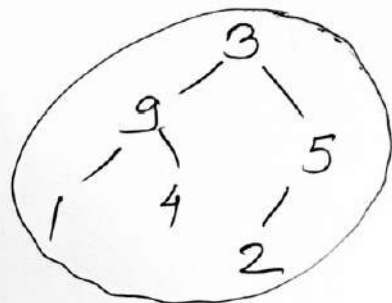
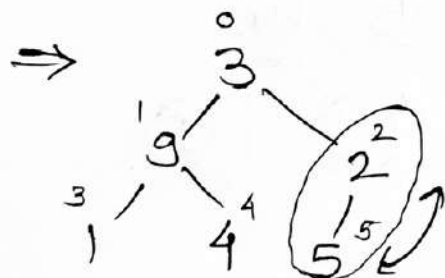
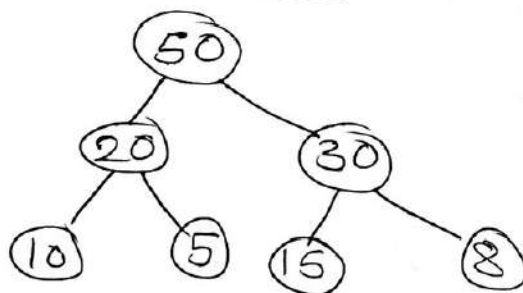
→ Max heap (Parent > child)

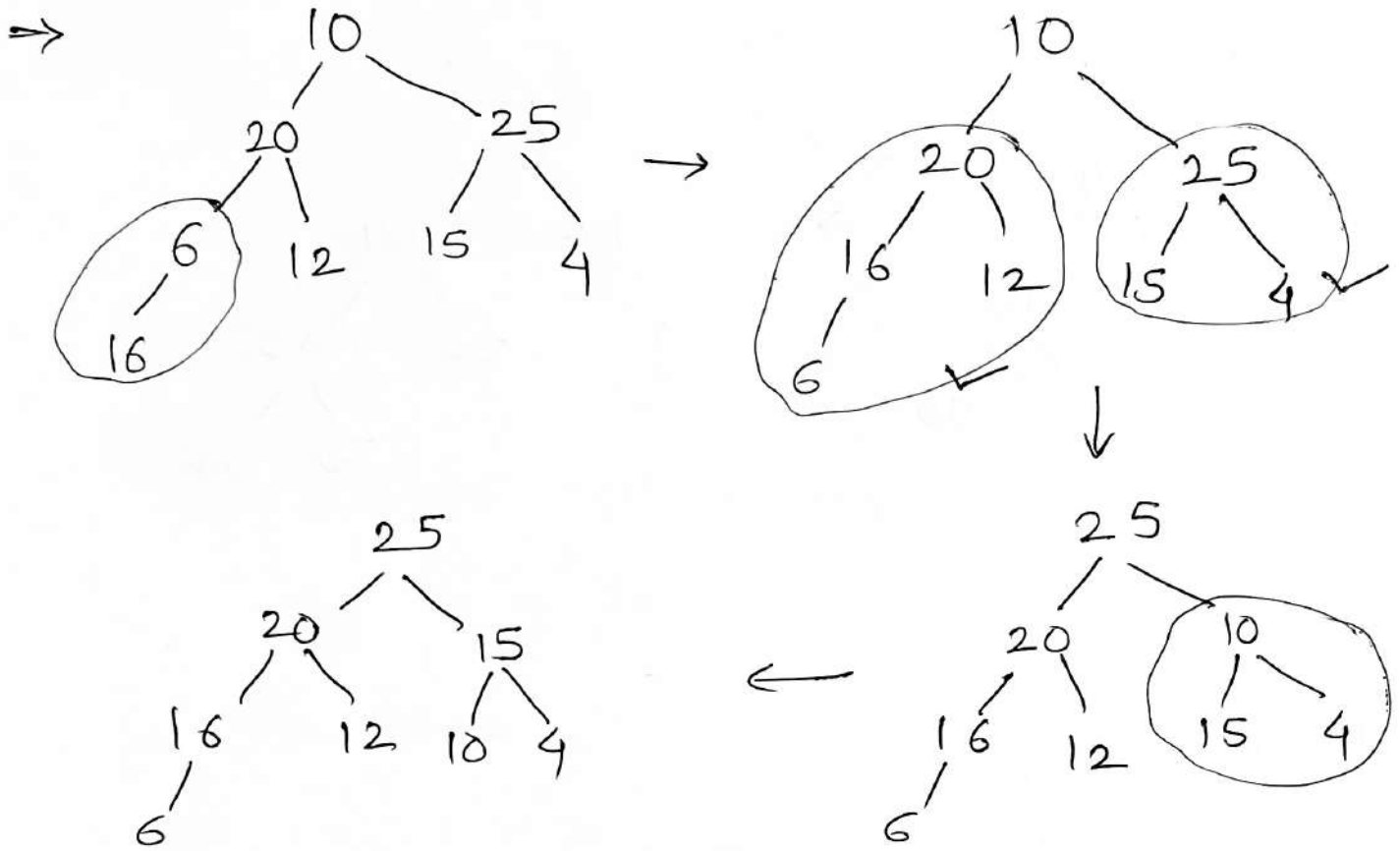
Min heap



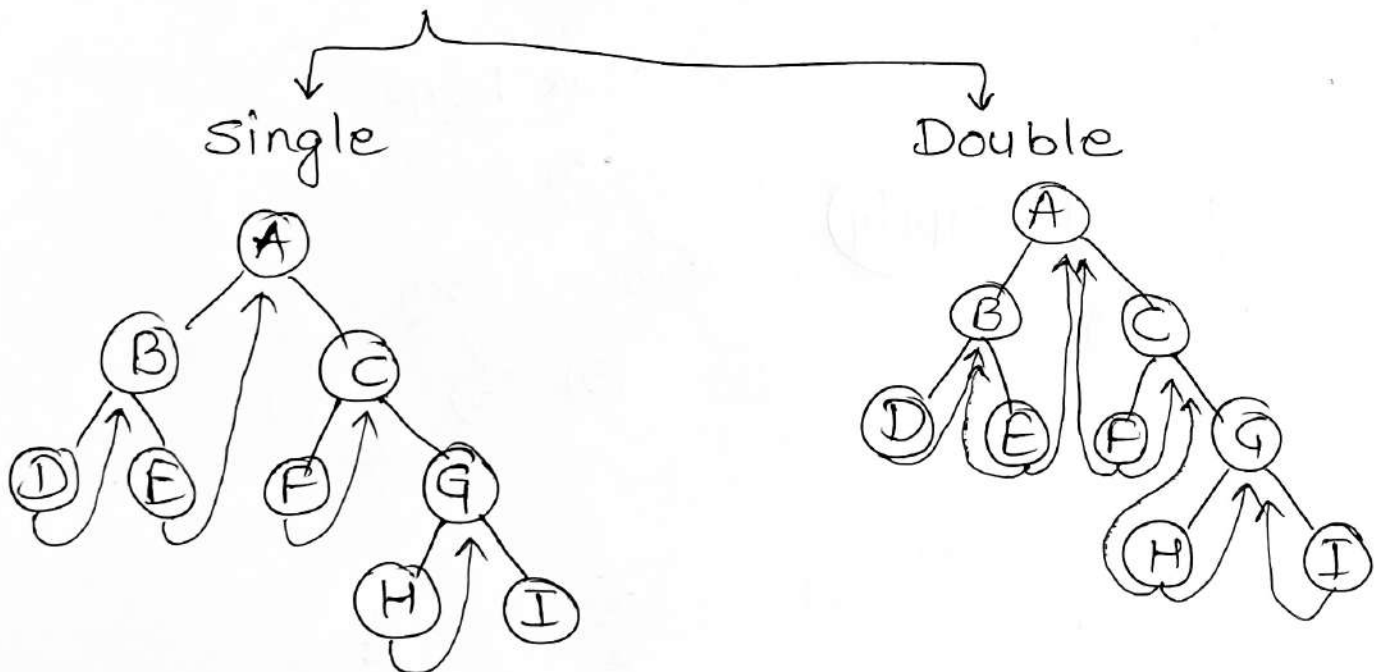
Heapify

Max heap





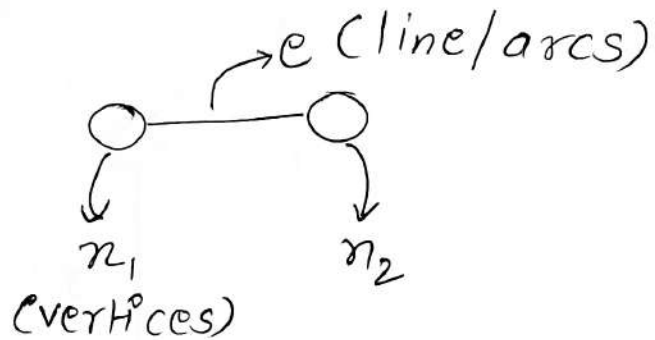
→ Threaded BT



Graph

→ NLDS

└ Nodes
└ Edge



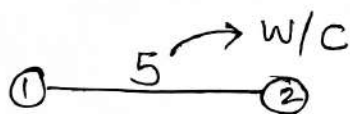
① Directed Graph



② Undirected Graph



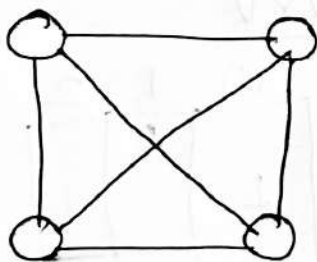
③ weighted Graph



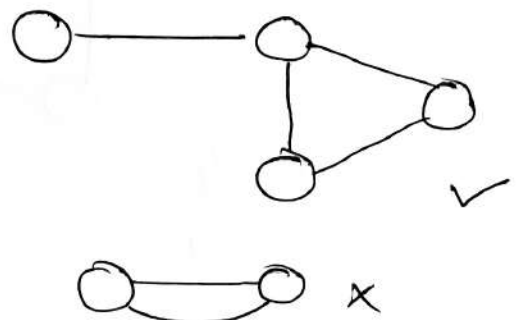
④ Unweighted Graph



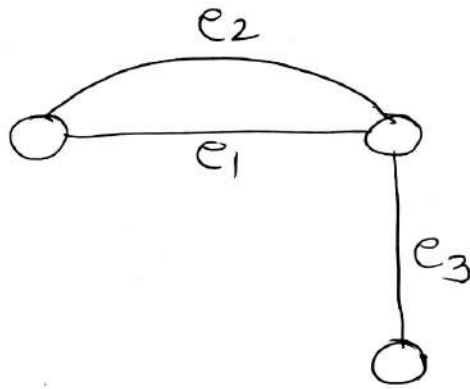
⑤ Complete Graph



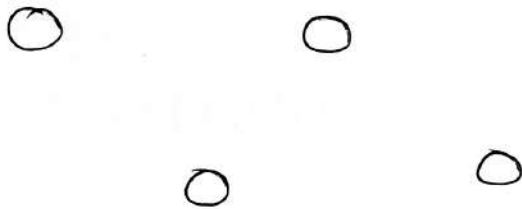
⑥ simple Graph



⑥ Multi Graph



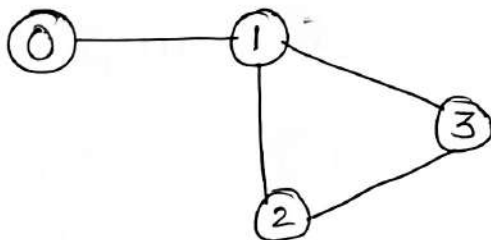
⑦ NULL Graph



⇒ Representation of graph:

- Adjacency Matrix
- Adjacency List

Eg:



$$V = 4$$

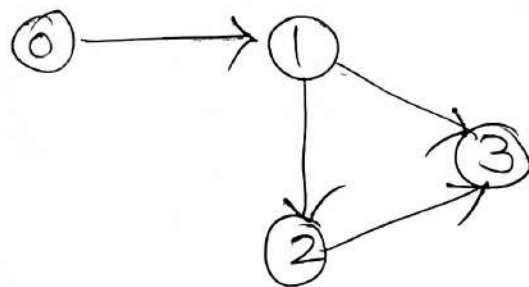
$$E = 4$$

$$1 \rightarrow \checkmark$$

$$0 \rightarrow \times$$

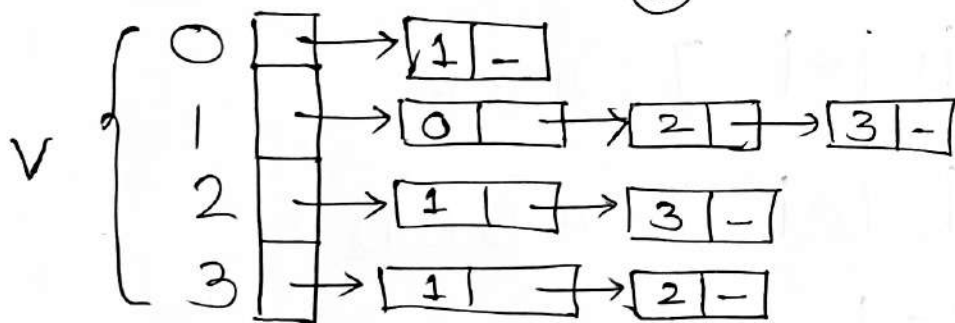
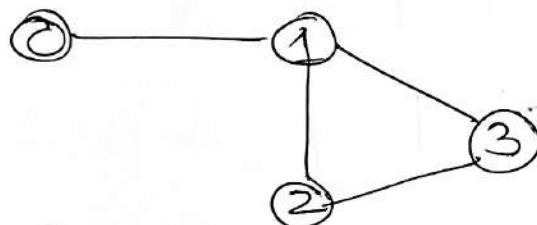
<u>VxV</u>					
		0	1	2	3
0	0		1		
1	0	1		1	1
2	0		1		
3	0		1	1	

Eg



	0	1	2	3
0		1		
1			1	1
2				1
3				1

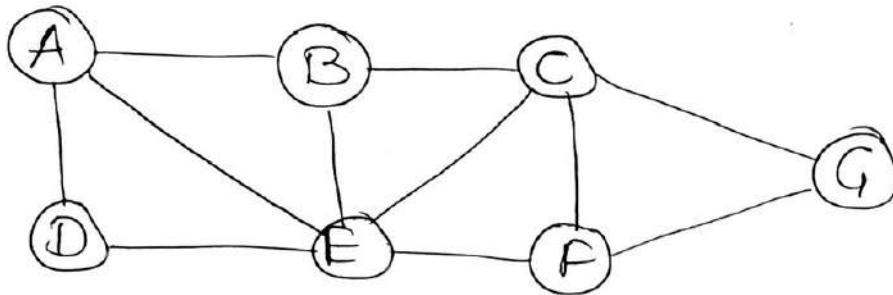
→ Adjacency List



→ Graph Traversal

→ BFS
→ DFS

BFS →

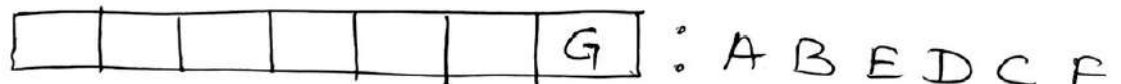
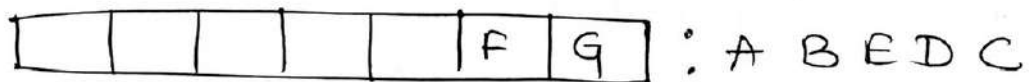
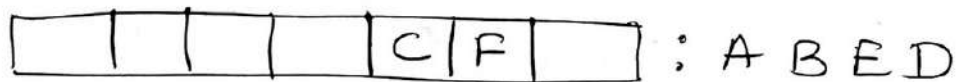
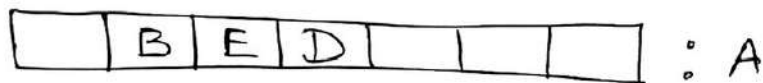


Start with (A)

⇒



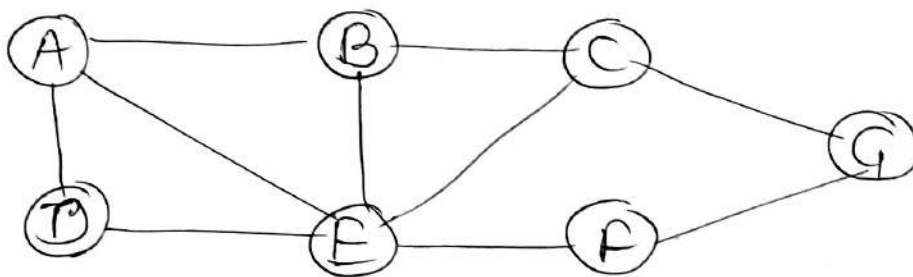
⇒



↓

A B E D C F G

DFS



$A \rightarrow B, D, E$

$B \rightarrow A, C, E$

$C \rightarrow B, E, F, G$

$D \rightarrow A, E$

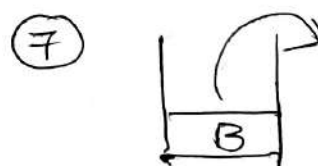
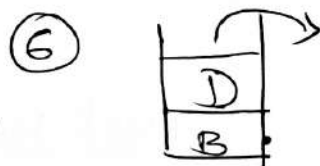
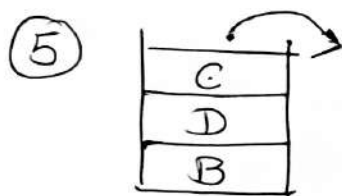
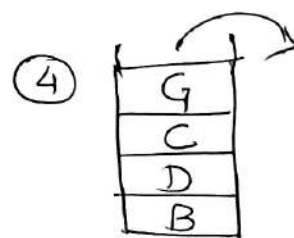
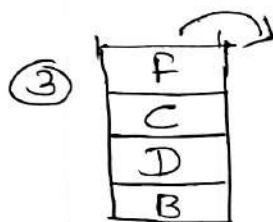
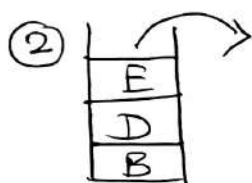
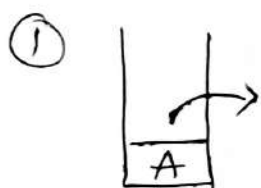
$E \rightarrow A, B, C, D, F$

$F \rightarrow C, E, G$

$G \rightarrow C, F$

" $\nwarrow \vee \searrow$
A B C D E F G"

Ans: A E F G C D B



→ Hashing

(Time ↓ Space ↑)

→ Key

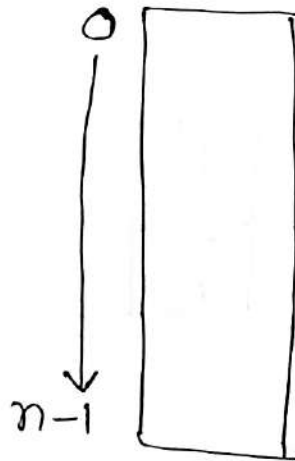
→ Location

→ Hash fn

→ Easy
→ Fast
→ low

→ Hash table.

eg: $f^n(H(k)) = k \bmod n$



$f^n(H(k)) = \underline{k \bmod n+1}$



keys $\rightarrow (10, 11, 12, 13, 14, 15, 16, 17, 18, 19)$

$$f^n: K \bmod n$$

$$n = 10$$

	$I \downarrow$
10	0
11	1
12	2
13	3
14	4
15	5
16	6
17	7
18	8
19	9
HT	

• Mid-Square

$\Rightarrow (12) \rightarrow \text{Square} \rightarrow 144$
 \downarrow
 $I=4$ (store 12)

$\Rightarrow (11) \rightarrow \text{Square} \rightarrow 121$
 \downarrow
 $I=2$ (store 11)

• $k = 123456$ (folding method)

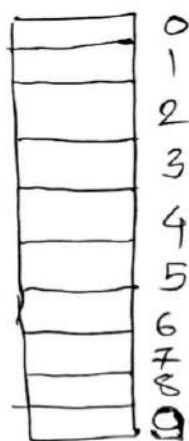
$\swarrow \quad \searrow$
123 + 456

\downarrow
 $\begin{array}{r} 123 \\ + 456 \\ \hline 579 \end{array} \bmod n$

$\therefore n = 10$, so $579 \bmod 10$

$\Rightarrow 9$ (store 123456)

Collision



(12, 42)

$$12 \bmod 10 = 2$$

$$42 \bmod 10 = 2$$

CRT

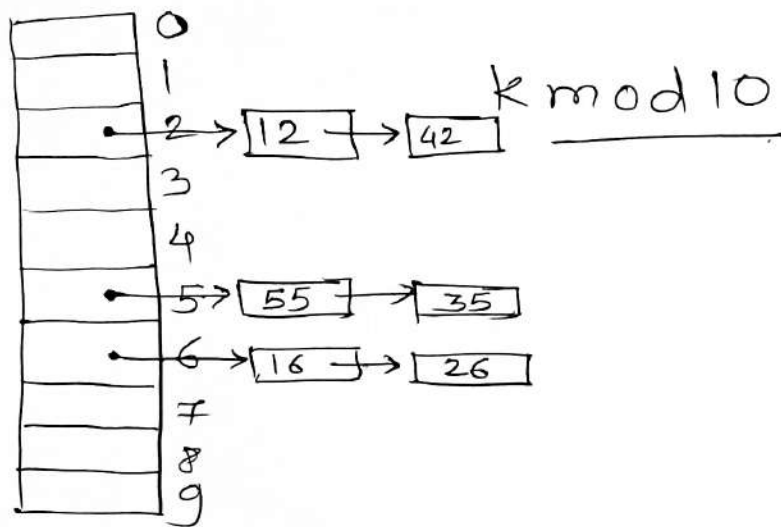
(Open H)
chaining

(Closed H)

- Linear Probing
- Quadratic Probing
- Double ~~Hash~~ Probing.

Chaining

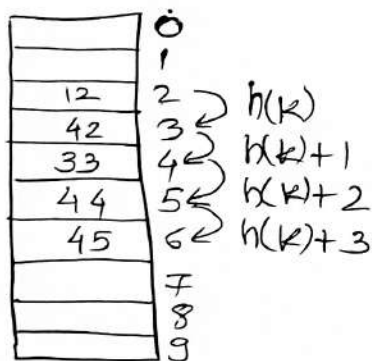
$$K \rightarrow (12, 42, 55, 35, 16, 26)$$



Linear Probing

$$k \bmod 10$$

$$K \rightarrow (12, 42, 33, 44, 45)$$
$$h(K) \rightarrow \begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 2 & 3 & 4 & 5 \end{matrix}$$



$$K = 42$$

$$h(K) = 2 \text{ (Collision)}$$

$$i = 1$$

$$h(K) + i = 2 + 1 = \underline{3}$$

Problems

→ Primary clustering

Search Time ↑

Quadratic Probing

$$h(k) = k \bmod 10$$

46	0
31	1
22	2
43	3
	4
	5
26	6
37	7
28	8
	9

$$k \rightarrow 22, 26, 31, 43, 28, 37, 46, 52$$

$$\Rightarrow (h(k) + i^2) \bmod 10$$

$$46 \bmod 10 = 6 \text{ [Collision]}$$

$$i = 1$$

$$(6 + (1)^2) \bmod 10 = 7$$

$$i = 2$$

$$(6 + (2)^2) \bmod 10 = 0$$

$$k_1 \quad k_2 \quad k_3$$

$$2 \quad 2 \quad 2$$

$$3 \quad 3 \quad 3$$

$$6 \quad 6 \quad 6$$

$$1 \quad 1 \quad 1$$

$$8 \quad 8 \quad 8$$

$$7 \quad 7 \quad 7$$

(Secondary clustering)

for 52

$$52 \bmod 10 = 2 \text{ [Collision]}$$

$$i = 1$$

$$(2 + (1)^2) \bmod 10 = 3$$

$$i = 2$$

$$(2 + (2)^2) \bmod 10 = 6$$

$$i = 3$$

$$(2 + (3)^2) \bmod 10 = 1$$

$$i = 4$$

$$(2 + (4)^2) \bmod 10 = 8$$

$$i = 5 \quad (2 + (5)^2) \bmod 10 = 7$$

$$i = 6 \quad (2 + (6)^2) \bmod 10 = 8$$

$$i = 7 \quad (2 + (7)^2) \bmod 10 = 1$$

$$i = 8 \quad (2 + (8)^2) \bmod 10 = 6$$

$$i = 9 \quad (2 + (9)^2) \bmod 10 = 3$$

Double Hashing

$$(h_1(k) + i h_2(k)) \% n$$

$$h_1(k) = k \bmod 13$$

$$h_2(k) = 1 + (k \bmod 11)$$

	0
79	1
	2
	3
69	4
	5
	6
98	7
72	8
	9
	10
	11
	12

$$k \rightarrow 79, 69, 98, 72$$

$$h_1(79) = 79 \% 13 = 1$$

$$h_1(69) = 69 \% 13 = 4$$

$$h_1(98) = 98 \% 13 = 7$$

$$h_1(72) = 72 \% 13 = 7 \text{ [Collision]}$$

$$\Rightarrow i = 1$$
$$[7 + 1 * (1 + 72 \% 11)] \% 13$$

$$\Rightarrow i = 2 \quad \rightarrow 1 \text{ [Collision]}$$

$$[7 + 2 * (1 + 72 \% 11)] \% 13$$
$$\rightarrow 8 \checkmark$$