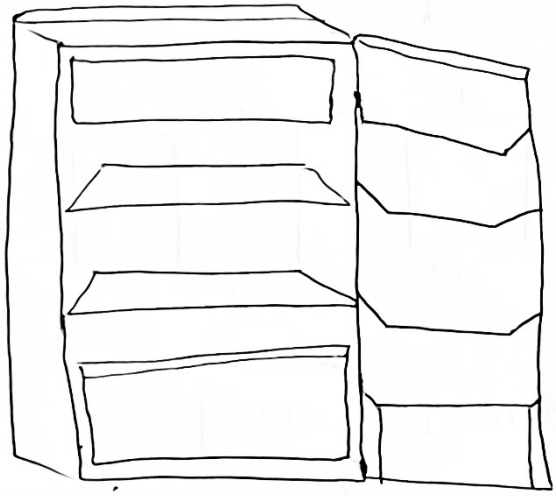
→ Introduction
→ Array
→ Linked List
→ Stack
→ Queue
→ Tree
→ Heap
→ Graph
→ Hashing

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

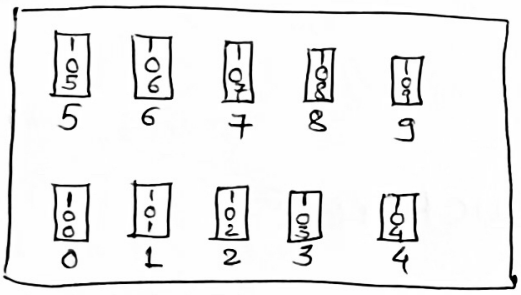
③ Array

→ store

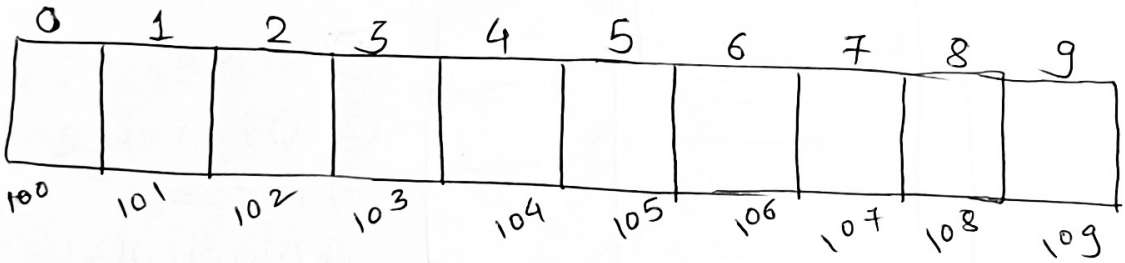
→ Same type

→ Contiguous Memory Locations

→ Index Accessing



CLASS (A)



→ Datatype ArrayName [A_size]

↓ int ↓ Arr ↓ [10]

→ Random Accessing

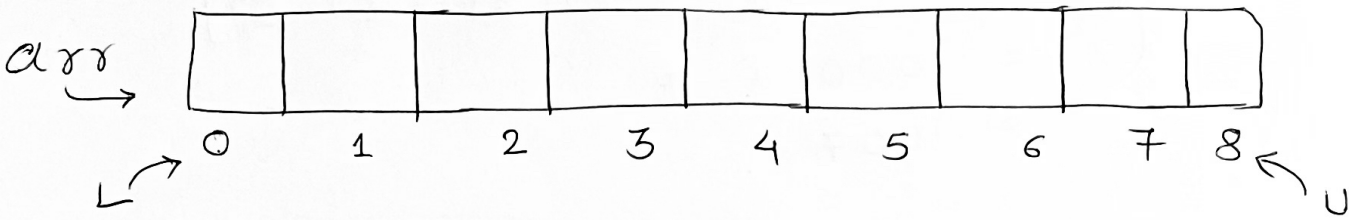
→ Easy Retrieval

→ Easy to Use & understand

→ Fixed nature

→ Insert / Delete Issues

→ 1D Array



$$Size(arr) = U - L + 1 = 8 - 0 + 1 = 9$$

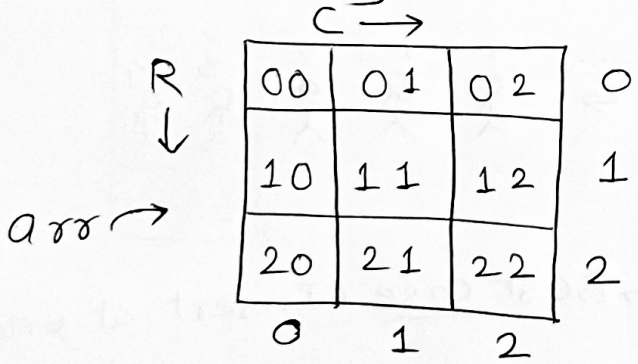
→ BA = 1000
U = 8
L = 0
S = 1

$$Addr(arr[i]) = BA + S[i - L]$$

$i = 0$ $= 1000 + 1[0 - 0]$
$= 1000$

$i = 1$ $= 1000 + 1[1 - 0]$
$= 1001$

→ 2D Array



BA = 1000
S = 1
$L_R = 0$
$U_R = 2$
$L_C = 0$
$U_C = 2$

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

$i = 1$, $j = 2$ $= 1000 + 1[(1-0)(3) + (2-0)]$
$= 1000 + 5$
$= 1005$

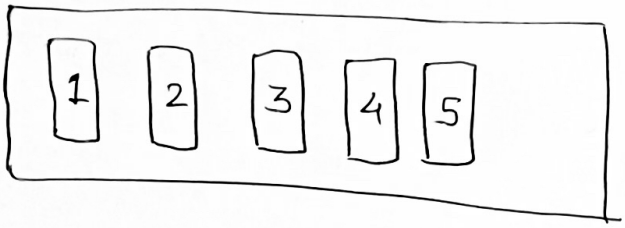
$$Addr(arr[i][j]) = BA + S[(j - L_C)(U_R - L_R + 1) + (i - L_R)]$$

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

$$= 1000 + 7$$

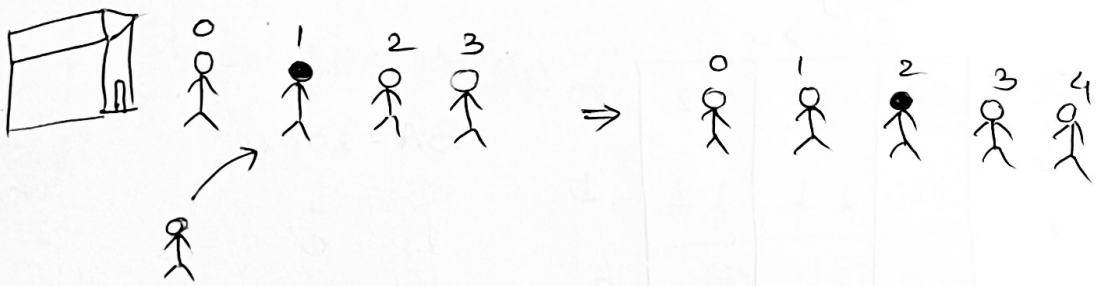
$$= 1007$$

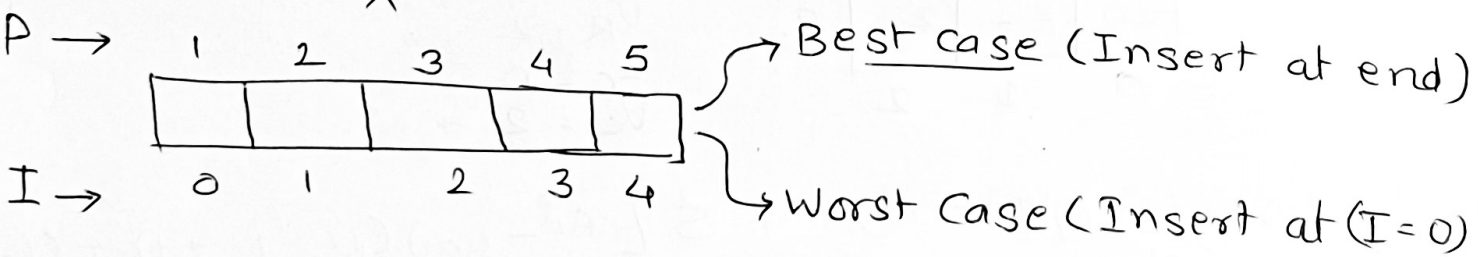
→ <u>Traversal</u> operation :-




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

→ <u>Insertion</u> operation :-



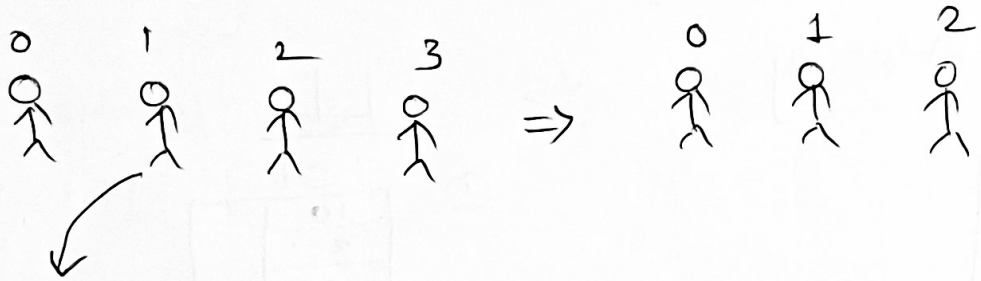


$P = I + 1$

or

$I = P - 1$

→ Deletion Operation



10	20	30	40	50
0	1	2	3	4

↓ delete (I=1)

10		30	40	50

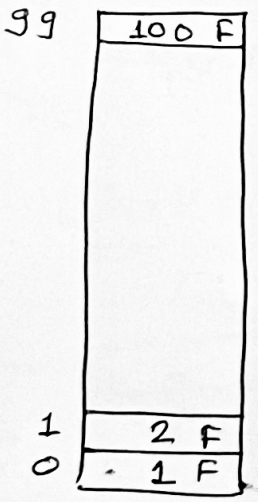
↓

10	30	40	50	

→ Best case (Last element)

→ Worst case (I=0)

→ Searching operation



→ Best case (1st floor)

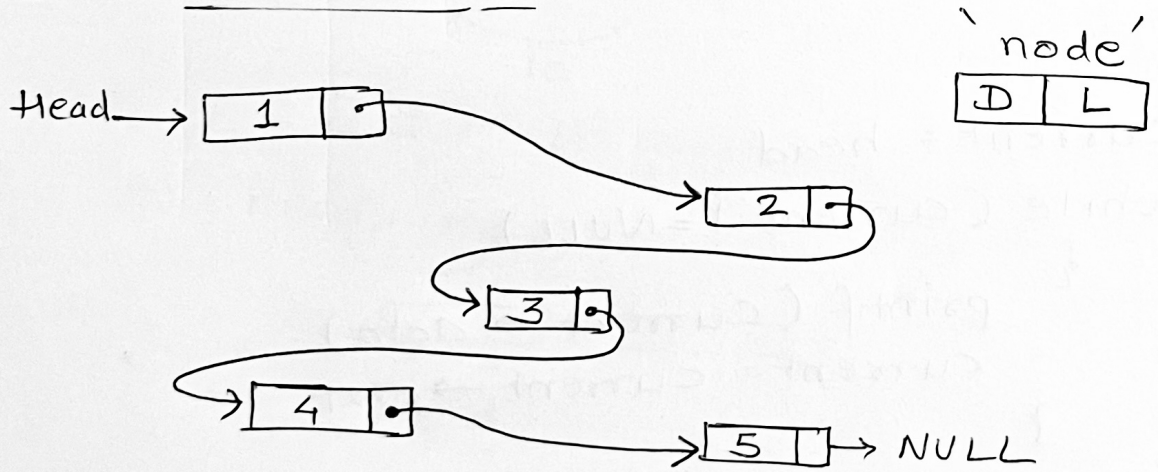
→ Worst case (100th floor)

Array Vs LL

	Array	LL
①	Contiguous ML	① Not
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

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

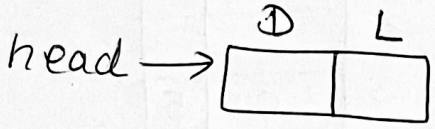
④ Linked List :




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

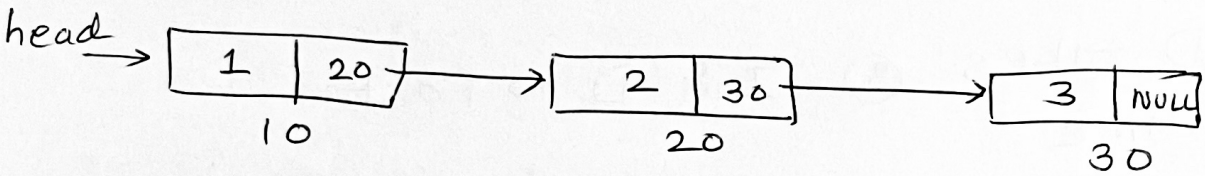
node * head = malloc (sizeof (node))

↓



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

<u>Traversal</u>

⇒ current = head

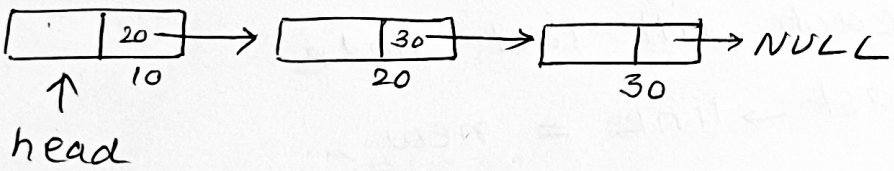
```
while (current != NULL)
{
    printf (current → data)
    current = current → next
}
```

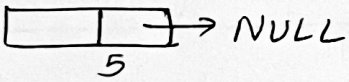
Searching

⇒ current = head

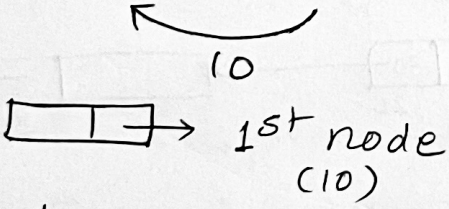
```
while (current != NULL)
{
    if (current -> data == key)
    {
        return current
    }
    current = current -> link
}
```

Insert → At Beginning.

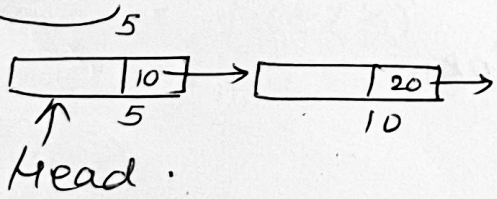


⇒ 

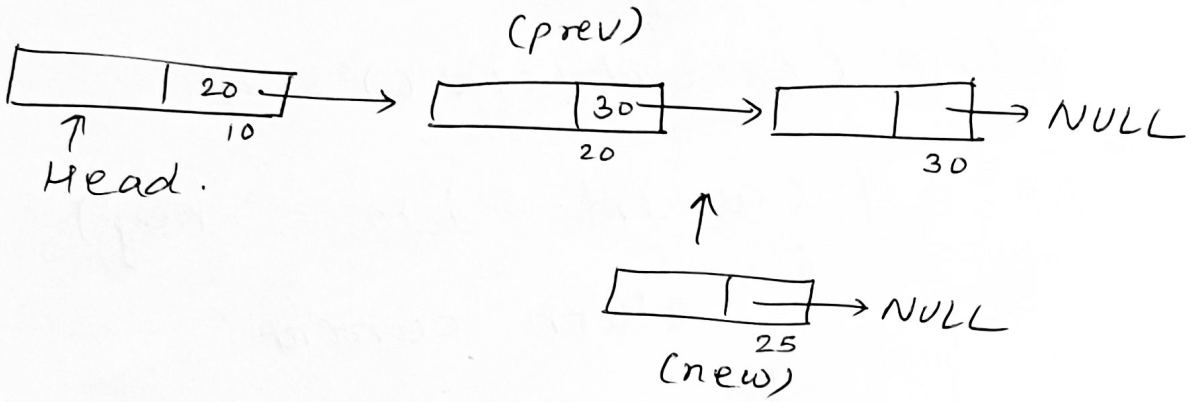
① new → next = head



② head = new.



Insert → after a node.



① new → next = prev → next

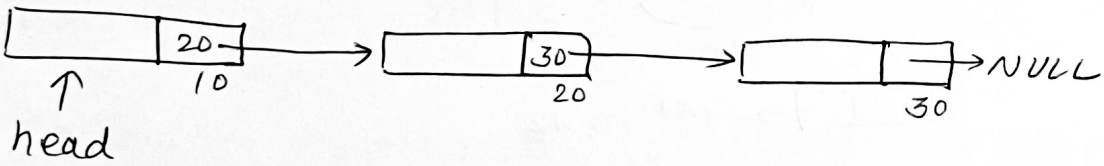
② prev → next = new.

Insert → at End

① Reach till last node

② last → = new

Delete → At Beginning.

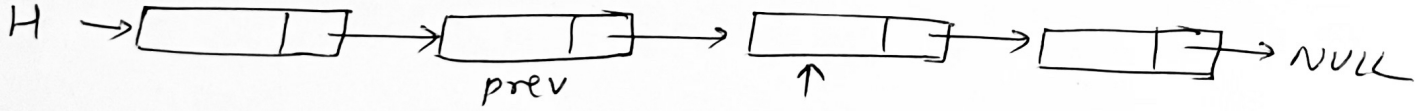


① del = head

② head = del → link

③ free (del)

Delete → after a node.



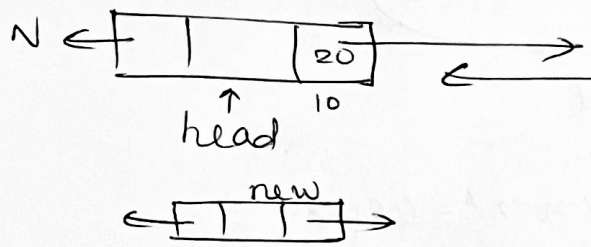
① del = prev → link

② prev → link = del → link

③ free (del)

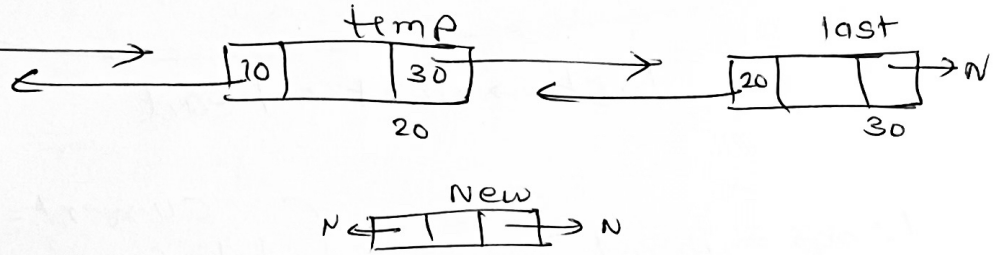
Insert

At Beginning



- new→next = head
- head →prev = new
- head = new

After a node



- new→next = temp→next
- temp→next→prev = new
- temp→next = new
- new→prev = temp

At End

- last→next = new
- new→next = last

Delete

At Beginning

- temp = head
- head = head→next
- head→prev = NULL
- free (temp)

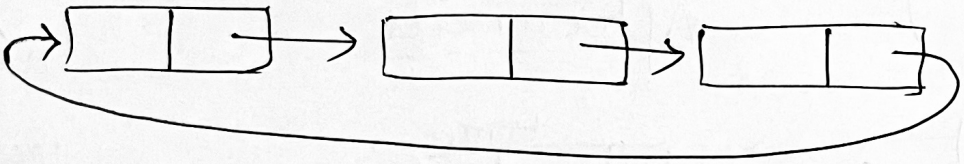
After a node

- p→next = p→prev→next
- p→next→prev = p→prev
- free(p)

At End.

- temp = p→next
- p→next = NULL
- free (temp.

→ Circular Linked List



last → next = head

② temp = head

③ head = head→next

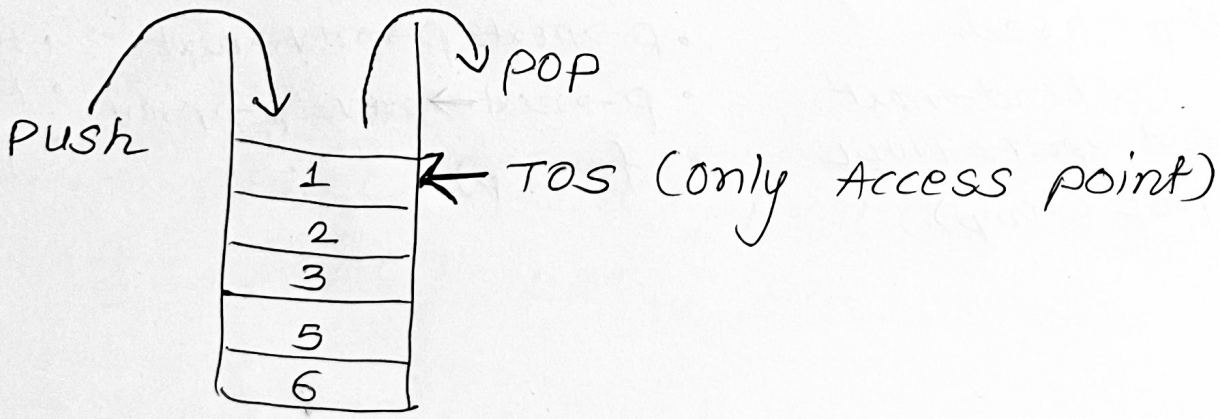
⑤ free(temp)

current = head
while (current→next != head)
{ current = current→next
}
①

④ current→next = head

⑤ Stack

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



→ Static Imp: Arrays
→ Dynamic Imp: Linked List

} I/D from "Only One End"

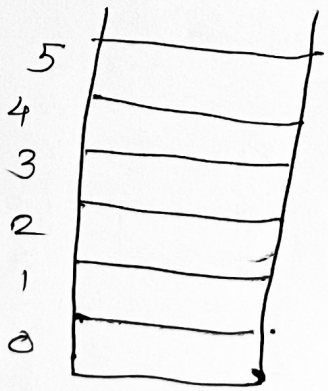
→ Push (arr, n, TOS, x)

```
{
    If (TOS == n-1)
      { printf (" stack is full")
      }
    TOS = TOS+1
    arr [TOS] = x
}
```

→ pop (arr, n, TOS)

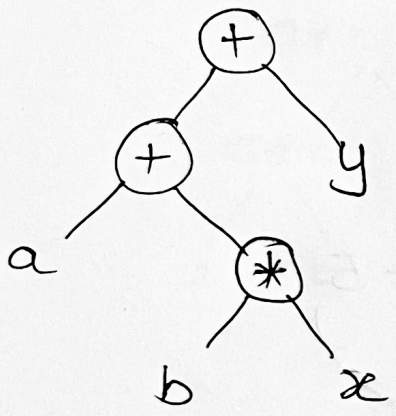
```
{
    If (TOS == -1)
      { printf (" stack is empty")
      }
    x = arr [TOS]
    TOS = TOS -1
    return (x)
}
```

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



Infix	Prefix	Postfix
a+b * x+y	++ a * b x y	a b x * + y +
(l-R-r)	(R-l-r)	(l-r-R)

(Tree)



	Stack	Postfix
a		a
+	+	a
b	+	ab
*	+*	ab
x	+*	abx
+	+	abx*+
y	+	abx*+y
		↓
		abx*+y+

a + b * x +y

↓ ↓ ↓

a+ bx * +y

↓ ↓

abx *+ +y

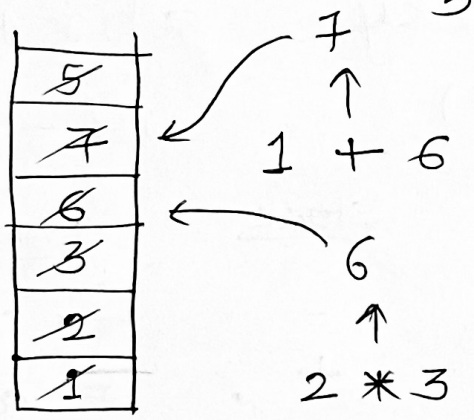
↓

→ abx*+y+ ←

↓ ^ (R→L)
* / (L→R)
+ - (L→R)

$\Rightarrow$ 1 2 3 * + 5 + (abx * + y +)

$5 + 7 = \underline{12}$



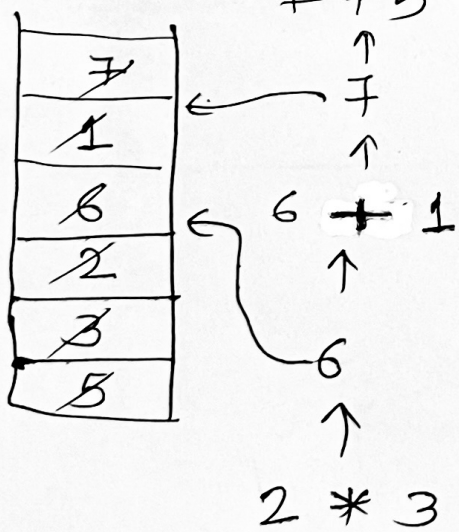
$a+b * x + y = 1 + \underbrace{2 * 3}_{6} + 5$

$1 + 6 \rightarrow 7$

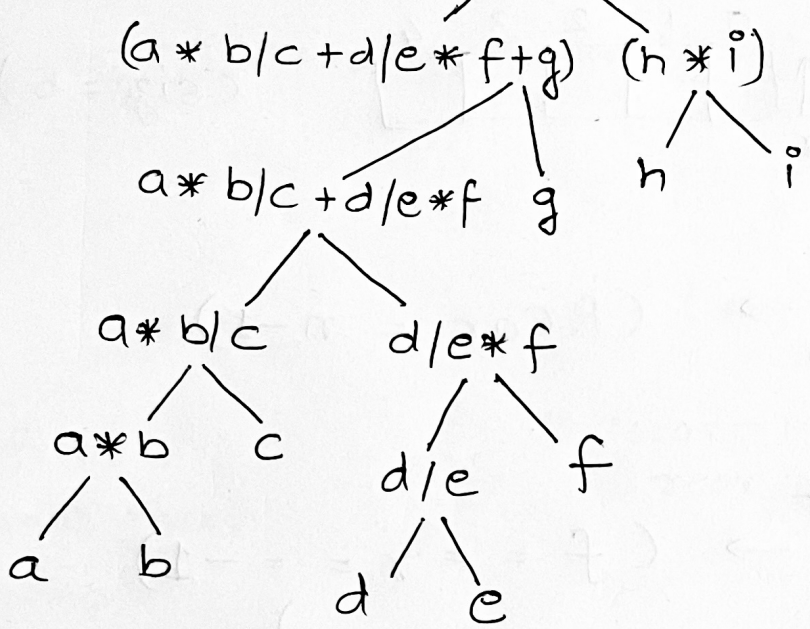
$7 + 5 \rightarrow \underline{12}$

++a * bxy = ++1 * 235

$7 + 5 = \underline{12}$

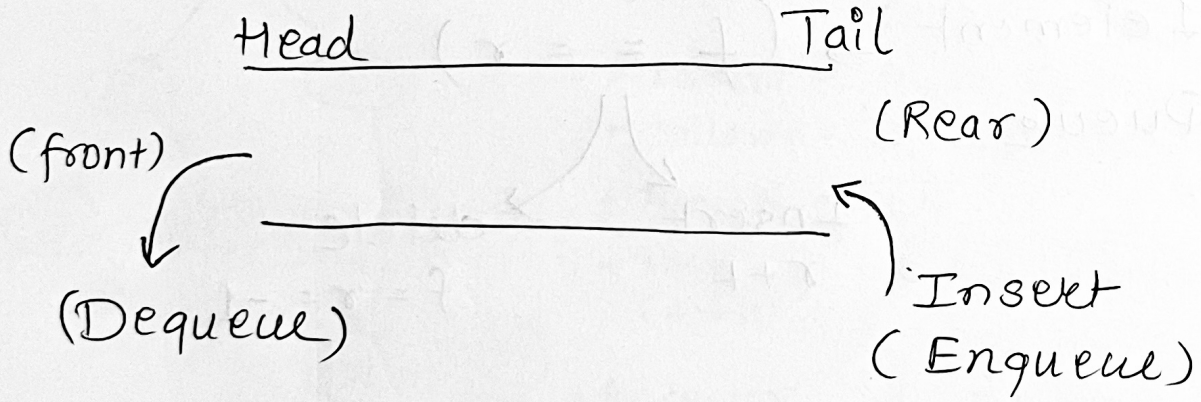


⇒ a * b/c + d/e * f + g − h * i



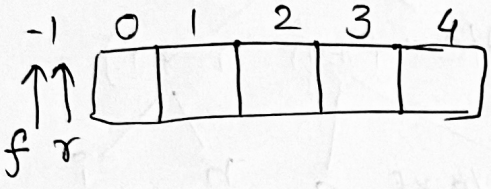
⇒ Queue

→ FIFO or LILO



Initially → front = rear = −1

→ Implementation using Array.

Initially → 

(size = 5)

Overflow Condition → (Rear = n−1)

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

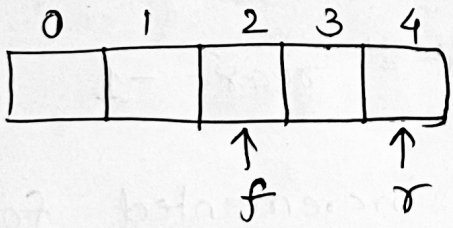
f = r = 0 (after insert)

Underflow Condition → (f = r = −1)

Only 1 element In Queue → (f == r)

Insert r++

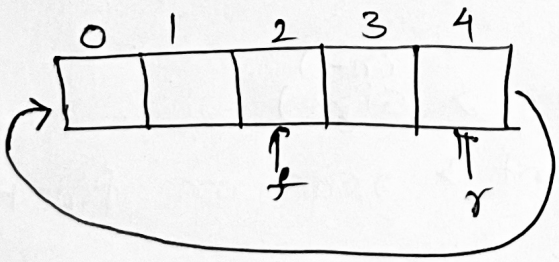
delete f = r = −1



Now insert won't be possible due to overflow. So circular Queue.

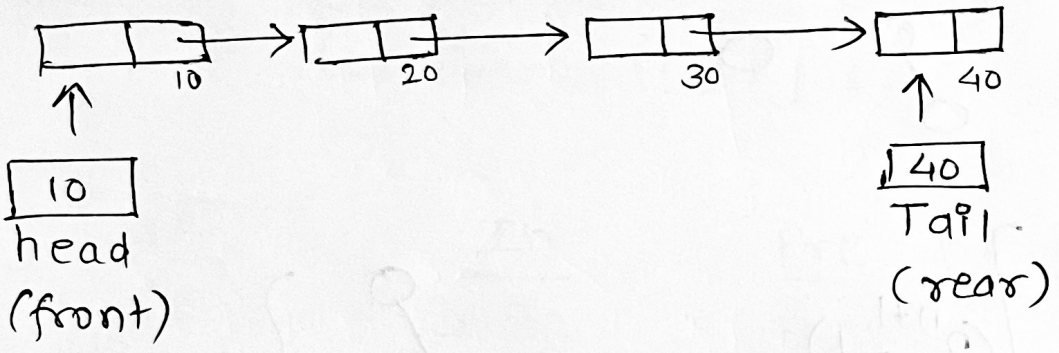
P++

(P+1) % n



here P → Rear
→ front

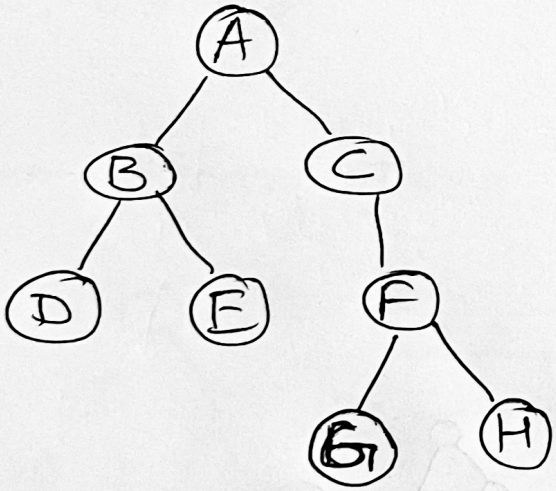
→ Implementation using LL



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

rear → next = new
rear = new

⇒ Tree



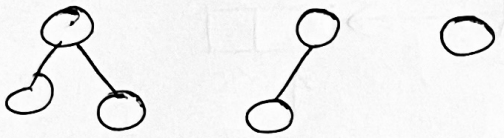
- root = A
- leaf = D, E, G, H
- Parent = A, B, C, F
- Children = B, C, D, E, E, G, H
- Non-LN = $\overline{LN}$
- Path (A → H) = A → C → F → H
- Ancestor (Before node) (A) & Decendant (After node) (F, H)
- Sibling → D, E ✓
  B, C ✓
  E, F ✗
- Degree of node → A = 2
  B = 2
  D = 0

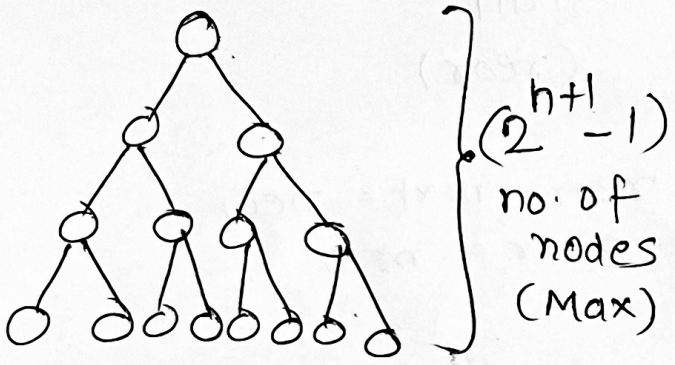
- Depth: Root → node
  (A) = 0 (D) = 2
  (B) = 1 (E) = 2
  (G) = 3 (H) = 3

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

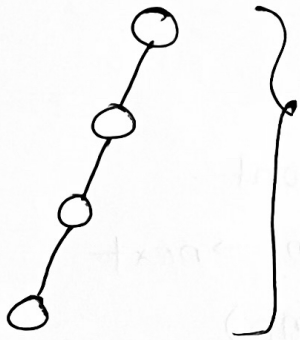
→ Binary Tree

↳ can have at most 2 children





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

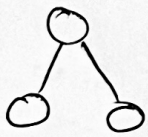


$(h+1)$ no. of nodes (min)

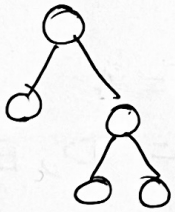
for $h=3$, $2^{3+1}-1 = \underline{15}$

for $h=3$, $3+1 = \underline{4}$

① full <u>BT</u> (0 or 2)

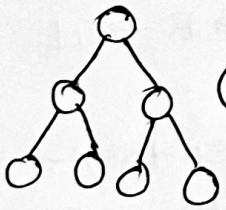


✓

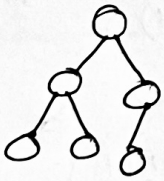


✗

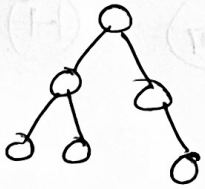
② Complete <u>BT</u>



✓

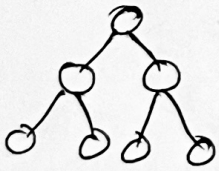


✓

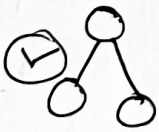


✗

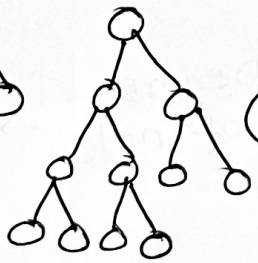
③ Perfect <u>BT</u>



✓



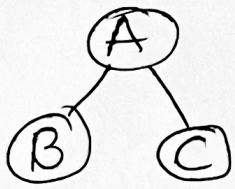
✓



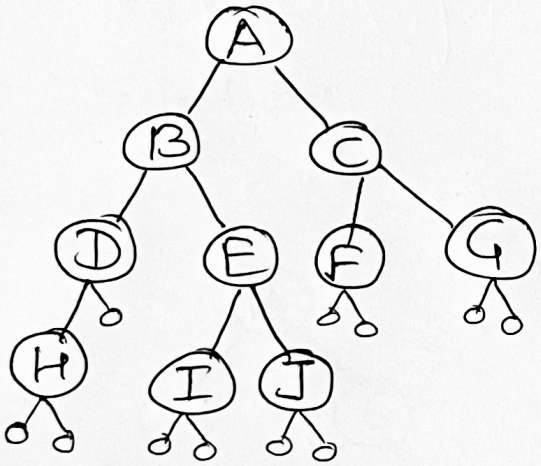
✗

⇒ Tree Traversal

- → preorder (R → l → r)
- → Inorder (l → R → r)
- → Post order (l → r → R)

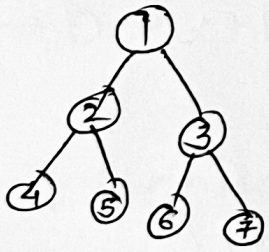


pre	In	Post
ABC	BAC	BCA



①	②	③
In	Pre	Post

→ Binary Tree (using Arrays)



1	2	3	4	5	6	7

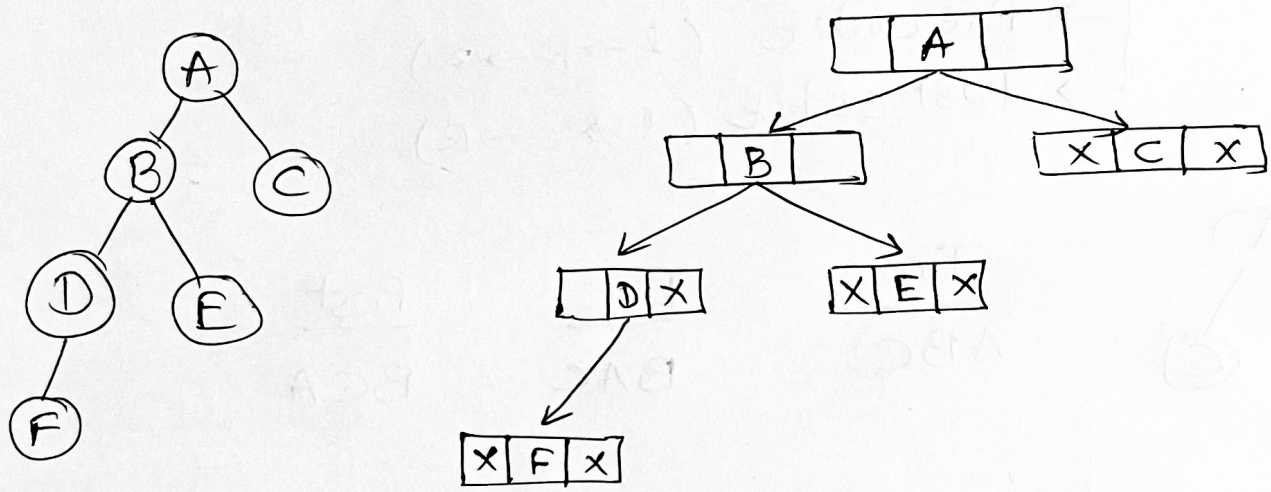
Root at → 1

Left child → 2 * i

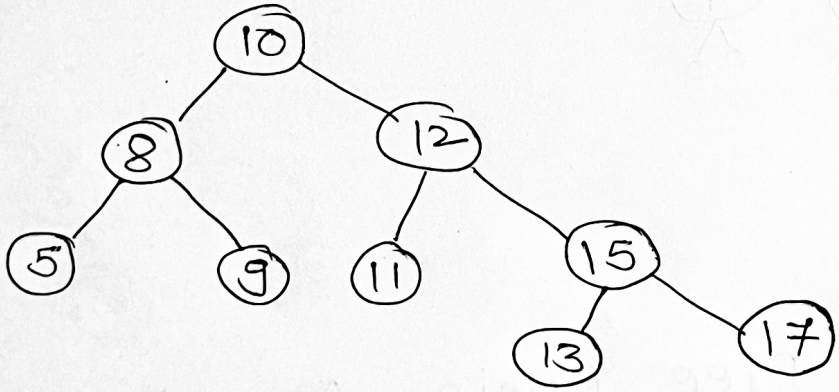
Right child → 2*i + 1

for eg:- i = 3 → L = 2x3 = 6
→ R = 2x3 + 1 = 7

→ Binary Tree (LL)

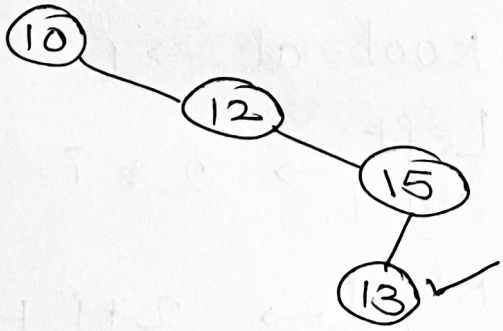


⇒ <u>BST</u>

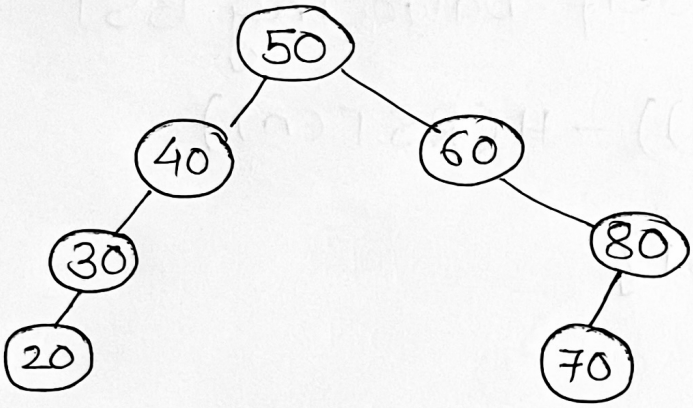


Inorder ⇒ 5, 8, 9, 10, 11, 12, 13, 15, 17

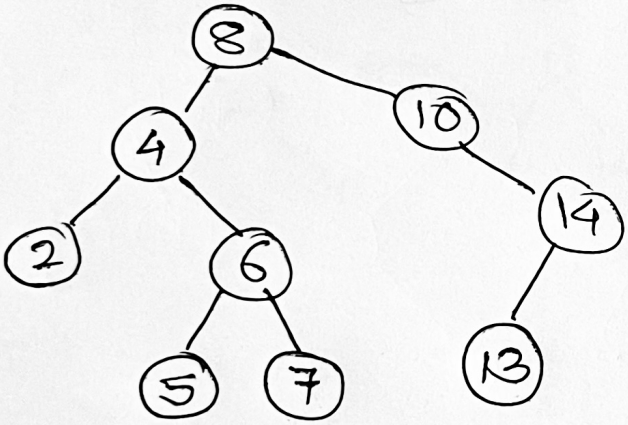
- Searching becomes easy (<u>13</u>)



- 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).

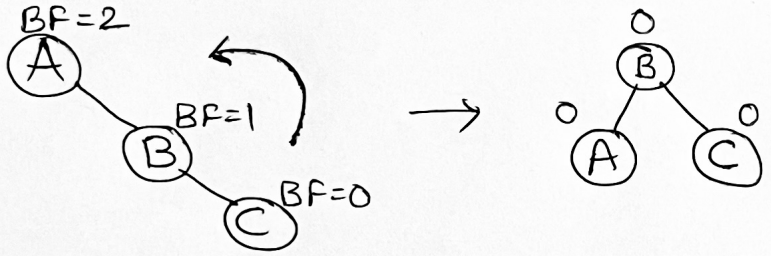
→ AVL Tree

↳ Its a self-balancing BST

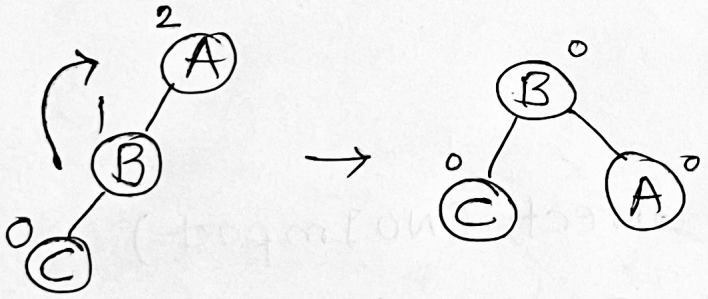
$Bf = H(LST(n)) - H(RST(n))$

↳ $\{-1, 0, +1\}$

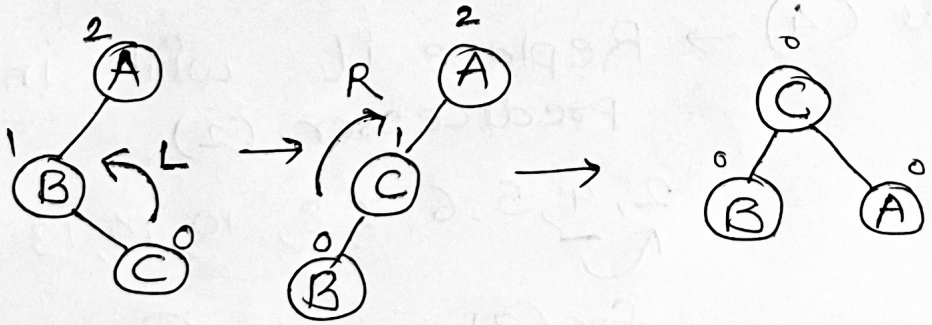
① Left Rotate



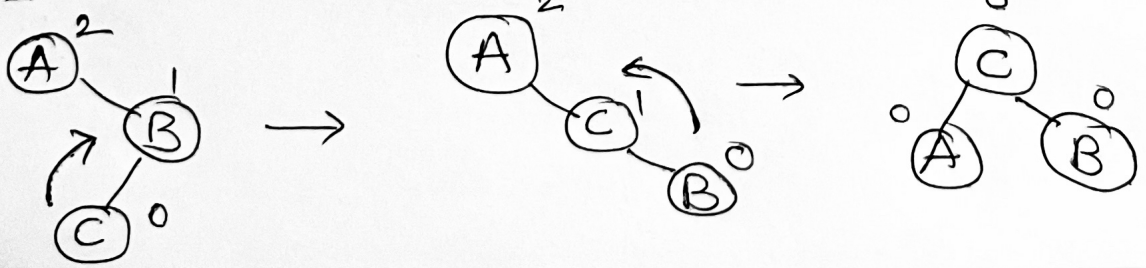
② Right Rotate



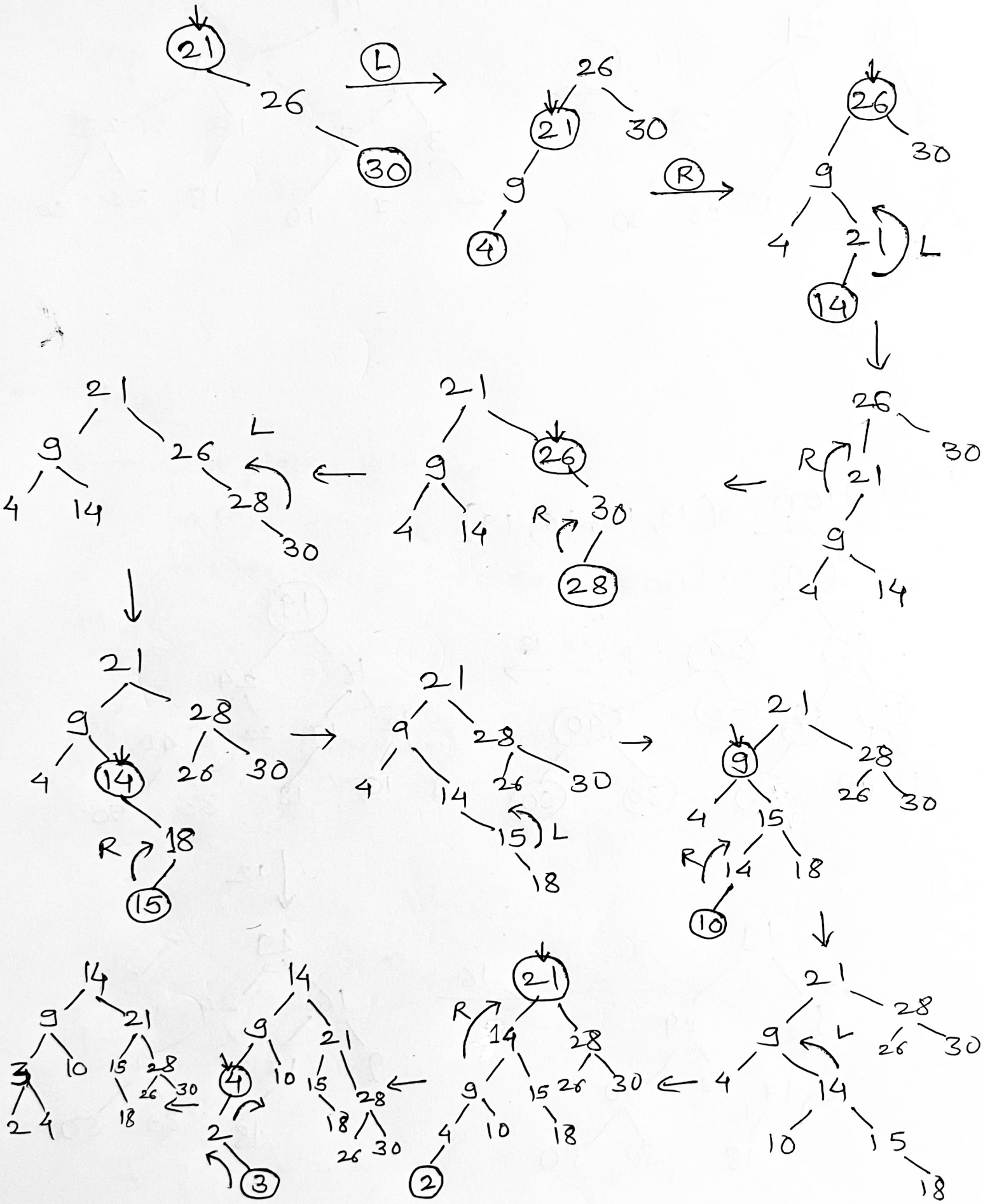
③ LR rotate

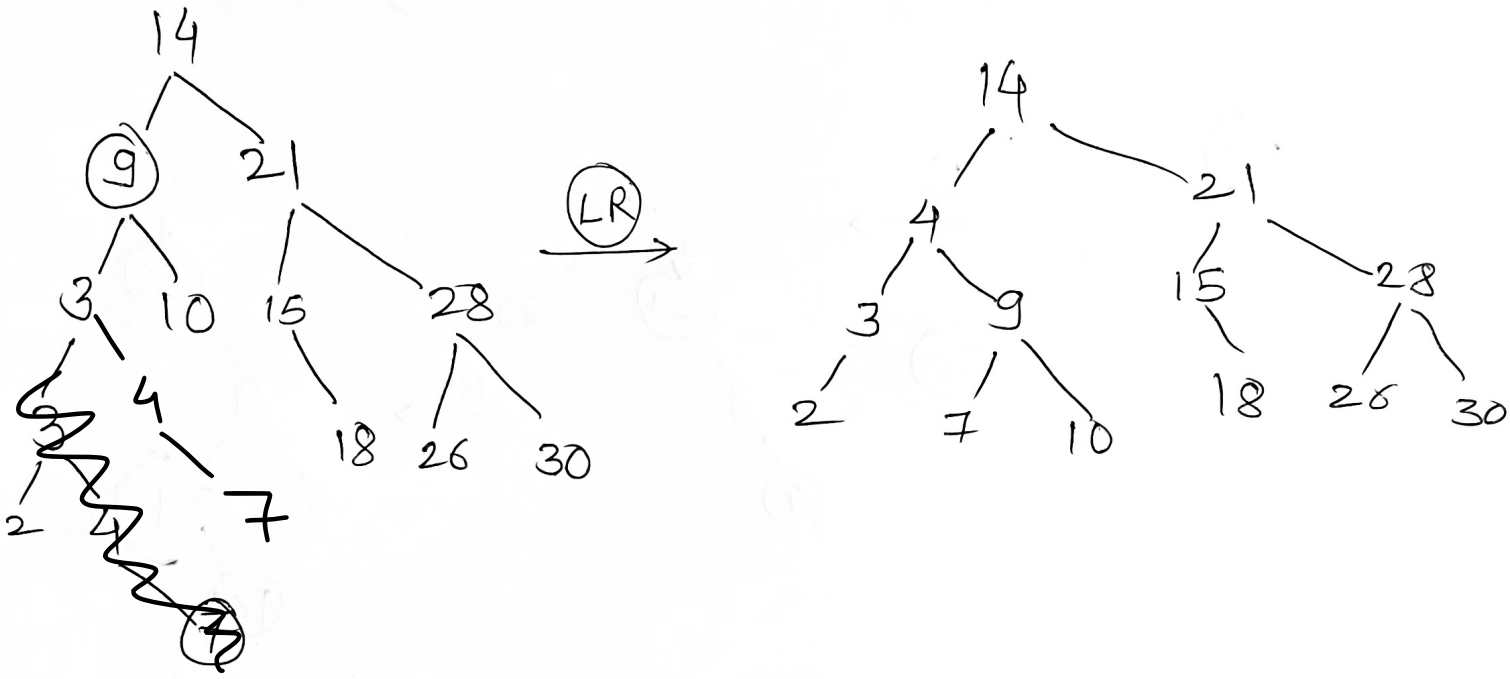


④ RL rotate

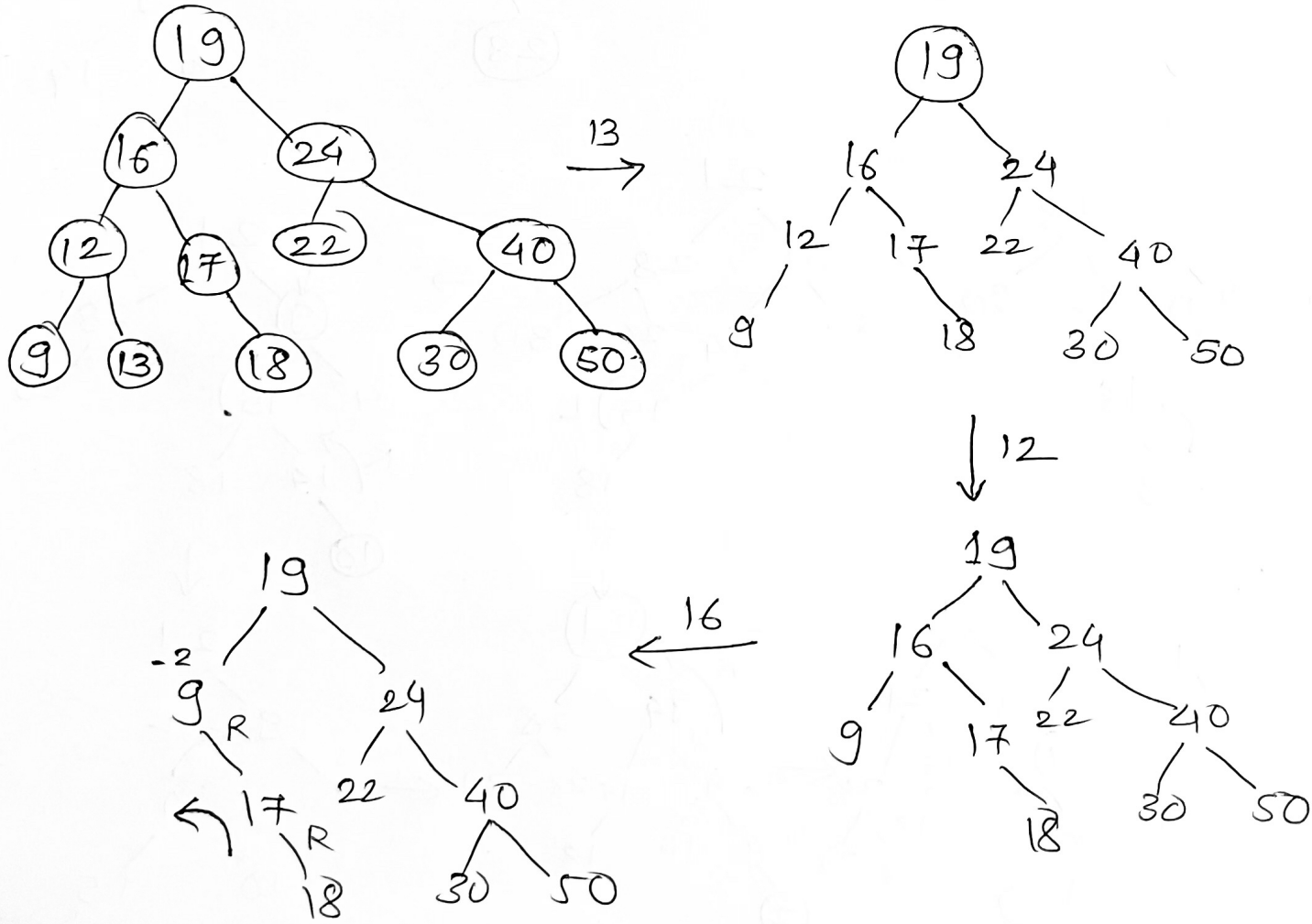


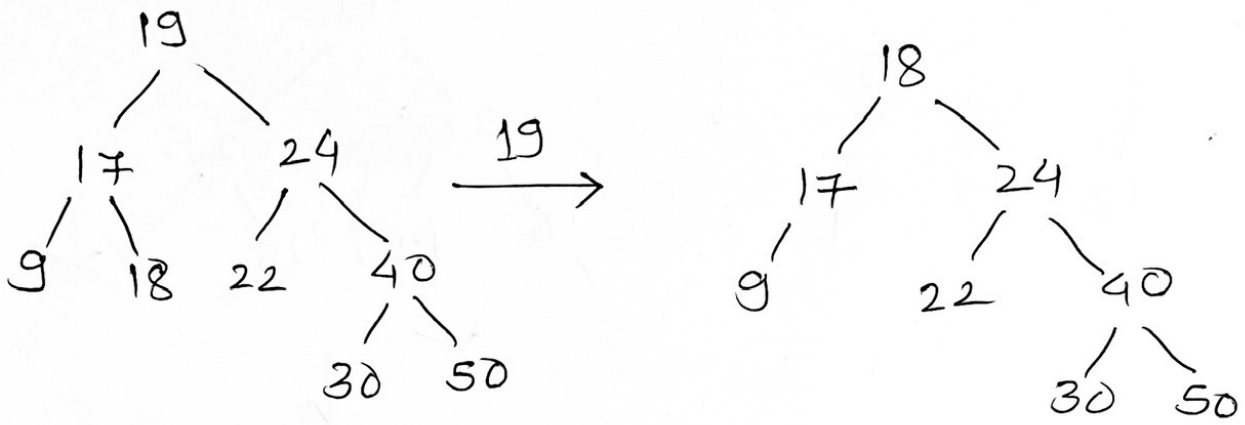
eg:- 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7





Deletion ⇒ (13, 12, 16, 19)

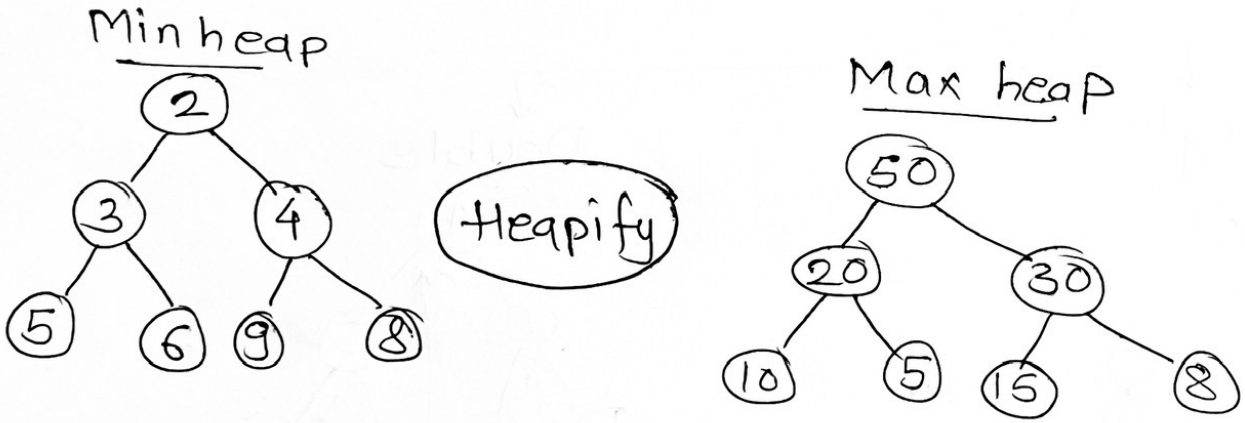


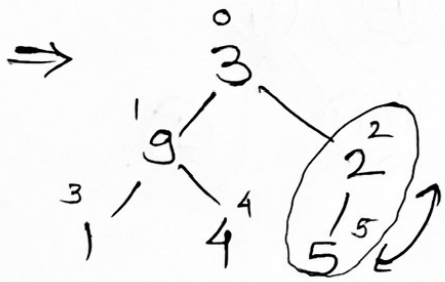


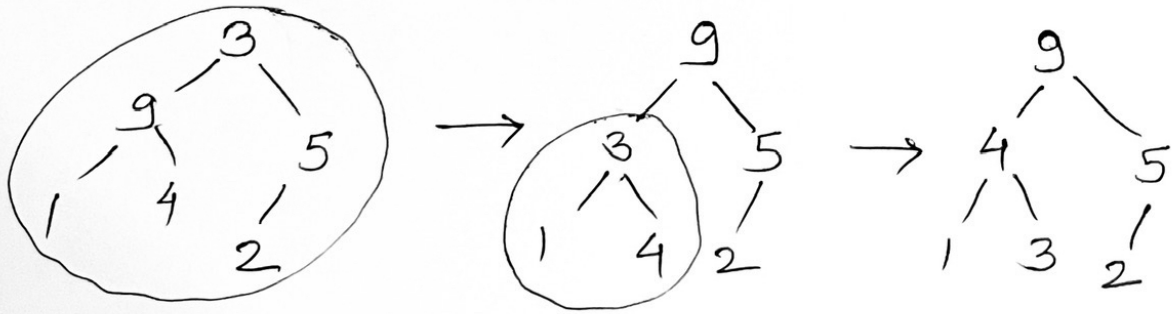
→ Heap

→ Complete Binary Tree

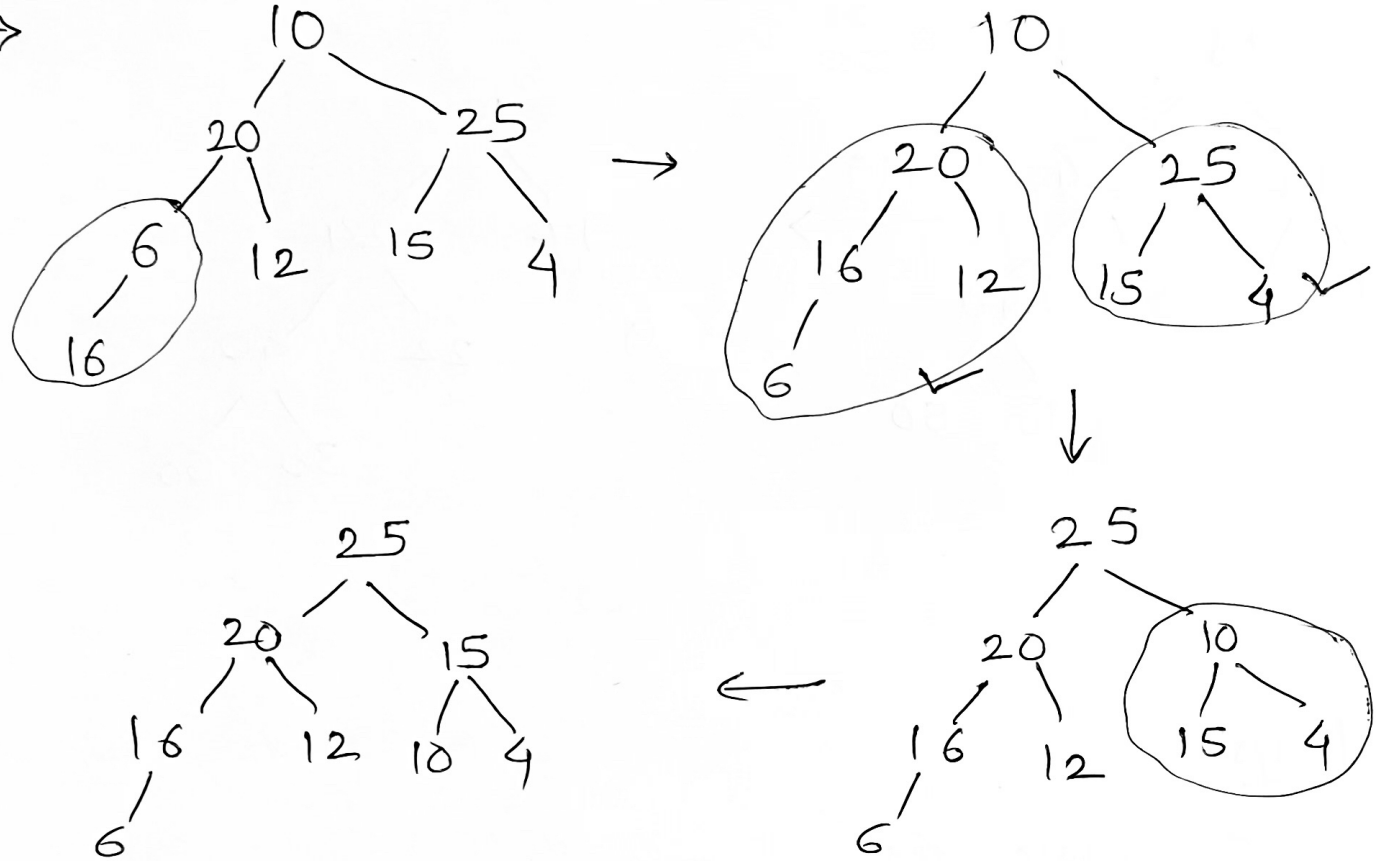
- → Min heap (Parent < child)
- → Max heap (Parent > child)



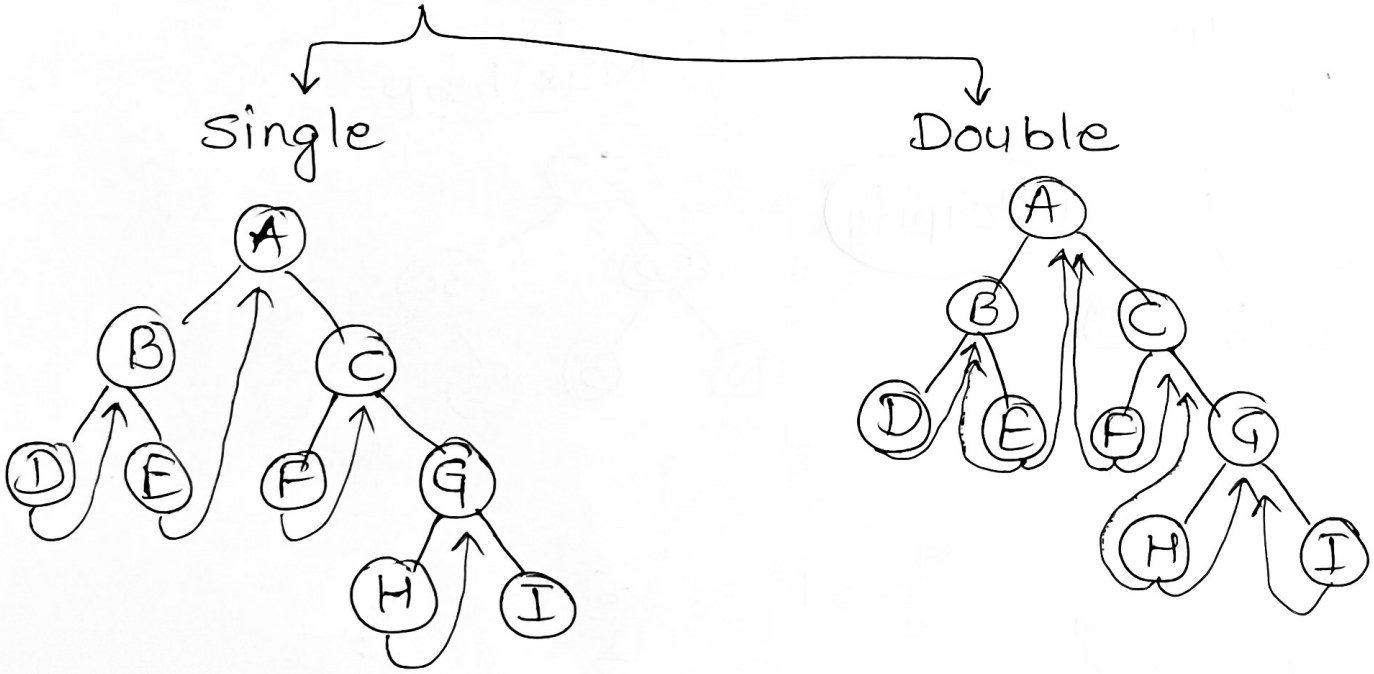




→

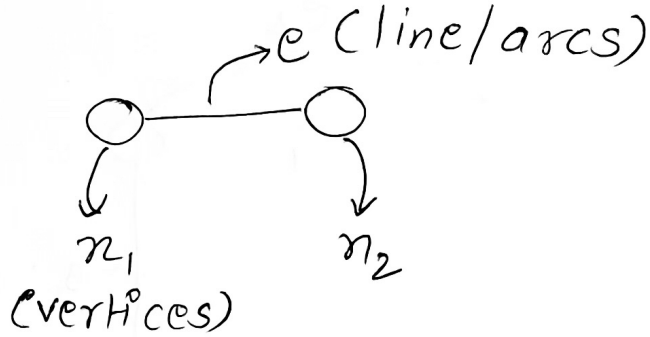

10
20
25
6
12
15
4
16
10
20
25
16
12
15
4
6
25
20
10
16
12
15
4
6
25
20
15
16
12
10
4
6

→ Threaded BT

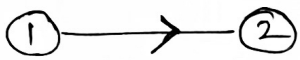

Single
Double
A
B
C
D
E
F
G
H
I
A
B
C
D
E
F
G
H
I

Graph

→ NLDS
- → Nodes
- → Edge

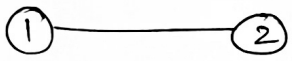


① Directed Graph



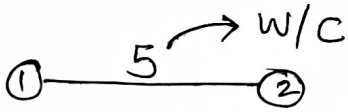
$1 \rightarrow 2$ (1,2)

② Undirected Graph

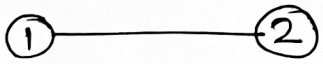


(1,2) (2,1)

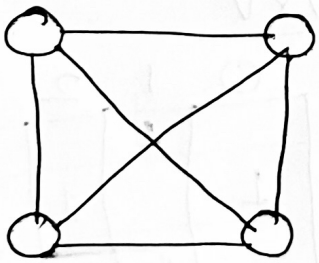
③ weighted Graph



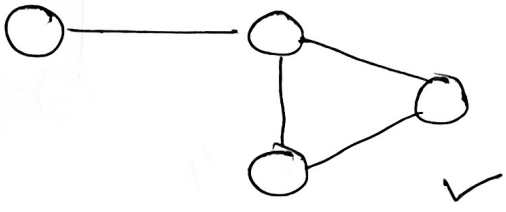
④ Unweighted Graph



⑤ Complete Graph



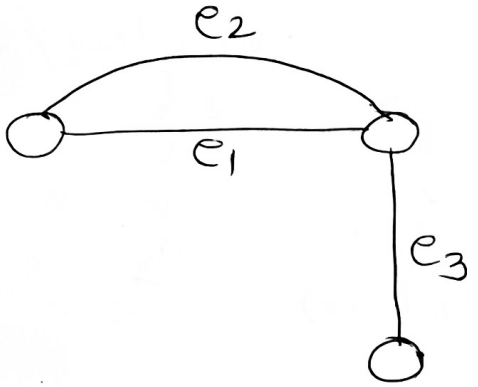
⑥ simple Graph



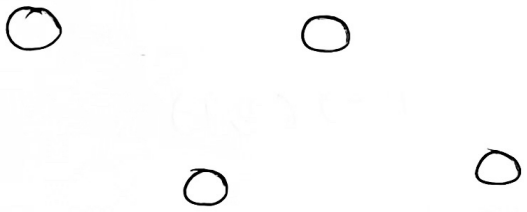
✓

✗

⑥ Multi Graph



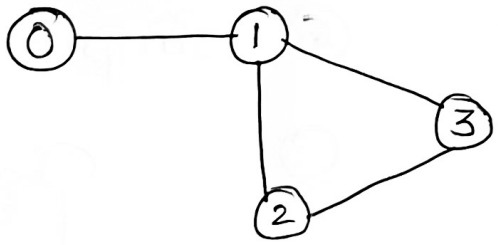
⑦ NULL Graph



⇒ Representation of graph:

- → Adjacency Matrix
- → Adjacency List

Eg:



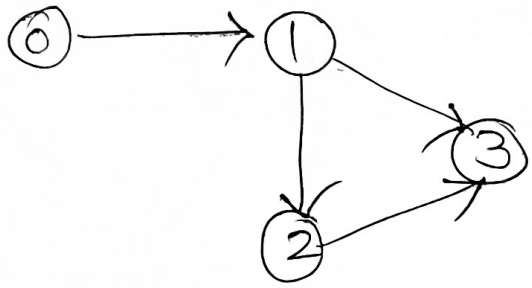
$V = 4$
$E = 4$

$1 \rightarrow \checkmark$
$0 \rightarrow \times$

V×V

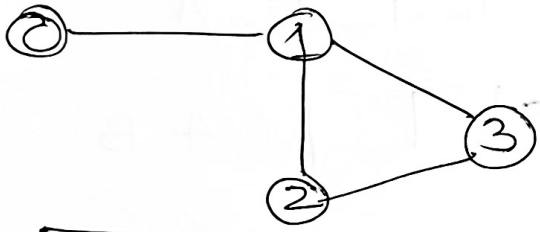
	0	1	2	3
0		1		
1	1		1	1
2		1		1
3		1	1	

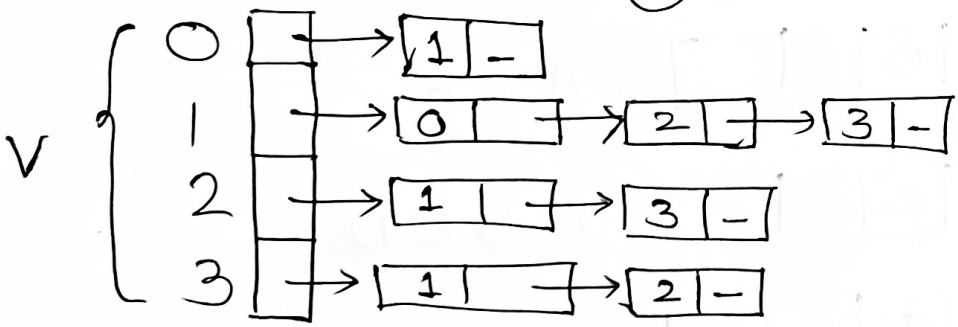
Eg



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

→ Adjacency List

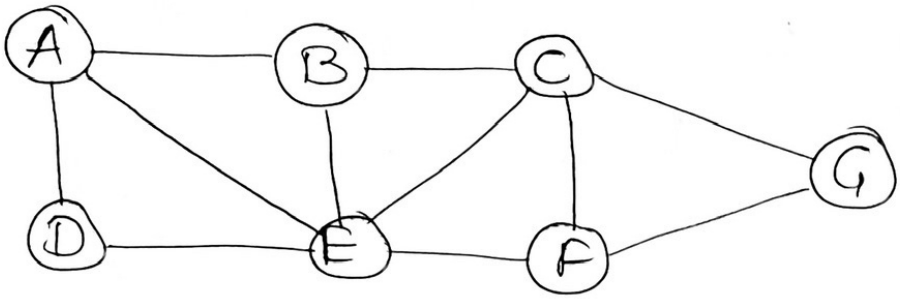




→ Graph Traversal

- → BFS
- → DFS

BFS →



Start with (A)

⇒

A						

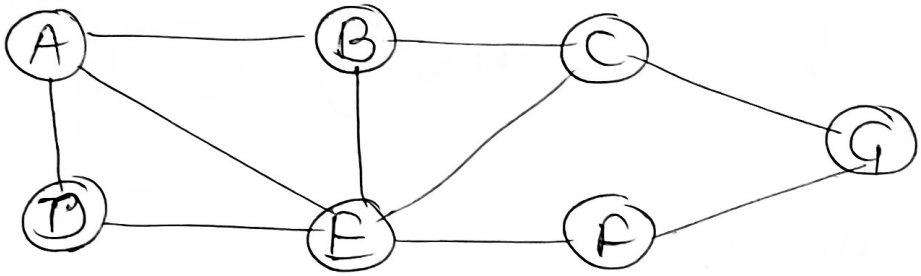
⇒

	B	E	D				: A
		E	D	C			: A B
			D	C	F		: A B E
				C	F		: A B E D
					F	G	: A B E D C
						G	: A B E D C F

↓

A B E D C F G

• DFS



A → B, D, E

B → A, C, E

C → B, E, F, G

D → A, E

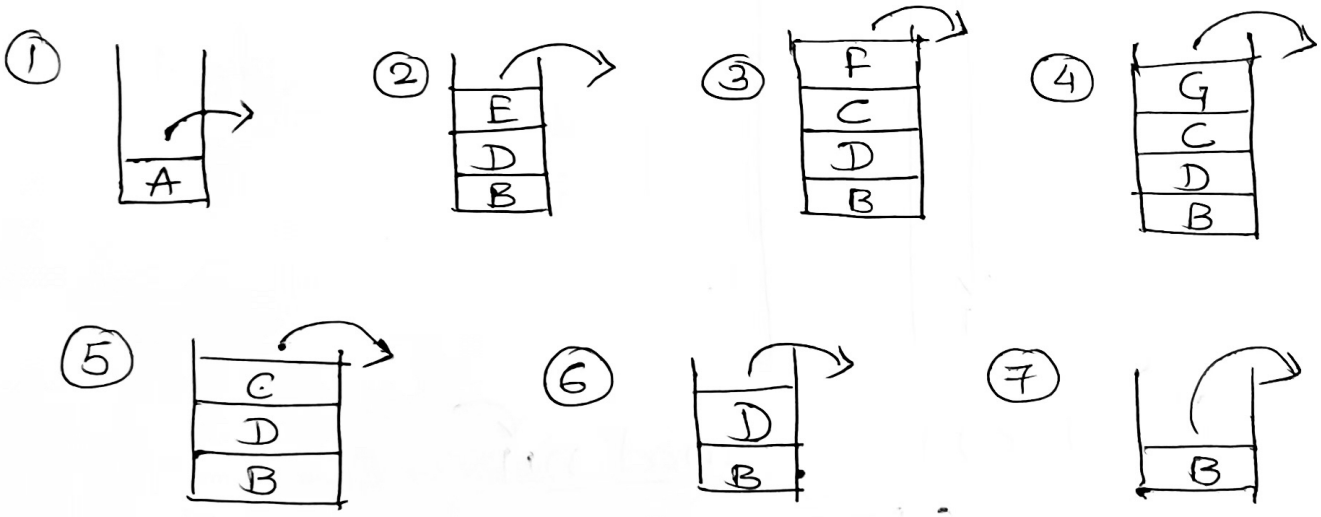
E → A, B, C, D, F

F → C, E, G

G → C, F

"A B C D E F G"

Ans: A E F G C D B



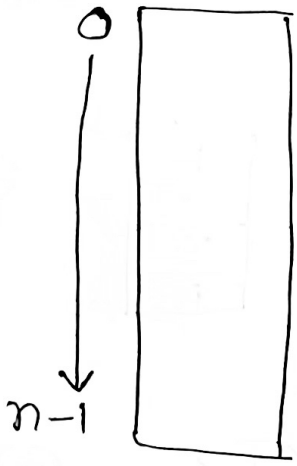
→ Hashing

(Time ↓ Space ↑)

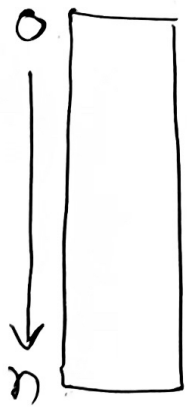
→ Key
→ Location
→ Hash f^n
  → Easy
  → Fast
  → low

→ Hash table.

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



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



keys → (10, 11, 12, 13, 14, 15, 16, 17, 18, 19)

f^n : K mod n

n = 10

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

HT

- Mid Square

⇒ 12 → square → 144

↓

I=4 (store 12)

⇒ 11 → square → 121

↓

I=2 (store 11)

- K = 123456 (folding method)

123 + 456

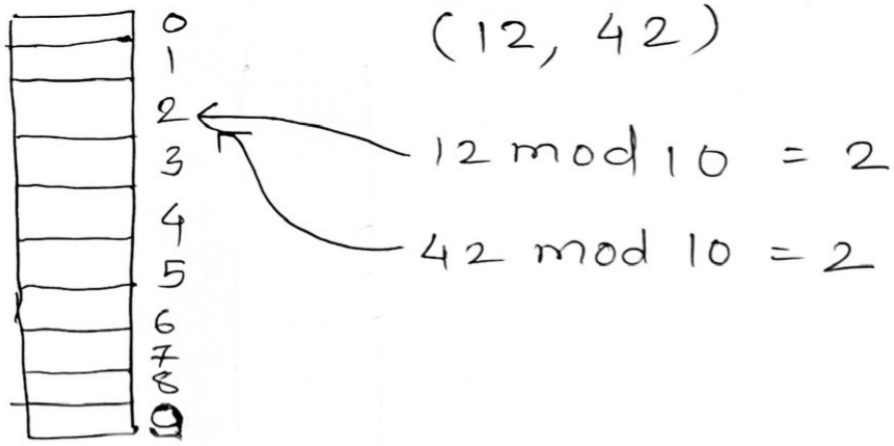
↓

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

∴ n = 10, so 579 mod 10

⇒ 9 (store 123456)

Collision

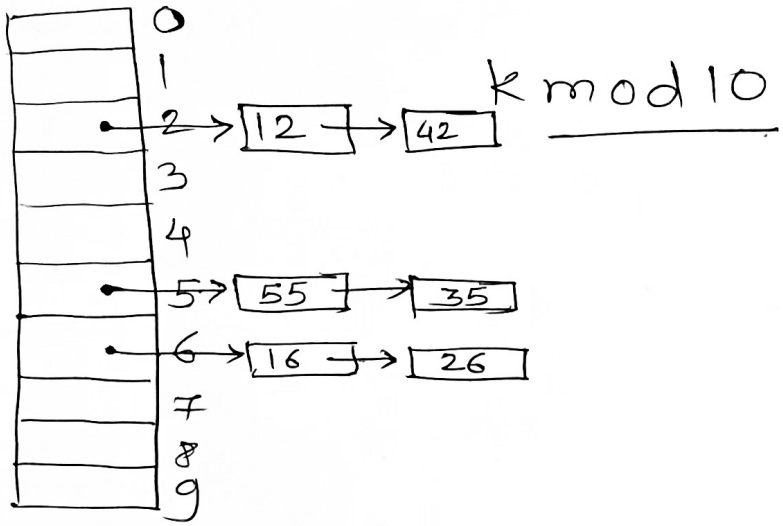


CRT

- (Open H) chaining
- (Closed H)
  - Linear Probing
  - Quadratic Probing
  - Double H ~~Probing~~.

Chaining

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

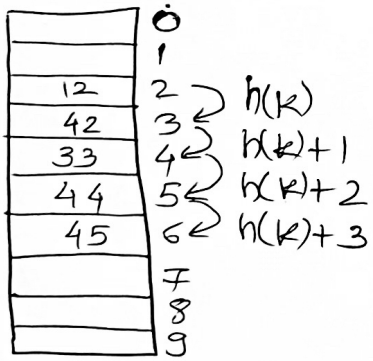


Linear Probing

k mod 10

$k \rightarrow (12, 42, 33, 44, 45)$

$h(k) \rightarrow$ 2, 2, 3, 4, 5



$k = 42$

$h(k) = 2$ (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

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

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

$46 \bmod 10 = 6$ [Collision]

$i = 1$

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

$i = 2$

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

k_1	k_2	k_3
2	2	2
3	3	3
6	6	6
1	1	1
8	8	8
7	7	7

(Secondary Clustering)

for 52

$52 \bmod 10 = 2$ [Collision]

$i = 1$

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

$i = 2$

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

$i = 3$

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

$i = 4$

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

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

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

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

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

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

Double Hashing

$(h1(k) + i\,h2(k)) \% n$

$h1(k) = k \bmod 13$

$h2(k) = 1 + (k \bmod 11)$

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

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

$h1(79) = 79 \% 13 = 1$

$h1(69) = 69 \% 13 = 4$

$h1(98) = 98 \% 13 = 7$

$h1(72) = 72 \% 13 = 7$ [Collision]

$\Rightarrow i = 1$

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

$\hookrightarrow 1$ [Collision

$\Rightarrow i = 2$

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

$\hookrightarrow 8$ ✓