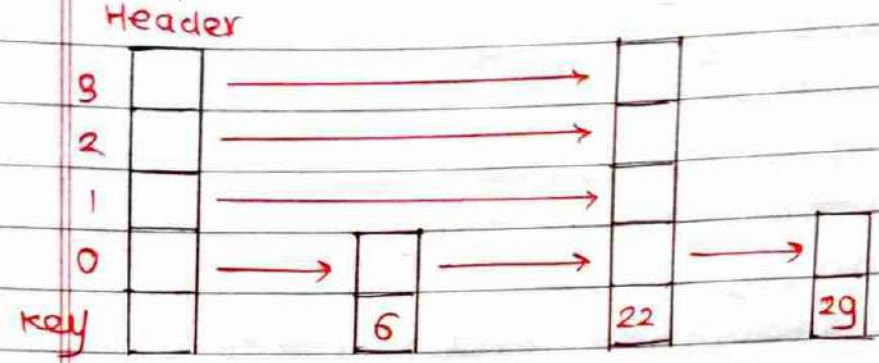
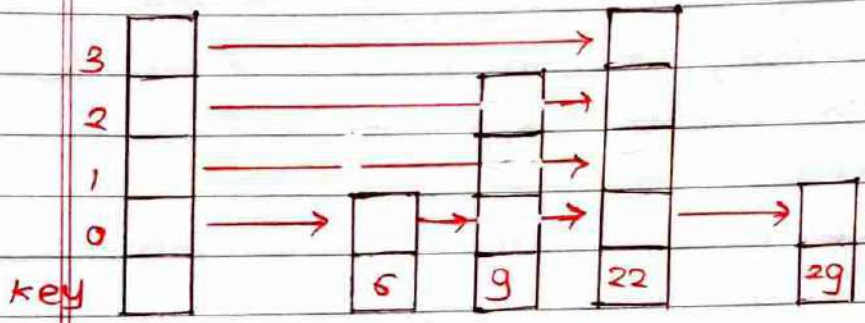
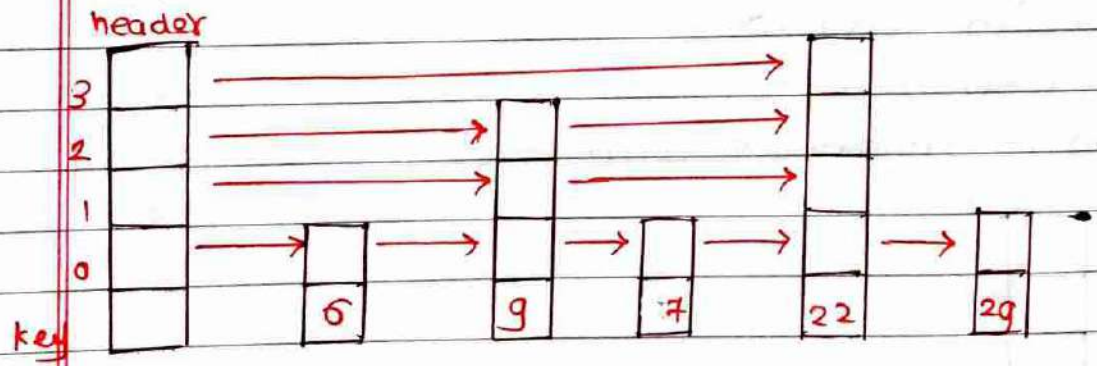
Step 3 : Insert 22 with level 4.



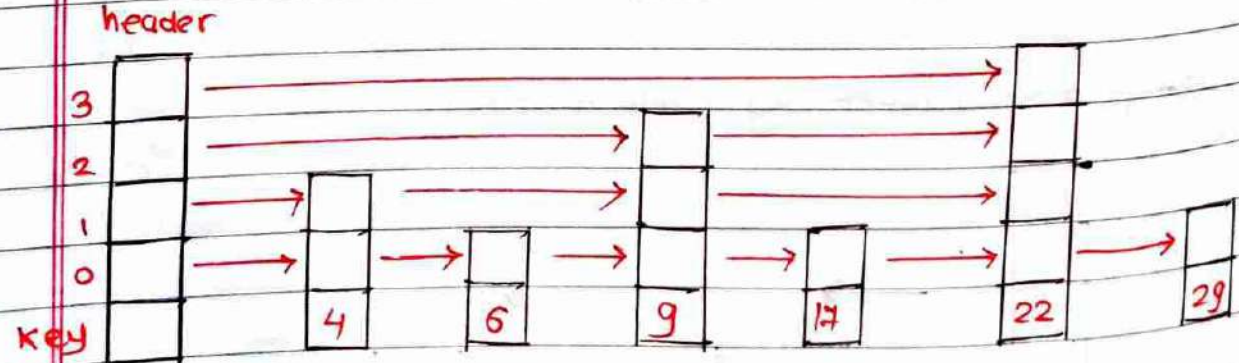
Step 4: Insert 9 with level 3.



Step 5: Insert 17 with level 1



Step 6: Insert 4 with level 2.



Stack :- A stack is a linear data structure that follows LIFO (Last-In-First-Out) principle. Stack has one end, whereas queue has two ends (front and rear).

A stack is a container in which insertion and deletion can be done from the end (one) known as the top of the stack.

A stack is an Abstract Data Type with a pre-defined capacity, which means that it can store elements of limited size.

Operations on the Stack :-

1). push ( ) :- When we insert an element in a stack then the operation is known as push. If stack is full overflow condition occurs.

2). pop ( ) :- when we delete an element from stack, the operation is called as pop ( ). If stack is empty means no elemebt exists in the stack, this state is known as an underflow state.

3). peek ( ) :- It returns the element at a given position.

4). count ( ) :- It returns the total number of elements available in a stack.

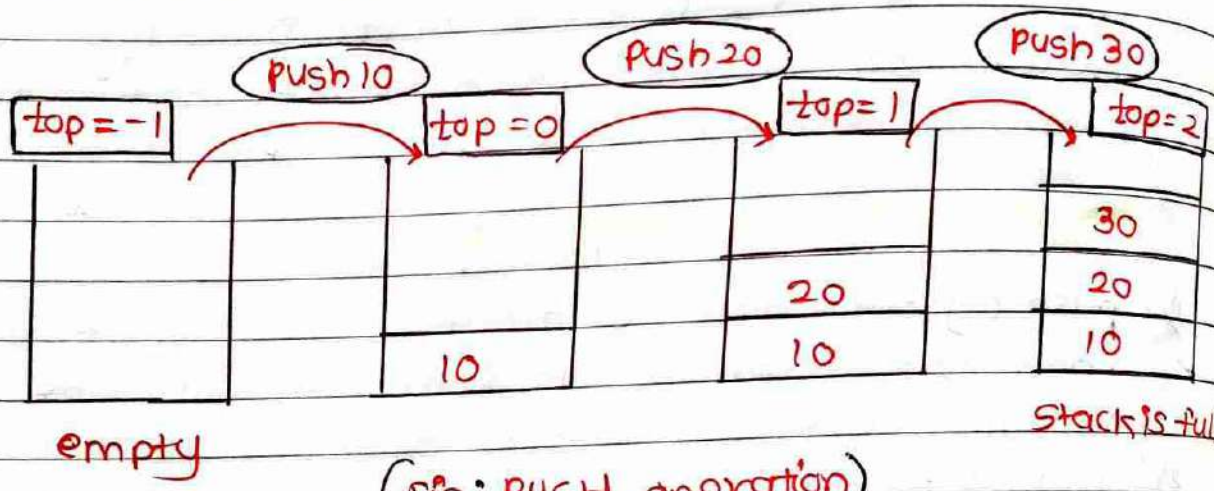
5). change ( ) :- It changes the element at the given position.

6). display ( ) :- It prints all the elements available in the stacks.

PUSH Operation :-

Steps - Before inserting an elemebt in the a stack, we check whether the stack is full.

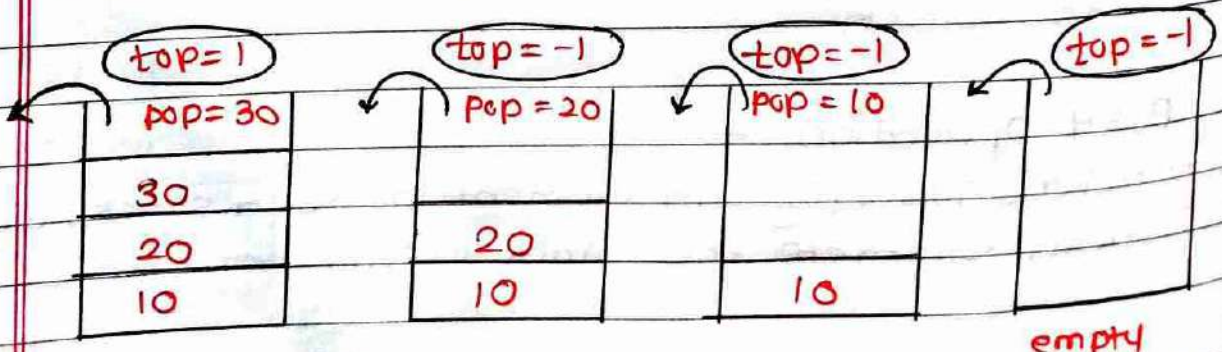
- If we try to insert element in a stack, and the stack is full, then overflow condition occurs.
- when we initialised a stack, we set the value of top as -1 to check that stack is empty
- The elements will be inserted until we reach the max size of the stack. top = top + 1.



(fig: PUSH operation)

POP Operation :-

- Before deleting the element from the stack, we check whether the stack is empty.
- If we try to delete the element from empty stack, then underflow condition occurs.
- first access the element which is pointed by top.
- once the top operation is performed, top is decremented by 1 i.e. top = top - 1.



Applications of stack :-

1) Recursion :- The recursion means that the function is calling itself again. To maintain the previous states, the compiler creates a system stack in which all previous records of function are maintained.

2) DFS (Depth first search) :- This search is implemented on a graph, graph uses stack d.s.

3) Backtracking :- If we have to create a path to solve maze problem, If we are moving in particular path and we realise that we come on the wrongway. In order to come at beginning of the path to create a new path, we use stack d.s.

4) Memory management :- The stack manages the memory. The memory is assign in the contiguous memory blocks.

Algo :- push operation :-

```
begin
  if top = n then stack full
  top = top + 1
  stack(top) := item ;
end
```

Time Complexity : O(1)

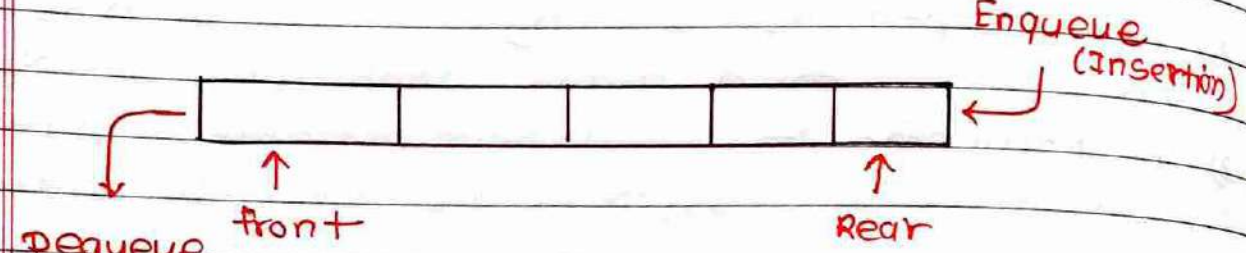
pop operation :-

```
begin
  if top = 0 then empty
  item := stack(top) ;
  top = top - 1 ;
  end.
```

Time complexity : O(1)

Queue :- A queue can be defined as ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.

- Queue can be referred as to be first In first out list.



Complexity of queue :-

	Average				Space comp.
	Access	search	Deletion	Insertion	Worst
Queue	O(n)	O(n)	O(1)	O(1)	O(n)

	Worst			
	Access	search	Insertion	Deletion
Queue	O(n)	O(n)	O(1)	O(1)

Operations on queue :-

1). Enqueue :- Enqueue is used to insert element at rear end of the queue. It returns void.

2). Dequeue :- dequeue operations performs the deletion from front end of queue. The deque operation can also be designed to void.

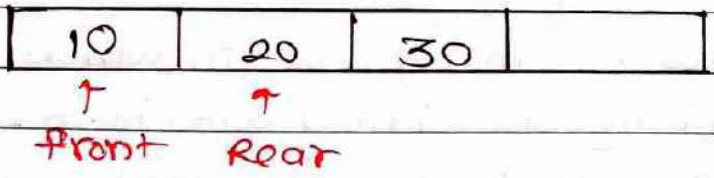
3). peek :- This returns, element which is pointed by front pointer in the queue but does not delete it.

4). queue overflow (is full) :- when queue is completely full, then it shows overflow condition.

5). queue underflow (isempty) :- when there is no element in the queue then it throws underflow condition.

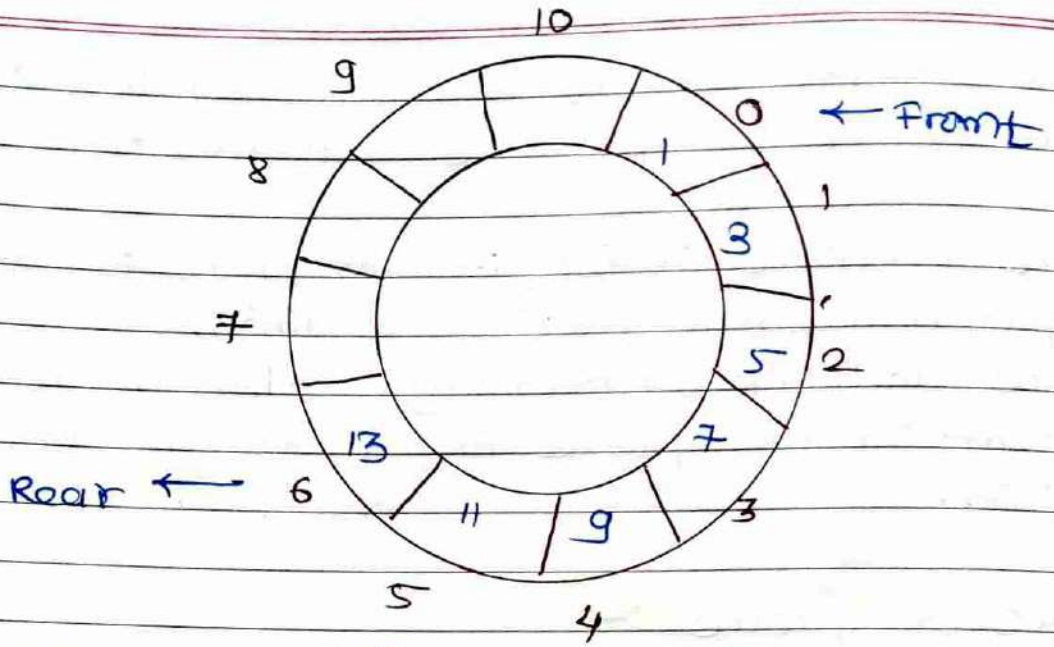
Types of queue :-

1). Linear queue :- In linear queue, an insertion takes place from one end while deletion occurs from another end. It strictly follows FIFO rule. The linear queue can be represented, as shown :



The elements are inserted from rear end, and if we insert more elements in queue, then rear value gets incremented on every insertion.
drawback is using linear queue is : insertion is done only from rear end. The linear queue shows the overflow condition as rear is pointing to last element of the queue.

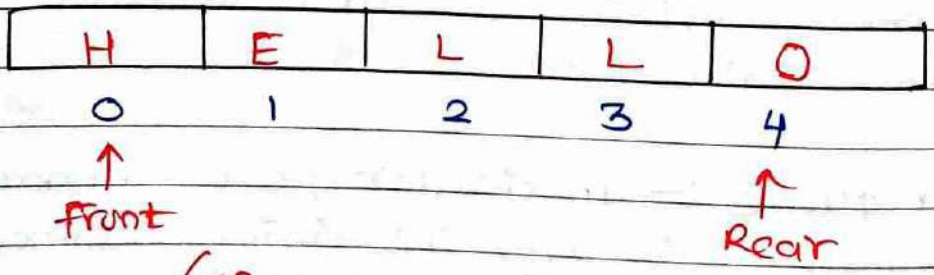
2). circular queue :- In circular queue, all nodes are represented as circular. It is similar to linear queue except that last element of the queue is connected to the first element. It is also known as ring buffer. as all ends are connected to another end.



Drawback of linear queue is overcomes in circular queue. If empty space is available, new element can be added in empty space by simply incrementing value of rear.

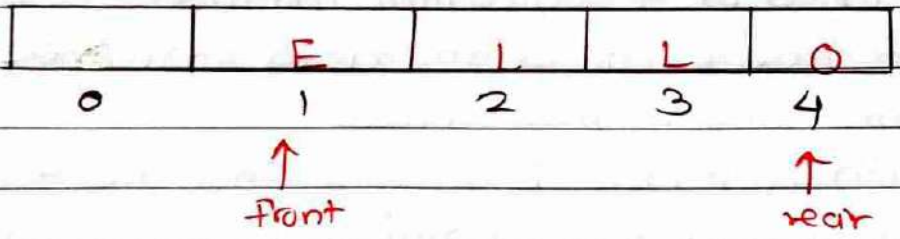
3). Priority queue :— The queue in which each element has some priority associated with it. Based on priority of the element, elements are arranged in a priority queue. If elements occur with same priority, then they are served according to FIFO principle.

* Array representation of queue :—



(fig : queue after inserting an element)

After deleting element, value of front will increase from -1 to 0, the queue will look like :



(Fig : queue after deleting an element)

Algorithm to insert any element in a queue :-
check if queue is already full by comparing rear to max -1.

Algo:
step 1 :- IF REAR = MAX -1
write OVERFLOW
Go to step [END OF IF]
step 2 :- IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE
SET REAR = REAR + 1 [END OF IF]
step 3 :- SET QUEUE [REAR] = NUM
step 4 :- EXIT.

Algorithm to delete an element from queue :-

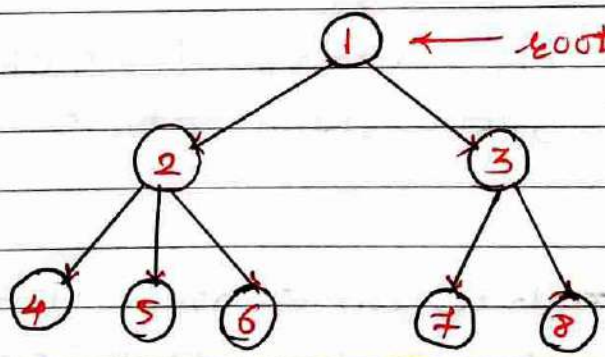
Algo:
step 1 :- IF FRONT = -1 or FRONT > REAR
write UNDERFLOW
ELSE
SET VAL = QUEUE [FRONT]
SET FRONT = FRONT + 1
[END of IF]
step 2 :- EXIT.

Tree :- We read data structure, like an array, linked list, stack and queue in which all elements are arranged in a sequential manner.

A tree is one of the data structures that represents hierarchical data.

defination :- A tree is a data structure defined as collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.

- A tree is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in tree are arranged in multiple levels.
- In tree data structure topmost node is called as root node. Each node contains some data, & data can be of any type.
- Each node contains some data & links or reference of other nodes that can be called children.



Some Basic terms of tree :-

1). **link :-** each node is labeled with some number. each array shown in fig is known as link between two nodes.

2). **Root :-** The root node is top most node in tree hierarchy. root node is one that doesn't have any parent. If node is directly linked to some other

node, then it would be called a parent-child relation-ship.

3). child node :- If the node is a descendant of any node, then node is called as child node.

4). parent :- If node contains any sub-node, then node is said to be parent of that sub-node.

5). sibling :- The nodes that have same parents are called siblings.

6). leaf node :- node which doesn't have any child node, a leaf a bottom-most node of tree.

7). ancestor node :- It is any predessor node on a path from root to that node. In the given fig, 1, 2, 5 are ancestors of node 10.

8). Descendant :- The immediate successor of given node is known as descendant of a node.

* Properties of tree data structures :-

1). Recursive data structure :- Tree is also known as recursive data structure. Recursion means reducing something in a self-similar manner.

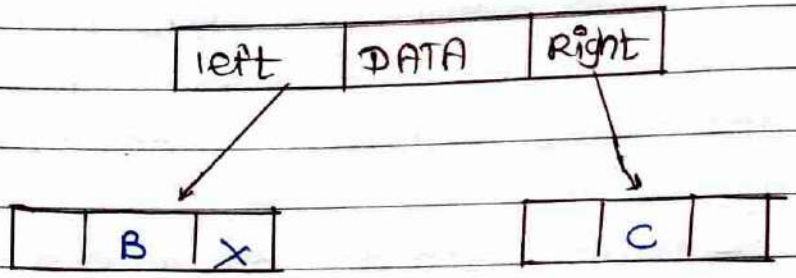
2). Number of edges :- If there are (n) nodes, then there would be (n-1) edges. each node, except root node, will have atleast one incoming line known as an edge.

3). Depth of node x :- It can be defined as length of path from root to node x. one edge contributes one unit length in the path, depth can be defined as no. of edges between root node and node (x)
The root node has depth 0.

4). Height of node x :- It is defined as longest path from node x to leaf node.

Implementation of tree :-

The tree data structure can be created nodes dynamically with help of pointers. The tree in memory can be represented as shown :




```
Struct node
 {
  int data;
 struct node *left;
 struct node *right;
 }
```

The above structure can only be defined for binary trees because binary tree can have utmost two children, and generic trees.

Application of trees :-

1). storing naturally hierarchical data :- File system, stored on disc drive, file and folder are in form of naturally heirarchical data and store in form of trees.

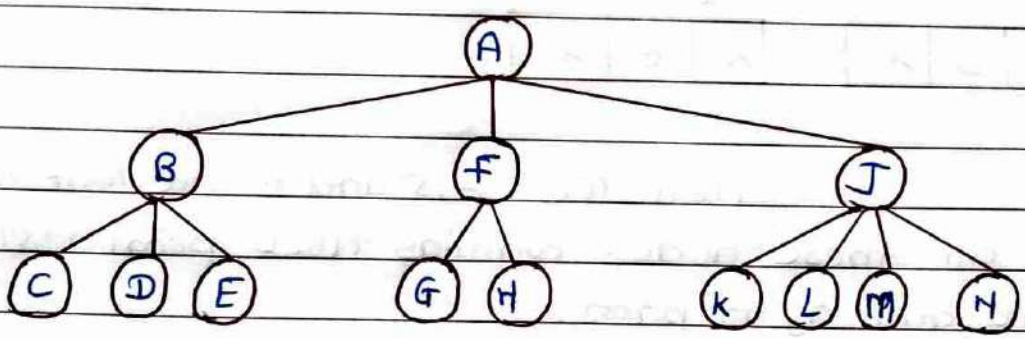
2). organize data :- It is used to organize data for efficient insertion, deletion and searching.

3). Trie :- It is special kind of tree that is used to store dictionary. It is fast and efficient way for dynamic spell checking.

4). Heap :- It is also a tree data structure implemented using arrays. It is used to implement priority queues.

Types of Tree data structure :—

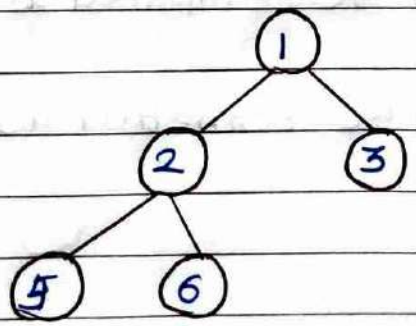
1) General Type :— In a general tree, a node can have either 0 or maximum n number of nodes. There is no restrictions imposed on the degree of node (number of nodes that a node can contain) The topmost node in a general tree is known as root node. The children of parent node are known as subtree.



There can be n number of subtrees in general tree. In general tree, subtrees are unordered as nodes in subtree cannot be ordered.

Every non empty tree has a downward edge, and these edges are connected to nodes known as child nodes. The nodes that have same parent are known as siblings.

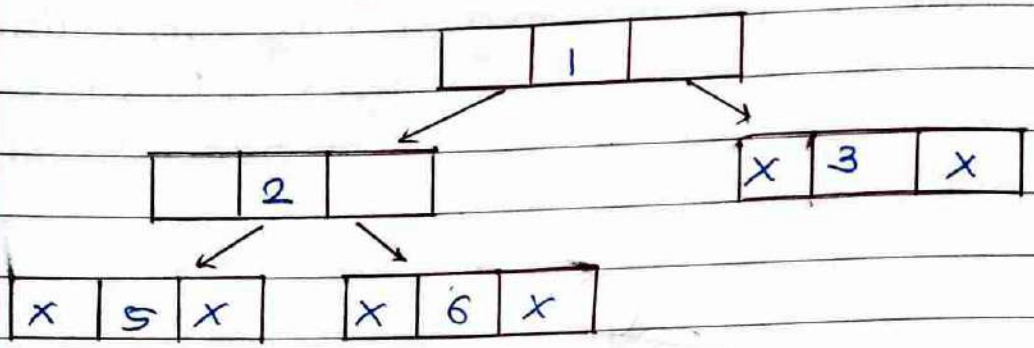
2) Binary Tree :— Binary tree means that the node can have maximum two children.



← given tree is a binary tree because
- each node contains atmost two children.

logical representation of above tree :-

In above tree, node 1 contains two pointers i.e. left and right pointer pointing to left and right node respectively.



The nodes 3, 5 and 6 are leaf nodes, so all these nodes contains NULL pointer on both left and right parts.

Properties of Binary tree :-

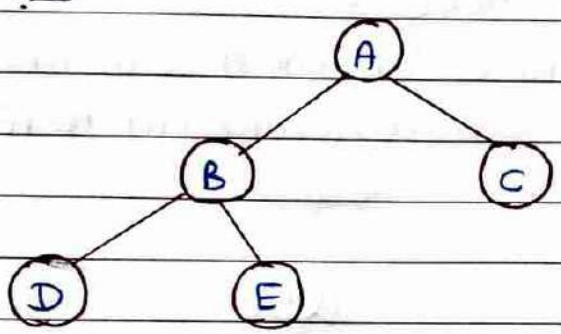
- At each level of i, the maximum number of nodes is 2^i.
- The height of tree is longest path from root node to leaf node. In general, maximum number of nodes possible at height is $(2^0 + 2^1 + 2^2 + .. 2^h)$
- The minimum number of nodes possible at height h is equal to h+1.
- If number of nodes is minimum, then height of tree would be maximum.
- minimum height can be computed as:
  $h = \log_2(n+1) - 1$
- maximum height can be computed as:
  $h = n - 1$.

Types of Binary tree :-

1). full / proper / strict Binary tree :-

If each node contains either 0 or two children. The tree in which each node must contain 2 children except left nodes.

Example :-



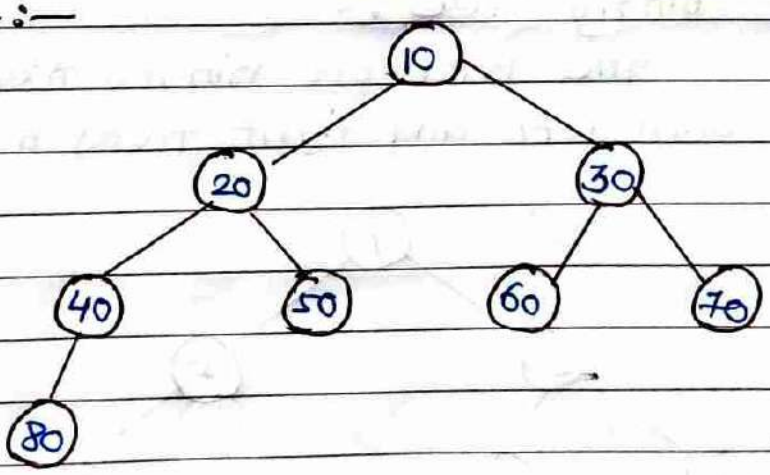
Properties :-

- maximum number of nodes : $2^{h+1} - 1$.
- minimum number of nodes : $2 * h - 1$
- minimum height $\log_2 (n+1) - 1$
- maximum height $h = \frac{n+1}{2}$

2). complete Binary tree :-

The tree in which all nodes are completely filled except the last level. In complete Binary tree nodes should be added from left.

Example :-



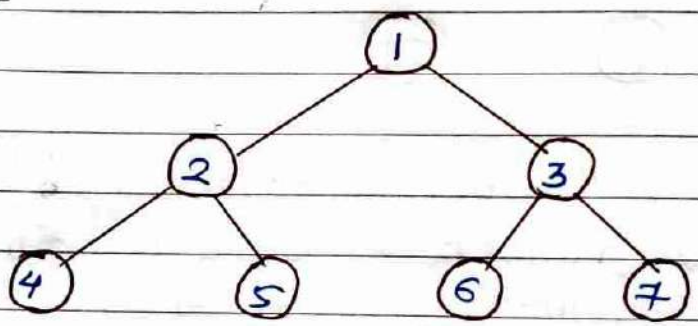
properties :-

- maximum number of nodes → $2^{h+1} - 1$.
- minimum number of nodes → 2^h
- minimum height → $\log_2 (n+1) - 1$.

3). Perfect Binary tree :-

A tree in which all the internal nodes have 2 children. and all leaf nodes are at the same level.

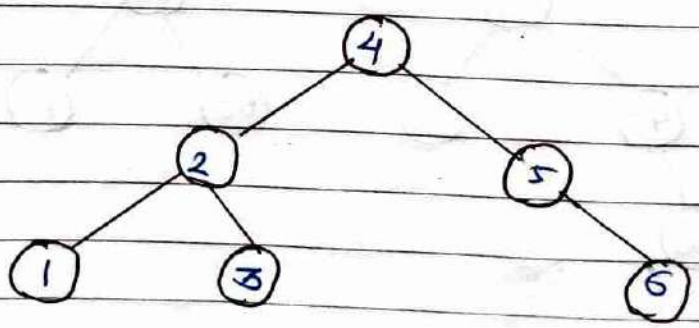
Example :-



Note :- All the perfect Binary trees are complete binary trees as well as the full Binary trees as But, vice versa is not true, all complete binary trees and full binary trees are the perfect Binary trees

4). Balanced Binary Tree :-

The balanced binary tree is a tree in which both left and right trees by almost 1.



above tree is balance : diff b/w left subtree & right S.T. is zero

Binary Tree implementation :-

```
struct node
 {
  int data;
  struct node *left, *right;
 }
```

Tree Traversal :-

The process of visiting nodes is called as tree traversal.

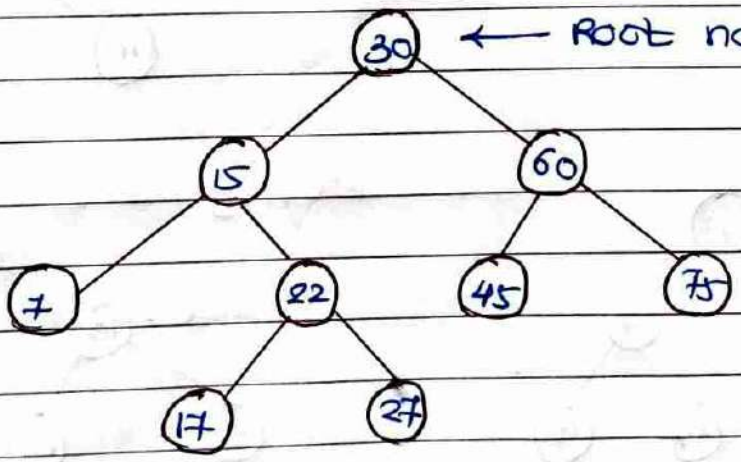
There are three types of traversals used to visit a node :

1). Inorder Traversal
2). preorder Traversal
3). postorder Traversal.

3). Binary Search Tree :-

defn :- Binary search tree can be defined as a class of binary trees, in which a nodes are arranged in a specific order. also called as ordered BinaryTree.

- similarly value of all nodes in right subtree is greater than or equal to value of root.



Example : create binary search tree using following data elements :-

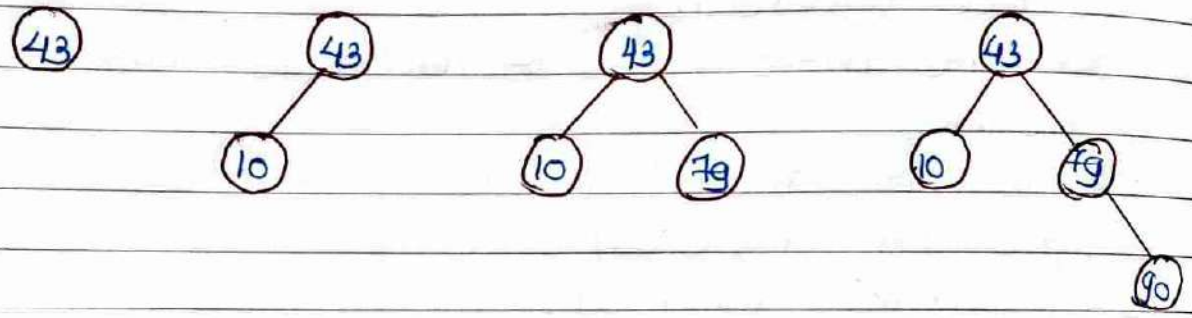
43, 10, 79, 90, 12, 54, 11, 9, 50.

→ 1). Insert 43 into tree as root of tree.

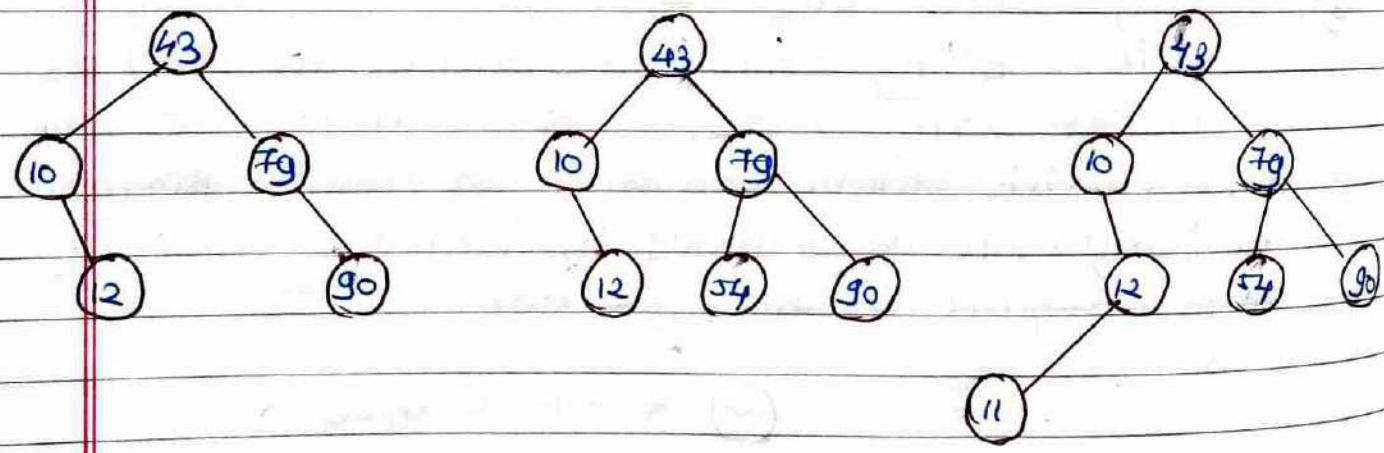
2). Read next element, if it is lesser root node element, insert it as root of left sub-tree.

3). otherwise, insert it as root of right of right subtree.

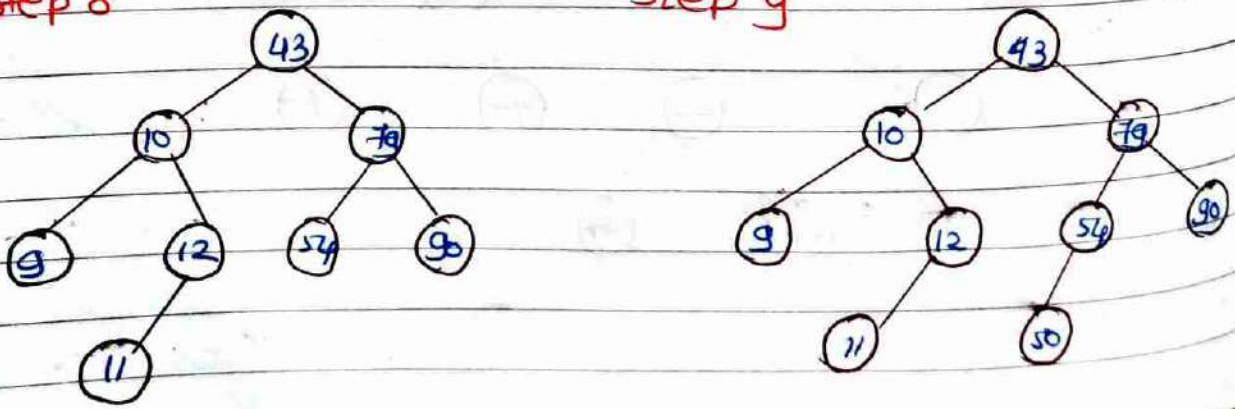
step 1 step 2 step 3 step 4



step 5 step 6 step 7



step 8 step 9



Operations on Binary Search Tree (BST) :—

Sr.No.	Operation	Description
1).	searching in BST	finding location of some specific element in a Binary search Tree.
2).	Insertion in BST	Adding a new element to the binary search tree at appropriate location so that property of BST do not violate.
3).	Deletion in BST.	Deleting some specific node from a BST, However, tree there can be various cases in deletion depending upon number of children, node have.

4). AVL Tree :— AVL Tree is invented by GM Adelson-Velsky and EM Landis in 1962. The tree is named as AVL in honour of its inventors.

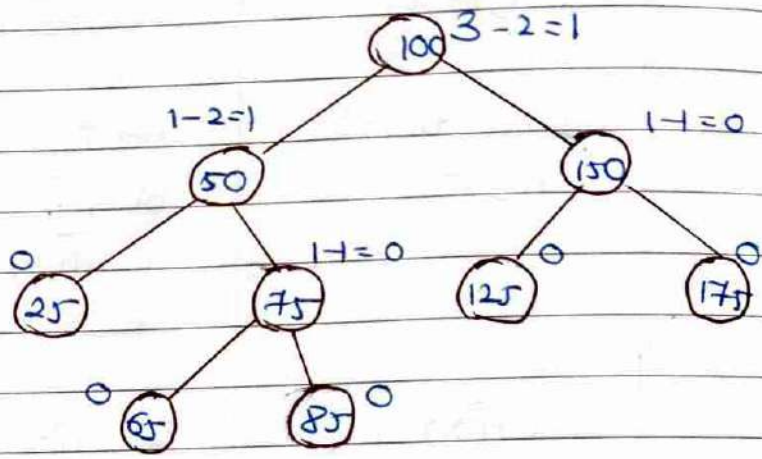
AVL tree is defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its R. subtree from its left subtree.

Balance factor (k) = height(left(k)) − height(right(k))

— If balance factor of any node is 1, it means that left sub-tree is one level higher than right subtree.

- If balance factor of any node is 0, it means that left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that left sub-tree is one level lower than right subtree.

Example :-



Here we see that, balance factor associated with each node is between -1 and +1.
∴ It is an example of AVL tree.

Complexity :-

Algorithm	Average case	Worst case
space	O(n)	O(n)
search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Why AVL Tree ? → AVL tree controls height of binary search tree by not letting it to be skewed me time taken by all operations in BST is O(h) However it will be extended to O(n) If BST became

skewed (worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n), where n is number of nodes.

Operations on AVL tree :-

Sr. No.	Operation	Description
1).	Insertion	Insertion is performed in same way it performed in BST. However, it may lead to violation in the AVL tree property and so tree may need balancing. and tree can be balanced by rotation.
2).	Deletion	Deletion is also same way performed as BST. It can be also disturb balance of tree, so various types of rotations are used to rebalance tree.

AVL Rotations :-

We perform rotations in AVL tree only in case if Balance factor is other than -1, 0 and 1. There are Basically four types of rotations which are as follows :