WILEY EDGE

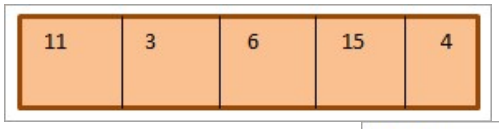
Data Structures in java

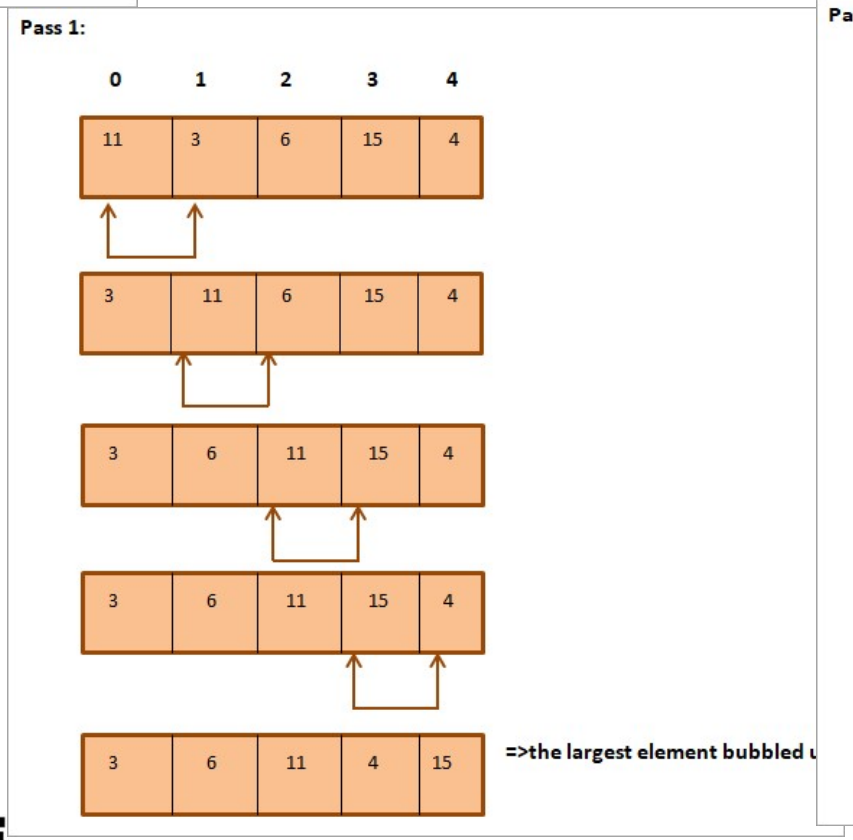
@Mandar Jog

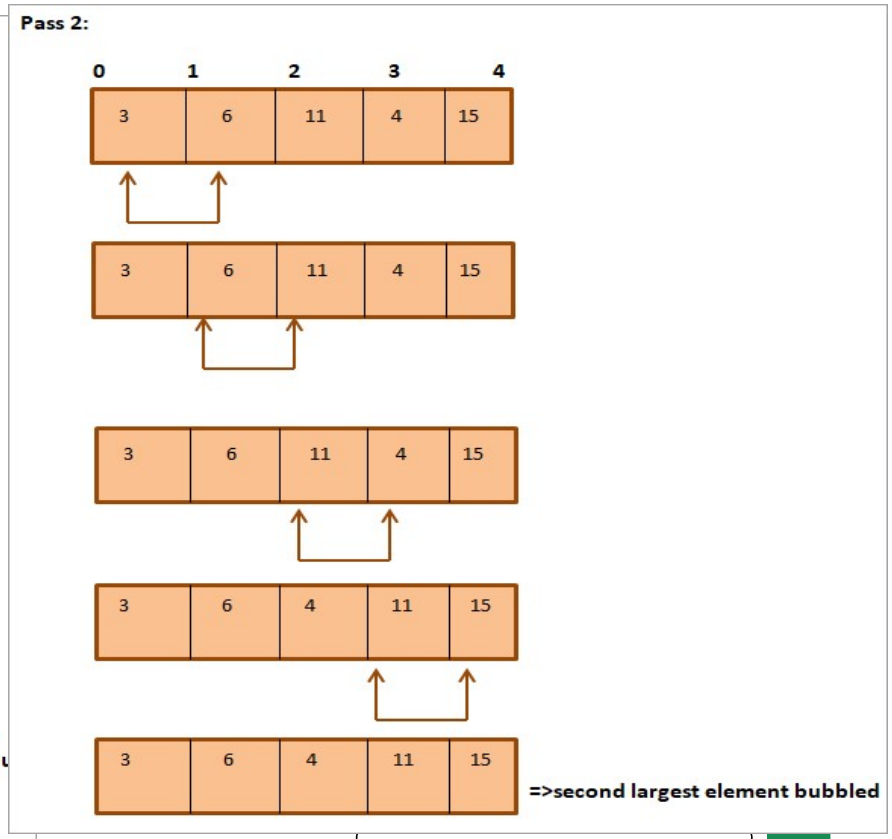
Sorting mechanisms

Sorting algorithm	Description
Bubble Sort	Compares the current element to adjacent elements repeatedly. At the end of each iteration, the heaviest element gets bubbled up at its proper place.
Insertion Sort	Inserts each element of the collection in its proper place.
Merge Sort	It follows the divide and conquer approach. Divides the collection into simpler sub-collections, sorts them and then merges everything
Quick Sort	Most efficient and optimized sorting technique. Uses divide and conquer to sort the collection.
Selection Sort	Finds the smallest element in the collection and put it in its proper place at the end of every iteration
Radix Sort	Linear sorting algorithm.
Heap Sort	Elements are sorted by building min heap or max heap.

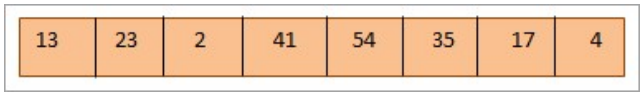
Bubble Sort

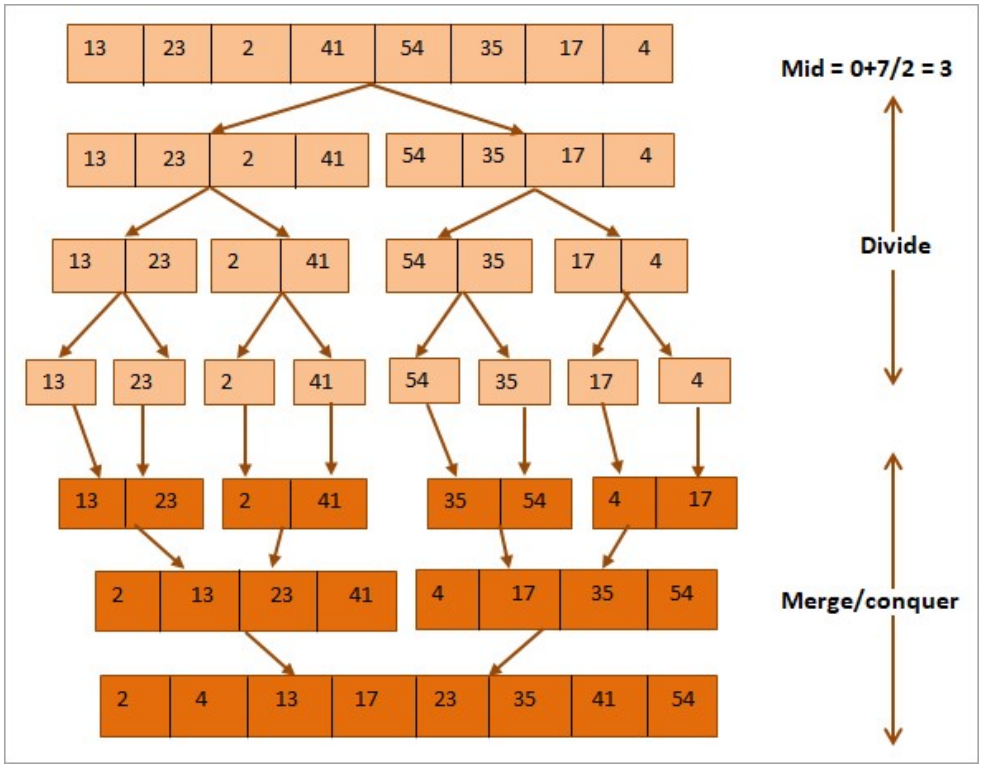

11
3
6
15
4


Pass 1:
0
1
2
3
4
11
3
6
15
4
3
11
6
15
4
3
6
11
15
4
3
6
11
15
4
3
6
11
4
15
=>the largest element bubbled u


Pass 2:
0
1
2
3
4
3
6
11
4
15
3
6
11
4
15
3
6
11
4
15
3
6
4
11
15
3
6
4
11
15
=>second largest element bubbled

Merge Sort

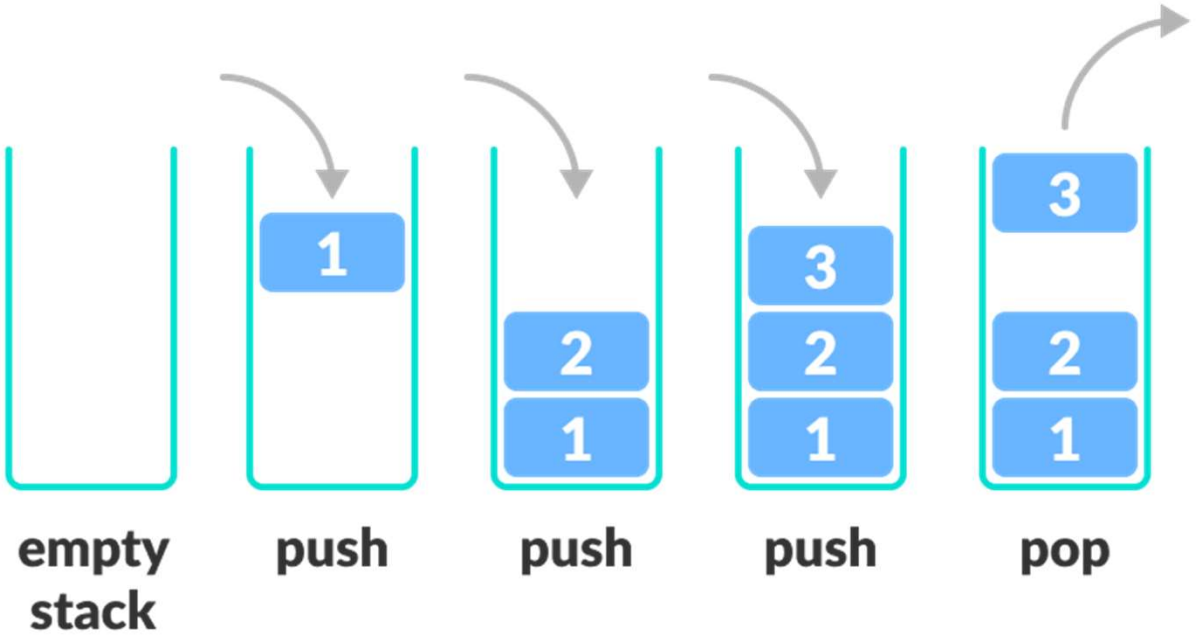

13
23
2
41
54
35
17
4


13 23 2 41 54 35 17 4
Mid = 0+7/2 = 3
13 23 2 41 54 35 17 4
13 23 2 41 54 35 17 4
Divide
13 23 2 41 54 35 17 4
13 23 2 41 54 35 17 4
13 23 2 41 35 54 4 17
2 13 23 41 4 17 35 54
Merge/conquer
2 4 13 17 23 35 41 54

Stack data structure

A stack is a linear data structure that follows the principle of **Last In First Out (LIFO)**. This means the last element inserted inside the stack is removed first.

LIFO Principle of Stack



Stack : Basic Operations

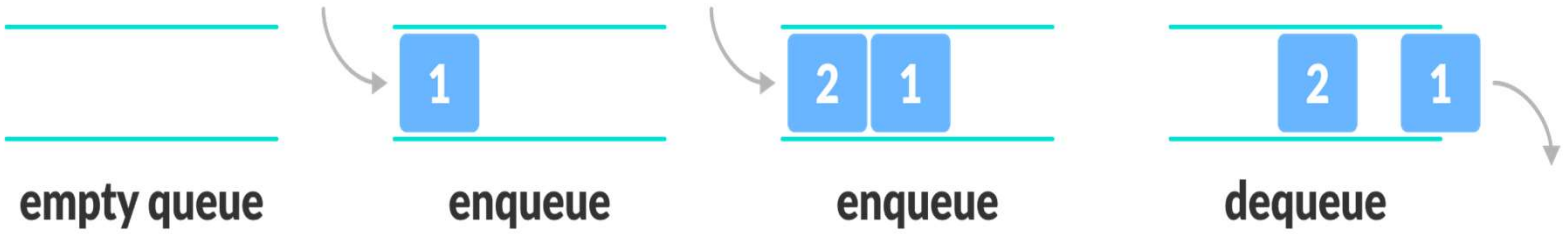
- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

Stack : Algo

1.A pointer called TOP is used to keep track of the top element in the stack.
2.When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
3.On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
4.On popping an element, we return the element pointed to by TOP and reduce its value.
5.Before pushing, we check if the stack is already full
6.Before popping, we check if the stack is already empty

Queue Data Structure

Queue follows the **First In First Out (FIFO)** rule - the item that goes in first is the item that comes out first.



Queue : Basic Operations

- **Enqueue**: Add an element to the end of the queue
- **Dequeue**: Remove an element from the front of the queue
- **IsEmpty**: Check if the queue is empty
- **IsFull**: Check if the queue is full
- **Peek**: Get the value of the front of the queue without removing it

Queue : Algo

Queue algorithm work as follows:

- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- initially, set value of FRONT and REAR to -1

Enqueue Operation

- check if the queue is full
- for the first element, set the value of FRONT to 0
- increase the REAR index by 1
- add the new element in the position pointed to by REAR

Dequeue Operation

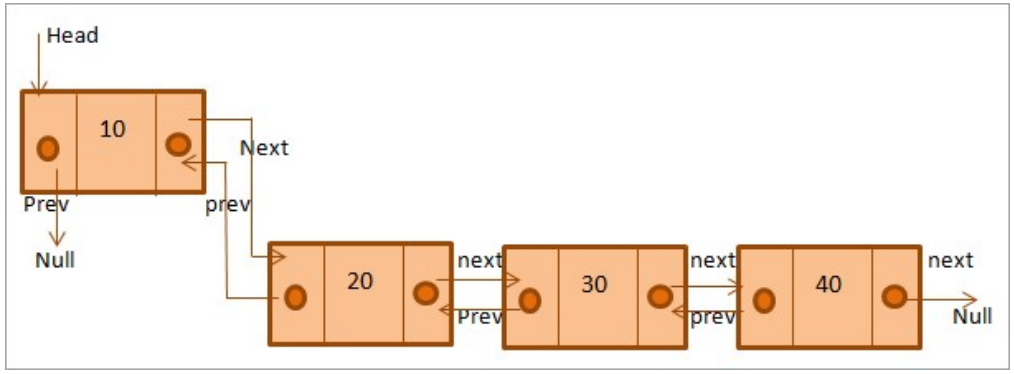
- check if the queue is empty
- return the value pointed by FRONT
- increase the FRONT index by 1
- for the last element, reset the values of FRONT and REAR to -1

LinkedList Data Structure (Singly LinkedList)

A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the **data** and the **address** of the next node.

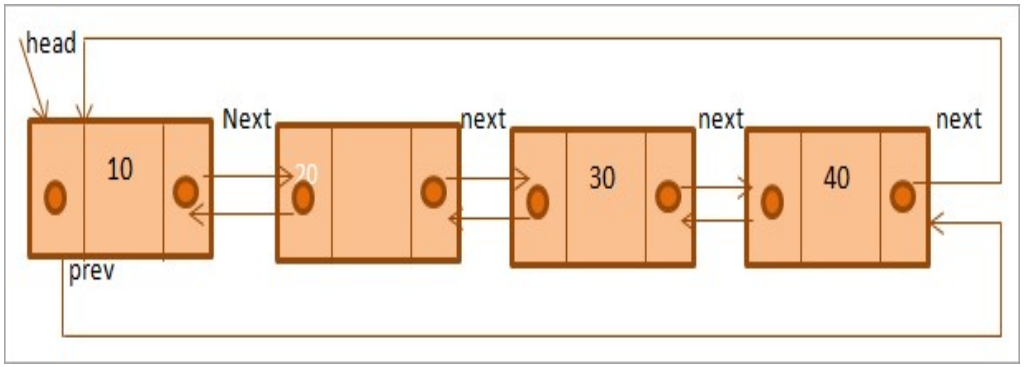
Doubly LinkedList

A doubly linked list has an additional pointer known as the previous pointer in its node apart from the data part and the next pointer as in the singly linked list.



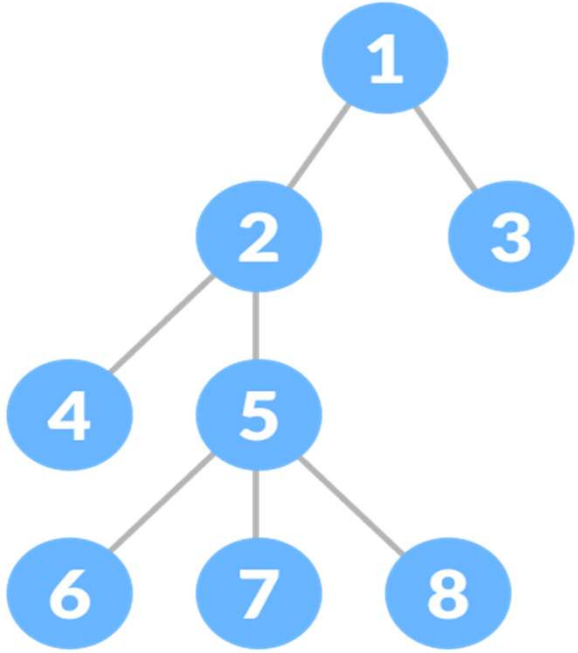
Circular Doubly LinkedList

In this list, the last node of the doubly linked list contains the address of the first node and the first node contains the address of the last node. Thus in a circular doubly linked list, there is a cycle and none of the node pointers are set to null.



Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



Tree Terminologies

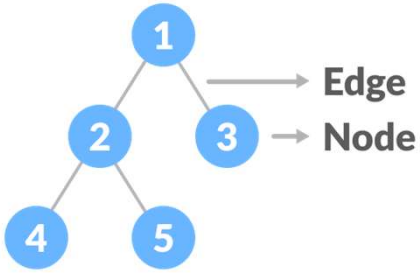
Node

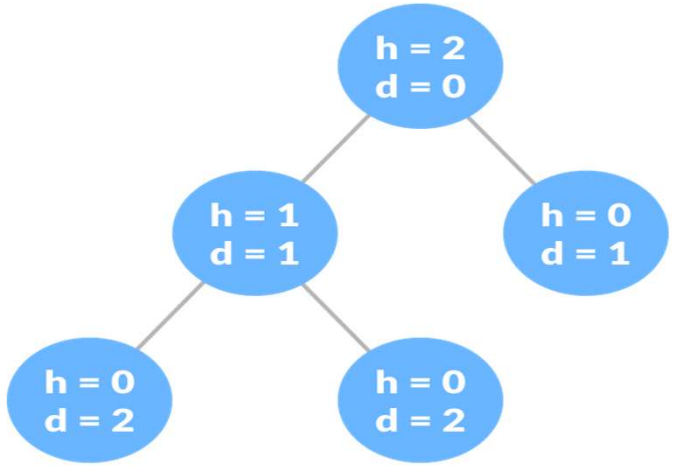
A node is an entity that contains a key or value and pointers to its child nodes.
The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.
The node having at least a child node is called an **internal node**.

Edge

It is the link between any two nodes.

Terminologies





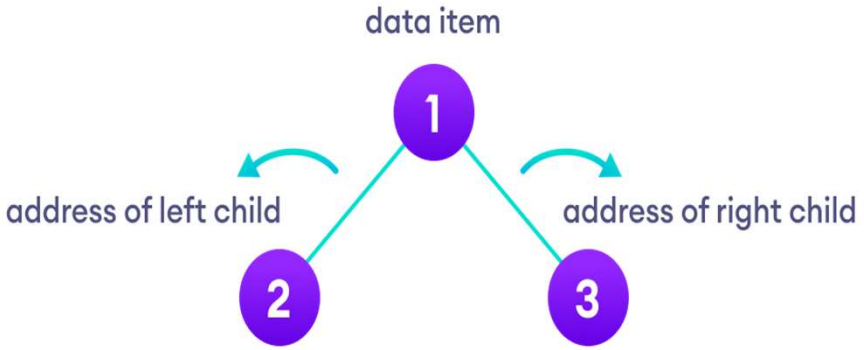
Depth of Node:
The depth of a node is the number of edges from the root to the node.

Height of Node:
The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Binary Tree

A binary tree is a tree data structure in which each parent node can have at most two children.

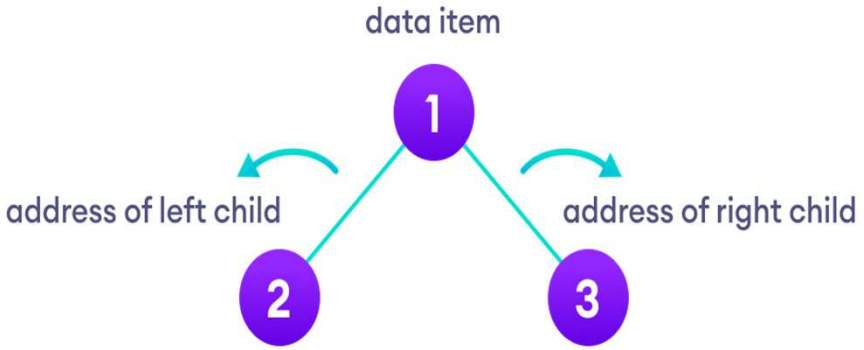
Each node of a binary tree consists of three items:
- data item
- address of left child
- address of right child



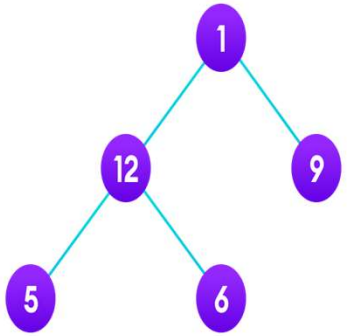
Binary Tree

A binary tree is a tree data structure in which each parent node can have at most two children.

Each node of a binary tree consists of three items:
- data item
- address of left child
- address of right child



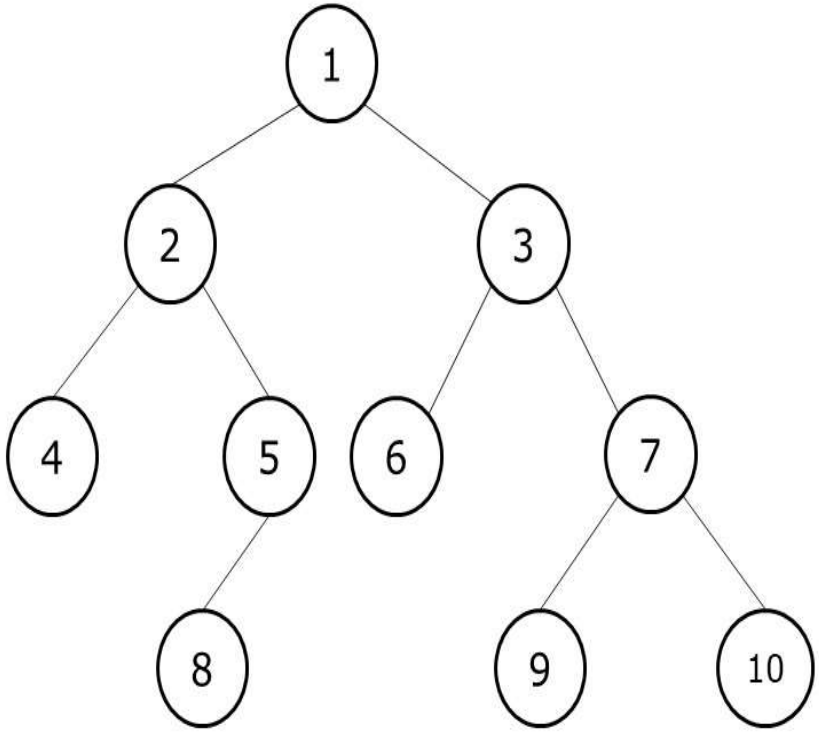
Tree Traversing



InOrder
1.First, visit all the nodes in the left subtree
2.Then the root node
3.Visit all the nodes in the right subtree

PreOrder
1.Visit root node
2.Visit all the nodes in the left subtree
3.Visit all the nodes in the right subtree

PostOrder
1.Visit all the nodes in the left subtree
2.Visit all the nodes in the right subtree
3.Visit the root node



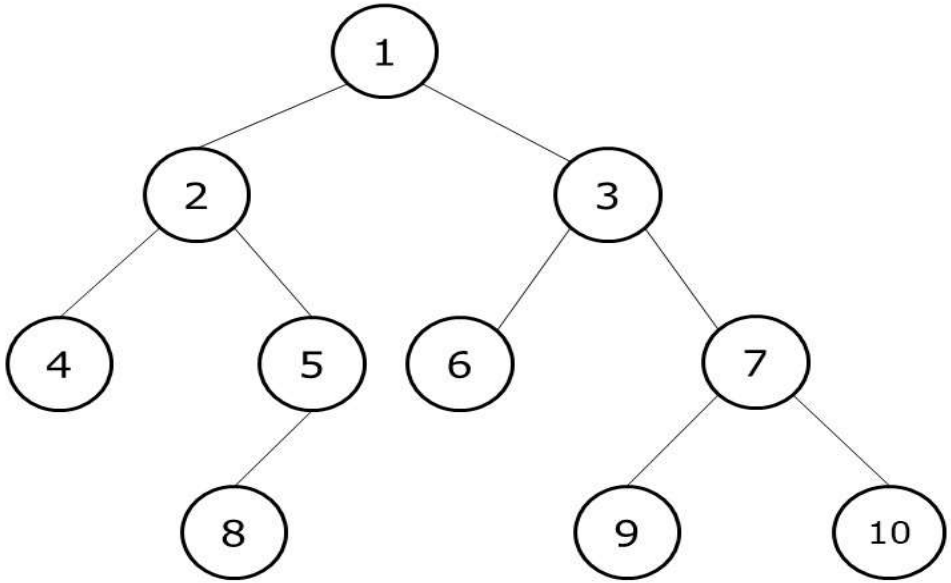
Inorder Traversal:

[left,root,right]

4	2	8	5	1	6	3	9	7	10

Types of Binary Tree

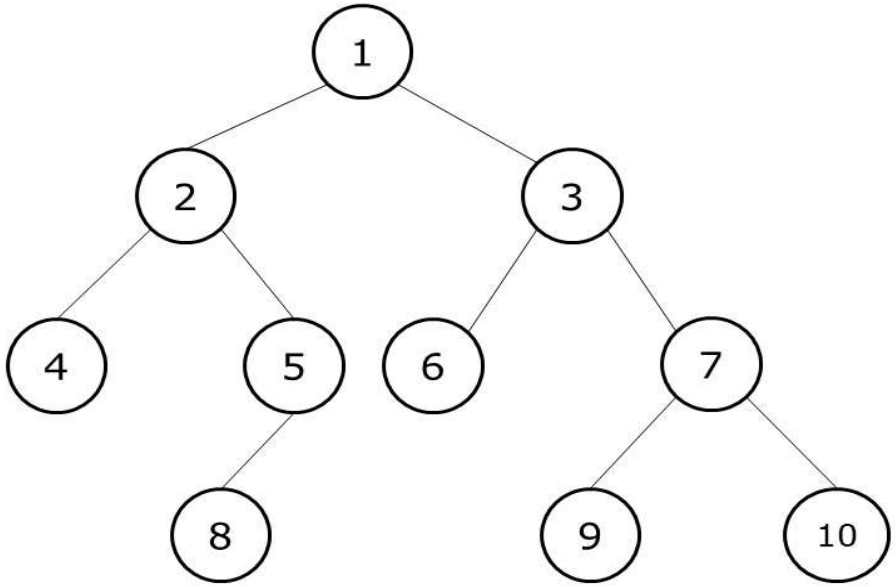
- Full Binary Tree
- Perfect Binary Tree
- Complete Binary Tree
- Degenerate Binary Tree
- Skewed Binary Tree
- Balanced Binary Tree
- Unbalanced Binary Tree



Preorder Traversal:

[root, left, right]

1	2	4	5	8	3	6	7	9	10



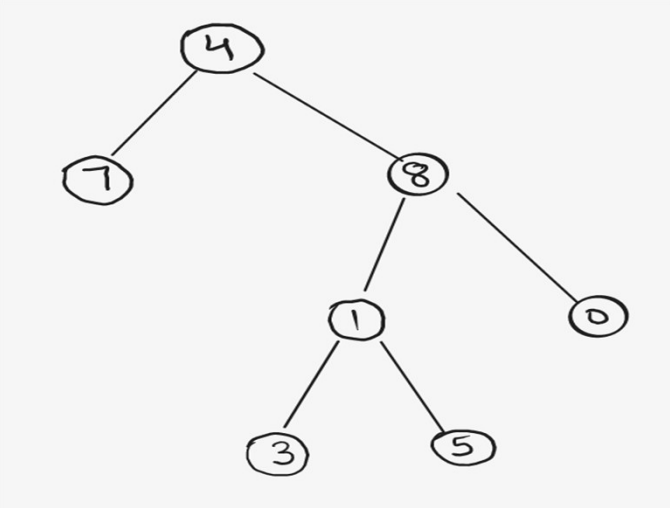
Postorder Traversal:

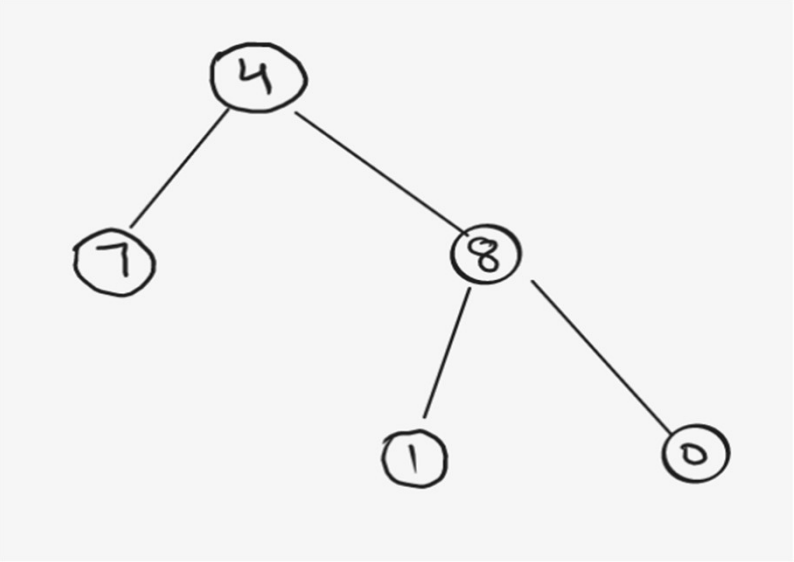
[left, right, root]

4	8	5	2	6	9	10	7	3	1

Binary Search Tree

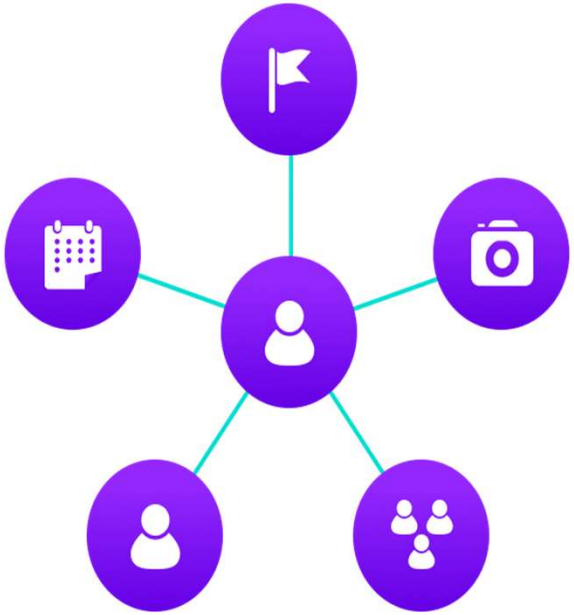
- Nodes to the left of the root are lesser than the root node and the nodes to the right of the root node are greater than the root node. A binary tree is balanced if, for all nodes in the tree, the difference between left and right subtree height is not more than 1
  left Node Values < root Node Value < Right Node Values





Graph

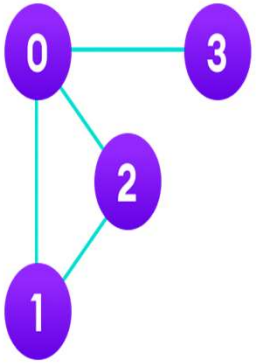
A graph data structure is a collection of nodes that have data and are connected to other nodes.



Graph

A graph is a data structure (V, E) that consists of

- A collection of vertices V
- A collection of edges E, represented as ordered pairs of vertices (u,v)




```
V = {0, 1, 2, 3}
E = {(0,1), (0,2), (0,3), (1,2)}
G = {V, E}
```

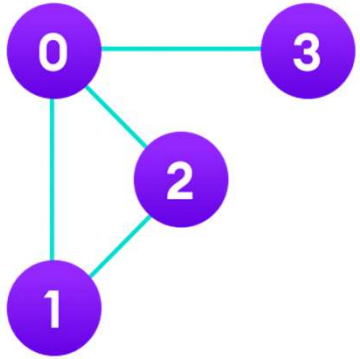
•**Adjacency**: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.

•**Path**: A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.

•**Directed Graph**: A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

Graph Representation: Adjecency Matrix

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.
If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.



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

Graph Representation: Adjecency List

An adjacency list represents a graph as an array of linked lists.
The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.