Bubble Sort

```
for(int turn=0; turn<arr.length-1; turn++) {
    for(int j=0; j<arr.length-1-turn; j++) {
        if(arr[j] > arr[j+1]) {
            //swap
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

```java
    Scanner sc = new Scanner(System.in);
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            matrix[i][j] = sc.nextInt();
        }
    }

public static String substring(String str, int si, int ei) {
    String substr = "";
    for(int i=si; i<ei; i++) {
        substr += str.charAt(i);
    }
    return substr;
}
```

```java
public static void main(String args[]) {
    //Substring
    String str = "HelloWorld";
    System.out.println(str.substring(0, 5));
    //System.out.println(substring(str, 0, 5));
}

public static void main(String args[]) {
    StringBuilder sb = new StringBuilder("");
    for(char ch='a'; ch<='z'; ch++) {
        sb.append(ch);
    }

    System.out.println(sb);
}
```

Access Modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	

Getters & Setters

Get : to return the value

Set : to modify the value

this : this keyword is used to refer to the current object

Encapsulation

Encapsulation is defined as the wrapping up of data & methods under a single unit. It also implements data hiding.

Constructors

Constructor is a special method which is invoked automatically at the time of object creation.

- Constructors have the same name as class or structure.
- Constructors don't have a return type. (Not even void)
- Constructors are only called once, at object creation.
- Memory allocation happens when constructor is called.

Inheritance

Inheritance is when properties & methods of base class are passed on to a derived class.

Polymorphism

- Compile Time Polymorphism
  - Method Overloading

- Run Time Polymorphism
  - Method Overriding

Method Overloading

Multiple functions with the same name but different parameters

Method Overriding

Parent and child classes both contain the same function with a different definition.

Abstraction

Hiding all the unnecessary details and showing only the important parts to the user.

Abstract Classes

Interfaces

abstract class A {

}

Abstract Class

- Cannot create create an instance of abstract class

- Can have abstract/non-abstract methods

- Can have constructors

Interfaces

Multiple Inheritance

Interface is a blueprint of a class

Interfaces

- All methods are public, abstract & without implementation

- Used to achieve total abstraction

- Variables in the interface are final, public and stati

```java
public static void mergeSort(int arr[], int si, int ei) {
    if(si >= ei) {
        return;
    }
    int mid = si + (ei - si)/2; // or = (si + ei) / 2;
    mergeSort(arr, si, mid);
    mergeSort(arr, mid+1, ei);

    merge(arr, si, mid, ei);
}
```

```java
public static void merge(int arr[], int si, int
    int temp[] = new int[ei-si+1];
    int i = si; //idx for 1st sorted part
    int j = mid+1; //idx for 2nd sorted part
    int k = 0; //idx for temp;

    while(i <= mid && j <= ei) {
        if(arr[i] < arr[j]) {
            temp[k] = arr[i];
            i++;
        } else {
            temp[k] = arr[j];
            j++;
        }
        k++;
    }

    //for leftover elements of 1st sorted part
    while(i <= mid) {
        temp[k++] = arr[i++];
    }

    //for leftover elements of 2nd sorted part
    while(j <= ei) {
        temp[k++] = arr[j++];
    }
```

Introduction to ArrayLists

Array

fixed size

primitive data types
can be stored

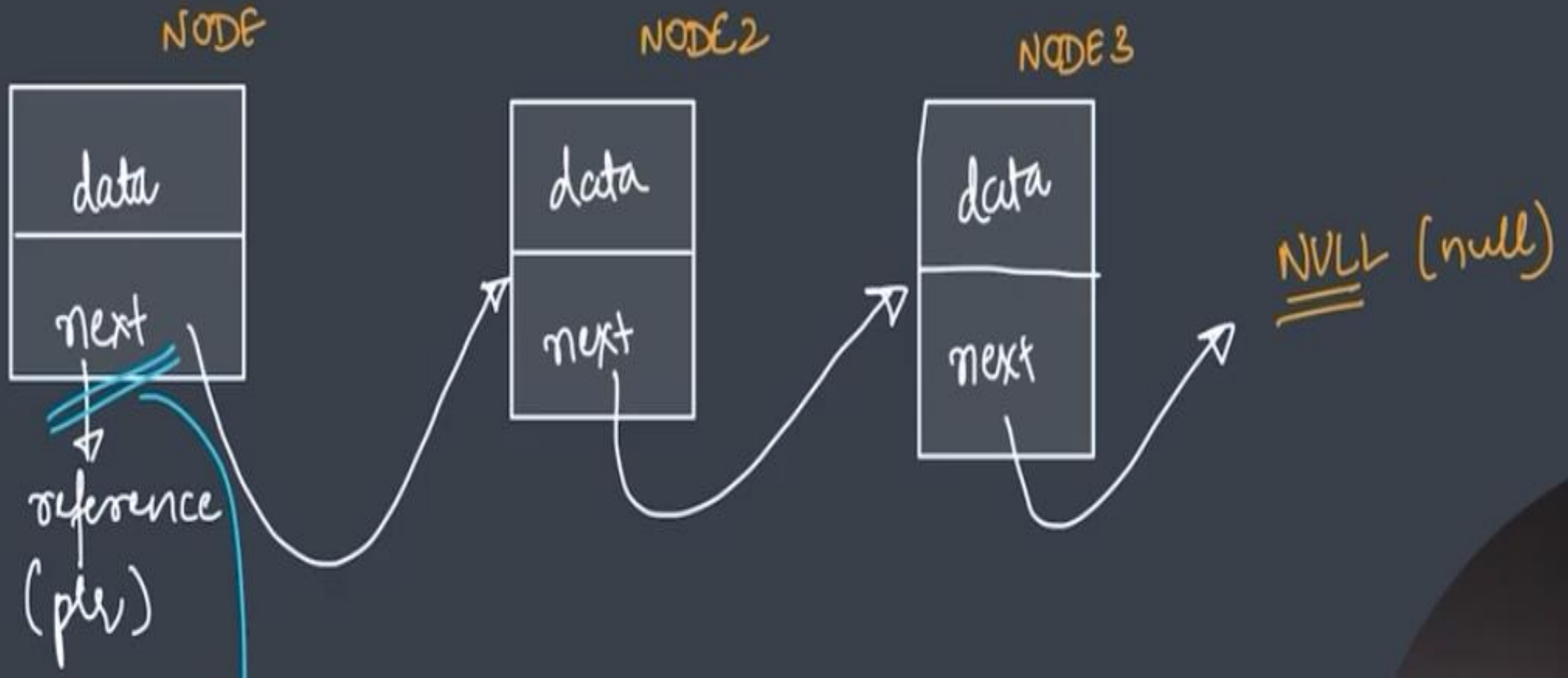
ArrayList

dynamic size

primitive data types
can't be stored directly

```java
import java.util.ArrayList;


public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        //Java Collection Framework
        // ClassName objectName = new ClassName();
        ArrayList<Integer> list = new ArrayList<>();
        ArrayList<String> list2 = new ArrayList<>();
        ArrayList<Boolean> list3 = new ArrayList<>();
    }
}
```

What is a Linked List?



```java
public class LinkedList {
    public static class Node {
        int data;
        Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;

    Run | Debug
    public static void main(String args[]) {
        LinkedList ll = new LinkedList();
        ll.head = new Node(1);
        ll.head.next = new Node(2);
    }
}
```

```java
import java.util.LinkedList; //JCF

public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        //create -
        LinkedList<Integer> ll = new LinkedList<>();

        //add
        ll.addLast(1);
        ll.addLast(2);
        ll.addFirst(0);
        //0->1->2
        System.out.println(ll);
        //remove
        ll.removeLast();
        ll.removeFirst();
        System.out.println(ll);
    }
}
```

```java
public class StackB {
    Run | Debug
    public static void main(String args[]) {
        //Stack s = new Stack();
        Stack<Integer> s = new Stack<>();
        s.push(1);
        s.push(2);
        s.push(3);

        while(!s.isEmpty()) {
            System.out.println(s.peek());
            s.pop();
        }
    }
}
```

```java
public class QueueB {
    Run | Debug
    public static void main(String args[]) {
        //Queue q = new Queue();
        Queue<Integer> q = new LinkedList<>();
        q.add(1);
        q.add(2);
        q.add(3);

        while(!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }

    }
}
```

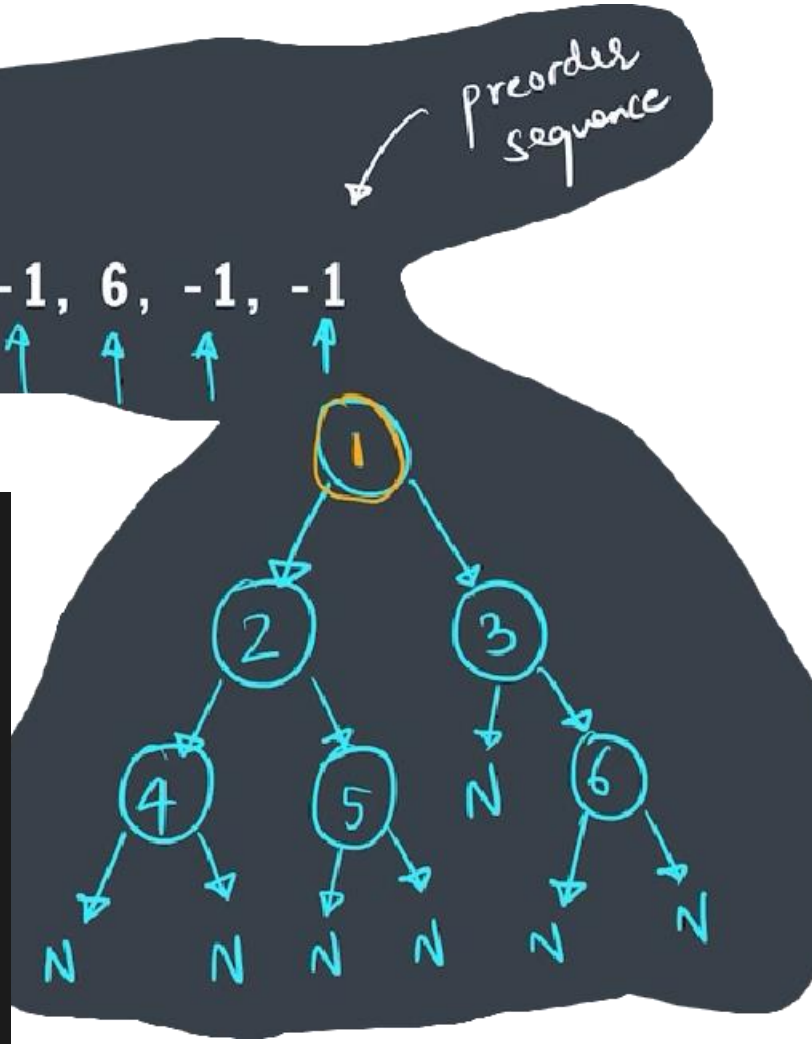
BINARY TREE

Hierarchical

Data Structure

Build Tree Preorder

1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1

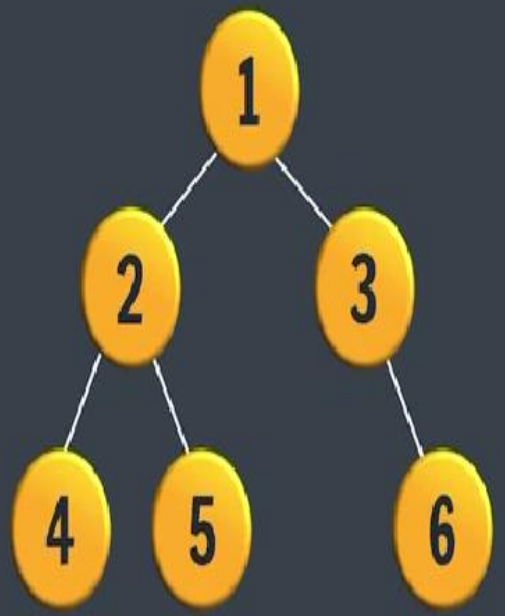



```java
static class BinaryTree {
    static int idx = -1;
    public static Node buildTree(int nodes[]) {
        idx++;
        if(nodes[idx] == -1) {
            return null;
        }

        Node newNode = new Node(nodes[idx]);
        newNode.left = buildTree(nodes);
        newNode.right = buildTree(nodes);

        return newNode;
    }
}
Run | Debug
public static void main(String args[]) {
    int nodes[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6,
    BinaryTree tree = new BinaryTree();
    Node root = tree.buildTree(nodes);
    System.out.println(root.data);
```

```java
public class BinaryTreesB {
    static class Node {
        int data;
        Node left;
        Node right;

        Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
```

Preorder

1 2 4 5 3 6



Root

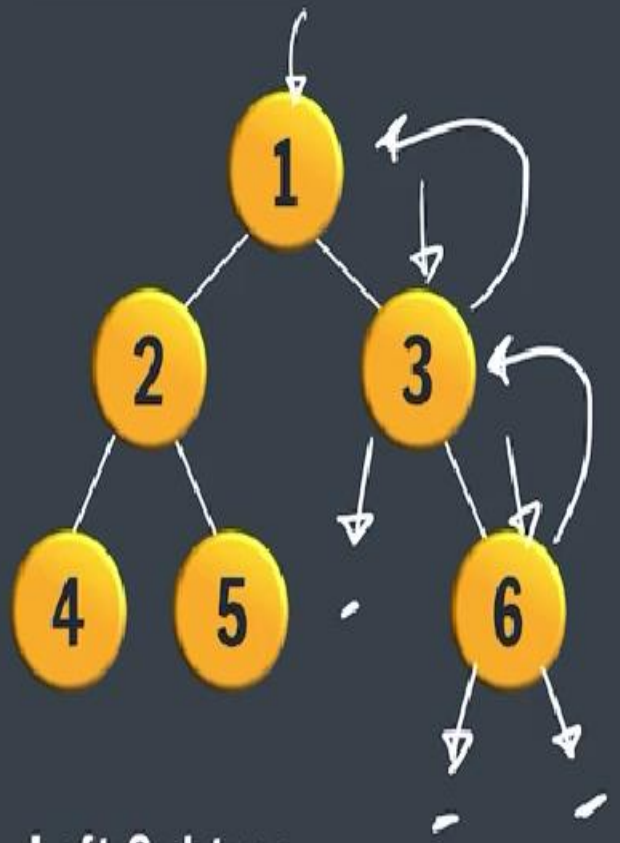
Left Subtree

Right Subtree

```
public static void preorder(Node root) {
    if(root == null) {
        return;
    }
    System.out.print(root.data+" ");
    preorder(root.left);
    preorder(root.right);
}
}
```

Inorder

4 2 5 1 3 6



Left Subtree

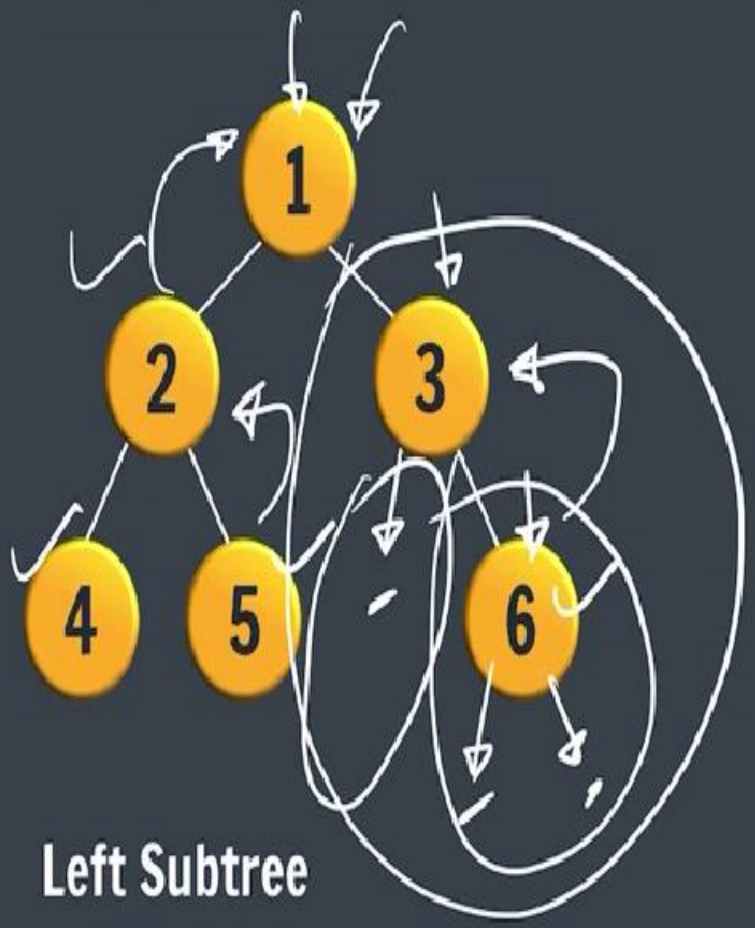
Root

Right Subtree

```java
public static void inorder(Node root) {
    if(root == null) {
        return;
    }
    inorder(root.left);
    System.out.print(root.data+" ");
    inorder(root.right);
}
```

Postorder

4 5 2 6 3 1



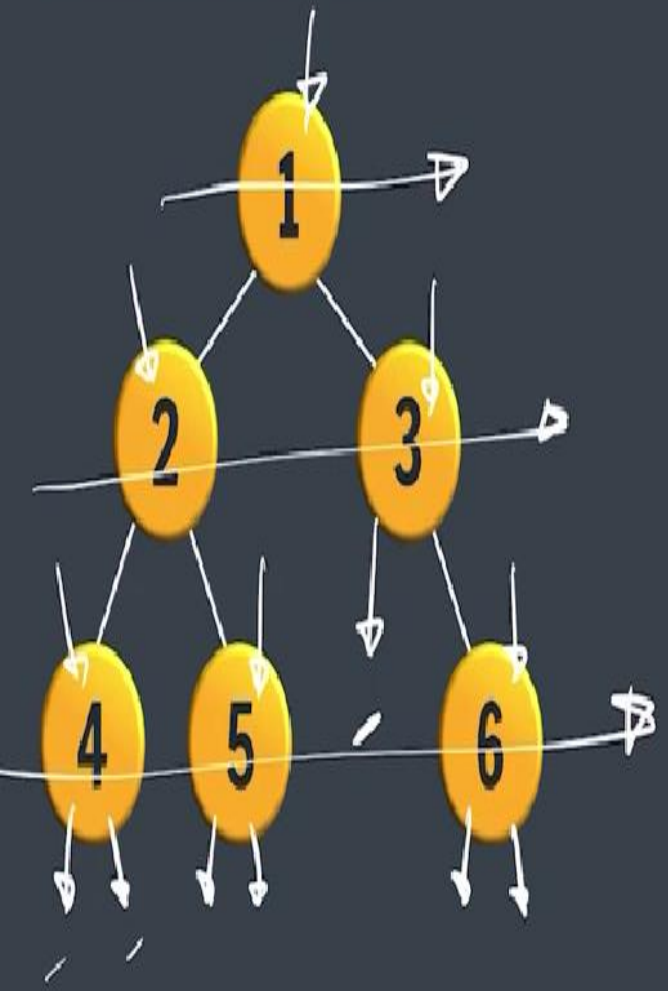
Left Subtree

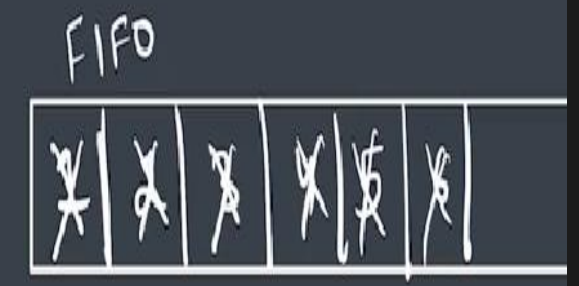
Right Subtree

Root

```java
public static void postorder(Node root) {
    if(root == null) {
        return;
    }
    postorder(root.left);
    postorder(root.right);
    System.out.print(root.data+" ");
}
}
```

Level Order





1 2 3 4 5 6

```java
//Level Order Traversal
public static void levelOrder(Node root) {
    if(root == null) {
        return;
    }

    Queue<Node> q = new LinkedList<>();
    q.add(root);
    q.add(null);

while(!q.isEmpty()) {
    Node currNode = q.remove();
    if(currNode == null) {
        System.out.println();
        if(q.isEmpty()) {
            break;
        } else {
            q.add(null);
        }
    } else {
        System.out.print(currNode.data+" ");
        if(currNode.left != null) {
            q.add(currNode.left);
        }
        if(currNode.right != null) {
            q.add(currNode.right);
        }
    }
}
```

What is a BST?

*Binary Tree

a. Left Subtree Nodes < Root

b. Right Subtree Nodes > Root

c. Left & Right Subtrees are also BST with no duplicates

Special Property

Inorder Traversal of BST gives a sorted sequence

```java
import java.util.PriorityQueue;

public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        pq.add(e: 3); //O(logn)
        pq.add(e: 4);
        pq.add(e: 1);
        pq.add(e: 7);

        while(!pq.isEmpty()) {
            System.out.println(pq.peek()); //O(1)
            pq.remove(); //O(logn)
        }
    }
}
```

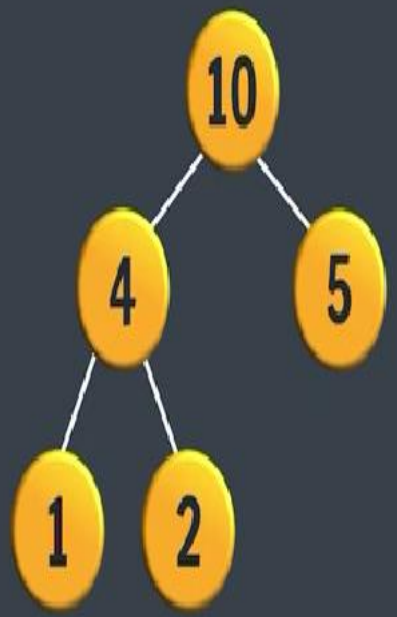
DEBUG CONSOLE | PROBLEMS 1 | OUTPUT | TERMINAL | JUPYTER

```
shradhakhapra@Shradhas-MacBook-Air Heaps % java Classroom.java
1
3
```

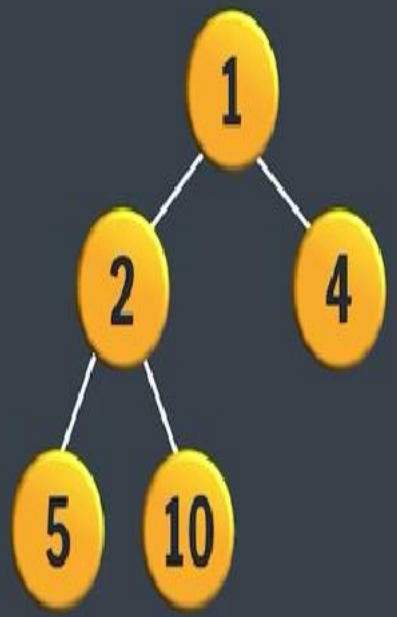
```java
import java.util.PriorityQueue;

public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        //1, 2, 3, 4, 5
        PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.reverseOrder());

        pq.add(e: 3); //O(logn)
        pq.add(e: 4);
        pq.add(e: 1);
        pq.add(e: 7);

        while(!pq.isEmpty()) {
            System.out.println(pq.peek()); //O(1)
            pq.remove(); //O(logn)
        }
    }
}
```

```java
import java.util.PriorityQueue;

public class Classroom {
    static class Student implements Comparable<Student> { //overriding
        String name;
        int rank;

        public Student(String name, int rank) {
            this.name = name;
            this.rank = rank;
        }

        @Override
        public int compareTo(Student s2) {
            return this.rank - s2.rank;
        }
    }
```

Heap

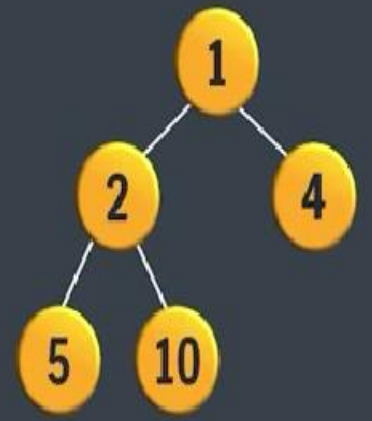


Max heap



Min heap

Heap



Binary Tree

at most 2 children

Complete Binary Tree

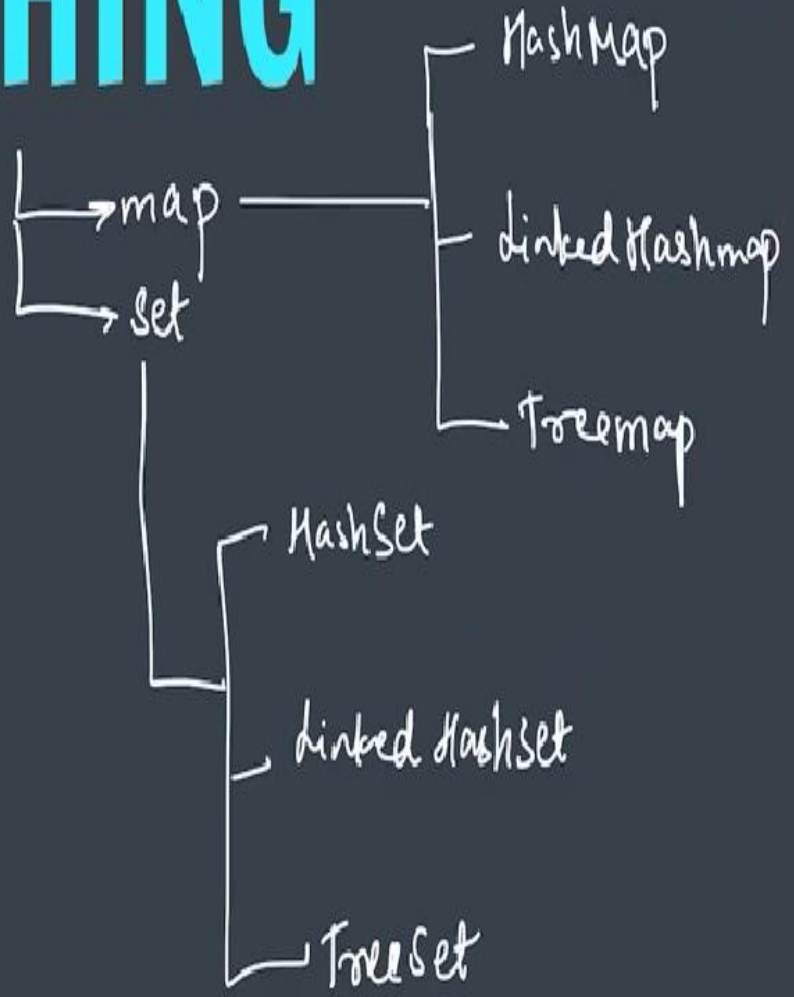
CBT is a BT in which all the levels are completely filled except possibly the last one, which is filled from the left to right.

Heap Order Property

Children >= Parent (MIN Heap

Children <= Parent (maxHeap)

#HASHING



HashMap

(key, value)

unique

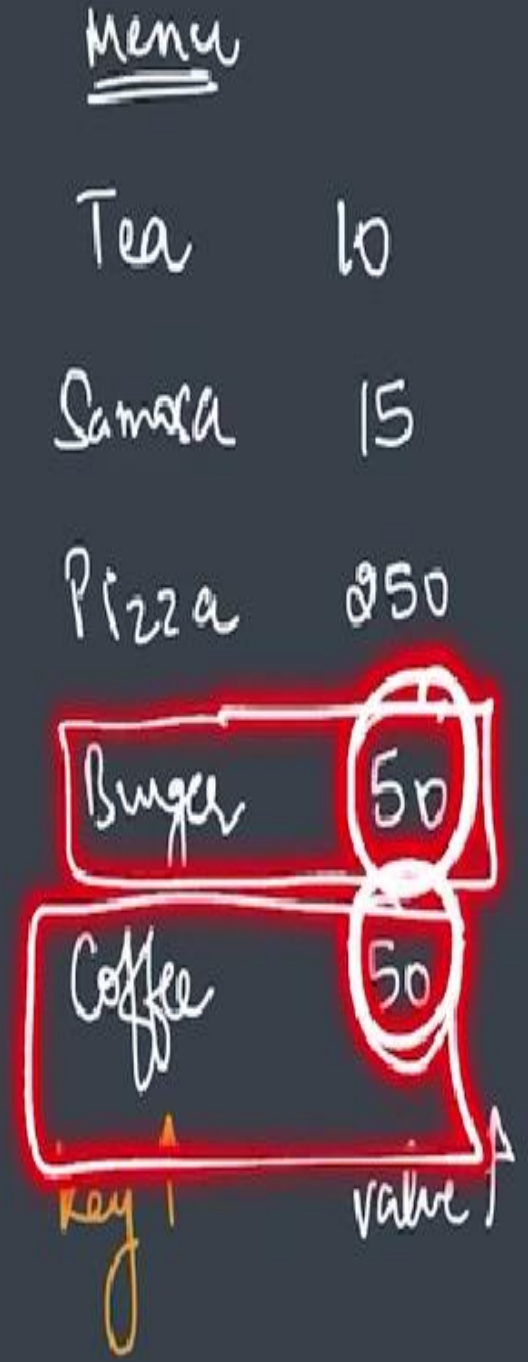
Menu

Tea	10
Samosa	15
Pizza	250

HashMap

(key, value)

unique




```java
import java.util.*;

public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        //Create
        HashMap<String, Integer> hm = new HashMap<>();

        //Insert
        hm.put(key: "India", value: 100);
        hm.put(key: "China", value: 150);
        hm.put(key: "US", value: 50);

        System.out.println(hm);
    }
}
```

DEBUG CONSOLE PROBLEMS 6 OUTPUT TERMINAL JUPYTER

```
hradhakhapra@Shradhas-MacBook-Air Hashing % java Classroom.java
{China=150, US=50, India=100}
```

LinkedHashMap

keys are insertion ordered

LinkedHashMap<K,V> hm = new LinkedHashMap<>();

put
get
remove
size
isEmpty
keySet()

Same Time complexity

```java
public static void main(String args[]) {
    LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();
    lhm.put(key: "India", value: 100);
    lhm.put(key: "China", value: 150);
    lhm.put(key: "US", value: 50);

    LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();
    lhm.put(key: "India", value: 100);
    lhm.put(key: "China", value: 150);
    lhm.put(key: "US", value: 50);

    System.out.println(lhm);
}
```

Tree Map

keys are sorted

put, get, remove are O(logn)

TreeMap< > hm = new TreeMap<>();

Property	Hash Map	Linked Hash Map	Tree Map
Time Complexity (Big O notation) Get, Put, Contains Key and Remove method	O(1)	O(1)	O(1)
Iteration Order	Random	Sorted according to either Insertion Order of Access Order (as specified during construction)	Sorted according to either natural Order of keys or comparator(as specified during construction)
Null Keys	allowed	allowed	Not allowed if keys uses Natural Ordering or Comparator does not support comparison on null Keys.
Interface	Map	Map	Map, Sorted Map and Navigable Map
Synchronization	None, use Collections. Synchronized Map()	None, use Collections. Synchronized Map()	None, use Collections. Synchronized Map()
Data Structure	List of buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black(a kind of self-balancing binary search tree) implementation of Binary Tree. This data structure offers O(log n) for insert, Delte and Search operations and O(n) space complexity.
Applications	General Purpose, fast retrieval, non-synchronized. Concurrent Hash Map can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of employees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hash Code() needs to be overwritten.	Equals() and hash Code() needs to be overwritten.	Comparator needs to be supplied for key implementation, otherwise natural order will be used to sort the keys.

HashSet

- no duplicates
- unordered
- NULL is allowed

HashSet Operations

import java.util.HashSet.*;

```
HashSet<Integer> hs = new HashSet<>();
```

add(key)	O(1)
contains(key)	O(1)
remove(key)	O(1)

```java
import java.util.*;

public class Classroom {
    Run | Debug
    public static void main(String args[]) {
        HashSet<Integer> set = new HashSet<>();

        set.add(e: 1);
        set.add(e: 2);
        set.add(e: 4);
        set.add(e: 2);
        set.add(e: 1);

        System.out.println(set);
    }
}
```


DEBUG CONSOLE PROBLEMS 2 OUTPUT TERMINAL JUPYTER: VARIABLES

```
shradhakhapra@Shradhas-MacBook-Air Hashing % java Classroom.java
[1, 2, 4]
```

Linked HashSet

Ordered using DLL

```
LinkedHashSet<String> lhs = new LinkedHashSet<>();
lhs.add(e: "Delhi");
lhs.add(e: "Mumbai");
lhs.add(e: "Noida");
lhs.add(e: "Bengaluru");
System.out.println(lhs);
```

TreeSet

Sorted in ascending order

NULL values are NOT allowed

```
4       public static void main(String args[]) {
5           HashSet<String> cities = new HashSet<>();
6           cities.add(e: "Delhi");
7           cities.add(e: "Mumbai");
8           cities.add(e: "Noida");
9           cities.add(e: "Bengaluru");
10          System.out.println(cities);
11
```

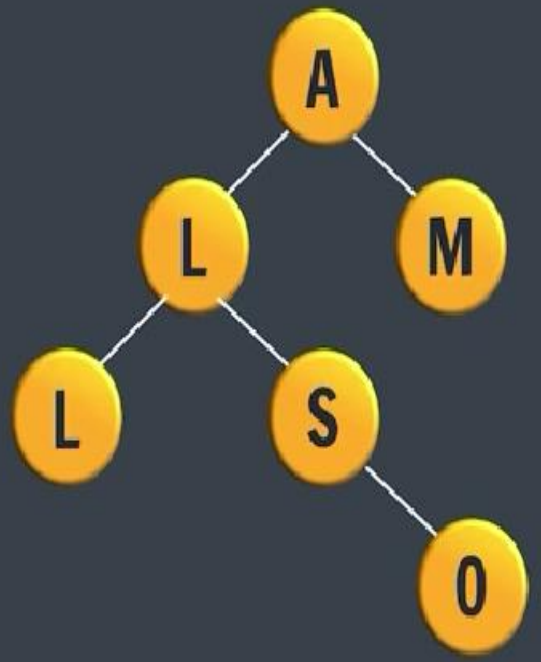
DEBUG CONSOLE PROBLEMS 2 OUTPUT TERMINAL JUPYTER: V

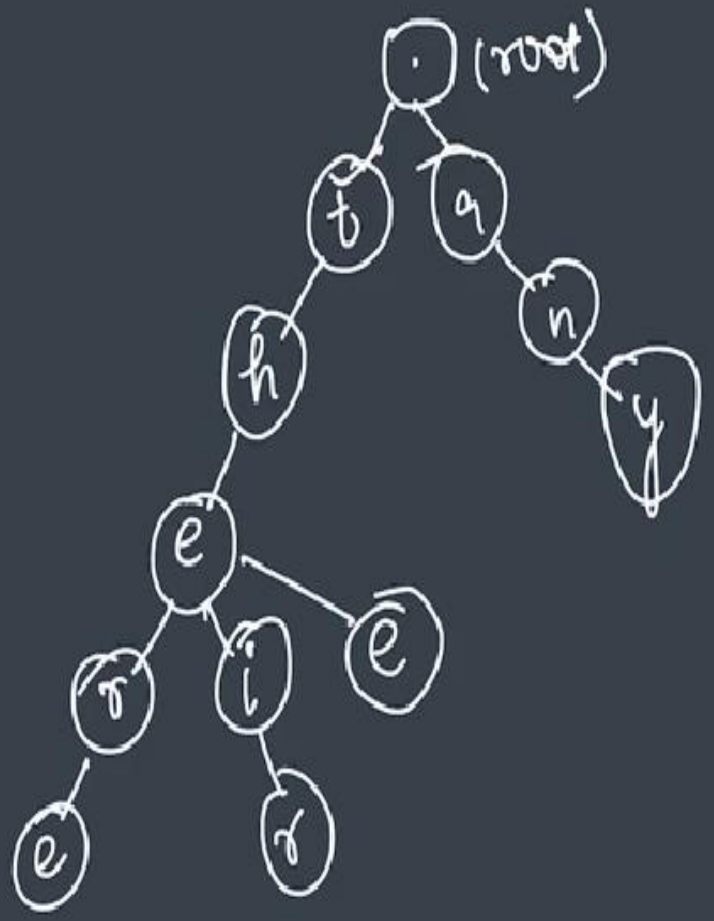
```
shradhakhapra@Shradhas-MacBook-Air Hashing % java Classroom.jav
[Delhi, Bengaluru, Noida, Mumbai]
[Delhi, Mumbai, Noida, Bengaluru]
[Bengaluru, Delhi, Mumbai, Noida]
```

What is a Trie?

Trie Data Structure

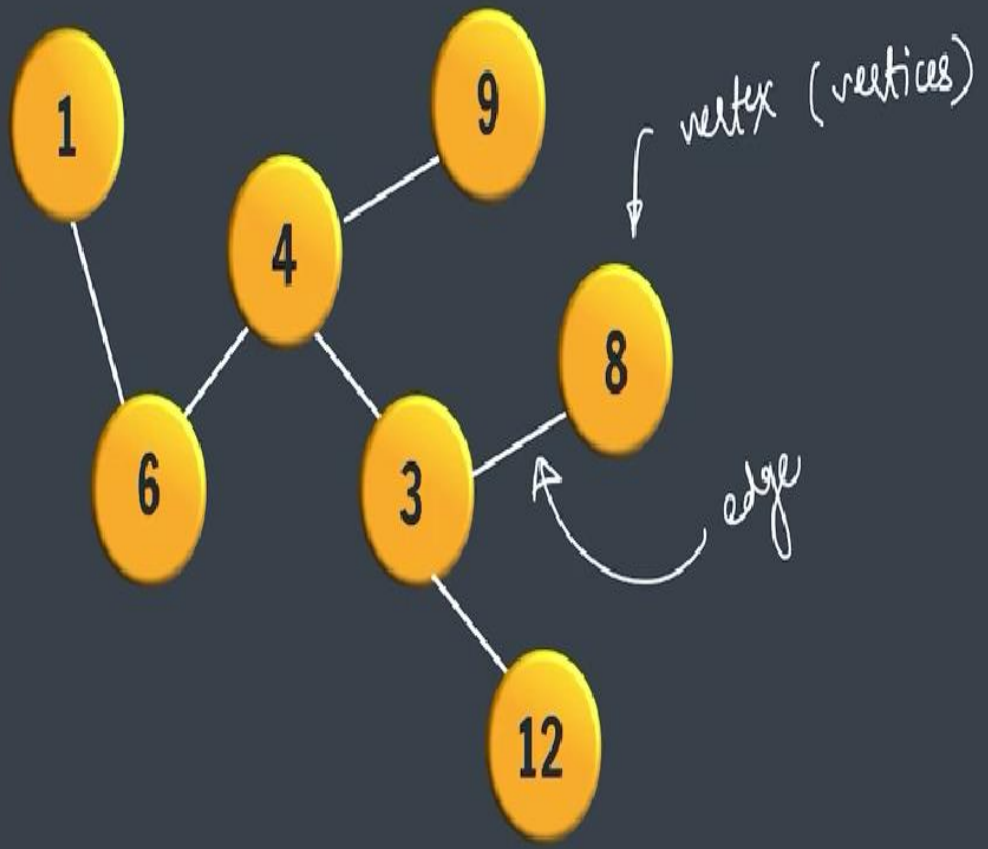
words[ ] = "the", "a", "there", "their", "any", "thee"





GRAPHS

└ network of nodes



Edges

① Uni-Directional

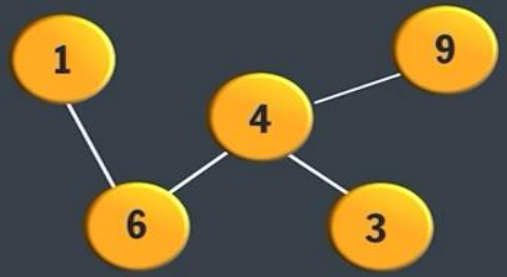
② Un-Directed

② Bi-Directional

Types

Weighted

UnWeighted



Storing a Graph — structure

Adjacency List

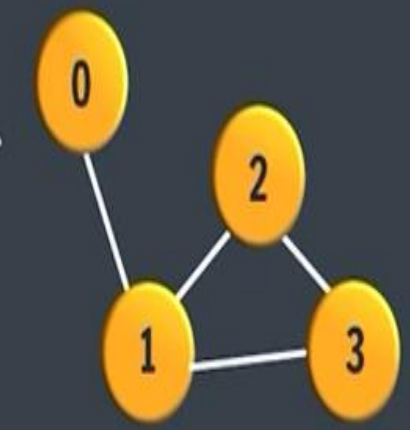
Adjacency Matrix

Edge List

2D Matrix (Implicit Graph)

Adjacency List

List of Lists



0 → (1) dist[0]

1 → (0, 2, 3) dist[1]

2 → (1, 3) dist[2]

3 → (1, 2) dist[3]

ArrayList <ArrayList>

Array <ArrayList>

hashmap <int, lists>

vertex (key)

(value)

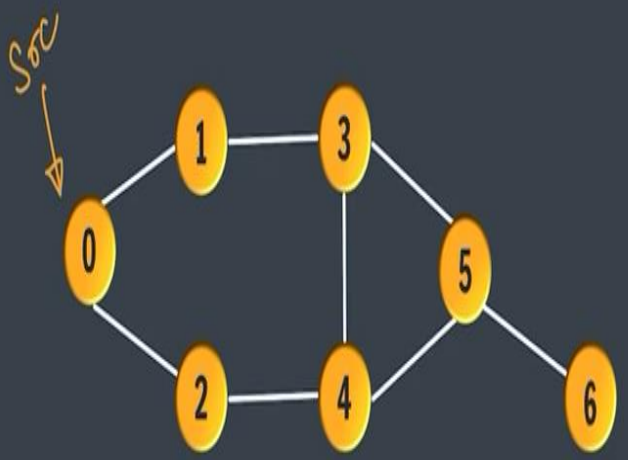
```java
import java.util.*;

public class Classroom {
    static class Edge {
        int src;
        int dest;
        int wt;

        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }
```

```
            (5)
        0 ------ 1
          |    /  \
          |  (1)/    \ (3)
          |   /       \
          |  2 ------ 3
          |  |    (1)
          | (2) |
          |  |
          |  4
*/
```

```java
int V = 5;
ArrayList<Edge>[] graph = new ArrayList[V]; // null -> empty arraylist

for(int i=0; i<V; i++) {
    graph[i] = new ArrayList<>();
}

//0 -vertex
graph[0].add(new Edge(s: 0, d: 1, w: 5));

//1 vertex
graph[1].add(new Edge(s: 1, d: 0, w: 5));
graph[1].add(new Edge(s: 1, d: 2, w: 1));
graph[1].add(new Edge(s: 1, d: 3, w: 3));

//2 vertex
graph[2].add(new Edge(s: 2, d: 1, w: 1));
graph[2].add(new Edge(s: 2, d: 3, w: 1));
graph[2].add(new Edge(s: 2, d: 1, w: 1));
```

BFS (Breadth First Search)

Go to immediate neighbors firs



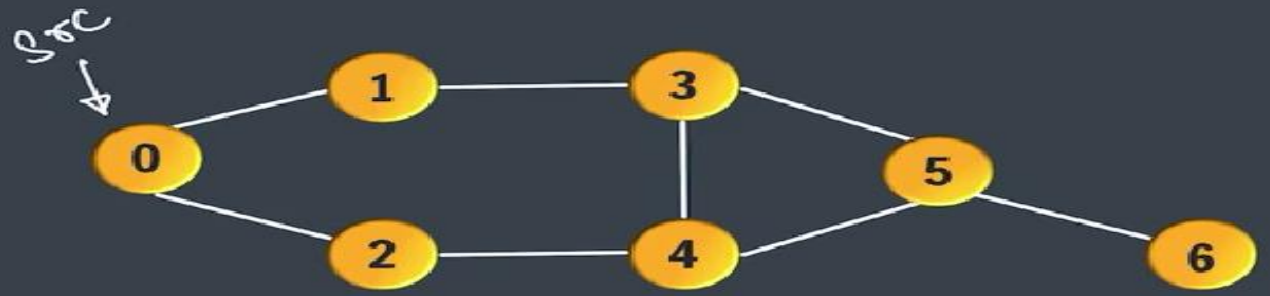
visited

F T

each node is visited once

```java
public static void bfs(ArrayList<Edge>[] graph) { //O(V+E)
    Queue<Integer> q = new LinkedList<>();
    boolean vis[] = new boolean[graph.length];
    q.add(e: 0); //source = 0

    while(!q.isEmpty()) {
        int curr = q.remove();

        if(!vis[curr]) { //visit curr
            System.out.print(curr+" ");
            vis[curr] = true;
            for(int i=0; i<graph[curr].size(); i++) {
                Edge e = graph[curr].get(i);
                q.add(e.dest);
            }
        }
    }
}
```

DFS (Depth First Search)



Keep going to the 1st neighbor

Recursion

① curr → visit

② for (int i=0 to K)
(! vis[curr])
dfs(neighbour)

```java
public static void dfs(ArrayList<Edge>[] graph, int curr, boolean vis[]) {
    //visit
    System.out.print(curr + " ");
    vis[curr] = true;

    for(int i=0; i<graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if(!vis[e.dest]) {
            dfs(graph, e.dest, vis);
        }
    }
}
```

What is DP?

DP is optimized recursion

How to identify DP?

a. Optimal Problem

b. some choice is given (multiple branches in recursion tree)

patterns

Ways of DP

1. Memoization (Top Down)

fibonacci

1. Recursion
2. subproblems → storage → reuse

2. Tabulation (Bottom Up) — Iteration

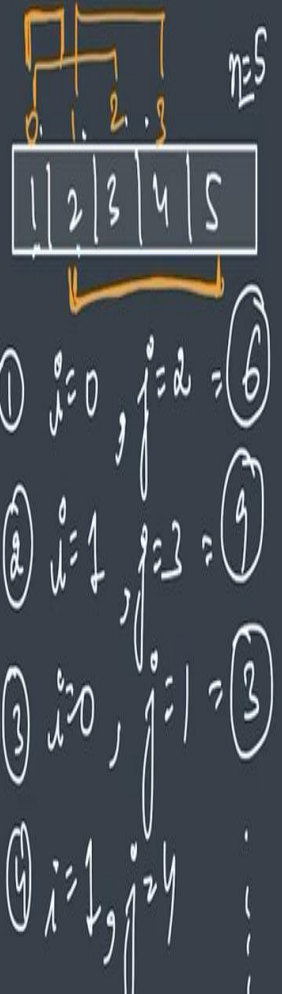
efficient

fib[]

recursion

Segment Trees

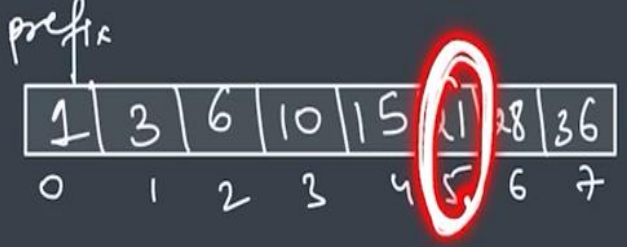
Range Queries



Why do we need Segment Trees?

arr: 1, 2, 3, 4, 5, 6, 7, 8 n=8

Approach2: PrefixSum



① Query → i=2, j=5

prefix[j] − prefix[i−1]

	Query	Update
	O(n)	O(1)

Why do we need Segment Trees?

arr: 1, 2, 3, 4, 5, 6, 7, 8

queries 10^5 updates 10^5

* O(log n) O(log n)

	Query	Update
①	O(n)	O(1)
②	O(1)	O(n)