1) What do you mean by a Data structure?

Ans: -Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage.

Q2. What are some of the applications of DS?

Ans: -Some of applications of DS are:

Array, Stack, Tree, Graph, Linked List.

3) What are the advantages of a Linked list over an array?

Ans: -The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk.

4) Write the syntax in C to create a node in the singly linked list.

Ans: -

```
        Struct Node
{
        int data;
        Node *next;
};
```

5) What is the use of a doubly-linked list when compared to that of a singly linked list?

Ans: -Doubly linked list allows element two way traversal. On other hand doubly linked list can be used to implement stacks as well as heaps and binary trees. Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored.

6) What is the difference between an Array and Stack?

Ans: - Array is a Linear Data Structure in which insertion and deletion can take place in any position. The elements can be retrieved randomly in arrays. Arrays have fixed size and contains only elements of same data type.

A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list. It follows LIFO (Last in First Out) principle. It has dynamic size and can contain elements of different data types.

7) What are the minimum number of Queues needed to implement the priority queue?

Ans: -Two queues are required to implement a priority queue. One queue stores the data and the other one stores the priorities.

8) What are the different types of traversal techniques in a tree?

Ans: -There are basically three traversal techniques for a binary tree that are,

1) Preorder Traversal: - Preorder traversal will create a copy of the tree. Preorder Traversal is also used to get the prefix expression of an expression.

1) Inorder Traversal: - For binary search trees (BST), Inorder Traversal specifies the nodes in non-descending order. In order to obtain nodes from BST in non-increasing order, a variation of inorder traversal may be used where inorder traversal is reversed.
2) Postorder Traversal: - Postorder traversal is used to get the postfix expression of an given expression.

9) Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?

Ans: -It is said that searching a node in binary search tree is easier than searching it in a simple tree because BSTs have nodes in certain order. The element smaller than root element goes to its left side whereas the element greater than it goes to its right side. Hence the process become easier.

10) What are the applications of Graph DS?

Ans: -

- **Google maps** uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
- In **Facebook**, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook's Friend suggestion algorithm uses graph theory. Facebook is an example of undirected graph.
- In **Operating System**, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle, then a deadlock will occur.

11) Can we apply Binary search algorithm to a sorted Linked list?

Ans: -No, it is possible to apply binary search algorithm to a sorted Linked list.

12) When can you tell that a Memory Leak will occur?

Ans: -A memory leak is any portion of an application which uses memory without eventually freeing it. By memory, we're talking about RAM, not permanent storage, like a hard drive.
A memory leak is caused when you allocated memory, haven't yet deallocated it, and you will never be able to deallocate it because you can't access it anymore.

13) How will you check if a given Binary Tree is a Binary Search Tree or not?

Ans: -

1) If a node is a left child, then its key and the keys of the nodes in its right subtree are less than its parent's key.
2) If a node is a right child, then its key and the keys of the nodes in its left subtree are greater than its parent's key.

14) Which data structure is ideal to perform recursion operation and why?

Ans: -STACK is ideal option to perform recursion operation as every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written explicit, stack is to be used.

15) What are some of the most important applications of a Stack?

Ans: -

- Memory Management
- Can be used for backtracking Problems
- For parenthesis Balancing
- For implementing Recrussion

17)Sorting a stack using a temporary stack

Ans: -

```
import java.util.*;
class SortStack
{
   public static Stack<Integer> sortstack(Stack<Integer>input)
   {
      Stack<Integer> tmpStack = new Stack<Integer>();
      while(!input.isEmpty())
      {
         int tmp = input.pop();
         while(!tmpStack.isEmpty() && tmpStack.peek()
                                    > tmp)
         {
          input.push(tmpStack.pop());
```

```
            }
            tmpStack.push(tmp);
        }
        return tmpStack;
    }
    public static void main(String args[])
    {
        Stack<Integer> input = new Stack<Integer>();
        input.add(65);
        input.add(45);
        input.add(41);
        input.add(77);
        input.add(82);
        input.add(29);
        Stack<Integer> tmpStack=sortstack(input);
        System.out.println("Sorted numbers are:");

        while (!tmpStack.empty())
        {
            System.out.print(tmpStack.pop()+" ");
        }
    }
}
```

18)Program to reverse a queue

Ans: -

```
import java.util.*;

public class Queue_reverse {

    static Queue<Integer> queue;

    static void Print ()
    {
        while (! queue.isEmpty()) {
            System.out.print( queue.peek() + ", ");
            queue.remove();
        }
    }

    static void reversequeue()
    {
        Stack<Integer> stack = new Stack<> ();
        while (! queue.isEmpty()) {
```

```
        stack.add(queue.peek());
        queue.remove();
      }
      while (! stack.isEmpty()) {
        queue.add(stack.peek());
        stack.pop();
      }
   }
    public static void main (String args[])
   {
      queue = new LinkedList<Integer> ();
for (int i=0; i<10;i++)
{
queue.add(i );
}
reversequeue();
      Print();
   }
}
```

19) Program to reverse first k elements of a queue

Ans: -

```
  public Queue<Integer> modifyQueue(Queue<Integer> q, int k)

  {
   Stack<Integer> stk=new Stack<Integer>();
   int temp=k;
   int size=q.size();
   while(--temp>=0)
   {
      stk.push(q.remove());

   }
   while(!stk.isEmpty())
   {
      q.add(stk.pop());
   }
   temp=size-k;
   while(temp-->0)
   {
      q.add(q.remove());
   }
   return q;
  }
```

20)Program to return the nth node from the end in a linked list

Ans: -

```
class Node
{
   int data;
   Node next;
   Node(int d)
   {
      data = d;
      next = null;
   } }
class LinkedList
{
   Node head; //the head of list
   public int GetNth(int index)
   {
      Node current = head;
      int count = 0;
      while (current != null)
      {
         if (count == index)
            return current.data;
         count++;
         current = current.next;
      }
       assert(false);
      return 0;
   }
   public void push(int new_data)
   {
      Node new_Node = new Node(new_data);
      new_Node.next = head;
      head = new_Node;
   }
   public static void main(String[] args)
   {
      LinkedList llist = new LinkedList();

      llist.push(10);
      llist.push(78);
```

```
    llist.push(42);
    llist.push(10);
    llist.push(7);
    System.out.println("Element at index 3 is "+llist.GetNth(3));
  } }
```

21)Reverse a linked list

Ans: -

```
public class ReverseLinkedList {
        public static void main(String[] args) {

                MyLinkedList myLinkedList = new MyLinkedList();

                myLinkedList.head = new Node(1);

                myLinkedList.head.next = new Node(2);

                myLinkedList.head.next.next = new Node(3);

                printLinkedList(myLinkedList);

                reverseLinkedList(myLinkedList);

                printLinkedList(myLinkedList);

        }

        public static void printLinkedList(MyLinkedList linkedList) {

                Node h = linkedList.head;

                while (linkedList.head != null) {

                        System.out.print(linkedList.head.data + " ");

                        linkedList.head = linkedList.head.next;

                }

                System.out.println();

                linkedList.head = h;

        }

        public static void reverseLinkedList(MyLinkedList linkedList) {

                Node previous = null;

                Node current = linkedList.head;

                Node next;
```

```
            while (current != null) {
                next = current.next;
                current.next = previous;
                previous = current;
                current = next;
            }
            linkedList.head = previous;
        }
}
```

22)Replace each element of the array by its rank in the array

Ans: -

```
import java.util.Arrays;
import java.util.Map;
import java.util.TreeMap;
class Main
{
    public static void transform(int[] arr)
    {
        Map<Integer, Integer> map = new TreeMap<>();
        for (int i = 0; i < arr.length; i++) {
            map.put(arr[i], i);
        }
        int rank = 1;
        for (var val : map.values()) {
            arr[val] = rank++;
        }
    }
```

```
	public static void main(String[] args)
	{
		int[] A = { 10, 8, 15, 12, 6, 20, 1 };

		transform(A);
		System.out.println(Arrays.toString(A));
	}
}
```

23) Check if a given graph is a tree or not

Ans: -

```
import java.util.*;
class Edge
{
	int source, dest;

	public Edge(int source, int dest) {
		this.source = source;
		this.dest = dest;
	}
}
class Graph
{
	List<List<Integer>> adjList = null;
	Graph(List<Edge> edges, int N)
	{
		adjList = new ArrayList<>();
		for (int i = 0; i < N; i++) {
```

```
                adjList.add(new ArrayList<>());
            }
            for (Edge edge: edges)
            {
                int src = edge.source;
                int dest = edge.dest;

                adjList.get(src).add(dest);
                adjList.get(dest).add(src);
            }
        }
}

class Main
{
        public static boolean DFS(Graph graph, int v, boolean[] discovered, int parent)
        {
            discovered[v] = true;
            for (int w : graph.adjList.get(v))
            {
                if (!discovered[w]) {
                    if (!DFS(graph, w, discovered, v))
                        return false;
                }
                else if (w != parent) {
                    return false;
                }
```

```
        }
        return true;
}
public static void main(String[] args)
{
        // List of graph edges as per above diagram
        List<Edge> edges = Arrays.asList(
                        new Edge(0, 1), new Edge(1, 2), new Edge(2, 3),
                        new Edge(3, 4), new Edge(4, 5), new Edge(5, 0)
        );

        final int N = 6;
        Graph graph = new Graph(edges, N);
        boolean[] discovered = new boolean[N];
        boolean isTree = DFS(graph, 0, discovered, -1);

        for (int i = 0; isTree && i < N; i++) {
                if (!discovered[i]) {
                        isTree = false;
                }
        }
        if (isTree) {
                System.out.println("Graph is a Tree");
        }
        else {
                System.out.println("Graph is not a Tree");
        }
```

```
        }
}
```

24) Find out the Kth smallest element in an unsorted array

Ans: -

```
import java.util.*;
class KthElement{
   public String sorting(int arr[]){
      Arrays.sort(arr);
      return "Array after sorting :: "+Arrays.toString(arr);
   }
   public int get(int arr[],int k){
    return arr[k-1];
   }
}
class Main{
   public static void main(String Arg[])
   {
      KthElement kob = new KthElement();
      Scanner scanner = new Scanner(System.in);
      System.out.println("\n Enter array size ::\n");
      int n = scanner.nextInt();
      int arr[] = new int[n];
```

```
        System.out.println("\n Enter array elements ::\n");

        for(int i=0;i<n;i++){

           arr[i] = scanner.nextInt();

        }

        System.out.println("\n Array is :: "+Arrays.toString(arr));

        System.out.println("\n "+kob.sorting(arr));

        System.out.println("\n Enter the k value :: \n");

        int k = scanner.nextInt();

        System.out.println("\n K value is :: "+k);

        System.out.println("\n"+k+"th  smallest element is  :: "+ko
b.get(arr , k));

    }

}
```

25) How to find the shortest path between two vertices

Ans: -

```
import java.util.PriorityQueue;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

class Vertex implements Comparable<Vertex>
{
   public final String name;
   public Edge[] adjacencies;
   public double minDistance = Double.POSITIVE_INFINITY;
   public Vertex previous;
   public Vertex(String argName) { name = argName; }
   public String toString() { return name; }
   public int compareTo(Vertex other)
   {
      return Double.compare(minDistance, other.minDistance);
```

```
    }

}
class Edge
{
    public final Vertex target;
    public final double weight;
    public Edge(Vertex argTarget, double argWeight)
    { target = argTarget; weight = argWeight; }
}

public class Dijkstra
{
    public static void computePaths(Vertex source)
    {
        source.minDistance = 0.;
        PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();
        vertexQueue.add(source);

        while (!vertexQueue.isEmpty()) {
            Vertex u = vertexQueue.poll();

            // Visit each edge exiting u
            for (Edge e : u.adjacencies)
            {
                Vertex v = e.target;
                double weight = e.weight;
                double distanceThroughU = u.minDistance + weight;
                if (distanceThroughU < v.minDistance) {
                    vertexQueue.remove(v);

                    v.minDistance = distanceThroughU ;
                    v.previous = u;
                    vertexQueue.add(v);
                }
            }
        }
    }

    public static List<Vertex> getShortestPathTo(Vertex target)
    {
        List<Vertex> path = new ArrayList<Vertex>();
        for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
            path.add(vertex);
```

```
        Collections.reverse(path);
        return path;
    }

    public static void main(String[] args)
    {
        // mark all the vertices
        Vertex A = new Vertex("A");
        Vertex B = new Vertex("B");
        Vertex D = new Vertex("D");
        Vertex F = new Vertex("F");
        Vertex K = new Vertex("K");
        Vertex J = new Vertex("J");
        Vertex M = new Vertex("M");
        Vertex O = new Vertex("O");
        Vertex P = new Vertex("P");
        Vertex R = new Vertex("R");
        Vertex Z = new Vertex("Z");

        // set the edges and weight
        A.adjacencies = new Edge[]{ new Edge(M, 8) };
        B.adjacencies = new Edge[]{ new Edge(D, 11) };
        D.adjacencies = new Edge[]{ new Edge(B, 11) };
        F.adjacencies = new Edge[]{ new Edge(K, 23) };
        K.adjacencies = new Edge[]{ new Edge(O, 40) };
        J.adjacencies = new Edge[]{ new Edge(K, 25) };
        M.adjacencies = new Edge[]{ new Edge(R, 8) };
        O.adjacencies = new Edge[]{ new Edge(K, 40) };
        P.adjacencies = new Edge[]{ new Edge(Z, 18) };
        R.adjacencies = new Edge[]{ new Edge(P, 15) };
        Z.adjacencies = new Edge[]{ new Edge(P, 18) };


        computePaths(A); // run Dijkstra
        System.out.println("Distance to " + Z + ": " + Z.minDistance);
        List<Vertex> path = getShortestPathTo(Z);
        System.out.println("Path: " + path);
    }
}
```