1. What do you mean by a Data structure?

Ans: Data Structure is a way to store and organise data so that it can be easily accessible and we can perform any operation on it very efficiently. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage.

2. What are some of the applications of DS?

Ans: Application of Data Structure is to efficiently solve the problems. We have different data structure for different problem scenario like

- HashSet is used to store unique elements.
- Trie is used for dictionary, pattern matching, autocomplete feature, spell checking etc.
- Stack is undo/redo, evaluation of expressions, even JVM is stack oriented.
- Trees are used to store hierarchical data like file systems.
- Priority Queue is used for process scheduling in kernel.
- Graph is used in the implementation of social media sites, network connection like routers.

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

Ans: Following are the advantages of Linked list over array

- Dynamic size: Size can grow and shrink based on the need.
- Efficient Insertion and Deletion: Insertion and Deletion in Linked List is very efficient as compared to Array as in array we have to shift every element.
- Space efficient: No space is wasted in Linked list.
- Memory allocation: Elements may or may not be stored in consecutive memory locations where as in array it is stored in consecutive memory locations.

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

Ans:

```
struct node
{
  int data;
  struct node *next;
};
```

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

Ans: A doubly-linked list basically has two pointers one for next node and other for previous node. So the main benefit of doubly-linked list over a singly linked list is that we can traverse in both direction (forward and backward) which makes it more facile to perform any operation on it.

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

Ans: The main difference between an array and a stack is the way to insert, access, modify the data. In array we can insert, access, modify the data of any index but in stack we can insert, access, modify the last element only at one end (LIFO). Other than that stack has dynamic size and can store heterogeneous data but array has fixed size and can store only homogeneous data.

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

Ans: We need at least two queues to implement priority queue, one queue is used for sorting priorities while the other queue is used for actual storage of data.

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

Ans: There are four types of tree traversal

- Pre-order Traversal (Root -> Left -> Right)
- In-order Traversal (Left -> Root -> Right)
- Post-order Traversal (Left -> Right Root)
- Level order Traversal (Level wise Left-Right)

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

Ans: Searching a node in a binary search tree is efficient than that of a simple binary tree because in binary search tree data is stored in such a way that left child's data is always smaller than root's data and right child's data is always greater than or equal to root's data. So we can search in O (height_of_tree) complexity while in simple binary tree we have to traverse the whole tree to search any data which is O (n).

10. What are the applications of Graph DS?

Ans: Graph is used everywhere in real life, some of the examples are social media sites like Facebook and LinkedIn, google maps and to allocate resource in operating system.

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

Ans: Yes, Binary search algorithm can be applied to sorted Linked list if count of the nodes is given. But it would be inefficient as we have to traverse a lot because in every strep we have to find the middle element by traversing the list.

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

Ans: Memory Leak occur when we initialize or create memory for the any object by using constructor and forget to call destructor for release or free that memory thread.

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

Ans: If the maximum element of the left subtree is less than the root and minimum element of right subtree is greater than or equal to the node and every subtree is a Binary Search Tree then it is a BST.

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

Ans: Stack is ideal to perform recursive operations as Stack has the LIFO (Last In First Out) property so it remembers it's ‘caller’ function and Therefore, it knows to whom it should return when the function has to return. On the other hand, recursion makes use of the system stack for storing the return addresses of the function calls.

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

Ans: Some most important applications of a Stack are undo/redo operations, evaluation of expressions, recursion, problems like balancing the parenthesis and even JVM is stack oriented.

16. Stack is undo/redo, evaluation of expressions, even JVM is stack oriented.

Ans: No expression is given.

17. Sorting a stack using a temporary stack.

Ans:

```
import java.util.Stack;

public class sortStack {
    public static Stack<Integer> sortStack(Stack<Integer> stack){
        Stack<Integer> tempStack = new Stack<>();
        while(!stack.isEmpty()){
            int cur = stack.pop();
            if (!tempStack.isEmpty() && tempStack.peek() >= cur) {
                while (!tempStack.isEmpty() && tempStack.peek() > cur) {
                    stack.push(tempStack.pop());
                }
            }
            tempStack.push(cur);
        }
        while(!tempStack.isEmpty()){
            stack.push(tempStack.pop());
        }
        return stack;
    }
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(7);
        stack.push(15);
        stack.push(21);
        stack.push(2);
        stack.push(48);
        Stack<Integer> ans=sortStack(stack);
        while (!ans.isEmpty()){
            System.out.print(ans.pop()+" ");
        }
    }
}
```

18. Program to reverse a queue.

Ans:
```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
public class reverseQueue {
    public static Queue<Integer> reverseQueue(Queue<Integer> queue){
        Stack<Integer> temp = new Stack<>();
        while(!queue.isEmpty()){
            temp.push(queue.poll());
        }
        while(!temp.isEmpty()){
            queue.add(temp.pop());
        }
        return queue;
    }
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        for(int i=1;i<=10;i++){
            queue.add(i);
        }
        Queue<Integer> ans= reverseQueue(queue);
        while (!ans.isEmpty()){
            System.out.print(queue.remove()+" ");
        }
    }
}
```

19. Program to reverse first k elements of a queue.

Ans:
```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class reverseTillKth {
    public static Queue<Integer> reverseTillKth(Queue<Integer> q, int k){
        Queue<Integer> ans = new LinkedList<>();
        Stack<Integer> stack = new Stack<>();
        while(k >0 ){
            stack.push(q.remove());
            k--;
        }
        while(!stack.isEmpty()){
            ans.add(stack.pop());
        }
        while(!q.isEmpty()){
            ans.add(q.remove());
        }
        return ans;
    }

    public static void main(String[] args) {
        Queue<Integer> queue= new LinkedList<>();
        for(int i=1;i<=10;i++){
            queue.add(i);
        }
        Queue<Integer> ans= reverseTillKth(queue,5);
        System.out.println(ans);
    }
}
```

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

Ar

```java
class Node<T>{
    T data;
    Node<T> next;
    public Node(T data){
        this.data=data;
        next=null;
    }
}

public class nthNodeFromEnd {
    public static int len (Node<Integer> node){
        int len=0;
        while(node!=null){
            len++;
            node=node.next;
        }
        return len;
    }
    public static Node<Integer> nthNodeFromEnd(Node<Integer> node, int n){
        int lengthFromStart = len(node)-n;
        while(lengthFromStart>0){
            node = node.next;
            lengthFromStart--;
        }
        return node;
    }

    public static void main(String[] args) {
        Node<Integer> node1 = new Node<>(1);
        Node<Integer> node2 = new Node<>(2);
        Node<Integer> node3 = new Node<>(3);
        Node<Integer> node4 = new Node<>(4);
        Node<Integer> node5 = new Node<>(5);
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
        node4.next=node5;
        Node<Integer> head = node1;
        System.out.println(nthNodeFromEnd(head,2).data);

    }
}
```

21. Reverse a linked list.

Ans:

```
class Node<T>{
    T data;
    Node<T> next;
    public Node(T data){
        this.data=data;
        next=null;
    }
}
public class ReverseLL {
    public static Node<Integer> reverseLL(Node<Integer> head){
        if(head==null || head.next==null){
            return head;
        }
        Node<Integer> reversedHead=head.next;
        Node<Integer> smallAns =reverseLL(head.next);
        reversedHead.next=head;
        head.next=null;
        return smallAns;
    }

    public static void main(String[] args) {
        Node<Integer> node1 = new Node<>(1);
        Node<Integer> node2 = new Node<>(2);
        Node<Integer> node3 = new Node<>(3);
        Node<Integer> node4 = new Node<>(4);
        Node<Integer> node5 = new Node<>(5);
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
        node4.next=node5;
        Node<Integer> head = node1;
        Node<Integer> ans = reverseLL(head);
        Node<Integer> temp=ans;
        while (temp!=null){
            System.out.print(temp.data+" ");
            temp=temp.next;
        }

    }

}
```

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

Ans:
```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class arrayRank {
    public static int[] rankArray(int[] arr)
    {
        int temp[] = new int[arr.length];
        for(int i = 0 ; i< arr.length ; i++){
            temp[i] = arr[i];
        }
        Arrays.sort(temp);
        Map<Integer, Integer> rank = new HashMap<>();
        for (int i = 0; i < temp.length; i++) {
            if (rank.get(temp[i]) == null) {
                rank.put(temp[i], i+1);
            }
        }
        for (int i = 0; i < arr.length; i++) {
            arr[i] = rank.get(arr[i]);
        }
        return arr;
    }

    public static void main(String[] args) {
        int[] arr = {25, 16, 2 ,0 , 10};
        int[] ans=rankArray(arr);
        for(int i=0;i<5;i++){
            System.out.print(ans[i]+" ");
        }
    }
}
```

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

Ans: An undirected graph is tree if it has following properties.

- There is no cycle.
- The graph is connected.

```java
import java.util.*;

class isTree {
    private int V;
    private LinkedList<Integer> adj[];
    public isTree(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }
    public static void main(String args[]) {
        isTree g1 = new isTree(5);
        g1.addEdge(1, 0);
        g1.addEdge(0, 2);
        g1.addEdge(0, 3);
        g1.addEdge(3, 4);
        if (g1.isTree())
            System.out.println("Graph is Tree");
        else
            System.out.println("Graph is not Tree");
        isTree g2 = new isTree(5);
        g2.addEdge(1, 0);
        g2.addEdge(0, 2);
        g2.addEdge(2, 1);
        g2.addEdge(0, 3);
        g2.addEdge(3, 4);
        if (g2.isTree())
            System.out.println("Graph is Tree");
        else
            System.out.println("Graph is not Tree");
    }
    public void addEdge(int v, int w) {
        adj[v].add(w);
        adj[w].add(v);
    }
    public Boolean isCyclicUtil(int v, Boolean visited[], int parent) {
        visited[v] = true;
        Integer i;
        Iterator<Integer> it = this.adj[v].iterator();
        while (it.hasNext()) {
            i = it.next();
            if (!visited[i]) {
                if (isCyclicUtil(i, visited, v))
                    return true;
            } else if (i != parent)
                return true;
        }
        return false;
    }
    public Boolean isTree() {
        Boolean visited[] = new Boolean[V];
        for (int i = 0; i < V; i++)
            visited[i] = false;
        if (isCyclicUtil(0, visited, -1))
            return false;
        for (int u = 0; u < V; u++)
            if (!visited[u])
                return false;
        return true;
    }
}
```

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

Ans:

```java
import java.util.PriorityQueue;

public class KthSmallest {
    public static int kthSmallest(int arr[], int k) {
        PriorityQueue<Integer> pq=new PriorityQueue<Integer>();
        for(int i=0;i<arr.length;i++){
            pq.add(arr[i]);
        }
        for(int j=0;j<k-1;j++){
            pq.remove();
        }
        return pq.remove();
    }

    public static void main(String[] args) {
        int[] arr = {798,1720,4524,51,486,257};
        System.out.println(kthSmallest(arr,3));
    }
}
```

25. How to find the shortest path between two vertices.

Ans:

```java
import java.util.ArrayList;
import java.util.*;

public class shortestPath {
    private static void addEdge(ArrayList<ArrayList<Integer>> adj, int i,
int j)
    {
        adj.get(i).add(j);
        adj.get(j).add(i);
    }
    private static void printShortestDistance(
            ArrayList<ArrayList<Integer>> adj,
            int s, int dest, int v) {
        int pred[] = new int[v];
        int dist[] = new int[v];
        if (BFS(adj, s, dest, v, pred, dist) == false) {
            System.out.println("Given source and destination" +
                    "are not connected");
            return;
        }
        LinkedList<Integer> path = new LinkedList<Integer>();
        int crawl = dest;
        path.add(crawl);
        while (pred[crawl] != -1) {
            path.add(pred[crawl]);
            crawl = pred[crawl];
        }
        System.out.println("Shortest path length is: " + dist[dest]);
        System.out.print("Path is :: ");
        for (int i = path.size() - 1; i >= 0; i--) {
            System.out.print(path.get(i) + " ");
        }
    }
    //continue in the next page
private static boolean BFS(ArrayList<ArrayList<Integer>> adj, int src,
                               int dest, int v, int pred[], int dist[]) {
        LinkedList<Integer> queue = new LinkedList<Integer>();
        boolean visited[] = new boolean[v];
        for (int i = 0; i < v; i++) {
            visited[i] = false;
            dist[i] = Integer.MAX_VALUE;
            pred[i] = -1;
        }
        visited[src] = true;
        dist[src] = 0;
        queue.add(src);
        while (!queue.isEmpty()) {
            int u = queue.remove();
            for (int i = 0; i < adj.get(u).size(); i++) {
                if (visited[adj.get(u).get(i)] == false) {
                    visited[adj.get(u).get(i)] = true;
                    dist[adj.get(u).get(i)] = dist[u] + 1;
                    pred[adj.get(u).get(i)] = u;
                    queue.add(adj.get(u).get(i));

                    if (adj.get(u).get(i) == dest)
                        return true;
                }
            }
        }
        return false;
    }
    public static void main(String args[])
    {

        int v = 8;

        ArrayList<ArrayList<Integer>> adj =
                new ArrayList<ArrayList<Integer>>(v);
        for (int i = 0; i < v; i++) {
            adj.add(new ArrayList<Integer>());
        }


        addEdge(adj, 0, 1);
        addEdge(adj, 0, 3);
        addEdge(adj, 1, 2);
        addEdge(adj, 3, 4);
        addEdge(adj, 3, 7);
        addEdge(adj, 4, 5);
        addEdge(adj, 4, 6);
        addEdge(adj, 4, 7);
        addEdge(adj, 5, 6);
        addEdge(adj, 6, 7);
        int source = 0, dest = 7;
        printShortestDistance(adj, source, dest, v);
    }
}
```