Q1: What do you mean by a Data Structure?
Ans: A Data structure is a programming method used to stores the data efficiently in the system's memory so that we can execute instructions on it and get meaningful results.
Examples of data structures are arrays, lists, maps,etc.

Q2: What are some applications of DS?
Ans: For example. Lists are used to store data and objects of same type . Maps are used to store key value pairs and can be used for dictionary purposes.
Trees are used to represent hierarchical structures such as file trees and stack is used to manage function calls. Hash Tables are store data in key-value pairs. It only stores data which has a key associated with it. Inserting and Searching operations are easily manageable while using Hash Tables.

And problems that can be solved using DS:-
Fibonacci number series
Knapsack problem
Shortest path by Dijkstra
Project scheduling

Q3. What are the advantages of a Linked List over an array?
Ans:
The size of a linked list is dynamic whereas for an array it is static.
Linked Lists can also store generic objects where as for an array the data types of its constituents should be the same.
Linked Lists are also flexible and we can create other data structures such as circular linked lists, doubly linked lists,etc.

Q4: Syntax to create a linked list node in C?
Ans:

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

Q5: What is the use of a doubly-linked list when compared to that of a single linked list?
Ans: While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

Q6: What is the difference between an array and stack?
Ans: In an array, we have a list of elements and we can access any of them at any time. But in a stack, there's no random-access operation; there are only Push, Peek and Pop, all of which deal exclusively with the element on the top of the stack. A stack is data-structure which has a last in first out policy.

Q7: Minimum number of queues required to implement a priority queue?
Ans: 2 Queues are required , one for storing data and one for the priorities.

Q8: What are different types of tree traversal techniques in a tree?

Ans: Tree traversal techniques are:

Pre order traversal (Root Left Right)

Inorder traversal (Left Root Right)

Post order traversal (Left Right Root)

Q9: Why is it said that searching a node in a binary search tree is efficient than that of simple binary tree.
Ans: Because binary Search tree keeps the keys in a sorted order for fast lookup. Using a binary search tree, length of tree traversal is reduced by comparing the search value with the root. The traversal path is reduced using this comparison , if the search value is greater than the root then the right side is traversed and left side is traversed in case of lower value.

Q10: What are the applications of Graph DS?
Ans: Graph DS is used to model scenarios where a component has relations with multiple components. Example of applications of graph:
In social media, users are considered to be the vertices and if they are friends then there is an edge running between them. Friend suggestion algorithm uses graph theory. Social media is an example of undirected graph.

Q11. Can we apply Binary search algorithm to a sorted Linked list?
Ans: Yes, Binary search is possible on the linked list if the list is ordered and you know the count of elements in list. But While sorting the list, you can access a single element at a time through a pointer to that node i.e. either a previous node or next node.

Q12: When can you tell that a Memory Leak will occur?
Ans: We can tell that a memory leak has occured, when a piece of memory which was previously allocated by the programmer. Then it is not deallocated properly by programmer. That memory is no longer in use by the program. So that place is reserved for no reason.

Q13: How will you check if a given Binary Tree is a binary search tree or not?
Ans: If the max of the left subtree in the binary tree is less than the root and minimum of right subtree is greater than the root node then this binary tree is a binary search tree.

Q14: Which data structure is ideal to perform recursion operation and why?
Ans: Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

Q15: Most important applications of Stack?
Ans :

Stacks can be used for expression evaluation.
Stacks can be used to check parenthesis matching in an expression.
Stack data structures are used in backtracking problems.
Stacks can be used for Conversion from one form of expression to another.
Stacks can be used for Memory Management.

Q16: Convert the below given expression to its equivalent prefix and post notations.
Ans: No expression is provided.
Sample example:

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

==========

Q17 : Sorting a stack using a temporary stack
Ans:

```
package com.rkm;

import java.util.Stack;

public class Test {

   public Stack<Integer> sorting(Stack<Integer> s) {
      Stack<Integer> tempStack = new Stack<>();

      while(!s.isEmpty()) {
         int x = s.pop();

         while(!tempStack.isEmpty() && tempStack.peek() > x) {
         s.push(tempStack.pop());
         }

         tempStack.push(x);
      }
      return tempStack;
   }

   public static void main(String args[]) {

      Stack<Integer> s = new Stack<>();
      s.add(15);
      s.add(7);
      s.add(13);
      s.add(98);
      s.add(21);
      System.out.println("Original Stack: " + s);
      Stack<Integer> sortedStack= s.sorting(s);
      System.out.println("Sorted Stack is:" + sortedStack);
   }
}
```

Q18: Program to reverse a queue.
Ans:

```
package com.rkm;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;


public class Test {

   static Queue<Integer> queue;

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

   static void revQ()
   {
      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[])
   {
      LinkedList<Integer> queue = new LinkedList<>();
      queue.add(90);
      queue.add(81);
      queue.add(72);
      queue.add(63);
      queue.add(54);
      queue.add(45);
      queue.add(36);
      queue.add(27);
      queue.add(18);
      queue.add(9);

      revQ();
      dispQ();
   }
}
```

Q19: Program to reverse first k elements of a queue.
Ans :

```
package com.rkm;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class Test {

   static Queue<Integer> queue;

   static void reverseQueueFirstKElements(int k)
   {
      if (queue.isEmpty() == true|| k > queue.size()) return;
      if (k <= 0) return;

      Stack<Integer> stack = new Stack<Integer>();

      for (int i = 0; i < k; i++) {
         stack.push(queue.peek());
         queue.remove();
      }

      while (!stack.empty()) {
         queue.add(stack.peek());
         stack.pop();
      }

      for (int i = 0; i < queue.size() - k; i++) {
         queue.add(queue.peek());
         queue.remove();
      }
   }
   static void display()
   {
      while (!queue.isEmpty()) {
         System.out.print(queue.peek() + " ");
         queue.remove();
      }
   }
   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      LinkedList<Integer> queue = new LinkedList<>();
      queue.add(45);    queue.add(36);
      queue.add(27);     queue.add(18);
      queue.add(9);      queue.add(90);
      queue.add(81);     queue.add(72);
      queue.add(63);     queue.add(54);

      System.out.println("Enter the Kth value Upto which you want to reverse: ")
      reverseQueueFirstKElements(sc.nextInt());
      display();
      sc.close();
   }
```

}
Q20: Program to return the nth node from the end in a linked list.
Ans :

```
package com.rkm;

class L{
   Node head;
   class Node {
      int data;
      Node next;
      Node(int d)
      {
         data = d;
         next = null;
      }
   }

   void printNthFromLast(int n)
   {
      int len = 0;
      Node temp = head;

      while (temp != null) {
         temp = temp.next;
         len++;
      }

      if (len < n) return;
      temp = head;

      for (int i = 1; i < len - n + 1; i++) temp = temp.next;

      System.out.println(temp.data);
   }

   public void push(int new_data)
   {

      Node new_node = new Node(new_data);
      new_node.next = head;
      head = new_node;
   }

}
public class Main{

public static void main(String[] args)
   {
      L list = new L()
      list.push(50);       list.push(9);
      list.push(12);       list.push(6);
      list.push(17);       list.push(39);
      list.push(45);       list.push(2);

      list.printNthFromLast(2);
   }
```

```
}
```

Q21 : Reverse a linked list.
Ans:

```
package com.rkm;

class L {

    static Node head;
    static class Node {

        int data;
        Node next;

        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    Node reverse(Node node)
    {
        Node prev = null;
        Node current = node;
        Node next = null;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        node = prev;
        return node;
    }

    void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    public static void main(String[] args)
    {
        L list = new L();
        list.head = new Node(85);
        list.head.next = new Node(15);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(20);

        list.printList(head);
        head = list.reverse(head);
        list.printList(head);
    }
}
```

Q22: Replace each element of the array by its rank in the array.
Ans:

```
package com.rkm;

import java.util.*;

public class Test {

   static void changeA(int[] arr)
   {
      int newArray[] = new int[arr.length];
      for(int i = 0 ; i < arr.length ; i++) newArray[i] = arr[i];
      Arrays.sort(newArray);
      Map<Integer, Integer> rankArr = new HashMap<>();
      int rank = 1;
      for (int i = 0; i < newArray.length; i++) {
            int element = newArray[i];
      if (ranks.get(element) == null) {
            rankArr.put(element, rank);
            rank++;
      }
      for (int i = 0; i < arr.length; i++) {
            int element = input[i];
            arr[i] = rankArr.get(input[i]);
      }

}
   public static void main(String[] args)
   {
      int[] a = { 15,55,26,48,25,65,87,26,32 };
      System.out.println(a);
      changeA(a);
      System.out.println(a);
   }
}
```

==========

Q23: Check if a given graph is a tree or not?
Ans : To check if a given graph is a tree or not, we have to check if the graph contains a cycle. A a graph is a tree if it doesn't contain a cycle. Since it is not mentioned if it is a directed or un-directed graph, a particular cycle detection algorithm can't be prescribed but techniques used to detect cycle are BFS or DFS cycle detection algorithm and Topological sorting.

Q24: Find out the Kth smallest element in an unsorted array?
Ans:

```
package com.rkm;

import java.util.Arrays;
import java.util.Collections;

public class Test {
	public static int kSmall(int[] a,int k,){
		Arrays.sort(a);
		return a[k - 1];
	}

	public static void main(String[] args)
	{
		int a[] = new int[] { 13,45,7,18,5,4,19 };
		int k = 4;
		System.out.print("K'th smallest element is " + kSmall(a, k));
	}
}
```

Q25: How to find the shortest path between two vertices?
Ans: Dijkstra's algorithm can be used to determine the shortest path from one node in a graph to every other node within the same graph data structure, provided that the nodes are reachable from the starting node.