Q1. What do you mean by a Data structure?

Ans. Data Structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. Some data Structure are Array, Linked List, Graph, Stack, Queue and Tree.

Q2. What are some of the applications of DS?

Ans. Data structure are used to store data in different- different scenario we need different kind of data structure some of the application of data structure is:-

1. Arrays: Implementation of other data structures, Execution of matrices and vectors, Dynamic memory allocation, Pointer container, Control tables.
2. Stack: Evaluation of expressions, Backtracking, Runtime memory management, Arrangement of books in a library.
3. Queue: Here, the data sent need not be received at the same rate at which it was sent. A certain system resource is to be shared between different processes.
4. Linked-List: Representation of sparse matrices, Non-contiguous data storage, Implementation of non-binary tree or other data structures, Dynamic memory management, Equalizing parenthesis, Symbol tables.
5. Set: Mapping of data, Common data storage.

Q3. What are the advantages of a Linked list over an array?

Ans. 1. It can grow, shrink it means it has variable size, while size of array is fixed.

2. Insertion/deletion of an element at beginning in a linked list is O(1) operation while in array it is O(n).

3. Insertion/ deletion at end can be O(1) if we use rear pointer (like in queue).

Q4. Write the syntax in C to create a node in the singly linked list.

Ans. type struct Node

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

Q5. What is the use of a doubly-linked list when compared to that of a singly

Linked list?

Ans. 1. Doubly linked list can be traversed in both forward and backward direction.
2 .The delete operation in doubly linked list is more efficient if pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node. In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In Doubly linked list, we can get the previous node using previous pointer.

Q6. What is the difference between an Array and Stack?

Ans. The main difference between array and stack is accessing of data and traversing of data in Array we can access data by index only or traverse the array from the index where we want but in case of stack we can only access the data in last in first out fashion and traversing is also done from last input to first input only.

Q7. What are the minimum number of Queues needed to implement the priority queue?

Ans. Two queues are required to implement the priority queue one for storing the data and other one for priority.

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

Ans. Three are three types of traversal technique in trees:-

1. Preorder Traversal (root -> left -> right)
2. Inorder Traversal (left -> root -> right)
3. Postorder Traversal (left ->right ->root)

Q9. 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 binary search tree is efficient because of property of binary search tree that each left side node is lesser than root node and each right side node is greater than root node so because of this searching will always follow a particular single path there is no need to traverse each node like binary tree.

Q10. What are the applications of Graph DS?

Ans. In Computer science graphs are used to represent the flow of computation.

- 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 World Wide Web, web pages are considered to be the vertices. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of Directed 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.

Q11. Can we apply Binary search algorithm to a sorted Linked list?
Ans. We cannot apply Binary search algorithm to a sorted linked list because binary search works on indices and linked list work on pointer.

Q12. When can you tell that a Memory Leak will occur?
Ans. A memory leak occurs when a program loses the ability to free a block of dynamically allocated memory.

Q13. 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.

Q14. Which data structure is ideal to perform recursion operation and why?

Ans. Stack is used to perform recursion operation 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. Every recursive function has its equivalent iterative (non-recursive) function.

Q15. What are some of the most important applications of a Stack?

Ans. 1. Infix to Postfix or Infix to Prefix Conversion.

2. Postfix or Prefix Evaluation.

3. Backtracking Procedure.

4. Recursion Procedure.

Q16. **Convert the below given expression to its equivalent Prefix and Postfix notations.**

Ans. No Expression is given.

Q17. **Sorting a stack using a temporary stack.**

Ans.

```
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;
  }
```

Q18. **Program to reverse a queue.**

Ans.

```
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();
    }
  }
```

Q19. **Program to reverse first k elements of a queue.**

Ans.

```
public class Reverse_k_element_queue {
  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 Print()
```

```
  {
    while (!queue.isEmpty()) {
      System.out.print(queue.peek() + " ");
      queue.remove();
    }
  }
```

Q20. 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;
    }
  }
  void printNthFromLast(int n)
  {
    Node main_ptr = head;
    Node ref_ptr = head;

    int count = 0;
    if (head != null) {
      while (count < n) {
        if (ref_ptr == null) {
          System.out.println(n + " is greater than the no "
                  + " of nodes in the list");
          return;
        }
        ref_ptr = ref_ptr.next;
        count++;
      }
      while (ref_ptr != null) {
        main_ptr = main_ptr.next;
        ref_ptr = ref_ptr.next;
      }
      System.out.println("Node no. " + n + " from last is " + main_ptr.data);
    }
  }
```

Q21. Reverse a linked list.

Ans. class LinkedList {

```
   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;
      }
   }
```

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

Ans. tatic void changeArr(int[] input)

```
{
    int newArray[]
        = Arrays
            .copyOfRange(input,0,input.length);
    Arrays.sort(newArray);
    int i;
    Map<Integer, Integer> ranks
        = new HashMap<>();

    int rank = 1;

    for (int index = 0;
         index < newArray.length;
         index++) {

        int element = newArray[index];
        if (ranks.get(element) == null) {

            ranks.put(element, rank);
            rank++;
        }
    }
    for (int index = 0;
         index < input.length;
         index++) {

        int element = input[index];
        input[index]
            = ranks.get(input[index]);

    }
}
```

Q23. Check if a given graph is a tree or not.

Ans. A tree will not contain a cycle, so if there is any cycle in the graph, it is not a tree. We can check it using another approach, if the graph is connected and it has V-1 edges, it could be a tree. Here V is the number of vertices in the graph.

```
class Graph
{
    private int V;   // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency List

    // Constructor
    Graph(int v)
```

```
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    void addEdge(int v,int w)
    {
        adj[v].add(w);
        adj[w].add(v);
    }
    Boolean isCyclicUtil(int v, Boolean visited[], int parent)
    {
        visited[v] = true;
        Integer i;
        Iterator<Integer> it = 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;
    }
    Boolean isTree()
    {
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;
    }
```

Q24.) Find out the Kth smallest element in an unsorted array.

Ans.

```
public static int kthSmallest(Integer[] arr, int k)
  {
     Arrays.sort(arr);
     return arr[k - 1];
  }
```

Q25. How to find the shortest path between two vertices.

Ans. 1. Find the paths between the source and the destination nodes.

2. Find the number of edges in all the paths and return the path having the minimum number of edges.