1. What do you mean by a Data structure?

A. Data Structure is a way of collecting and storing the data in such a way that we can perform operations on this data in an effective way. Data Structure is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have data which have a student's name "Sanchit" . Here, we can save "Sanchit" as string.

2. What are some of the applications of DS?

A. Data structures gives us a way to store and manage large amounts of data efficiently for uses such as large databases and cloud services. Because of this it is considered that, efficient data structures are key to designing efficient algorithms. We use different DS for different purpose. For example, arrangement of leader-board of a game can be done simply through arrays to store the score and arrange them in descending order to clearly make out the rank of each player in the game.

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

A. The main and major 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. We can also insert and remove any node at any point in linked list with just three steps, but in arrays we have reallocate all the elements when an element is inserted or deleted. Linked list is dynamic and it can store generic objects but arrays cannot. Write the syntax in C to create a node in the singly linked list.

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

A.
```
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?

A. Doubly linked list allows element two way traversal. It also 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?

A. The major difference is how we retrieve the data. In array, we can retrieve any element if we know the index of it. But in Stack, we can only retrieve the top element. In simple words, in stack there is no random access whereas in array, we can randomly access any element.

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

A. We need two queues to implement the priority queue. One is used for storing the data, another is used for priorities. Priority queues are implemented using 2-D array where it has two rows one for element and second for priority ,so minimum numbers of queues are needed to implement are two.

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

A. There are three types of traversal techniques in trees.

- Inorder (Left, Root, Right)
- Preorder (Root, Left, Right)
- Postorder (Left, Right, Root)

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

A. Searching in binary search tree (BST) is same as binary search in arrays. We compare with middle element in array but with root in BST and continue according to the result as same in binary search. But, in simple binary tree we have to check all the elements till we find the element. So, time complexity for search in simple binary tree is O(n) and it is O(h) in BST, where 'h' is the height of the tree. Hence, searching a node in a binary search tree is efficient than that of a simple binary tree.

10. What are the applications of Graph DS?

A. We basically use graphs to show the flow of computation. Some real life applications include,

- Google maps uses graphs for building transportation systems
- Facebook's friend suggestion algorithm uses graph theory. It is an example of undirected graph.
- In internet (WWW), 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.

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

A. There are two possibilities here, a Yes and a No. Yes, if the list is ordered and you know the count of elements in list. But it would be too complex to implement. No, because linked list doesn’t allow random access and binary search is a divide and conquer algorithm which strictly depends upon random access. And we basically encourage this approach because it would be too complex.

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

A. It may occur when the memory allocated to the objects which are not used or not of any use anymore is not released. It can also occur when an object which is already stored cannot be accessed by the code.

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

A. There are three conditions a Binary Search Tree should satisfy. They are:

- All nodes in the left sub-tree of a node (including parent) have values less than the node's value.
- All nodes in the right sub-tree of a node (including parent) have values greater than the node's value.
- Both left and right sub-trees are also binary search trees.

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

A. Stack data structure is the best to implement recursion. This is because of its Last In First Out property. Recursion uses stack to store the return addresses of the function call.

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

A. Some major applications are:

- Recursion
- Backtracking
- Parsing syntax
- Evaluation of notations(postfix, prefix, infix)

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

A. We need an expression for that. It can be done with the help of stack.

17. Sorting a stack using a temporary stack.

A.

```
public static Stack<Integer> sort(Stack<Integer> s)
{
      Stack<Integer> temps = new Stack<Integer>();
      while(!s.isEmpty())
      {
            int temp=s.pop();
            while(!temps.isEmpty() && temps.peek()>temp)
            {
                  s.push(temps.pop());
            }
            temps.push(temp);
      }
      return temps;
}
```

18. Program to reverse a queue.

A.

```
static Queue<Integer> reverse(Queue<Integer> q)
{
    int s=q.size();
    Queue<Integer> rev=new LinkedList<>();
    for (int i=0;i<s;i++)
    {
        for(int j=0; j<q.size()-1;j++)
        {
            int x=q.peek();
            q.remove();
            q.add(x);
        }
        rev.add(q.peek());
        q.remove();
    }
    return rev;
}
```

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

A.

```
public Queue<Integer> reversek(Queue<Integer> q, int k)
{
      Stack<Integer> s=new Stack<Integer>();
      int temp=k;
      int size=q.size();
      while(--temp>=0)
      {
            s.push(q.remove());
      }
      while(!s.isEmpty())
      {
            q.add(s.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.

A.

```
void nth_from_last(int n)
{
      Node p1 = head;
    Node p2 = head;
    int count = 0;
    if(head!=null) {
     while (count < n) {
           if(p2==null) {
                 System.out.println(n + " is greater than the
no "+ " of nodes in the list");
                return;
            }
            p2=p2.next;
            count++;
       }
       while (p2!=null) {
          p1=p1.next;
           p2=p2.next;
       }
       System.out.println("Node no. " + n + " from last is "
+ main_ptr.data);
    }
}
```

21. Reverse a linked list.

A.

```
public ListNode reverse(ListNode head)
{
    if (head==null || head.next==null)
     return head;
    ListNode prev=null;
    ListNode c=head
    ListNode next=head.next;
    while (next!=null)
    {
        c.next=prev;
        prev=c;
        c=next;
        next=next.next;
    }
    return c;
}
```

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

A.

```
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++;
      }
}
```

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

A.

```
Boolean isTree()
{
      Boolean checked[] = new Boolean[V];
      for (int i = 0; i < V; i++)
            checked[i] = false;
      if (isCyclicUtil(0, checked, -1))
            return false;
      for (int u = 0; u < V; u++)
            if (!checked[u])
                  return false;
      return true;
}
```

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

A.

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

Note: For this code to work, we must import java.util.Arrays library.

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

A. This can be by implementing Dijkstra's algorithm. The steps we should take are:

1. Every time that we set out to visit a new node, we will choose the node with the smallest known distance/cost to visit first.
2. Once we've moved to the node we're going to visit, we will check each of its neighbouring nodes.
3. For each neighbouring node, we'll calculate the distance/cost for the neighbouring nodes by summing the cost of the edges that lead to the node we're checking from the starting vertex.
4. Finally, if the distance/cost to a node is less than a known distance, we'll update the shortest distance that we have on file for that vertex.