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

Q2: What are some applications of DS?
Ans: Data structure are used to store and manage data. 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.

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

Q5: What is the use of a doubly-linked list when compared to that of a single linked list?
Ans: A doubly linked list can be traversed bidirectionally therefore it saves the overhead of traversal in cases when we roughly know the place of an element.

Q6: What is the difference between an array and stack?
Ans: The main difference between an array and stack is how they return the values, in an array we can access an index of an element and get the data whereas in a stack, the data is returned in a first in last out fashion, that is that the data that went in first is the last in order when we get data from the stack.

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:

- Inorder traversal (Left → Root → Right)
- Pre order traversal (Root → Left → 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: 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.
Hence the searching complexity of binary tree O(n) is reduced to O(h) where h is the height of the tree.

Q10: What are the applications of Graph DS?
Ans: Graph DS is used to model scenarios where a component has relations with multiple components. Graph DS is used for modelling networks such as social media networks etc.

Q11: Can we use binary search algorithm to a sorted linked list?
Ans : We cannot apply binary search algorithm to a sorted linked list because the binary search heavily relies on indexes , and the overhead to bootstrap an index system defeats the purpose.

Q12: When can you tell that a Memory Leak will occur?
Ans: A memory leak will occur when the unused objects aren't destroyed to free up the memory held by them.

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 is less than the root and minimum of right subtree is greater than the node it is a binary search tree.

Q14: Which data structure is ideal to perform recursion operation and why?
Ans: Stacks are used to perform recursion as function on the top of the stack can executed , the function can be popped and the data can be passed to the next function, this cycle continues until the required data is passed to the original function.

Q15: Most important applications of Stack?
Ans : Stacks are used to implement recursion, can be used to balance parenthesis and is used to implement calculations following BODMAS rule .

Q16: Convert the below given expression to its equivalent prefix and post notations.
Ans: –

Q17 : Sorting a stack using a temporary stack
Ans:
```
public Stack<Integer> sortStack(Stack<Integer> stack){
    Stack<Integer> temp = new Stack<>();
    while(!stack.isEmpty()){
        int currentVal = stack.pop();
        if(temp.isEmpty() || temp.peek() < currentVal){
            temp.push(currentVal);
        }else{
            while(temp.peek()>currentVal){
                stack.push(temp.pop());
            }
            temp.push(currentVal);
        }
    }
    while(!temp.isEmpty()){
        stack.push(temp.pop());
    }
    return stack;
}
```

Q18: Program to reverse a queue

```
public Queue<Integer> reverseQueue(Queue<Integer> s){

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

     while(!s.isEmpty()){
     stack.push(s.poll());
     }
     while(!stack.isEmpty()){

     s.add(stack.pop());

     }
     return s;

}
```

Q19: Program to reverse first k elements of a queue

Ans :
```
public Queue<Integer> reverseK(Queue<Integer> queue, int k){

     Queue<Integer> newQueue = new LinkedList<>();
     Stack<Integer> stack = new Stack<>();

     while(k-- >0 ){
     stack.push(queue.poll());
     }

     while(!stack.isEmpty()){
     newQueue.add(stack.pop());
     }
     while(!queue.isEmpty()){
     newQueue.add(queue.poll());
     }
     return newQueue;

}
```

Q20: Program to return the nth node from the end in a linked lis

Ans :
```
public int getLength(Node n,int k){
     if(n == null){
          return k;
     }
     return getLength(n.next, k+1);

     }
     public int nthNode(Node n, int nth){
          int getLengthFromStart = getLength(n,0) - nth;

          int ret = 0;
          Node current = n;
          while(getLengthFromStart-- > 0){
          ret = Node.data;
          current = current.next;
          }
          return ret;

     }
```

Q21 : Reverse a linked list

```
Ans: class Node{
    Node next;
    int i;
    public Node(){
        next = null;
    }
    public Node(int i){
        this.i = i;
        this.next = null;
    }

}

public Node reverseNode(Node n){
    Node prev = null;
    Node current = n;
    while(n != null){
        Node revNode = new Node();
        revNode.i = n.i;
        revNode.next = prev;

        prev = revNode;

        n = n.next;

    }
    return prev;

}
```

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

Ans:
```
static void changeArr(int[] input)
    {

        int newArray[]  = new int[input.length];
           for(int I = 0 ; I < input.length ; i++){
                newArray[i] = input[i];
           }


        Arrays.sort(newArray);
        int i = 0;

        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 : 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:
```
public int kthElement(int[] array , int k){
      Arrays.sort(array);
      return array[k-1];
}
```

Q25: How to find the shortest path between two vertices?
Ans: Shortest path between two vertices is determined by using a shortest path algorithm called Dijkstra’s Algorithm.
Dijkstra’s Algorithm is also called single source shortest path algorithm.