1) What do you mean by a Data structure?

Ans. DS is a way of organizing data in a computer so that it can be used effectively. Each data structure contains information about the data values, relationships between the data and functions that can be applied to the data.

2) What are some of the applications of DS?

Ans. Array, Linked List, Stack, Queue, Tree, Graph etc.

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

Advantages of linked list over array:-

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)

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

```
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 DLL can be traversed in both forward and backward direction.
The delete operation in DLL is more efficient if pointer to the node to be deleted is given. It can also be used to implement stacks as well as heaps and binary trees.

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

Array	Stack
Array has a fixed size.	Stack has a dynamic size
Stack can contain elements of different data type	Array contains elements of same data type
Array contains elements of same data type	Stacks are based on the LIFO
Insertion and deletion in array can be done at any index in the array.	Insertion and deletion in stacks take place only from one end of the list called the top.

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

Priority queues r applied 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?

- In-order
- Pre-order
- Post-order

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

BINARY TREE is unordered hence slower in process of insertion, deletion and searching. IN BINARY SEARCH TREE the left subtree has elements less than the nodes element and the right subtree has elements greater than the nodes element.

10) What are the applications of Graph DS?

- 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 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 the vertices. There is an edge from a page u to another 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?

Yes.

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

Memory leak occurs when programmers create a memory in heap and forget to delete it.

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

A Binary Search Tree (BST) is a binary tree with the following properties:

- The left subtree of a node will always contain nodes whose keys are less than that node's key.
- The right subtree of a node will always contain nodes with keys greater than that node's key.
- The left and right subtree of a node will also, in turn, be binary search trees.

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

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. Every recursive function has its equivalent iterative (non-recursive) function.

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

- Expression Evaluation
- Expression Conversion
    - i. Infix to Postfix
    - ii. Infix to Prefix
    - iii. Postfix to Infix
    - iv. Prefix to Infix
- Backtracking
- Memory Management

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

17) Sorting a stack using a temporary stack

Algorithm to sort stack:-

- Create a temporary stack say temp_stack.
- Repeat until the input_stack is not empty
- Pop an element from input_stack call it temp.
- While temp_stack is not empty and top of the temp_stack > temp, pop data from temp_stack and push it to the input_stack
- Push temp to the temp_stack
- Print the temp_stack.

18) Program to reverse a queue

```
#include <bits/stdc++.h>
using namespace std;
void Print(queue<int>& Queue)
{
   while (!Queue.empty()) {
      cout << Queue.front() << " ";
      Queue.pop();
   }
}
 void reverseQueue(queue<int>& Queue)
{
   stack<int> Stack;
   while (!Queue.empty()) {
      Stack.push(Queue.front());
      Queue.pop();
   }
   while (!Stack.empty())
{
      Queue.push(Stack.top());
      Stack.pop();
   }
}
int main()
{
   queue<int> Queue;
   Queue.push(10);
   Queue.push(20);
   Queue.push(30);
   Queue.push(40);
   Queue.push(50);
   Queue.push(60);
   Queue.push(70);
   Queue.push(80);
   Queue.push(90);
   Queue.push(100);

   reverseQueue(Queue);
   Print(Queue);
}
```

19) Program to reverse first k elements of a queue

```
#include <bits/stdc++.h>
using namespace std;
void reverseQueueFirstKElements (int k, queue<int>& Queue)
{
   if (Queue.empty() == true || k > Queue.size())
      return;
   if (k <= 0)
      return;
```

```
    stack<int> Stack;

    for (int i = 0; i < k; i++)
   {
        Stack.push(Queue.front());
        Queue.pop();
    }

    while (!Stack.empty())
   {
        Queue.push(Stack.top());
        Stack.pop();
    }

    for (int i = 0; i < Queue.size() - k; i++)
  {
        Queue.push(Queue.front());
        Queue.pop();
    }
}

void Print(queue<int>& Queue)
{
    while (!Queue.empty()) {
        cout << Queue.front() << " ";
        Queue.pop();
    }
}

int main()
{
    queue<int> Queue;
    Queue.push(10);
    Queue.push(20);
    Queue.push(30);
    Queue.push(40);
    Queue.push(50);
    Queue.push(60);
    Queue.push(70);
    Queue.push(80);
    Queue.push(90);
    Queue.push(100);

    int k = 5;
    reverseQueueFirstKElements(k, Queue);
    Print(Queue);
}
```

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

```
#include <bits/stdc++.h>
using namespace std;
  struct Node {
    int data;
    struct Node* next;
};
void printNthFromLast(struct Node* head, int n)
{
    int len = 0, i;
    struct Node* temp = head;
    while (temp != NULL)
   {
        temp = temp->next;
        len++;
    }
    if (len < n)
        return;

    temp = head;
    for (i = 1; i < len - n + 1; i++)
        temp = temp->next;

    cout << temp->data;

    return;
}

void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
int main()
{
    struct Node* head = NULL;
    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 35);

    printNthFromLast(head, 4);
    return 0;
}
```

21) Reverse a linked list

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    struct Node* next;
    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
};

struct LinkedList
  {
    Node* head;
    LinkedList()
    {
        head = NULL;
    }
    void reverse()
    {
        Node* current = head;
        Node *prev = NULL, *next = NULL;

        while (current != NULL) {
            next = current->next;
           current->next = prev;
            prev = current;
            current = next;
        }
        head = prev;
    }
    void print()
    {
        struct Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
    }

    void push(int data)
    {
        Node* temp = new Node(data);
        temp->next = head;
        head = temp;
    }
};
int main()
{
    LinkedList ll;
    ll.push(20);
    ll.push(4);
    ll.push(15);
    ll.push(85);
```

```
   cout << "Given linked list\n";
   ll.print();

   ll.reverse();

   cout << "\nReversed Linked list \n";
   ll.print();
   return 0;
}
```

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

```
#include <bits/stdc++.h>
using namespace std;
void transform(int arr[], int n)
{
map<int, int> map;
for (int i = 0; i < n; i++)
{
map[arr[i]] = i;
}
int rank = 1;

for (auto i: map)
arr[i.second] = rank++;

}

int main()
{
int n;
cout <<"\nEnter the number of elements : ";
cin >> n;
int arr[n];
cout << "\nInput the array elements : ";
for (int i = 0; i < n; i++)
{
cin >> arr[i];
}
transform(arr, n);
cout <<"\nAfter ranking : ";
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
return 0;
}
```

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

We can simply find it by checking the criteria of a tree. 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.

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

```
#include <algorithm>
#include <iostream>
using namespace std;
int kthSmallest(int arr[], int n, int k)
{
   sort(arr, arr + n);
   return arr[k - 1];
}
int main()
{
   int arr[] = { 12, 3, 5, 7, 19 };
   int n = sizeof(arr) / sizeof(arr[0]), k = 2;
   cout << "K'th smallest element is " << kthSmallest(arr, n, k);
   return 0;
}
```

25) How to find the shortest path between two vertices

- Depth-First Search (DFS)
- Breadth-First Search (BFS)
- Bidirectional Search
- Dijkstra’s Algorithm
- Bellman-Ford Algorithm