1. What do you mean by a Data structure?

Answer:

Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

2. What are some of the applications of DS?

Answer:

Arrays, Queue, Stack, Graphs, Linked List.

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

Answer:

Both Arrays and Linked List can be used to store linear data of similar types, but they both have some advantages and disadvantages over each other.

Following are the points in favors of Linked Lists.

(1) The size of the arrays is fixed: So, we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.

(2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

For example, suppose we maintain a sorted list of IDs in an array Id [].

Id [] = [1000, 1010, 1050, 2000, 2040, …..].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in Id [], everything after 1010 has to be moved.

So Linked list provides following two advantages over arrays
1) Dynamic size
2) Ease of insertion/deletion

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

Answer:

```
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?

Answer:

- SLL has nodes with only a data field and next link field. While in DLL has nodes with a data field, a previous link field and a next link field.
- In SLL, the traversal can be done using the next node link only. While in DLL, the traversal can be done using the previous node link or the next node link.
- The SLL occupies less memory than DLL as it has only 2 fields. While int DLL occupies more memory than SLL as it has 3 fields.
- The SLL has less efficient access to elements. While in DLL has more efficient access to elements.

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

Answer:

Stack	**Array**
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket e.g. arr[10].
Insertion and deletion in stacks take place only from one end of the list called the top.	Insertion and deletion in array can be done at any index in the array.
Stack has a dynamic size.	Array has a fixed size.
Stack can contain elements of different data type.	Array contains elements of same data type.
We can do only linear search.	We can do both linear and Binary search.

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

Answer:

The minimum number of queue required is 2.

One for storing the actual data and another one for storing the priorities.

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

Answer:

There are three types of traversal techniques in a tree:

1) **Inorder Traversal:**

   Algorithm Inorder(tree)

   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)

2) **Preorder Traversal:**

   Algorithm Preorder(tree)

   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)

3) **Postorder Traversal:**

   Algorithm Postorder(tree)

   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root

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

Answer:

While Binary search tree is data structure which uses the concept of binary search. Binary search tree has a special quality that every node has at most 2 nodes and left child of every node has less value than node and right child has more value than node. By this method it become easy to search a node in a tree.

10. What are the applications of Graph DS?

Answer:

Application of Graphs:

- **Computer Science:** In computer science, graph is used to represent networks of communication, data organization, computational devices etc.
- **Physics and Chemistry:** Graph theory is also used to study molecules in chemistry and physics.
- **Social Science:** Graph theory is also widely used in sociology.
- **Mathematics:** In this, graphs are useful in geometry and certain parts of topology such as knot theory.
- **Biology:** Graph theory is useful in biology and conservation efforts.

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

Answer:

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.

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

Answer:

A memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code.

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

Answer:

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

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

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

Answer:

Recursion makes use of the system stack for storing the return addresses of the function calls.

Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written explicit, stack is to be used.

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

Answer:

The Stack is Last In First Out (LIFO) data structure. This data structure has some important applications in different aspect. These are like below –

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

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

Expression not given

The steps for conversation from Prefix to Postfix are:

1. Read the Prefix expression in reverse order
2. If the symbol is an operand, then push it onto the Stack
3. If the symbol is an operator, then pop two operands from the Stack
   string = operand1 + operand2 + operator
4. Push the resultant string back to Stack

5. Repeat the above steps until end of Prefix expression.

17. Sorting a stack using a temporary stack

Answer:

```
import java.util.*;

class SortStack
{
    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;
    }

    public static void main(String args[])
    {
            Stack<Integer> input = new Stack<Integer>();
            input.add(50);
            input.add(25);
            input.add(45);
            input.add(90);
            input.add(85);
            input.add(15);
            Stack<Integer> tmpStack=sortstack(input);
            System.out.println("Sorted numbers are:");

            while (!tmpStack.empty())
            {
                    System.out.print(tmpStack.pop()+" ");
            }
    }
}
```

18. Program to reverse a queue

Answer:

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
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();
        }
    }
    public static void main(String args[])
    {
        queue = new LinkedList<Integer>();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        queue.add(40);
        queue.add(50);
        queue.add(60);
        queue.add(70);
        queue.add(80);
        queue.add(90);
        queue.add(100);

        reversequeue();
        Print();
```

```
    }
}
```

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

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

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();
        }
    }
```

```
    public static void main(String args[])
    {
        queue = new LinkedList<Integer>();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        queue.add(40);
        queue.add(50);
        queue.add(60);
        queue.add(70);
        queue.add(80);
        queue.add(90);
        queue.add(100);

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

20. Program to return the nth node from the end in a linked list.
Answer:

```
class LinkedList {
    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 static void main(String[] args)
    {
            LinkedList llist = new LinkedList();
            llist.push(20);
            llist.push(4);
            llist.push(15);
            llist.push(35);

            llist.printNthFromLast(4);
    }
}
```

21. Reverse a linked list

Answer:

```
    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;
```

```
        }
    }
    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();
        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);

        System.out.println("Given Linked list");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("");
        System.out.println("Reversed linked list ");
        list.printList(head);
    }
}
```

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

```
import java.util.*;

class GFG {
    static 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]);

        }
    }
    public static void main(String[] args)
    {
        int[] arr = { 100, 2, 70, 2 };
        changeArr(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

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

```
import java.io.*;
import java.util.*;
```

```
class Graph
{
   private int V;
   private LinkedList<Integer> adj[];
   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 visited[] = new 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;
```

```
    }

    public static void main(String args[])
    {
        // Create a graph given in the above diagram
        Graph g1 = new Graph(4);
        g1.addEdge(0,1);
        g1.addEdge(1, 2);
        g1.addEdge(2, 3);
        g1.addEdge(3, 4);
        if (g1.isTree())
            System.out.println("Graph is Tree");
        else
            System.out.println("Graph is not Tree");

        Graph g2 = new Graph(5);
        g2.addEdge(0,1);
        g2.addEdge(1,2);
        g2.addEdge(2,3);
        g2.addEdge(3,4);
        g2.addEdge(4,5 );
        if (g2.isTree())
            System.out.println("Graph is Tree");
        else
            System.out.println("Graph is not Tree");
    }
}
```

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

Answer:

```
import java.util.Arrays;
import java.util.Collections;
class GFG {
    //returns kth smallest element in a given array
    public static int kthSmallest(Integer[] arr, int k)
    {
        // Sorting
        Arrays.sort(arr);
        // Return kth element in the sorted array
        return arr[k - 1];
    }

    // driver program
    public static void main(String[] args)
```

```
    {
        Integer arr[] = new Integer[] {1,2,3,4,5,6,7,8,9,10};
        int k = 2;
        System.out.print("K'th smallest element is " + kthSmallest(arr,k));
    }
}
```

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

```
import java.io.*;
import java.util.*;
import java.util.LinkedList;

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); }
    Boolean isReachable(int s, int d)
    {
            LinkedList<Integer>temp;

            // Mark all the vertices as not visited(By default set
            // as false)
            boolean visited[] = new boolean[V];

            // Create a queue for BFS
            LinkedList<Integer> queue = new LinkedList<Integer>();

            // Mark the current node as visited and enqueue it
            visited[s]=true;
            queue.add(s);
```

```
        Iterator<Integer> i;
        while (queue.size()!=0)
        {
                // Dequeue a vertex from queue and print it
                s = queue.poll();

                int n;
                i = adj[s].listIterator();

                while (i.hasNext())
                {
                        n = i.next();
                        if (n==d)
                                return true;

                        // Else, continue to do BFS
                        if (!visited[n])
                        {
                                visited[n] = true;
                                queue.add(n);
                        }
                }
        }
        return false;
}
public static void main(String args[])
{
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        int u = 1;
        int v = 3;
        if (g.isReachable(u, v))
                System.out.println("There is a path from " + u +" to " + v);
        else
                System.out.println("There is no path from " + u +" to " + v);;

        u = 3;
```

```
            v = 1;
            if (g.isReachable(u, v))
                  System.out.println("There is a path from " + u +" to " + v);
            else
                  System.out.println("There is no path from " + u +" to " + v);;
      }
}
```