ASSIGNMENT-2

1) What do you mean by a Data structure?

Logical or mathematical arrangement of a data to make the processing of data simpler.

2) What are some of the applications of DS?

- Array-Storing contacts in a phone
- Stack-Undo operation
- Queue-In Operating System - in job scheduling
- Linked List-Web pages

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

Both Arrays and Linked list can be used to store linear data of similar types, but

- Linked list has dynamic size whereas for array it is fixed size
- And insertion and deletion is easy in linked list.

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

```
struct node {
   int data;
   struct node *next;
};
struct node *head, *tail = NULL;
void addNode (int data) {
struct node *newNode = (struct node*) malloc (sizeof(struct node));
newNode->data = data;
newNode->next = NULL;
if (head == NULL) {
head = newNode;
tail = newNode;}
else {
tail->next = newNode;
tail = newNode;}}
void display () {
   struct node *current = head;
```

```
    if (head == NULL) {
        printf ("list is empty\n");
        return;
    }
    printf ("nodes of single linked list \n");
    while (current! = NULL) {
        printf ("%d ", current->data);
        current = current->next;
}   }
```

5) What is the use of a doubly-linked list when compared to that of a singly linked list?

- Singly linked list allows traversal elements only in one way whereas in doubly linked list allows element two-way traversal.
- Singly linked list is generally used for implementation of stacks on other hand doubly linked list can be used to implement stacks as well as heaps and binary trees.

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

In an array, you have a list of elements and you can access any of them at any time. But in a stack, there's no random-access operation; there are only Push, Peek and Pop, all of which deal exclusively with the element on the top of the stack. A stack follows a last in first out policy.

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

The minimum number of Queues needed to implement the priority queue are two, first for storing data and other for priorities.

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

There are three types of traversal techniques i.e:

1. PRE-order
2. In-order
3. 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?

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?

- 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.

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

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?

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?

A **Binary Search Tree (BST)** 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?

Stack, is ideal data structure to perform recursion operation because of its LIFO property (Last In First Out) it remembers its 'caller' and returns the value to its caller. Recursion uses stack to return values

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

- To reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
- An "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.
  - Undo/Redo stacks in Excel or Word.
- Language processing:
  - space for parameters and local variables is created internally using a stack.
  - compiler's syntax check for matching braces is implemented by using stack.

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

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

```
import java.util.*;

class SortStack
{
   public static Stack<Integer> sortstack(Stack<Integer> input)
   {
      Stack<Integer> tmpStack = new Stack<>();
      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<>();
      input.add(20);
      input.add(35);
      input.add(25);
      input.add(55);
      input.add(65);
      input.add(70);
      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

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
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<>();
        queue.add(21);
        queue.add(22);
        queue.add(23);
        queue.add(24);
        queue.add(25);
        queue.add(26);
        queue.add(27);
        queue.add(28);
        queue.add(29);
        queue.add(30);
        reversequeue();
        print();
    }
}
```

19) Program to reverse first k elements of a queue

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

class Reverse_k_element_queue {
    static Queue<Integer> queue;
    static void reverseQueueFirstKElements(int k)
    {
        if (queue.isEmpty()
                || k > queue.size())
            return;
        if (k <= 0)
            return;

        Stack<Integer> stack = new Stack<>();
        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<>();
        queue.add(21);
```

```
        queue.add(22);
        queue.add(23);
        queue.add(24);
        queue.add(25);
        queue.add(26);
        queue.add(27);
        queue.add(28);
        queue.add(29);
        queue.add(30);
        int k =10;
        reverseQueueFirstKElements(k);
        print();
    }
}
```

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

```
class LinkedList {
        Node head;
    static class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }
    void printNthFromLast()
    {
        int len = 0;
        Node temp = head;
        while (temp != null) {
            temp = temp.next;
            len++;
        }
        if (len < 4)
            return;
        temp = head;
        for (int i = 1; i < len - 4 + 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(1);
        llist.push(2);
        llist.push(3);
        llist.push(4);
        llist.printNthFromLast();
    }
}
```

21)Reverse a linked list

```
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;
        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();
        head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = new Node(4);
        System.out.println("given linked list\n");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("reversed linked list\n");
        list.printList(head);
    }
}
```

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

```
class GFG {
    static void changeArr(int[] input)
    {
        int[] newArray = Arrays.copyOfRange(input,0,input.length);
        Arrays.sort(newArray);
        Map<Integer, Integer> ranks = new HashMap<>();
        int rank = 1;
        for (int element : newArray)
        {
            if (ranks.get(element) == null) {
                ranks.put(element, rank);
                rank++;
            }
        }
```

```
        for (int index = 0; index < input.length; index++)
        {
            input[index] = ranks.get(input[index]);
        }
    }
    public static void main(String[] args)
    {
        int[] arr = { 100,90,80,70};
        changeArr(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

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

```
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[])
{

    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("it is Tree");
    else
        System.out.println("it not Tree");
```

```
    }
}
```

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

```
import java.util.Arrays;
import java.util.Collections;
class GFG {

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

1. Input the nodes of the graph

2. Get the input of the source and destination nodes.

3. Find the paths between the source and destination nodes.

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

5. The shortest path between the two vertices is found.