1. **What do you mean by a Data structure?**

**Ans:** Data Structure is collection of data values the relationships among them, and the functions or operations that can be applied to the data.. It is a data organization, management, and storage format that enables efficient access and modification.

1. **What are some of the applications of DS?**

**Ans:** 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.

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

**Ans:** Operations like insertion and deletion in arrays consume a lot of time. The performance of these operations in Linked lists is fast. Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expand and contract its size. Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.

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

Ans: struct node

{

int data;

struct node \*next;

};

1. **What is the use of a doubly-linked list when compared to that of a singly**  
   **linked list?**

Ans: 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.

1. **What is the difference between an Array and Stack?**

Ans: A stack is a liner data structure represented by a sequential collection of elements in fixed order. An array is a collection of homogeneuos data values which is identified by index value. The element of adding and deleting an element is happend by following LIFO rules(Element is adding in last of stack and Element is deleting in first of stack). In case of array,it is a random access operation and everything goes down to the begining of the array.Insertion and Deletion takes place in any position.

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

Ans: 2 queues. one is used for storing data... another is used for priorities. 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.

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

Ans: There are three types of traversal techniques in trees.

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

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

Ans: It supports three main operations: searching of elements, insertion of elements, and deletion of elements. Binary Search Tree allows for fast retrieval of elements stored in the tree as each node key is thoroughly compared with the root node, which discards half of the tree.

1. **What are the applications of Graph DS?**

Ans: 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 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. In World Wide Web, 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. It was the basic idea behind Google Page Ranking Algorithm.

In Operating System, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.

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

Ans: Yes, Binary search is possible on the linked list if the list is ordered and you know the count of elements in list.

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

Ans: 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.

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

Ans:

A: If a node is a left child, then its key and the keys of the nodes in its right subtree are less than its parent's key.

B: If a node is a right child, then its key and the keys of the nodes in its left subtree are greater than its parent's key.

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

Ans: Stack is ideal to perform recursion operation . 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.

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

Ans: The stack can be used to convert some infix expression into its postfix equivalent, or prefix equivalent. These postfix or prefix notations are used in computers to express some expressions.

1. **Convert the below given expression to its equivalent Prefix And Postfix notations.**

Ans:

Expression : a+b\*(c^d-e)^(f+g\*h)-i

Postfix : abcd^e-fgh\*+^\*+i-

Prefix : +a-\*b^-^cde+f\*gh

1. **Sorting a stack using a temporary stack**

Ans:   
#include <bits/stdc++.h>

using namespace std;

stack<int> sort\_stack(stack<int> &stack1)

{

stack<int> stack2;

while(!stack1.empty())

{

int temp = stack1.top();

stack1.pop();

while(!stack2.empty() && stack2.top() > temp)

{

stack1.push(stack2.top());

stack2.pop();

}

stack2.push(temp);

}

return stack2;

}

int main()

{

stack<int> stack1;

int n;

cout << "\nEnter the number of elements to be pushed to the stack : ";

cin >> n;

cout << "\nEnter the stack elements : ";

for(int i = 0; i < n; i++)

{

int data;

cin >> data;

stack1.push(data);

}

stack<int> stack2 = sort\_stack(stack1);

cout <<"\nSorted Stack : ";

while(!stack2.empty())

{

cout << stack2.top()<< " ";

stack2.pop();

}

cout << endl;

return 0;

}

**18)Program to reverse a queue**

**Ans:**

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();

}

}

// Function to reverse the queue

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();

}

}

// Driver code

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(10);

queue.add(9);

queue.add(8);

queue.add(7);

queue.add(6);

queue.add(5);

queue.add(4);

queue.add(3);

queue.add(2);

queue.add(1);

reversequeue();

Print();

}

}

1. **Program to reverse first k elements of a queue**

Ans:

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();

}

}

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

**Ans:**

class Node

{

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

class LinkedList

{

Node head; //the head of list

/\* Takes index as argument and return data at index\*/

public int GetNth(int index)

{

Node current = head;

int count = 0; /\* index of Node we are

currently looking at \*/

while (current != null)

{

if (count == index)

return current.data;

count++;

current = current.next;

}

/\* if we get to this line, the caller was asking

for a non-existent element so we assert fail \*/

assert(false);

return 0;

}

/\* Given a reference to the head of a list and an int,

inserts a new Node on the front of the list. \*/

public void push(int new\_data)

{

/\* 1. alloc the Node and put data\*/

Node new\_Node = new Node(new\_data);

/\* 2. Make next of new Node as head \*/

new\_Node.next = head;

/\* 3. Move the head to point to new Node \*/

head = new\_Node;

}

/\* Driver program to test above functions\*/

public static void main(String[] args)

{

/\* Start with empty list \*/

LinkedList llist = new LinkedList();

/\* Use push() to construct below list

1->12->1->4->1 \*/

llist.push(1);

llist.push(4);

llist.push(1);

llist.push(12);

llist.push(1);

/\* Check the count function \*/

System.out.println("Element at index 3 is "+llist.GetNth(3));

}

}

**21)Reverse a linked list**

**Ans:**

**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**

**Ans:**

**import java.util.Arrays;**

**import java.util.Map;**

**import java.util.TreeMap;**

**class Main**

**{**

**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 (Map.Entry<Integer, Integer> entry : map.entrySet()) {**

**arr[entry.getValue()] = rank++;**

**}**

**}**

**public static void main(String[] args)**

**{**

**int[] A = { 10, 8, 15, 12, 6, 20, 1 };**

**transform(A);**

**System.out.println(Arrays.toString(A));**

**}**

**}**

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

**Ans:**

**import java.util.\*;**

**class Graph**

**{**

**private int V; // No. of vertices**

**private LinkedList<Integer> adj[]; //Adjacency List**

**Graph(int v)**

**{**

**V = v;**

**adj = new LinkedList[v];**

**for (int i=0; i<v; ++i)**

**adj[i] = new LinkedList();**

**}**

**// Function to add an edge into the graph**

**void addEdge(int v,int w)**

**{**

**adj[v].add(w);**

**adj[w].add(v);**

**}**

**Boolean isCyclicUtil(int v, Boolean visited[], int parent)**

**{**

**// Mark the current node as visited**

**visited[v] = true;**

**Integer i;**

**// Recur for all the vertices adjacent to this vertex**

**Iterator<Integer> it = adj[v].iterator();**

**while (it.hasNext())**

**{**

**i = it.next();**

**// If an adjacent is not visited, then recur for**

**// that adjacent**

**if (!visited[i])**

**{**

**if (isCyclicUtil(i, visited, v))**

**return true;**

**}**

**// If an adjacent is visited and not parent of**

**// current vertex, then there is a cycle.**

**else if (i != parent)**

**return true;**

**}**

**return false;**

**}**

**Boolean isTree()**

**{**

**// Mark all the vertices as not visited and not part**

**// of recursion stack**

**Boolean visited[] = new Boolean[V];**

**for (int i = 0; i < V; i++)**

**visited[i] = false;**

**// The call to isCyclicUtil serves multiple purposes**

**// It returns true if graph reachable from vertex 0**

**// is cyclcic. It also marks all vertices reachable**

**// from 0.**

**if (isCyclicUtil(0, visited, -1))**

**return false;**

**// If we find a vertex which is not reachable from 0**

**// (not marked by isCyclicUtil(), then we return false**

**for (int u = 0; u < V; u++)**

**if (!visited[u])**

**return false;**

**return true;**

**}**

**}**

**public class Answer23 {**

**public static void main(String[] args) {**

**// Create a graph given in the above diagram**

**Graph g1 = new Graph(5);**

**g1.addEdge(1, 0);**

**g1.addEdge(0, 2);**

**g1.addEdge(0, 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(1, 0);**

**g2.addEdge(0, 2);**

**g2.addEdge(2, 1);**

**g2.addEdge(0, 3);**

**g2.addEdge(3, 4);**

**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**

**Ans:**

**import java.util.Arrays;**

**import java.util.Collections;**

**class program11 {**

**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[] { 12, 3, 5, 7, 19 };**

**int k = 2;**

**System.out.print("K'th smallest element is " + kthSmallest(arr, k));**

**}**

**}**

**25) How to find the shortest path between two vertices**

**Ans:**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**import java.util.LinkedList;**

**public class pathUnweighted {**

**public static void main(String args[])**

**{**

**int v = 8;**

**ArrayList<ArrayList<Integer>> adj =**

**new ArrayList<ArrayList<Integer>>(v);**

**for (int i = 0; i < v; i++) {**

**adj.add(new ArrayList<Integer>());**

**}**

**addEdge(adj, 0, 1);**

**addEdge(adj, 0, 3);**

**addEdge(adj, 1, 2);**

**addEdge(adj, 3, 4);**

**addEdge(adj, 3, 7);**

**addEdge(adj, 4, 5);**

**addEdge(adj, 4, 6);**

**addEdge(adj, 4, 7);**

**addEdge(adj, 5, 6);**

**addEdge(adj, 6, 7);**

**int source = 0, dest = 7;**

**printShortestDistance(adj, source, dest, v);**

**}**

**private static void addEdge(ArrayList<ArrayList<Integer>> adj, int i, int j)**

**{**

**adj.get(i).add(j);**

**adj.get(j).add(i);**

**}**

**private static void printShortestDistance(**

**ArrayList<ArrayList<Integer>> adj,**

**int s, int dest, int v)**

**{**

**// predecessor[i] array stores predecessor of**

**// i and distance array stores distance of i**

**// from s**

**int pred[] = new int[v];**

**int dist[] = new int[v];**

**if (BFS(adj, s, dest, v, pred, dist) == false) {**

**System.out.println("Given source and destination" +**

**"are not connected");**

**return;**

**}**

**LinkedList<Integer> path = new LinkedList<Integer>();**

**int crawl = dest;**

**path.add(crawl);**

**while (pred[crawl] != -1) {**

**path.add(pred[crawl]);**

**crawl = pred[crawl];**

**}**

**System.out.println("Shortest path length is: " + dist[dest]);**

**System.out.println("Path is ::");**

**for (int i = path.size() - 1; i >= 0; i--) {**

**System.out.print(path.get(i) + " ");**

**}**

**}**

**private static boolean BFS(ArrayList<ArrayList<Integer>> adj, int src,**

**int dest, int v, int pred[], int dist[])**

**{**

**LinkedList<Integer> queue = new LinkedList<Integer>();**

**boolean visited[] = new boolean[v];**

**for (int i = 0; i < v; i++) {**

**visited[i] = false;**

**dist[i] = Integer.MAX\_VALUE;**

**pred[i] = -1;**

**}**

**visited[src] = true;**

**dist[src] = 0;**

**queue.add(src);**

**while (!queue.isEmpty()) {**

**int u = queue.remove();**

**for (int i = 0; i < adj.get(u).size(); i++) {**

**if (visited[adj.get(u).get(i)] == false) {**

**visited[adj.get(u).get(i)] = true;**

**dist[adj.get(u).get(i)] = dist[u] + 1;**

**pred[adj.get(u).get(i)] = u;**

**queue.add(adj.get(u).get(i));**

**if (adj.get(u).get(i) == dest)**

**return true;**

**}**

**}**

**}**

**return false;**

**}**

**}**