**1.What do you mean by data structure?**

Data Structures are a specialized means of organizing and storing data in computers in such a way that we can perform operations on the stored

data more efficiently.

**2.What are some applications of data structures?**

DS are used to store, retrieve and organize data. Each data structure has different application and can be used as per convenience.

**3.Advantages of linked list over array.**

Operations such as insert and delete are faster in linked list than array.

Arrays have fixed size whereas linked list have flexible size and can expand and contract at their convenience.

**4.Create a node in linked list**

struct Node{

int data;

struct Node\* next;

};

**5.Use of doubly linked list over single linked list**

Doubly linked list helps us to store addresses of both previous and next node in a node. Advantage of doubly linked list is that we can traverse

in both forward and backward direction.

**6.Difference between and array and stack.**

In array we can store data in at any place but in stack we can only store and access data from top of the stack only.

Array has values of same data type whereas stack can store values of different data types.

**7.Minimum number of queue to implement a priority queue.**

It can be done using two queues. One can be used to store the values while other can be used to store the priorities.

**8.Different traversal techniques in a tree.**

PreOrder: root->left->right

PostOrder: left->right->root

InOrder: left->root->rigth

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

It is easy to search a node in binary search tree than a simple binary because values are sorted in BST. Values smaller than

current value are on the left side of the node and values bigger than the current value are on the right side of the node.

**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.

In Computer science graphs are used to represent the flow of computation.

**11.Can binary search be applied on sorted linked list?**

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

**12.When can you tell a memory leak can occur?**

The simplest way to detect a memory leak is also the way you’re most likely to find one: running out of memory. That’s also the worst

way to discover a leak! Before you run out of memory and crash your application, you’re likely to notice your system slowing down.

If you do, it can be time to start digging into your code to figure out just what’s using up all your RAM.

Another method for memory leak detection is to use logging intelligently. Sometimes, faulty code doesn’t cause a memory leak, but your

users do. Maybe a user has uploaded a very large file that they’re trying to access on your servers. If you’re loading that entire file

into memory, you might exhaust the application’s memory through no fault of your own. Mature software organizations will often use automated

tools to detect memory leaks in running applications.

**13.How to check a given binary tree is BST or not?**

In a BST all the values in the left subtree of any node are smaller than it. Whereas all the values in its right subtree are greater than it.

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

Stack is ideal to perform recursion operation. Because of its 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.

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

1) Expression Handling: 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. These expressions are not so much familiar to the infix expression, but they have some great advantages also. We do not need to maintain operator ordering, and parenthesis. After converting into prefix or postfix notations, we have to evaluate the expression to get the result. For that purpose, also we need the help of stack data structure.

2) Backtracking : Backtracking is one of the algorithm designing technique. For that purpose, we dive into some way, if that way is not efficient, we come back to the previous state and go into some other paths. To get back from current state, we need to store the previous state. For that purpose, we need stack. Some examples of backtracking is finding the solution for Knight Tour problem or N-Queen Problem etc.

3) Recursion: Recursion is a process where a function iteratively calls itself until a base case is reached. An example of recursion is finding the factorial of a number.

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

**17.Sort a stack using temporary stack.**

1.Create a temporary stack.

2.Repeat until input stack is not empty:

* Pop element from input stack and call it temp.
* While temporary stack is not empty and top of temporary stack is greater than temp, pop from temporary stack and push it to the input stack.
* Push temp in temporary stack.

3.Sorted elements are in temporary stack.

**18.Program to reverse a queue.**

import java.util.\*;

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(100);

queue.add(90);

queue.add(80);

queue.add(70);

queue.add(60);

reversequeue();

Print();

}

}

**19.Reverse first k elements of queue.**

import java.util.\*;

public class Reverse {

static Queue<Integer> queue;

static void reverseFirstKElements(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);

int k = 3;

reverseFirstKElements(k);

Print();

}

}

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

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.**

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 in array by its rank.**

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.**

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;

}

}

class Answer23 {

public static void main(String[] args) {

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.Kth smallest element in unsorted array.**

package CipherSchool;

import java.util.\*;

class Answer24 {

static int kthsmallest(int arr[],int k)

{

Arrays.sort(arr);

return arr[k-1];

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("ENter size");

int n=sc.nextInt();

int array[]=new int[n];

System.out.println("ENter array");

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

{

array[i]=sc.nextInt();

}

System.out.println("Enter k");

int K=sc.nextInt();

System.out.println(kthsmallest(array,K));

}

}

**25.Shortest path between two vertices.**

The idea is to perform BFS from one of given input vertex(u). At the time of BFS maintain an array of distance[n] and initialize it to zero for all vertices. Now, suppose during BFS, vertex x is popped from queue and we are pushing all adjacent non-visited vertices(i) back into queue at the same time we should update distance[i] = distance[x] + 1;.  
Finally distance[v] gives the minimum number of edges between u and v.