**Q1.** What do you mean by a Data structure?

**Ans.** Data Structure is a representation of the logical relationship existing between individual elements of data. It is about structuring or arranging and storing of data in such a way so that it can be accessed and used efficiently. It is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

**Q2.**  What are some of the applications of DS?

**Ans.**  I. Linked List – Web browser history (or on pressing back button) last web page is fetched through the previous node’s data as same as to that of traversal of a linked list.

II. ‘Undo’ or ‘Redo’ operation because this operation is accomplished by keeping all text changes in a stick.

III. Language processing i.e. syntax check for matching braces is basically implemented using stack. To switch between the applications opened in the taskbar (that we use by alt+ tab+ left\right cursor key) is done using stack.

IV. CPU Scheduling and Disk Scheduling are two of the applications which are implemented using queue.

V. Maintaining different directories, folders and sub-folders which is basically done using ‘Tree’ DS. ‘HTML’ text, attributes are stored and can be fetched via tree called ‘Document Object Model’ (DOM). Network routing is also handled by tree data structure.

**Q3.** What are the advantages of a Linked list over an array?

**Ans.** Advantages of Linked List over an array are:

* Less wastage of memory in case of linked list as there is dynamic allocation of the data elements.
* Insert and delete operations are comparatively easier to that of an array.
* Linked list allows fast access time and furthermore can be expanded in constant time without facing any memory overhead.

**Q4.** Write the syntax in C to create a node in the singly linked list.

Ans. struct node{

int data;

struct node \*next;

};

struct node \*start , \*end = NULL;

 void addNode(int data) {

   struct node \*newNode = (struct node\*)malloc(sizeof(struct(node));

    newNode->data = data;

    newNode->next = NULL;

   if (start == NULL) {

        start = newNode;

        end= newNode;

    }

    else{

        end->next = newNode;

        end = newNode;

    }

}

**Q5.** What is the use of a doubly-linked list when compared to that of a singly linked list?

**Ans.** I. A doubly-linked list can be traverse in both the directions i.e. forward and backward.

II. The operations are simpler and more efficient (other than first nodes) as there is no need to keep track of the previous node during traversal.

III. Insertion and deletion at a desired position takes a complexity of O(1).

**Q6.** What is the difference between an Array and Stack?

**Ans.** Array**:** It is a data structure which contains similar datatypes. It is of static size because it gets fixed once allocated. Objects can be accessed at any time in random fashion due to their arrangements. It supports many operations such as insertion, deletion, traversing, merging and sorting.

Stack: It is also a linear data structure represented by sequential collection of elements in a fixed manner. Object arrangement is in such a manner that deletion and insertion happens only at one end. Elements can be added or removed in ‘Last in First Out’ (LIFO) manner in which the last inserted being the first one to be retrieved afterwards. It supports only two main operations push (to add objects) and pop (to remove objects). It can contain value which belong to different datatypes.

**Q7. What are the minimum number of Queues needed to implement the priority queue?**

**Ans.** Two queues are required to implement the priority queue; one to store the values and the other one to store the priorities.

**Q8. What are the different types of traversal techniques in a tree?**

**Ans.** There are mainly three types of traversal techniques in a tree:

1. Inorder traversal.
2. Preorder traversal.
3. Postorder traversal.

**Q9. Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?**

**Ans.** Searching in ‘Binary Search Tree’ is efficient than that of a simple binary tree because the keys are already in sorted order for fast lookup i.e. the values of nodes in the left subtree are always less than or equal to the root node and the values of nodes in the right subtree are always greater than the root node. Hence, eventually it gets easier to perform the searching in binary search tree.

**Q10. What are the applications of Graph DS?**

**Ans.** I. Social Graphs: Social graphs draw edges between you and the people, places and things you interact with online. Facebook's Graph API is perhaps the best example of application of graphs to real life problems.

II. Google's Knowledge Graph: A knowledge graph has something to do with linking data and graphs and some kind of graph-based representation of knowledge.

III. Recommendation Engines: The Local Graph API promises to make it easier for developers to integrate Yelp's data and share great local businesses through their apps. It creates edges with relationships such as the location of a business with a certain name, the opening hours of a business, the reviews of a business, and the category of a business.

IV. Google Maps Platform (Maps, Routes APIs): Google maps and routes APIs are classic Shortest Path APIs. This a graph problem that's very easy to solve with edge-weighted directed graphs (digraphs).

**Q11. Can we apply Binary search algorithm to a sorted Linked list?**

**Ans.** Yes we can apply binary search algorithm to a sorted linked list. If it is a singly linked list, which uses only one pointer, it is not easy to find its middle element. To mid of singly linked list, we use two pointer approaches.

**Q12. When can you tell that a Memory Leak will occur?**

**Ans.** A memory leak is any portion of an application which uses memory without eventually freeing it. By memory, we’re talking about RAM, not permanent storage, like a hard drive. A memory leak is caused when you allocated memory, haven't yet deallocated it, and you will never be able to deallocate it because you can't access it anymore.

**Q13. How will you check if a given Binary Tree is a Binary Search Tree or not?**

**Ans.** Binary Tree is a BST based on the following conditions:

a. The left subtree of a node contains only nodes with values less than or equal to the root node’s value.

b. The right subtree of a node contains only nodes with values greater than the root node’s value.

c. Both the left and right subtrees should also be BST.

**Q14. Which data structure is ideal to perform recursion operation and why?**

**Ans.** Stack is the data structure suitable to perform recursion operation because of its characteristics ‘Last in First Out’ (LIFO). Therefore, it knows to whom it should return when the function has to return. On the other hand, recursion makes use of the system stack for storing the return addresses of the function calls.

**Q15. What are some of the most important applications of a Stack?**

**Ans.**  I**.** Infix to Postfix or Infix to Prefix Conversion: 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.

II. Backtracking procedure: 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.

III. Great use of stack is during the function call and return process. When we call a function from one other function, that function call statement may not be the first statement. After calling the function, we also have to come back from the function area to the place, where we have left our control.

**Q16. Sorting a stack using a temporary stack.**

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

public class StackSort {

public static Stack<Integer> sortStack(Stack<Integer> input){

Stack<Integer> tmpStack = new Stack<Integer>();

while(!input.isEmpty()) {

int tmp = input.pop();

System.out.println("Element taken out: "+tmp);

while(!tmpStack.isEmpty() && tmpStack.peek() > tmp) {

input.push(tmpStack.pop());

}

tmpStack.push(tmp);

System.out.println("input: "+input);

System.out.println("tmpStack: "+tmpStack);

}

return tmpStack;

}

public static void main(String a[]){

Stack<Integer> input = new Stack<Integer>();

Scanner scanner = new Scanner(System.in);

for(int i=0;i<7;i++)

{

input.add(scanner.nextInt());

}

// System.out.println("input: "+input);

System.out.println("final sorted list: "+sortStack(input));

} }

**Q17. Program to reverse a queue.**

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

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>();

for(int i=0;i<10;i++)

{

queue.add(i );

}

reversequeue();

        Print();

    }

}

**Q18. Program to reverse first k elements of a queue.**

**Ans.** import java.util.\*;

class ReversingTheFirstKElementsOfAQueue {

public void reverseKElements(Queue<Integer> queue, int k) {

if (k < 0 || k >= queue.size() || queue.isEmpty()) {

System.out.println("Invalid Input");

return;

}

int n = queue.size();

Stack<Integer> stack = new Stack<>();

for (int i = 0; i < k; i++) {

int curr = queue.poll();

stack.push(curr);

}

while (!stack.isEmpty()) {

int curr = stack.pop();

queue.add(curr);

}

for (int i = 0; i < n - k; i++) {

int curr = queue.poll();

queue.add(curr);

}

for (Integer i : queue) {

System.out.print(i + " ");

}

System.out.println();

}

}

class Main{

public static void main(String[] args) {

ReversingTheFirstKElementsOfAQueue robj = new ReversingTheFirstKElementsOfAQueue();

Queue<Integer> q1 = new LinkedList<>();

int k1 = 3;

Scanner scanner = new Scanner(System.in);

for(int i=0;i<8;i++)

{

q1.add(scanner.nextInt());

}

robj.reverseKElements(q1, k1);

Queue<Integer> q2 = new LinkedList<>();

int k2 = 2;

for(int i=0;i<5;i++)

{

q2.add(scanner.nextInt());

}

robj.reverseKElements(q2, k2);

}

}

**Q19. Program to return the nth node from the end in a linked list.**

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

public class LinkedList{

 private Node head;

private static class Node {

private int value;

private Node next;

Node(int value) {

this.value = value;

}

}

public void addToTheLast(Node node) {

if (head == null) {

head = node;

} else {

Node temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = node;

}

}

public void printList() {

Node temp = head;

while (temp != null) {

System.out.format("%d ", temp.value);

temp = temp.next;

}

System.out.println();

}

public Node nthFromLastNode(Node head,int n)

{

Node firstPtr=head;

Node secondPtr=head;

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

firstPtr=firstPtr.next;

}

while(firstPtr!=null)

{

firstPtr=firstPtr.next;

secondPtr=secondPtr.next;

}

return secondPtr;

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

//Node head=new Node(5);

Scanner scanner = new Scanner(System.in);

//list.addToTheLast(head);

For(int i=0;i<6;i++)

{

List.addToTheLast(new Node(scanner.nextInt()));

}

list.printList();

Node nthNodeFromLast= list.nthFromLastNode(head,3);

System.out.println("3th node from end is : "+ nthNodeFromLast.value);

}

}

**Q20. Reverse a linked list.**

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

public class LinkedListProblem {

public static void main(String[] args){

 SinglyLinkedList.Node head = new SinglyLinkedList.Node(1);

SinglyLinkedList linkedlist = new SinglyLinkedList(head);  
Scanner scanner = new Scanner(System.in);

For(int i=0;i<6;i++)

{

linkedlist.add(new SinglyLinkedList.Node(sc.nextInt()));

}

linkedlist.print();

linkedlist.reverse();

linkedlist.print();

}

}

class SinglyLinkedList {

static class Node {

 private int data;

 private Node next;

public Node(int data) {

this.data = data; }

public int data() {

return data; }  
 public Node next()

{ return next; }

}

 private Node head;

 public SinglyLinkedList(Node head) {

this.head = head; }

public void add(Node node) {

Node current = head;

while (current != null) {

if (current.next == null) {

current.next = node;

break; }

current = current.next; }

}

public void print() {

Node node = head;

while (node != null) {

System.out.print(node.data() + " ");

node = node.next(); }

 System.out.println("");

}

public void reverse() {

Node pointer = head;

Node previous = null, current = null;

while (pointer != null) {

current = pointer;

pointer = pointer.next;

current.next = previous;

 previous = current; head = current;

}

}

}

**Q21. Replace each element of the array by its rank in the array.**

**Ans.** import java.util.\*;

class Rank{

    public void ranking(int[] arr){

        Map<Integer, Integer> map = new TreeMap<>();

    for (int i = 0; i < arr.length; i++) {

    map.put(arr[i], i);

}

int rank\_val = 1;

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

arr[entry.getValue()] = rank\_val++;

    }

}

}

class Main

 {

     public static void main(String[] args)

{

    Rank robj = new Rank();

    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    System.out.println("\n Array size is ::  "+n);

    int[] arr = new int[n];

    for(int i=0;i<arr.length;i++){

        arr[i] = sc.nextInt();

    }

System.out.println(“\n Array elements are ::\n”+Arrays.toString(arr));

    robj.ranking(arr);

System.out.println(“\n According to the rank of array elements ::\n”);

    System.out.println(Arrays.toString(arr));

}

}

**Q22. Check if a given graph is a tree or not.**

**Ans.**  A graph is said to be a tree if and only if there is a unique path between any two vertices. If a cycle is there in a graph then it is definitely not a tree because it violates the property of a tree.

**Q23. Find out the Kth smallest element in an unsorted array.**

**Ans.** import java.util.\*;

class KthElement{

    public String sorting(int arr[]){

        Arrays.sort(arr);

        return "Array after sorting :: "+Arrays.toString(arr);

    }

    public int get(int arr[],int k){

        return arr[k-1];

    }

}

class Main{

    public static void main(String Arg[])

    {

        KthElement kob = new KthElement();

        Scanner scanner = new Scanner(System.in);

        System.out.println("\n Enter array size ::\n");

        int n = scanner.nextInt();

        int arr[] = new int[n];

        System.out.println("\n Enter array elements ::\n");

        for(int i=0;i<n;i++){

            arr[i] = scanner.nextInt();

        }

        System.out.println("\n Array is :: "+Arrays.toString(arr));

        System.out.println("\n "+kob.sorting(arr));

        System.out.println("\n Enter the k value :: \n");

        int k = scanner.nextInt();

        System.out.println("\n K value is :: "+k);

        System.out.println("\n"+k+"th  smallest element is  :: "+kob.get(arr , k));

    }

}

**Q24. How to find the shortest path between two vertices?**

**Ans.** It can be found out using any of the single shortest path algorithms (like Dijkstra algo.) This is a greedy approach to solve the given problem as it is a paradigm that always chooses the next piece of solution that offers the most beneficial and the obvious. Choosing a local optimal solution which eventually becomes global optimal solution. We keep visiting each node and its neighbor to find the shortest path between the two and keep finding the best solution for the current scenario and finally obtaining the end result which becomes the best solution for the given problem.