Q1: What is data structure and algorithms?

ANS : A data structure is a named location that can be used to store and organize data. And, an algorithm is a collection of steps to solve a particular problem. Learning data structures and algorithms allow us to write efficient and optimized computer programs.

Data Structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. It is a special format for Organizing and Storing Data.

Data Structure are classified into two types:

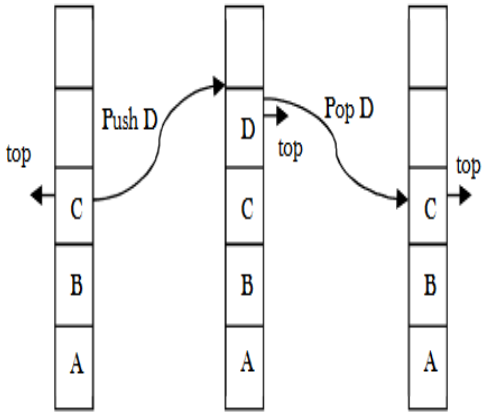
1. Linear Data Structures: Elements are accessed in a sequential order but it is not compulsory to store all elements sequentially.
   Examples: Linked Lists, Stack and Queues.
2. Non Linear Data Structures: Elements of this data structure are stored/accessed in a non-linear order.
   Examples: Trees and Graphs

An Algorithm is the step-by-step unambiguous instructions to solve a given problem

Q2: What is stack? Write a full program of stack.

Stack: A Stack is a simple data structure used for storing data. In Stack the order in which data arrives is important. A pile of plates in a cafeteria is a good example of stack.

A stack is an ordered list in which insertion and deletion are done at one end called top. The last element inserted is the first one to be deleted. Hence, it is called the Last In First Out (**LIFO**) of First In Last Out (**FILO**) .



Basic Operations of Stack

A stack is an object (an abstract data type - ADT) that allows the following operations:

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

Q3: What is the condition of stack underflow and overflow?

Now that we have a basic picture in mind of what a stack conceptually looks like, we can define what underflow and overflow are.

Stack underflow happens when we try to pop (remove) an item from the stack, when nothing is actually there to remove. This will raise an alarm of sorts in the computer because we told it to do something that cannot be done.

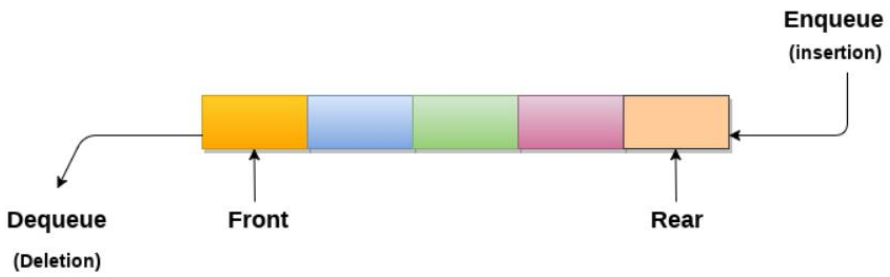
Stack overflow happens when we try to push one more item onto our stack than it can actually hold. You see, the stack usually can hold only so much stuff. Typically, we allocate (set aside) where the stack is going to be in memory and how big it can get. So, when we stick too much stuff there or try to remove nothing, we will generate a stack overflow condition or stack underflow condition, respectively.

Q4: What is Queue?

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



There are Four Types of Queue:

1. **Linear Queue**
2. **Circular Queue**
3. **Priority Queue**
4. **Deque**

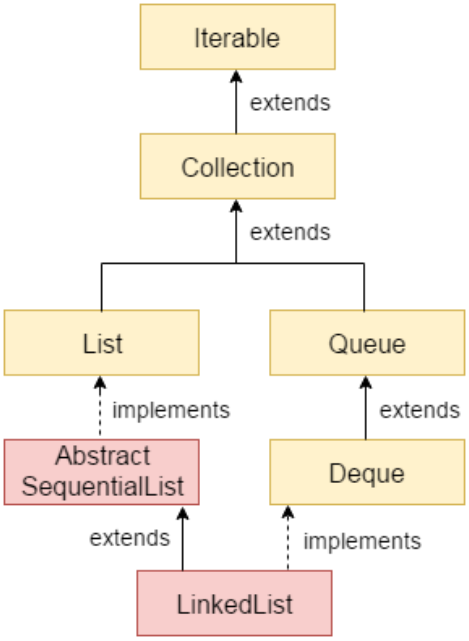
Q5: What is Big-O Notation?

Big O notation is an asymptotic notation that measures the performance of an algorithm and used for calculating the running time complexity of an algorithm by simply providing the order of growth of the function.

This notation provides an upper bound on a function which ensures that the function never grows faster than the upper bound. So, it gives the least upper bound on a function so that the function never grows faster than this upper bound.

Q6: What is LinkedList?

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the Abstract List class and implements List and Deque interfaces.



The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.

There are two Types of Linked List:

1. Singly Linked List
2. Doubly Linked List

Q7: Recursion

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

```
import java.util.ArrayList;
import java.util.Scanner;

public class Recursion {
    //    Simple Fib
    public static int fibonacci2(int number){
        if(number == 1 || number == 2){
            return 1;
        }
        int fibo1=1, fibo2=1, fibonacci=1;
        for(int i= 3; i<= number; i++){

            //Fibonacci number is sum of previous two Fibonacci number
            fibonacci = fibo1 + fibo2;
            fibo1 = fibo2;
            fibo2 = fibonacci;

        }
        return fibonacci; //Fibonacci number

    }
    //    Factorial in Recursion
    static int factorial(int n){
        if (n == 0 || n == 1){
            return 1;
        }
```

```
        else{
            return n * factorial(n-1);
        }
    }
    //    Fibonacci Series in Recursion
    static int fibonacci(int n){
        if(n <= 1){
            return n;
        }
        return fibonacci(n-1) + fibonacci(n-2);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Number: ");
        int n = sc.nextInt();
        System.out.println("Here is Your Factorial of " + n + " is " +
factorial(n));
        System.out.println("Here is Your Fibonacci of " + n + " is " +
fibonacci(n));

System.out.println("_____\n");
        System.out.print("Fibonacci series -->");
        for (int i = 0; i <= n; i++) {
            System.out.print(" " + fibonacci(i));
        }
    }
}
```

Q8: Infix, Prefix & Postfix

Infix Notation

We write expression in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation

In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands. For example, **+ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**.

Postfix Notation

This notation style is known as **Reversed Polish Notation**. In this notation style, the operator is **postfix**ed to the operands i.e., the operator is written after the operands. For example, **ab+**. This is equivalent to its infix notation **a + b**.

Q->Write a prgram of stack using a singly linkedlist which contains prime number's only

---

```
//Q->Write a program  of stack using a singly linked list which contains prime number's only

import java.util.*;

class Nod
{
    int head;
    Nod next;
}

public class SinglyLinkedList_OnlyPrimeNumbers
{
    Nod head;
    SinglyLinkedList_OnlyPrimeNumbers()
    {
        head = null;
    }

    //    This Code will check is Number is prime or not
    static int isPrime(int n){

        int i,m=0,Counter=0;

        m = n/2;

        if(n==0 || n==1){
            return m;
        }else{
            for(i=2; i<=m; i++){
                if(n%i == 0){
                    Counter=1;
                    break;
                }
            }
            if(Counter==0)  { return n; }
        }//end of else
        return m;
    }

    void traverse(){
        if (head == null){
            System.out.println("List is Empty");
        }
        else{
            Nod curr;
            for(curr = head; curr != null; curr = curr.next){
                System.out.println(" "+ curr.head);
            }
        }
    }
    void addNode()
    {

        System.out.println("Enter your roll no.");
        Scanner sc2 = new Scanner(System.in);
        int rn = sc2.nextInt();
        if(isPrime(rn) == rn){
            // creating a new node
            Nod newNode = new Nod();
            newNode.head=rn;
            newNode.next=null;
            if (head==null)
            {
```

```
                head=newNode;
            }
            else
            {
                Nod current = head;
                while (current.next!=null)
                {
                    current=current.next;
                } current.next=newNode;
            }
            System.out.println("Data inserted...");
        }
        else{
            System.out.println("Sorry "+ rn + " is not a Prime Number");
        }

    }

    public static void main(String[] args)
    {
        SinglyLinkedList_OnlyPrimeNumbers obj = new
SinglyLinkedList_OnlyPrimeNumbers();

        while (true)
        {
            System.out.println("Press 1 to insert");
            System.out.println("Press 2 to traverse");
            System.out.println("Press 3 to exit");
            System.out.println("Enter your choice");
            Scanner sc = new Scanner(System.in);
            int ch = sc.nextInt();

            switch (ch)
            {
                case 1:
                    obj.addNode();
                    break;
                case 2:
                    obj.traverse();
                    break;
                case 3:
                    System.exit(0);
                default:
                    System.out.println("Wrong Choice");
            }
        }
    }
}
```

Q-Find the highest value in a circular queue

```
// Find the Highest value in a Circular Queue

import java.util.*;
public class CircularQueue_FindHighest {

    int ar[];
    int front,rear;
    int high = 0;

    CircularQueue_FindHighest() {
        ar = new int [5];
        front=rear=-1;
    }

    int checkHighest(int data){
        int def = high;
        if(data > def){
            def = data;
        }
        return def;
    }

    void insert() {
        if ((front==0 && rear==4) || (rear==front-1)){
            System.out.println("Circular Queue is full!!!");
        }
        else{
            System.out.println("Enter data...");
            Scanner sc2 = new Scanner(System.in);
            int data = sc2.nextInt();
            if (front==-1) { front=0; }
            if (rear<4) { rear+=1; }
            else if (rear==4 && front!=0) { rear=0; }
            ar[rear]=data;
            high = checkHighest(data);
            System.out.println("data inserted...");
        }
    }
    void checkBecauseOfDeletion(){
        try {
            if(high == ar[front]){
                high = ar[front+1];
            }
        }
        catch (Exception e){
            high = 0;
        }

    }

    void delete() {
        if (front==-1 || rear==-1)
            System.out.println("Circular Queue is empty!!!");
        else {
            if (front==rear){
                System.out.println(" deleted : " + ar[front]);
                checkBecauseOfDeletion();
                front=rear=-1;
            }
            else if (front<rear){
                System.out.println("deleted : " + ar[front]);
                checkBecauseOfDeletion();
                front+=1;
            }
            else if(front==4){
                System.out.println("deleted : " + ar[front]);
```

```
                checkBecauseOfDeletion();
                front=0;
            }
            else if (rear<front){
                System.out.println("deleted : " + ar[front] );
                checkBecauseOfDeletion();
                front+=1;
            }
        }
    }

    void findHighest(){
        System.out.println("\nHighest Value is: " + high);
        System.out.println();
    }

    void traverse() {
        if (front==-1 || rear==-1){
            System.out.println("Circular Queue is empty!!!");
        }
        else {
            if (front<=rear){
                for (int i = front; i<=rear; i++){
                    System.out.print(" "+ar[i]);
                }
            }
            else {
                for (int i=front;i<=4;i++)
                {
                    System.out.print(" "+ar[i]);
                }
                for (int i=0;i<=rear;i++){
                    System.out.print(" "+ar[i]);
                }
            }
        }

    }

    public static void main(String[] args) {
        CircularQueue_FindHighest obj = new CircularQueue_FindHighest();
        while (true) {
            System.out.println("\nPress 1 for insert!");
            System.out.println("Press 2 for delete!");
            System.out.println("Press 3 for traverse!");
            System.out.println("Press 4 for Find Highest!");
            System.out.println("Press 5 for exit!");

            System.out.println("Enter your choice....");
            Scanner sc2 = new Scanner(System.in);
            int ch = sc2.nextInt();

            switch (ch) {
                case 1 -> obj.insert();
                case 2 -> obj.delete();
                case 3 -> obj.traverse();
                case 4 -> obj.findHighest();
                case 5 -> System.exit(0);
                default -> System.out.println("Invalid Choice!!!");
            }
        }
    }
}
```

Q-Find, How many even and odd no's in a stack by using linked list

```java
//Q-Find, How many even and odd no's in a stack by using linked list

import java.util.*;

class Node2
{
    int head;
    Node2 next;
}

public class SinglyLinkedList_CountEvenOdd
{
    //    Creating Even Odd Counter
    int oddCounter = 0;
    int evenCounter = 0;

    Node2 head;
    SinglyLinkedList_CountEvenOdd()
    {
        head = null;
    }

    void counterIncrement(int data){
        if(data % 2 == 0){
            evenCounter += 1;
        }
        else {
            oddCounter += 1;
        }
    }

    void traverse(){
        if (head == null){
            System.out.println("List is Empty");
        }
        else{
            Node2 curr;
            for(curr = head; curr != null; curr = curr.next){
                System.out.println(" "+ curr.head);
            }
        }
    }

    void countEvenOdd(){
        System.out.println("\nTotal Even Values Are: " + evenCounter);
        System.out.println("Total Odd Values Are: " + oddCounter);
        System.out.println();
    }

    void addNode()
    {
        System.out.println("Enter your roll no.");
        Scanner sc2 = new Scanner(System.in);
        int rn = sc2.nextInt();
        counterIncrement(rn);
        // creating a new node
        Node2 newNode = new Node2();
        newNode.head=rn;
        newNode.next=null;
        if (head==null)
        {
            head=newNode;
        }
        else
        {
            Node2 current = head;
            while (current.next!=null)
            {
```

```
                current=current.next;
            } current.next=newNode;
        }
        System.out.println("Data inserted...");
    }

    public static void main(String[] args)
    {
        SinglyLinkedList_CountEvenOdd obj = new SinglyLinkedList_CountEvenOdd();

        while (true)
        {
            System.out.println("Press 1 to insert");
            System.out.println("Press 2 to traverse");
            System.out.println("Press 3 to print Count Of Even & Odd Nums");
            System.out.println("Press 4 to exit");
            System.out.println("Enter your choice");
            Scanner sc = new Scanner(System.in);
            int ch = sc.nextInt();

            switch (ch)
            {
                case 1:
                    obj.addNode();
                    break;
                case 2:
                    obj.traverse();
                    break;
                case 3:
                    obj.countEvenOdd();
                    break;
                case 4:
                    System.exit(0);
                default:
                    System.out.println("Wrong Choice");
            }
        }
    }
}
```

Q-> Complete a program of Singly Queue ----------using linkedlist

```
interface ADTQueue
{
        abstract void insert(int data);
        abstract int delete();
        abstract booelan isEmpty();
        abstract void traverse();
        abstract boolean search(int sea);
}
class Queueexample implements ADTQueue
{
        Queueexample()
        {
        }
        public  void insert(int data)
        {
        }
        public int delete()
        {
        }
        public boolean isEmpty()
        {
        }
        public void traverse()
        {
        }
        public boolean search(int sea)
        {
        }
        public static void main(String args[])
        {

        }
}
```

```
//Q-> Complete a program of Singly Queue ------ using linkedlist

import java.util.Scanner;

interface ADTQueues{
    abstract void insert(int data);
    abstract int delete();
    abstract boolean isEmpty();
    abstract  void traverse();
    abstract boolean search(int sea);
}

class Queueexample implements ADTQueues {
    int ar[];
    int Front,Rear;
    Queueexample(){
        ar = new int[5];
        Front = Rear = -1;
    }

    @Override
    public void insert(int data) {
        if (Rear==4)
        {
            System.out.println("Queue is full");
```

```
        }
        else
        {
            if (Front==-1)
            {
                Front=0;
            }
            Rear=Rear+1;
            ar[Rear]=data;
            System.out.println("Data Inserted...");
        }
    }

    @Override
    public int delete() {
        if (Front==-1 && Rear==-1)
        {
            System.out.println("Queue is empty!!!");
        }
        else
        {
            if (Front==Rear)
            {
                System.out.println("Deleted: " + ar[Front]);
                Front=Rear=-1;
                return Front;
            }
            else
            {
                System.out.println("Deleted: " + ar[Front]);
                Front+=1;
                return Front;
            }
        }
        return Front;
    }

    @Override
    public boolean isEmpty() {
        if (Front==-1 && Rear==-1)
        {
            return true;
        }
        return false;
    }

    @Override
    public void traverse() {
        if (Front == -1 || Rear == -1)
        {
            System.out.println("Queue is empty!!!");
        }
        else
        {
            for (int i = Front; i<=Rear;i++)
            {
                System.out.print(" "+ ar[i]);
            }
        }
    }

    @Override
    public boolean search(int sea) {
        return false;
    }

    public static void main(String[] args) {
        Queueexample obj = new Queueexample();

        while (true)
```

```
        {
            System.out.println("Press 1 for insert!");
            System.out.println("Press 2 for delete!");
            System.out.println("Press 3 for traverse!");
            System.out.println("Press 4 for Search!");
            System.out.println("Press 5 for isEmpty!");
            System.out.println("Press 6 for Exit!");

            System.out.println("Enter your choice....");
            Scanner sc2 = new Scanner(System.in);
            int ch = sc2.nextInt();

            switch (ch) {
                case 1: {
                    System.out.println("Enter data!!!");
                    int data = sc2.nextInt();
                    obj.insert(data);
                }
                case 2: {
                    obj.delete();
                }
                case 3: {
                    obj.traverse();
                }
                case 4: {
                    obj.search(5);
                }
                case 5: {
                    obj.isEmpty();
                }
                case 6: {
                    System.exit(0);
                }
                default: {
                    System.out.println("Invalid Choice!!!");
                }
            }
        }
    }
}
```

Q-> Complete a program of STACK ----------

```
interface ADTstackdem
{
        abstract void push(int data);
        abstract int pop();
        abstract int peek();
        abstract booelan isEmpty();
        abstract void traverse();
}
class stackexample implements ADTstackdem
{
        stackexample()
        {
        }
        public  void push(int data)
        {
        }
        public int pop()
        {
        }
        public int peek()
        {
        }
        public boolean isEmpty()
        {
        }
        public void traverse()
        {
        }
        public static void main(String args[])
        {
        }
}
```

```
import java.util.Scanner;

interface ADTStackDem {
    abstract void push(int data);
    abstract void pop();
    abstract void peek();
    abstract boolean isEmpty();
    abstract void traverse();
}

class StackExample implements ADTStackDem{

    int[] arr;
    int top;
    StackExample()
    {
        arr = new int[5];
        top=-1;
    }

    @Override
    public void push(int data) {
        if (top==4)
        {
            System.out.println("Stack is full!");
        }
        else
        {
            top=top+1;
            arr[top]=data;
            System.out.println("Data inserted!!!");
        }
    }
```

```
    @Override
    public void pop() {
        if (isEmpty()==true)
        {
            System.out.println("Stack is empty!!!");
        }
        else
        {
            System.out.println("Deleted element...");
            top=top-1;
        }
    }

    @Override
    public void peek() {
        if (isEmpty()==true)
        {
            System.out.println("Stack is empty");
        }
        else
        {
            System.out.println(arr[top]);
        }
    }

    @Override
    public boolean isEmpty() {
        if (top == -1){
            return true;
        }
        return false;
    }

    @Override
    public void traverse() {
        if (isEmpty()==true)
        {
            System.out.println("Stack is empty");
        }
        else
        {
            for (int i=top;i>=0;i--)
            {
                System.out.print(arr[i]);
                System.out.print(" \n");
            }
        }
    }

    public static void main(String[] args) {
        StackExample obj =new StackExample();
        while (true)
        {
            System.out.println("Press 1 for push");
            System.out.println("Press 2 for pop");
            System.out.println("Press 3 for traverse");
            System.out.println("Press 4 for peek");
            System.out.println("Press 5 for isEmpty");
            System.out.println("Press 6 for exit");
            System.out.println();
            System.out.println("Enter your choice...");

            Scanner sc = new Scanner(System.in);
            int choice = sc.nextInt();
            switch (choice)
            {
                case 1:
                    System.out.println("Enter any data");
                    Scanner sc2 = new Scanner(System.in);
                    int data = sc2.nextInt();
                    obj.push(data);
                    break;
                case 2:
```

```
                    obj.pop();
                    break;
                case 3:
                    obj.traverse();
                    break;
                case 4:
                    obj.peek();
                    break;
                case 5:
                    obj.isEmpty();
                    break;
                case 6:
                    System.exit(0);
                default:
                    System.out.println("Wrong choice");
            }
        }
    }
}
```

Q-
a) Convert infix to postfix
(input)infix - A+B*C
(Output)postfix - ABC*+

steps-

Input String	Output Stack	Operator Stack
A+B*C	A	
A+B*C	A	+
A+B*C	AB	+
A+B*C	AB	+*
A+B*C	ABC	+*
A+B*C	ABC*+	
A+B*C	ABC*+	

b)convert postfix to infix
(input)postfix - abc*+
(output) infix - a+b*c

steps-

abc*+	bc*+	a
abc*+	c*+	ab
abc*+	*+	abc
abc*+	+	a(b*c)
abc*+		(a+(b*c))

Question Never Come in EXAM

Q->Write a program of singly linkedlist which contains student data
... studentid(int) and studentname (String)
ex.

```
class Student
{
        int studentid;
```

```
        String studentname;
        Student next;
}
```

Q- Write a program to transfer even data from singly linked list to doubly linked list

Ex - Singly linked List
2,10,5,21,14,31,11,42,55,4

After transfer to doubly linked list
2,10,14,42,4

Q-5 write a program of doubly linked list which contains 3 fields studentId ,studentName, studentAge

```
class Student
{
        int studentid;
        String studentname;
        int studentAge;
        Student next;
        Student prev;
}
```

input:
1 Aman 20
5 Srajan 21
3 Rajnish 18
10 Naman 17

desired Output:
01) By name
1 Aman 20
10 Naman 17
3 Rajnish 18
5 Srajan 21
2) By ID
1 Aman 20
3 Rajnish 18
5 Srajan 21
10 Naman 17
3) By Age
10 Naman 17
3 Rajnish 18
1 Aman 20
5 Srajan 21

Q-Write a program of doubly linkedlist which perform following operation :
-insert a node at begining
-delete a node at last
