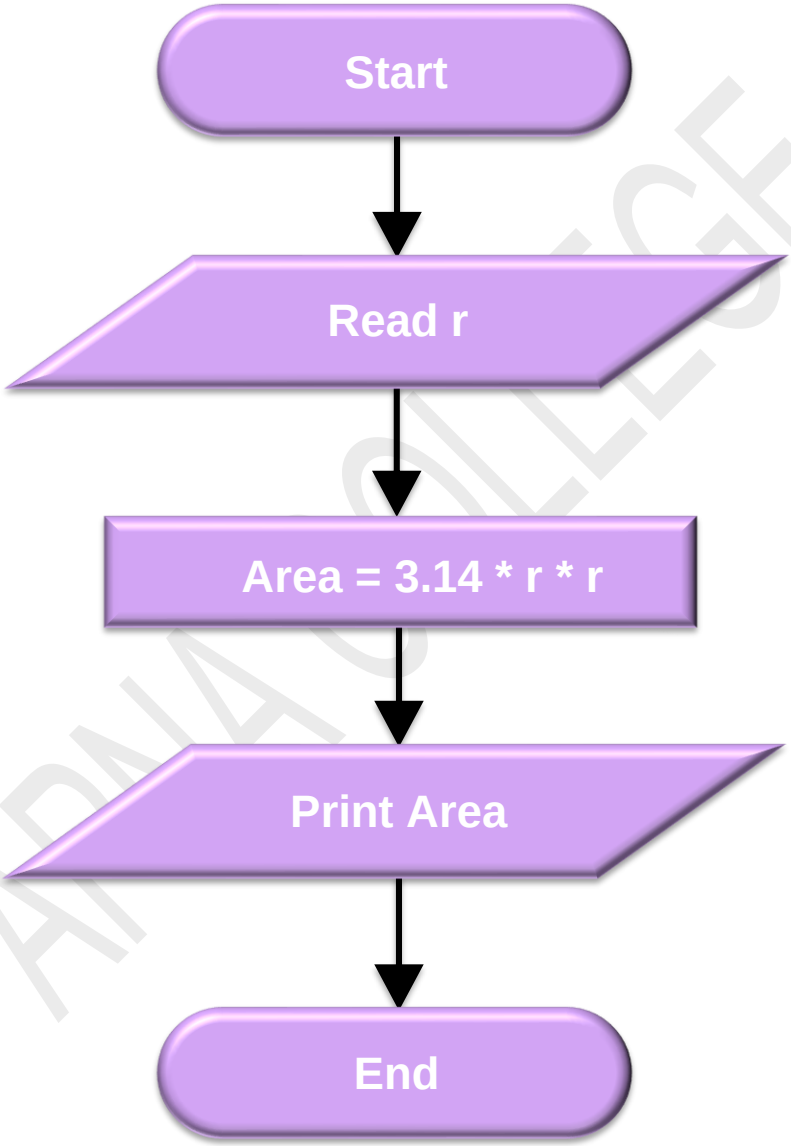
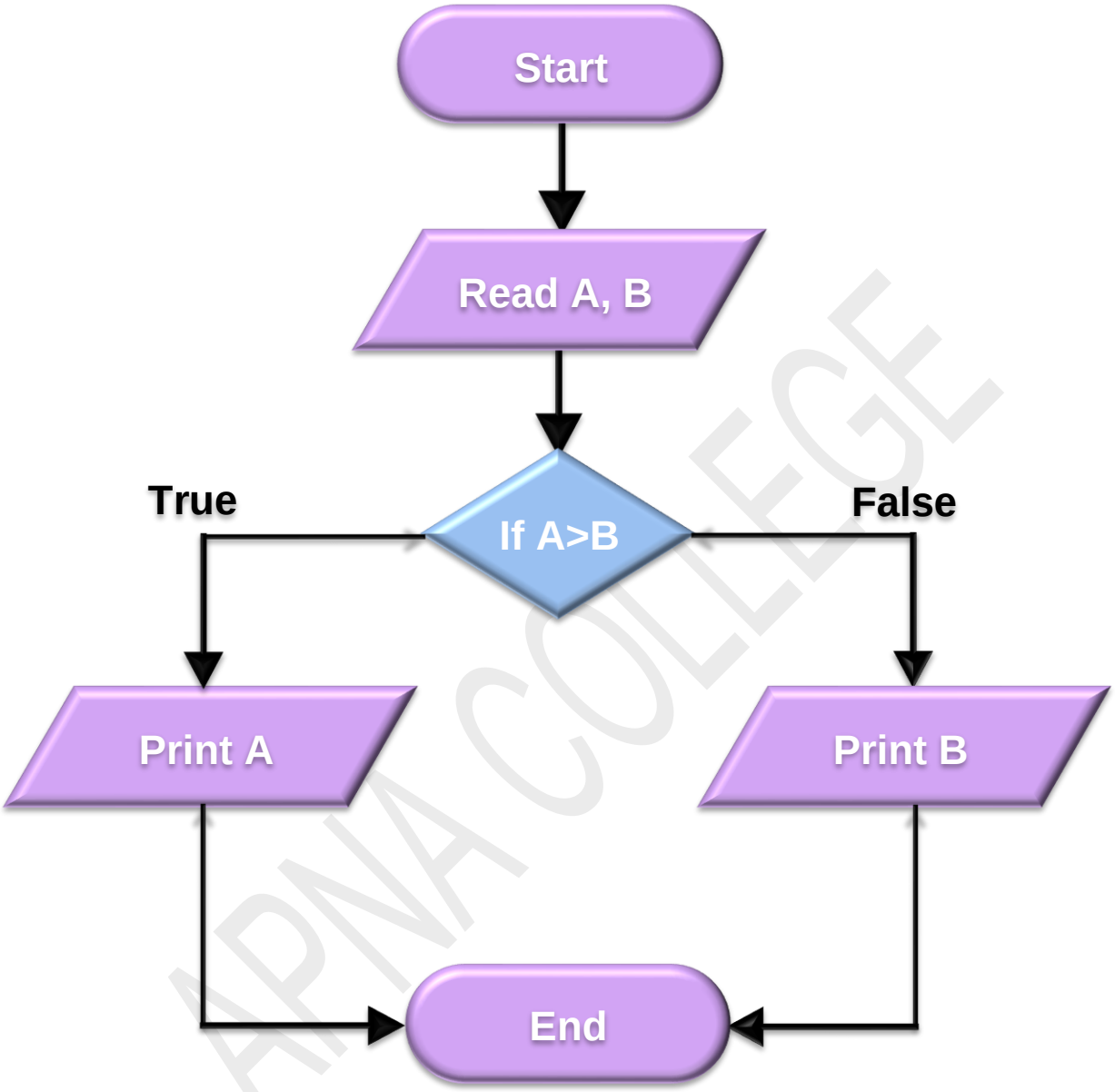
# FLOWCHARTS SOLUTIONS

**Solution 1:**




shekharkrish13@gmail.com

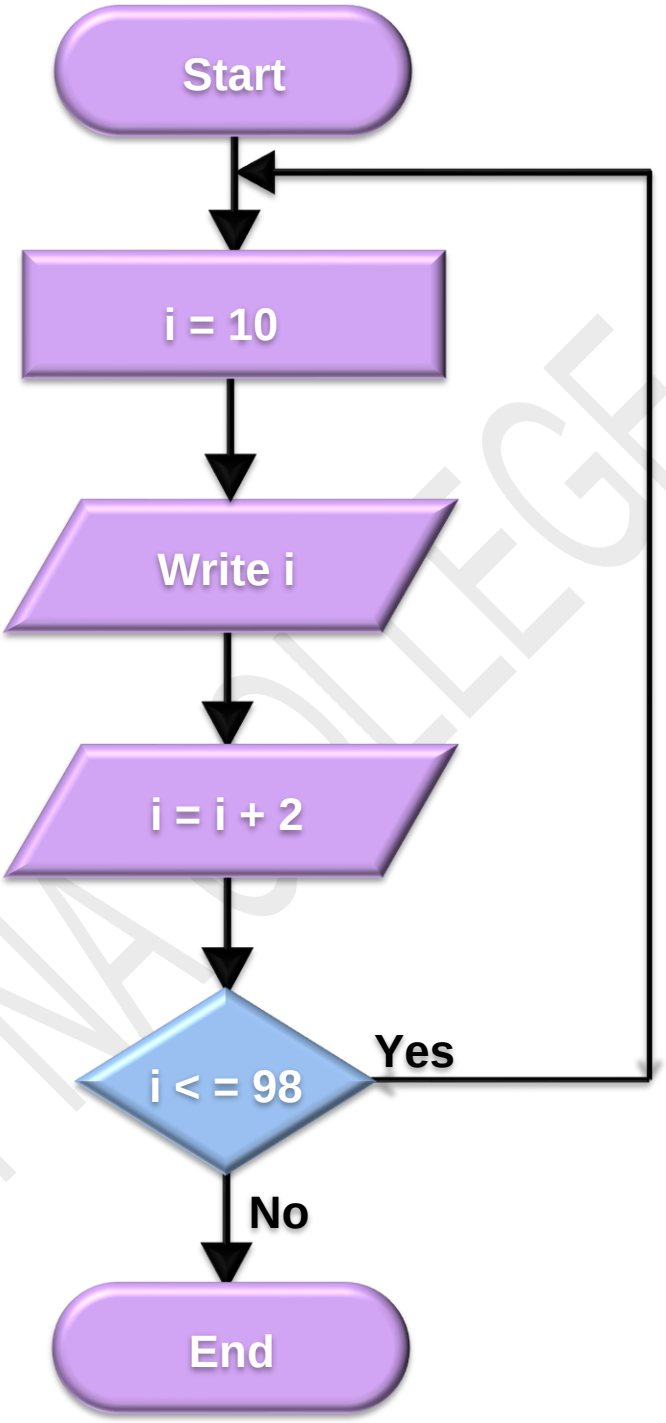
**Solution 2:**




APNA COLLEGE

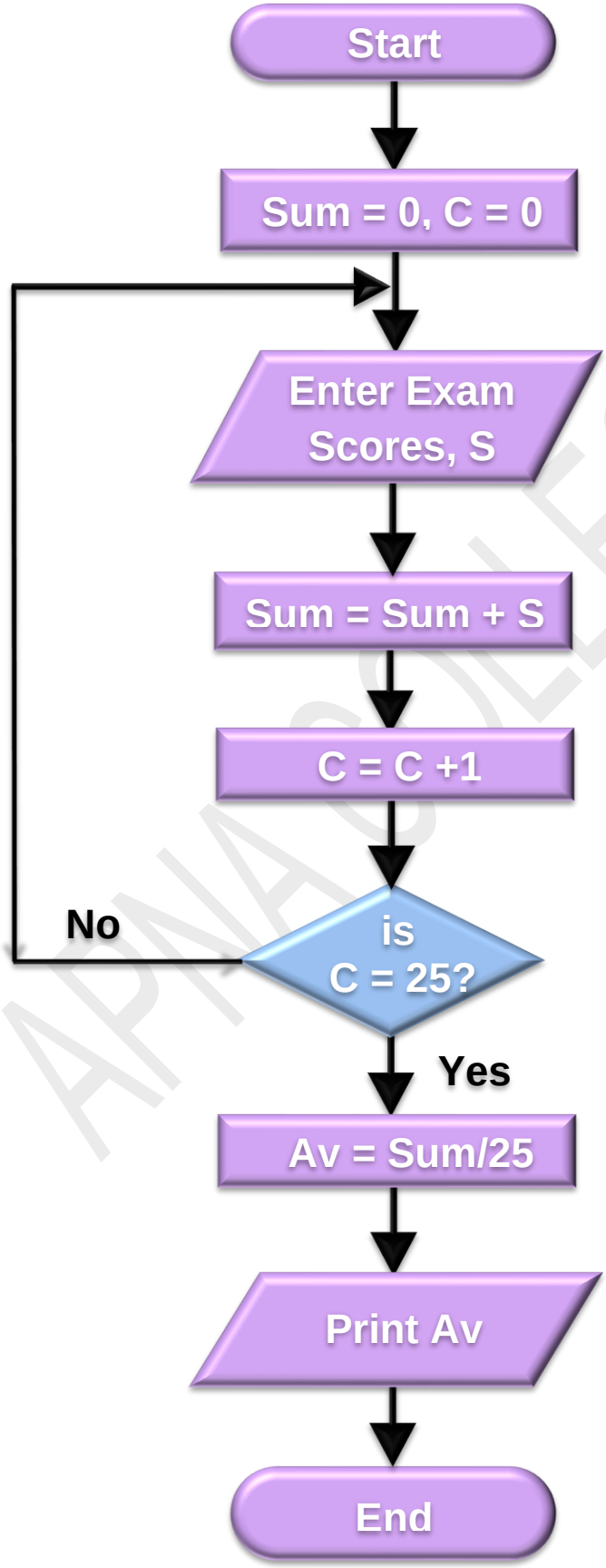
shekharkrish13@gmail.com

**Solution 3:**



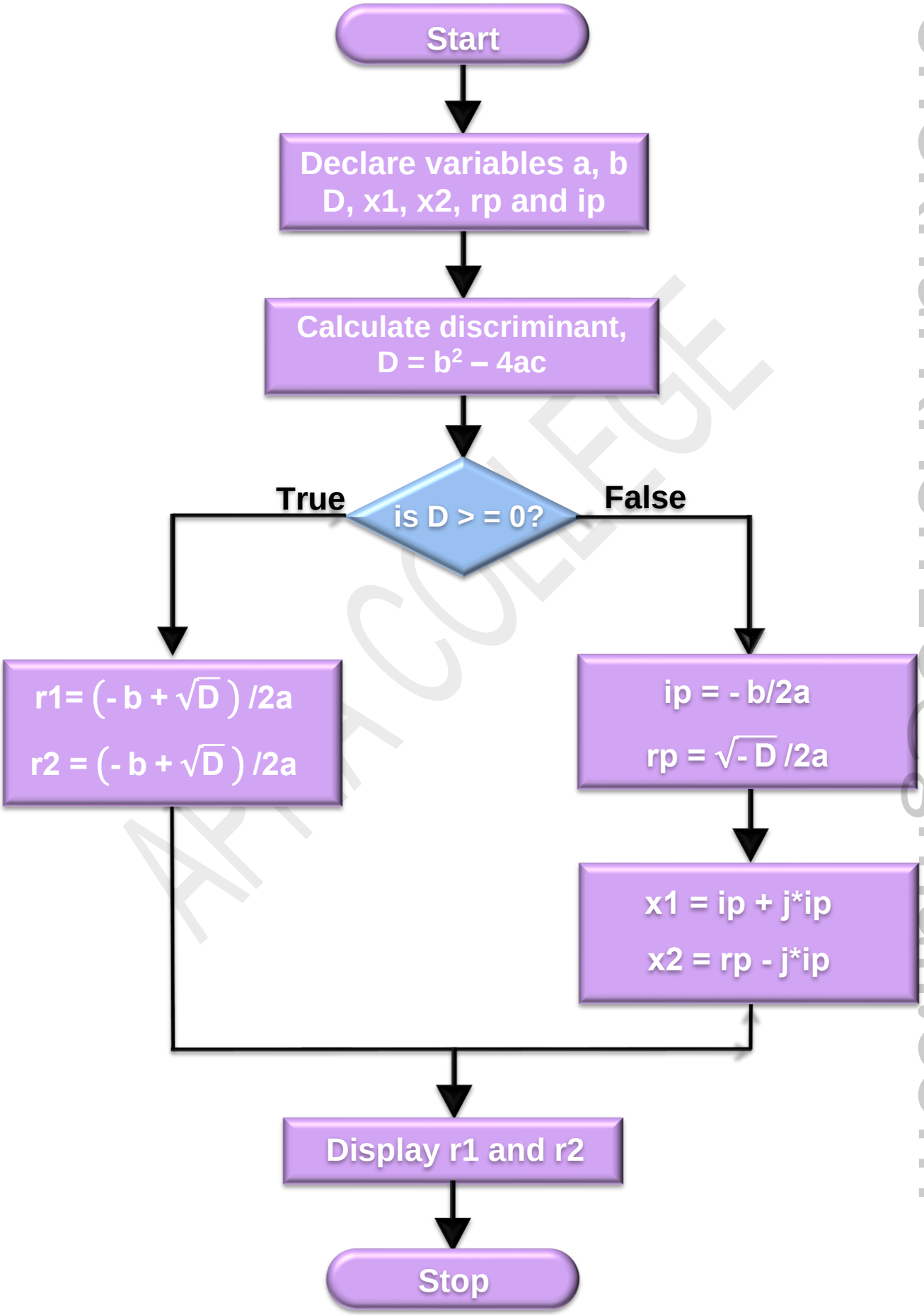

APNA COLLEGE
shekharkrish13@gmail.com

**Solution 4:**




APNA COLLEGE
shekharkrish13@gmail.com

**Solution 5:**




shekharkrish13@gmail.com

APNA COLLEGE

# VARIABLES & DATA TYPES SOLUTIONS

shekharkrish13@gmail.com

## Solution 1:

```
import java.util.*;

// Average of 3 numbers
public class Solution {
   public static void main(String args[]) {
       Scanner sc = new Scanner(System.in);
       int A = sc.nextInt();
       int B = sc.nextInt();
       int C = sc.nextInt();

       int average = (A + B + C) / 3;

       System.out.println("average is : " + average);
   }
}
```

COLLEGE

## Solution 2:

```
import java.util.*;

// Area of a Square
public class Solution {
   public static void main(String args[]) {
       Scanner sc = new Scanner(System.in);
       int side = sc.nextInt();

       int area = side * side;

       System.out.println("area of the square is : " + area);
   }
}
```

## Solution 3:

```java
import java.util.*;

// Bill of Items
public class Solution {
   public static void main(String args[]) {
       Scanner sc = new Scanner(System.in);
       float pencil = sc.nextFloat();
       float pen = sc.nextFloat();
       float eraser = sc.nextFloat();

       float total = pencil + pen + eraser;

       System.out.println("Bill is : " + total);



       //Add on - with 18% tax
       float newTotal = total + (0.18f * total);

       System.out.println("Bill with 18% tax : " + newTotal);
   }
}
```

## Solution 4:

In the mentioned code, the result variable will be of double type because of **type conversion**.

## Solution 5:

No, the statement will not give any error.

Names of variables are called identifiers in Java. Identifier rule says, identifiers can start with any alphabet or underscore ("_") or dollar ("$").

According to the rule the given variable name is a valid identifier.

shekharkrish13@gmail.com

# OPERATORS SOLUTIONS

Solution 1: Output is : 5 , 4

Solution 2: Output is : Java

Solution 3: Output is : 4, 0, 0

Solution 4: Output is : 278

Solution 5: Output is : 20, 20

shekharkrish13@gmail.com

# CONDITIONAL STATEMENTS SOLUTIONS

## Solution 1:

```
import java.util.*;

public class Solution {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int x = sc.nextInt();

       if (x > 0) {
           System.out.println("x is greater than 0");
       } else {
           System.out.println("x is less than or equal 0");
       }
   }
}
```

## Solution 2:

```
public class Solution{
   public static void main(String[] args) {
       double temp = 103.5;
       if (temp > 100) {
           System.out.println("You have a fever");
       } else {
           System.out.println("You don't have a fever");
       }
   }
}
```

## Solution 3:

```
import java.util.*;

public class Solution {
```

shekharkrish13@gmail.com

```
public static void main(String args[]) {
  Scanner sc = new Scanner(System.in);
  /* Input week number from user */
  System.out.println("Enter week number(1-7): ");
  int week = sc.nextInt();

  switch(week) {
      case 1:
          System.out.println("Monday");
          break;
      case 2:
          System.out.println("Tuesday");
          break;
      case 3:
          System.out.println("Wednesday");
          break;
      case 4:
          System.out.println("Thursday");
          break;
      case 5:
          System.out.println("Friday");
          break;
      case 6:
          System.out.println("Saturday");
          break;
      case 7:
          System.out.println("Sunday");
          break;
      default:
              System.out.println("Invalid input! Please enter week number between
1-7.");
  }
 }
}
```

## Solution 4:

Value of x = false & y = 63..

shekharkrish13@gmail.com

Solution 5:

```
import java.util.*;

public class Solution {
 public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.print("Input the year: ");
       int year = sc.nextInt();

       boolean x = (year % 4) == 0;
       boolean y = (year % 100) != 0;
       boolean z = ((year % 100 == 0) && (year % 400 == 0));

       if (x && (y || z)) {
           System.out.println(year + " is a leap year");
       } else {
           System.out.println(year + " is not a leap year");
       }
   }
}
```

shekharkrish13@gmail.com

# LOOPS SOLUTIONS

Solution 1: Hello is printed 2 times.

Solution 2:

```
import java.util.Scanner;
public class Solution {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);

       int number;
       int choice;
       int evenSum = 0;
       int oddSum = 0;

       do {
           System.out.print("Enter the number ");
           number = sc.nextInt();

           if( number % 2 == 0) {
               evenSum += number;
           } else {
               oddSum += number;
           }

              System.out.print("Do you want to continue? Press 1 for yes or 0 for
no");
           choice = sc.nextInt();

       } while(choice==1);

       System.out.println("Sum of even numbers: " + evenSum);
       System.out.println("Sum of odd numbers: " + oddSum);
   }
}
```

shekharkrish13@gmail.com

APNA COLLEGE

## Solution 3:

```
import java.util.Scanner;
public class Solution {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int num; // To hold number
       int fact = 1; // To hold factorial

       System.out.print("Enter any positive integer: ");
       num = sc.nextInt();

       for(int i=1; i<=num; i++) {
           fact *= i;
       }

       System.out.println("Factorial: "+ fact);
   }
}
```

## Solution 4:

```
import java.util.*;
class MultiplicationTable {
   public static void printMultiplicationTable(int number){
       Scanner sc = new Scanner(System.in);
       System.out.print("Enter number:");
       int n = sc.nextInt();
       for(int i=1; i<=10; i++) {
           System.out.println(n + " * " + i + " = " + n*i);
       }
   }
   public static void main(String s[]) {
       printMultiplicationTable(5);
   }
}
```

## Solution 5:

Scope of variable is referred to the part of the program where the variable can be used.

shekharkrish13@gmail.com

In this program variable `i` is declared in the `for` loop. So scope of variable `i` is limited to the `for` loop only that is between `{` and `}` of the `for` loop. There is a display statement after the for loop where variable `i` is used which means `i` is used out of scope. This leads to compilation errors.

The program given will not run and give an error instead. To correct the program, the variable i needs to be declared outside the for loop.

shekharkrish13@gmail.com

# FUNCTIONS SOLUTIONS

## Solution 1:

```
import java.util.Scanner;
public class Solution {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input the first number: ");
        double x = sc.nextDouble();
        System.out.print("Input the second number: ");
        double y = sc.nextDouble();
        System.out.print("Input the third number: ");
        double z = sc.nextDouble();
        System.out.print("The average value is " + average(x, y, z)+"\n" );
    }

 public static double average(double x, double y, double z) {
        return (x + y + z) / 3;
    }
}
```

## Solution 2:

```
import java.util.*;

public class Solution {
   public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;

        System.out.print("Enter an integer: ");
        num = sc.nextInt();

        if(isEven(num)) {
            System.out.println("Number is even");
        } else {
            System.out.println("Number is odd");
```

shekharkrish13@gmail.com

```
        }
    }

    public static boolean isEven(int number) {
        if(number % 2 == 0) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

Solution 3:

```
import java.util.Scanner;

public class Solution {
    public static void main(String args[]) {
        System.out.println("Please Enter a number : ");
        Scanner sc = new Scanner(System.in);
        int palindrome = sc.nextInt();

        if(isPalindrome(palindrome)) {
            System.out.println("Number : " + palindrome + " is a palindrome");
        } else {
            System.out.println("Number : " + palindrome + " is not a palindrome");
        }

    }
    public static boolean isPalindrome(int number) {
        int palindrome = number; // copied number into variable
        int reverse = 0;

        while (palindrome != 0) {
            int remainder = palindrome % 10;
            reverse = reverse * 10 + remainder;
            palindrome = palindrome / 10;
        }
```

shekharkrish13@gmail.com

```
        // if original and the reverse of number is equal means
        // number is palindrome in Java
        if (number == reverse) {
            return true;
        }
        return false;
    }
}
```

Solution 4: This is a DIY question & should be solved on your own.

Solution 5:

```
import java.util.Scanner;
public class Solution {
 public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     System.out.print("Input an integer: ");
     int digits = sc.nextInt();
     System.out.println("The sum is " + sumDigits(digits));
    }

public static int sumDigits(int n) {
        int sumOfDigits = 0;
        while(n > 0) {
            int lastDigit = n % 10;
            sumOfDigits += lastDigit;
            n /= 10;
        }

        return sumOfDigits;
    }
}
```

shekharkrish13@gmail.com

APNA COLLEGE

# Arrays - Solutions

**NOTE - Only functions will be given in the code. Please complete the entire code by writing themain function & class on your own.**

## Question 1 (DSA Sheet #4)

### Approach 1 - Brute Force (O(n^2))

```
public boolean containsDuplicate(int[] nums) {
   for(int i=0; i<nums.length-1 ; i++) {
       for(int j=i+1; j<nums.length ; j++ ) {
          if( nums[i] == nums[j] ) {
              return true ;
           }
       }
   }
   return false;
}
```

### Approach 2 - using Sets (O(n))

**/* You should have a basic knowledge about Java HashSets before following Approach 2. It willbe taught to you in later chapters. */**

```
public boolean containsDuplicate(int[] nums) {
   HashSet<Integer> set = new HashSet<>();
   for(int i=0; i<nums.length; i++) {
       if(set.contains(nums[i])) {
           return true;
       } else {
           set.add(nums[i]);
       }
   }

   return false;
}
```

shekharkrish13@gmail.com

**Question 2** **(DSA Sheet #6)**

**Approach - Based on Binary Search**

```
public int search(int[] nums, int target) {
   //min will have index of  minimum element of nums
   int min = minSearch(nums);
   //find in sorted left
   if(nums[min] <= target && target <= nums[nums.length-1]){
        return search(nums,min,nums.length-1,target);
   }
   //find in sorted right
   else{
       return search(nums,0,min,target);
   }
}

//binary search to find target in left to right boundary
public int search(int[] nums,int left,int right,int target){
   int l = left;
   int r = right;
   // System.out.println(left+" "+right);
   while(l <= r){
       int mid = l + (r - l)/2;
       if(nums[mid] == target){
           return mid;
       }
       else if(nums[mid] > target){
           r = mid-1;
       }
       else{
           l = mid+1;
       }
   }
   return -1;
}

//smallest element index
public int minSearch(int[] nums){
   int left = 0;
   int right = nums.length-1;
```

shekharkrish13@gmail.com

```
    while(left < right){
        int mid = left + (right - left)/2;
        if(mid > 0 && nums[mid-1] > nums[mid]){
            return mid;
        }
        else if(nums[left] <= nums[mid] && nums[mid] > nums[right]){
            left = mid+1;
        }
        else{
            right = mid-1;
        }
    }

    return left;
}
```

---

**Question 3** **(DSA Sheet #8)**

### Approach

```
public int maxProfit(int[] prices) {
    int buy = prices[0];
    int profit = 0;

    for (int i=1; i<prices.length; i++) {
        if (buy < prices[i]) {
            profit = Math.max(prices[i] - buy, profit);
        }
        else {
            buy = prices[i];
        }
    }

    return profit;
}
```

---

shekharkrish13@gmail.com

**Question 4** (DSA Sheet #11)

**/* This problem can be a little difficult for beginners to solve. Please analyze the solution if youare not able to come up with the code. */**

**Approach**

```
public int trap(int[] height) {
    int n = height.length;

    int res = 0, l = 0, r = n - 1;
    int rMax = height[r], lMax = height[l];

    while (l < r) {
        if (lMax < rMax) {
            l++;
            lMax = Math.max(lMax, height[l]);
            res += lMax - height[l];
        } else {
            r--;
            rMax = Math.max(rMax, height[r]);
            res += rMax - height[r];
        }
    }

    return res;
}
```

---

**Question 5** (DSA Sheet #16)

**/* This problem uses List to return the numbers & HashSets to store them. These are new data structures that we will study about in later chapters. */**

**Approach**

**Let us try to understand the problem statement. The first part of the problem statement is clear, we are asked to find out all the triplets in the given array whose sum is equal to zero. A triplet is nothing but a set of three numbers in the given array. For example, if nums=[1,2, 3,4] is the given array, [1,2,3] [2,3,4] [1,3,4] etc are its triplets.**

**What does the condition i != j, i != k, and j != k mean?**

shekharkrish13@gmail.com

APNA COLLEGE

It means that we are not allowed to reuse any number from the array within a triplet. Example, for the given array nums = [1,2,3,4], triplets [1,1,1] or [1,1,2] or [1,2,2] etc are not considered validtriplets.

The last condition that we need to consider is that we cannot have duplicate triplets in our final result. Example if [-2,-2,0,2] is the given array, we can only consider one of [-2,0,2] from indexes 0, 2, 3 and [-2,0,2] from indexes 1, 2, 3 in our final result.

BRUTE FORCE - The simple solution to this problem is to find every possible triplet from the given array, see if its sum is equal to zero and return the result (ensuring there are no duplicatetriplets in the result).

This algorithm involves the following steps:

1. Use three loops to generate all possible triplets for the given array, with each loop variable keeping track of 1 triplet element each.
2. Next we calculate the sum for each triplet generated in step 1.
3. If the sum is equal to 0 we need to check if it is a unique triplet (not already in our result set). We can ensure the triplets in our result set are unique by sorting the triplets and adding it to a hashmap (hashmap overwrites data if the same value iswritten to the same key multiple times thereby eliminating duplicates).
4. Once we have added all the triplets whose sum is equal to 0 into the hashmap, we iterate through the hashmap and add it to our result array.
5. Finally we return the result array.

APNA COLLEGE

```
public List<List<Integer>> threeSum(int nums[]) {
   List<List<Integer>> result = new ArrayList <List<Integer>> ();

   for(int i=0; i<nums.length; i++) {
       for(int j=i+1; j<nums.length; j++) {
           for(int k=j+1; k<nums.length; k++) {
               if(nums[i] + nums[j] + nums[k] == 0) {
```

shekharkrish13@gmail.com

```
                    List<Integer> triplet = new ArrayList < Integer > ();
                    triplet.add(nums[i]);
                    triplet.add(nums[j]);
                    triplet.add(nums[k]);
                    Collections.sort(triplet);
                    result.add(triplet);
                }
            }
        }
    }
    result = new ArrayList<List<Integer>> (new LinkedHashSet<List<Integer>> (result));
    return result;
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# BASIC SORTING SOLUTIONS

## Solution (a): Bubble Sort

```
public static void bubbleSortDescending(int arr[]) {
        for(int turn=0; turn<arr.length-1; turn++) {
            for(int j=0; j<arr.length-1-turn; j++) {
                if(arr[j] < arr[j+1]) {
                    //swap
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
```

## Solution (b): Selection Sort

```
public static void selectionSortDescending(int arr[]) {
        for(int turn=0; turn<arr.length; turn++) {
            int minPos = turn;
            for(int j=turn+1; j<arr.length; j++) {
                if(arr[minPos] < arr[j]) {
                    minPos = j;
                }
            }

            //swap
            int temp = arr[turn];
            arr[turn] = arr[minPos];
            arr[minPos] = temp;
        }
    }
```

## Solution (c) : Insertion Sort

```
public static void insertionSortDescending(int arr[]) {
```

shekharkrish13@gmail.com

```
        for(int i=1; i<arr.length; i++) {
            int curr = arr[i];
            int prev = i-1;
            //to find the index where curr is to be inserted
            while(prev >= 0 && arr[prev] < curr) {
                arr[prev+1] = arr[prev];
                prev--;
            }
            arr[prev+1] = curr;
        }
    }
```

Solution (d): Counting Sort

```
public static void countingSortDescending(int arr[]) {
        int largest = Integer.MIN_VALUE;
        for(int i=0; i<arr.length; i++) {
            largest = Math.max(largest, arr[i]);
        }

        int count[] = new int[largest+1];
        for(int i=0; i<arr.length; i++) {
            count[arr[i]]++;
        }
        int j = 0;
        for(int i=count.length-1; i>=0; i--) {
            while(count[i] > 0) {
                arr[j] = i;
                j++;
                count[i]--;
            }
        }
    }
```

shekharkrish13@gmail.com

# 2D ARRAYS SOLUTIONS

## Solution 1:

```
public class Solution {
   public static void main(String[] args) {
       int[][] array = { {4, 7, 8}, {8, 8, 7} };

       int countOf7 = 0;
       for(int i=0; i<array.length; i++) {
           for(int j=0; j<array[0].length; j++) {
               if(array[i][j] == 7) {
                   countOf7++;
               }
           }
       }

       System.out.println("count of 7 is : " + countOf7);
   }
}
```

## Solution 2:

```
public class Solution {
   public static void main(String[] args) {
       int[][] nums = { {1,4,9},{11,4,3},{2,2,3} };
       int sum = 0;

       //sum of 2nd row elements
       for(int j=0; j<nums[0].length; j++) {
           sum += nums[1][j];
       }

       System.out.println("sum is : " + sum);
   }
}
```

shekharkrish13@gmail.com

Solution 3 :

```java
public class Solution {
   public static void main(String[] args) {
       int row = 2, column = 3;
       int[][] matrix = { {2, 3, 7}, {5, 6, 7} };

       // Display original matrix
       printMatrix(matrix);

       // Transpose the matrix
       int[][] transpose = new int[column][row];
       for(int i = 0; i<row; i++) {
           for (int j = 0; j<column; j++) {
               transpose[j][i] = matrix[i][j];
           }
       }

       // print the transposed matrix
       printMatrix(transpose);
   }

   public static void printMatrix(int[][] matrix) {
       System.out.println("The matrix is: ");
       for(int i=0; i<matrix.length; i++) {
           for (int j=0; j<matrix[0].length; j++) {
               System.out.print(matrix[i][j] + "    ");
           }
           System.out.println();
       }
   }
}
```

shekharkrish13@gmail.com

# STRINGS SOLUTIONS

## Solution 1:

```
import java.util.*;

public class Solution {
   public static void main(String[] args) {
       String str = new Scanner(System.in).next();
       int count = 0;

       for(int i=0; i<str.length(); i++) {
           char ch = str.charAt(i);
           if(ch == 'a' || ch == 'e' ||ch == 'i' || ch == 'o' ||ch == 'u') {
               count++;
           }
       }
       System.out.println("count of vowels is :" + count);
   }
}
```

## Solution 2: Output will be :

false true

(If you need an explanation, please r-ewatch the video about how Strings work in memory?)

## Solution 3 : Output will be :

ApnaCoege

Following are some methods in Java which are used to replace characters:

| | |
|---|---|
| String | replace(char oldChar, char newChar)<br>Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. |
| String | replace(CharSequence target, CharSequence replacement)<br>Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. |
| String | replaceAll(String regex, String replacement)<br>Replaces each substring of this string that matches the given regular expression with the given replacement. |
| String | replaceFirst(String regex, String replacement)<br>Replaces the first substring of this string that matches the given regular expression with the given replacement. |

shekharkrish13@gmail.com

Solution 4:

```
import java.util.Arrays;
public class Solution {
  public static void main(String[] args) {
       String str1 = "earth";
       String str2 = "heart";

       //Convert Strings to lowercase. Why? so that we don't have to check
separately for lower & uppercase.
       str1 = str1.toLowerCase();
       str2 = str2.toLowerCase();

       // First check - if the lengths are the same
       if(str1.length() == str2.length())  {
            // convert strings into char array
            char[] str1charArray = str1.toCharArray();
            char[] str2charArray = str2.toCharArray();
            // sort the char array
            Arrays.sort(str1charArray);
            Arrays.sort(str2charArray);
            // if the sorted char arrays are same or identical then the strings are
anagram
            boolean result = Arrays.equals(str1charArray, str2charArray);
            if(result) {
               System.out.println(str1 + " and " + str2 + " are anagrams of each
other.");
             } else {
               System.out.println(str1 + " and " + str2 + " are not anagrams of
each other.");
             }
       } else {
           // case when lengths are not equal
          System.out.println(str1 + " and " + str2 + " are not anagrams of each
other.");
       }
   }
}
```

shekharkrish13@gmail.com

# BIT MANIPULATION SOLUTIONS

**Solution 1:** The value of x^x = 0.

Think about it, xor gives 0 when the bits are the same. If we compare the same number to itself, the bits will always be the same. So, the answer of x^x will always be 0.

**Solution 2:** The idea is to use XOR operators to swap two numbers by their property x ^ x = 0

```
public class Solution {
   public static void main(String[] args) {
       int x = 3, y = 4;
       System.out.println("Before swap: x = " + x + " and y = " + y);
       //swap using xor
       x = x ^ y;
       y = x ^ y;
       x = x ^ y;
       System.out.println("After swap: x = " + x + " and y = " + y);
   }
}
```

**Solution 3 :** The expression -~x will add 1 to an integer x. We know that to get negative of a number, invert its bits and add 1 to it (Remember negative numbers are stored in 2's complement form), i.e.,

-x = ~x + 1;

-~x = x + 1 (by replacing x by ~x)

```
public class Solution {
   public static void main(String[] args) {
       int x = 6;
       System.out.println(x + " + " + 1 + " is " + -~x);
       x = -4;
       System.out.println(x + " + " + 1 + " is " + -~x);
       x = 0;
       System.out.println(x + " + " + 1 + " is " + -~x);
   }
}
```

shekharkrish13@gmail.com

Solution 4 :

```
public class Solution {
   public static void main(String[] args) {
       // Convert uppercase character to lowercase
       for (char ch = 'A'; ch <= 'Z'; ch++) {
           System.out.println((char)(ch | ' '));
            // prints abcdefghijklmnopqrstuvwxyz
       }
   }
}
```

shekharkrish13@gmail.com

# OOPS SOLUTIONS

## Solution 1:

```
import java.util.*;
class Complex{
 int real;
 int imag;
 public Complex(int r, int i){
   real = r;
   imag = i;
 }

 public static Complex add(Complex a, Complex b){
   return new Complex((a.real+b.real),(a.imag+b.imag));
 }

 public static Complex diff(Complex a, Complex b){
   return new Complex((a.real-b.real),(a.imag-b.imag));
 }

 public static Complex product(Complex a, Complex b){
                                                        return                    new
Complex(((a.real*b.real)-(a.imag*b.imag)),((a.real*b.imag)+(a.imag*b.real)));
 }

 public void printComplex(){
   if(real == 0 && imag!=0){
     System.out.println(imag+"i");
   }
   else if(imag == 0 && real!=0){
     System.out.println(real);
   }
   else{
     System.out.println(real+"+"+imag+"i");
   }
 }

}
```

shekharkrish13@gmail.com

```
class Solution {
 public static void main(String[] args){
   Complex c = new Complex(4,5);
   Complex d = new Complex(9,4);

   Complex e = Complex.add(c,d);
   Complex f = Complex.diff(c,d);
   Complex g = Complex.product(c,d);
   e.printComplex();
   f.printComplex();
   g.printComplex();
 }
}
```

Solution 2: B. Driving electric car

The drive() method in the Car class does not override the version in the Automobile class since the method is not visible to the Car class.

The drive() method in the ElectricCar class is a valid override of the method in the Car class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect.

In the main() method, the object created is an ElectricCar, even if it is assigned to a Car reference. Due to polymorphism, the method from the ElectricCar will be invoked, making Option B the correct answer.

Solution 3 : B. public and protected both can be used.

You can provide only a less restrictive or same-access modifier when overriding a method.

Solution 4 : C. 13245

The class is loaded first, with the static initialization block called and 1 is outputted first. When the BlueCar is created in the main() method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next. Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making Option C the correct answer.

shekharkrish13@gmail.com

# RECURSION (Basics) SOLUTIONS

## Solution 1:

```
public class Solution {
   public static void allOccurences(int arr[], int key, int i) {
       if(i == arr.length) {
           return;
       }

       if(arr[i] == key) {
           System.out.print(i+" ");
       }
       allOccurences(arr, key, i+1);
   }
   public static void main(String[] args) {
       int arr[] = {3, 2, 4, 5, 6, 2, 7, 2, 2};
       int key = 2;
       allOccurences(arr, key, 0);
       System.out.println();
   }
}
```

## Solution 2:

```
public class Solution {
    static String digits[] = {"zero", "one", "two", "three", "four", "five", "six",
"seven", "eight", "nine"};

   public static void printDigits(int number) {
       if(number == 0) {
           return;
       }

       int lastDigit = number%10;
       printDigits(number/10);
       System.out.print(digits[lastDigit]+" ");
   }
   public static void main(String[] args) {
       printDigits(1234);
```

shekharkrish13@gmail.com

```
        System.out.println();
    }
}
```

Solution 3 :

```
public class Solution {
    public static int length(String str) {
        if(str.length() == 0) {
            return 0;
        }

        return length(str.substring(1)) + 1;
    }
    public static void main(String[] args) {
        String str = "abcde";
        System.out.println(length(str));
    }
}
```

Solution 4 :

```
public class Solution {
    public static int countSubstrs(String str, int i, int j, int n) {
        if (n == 1) {
            return 1;
        }
        if (n <= 0) {
            return 0;
        }

        int res = countSubstrs(str, i + 1, j, n - 1) +
                countSubstrs(str, i, j - 1, n - 1) -
                countSubstrs(str, i + 1, j - 1, n - 2);

        if (str.charAt(i) == str.charAt(j)) {
            res++;
        }
```

shekharkrish13@gmail.com

```
        return res;
    }
    public static void main(String[] args) {
        String str = "abcab";
        int n = str.length();
        System.out.print(countSubstrs(str, 0, n-1, n));
    }
}
```

Solution 5 :

```
public class Solution {
    public static void towerOfHanoi(int n, String src, String helper, String dest) {
        if(n == 1) {
        System.out.println("transfer disk " + n + " from " + src + " to " + dest);
        return;
        }

        //transfer top n-1 from src to helper using dest as 'helper'
        towerOfHanoi(n-1, src, dest, helper);
        //transfer nth from src to dest
        System.out.println("transfer disk " + n + " from " + src + " to " + helper);
        //transfer n-1 from helper to dest using src as 'helper'
        towerOfHanoi(n-1, helper, src, dest);
    }

    public static void main(String args[]) {
        int n = 4;
        towerOfHanoi(n, "A", "B", "C");
    }
}
```

The Solution for this particular question has also been discussed here : https://www.youtube.com/watch?v=u-HgzgYe8KA

At timestamp : 00:05

shekharkrish13@gmail.com

# Divide & Conquer SOLUTIONS

shekharkrish13@gmail.com

## Solution 1:

```
class Solution {
    //function to mergeSort 2 arrays
    public static String[] mergeSort(String[] arr, int lo, int hi) {
        if (lo == hi) {
            String[] A = { arr[lo] };
            return A;
        }
        int mid = lo + (hi - lo) / 2;
        String[] arr1 = mergeSort(arr, lo, mid);
        String[] arr2 = mergeSort(arr, mid + 1, hi);

        String[] arr3 = merge(arr1, arr2);
        return arr3;
    }

    static String[] merge(String[] arr1, String[] arr2) {
        int m = arr1.length;
        int n = arr2.length;
        String[] arr3 = new String[m + n];

        int idx = 0;

        int i = 0;
        int j = 0;

        while (i < m && j < n) {
            if (isAlphabeticallySmaller(arr1[i], arr2[j])) {
                arr3[idx] = arr1[i];
                i++;
                idx++;
            }
            else {
                arr3[idx] = arr2[j];
                j++;
                idx++;
            }
```

```
        }
        while (i < m) {
            arr3[idx] = arr1[i];
            i++;
            idx++;
        }
        while (j < n) {
            arr3[idx] = arr2[j];
            j++;
            idx++;
        }
        return arr3;
    }

    // Return true if str1 appears before str2 in alphabetical order
    static boolean isAlphabeticallySmaller(String str1, String str2) {
        if (str1.compareTo(str2) < 0) {
            return true;
        }
        return false;
    }
    public static void main(String[] args) {
        String[] arr = { "sun", "earth", "mars", "mercury" };
        String[] a = mergeSort(arr, 0, arr.length - 1);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
}
```

## Solution 2:

Approach 1 - Brute Force Approach

Idea : Count the number of times each number occurs in the array & find the largest count.

Time complexity - O(n^2)

```
class Solution {
    public static int majorityElement(int[] nums) {
        int majorityCount = nums.length/2;
```

shekharkrish13@gmail.com

```
        for (int i=0; i<nums.length; i++) {
            int count = 0;
            for (int j=0; j<nums.length; j++) {
                if (nums[j] == nums[i]) {
                    count += 1;
                }
            }
            if (count > majorityCount) {
                return nums[i];
            }
        }
        return -1;
    }
    public static void main(String args[]) {
        int nums[] = {2,2,1,1,1,2,2};
        System.out.println(majorityElement(nums));
    }
}
```

Approach 2 - Divide & Conquer

Idea : If we know the majority element in the left and right halves of an array, we can determine which is the global majority element in linear time.

Here, we apply a classical divide & conquer approach that recurses on the left and right halves of an array until an answer can be trivially achieved for a length-1 array. Note that because actually passing copies of subarrays costs time and space, we instead pass lo and hi indices that describe the relevant slice of the overall array. In this case, the majority element for a length-1 slice is trivially its only element, so the recursion stops there. If the current slice is longer than length-1, we must combine the answers for the slice's left and right halves. If they agree on the majority element, then the majority element for the overall slice is obviously the same[1]. If they disagree, only one of them can be "right", so we need to count the occurrences of the left and right majority elements to determine which subslice's answer is globally correct. The overall answer for the array is thus the majority element between indices 0 and n.

Time complexity - O(n*logn)

```
class Solution {
    private static int countInRange(int[] nums, int num, int lo, int hi) {
        int count = 0;
        for (int i = lo; i <= hi; i++) {
            if (nums[i] == num) {
```

shekharkrish13@gmail.com

```
                count++;
            }
        }
        return count;
    }

    private static int majorityElementRec(int[] nums, int lo, int hi) {
        // base case; the only element in an array of size 1 is the majority
        // element.
        if (lo == hi) {
            return nums[lo];
        }

        // recurse on left and right halves of this slice.
        int mid = (hi-lo)/2 + lo;
        int left = majorityElementRec(nums, lo, mid);
        int right = majorityElementRec(nums, mid+1, hi);

        // if the two halves agree on the majority element, return it.
        if (left == right) {
            return left;
        }

        // otherwise, count each element and return the "winner".
        int leftCount = countInRange(nums, left, lo, hi);
        int rightCount = countInRange(nums, right, lo, hi);

        return leftCount > rightCount ? left : right;
    }

    public static int majorityElement(int[] nums) {
        return majorityElementRec(nums, 0, nums.length-1);
    }

    public static void main(String args[]) {
        int nums[] = {2,2,1,1,1,2,2};
        System.out.println(majorityElement(nums));
    }
}
```

shekharkrish13@gmail.com

(You can also find this problem at - https://leetcode.com/problems/majority-element/)

## Solution 3 :

Approach 1 - Brute Force Approach

Idea : Traverse through the array, and for every index, find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all indexes in the array and print the sum.

- Traverse through the array from start to end
- For every element, find the count of elements smaller than the current number up to that index using another loop.
- Sum up the count of inversion for every index.
- Print the count of inversions.

Time complexity - O(n^2)

```
class Solution {
   public static int getInvCount(int arr[]) {
       int n = arr.length;
       int invCount = 0;
       for (int i = 0; i < n - 1; i++) {
           for (int j = i + 1; j < n; j++) {
               if (arr[i] > arr[j]) {
                   invCount++;
               }
           }
       }
       return invCount;
   }
   public static void main(String[] args) {
       int arr[] = {1, 20, 6, 4, 5};
       System.out.println("Inversion Count = "+ getInvCount(arr));
   }
}
```

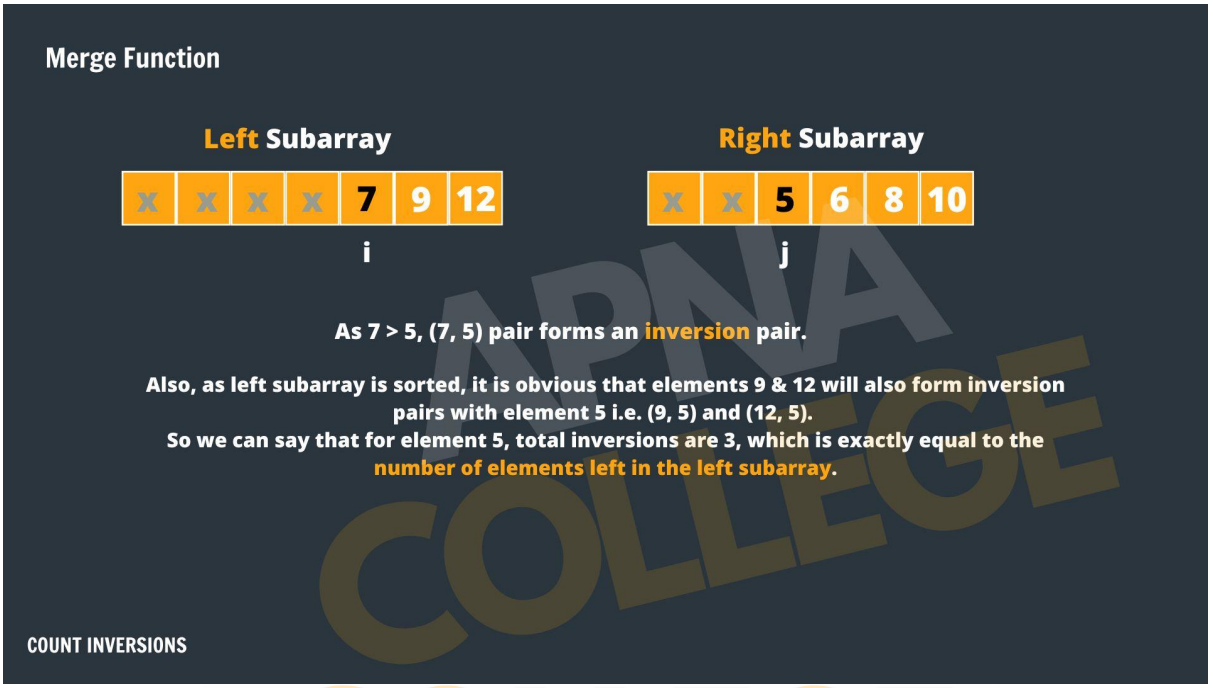
Approach 2 - Modified Merge Sort

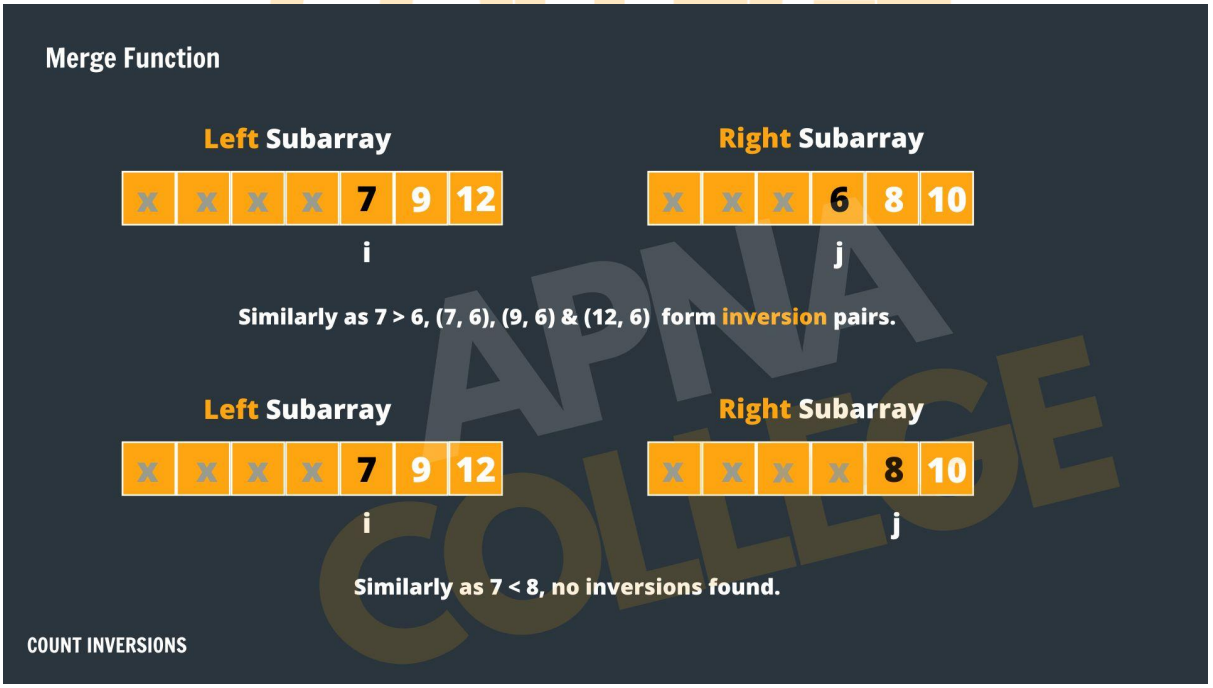
Idea : Suppose the number of inversions in the left half and right half of the array (let be inv1 and inv2); what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is – the inversions that need to be counted during the merge step. Therefore, to get the total number of inversions that need to be added are the number of inversions in the left subarray, right subarray, and merge().

shekharkrish13@gmail.com

Basically, for each array element, count all elements more than it to its left and add the count to the output. This whole magic happens inside the merge function of merge sort.

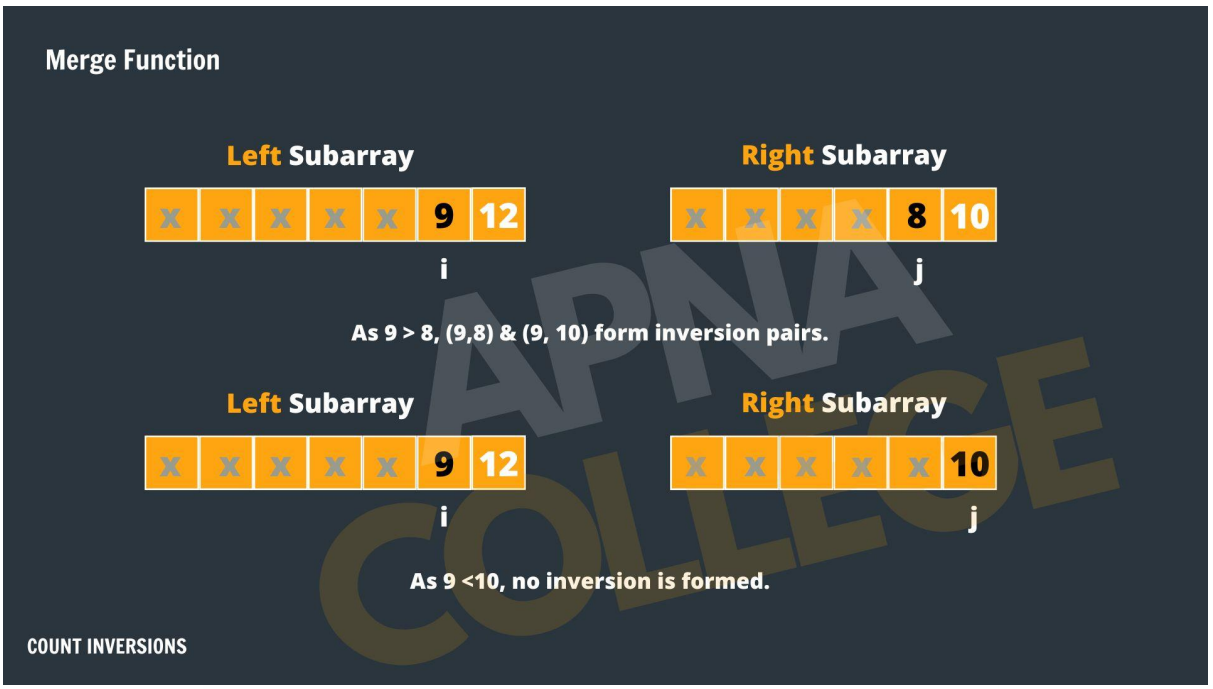
Let's consider two subarrays involved in the merge process:

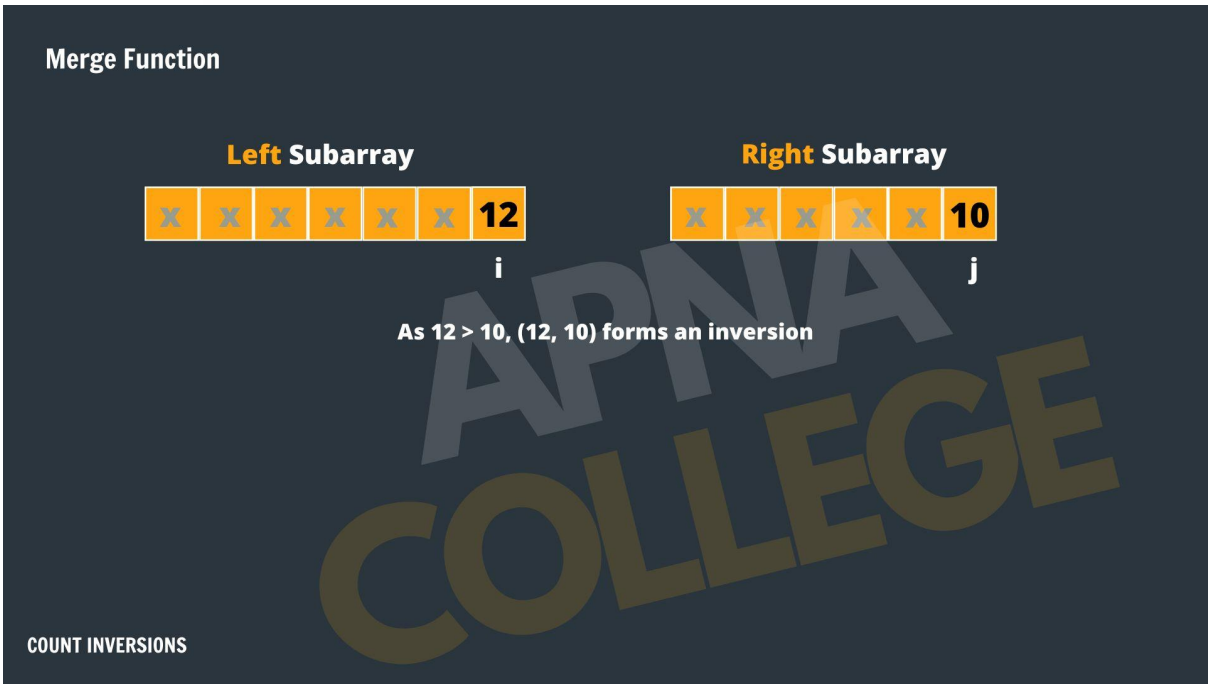



shekharkrish13@gmail.com

APNA COLLEGE




Algorithm:

- Split the given input array into two halves, left and right similar to merge sort recursively.
- Count the number of inversions in the left half and right half along with the inversions found during the merging of the two halves.
- Stop the recursion, only when 1 element is left in both halves.
- To count the number of inversions, we will use a two pointers approach. Let us consider two pointers i and j, one pointing to the left half and the other pointing towards the right half.

shekharkrish13@gmail.com

- While iterating through both the halves, if the current element A[i] is less than A[j], add it to the sorted list, else increment the count by mid – i.

Time complexity - O(n* logn)

```
public class Solution {
   public static int merge(int arr[], int left, int mid, int right) {
       int i = left, j = mid, k = 0;
       int invCount = 0;
       int temp[] = new int[(right - left + 1)];

       while ((i < mid) && (j <= right)) {
           if (arr[i] <= arr[j]) {
               temp[k] = arr[i];
               k++;
               i++;
           }
           else {
               temp[k] = arr[j];
               invCount += (mid - i);
               k++;
               j++;
           }
       }

       while (i < mid) {
           temp[k] = arr[i];
           k++;
           i++;
       }

       while (j <= right) {
           temp[k] = arr[j];
           k++;
           j++;
       }

       for (i = left, k = 0; i <= right; i++, k++) {
           arr[i] = temp[k];
       }
```

shekharkrish13@gmail.com

```
        return invCount;
    }

    private static int mergeSort(int arr[], int left, int right) {
        int invCount = 0;

        if (right > left) {
            int mid = (right + left) / 2;

            invCount = mergeSort(arr, left, mid);
            invCount += mergeSort(arr, mid + 1, right);
            invCount += merge(arr, left, mid + 1, right);
        }
        return invCount;
    }

    public static int getInversions(int arr[]) {
        int n = arr.length;
        return mergeSort(arr, 0, n - 1);
    }

    public static void main(String args[]) {
        int arr[] = {1, 20, 6, 4, 5};
        System.out.println("Inversion Count = " + getInversions(arr));
    }
}
```

shekharkrish13@gmail.com

# TIME COMPLEXITY SOLUTIONS

Solution :

a. Option A -> Time complexity = O(n*logn)

   In the loop, j keeps doubling till it is less than or equal to n. Several times, we can double a number till it is less than n would be log(n).

   Let's take the examples here.

   for n = 16, j = 2, 4, 8, 16

   for n = 32, j = 2, 4, 8, 16, 32

   So, j would run for O(log n) steps.

   i runs for n/2 steps.

   So, total steps = O(n/ 2 * log (n)) = O(n*logn)

b. Option C -> Time complexity = O(logkn)

   Because loops for the kn-1 times, so after taking log it becomes logkn.

c. Option B. -> false

   The Big-O notation provides an asymptotic comparison in the running time of algorithms. For n < n0, algorithm A might run faster than algorithm B, for instance.

d. Time complexity - O(√n)

   Space complexity - O(1)

e. Time complexity - O(n^2)

   Space complexity - O(1)

APNA COLLEGE

shekharkrish13@gmail.com

# BACKTRACKING SOLUTIONS

## Solution 1:

Algorithm -

1. Create a solution matrix, initially filled with 0's.
2. Create a recursive function, which takes the initial matrix, output matrix and position of rat (i, j).
3. if the position is out of the matrix or the position is not valid then return.
4. Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
5. Recursively call for position (i+1, j) and (i, j+1).
6. Unmark position (i, j), i.e output[i][j] = 0.

```
public class Solution {
    public static void printSolution(int sol[][]) {
        for (int i = 0; i<sol.length; i++) {
            for (int j = 0; j<sol.length; j++) {
                System.out.print(" " + sol[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static boolean isSafe(int maze[][], int x, int y) {
        // if (x, y outside maze) return false
        return (x >= 0 && x < maze.length
            && y >= 0 && y < maze.length && maze[x][y] == 1);
    }

    public static boolean solveMaze(int maze[][]) {
        int N = maze.length;
        int sol[][] = new int[N][N];
        if (solveMazeUtil(maze, 0, 0, sol) == false) {
            System.out.print("Solution doesn't exist");
            return false;
        }
        printSolution(sol);
        return true;
```

shekharkrish13@gmail.com

```java
    }

    public static boolean solveMazeUtil(int maze[][], int x, int y, int sol[][]) {
        if (x == maze.length - 1 && y == maze.length - 1 && maze[x][y] == 1) {
            sol[x][y] = 1;
            return true;
        }

        // Check if maze[x][y] is valid
        if (isSafe(maze, x, y) == true) {
            if (sol[x][y] == 1)
                return false;
            sol[x][y] = 1;
            if (solveMazeUtil(maze, x + 1, y, sol))
                return true;
            if (solveMazeUtil(maze, x, y + 1, sol))
                return true;
            sol[x][y] = 0;
            return false;
        }

        return false;
    }

    public static void main(String args[]){
        int maze[][] = { { 1, 0, 0, 0 },
                         { 1, 1, 0, 1 },
                         { 0, 1, 0, 0 },
                         { 1, 1, 1, 1 } };

        solveMaze(maze);
    }
}
```

shekharkrish13@gmail.com

Solution 2:

```
public class Solution {
   final static char[][] L = {{},{},{'a','b','c'},{'d','e','f'},{'g','h','i'},
                        {'j','k','l'},{'m','n','o'},{'p','q','r','s'},
                        {'t','u','v'},{'w','x','y','z'}};

   public static void letterCombinations(String D) {
       int len = D.length();
       if (len == 0) {
           System.out.println("");
           return;
       }
       bfs(0, len, new StringBuilder(), D);
   }

   public static void bfs(int pos, int len, StringBuilder sb, String D) {
       if (pos == len){
           System.out.println(sb.toString());
       }
       else {
           char[] letters = L[Character.getNumericValue(D.charAt(pos))];
           for (int i = 0; i < letters.length; i++)
               bfs(pos+1, len, new StringBuilder(sb).append(letters[i]), D);
       }
   }
   public static void main(String args[]){
       letterCombinations("2");
   }
}
```

Solution 3 :

```
public class Solution {
   static int N = 8;

   public static boolean isSafe(int x, int y, int sol[][]){
```

shekharkrish13@gmail.com

```
        return (x >= 0 && x < N && y >= 0 && y < N
                && sol[x][y] == -1);
    }

    public static void printSolution(int sol[][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++)
                System.out.print(sol[x][y] + " ");
            System.out.println();
        }
    }

    public static boolean solveKT() {
        int sol[][] = new int[8][8];
        for (int x = 0; x < N; x++)
            for (int y = 0; y < N; y++)
                sol[x][y] = -1;

        int xMove[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
        int yMove[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

        //As the Knight starts from cell(0,0)
        sol[0][0] = 0;

        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
            System.out.println("Solution does not exist");
            return false;
        }
        else
            printSolution(sol);

        return true;
    }

    public static boolean solveKTUtil(int x, int y, int movei, int sol[][],
                                      int xMove[],int yMove[]) {
        int k, next_x, next_y;
        if (movei == N * N)
            return true;
```

shekharkrish13@gmail.com

```
        for (k = 0; k < 8; k++) {
            next_x = x + xMove[k];
            next_y = y + yMove[k];
            if (isSafe(next_x, next_y, sol)) {
                sol[next_x][next_y] = movei;
                if (solveKTUtil(next_x, next_y, movei + 1,
                                sol, xMove, yMove))
                    return true;
                else
                    sol[next_x][next_y]
                        = -1; // backtracking
            }
        }
        return false;
    }

    public static void main(String args[]){
        solveKT();
    }
}
```

shekharkrish13@gmail.com

# ARRAYLIST SOLUTIONS

Solution 1:

```
public boolean isMonotonic(ArrayList<Integer> A) {
        boolean inc = true;
        boolean dec = true;
        for (int i=0; i<A.size()-1; i++) {
            if (A.get(i) > A.get(i+1))
                inc = false;
            if (A.get(i) < A.get(i+1))
                dec = false;
        }

        return inc || dec;
    }
```

Solution 2:

```
public ArrayList<Integer> findLonely(ArrayList<Integer> nums) {
        Collections.sort(nums);
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=1; i < nums.size()-1; i++) {
              if (nums.get(i-1) + 1 < nums.get(i) && nums.get(i) + 1 < nums.get(i+1))
{
                list.add(nums.get(i));
            }
        }
        if (nums.size() == 1) {
            list.add(nums.get(0));
        }
        if (nums.size() > 1) {
            if (nums.get(0) + 1 < nums.get(1)) {
                list.add(nums.get(0));
            }
            if (nums.get(nums.size()-2) + 1 < nums.get(nums.size()-1)) {
```

shekharkrish13@gmail.com

```
                list.add(nums.get(nums.size()-1));
            }
        }
        return list;
    }
```

Solution 3 :

```
public int mostFrequent(ArrayList<Integer> nums, int key) {
        int[]result = new int[1000];

        for(int i=0; i<nums.size()-1; i++){
            if(nums.get(i) == key){
                result[nums.get(i+1)-1]++;
            }
        }

        int max = Integer.MIN_VALUE;
        int ans = 0;

        for(int i=0; i<1000; i++){
            if(result[i] > max){
                max = result[i];
                ans = i+1;
            }
        }

        return ans;
    }
```

Solution 4 :

Approach 1 (Iterative)

We can see that if we separate odd and even numbers then there is no possibility that even numbers will violate the rule with odd numbers and vice versa. Now we have to arrange even and odd numbers in such a way that they do not violate rules with themselves. For doing so

shekharkrish13@gmail.com

first let's say we have a beautiful arraylist of size n and we want to make n+1 size of arraylist so what we do is first put all odd numbers that lie within 1 to n+1 and then even(WE can do even then odd also) . Beautiful arraylist has the property that if we multiply any number with arraylist then it still remains beautiful or if we add or subtract any number from arraylist then it still remains beautiful. For only obtaining even number from n size arraylist we do 2* num and for obtaining odd size arraylist we do 2 * n-1 .

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> ans = new ArrayList<>();
        ans.add(1);

        for(int i=2;i<=n;i++){
            ArrayList<Integer>temp=new ArrayList<>();
            for(Integer e:ans){
                if(2*e<=n)temp.add(e*2);
            }
            for(Integer e:ans){
                if(2*e-1<=n)temp.add(e*2-1);
            }

            ans=temp;
        }

        return ans;
    }
```

Approach 2 (Divide & Conquer)

Let's start from a simple 3 numbers case: (1, 2, 3) -> the only thing we need to do is move 2 out of 1 and 3 -> (1, 3, 2).

Then what if the case is (1, 5, 9 ) which has increment = 4? It's the same thing -> move 3 out of 1 and 5 -> (1, 9, 5).

Now, what if the case is (1, 3, 5, 7, 9) ? With the odd + even or divide + conquer idea in mind, we can simply divide it to (1, 5, 9) and (3, 7). Since no change is needed for the 2 numbers case, after following the above steps, we can conquer them to (1, 9, 5, 3, 7).

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> res = new ArrayList<>();
        divideConque(1, 1, res, n);
        return res;
    }
```

shekharkrish13@gmail.com

```
    private void divideConque(int start, int increment, ArrayList<Integer> res, int
n) {
        if (start + increment > n) {
            res.add(start);
            return;
        }
        divideConque(start, 2 * increment, res, n);
        divideConque(start + increment, 2 * increment, res, n);
    }
```

shekharkrish13@gmail.com

# Linked List Solutions

## Solution 1:

Time Complexity  : o(m*n)
Space Complexity: o(1)

```
class Solution {

        static class Node {
                int data;
                Node next;
                Node(int d){
                        data = d;
                        next = null;
                }
        }

        public Node getIntersectionNode(Node head1, Node head2)
        {
                while (head2 != null) {
                        Node temp = head1;
                        while (temp != null) {
                                if (temp == head2) {
                                        return head2;
                                }
                                temp = temp.next;
                        }
                        head2 = head2.next;
                }
                return null;
        }

        public static void main(String[] args){
                Solution list = new Solution();

                Node head1, head2;
                head1 = new Node(10);
                head2 = new Node(3);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
            Node newNode = new Node(6);
            head2.next = newNode;

            newNode = new Node(9);
            head2.next.next = newNode;

            newNode = new Node(15);
            head1.next = newNode;
            head2.next.next.next = newNode;

            newNode = new Node(30);
            head1.next.next = newNode;

            head1.next.next.next = null;

            Node intersectionPoint
                    = list.getIntersectionNode(head1, head2);

            if (intersectionPoint == null) {
                    System.out.print(" No Intersection Point \n");
            }
            else {
                    System.out.print("Intersection Point: "
                                                    + intersectionPoint.data);
            }
        }
}
```

## Solution 2:

Time Complexity  : o(n)
Space Complexity: o(1)

```
import java.util.*;
class Solution{

static class Node{
        int data;
        Node next;
```

shekharkrish13@gmail.com

```
};

static Node push( Node head_ref, int new_data){
        Node new_node = new Node();
        new_node.data = new_data;
        new_node.next = (head_ref);
        (head_ref) = new_node;
        return head_ref;
}

static void printList( Node head){
        Node temp = head;
        while (temp != null){
                System.out.printf("%d ", temp.data);
                temp = temp.next;
        }
        System.out.printf("\n");
}

static void skipMdeleteN( Node head, int M, int N){
        Node curr = head, t;
        int count;

        while (curr!=null){
                for (count = 1; count < M && curr != null; count++)
                        curr = curr.next;

                if (curr == null)
                        return;
                t = curr.next;
                for (count = 1; count <= N && t != null; count++){
                        Node temp = t;
                        t = t.next;
                }

                curr.next = t;
                curr = t;
        }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

```
public static void main(String args[]){
        Node head = null;
        int M=2, N=3;
        head=push(head, 10);
        head=push(head, 9);
        head=push(head, 8);
        head=push(head, 7);
        head=push(head, 6);
        head=push(head, 5);
        head=push(head, 4);
        head=push(head, 3);
        head=push(head, 2);
        head=push(head, 1);

        System.out.printf("M = %d, N = %d \n" +
                                        "Linked list we have is :\n", M, N);
        printList(head);

        skipMdeleteN(head, M, N);

        System.out.printf("\nLinked list on deletion is :\n");
        printList(head);
}
}
```

## Solution 3 :

Time Complexity  : o(n)
Space Complexity: o(1)

```
class Node {
        int data;
        Node next;
        Node(int d){
                data = d;
                next = null;
        }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
}

class Solution {
        Node head;

        public void swapNodes(int x, int y){
                if (x == y)
                        return;

                Node prevX = null, currX = head;
                while (currX != null && currX.data != x) {
                        prevX = currX;
                        currX = currX.next;
                }

                Node prevY = null, currY = head;
                while (currY != null && currY.data != y) {
                        prevY = currY;
                        currY = currY.next;
                }

                if (currX == null || currY == null)
                        return;

                if (prevX != null)
                        prevX.next = currY;
                else
                        head = currY;

                if (prevY != null)
                        prevY.next = currX;
                else
                        head = currX;

                Node temp = currX.next;
                currX.next = currY.next;
                currY.next = temp;
        }

        public void push(int new_data){
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        Node new_Node = new Node(new_data);
        new_Node.next = head;
        head = new_Node;
    }

    public void printList(){
        Node tNode = head;
        while (tNode != null) {
            System.out.print(tNode.data + " ");
            tNode = tNode.next;
        }
    }

    public static void main(String[] args){
        Solution llist = new Solution();

        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.print(
            "\n Linked list before ");
        llist.printList();

        llist.swapNodes(4, 3);

        System.out.print(
            "\n Linked list after ");
        llist.printList();
    }
}
```

## Solution 4 :

Time Complexity  : o(n)
Space Complexity: o(1)

shekharkrish13@gmail.com

```
class Solution{
        Node head;
        class Node{
                int data;
                Node next;
                Node(int d){
                        data = d;
                        next = null;
                }
        }

        void segregateEvenOdd(){
                Node end = head;
                Node prev = null;
                Node curr = head;

                while (end.next != null)
                        end = end.next;

                Node new_end = end;

                while (curr.data %2 !=0 && curr != end){
                        new_end.next = curr;
                        curr = curr.next;
                        new_end.next.next = null;
                        new_end = new_end.next;
                }

                if (curr.data %2 ==0){
                        head = curr;
                        while (curr != end){
                                if (curr.data % 2 == 0){
                                        prev = curr;
                                        curr = curr.next;
                                }
                                else{
                                        prev.next = curr.next;
                                        curr.next = null;
                                        new_end.next = curr;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                    new_end = curr;
                    curr = prev.next;
                }
            }
        }
        else prev = curr;
        if (new_end != end && end.data %2 != 0){
            prev.next = end.next;
            end.next = null;
            new_end.next = end;
        }
}
void push(int new_data){
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
}

void printList(){
        Node temp = head;
        while(temp != null){
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
}

public static void main(String args[]){
        Solution llist = new Solution();
        llist.push(11);
        llist.push(10);
        llist.push(8);
        llist.push(6);
        llist.push(4);
        llist.push(2);
        llist.push(0);
        System.out.println("Linked List");
        llist.printList();

        llist.segregateEvenOdd();
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        System.out.println("updated Linked List");
        llist.printList();
    }
}
```

## Solution 5 :

Time Complexity  : o(n*logk)
Space Complexity: o(n)

```
public class Solution {
    public static Node SortedMerge(Node a, Node b){
        Node result = null;
        if (a == null)
            return b;
        else if (b == null)
            return a;
        if (a.data <= b.data) {
            result = a;
            result.next = SortedMerge(a.next, b);
        }
        else {
            result = b;
            result.next = SortedMerge(a, b.next);
        }

        return result;
    }

    public static Node mergeKLists(Node arr[], int last)
    {
        while (last != 0) {
            int i = 0, j = last;
            while (i < j) {
                arr[i] = SortedMerge(arr[i], arr[j]);
                i++;
                j--;
                if (i >= j)
                    last = j;
            }
```

shekharkrish13@gmail.com

```
            }

            return arr[0];
        }
        public static void printList(Node node){
            while (node != null) {
                System.out.print(node.data + " ");
                node = node.next;
            }
        }

        public static void main(String args[]){
            int k = 3;
            int n = 4;
            Node arr[] = new Node[k];

            arr[0] = new Node(1);
            arr[0].next = new Node(3);
            arr[0].next.next = new Node(5);
            arr[0].next.next.next = new Node(7);

            arr[1] = new Node(2);
            arr[1].next = new Node(4);
            arr[1].next.next = new Node(6);
            arr[1].next.next.next = new Node(8);

            arr[2] = new Node(0);
            arr[2].next = new Node(9);
            arr[2].next.next = new Node(10);
            arr[2].next.next.next = new Node(11);

            Node head = mergeKLists(arr, k - 1);
            printList(head);
        }
}

class Node {
        int data;
        Node next;
        Node(int data){
```

shekharkrish13@gmail.com

```
        this.data = data;
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

APNA COLLEGE

# Stack Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

class Solution {
        public static void main(String args[]){
                Node one = new Node(1);
                Node two = new Node(2);
                Node three = new Node(3);
                Node four = new Node(4);
                Node five = new Node(3);
                Node six = new Node(2);
                Node seven = new Node(1);
                one.ptr = two;
                two.ptr = three;
                three.ptr = four;
                four.ptr = five;
                five.ptr = six;
                six.ptr = seven;
                boolean condition = isPalindrome(one);
                System.out.println("Palindrome :" + condition);
        }
        static boolean isPalindrome(Node head){

                Node slow = head;
                boolean ispalin = true;
                Stack<Integer> stack = new Stack<Integer>();

                while (slow != null) {
                        stack.push(slow.data);
                        slow = slow.ptr;
                }

                while (head != null) {
```

APNA COLLEGE

shekharkrish13@gmail.com

```
                int i = stack.pop();
                if (head.data == i) {
                        ispalin = true;
                }
                else {
                        ispalin = false;
                        break;
                }
                head = head.ptr;
        }
        return ispalin;
    }
}

class Node {
    int data;
    Node ptr;
    Node(int d){
        ptr = null;
        data = d;
    }
}
```

## Solution 2 :

Time Complexity  : o(n)
Space Complexity: o(1)

```
import java.io.*;
import java.util.*;

class Solution{
    public static void main(String []args){
        String str = new String("/a/./b/../../c/");
        String res = simplify(str);
        System.out.println(res);
    }
```

shekharkrish13@gmail.com

```
static String simplify(String A){
        Stack<String> st = new Stack<String>();
        String res = "";
        res += "/";
        int len_A = A.length();

        for (int i = 0; i < len_A; i++){
                String dir = "";
                while (i < len_A && A.charAt(i) == '/')
                        i++;

                while (i < len_A && A.charAt(i) != '/'){
                        dir += A.charAt(i);
                        i++;
                }

                if (dir.equals("..") == true){
                        if (!st.empty())
                                st.pop();
                }

                else if (dir.equals(".") == true)
                        continue;

                else if (dir.length() != 0)
                        st.push(dir);
        }

        Stack<String> st1 = new Stack<String>();
        while (!st.empty()){

                st1.push(st.pop());
        }

        while (!st1.empty()){
                if (st1.size() != 1)
                        res += (st1.pop() + "/");
                else
                        res += st1.pop();
        }
```

shekharkrish13@gmail.com

```
        return res;
    }

}
```

## Solution 3 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.util.Stack;

class Solution{
    static String decode(String str){
        Stack<Integer> integerstack = new Stack<>();
        Stack<Character> stringstack = new Stack<>();
        String temp = "", result = "";
        for (int i = 0; i < str.length(); i++){
            int count = 0;
            if (Character.isDigit(str.charAt(i))){
                while (Character.isDigit(str.charAt(i))){
                    count = count * 10 + str.charAt(i) - '0';
                    i++;
                }

                i--;
                integerstack.push(count);
            }

            else if (str.charAt(i) == ']'){
                temp = "";
                count = 0;

                if (!integerstack.isEmpty()){
                    count = integerstack.peek();
                    integerstack.pop();
                }

                while (!stringstack.isEmpty() && stringstack.peek()!='[' ){
```

shekharkrish13@gmail.com

```
                temp = stringstack.peek() + temp;
                stringstack.pop();
            }

            if (!stringstack.empty() && stringstack.peek() == '[')
                stringstack.pop();

            for (int j = 0; j < count; j++)
                result = result + temp;

            for (int j = 0; j < result.length(); j++)
                stringstack.push(result.charAt(j));

            result = "";
        }

        else if (str.charAt(i) == '['){
            if (Character.isDigit(str.charAt(i-1)))
                stringstack.push(str.charAt(i));

            else{
                stringstack.push(str.charAt(i));
                integerstack.push(1);
            }
        }

        else
            stringstack.push(str.charAt(i));
    }

    while (!stringstack.isEmpty()){
        result = stringstack.peek() + result;
        stringstack.pop();
    }
    return result;
}

public static void main(String args[]){
    String str = "3[b2[ca]]";
    System.out.println(decode(str));
```

shekharkrish13@gmail.com

```
        }
}
```

## Solution 4 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.io.*;
import java.util.*;

class Solution{

        public static int maxWater(int[] height){
                Stack<Integer> stack = new Stack<>();
                int n = height.length;
                int ans = 0;
                for (int i = 0; i < n; i++) {
                        while ((!stack.isEmpty())
                                && (height[stack.peek()] < height[i])) {
                                int pop_height = height[stack.peek()];
                                stack.pop();
                                if (stack.isEmpty())
                                        break;
                                int distance = i - stack.peek() - 1;
                                int min_height
                                        = Math.min(height[stack.peek()],
                                                        height[i])
                                        - pop_height;

                                ans += distance * min_height;
                        }
                        stack.push(i);
                }

                return ans;
        }

        public static void main(String[] args){
                int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
```

shekharkrish13@gmail.com

```
        System.out.print(maxWater(arr));
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# Queue Solutions

## Solution 1 :

Time Complexity  : o(n)
Space Complexity: o(n)

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

public class Solution {
	static void generatePrintBinary(int n){
		Queue<String> q = new LinkedList<String>();
		q.add("1");
		while (n-- > 0) {
			String s1 = q.peek();
			q.remove();
			System.out.println(s1);
			String s2 = s1;
			q.add(s1 + "0");
			q.add(s2 + "1");
		}
	}

	public static void main(String[] args){
		int n = 10;
		generatePrintBinary(n);
	}
}
```

## Solution 2 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.util.*;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
class Solution{
	static int minCost(int arr[], int n){
		PriorityQueue<Integer> pq
			= new PriorityQueue<Integer>();

		for (int i = 0; i < n; i++) {
			pq.add(arr[i]);
		}

		int res = 0;
		while (pq.size() > 1) {
			int first = pq.poll();
			int second = pq.poll();
			res += first + second;
			pq.add(first + second);
		}
		return res;
	}

	public static void main(String args[]){
		int len[] = { 4, 3, 2, 6 };
		int size = len.length;
		System.out.println("Total cost for connecting"
						+ " ropes is "
						+ minCost(len, size));
	}
}
```

## Solution 3 :

Time Complexity : o(nlogn)
Space Complexity: o(n)

```
import java.util.*;

class Solution {
	static class Job {
		char job_id;
		int deadline;
		int profit;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        Job(char job_id, int deadline, int profit){
                this.deadline = deadline;
                this.job_id = job_id;
                this.profit = profit;
        }
}

static void printJobScheduling(ArrayList<Job> arr){
        int n = arr.size();
        Collections.sort(arr, (a, b) -> {
                return a.deadline - b.deadline;
        });
        ArrayList<Job> result = new ArrayList<>();
        PriorityQueue<Job> maxHeap = new PriorityQueue<>(
                (a, b) -> { return b.profit - a.profit; });
        for (int i = n - 1; i > -1; i--) {
                int slot_available;
                if (i == 0) {
                        slot_available = arr.get(i).deadline;
                }
                else {
                        slot_available = arr.get(i).deadline
                                                - arr.get(i - 1).deadline;
                }
                maxHeap.add(arr.get(i));
                while (slot_available > 0
                        && maxHeap.size() > 0) {
                        Job job = maxHeap.remove();
                        slot_available--;
                        result.add(job);
                }
        }

        Collections.sort(result, (a, b) -> {
                return a.deadline - b.deadline;
        });

        for (Job job : result) {
                System.out.print(job.job_id + " ");
        }
```

shekharkrish13@gmail.com

```
        System.out.println();
    }

    public static void main(String[] args){
        ArrayList<Job> arr = new ArrayList<Job>();

        arr.add(new Job('a', 2, 100));
        arr.add(new Job('b', 1, 19));
        arr.add(new Job('c', 2, 27));
        arr.add(new Job('d', 1, 25));
        arr.add(new Job('e', 3, 15));

        System.out.println("Following is maximum "
                                + "profit sequence of jobs");
        printJobScheduling(arr);
    }
}
```

## Solution 4 :

Time Complexity  : o(n+k)
Space Complexity: o(k)

```
import java.io.*;
import java.util.*;

import java.util.*;

class Solution {

    static class cell {
        int x, y;
        int dis;
        public cell(int x, int y, int dis){
            this.x = x;
            this.y = y;
            this.dis = dis;
```

shekharkrish13@gmail.com

```
        }
}

static boolean isInside(int x, int y, int N){
        if (x >= 1 && x <= N && y >= 1 && y <= N)
                return true;
        return false;
}

static int minStepToReachTarget(
        int knightPos[], int targetPos[],
        int N){
        int dx[] = { -2, -1, 1, 2, -2, -1, 1, 2 };
        int dy[] = { -1, -2, -2, -1, 1, 2, 2, 1 };

        Vector<cell> q = new Vector<>();
        q.add(new cell(knightPos[0], knightPos[1], 0));
        cell t;
        int x, y;
        boolean visit[][] = new boolean[N + 1][N + 1];
        visit[knightPos[0]][knightPos[1]] = true;
        while (!q.isEmpty()) {
                t = q.firstElement();
                q.remove(0);
                if (t.x == targetPos[0] && t.y == targetPos[1])
                        return t.dis;
                for (int i = 0; i < 8; i++) {
                        x = t.x + dx[i];
                        y = t.y + dy[i];
                        if (isInside(x, y, N) && !visit[x][y]) {
                                visit[x][y] = true;
                                q.add(new cell(x, y, t.dis + 1));
                        }
                }
        }
        return Integer.MAX_VALUE;
}

public static void main(String[] args){
        int N = 30;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
            int knightPos[] = { 1, 1 };
            int targetPos[] = { 30, 30 };
            System.out.println(
                    minStepToReachTarget(
                            knightPos, targetPos, N));
        }
}
```

## Solution 5 :

Time Complexity  : o(n)
Space Complexity: o(k)

```
import java.util.Deque;
import java.util.LinkedList;

public class Solution {

    static void printMax(int arr[], int n, int k){

        Deque<Integer> Qi = new LinkedList<Integer>();
        int i;
        for (i = 0; i < k; ++i) {
            while (!Qi.isEmpty() && arr[i] >=
                        arr[Qi.peekLast()])
                Qi.removeLast();
            Qi.addLast(i);
        }
        for (; i < n; ++i) {
            System.out.print(arr[Qi.peek()] + " ");
            while ((!Qi.isEmpty()) && Qi.peek() <=
                                i - k)
                Qi.removeFirst();
            while ((!Qi.isEmpty()) && arr[i] >=
                        arr[Qi.peekLast()])
                Qi.removeLast();
            Qi.addLast(i);
        }
        System.out.print(arr[Qi.peek()]);
    }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
    public static void main(String[] args){
        int arr[] = { 12, 1, 78, 90, 57, 89, 56 };
        int k = 3;
        printMax(arr, arr.length, k);
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# Greedy Solutions

## Solution 1:

Time Complexity  : o(n)
Space Complexity: o(1)

```
import java.util.*;

class Solution {

static int BalancedPartition(String str, int n){

        if (n == 0)
                return 0;

        int r = 0, l = 0;
        int ans = 0;
        for(int i = 0; i < n; i++)  {
        if (str.charAt(i) == 'R'){
                r++;
        }

        else if (str.charAt(i) == 'L'){
                l++;
        }

        if (r == l){
                ans++;
        }
        }

        return ans;
}

public static void main(String[] args){
        String str = "LLRRRLLRRL";
        int n = str.length();

        System.out.print(BalancedPartition(str, n) + "\n");
}
}
```

APNA COLLEGE
shekharkrish13@gmail.com

## Solution 2 :

Time Complexity  : o(1)
Space Complexity: o(1)

```
class Solution {

        public static int kthOdd(int[] range, int K) {

                if (K <= 0)
                        return 0;

                int L = range[0];
                int R = range[1];

                if ((R & 1) > 0) {
                        int Count = (int) Math.ceil((R - L + 1) / 2);
                        if (K > Count)
                                return 0;
                        else
                                return (R - 2 * K + 2);
                } else {
                        int Count = (R - L + 1) / 2;
                        if (K > Count)
                                return 0;
                        else
                                return (R - 2 * K + 1);
                }
        }

        public static void main(String args[]){
                int[] p = { -10, 10 };
                int k = 8;
                System.out.println(kthOdd(p, k));
        }
}
```

## Solution 3 :

shekharkrish13@gmail.com

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.util.Arrays;
public class Main {
	public static char[] lexo_small(int n, int k){
		char arr[] = new char[n];
		Arrays.fill(arr, 'a');
		for (int i = n - 1; i >= 0; i--) {
			k -= i;
			if (k >= 0) {
				if (k >= 26) {
					arr[i] = 'z';
					k -= 26;
				}

				else {
					arr[i] = (char)(k + 97 - 1);
					k -= arr[i] - 'a' + 1;
				}
			}

			else
				break;

			k += i;
		}
		return arr;
	}

	public static void main(String[] args){
		int n = 5, k = 42;

		char arr[] = lexo_small(n, k);

		System.out.print(new String(arr));
	}
}
```

APNA COLLEGE
shekharkrish13@gmail.com

## Solution 4 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Solution {
static int maxProfit(int prices[], int n)
{
        int buy = prices[0], max_profit = 0;
        for (int i = 1; i < n; i++) {
        if (buy > prices[i])
                buy = prices[i];
        else if (prices[i] - buy > max_profit)
                max_profit = prices[i] - buy;
        }
        return max_profit;
}

public static void main(String args[]){
        int prices[] = { 7, 1, 5, 6, 4 };
        int n = prices.length;
        int max_profit = maxProfit(prices, n);
        System.out.println(max_profit);
}
}
```

## Solution 5 :

Time Complexity :O((N−1)c(K−1))
(Here 'c' here depicts combinations i.e. ((n-1)!/((n-k)!*(k-1)!) Where N is the number of elements of the array and K is the number of divisions that we are having)
Space Complexity: o(n)

```
class Solution {
        public static int ans = 10000000;
        public static void solve(int a[], int n, int k,
                                                int index, int sum, int maxsum){
```

shekharkrish13@gmail.com

```
            if (k == 1) {
                maxsum = Math.max(maxsum, sum);
                sum = 0;
                for (int i = index; i < n; i++) {
                    sum += a[i];
                }
                maxsum = Math.max(maxsum, sum);
                ans = Math.min(ans, maxsum);
                return;
            }
            sum = 0;
            for (int i = index; i < n; i++) {
                sum += a[i];
                maxsum = Math.max(maxsum, sum);
                solve(a, n, k - 1, i + 1, sum, maxsum);
            }
        }
        public static void main(String[] args){
            int arr[] = { 1, 2, 3, 4 };
            int k = 3; // K divisions
            int n = 4; // Size of Array
            solve(arr, n, k, 0, 0, 0);
            System.out.println(ans + "\n");
        }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# Binary Tree Solutions

## Solution 1:

Time Complexity  : o(h)
Space Complexity: o(1)

```
import java.util.*;
class Solution{

static class Node{
        int data;
        Node left;
        Node right;
};

static Node newNode(int data){
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return (temp);
}

static boolean isUnivalTree(Node root){

        if (root == null){
                return true;
        }

        if (root.left != null
                && root.data != root.left.data)
                return false;

        if (root.right != null
                && root.data != root.right.data)
                return false;

        return isUnivalTree(root.left)
                && isUnivalTree(root.right);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
}

public static void main(String[] args){

        Node root = newNode(1);
        root.left = newNode(1);
        root.right = newNode(1);
        root.left.left = newNode(1);
        root.left.right = newNode(1);
        root.right.right = newNode(1);

        if (isUnivalTree(root))  {
                System.out.print("YES");
        }
        else{
                System.out.print("NO");
        }
}
}
```

## Solution 2 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
class Node{
        int data;
        Node left, right;

        public Node(int item)
        {
                data = item;
                left = right = null;
        }
}

class Solution{
        Node root;

        void mirror(){
                root = mirror(root);
        }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
Node mirror(Node node){
        if (node == null)
                return node;

        /* do the subtrees */
        Node left = mirror(node.left);
        Node right = mirror(node.right);

        /* swap the left and right pointers */
        node.left = right;
        node.right = left;

        return node;
}

void inOrder(){
        inOrder(root);
}

void inOrder(Node node){
        if (node == null)
                return;

        inOrder(node.left);
        System.out.print(node.data + " ");

        inOrder(node.right);
}

public static void main(String args[]){
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("Inorder traversal of input tree is :");
        tree.inOrder();
        System.out.println("");
        tree.mirror();
        System.out.println("Inorder traversal of binary tree is : ");
        tree.inOrder();

}
```

APNA COLLEGE
shekharkrish13@gmail.com

```
}
```

## Solution 3 :

Time Complexity  : o(n)
Space Complexity: o(1)

```
class Solution {

static class Node {
        int data;
        Node left, right;
}

static Node newNode(int data){
        Node newNode = new Node();
        newNode.data = data;
        newNode.left = null;
        newNode.right = null;
        return (newNode);
}

static Node deleteLeaves(Node root, int x){
        if (root == null)
                return null;
        root.left = deleteLeaves(root.left, x);
        root.right = deleteLeaves(root.right, x);

        if (root.data == x && root.left == null && root.right == null) {

                return null;
        }
        return root;
}

static void inorder(Node root){
        if (root == null)
                return;
        inorder(root.left);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        System.out.print(root.data + " ");
        inorder(root.right);
}

public static void main(String[] args){
        Node root = newNode(10);
        root.left = newNode(3);
        root.right = newNode(10);
        root.left.left = newNode(3);
        root.left.right = newNode(1);
        root.right.right = newNode(3);
        root.right.right.left = newNode(3);
        root.right.right.right = newNode(3);
        deleteLeaves(root, 3);
        System.out.print("Inorder traversal after deletion : ");
        inorder(root);
}
}
```

## Solution 4 :

Time Complexity  : o(n*n)
Space Complexity: o(n*n)

```
import java.util.HashMap;
public class Solution {

        static HashMap<String, Integer> m;
        static class Node {
                int data;
                Node left;
                Node right;
                Node(int data){
                        this.data = data;
                        left = null;
                        right = null;
                }
        }
```

shekharkrish13@gmail.com

```
    static String inorder(Node node){
        if (node == null)
            return "";

        String str = "(";
        str += inorder(node.left);
        str += Integer.toString(node.data);
        str += inorder(node.right);
        str += ")";

        if (m.get(str) != null && m.get(str)==1 )
            System.out.print( node.data + " ");

        if (m.containsKey(str))
            m.put(str, m.get(str) + 1);
        else
            m.put(str, 1);

        return str;
    }

    static void printAllDups(Node root){
        m = new HashMap<>();
        inorder(root);
    }

    public static void main(String args[]){
        Node root = null;
        root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.right.left = new Node(2);
        root.right.left.left = new Node(4);
        root.right.right = new Node(4);
        printAllDups(root);
    }
}
```

Solution 5 :

shekharkrish13@gmail.com

Time Complexity  : o(n)
Space Complexity: o(1)

```
class Node {

        int data;
        Node left, right;

        public Node(int item) {
                data = item;
                left = right = null;
        }
}

class Res {
        public int val;
}

class Solution {
        Node root;
        int findMaxUtil(Node node, Res res)   {

                if (node == null)
                        return 0;

                int l = findMaxUtil(node.left, res);
                int r = findMaxUtil(node.right, res);

                int max_single = Math.max(Math.max(l, r) + node.data,
                                                        node.data);

                int max_top = Math.max(max_single, l + r + node.data);

                res.val = Math.max(res.val, max_top);

                return max_single;
        }

        int findMaxSum() {
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        return findMaxSum(root);
    }

    int findMaxSum(Node node) {

        Res res = new Res();
        res.val = Integer.MIN_VALUE;

        findMaxUtil(node, res);
        return res.val;
    }

    public static void main(String args[]) {
        Solution tree = new Solution();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(10);
        tree.root.left.left = new Node(20);
        tree.root.left.right = new Node(1);
        tree.root.right.right = new Node(-25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);
        System.out.println("maximum path sum is : " +
                                                tree.findMaxSum());
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# BST Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

class Solution{

        static class Node{
        int val;
        Node left, right;
        };

        static Node newNode(int item){
        Node temp = new Node();
        temp.val = item;
        temp.left = temp.right = null;
        return temp;
        }


        static int sum = 0;

        static int rangeSumBST(Node root, int low,
                                                int high){
                if (root == null)
                return 0;

                Queue<Node> q = new LinkedList<Node>();
                q.add(root);

                while (q.isEmpty() == false){

                        Node curr = q.peek();
                        q.remove();
                        if (curr.val >= low &&
                                curr.val <= high){
                                sum += curr.val;
                        }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                if (curr.left != null &&
                        curr.val > low) q.add(curr.left);
                if (curr.right != null &&
                        curr.val < high)
                        q.add(curr.right);
            }
            return sum;
        }

        static Node insert(Node node, int data){

            if (node == null)
                return newNode(data);
            if (data <= node.val)
                node.left = insert(node.left,
                                        data);
            else
                node.right = insert(node.right,
                                    data);
            return node;
        }

        public static void main(String[] args){

            Node root = null;
            root = insert(root, 10);
            insert(root, 5);
            insert(root, 15);
            insert(root, 3);
            insert(root, 7);
            insert(root, 18);

            int L = 7, R = 15;
            System.out.print(rangeSumBST(root, L, R));
        }
        }
```

## Solution 2:

Time Complexity : o(h)

Space Complexity: o(1)

```
class Solution{
```

shekharkrish13@gmail.com

```
        static int min_diff, min_diff_key; static class Node{
        int key;
        Node left, right;
};

static Node newnode(int key){
        Node node = new Node();
        node.key = key;
        node.left = node.right = null;
        return (node);
}

         static void
  maxDiffUtil(Node ptr, int
      k){ if (ptr == null)
                return ;

        if (ptr.key == k){
                min_diff_key = k;
                return;
        }

        if (min_diff > Math.abs(ptr.key - k)){
                min_diff = Math.abs(ptr.key - k);
                min_diff_key = ptr.key;
        }

        if (k < ptr.key)
                maxDiffUtil(ptr.left, k);
        else
                maxDiffUtil(ptr.right, k);
}

static int maxDiff(Node root, int k){
        min_diff = 999999999;
        min_diff_key = -1;
        maxDiffUtil(root, k);
        return min_diff_key;
}
```

APNA COLLEGE
shekharkrish13@gmail.com

```
public static void main(String args[]){
        Node root = newnode(9); root.left = newnode(4);
        root.right = newnode(17);
        root.left.left = newnode(3);
        root.left.right = newnode(6);
        root.left.right.left = newnode(5);
        root.left.right.right = newnode(7);
        root.right.right = newnode(22);
        root.right.right.left = newnode(20);
        int k = 18;
        System.out.println( maxDiff(root, k));

}
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(h)

```
import java.io.*;

class Node {
        int data;
        Node left, right;
        Node(int x)
        {
                data = x;
                left = right = null;
        }
}

class Solution {

        static int count = 0;
        public static Node insert(Node
                root, int x) { if (root ==
                null)
                        return new Node(x);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        if (x < root.data)
            root.left = insert(root.left, x);
        else if (x > root.data) root.right = insert(root.right, x);
        return root;
    }

    public static Node kthSmallest(Node root, int k){

        if (root == null)
            return null;

        Node left = kthSmallest(root.left, k);
        if (left != null)
            return left;
        count++;
        if (count == k)
            return root;
        return kthSmallest(root.right, k);
    }

    public static void printKthSmallest(Node root, int k){
        Node res = kthSmallest(root, k);
        if (res == null)
          System.out.println("There are less than k nodes in the BST");
        else
            System.out.println("K-th Smallest Element is " + res.data);
    }

    public static void main(String[] args){
        Node root = null;
        int keys[] = { 20, 8, 22, 4, 12, 10, 14 };
        for (int x : keys)
            root = insert(root, x);
        int k = 3;
        printKthSmallest(root, k);
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

Solution 4 :

Time Complexity : o(n1+n2)
Space Complexity: o(h1+h2)

```
import java.util.Stack;
public class Solution {

        static class Node {
                int data;
                Node left, right;

                public Node(int data) {
                        this.data = data;
                        left = null;
                        right = null;
                }
        }

        static Node root1;
        static Node root2;
        static int countPairs(Node root1, Node root2,
                                                        int x)
        {
                if (root1 == null || root2 == null)
                        return 0;
                Stack<Node> st1 = new Stack<>();
                Stack<Node> st2 = new Stack<>();
                Node top1, top2;

                int count = 0;
                while (true) {
                        while (root1 != null) {
                                st1.push(root1);
                                root1 = root1.left;
                        }
                        while (root2 != null) {
                                st2.push(root2);
                                root2 = root2.right;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                }
                if (st1.empty() || st2.empty()) break;

                top1 = st1.peek();
                top2 = st2.peek();
                if ((top1.data + top2.data) == x) {
                        count++;
                        st1.pop();
                        st2.pop();
                        root1 = top1.right;
                        root2 = top2.left;
                }

                else if ((top1.data + top2.data) < x) {
                        st1.pop();
                        root1 = top1.right;
                }

                else {
                        st2.pop();
                        root2 = top2.left;
                }
        }
        return count;
}

public static void main(String args[])
{
        root1 = new Node(5);
        root1.left = new Node(3);
        root1.right = new Node(7);
        root1.left.left = new Node(2);
        root1.left.right = new Node(4);
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        root2 = new Node(10);
        root2.left = new Node(6);
        root2.right = new Node(15);
        root2.left.left = new Node(3);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
            root2.left.right = new Node(8); root2.right.left = new Node(11);
            root2.right.right = new Node(18);

            int x = 16;
            System.out.println("Pairs = "
                    + countPairs(root1, root2, x));
        }
}
```

## Solution 5 :

Time Complexity : o(n)
Space Complexity: o(n)

```
class Solution {

static class Node{
        Node left;
        Node right;
        int data;

        Node(int data){
                this.data = data;
                this.left = null;
                this.right = null;
        }
};

static class Info{
        int max;
        int min;
        boolean isBST;
        int sum;
        int currmax;

        Info(int m,int mi, boolean is,
                int su, int cur) {
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        max = m;
                min = mi;
                isBST = is;
                sum = su;
                currmax = cur;
        }
        Info(){}
};

static class INT{
        int a;
}

static Info MaxSumBSTUtil( Node root, INT maxsum){
        if (root == null)
                return new Info( Integer.MIN_VALUE,
                                Integer.MAX_VALUE, true, 0, 0 );
        if (root.left == null && root.right == null){
                maxsum.a = Math.max(maxsum.a, root.data);
                return new Info( root.data, root.data,
                                true, root.data, maxsum.a );
        }

        Info L = MaxSumBSTUtil(root.left, maxsum);
        Info R = MaxSumBSTUtil(root.right, maxsum);

        Info BST=new Info();
        if (L.isBST && R.isBST && L.max < root.data &&
                                R.min > root.data) {

                BST.max = Math.max(root.data, Math.max(L.max, R.max));
                BST.min = Math.min(root.data, Math.min(L.min, R.min));

                maxsum.a = Math.max(maxsum.a, R.sum +
                root.data + L.sum); BST.sum = R.sum + root.data +
                L.sum;
                BST.currmax = maxsum.a;

                BST.isBST = true;
                return BST;
```

shekharkrish13@gmail.com

```
        }
        BST.isBST = false;
        BST.currmax = maxsum.a;
        BST.sum = R.sum + root.data + L.sum;
                                                return BST;

}


static int MaxSumBST( Node root){
        INT maxsum = new INT();
        maxsum.a = Integer.MIN_VALUE;
        return MaxSumBSTUtil(root,
maxsum).currmax; }

public static void main(String args[]){
        Node root = new Node(5);
        root.left = new Node(14);
        root.right = new Node(3);
        root.left.left = new Node(6);
        root.right.right = new Node(7);
        root.left.left.left = new Node(9);
        root.left.left.right = new Node(1);

        System.out.println( MaxSumBST(root));
}
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# Heap Solutions

## Solution 1:

Time Complexity  : o(logk)
Space Complexity: o(1)

```
import java.io.*;
import java.util.*;

class Solution {
static PriorityQueue<Integer> min;
static int k;

static List<Integer> getAllKthNumber(int arr[]){

        List<Integer> list = new ArrayList<>();
        for (int val : arr) {
        if (min.size() < k)
                min.add(val);
        else {
                if (val > min.peek()) {
                min.poll();
                min.add(val);
                }
        }

        if (min.size() >= k)
                list.add(min.peek());
        else
                list.add(-1);
        }
        return list;
}

public static void main(String[] args){
        min = new PriorityQueue<>();
        k = 4;
        int arr[] = { 1, 2, 3, 4, 5, 6 };
        List<Integer> res = getAllKthNumber(arr);
        for (int x : res)
        System.out.print(x + " ");
}
```

shekharkrish13@gmail.com

}

## Solution 2 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.io.*;
import java.util.*;
class Solution{
        public static void minTime(int arr[],
                                                int N, int K){

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

                boolean vis[] = new boolean[N + 1];
                int time = 0;

                for (int i = 0; i < K; i++) {
                        q.add(arr[i]);
                        vis[arr[i]] = true;
                }

                while (q.size() > 0) {
                        for (int i = 0; i < q.size(); i++) {
                                int curr = q.poll();
                                if (curr - 1 >= 1 &&
                                        !vis[curr - 1]) {
                                        vis[curr - 1] = true;
                                        q.add(curr - 1);
                                }

                                if (curr + 1 <= N &&
                                        !vis[curr + 1]) {
                                        vis[curr + 1] = true;
                                        q.add(curr + 1);
                                }
                        }
                        time++;
```

shekharkrish13@gmail.com

```
            }

            System.out.println(time - 1);
        }

        public static void main(String[] args){
            int N = 6;
            int arr[] = { 2, 6 };
            int K = arr.length;
            minTime(arr, N, K);
        }
}
```

## Solution 3 :

Time Complexity  : o(n2logn)
Space Complexity: o(n2)

```
import java.util.Stack;

class Solution{
        static String decode(String str){
            Stack<Integer> integerstack = new Stack<>();
            Stack<Character> stringstack = new Stack<>();
            String temp = "", result = "";
            for (int i = 0; i < str.length(); i++){
                int count = 0;
                if (Character.isDigit(str.charAt(i))){
                    while (Character.isDigit(str.charAt(i))){
                        count = count * 10 + str.charAt(i) - '0';
                        i++;
                    }

                    i--;
                    integerstack.push(count);
                }

                else if (str.charAt(i) == ']'){
                    temp = "";
                    count = 0;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                if (!integerstack.isEmpty()){
                        count = integerstack.peek();
                        integerstack.pop();
                }

                while (!stringstack.isEmpty() && stringstack.peek()!='[' ){
                        temp = stringstack.peek() + temp;
                        stringstack.pop();
                }

                if (!stringstack.empty() && stringstack.peek() == '[')
                        stringstack.pop();

                for (int j = 0; j < count; j++)
                        result = result + temp;

                for (int j = 0; j < result.length(); j++)
                        stringstack.push(result.charAt(j));

                result = "";
        }

        else if (str.charAt(i) == '['){
                if (Character.isDigit(str.charAt(i-1)))
                        stringstack.push(str.charAt(i));

                else{
                        stringstack.push(str.charAt(i));
                        integerstack.push(1);
                }
        }

        else
                stringstack.push(str.charAt(i));
}

while (!stringstack.isEmpty()){
        result = stringstack.peek() + result;
        stringstack.pop();
```

shekharkrish13@gmail.com

```
        }
        return result;
    }

    public static void main(String args[]){
        String str = "3[b2[ca]]";
        System.out.println(decode(str));
    }
}
```

## Solution 4 :

Time Complexity  : o(nlogn)
Space Complexity: o(n)

```
import java.util.*;
import java.io.*;

class Solution{

static int minops(ArrayList<Integer> nums){
    int sum = 0;
    for(int i = 0 ; i < nums.size() ; i++){
    sum += nums.get(i);
    }

    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    for(int i = 0 ; i < nums.size() ; i++){
    pq.add(-nums.get(i));
    }

    double temp = sum;
    int cnt = 0;
    while (temp > sum / 2) {
    int x = -pq.peek();
    pq.remove();
    temp -= Math.ceil(x * 1.0 / 2);
    pq.add(x / 2);
    cnt++;
    }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        return cnt;
}

public static void main(String args[]){
        ArrayList<Integer> nums = new ArrayList<Integer>(
        List.of(
                4, 6, 3, 9, 10, 2
        )
        );
        int count = minops(nums);
        System.out.println(count);
}
}
```

## Solution 5 :

Time Complexity  : o(n*k*logk)
Space Complexity: o(k)

```
import java.io.*;
import java.util.*;

class Node {
        int data;
        Node next;

        Node(int key){
                data = key;
                next = null;
        }
}

class NodeComparator implements Comparator<Node> {
        public int compare(Node k1, Node k2){
                if (k1.data > k2.data)
                        return 1;
                else if (k1.data < k2.data)
                        return -1;
                return 0;
        }
```

APNA COLLEGE
shekharkrish13@gmail.com

```
}
class Solution {
	static Node mergeKList(Node[] arr, int K){
		PriorityQueue<Node> queue
			= new PriorityQueue<>(new NodeComparator());
		Node at[] = new Node[K];
		Node head = new Node(0);
		Node last = head;
		for (int i = 0; i < K; i++) {
			if (arr[i] != null) {
				queue.add(arr[i]);
			}
		}
		if (queue.isEmpty())
			return null;
		while (!queue.isEmpty()) {
			Node curr = queue.poll();
			last.next = curr;
			last = last.next;
			if (curr.next != null) {
				queue.add(curr.next);
			}
		}
		return head.next;
	}
	public static void printList(Node node){
		while (node != null) {
			System.out.print(node.data + " ");
			node = node.next;
		}
	}

	public static void main(String[] args){
		int N = 3;
		Node[] a = new Node[N];
		Node head1 = new Node(1);
		a[0] = head1;
		head1.next = new Node(3);
		head1.next.next = new Node(5);
		head1.next.next.next = new Node(7);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        Node head2 = new Node(2);
        a[1] = head2;
        head2.next = new Node(4);
        head2.next.next = new Node(6);
        head2.next.next.next = new Node(8);

        Node head3 = new Node(0);
        a[2] = head3;
        head3.next = new Node(9);
        head3.next.next = new Node(10);
        head3.next.next.next = new Node(11);

        Node res = mergeKList(a, N);

        if (res != null)
            printList(res);
        System.out.println();
    }
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# HASHING SOLUTIONS

## Solution 1:

Using the concept of Horizontal Distance (HD), as discussed in Top View of a Binary Tree Question in the Trees chapter.

```
import java.util.*;

class Solution {
   static class Node {
       int data;
       int hd;
       Node left, right;

       public Node(int key) {
           this.data = key;
           this.hd = Integer.MAX_VALUE;
           this.left = this.right = null;
       }
   }

       public static void bottomViewHelper(Node root, int curr, int hd,
TreeMap<Integer, int[]> m) {
       if (root == null)
           return;

       // If node for a particular HD is not present, add to the map.
       if (!m.containsKey(hd)) {
           m.put(hd, new int[]{ root.data, curr });
       }

       // Compare height for already
       // present node at similar horizontal
       // distance
       else {
           int[] p = m.get(hd);
           if (p[1] <= curr) {
               p[1] = curr;
```

shekharkrish13@gmail.com

```
            p[0] = root.data;
        }
        m.put(hd, p);
    }

    // call for left subtree
    bottomViewHelper(root.left, curr + 1, hd - 1, m);

    // call for right subtree
    bottomViewHelper(root.right, curr + 1, hd + 1, m);
}

public static void printBottomView(Node root) {
    // Map to store Horizontal Distance, Height and Data.
    TreeMap<Integer, int[]> m = new TreeMap<>();

    bottomViewHelper(root, 0, 0, m);

    // Prints the values stored by printBottomViewUtil()
    for(int val[] : m.values())
    {
        System.out.print(val[0] + " ");
    }
}

public static void main(String[] args) {
    Node root = new Node(20);
    root.left = new Node(8);
    root.right = new Node(22);
    root.left.left = new Node(5);
    root.left.right = new Node(3);
    root.right.left = new Node(4);
    root.right.right = new Node(25);
    root.left.right.left = new Node(10);
    root.left.right.right = new Node(14);

    System.out.println(
        "Bottom view of the given binary tree:");

    printBottomView(root);
```

shekharkrish13@gmail.com

```
    }
}
```

Solution 2:

```
public int[] twoSum(int[] arr, int target) {
        Map<Integer, Integer> visited = new HashMap<>();
        for(int i = 0; i<arr.length; i++) {
            //diff = given target - number given at ith index
            int diff = target - arr[i];

            // check if found difference is present in the MAP list
            if(visited.containsKey(diff)) {
                //if difference in map matches with the ith index element in array
                return new int[] { i, visited.get(diff) };
            }
            //add arr element in map to match with future element if forms a pair
            visited.put(arr[i],i);
        }
        //if no matches are found
        return new int[] {0, 0};
    }
```

Solution 3 :

```
public String frequencySort(String s) {
        HashMap<Character , Integer>map = new HashMap<>();
        for(int i=0;i<s.length();++i)
            map.put(s.charAt(i),map.getOrDefault(s.charAt(i),0)+1);

      PriorityQueue<Map.Entry<Character,Integer>> pq =
          new PriorityQueue<>((a,b)->a.getValue()== b.getValue()?
            a.getKey()-b.getKey(): b.getValue()-a.getValue());

        for(Map.Entry<Character,Integer>e:map.entrySet())  pq.add(e);
```

shekharkrish13@gmail.com

```
        StringBuilder res = new StringBuilder();
      while(pq.size()!=0){
          char ch = pq.poll().getKey();
          int val = map.get(ch);
         while(val!=0){
             res.append(ch);
             val--;
         }
      }
      return res.toString();
  }
```

shekharkrish13@gmail.com

# TRIES SOLUTIONS

## Solution 1:

We solve using Tries.

```
iclass TrieNode {
   List<String> data;
   TrieNode children[];
   boolean isEnd;

   TrieNode(){
       data = new ArrayList<>();
       children = new TrieNode[26];
       isEnd = false;
   }
}

class Solution {
   static TrieNode root;
   List<List<String>> ans;
   public List<List<String>> groupAnagrams(String[] strs) {
       ans = new ArrayList<>();

       root = new TrieNode();

       for(String word: strs){
           build(word);
       }

       dfs(root);

       return ans;
   }

   public void build(String s){
       TrieNode temp = root;
       char[] word = s.toCharArray();
       Arrays.sort(word);
       for(char c: word){
```

shekharkrish13@gmail.com

```
            TrieNode child = temp.children[c-'a'];
            if(child == null) temp.children[c-'a'] = new TrieNode();
            temp = temp.children[c-'a'];
        }
        temp.isEnd = true;
        temp.data.add(s);
    }

    public void dfs(TrieNode rt){
        if(rt.isEnd){
            ans.add(rt.data);
        }

        for(int i = 0; i < 26; i++){
            if(rt.children[i] != null) dfs(rt.children[i]);
        }
    }
}
```

Solution 2:

```
class Solution {
    private static class Node {
        private char data;
        private String word;
        private boolean isEnd;
        private Node[] children;

        public Node(char data) {
            this.data = data;
            this.word = null;
            this.isEnd = false;
            this.children = new Node[26];
        }
    }

    private Node root = new Node('/');
    private String ans = "";
```

shekharkrish13@gmail.com

```
    private void insert(String word) {
        Node curr = this.root;
        for (int i = 0; i < word.length(); i++) {
            int childIdx = word.charAt(i) - 'a';
            if (curr.children[childIdx] == null) {
                curr.children[childIdx] = new Node(word.charAt(i));
            }
            curr = curr.children[childIdx];
        }
        curr.isEnd = true;
        curr.word = word;
    }

    private void dfs(Node node) {
        if (node == null) {
            return;
        }
        if (node.word != null) {
            if (node.word.length() > ans.length()) {
                ans = node.word;
                            } else if (node.word.length() == ans.length() &&
node.word.compareTo(ans) < 0) {
                ans = node.word;
            }

        }
        for (Node child : node.children) {
            if (child != null && child.word != null) {
                dfs(child);
            }
        }
    }

    public String longestWord(String[] words) {
        for (String word : words) {
            insert(word);
        }
        Node curr = this.root;
        dfs(curr);
        return ans;
```

shekharkrish13@gmail.com

```
    }
}
```

shekharkrish13@gmail.com

# Graph-1 Solutions

## Solution 1:

Time Complexity : o(v+e)
Space Complexity: o(v+e)

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;

class Solution{

public static void main(String arg[]){

        int V = 4;
        @SuppressWarnings("unchecked")
        ArrayList <Integer> adj[] = new ArrayList[V];
        for(int i = 0; i < 4; i++)
        adj[i] = new ArrayList<Integer>();

        addEdge(adj, 0, 1);
        addEdge(adj, 1, 2);
        addEdge(adj, 2, 0);
        addEdge(adj, 2, 3);

        if (isCyclicDisconntected(adj, V))
        System.out.println("Yes");
        else
        System.out.println("No");
}

static void addEdge(ArrayList<Integer> adj[], int u, int v){
        adj[u].add(v);
        adj[v].add(u);
}

static boolean isCyclicConntected(
                                ArrayList<Integer> adj[], int s,
                                        int V, boolean visited[]){
        int parent[] = new int[V];
```

shekharkrish13@gmail.com

```
        Arrays.fill(parent, -1);

        Queue<Integer> q = new LinkedList<>();
        visited[s] = true;
        q.add(s);

        while (!q.isEmpty()){
        int u = q.poll();
        for (int i = 0; i < adj[u].size(); i++){
                int v = adj[u].get(i);
                if (!visited[v]){
                visited[v] = true;
                q.add(v);
                parent[v] = u;
                }
                else if (parent[u] != v)
                return true;
        }
        }
        return false;
}

static boolean isCyclicDisconntected(
                                ArrayList<Integer> adj[], int V){
        boolean visited[] = new boolean[V];
        Arrays.fill(visited,false);

        for (int i = 0; i < V; i++)
        if (!visited[i] &&
                isCyclicConntected(adj, i, V, visited))
                return true;
        return false;
}
}
```

## Solution 2:

Time Complexity : o(n)

Space Complexity: o(1)

```
import java.util.*;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
class Solution{

static class Node{
        int data;
        Node left, right;
}

static class qItem{
        Node node;
        int depth;

        public qItem(Node node, int depth){
                this.node = node;
                this.depth = depth;
        }
}

static int minDepth(Node root){
        if (root == null)
                return 0;
        Queue<qItem> q = new LinkedList<>();
        qItem qi = new qItem(root, 1);
        q.add(qi);

        while (q.isEmpty() == false){
                qi = q.peek();
                q.remove();
                Node node = qi.node;
                int depth = qi.depth;
                if (node.left == null && node.right == null)
                        return depth;
                if (node.left != null){
                        qi.node = node.left;
                        qi.depth = depth + 1;
                        q.add(qi);
                }

                if (node.right != null){
                        qi.node = node.right;
                        qi.depth = depth + 1;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                        q.add(qi);
                }
        }
        return 0;
}

static Node newNode(int data){
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return temp;
}

public static void main(String[] args){
        Node root = newNode(1);
        root.left = newNode(2);
        root.right = newNode(3);
        root.left.left = newNode(4);
        root.left.right = newNode(5);

        System.out.println(minDepth(root));
}
}
```

## Solution 3 :

Time Complexity : o(r*c)
Space Complexity: o(r*c)

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

public class Solution{
        public final static int R = 3;
        public final static int C = 5;

        static class Ele{
                int x = 0;
```

shekharkrish13@gmail.com

```
        int y = 0;
        Ele(int x,int y){
            this.x = x;
            this.y = y;
        }
    }

    static boolean isValid(int i, int j){
        return (i >= 0 && j >= 0 && i < R && j < C);
    }

    static boolean isDelim(Ele temp){
        return (temp.x == -1 && temp.y == -1);
    }

    static boolean checkAll(int arr[][]){
        for (int i=0; i<R; i++)
            for (int j=0; j<C; j++)
                if (arr[i][j] == 1)
                    return true;
        return false;
    }

    static int Solution(int arr[][]){
        Queue<Ele> Q=new LinkedList<>();
        Ele temp;
        int ans = 0;
        for (int i=0; i < R; i++)
        for (int j=0; j < C; j++)
            if (arr[i][j] == 2)
                Q.add(new Ele(i,j));

        Q.add(new Ele(-1,-1));

        while(!Q.isEmpty()){
            boolean flag = false;
            while(!isDelim(Q.peek())){
                temp = Q.peek();
                if(isValid(temp.x+1, temp.y) && arr[temp.x+1][temp.y] == 1){
                    if(!flag){
```

shekharkrish13@gmail.com

```
                ans++;
                flag = true;
            }
            arr[temp.x+1][temp.y] = 2;

            temp.x++;
            Q.add(new Ele(temp.x,temp.y));
            temp.x--;
        }

        if (isValid(temp.x-1, temp.y) && arr[temp.x-1][temp.y] == 1){
                if (!flag){
                    ans++;
                    flag = true;
                }
                arr[temp.x-1][temp.y] = 2;
                temp.x--;
                Q.add(new Ele(temp.x,temp.y)); // push this cell
to Queue
                temp.x++;
        }

        if (isValid(temp.x, temp.y+1) && arr[temp.x][temp.y+1] == 1) {
                if(!flag){
                    ans++;
                    flag = true;
                }
                arr[temp.x][temp.y+1] = 2;
                temp.y++;
                Q.add(new Ele(temp.x,temp.y)); // Push this cell
to Queue
                temp.y--;
            }
        if (isValid(temp.x, temp.y-1) && arr[temp.x][temp.y-1] == 1){
                if (!flag){
                    ans++;
                    flag = true;
                }
                arr[temp.x][temp.y-1] = 2;
                temp.y--;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                                        Q.add(new Ele(temp.x,temp.y)); // push this cell
to Queue
                                }
                                Q.remove();

                        }
                        Q.remove();
                        if (!Q.isEmpty())
                        {
                                Q.add(new Ele(-1,-1));
                        }
                }
                return (checkAll(arr))? -1: ans;
        }

        public static void main(String[] args){
                int arr[][] = { {2, 1, 0, 2, 1},
                                        {1, 0, 1, 2, 1},
                                        {1, 0, 0, 2, 1}};
                int ans = Solution(arr);
                if(ans == -1)
                        System.out.println("All oranges cannot rot");
                else
                        System.out.println("Time required for all oranges to rot = " + ans);
        }

}
```

## Solution 4 :

Time Complexity : o(r*c)
Space Complexity: o(r*c)

```
import java.io.*;
import java.util.*;

class Solution {
```

shekharkrish13@gmail.com

```
static int ROW, COL, count;
static boolean isSafe(int[][] M, int row, int col,
                                        boolean[][] visited)
{
        return (
                (row >= 0) && (row < ROW) && (col >= 0)
                && (col < COL)
                && (M[row][col] == 1 && !visited[row][col]));
}

static void DFS(int[][] M, int row, int col,
                                boolean[][] visited){
        int[] rowNbr = { -1, -1, -1, 0, 0, 1, 1, 1 };
        int[] colNbr = { -1, 0, 1, -1, 1, -1, 0, 1 };

        visited[row][col] = true;
        for (int k = 0; k < 8; k++) {
                if (isSafe(M, row + rowNbr[k], col + colNbr[k],
                                visited)) {
                        count++;
                        DFS(M, row + rowNbr[k], col + colNbr[k],
                                visited);
                }
        }
}

static int largestRegion(int[][] M)        {

        boolean[][] visited = new boolean[ROW][COL];
        int result = 0;
        for (int i = 0; i < ROW; i++) {
                for (int j = 0; j < COL; j++) {
                        if (M[i][j] == 1 && !visited[i][j]) {
                                count = 1;
                                DFS(M, i, j, visited);
                                result = Math.max(result, count);
                        }
                }
        }
        return result;
```

shekharkrish13@gmail.com

```
        }

        public static void main(String args[]){
                int M[][] = { { 0, 0, 1, 1, 0 },
                                { 1, 0, 1, 1, 0 },
                                { 0, 1, 0, 0, 0 },
                                { 0, 0, 0, 0, 1 } };
                ROW = 4;
                COL = 5;
                System.out.println(largestRegion(M));
        }
}
```

## Solution 5 :

Time Complexity : o(n2*m)
Space Complexity: o(n*m)

```
import java.util.*;

class Solution {
static int shortestChainLen(String start,
                                String target,
                                Set<String> D){

        if(start == target)
        return 0;
        if (!D.contains(target))
                return 0;
        int level = 0, wordlength = start.length();
        Queue<String> Q = new LinkedList<>();
        Q.add(start);
        while (!Q.isEmpty()){
                ++level;
                int sizeofQ = Q.size();
                for (int i = 0; i < sizeofQ; ++i){
```

APNA COLLEGE
shekharkrish13@gmail.com

```
                char []word = Q.peek().toCharArray();
                Q.remove();
                for (int pos = 0; pos < wordlength; ++pos){
                        char orig_char = word[pos];
                        for (char c = 'a'; c <= 'z'; ++c){
                                word[pos] = c;
                                if (String.valueOf(word).equals(target))
                                        return level + 1;
                                if (!D.contains(String.valueOf(word)))
                                        continue;
                                D.remove(String.valueOf(word));
                                Q.add(String.valueOf(word));
                        }
                        word[pos] = orig_char;
                }
            }
        }

        return 0;
}

public static void main(String[] args){
        Set<String> D = new HashSet<String>();
        D.add("poon");
        D.add("plee");
        D.add("same");
        D.add("poie");
        D.add("plie");
        D.add("poin");
        D.add("plea");
        String start = "toon";
        String target = "plea";
        System.out.print("Length of shortest chain is: "
                + shortestChainLen(start, target, D));
}
}
```

APNA COLLEGE
shekharkrish13@gmail.com

APNA COLLEGE

shekharkrish13@gmail.com

# Graph-02 Solutions

## Solution 1:

Time Complexity  : o(v+e)
Space Complexity: o(v+e)

```
import java.util.*;

class Solution{

static void addEdge(int u, int v,
 ArrayList<ArrayList<Integer>> adj){
   adj.get(u).add(v);
}

static void DFSUtil(ArrayList<ArrayList<Integer>> g,
                int v, boolean[] visited){
   visited[v] = true;

   for(int x : g.get(v)){
      if (!visited[x]){
         DFSUtil(g, x, visited);
      }
   }
}

static int motherVertex(ArrayList<ArrayList<Integer>>g,
                    int V){

   boolean[] visited = new boolean[V];
   int v = -1;

   for(int i = 0; i < V; i++){
      if (!visited[i]){
         DFSUtil(g, i, visited);
         v = i;
      }
   }

   boolean[] check = new boolean[V];
   DFSUtil(g, v, check);
```

APNA COLLEGE
shekharkrish13@gmail.com

```
    for(boolean val : check){
        if (!val){
            return -1;
        }
    }
    return v;
}

 public static void main(String[] args){
    int V = 7;
    int E = 8;

    ArrayList<
    ArrayList<Integer>> adj = new ArrayList<
                            ArrayList<Integer>>();
    for(int i = 0; i < V; i++){
        adj.add(new ArrayList<Integer>());
    }
    addEdge(0, 1,adj);
    addEdge(0, 2,adj);
    addEdge(1, 3,adj);
    addEdge(4, 1,adj);
    addEdge(6, 4,adj);
    addEdge(5, 6,adj);
    addEdge(5, 2,adj);
    addEdge(6, 0,adj);

    System.out.println("The mother vertex is " +
                    motherVertex(adj, V));
}
}
```

## Solution 2:

Time Complexity : o(v+e)
Space Complexity: o(1)

```
import java.io.*;
import java.lang.*;
import java.util.*;

class Solution {
            int V, E;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
Edge edge[];

class Edge {
        int src, dest;
};

Solution(int v, int e){
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
                edge[i] = new Edge();
}

int find(int parent[], int i){
        if (parent[i] == i)
                return i;
        return find(parent, parent[i]);
}

void Union(int parent[], int x, int y){
        parent[x] = y;
}

int isCycle(Solution graph){
        int parent[] = new int[graph.V];

        for (int i = 0; i < graph.V; ++i)
                parent[i] = i;

        for (int i = 0; i < graph.E; ++i) {
                int x = graph.find(parent, graph.edge[i].src);
                int y = graph.find(parent, graph.edge[i].dest);

                if (x == y)
                        return 1;

                graph.Union(parent, x, y);
        }
        return 0;
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        }

        public static void main(String[] args){
                int V = 3, E = 3;
                Solution graph = new Solution(V, E);
                graph.edge[0].src = 0;
                graph.edge[0].dest = 1;
                graph.edge[1].src = 1;
                graph.edge[1].dest = 2;
                graph.edge[2].src = 0;
                graph.edge[2].dest = 2;

                if (graph.isCycle(graph) == 1)
                        System.out.println("Graph contains cycle");
                else
                        System.out.println(
                                "Graph doesn't contain cycle");
        }
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

public class Solution {

        static class pair {
                int first, second;

                pair(int first, int second){
                        this.first = first;
                        this.second = second;
                }
        }

        static ArrayList<ArrayList<Integer> >
```

APNA COLLEGE
shekharkrish13@gmail.com

```
make_graph(int numTasks, Vector<pair> prerequisites){
        ArrayList<ArrayList<Integer> > graph
                = new ArrayList<ArrayList<Integer> >(numTasks);

        for (int i = 0; i < numTasks; i++) {
                graph.add(new ArrayList<Integer>());
        }

        for (pair pre : prerequisites)
                graph.get(pre.second).add(pre.first);

        return graph;
}

static int[] compute_indegree(
        ArrayList<ArrayList<Integer> > graph)
{
        int degrees[] = new int[graph.size()];

        for (ArrayList<Integer> neighbors : graph)
                for (int neigh : neighbors)
                        degrees[neigh]++;

        return degrees;
}

static boolean canFinish(int numTasks,
                                        Vector<pair> prerequisites){
        ArrayList<ArrayList<Integer> > graph
                = make_graph(numTasks, prerequisites);
        int degrees[] = compute_indegree(graph);

        for (int i = 0; i < numTasks; i++) {
                int j = 0;
                for (; j < numTasks; j++)
                        if (degrees[j] == 0)
                                break;

                if (j == numTasks)
                        return false;
```

shekharkrish13@gmail.com

```
                degrees[j] = -1;
                for (int neigh : graph.get(j))
                        degrees[neigh]--;
        }

        return true;
    }

    public static void main(String args[]){
        int numTasks = 4;
        Vector<pair> prerequisites = new Vector<pair>();

        prerequisites.add(new pair(1, 0));
        prerequisites.add(new pair(2, 1));
        prerequisites.add(new pair(3, 2));

        if (canFinish(numTasks, prerequisites)) {
                System.out.println(
                        "Possible to finish all tasks");
        }
        else {
                System.out.println(
                        "Impossible to finish all tasks");
        }
    }
}
```

## Solution 4 :

Time Complexity : o(v+e)
Space Complexity: o(v+e)

```
class Solution {
   public String alienOrder(String[] words) {
      Map<Character, Set<Character>> map = new HashMap<>();
      Map<Character, Integer> degree = new HashMap<>();
      String result = "";
      if (words == null || words.length == 0) { return result; }
```

shekharkrish13@gmail.com

```
for (String s: words) {
    for (char c: s.toCharArray()) {
        degree.put(c, 0);
    }
}

for (int i = 0; i < words.length - 1; i++) {
    String curr = words[i];
    String next = words[i + 1];
    int min = Math.min(curr.length(), next.length());
    for (int j = 0; j < min; j++) {
        char c1 = curr.charAt(j);
        char c2 = next.charAt(j);
        if (c1 != c2) {
            Set<Character> set = map.getOrDefault(c1, new HashSet<>());
            if (!set.contains(c2)) {
                set.add(c2);
                map.put(c1, set);
                degree.put(c2, degree.get(c2) + 1); // update c2, c1 < c2
            }
            break;
        }
    }
}

LinkedList<Character> q = new LinkedList<>();
for (char c: degree.keySet()) {
    if (degree.get(c) == 0) {
        q.add(c);
    }
}

while (!q.isEmpty()) {
    char c = q.poll();
    result += c;
    if (map.containsKey(c)) {
        for (char next: map.get(c)) {
            degree.put(next, degree.get(next) - 1);
            if (degree.get(next) == 0) {
                q.offer(next);
```

shekharkrish13@gmail.com

```
                }
            }
        }
    }

    return result.length() == degree.size() ? result : "";
  }
}
```

## Solution 5 :

Time Complexity : o(n*m)
Space Complexity: o(n*m)

```
import java.util.*;

class Solution{

static void dfs(int[][] matrix, boolean[][] visited,
            int x, int y, int n, int m,
            boolean hasCornerCell){

  if (x < 0 || y < 0 || x >= n || y >= m ||
    visited[x][y] == true || matrix[x][y] == 0)
    return;

  if (x == 0 || y == 0 ||
    x == n - 1 || y == m - 1){
    if (matrix[x][y] == 1)
      hasCornerCell = true;
  }

  visited[x][y] = true;

  dfs(matrix, visited, x + 1, y, n, m,
    hasCornerCell);
  dfs(matrix, visited, x, y + 1, n, m,
```

APNA COLLEGE
shekharkrish13@gmail.com

```
        hasCornerCell);
    dfs(matrix, visited, x - 1, y, n, m,
        hasCornerCell);
    dfs(matrix, visited, x, y - 1, n, m,
        hasCornerCell);
}

static int countClosedIsland(int[][] matrix, int n,
                             int m){

    boolean[][] visited = new boolean[n][m];
    int result = 0;

    for(int i = 0; i < n; ++i) {
        for(int j = 0; j < m; ++j){
            if ((i != 0 && j != 0 &&
                i != n - 1 && j != m - 1) &&
                matrix[i][j] == 1 &&
                visited[i][j] == false) {

                boolean hasCornerCell = false;
                dfs(matrix, visited, i, j, n, m,
                    hasCornerCell);
                if (!hasCornerCell)
                    result = result + 1;
            }
        }
    }
    return result;
}

public static void main(String[] args){
    int N = 5, M = 8;

    int[][] matrix = { { 0, 0, 0, 0, 0, 0, 0, 1 },
                       { 0, 1, 1, 1, 1, 0, 0, 1 },
                       { 0, 1, 0, 1, 0, 0, 0, 1 },
                       { 0, 1, 1, 1, 1, 0, 1, 0 },
                       { 0, 0, 0, 0, 0, 0, 0, 1 } };
    System.out.print(countClosedIsland(matrix, N, M));
```

APNA COLLEGE
shekharkrish13@gmail.com

```
}
}
```

APNA COLLEGE
shekharkrish13@gmail.com

# DP Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.io.*;

class Solution {

	static void printTrib(int n){
		int dp[]=new int[n];
		dp[0] = dp[1] = 0;
		dp[2] = 1;

		for (int i = 3; i < n; i++)
			dp[i] = dp[i - 1] +dp[i - 2] +dp[i - 3];

		for (int i = 0; i < n; i++)
			System.out.print(dp[i] + " ");
	}

	public static void main(String args[]){
		int n = 10;
		printTrib(n);
	}
}
```

## Solution 2 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.io.*;

class Solution{
```

APNA COLLEGE
shekharkrish13@gmail.com

```
    static void _printParenthesis(char str[], int pos, int n, int open, int close) {
        if(close == n)  {
            for(int i=0;i<str.length;i++)
                System.out.print(str[i]);
            System.out.println();
            return;
        }
        else{
            if(open > close) {
                str[pos] = '}';
                _printParenthesis(str, pos+1, n, open, close+1);
            }
            if(open < n) {
                str[pos] = '{';
                _printParenthesis(str, pos+1, n, open+1, close);
            }
        }
    }

    static void printParenthesis(char str[], int n) {
        if(n > 0)
        _printParenthesis(str, 0, n, 0, 0);
        return;
    }

    public static void main (String[] args) {
        int n = 3;
        char[] str = new char[2 * n];
        printParenthesis(str, n);
    }
}
```

## Solution 3 :

Time Complexity : o(n2)
Space Complexity: o(1)

```
import java.util.*;
```

shekharkrish13@gmail.com

```
class Solution{

static int max_profit(int a[],int b[],int n,int fee){
int i, j, profit;
int l, r, diff_day = 1, sum = 0;

        b[0]=0;
        b[1]=diff_day;
for(i=1;i<n;i++){
        l=0;
        r=diff_day;
                sum=0;

        for(j=n-1;j>=i;j--){
                        profit=(a[r]-a[l])-fee;
                        if(profit>0){
                                sum=sum+profit;
                                }
                        l++;
                        r++;
                        }
        if(b[0] < sum){
        b[0] = sum;
        b[1] = diff_day;
        }
diff_day++;
}
return 0;
}

public static void main(String args[]){
        int arr[] = { 6, 1, 7, 2, 8, 4 };
        int n = arr.length;
        int[] b = new int[2];
        int tranFee = 2;

        max_profit(arr, b, n, tranFee);
        System.out.println(b[0]+", "+b[1]);
}
```

APNA COLLEGE
shekharkrish13@gmail.com

APNA COLLEGE

```
}
```

## Solution 4 :

Time Complexity : o(n2)
Space Complexity: o(n2)

```
import java.util.*;

class Solution {

        static int LIP(int dp[][], int mat[][], int n,
                                int m, int x, int y){
                if (dp[x][y] < 0) {
                        int result = 0;
                        if (x == n - 1 && y == m - 1)
                                return dp[x][y] = 1;
                        if (x == n - 1 || y == m - 1)
                                result = 1;
                        if (x + 1 < n && mat[x][y] < mat[x + 1][y])
                                result = 1 + LIP(dp, mat, n, m, x + 1, y);
                        if (y + 1 < m && mat[x][y] < mat[x][y + 1])
                                result = Math.max(result, 1 + LIP(dp, mat, n, m, x, y + 1));
                        dp[x][y] = result;
                }
                return dp[x][y];
        }

        static int wrapper(int mat[][], int n, int m){
                int dp[][] = new int[10][10];
                for (int i = 0; i < 10; i++)
                        Arrays.fill(dp[i], -1);

                return LIP(dp, mat, n, m, 0, 0);
        }

        public static void main(String[] args){
                int mat[][] = {
```

shekharkrish13@gmail.com

```
                    { 1, 2, 3, 4 },
                    { 2, 2, 3, 4 },
                    { 3, 2, 3, 4 },
                    { 4, 5, 6, 7 },
                };
                int n = 4, m = 4;
                System.out.println(wrapper(mat, n, m));
        }
}
```

Solution 5 :

```
public static int helper(int left, int right) {
     if (left == 0 && right == 0){
   ans++;
 }
 if (left > right){
   return 0;
 }

 if (left > 0){
   helper(left-1, right);
 }
 if (right > 0){
   helper(left, right-1);
 }
 return ans;

  }

  // Find possible ways for balanced parentheses
  private static int countWays(int n){
     // If n is odd no possible valid parentheses
     if ((n & 1) != 0)
        return 0;
     return helper(n / 2, n / 2);
  }
```

APNA COLLEGE
shekharkrish13@gmail.com