1. Maximum Subarray Sum – Kadane's Algorithm:
   Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.
   Input: arr[] = {2, 3, -8, 7, -1, 2, 3}
   Output: 11
   Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

   Input: arr[] = {-2, -4}
   Output: –2
   Explanation: The subarray {-2} has the largest sum -2.

```
 GNU nano 7.2
import java.util.*;
public class KadanesAlgo{
        static int FindMaxSumSubSeq(int arr[],int n){
                int sum=0,maxi=Integer.MIN_VALUE;
                for(int i=0;i<n;i++){
                        sum+=arr[i];
                        maxi=Math.max(maxi,sum);
                        if(sum<0){sum=0;}
                }
                return maxi;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(FindMaxSumSubSeq(arr,n));
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(n)

TestCases:

```
  GNU nano 7.2
5
7
2 3 -8 7 -1 2 3
2
-2 -4
2
-4 -2
5
1 2 3 4 5
6
1 -2 3 -4 5 -6
```

OutPut:

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java KadanesAlgo < KadanesAlgo.txt
Test Case: 1
11
Test Case: 2
-2
Test Case: 3
-2
Test Case: 4
15
Test Case: 5
5
```

2. Maximum Product Subarray

Given an integer array, the task is to find the maximum product of any subarray.

Input: arr[] = {-2, 6, -3, -10, 0, 2}

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180

Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

```
  GNU nano 7.2                                                    MaxPro
import java.util.*;
public class MaxProdSub{
        static int findMaxProd(int arr[],int n){
                int firstToLast=1,lastToFirst=1,maxi=Integer.MIN_VALUE;
                for(int i=0;i<n;i++){
                        if(firstToLast==0){firstToLast=1;}
                        if(lastToFirst==0){lastToFirst=1;}
                        firstToLast*=arr[i];
                        lastToFirst*=arr[n-i-1];
                        maxi=Math.max(maxi,Math.max(firstToLast,lastToFirst));
                }
                return maxi;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(findMaxProd(arr,n));
                }
        }
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

TestCases:

```
5
6
-2 6 -3 -10 0 2
5
-1 -3 -10 0 60
4
-1 -1 -1 -1
5
-1 2 -3 4 -5
2
-1 0
```

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java MaxProdSub < MaxProdSub.txt
Test Case: 1
180
Test Case: 2
60
Test Case: 3
1
Test Case: 4
120
Test Case: 5
0
```

3. Search in a sorted and rotated Array

   Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

   Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0

   Output : 4

   Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

   Output : -1

   Input : arr[] = {50, 10, 20, 30, 40}, key = 10

   Output : 1

```
 GNU nano 7.2
import java.util.*;
public class FindInRotSort{
        static int findElemInRotSort(int arr[],int low,int high,int num){
                while(low<=high){
                        int mid=(low+high)/1;
                        if(arr[mid]==num){return mid;}
                        if(arr[low]<=arr[mid]){
                                if(arr[low]<=num && num<=arr[mid]){
                                        high=mid-1;
                                }
                                else{
                                        low=mid+1;
                                }
                        }
                        else if(arr[mid]<=arr[high]){
                                if(arr[mid]<=num && num<=arr[high]){
                                        low=mid+1;
                                }
                                else{
                                        high=mid-1;
                                }
                        }
                }
                return -1;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int num=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(findElemInRotSort(arr,0,n-1,num));
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(logn)

TestCases:

4)

Given n non-negative integers $a_1, a_2, \ldots, a_n$ where each represents a point at coordinate (i, a_i). ' n ' vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

Input: arr = [1, 5, 4, 3]

Output: 6

Explanation:

5 and 3 are distance 2 apart. So the size of the base = 2.

Height of container = min(5, 3) = 3. So total area = 3 * 2 = 6

Input: arr = [3, 1, 2, 4, 5]

Output: 12

Explanation:

5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12

```
 GNU nano 7.2                                                                W
import java.util.*;
public class WaterContainer{
        static int containerWithMostWater(int arr[],int n){
                int left=0,right=n-1;
                int maxi=0;
                while(left<right){
                        maxi=Math.max(maxi,(right-left)*Math.min(arr[left],arr[right]));
                        if(arr[right]>=arr[left]){
                                left++;
                        }
                        else{
                                right--;
                        }
                }
                return maxi;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(containerWithMostWater(arr,n));
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(n)

TestCases:

```
5
4
1 5 4 3
5
3 1 2 4 5
5
1 3 2 4 5
7
1 2 3 4 5 6 7
4
4 3 2 1
```

```
Test Case: 1
6
Test Case: 2
12
Test Case: 3
9
Test Case: 4
12
Test Case: 5
4
```

5. Find the Factorial of a large number
   Input: 100
   Output:
   933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

   Input: 50
   Output: 30414093201713378043612608166064768844377641568960512000000000000

```
  GNU nano 7.2
import java.util.*;
import java.math.BigInteger;
public class bigFact{
        static String fact(int n){
                BigInteger res=new BigInteger("1");
                for(int i=1;i<=n;i++){
                        res=res.multiply(new BigInteger(i+""));
                }
                return res.toString();
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(fact(n));
                }
        }
}
```

SpaceComplexity:
TimeComplexity:

TestCases:

```
3
100
50
200
```

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java bigFact < bigFact.txt
Test Case: 1
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565
182862536979208272237582511852109168640000000000000000000000000
Test Case: 2
304140932017133780436126081660647688443776415689605120000000000000
Test Case: 3
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984
623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445
621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062
351700858796178922222789623703897374720000000000000000000000000000000000000000000000000000000000
```

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.
   Input: arr[] = {3, 0, 1, 0, 4, 0, 2}
   Output: 10
   Explanation: The expected rainwater to be trapped is shown in the above image.

   Input: arr[] = {3, 0, 2, 0, 4}
   Output: 7
   Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

   Input: arr[] = {1, 2, 3, 4}
   Output: 0
   Explanation : We cannot trap water as there is no height bound on both sides

```java
import java.util.*;
public class TrapRain{
        static int Cap(int arr[],int n){
                int lmax[]=new int[n],rmax[]=new int[n];
                int maxi1=lmax[0]=arr[0];
                int maxi2=rmax[n-1]=arr[n-1];
                for(int i=1;i<n-1;i++){
                        maxi1=Math.max(maxi1,arr[i]);
                        maxi2=Math.max(maxi2,arr[n-i-1]);
                        lmax[i]=maxi1;
                        rmax[n-i-1]=maxi2;
                }
                int ans=0;
                for(int i=1;i<n-1;i++){
                        ans+=Math.min(lmax[i],rmax[i])-arr[i];
                }
                return ans;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(Cap(arr,n));
                }
        }
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

TestCases:

```
4
7
3 0 1 0 4 0 2
5
3 0 2 0 4
4
1 2 3 4
4
10 9 0 5
```

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java TrapRain < TrapRain.txt
Test Case: 1
10
Test Case: 2
7
Test Case: 3
0
Test Case: 4
5
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$
```

7. Chocolate Distribution Problem

Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

Each student gets exactly one packet.
The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.
Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3
Output: 2
Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5
Output: 7
Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

GNU nano 7.2

```java
import java.util.*;
public class Chocolate{
        static int distribute(ArrayList<Integer> arr,int n,int m){
                arr.sort(null);
                int ans=arr.get(m-1)-arr.get(0);
                for(int i=0;i<n;i++){
                        if(i+m-1==n){break;}
                        ans=Math.min(arr.get(i+m-1)-arr.get(i),ans);
                }
                return ans;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        ArrayList<Integer> arr=new ArrayList<>();
                        for(int j=0;j<n;j++){
                                arr.add(sc.nextInt());
                        }
                        int m=sc.nextInt();
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(distribute(arr,n,m));
                }
        }
}
```

SpaceComplexity:O(1)

TimeComplexity:O(nlogn+n)

```
4
7
7 3 2 4 9 12 56
3
7
7 3 2 4 9 12 56
5
6
1 2 3 4 5 6
4
5
1 2 15 14 40
3
```

TestCases:

```
Test Case: 1
2
Test Case: 2
7
Test Case: 3
3
Test Case: 4
13
```

8. Merge Overlapping Intervals

   Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

   Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]

   Output: [[1, 4], [6, 8], [9, 10]]

   Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

```
 GNU nano 7.2                                                  MergeIntervals.java *
import java.util.*;
public class MergeIntervals{
    static ArrayList<ArrayList<Integer>> mergeOlapping(ArrayList<ArrayList<Integer>> arr,int n){
        arr.sort((a, b) -> a.get(0) - b.get(0));
        ArrayList<ArrayList<Integer>> merged=new ArrayList<>();
        int msize=0;
        for(int i=0;i<n;i++){
            int start=arr.get(i).get(0);
            int end=arr.get(i).get(1);
            if(msize-1>=0 && end<=merged.get(msize-1).get(1)){
                continue;
            }
            for(int j=i+1;j<n;j++){
                if(arr.get(j).get(0)<=end){
                    end=Math.max(end,arr.get(j).get(1));
                }
                else{
                    break;
                }
            }
            ArrayList<Integer> temp=new ArrayList<>();
            temp.add(start);
            temp.add(end);
            merged.add(temp);
            msize++;
        }
        return merged;
    }
    public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
```

```
                        ArrayList<ArrayList<Integer>> arr=new ArrayList<>();
            //ArrayList<Integer> temp=new ArrayList<>();
                        for(int j=0;j<n;j++){
                ArrayList<Integer> temp=new ArrayList<>();
                temp.add(sc.nextInt());
                temp.add(sc.nextInt());
                arr.add(temp);
                        }
                        System.out.println("Test Case: "+(i+1));
            System.out.println(mergeOlapping(arr, n));
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(nlogn+n)

TestCases:

```
4
4
1 3 2 4 6 8 9 10
4
7 8 1 5 2 4 4 6
3
1 2 1 3 1 4
2
1 2 3 4
```

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java MergeIntervals < MergeIntervals.txt
Test Case: 1
[[1, 4], [6, 8], [9, 10]]
Test Case: 2
[[1, 6], [7, 8]]
Test Case: 3
[[1, 4]]
Test Case: 4
[[1, 2], [3, 4]]
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$
```

9. A Boolean Matrix Question

   Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

   Input: {{1, 0},
   {0, 0}}
   Output: {{1, 1}
   {1, 0}}
   Input: {{0, 0, 0},
   {0, 0, 1}}
   Output: {{0, 0, 1},
   {1, 1, 1}}

```
  GNU nano 7.2
import java.util.Scanner;

public class BoolMat {
    static void changeBoolMath(int arr[][],int n,int m){
        int row[]=new int[n];
        int col[]=new int[m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(arr[i][j]==1){
                    row[i]=1;
                    col[j]=1;
                }
            }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(row[i]==1 || col[j]==1){
                    arr[i][j]=1;
                }
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }

    }
```

SpaceComplexity:O(n)
TimeComplexity:O(n*m)

TestCases:

```
3
2
2
1 0
0 0
2
3
0 0 0
0 0 1
3
4
1 0 0 1
0 0 1 0
0 0 0 0
```

```
shriramjayanth@GoingMerry:
shriramjayanth@GoingMerry:
shriramjayanth@GoingMerry:
Test Case: 1
1 1
1 0
Test Case: 2
0 0 1
1 1 1
Test Case: 3
1 1 1 1
1 1 1 1
1 0 1 1
shriramjayanth@GoingMerry:
```

10. Print a given matrix in spiral form

    Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

    Input: matrix = {{1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12},
    {13, 14, 15, 16 }}

    Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

    Input: matrix = { {1, 2, 3, 4, 5, 6},
    {7, 8, 9, 10, 11, 12},
    {13, 14, 15, 16, 17, 18}}

    Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

    Explanation: The output is matrix in spiral format.

```java
import java.util.*;
public class MatSpiral {
    static void Spiral(int arr[][],int n,int m){
        int left=0,right=m-1,top=0,bottom=n-1;
        while(top<=bottom && left<=right){
            for(int i=left;i<=right;i++){
                System.out.print(arr[top][i]+" ");
            }
            top++;
            for(int i=top;i<=bottom;i++){
                System.out.print(arr[i][right]+" ");
            }
            right--;
            if(top<=bottom){
                for(int i=right;i>=left;i--){
                    System.out.print(arr[bottom][i]+" ");
                }
                bottom--;
            }
            if(left<=right){
                for(int i=bottom;i>=top;i--){
                    System.out.print(arr[i][left]+" ");
                }
                left++;
            }
        }
        System.out.println();
    }
```

```
    public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
            int m=sc.nextInt();
                        int[][] arr=new int[n][m];
                        for(int j=0;j<n;j++){
                                for(int k=0;k<m;k++){
                    int num=sc.nextInt();
                    arr[j][k]=num;
                }
                        }
                        System.out.println("Test Case: "+(i+1));
            Spiral(arr, n, m);
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(n*m)

TestCases:

```
2
4
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
3
6
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
```

```
Test Case: 1
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Test Case: 2
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

13. Check if given Parentheses expression is balanced or not
Given a string str of length N, consisting of '(' and ')' only, the task is to check whether it is balanced or not.

```
  GNU nano 7.2                                                    Ba
import java.util.*;
public class BalancedParanthesis{
        static String isBalanced(String s,int n){
                if(n%2!=0){return "Not Balanced";}
                int ans=0;
                for(int i=0;i<n;i++){
                        if(s.charAt(i)=='('){ans++;}
                        else{ans--;}
                        if(ans<0){return "Not Balanced";}
                }
                if(ans!=0)return "Not Balanced";
                return "Balanced";
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        String s=sc.next();
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(isBalanced(s,n));
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(n)

TestCases:

```
4
5
(()))
4
(())
10
((()))()()
4
((((
```

```
Test Case: 1
Not Balanced
Test Case: 2
Balanced
Test Case: 3
Balanced
Test Case: 4
Not Balanced
```

14. Check if two Strings are Anagrams of each other

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

```
  GNU nano 7.2
import java.util.*;
public class anagrams{
        static boolean isAnagram(String s1,String s2){
                if(s1.length()!=s2.length()){return false;}
                int arr1[]=new int[26];
                int arr2[]=new int[26];
                for(int i=0;i<s1.length();i++){
                        arr1[s1.charAt(i)-'a']++;
                        arr2[s2.charAt(i)-'a']++;
                }
                for(int i=0;i<26;i++){
                        if(arr1[i]!=arr2[i]){return false;}
                }
                return true;
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        String a=sc.next();
                        String b=sc.next();
                        System.out.println("Test Case: "+(i+1));
                        System.out.println(isAnagram(a,b));
                }
        }
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

TestCases:

```
3
geeks
kseeg
allergy
allergic
g
g
```

```
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ nano anagrams.txt
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$ java anagrams < an
Test Case: 1
true
Test Case: 2
false
Test Case: 3
true
shriramjayanth@GoingMerry:~/Desktop/CodingProbs$
```

15. Longest Palindromic Substring

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

Input: str = “Geeks”

Output: “ee”

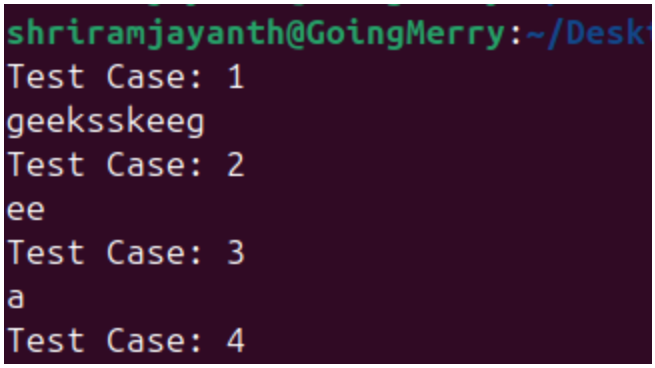
```java
import java.util.*;
public class Palindrome {
    static String LongPalSubStr(String s,int n){
        if(n==0){
            return " ";
        }
        int arr[][]=new int[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                arr[i][j]=-1;
            }
        }
        int start=0,count=1,maxi=1;
        for(int i=1;i<=n;i++){
            for(int j=0;j+i-1<n;j++){
                if(i==1){
                    arr[j][j]=1;
                }
                else if(i==2){
                    if(s.charAt(j)==s.charAt(j+1)){
                        arr[j][j+1]=2;
                        start=j;
                        count=2;
                    }
                }
                else{
                    if(s.charAt(j)==s.charAt(j+i-1) && arr[j+1][j+i-2]!=-1){
                        arr[j][j+i-1]=arr[j+1][j+i-2]+2;
                        if(arr[j][j+i-1]>maxi){
                            maxi=arr[j][j+i-1];
                            start=j;
                            count=i;
                        }
                    }
                }
            }
        }
        String ans="";
        for(int i=start;i<start+count;i++){
            ans+=s.charAt(i);
        }
```

```
        for(int i=start;i<start+count;i++){
            ans+=s.charAt(i);
        }
        return ans;
    }
    public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        String s = n == 0 ? "" : sc.next();
                        System.out.println("Test Case: "+(i+1));
            //System.out.print(s);
            //System.out.println(n);
                        System.out.println(LongPalSubStr(s,n));
                }
        }
}
```

SpaceComplexity:O(n2)**
TimeComplexity:O(n2)**

TestCases:

```
4
16
forgeeksskeegfor
5
geeks
3
abc
0
```


shriramjayanth@GoingMerry:~/Desk
Test Case: 1
geeksskeeg
Test Case: 2
ee
Test Case: 3
a
Test Case: 4

16. Longest Common Prefix using Sorting

    Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there's no prefix common in all the strings, return "-1".

    Input: arr[] = ["geeksforgeeks", "geeks", "geek", "geezer"]

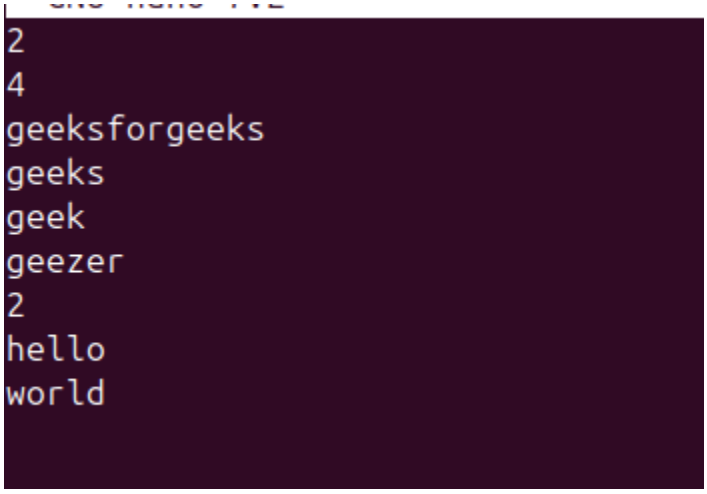
    Output: gee

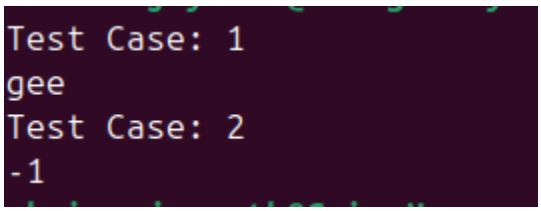
    Explanation: "gee" is the longest common prefix in all the given strings.

```
  GNU nano 7.2
import java.util.*;
public class LongComPre{
        static void Prefix(String arr[],int n,int m){
                String ans="";
        /**     System.out.println(n);
                for(int i=0;i<n;i++){
                        System.out.println(arr[i]);
                }**/
                for(int i=0;i<m;i++){
                        boolean flag=true;
                        for(int j=0;j<n-1;j++){
                                if(arr[j].charAt(i)!=arr[j+1].charAt(i)){
                                        flag=false;
                                        break;
                                }
                        }
                        if(flag==true){ans+=arr[0].charAt(i);}
                        else{break;}
                }
                if(ans.length()==0){
                        System.out.println(-1);
                }
                else{
                        System.out.println(ans);
                }
        }
        public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        String arr[]=new String[n];
                        int size=Integer.MAX_VALUE;
                        for(int j=0;j<n;j++){
                                arr[j]=sc.next();
                                //System.out.println(arr[j]);
                                size=Math.min(size,arr[j].length());
                        }
                        System.out.println("Test Case: "+(i+1));
                        Prefix(arr,n,size);
                }
        }
}
```

SpaceComplexity:O(1)
TimeComplexity:O(n*(len of smallest word))

TestCases:





17. Delete middle element of a stack

    Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

    Input : Stack[] = [1, 2, 3, 4, 5]

    Output : Stack[] = [1, 2, 4, 5]

```
import java.util.*;
public class StackMidRem {
    static void RemoveMid(Stack<Integer> st,int n,int curr){
        if(st.empty() || curr==n){
            return;
        }
        int x=st.peek();
        st.pop();
        RemoveMid(st,n,curr+1);
        if(curr!=(n/2)){
            st.push(x);
        }
    }
    public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        Stack<Integer> st=new Stack<>();
                        for(int j=0;j<n;j++){
                                st.push(sc.nextInt());
                        }
                        System.out.println("Test Case: "+(i+1));
                    RemoveMid(st,n,0);
            System.out.println(st);

                }
        }
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

TestCases:

```
2
5
1 2 3 4 5
6
1 2 3 4 5 6
```

```
Test Case: 1
[1, 2, 4, 5]
Test Case: 2
[1, 2, 4, 5, 6]
```

18. Next Greater Element (NGE) for every element in given Array
    Given an array, print the Next Greater Element (NGE) for every element.

    Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.
    Input: arr[] = [4 , 5 , 2 , 25]
    Output: 4 –> 5
    5 –> 25
    2 –> 25
    25 –> -1
    Explanation: Except 25 every element has an element greater than them present on the right side

```java
import java.util.*;
public class Nge {
    static int[] NextGElement(int arr[],int n){
        Stack<Integer> st=new Stack<>();
        int ans[]=new int[n];
        ans[n-1]=-1;
        st.push(arr[n-1]);
        for(int i=n-2;i>=0;i--){
            while(!st.empty() && st.peek()<=arr[i]){
                st.pop();
            }
            if(st.empty()){
                ans[i]=-1;
            }
            else{
                ans[i]=st.peek();
            }
            st.push(arr[i]);
        }
        return ans;
    }
    public static void main(String args[]){
                int tt;
                Scanner sc=new Scanner(System.in);
                tt=sc.nextInt();
                for(int i=0;i<tt;i++){
                        int n=sc.nextInt();
                        int arr[]=new int[n];
                        for(int j=0;j<n;j++){
                                arr[j]=sc.nextInt();
                        }
            System.out.println("Test Case: "+(i+1));
                        int ans[]=NextGElement(arr,n);
            for(int j=0;j<n;j++){
                System.out.print(ans[j]+" ");
            }
            System.out.println();

                }
        }
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

TestCases:

```
2
4
4 5 2 25
4
13 7 6 12
```

```
shriramjayanth@GoingMerry:~/Deskt
Test Case: 1
5 25 25 -1
Test Case: 2
-1 12 12 -1
shriramjayanth@GoingMerry:~/Deskt
```

19. Print Right View of a Binary Tree
    Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

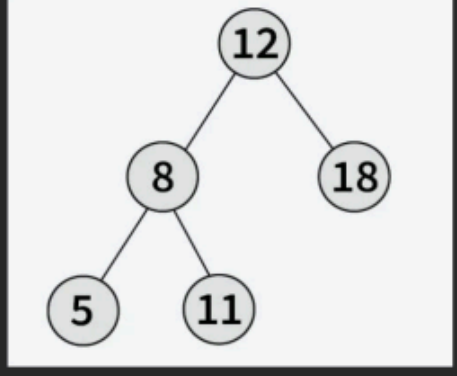
```java
import java.util.*;
class Node{
    int data;
    Node left;
    Node right;
    Node(int data){
        this.data=data;
        left=right=null;
    }
}

public class RightSideBT {
    static void rightView(Node st,ArrayList<Integer> arr,int level){
        if(st==null){
            return;
        }
        if(level==arr.size()){
            arr.add(st.data);
        }
        rightView(st.right, arr, level+1);
        rightView(st.left, arr, level+1);
    }
```

SpaceComplexity:O(n)
TimeComplexity:O(n)

20. Maximum Depth or Height of Binary Tree

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

Example 1: The height of the below binary tree is 3.




```
int findHeight(binTree node){
        if(node==null){return 1;}
        int leftHeight=findHeight(node.left);
        int rightHeight=findHeight(node.right);
        return 1+Math.max(leftHeight,rightHeight);
}
```

SpaceComplexity:O(n)
TimeComplexity:O(n)