Q1. Take as input S, a string. Write a function that does basic string compression. Print the value returned. E.g. for input "aaabbccds" print out a3b2c2d1s1.

Input Format

A single String S

Constraints

1 < = length of String < = 1000

Output Format

The compressed String.

Sample Input

```
aaabbccds
```

Sample Output

```
a3b2c2d1s1
```

Explanation

In the given sample test case 'a' is repeated 3 times consecutively, 'b' is repeated twice, 'c' is repeated twice and 'd and 's' occurred only once.

SOLUTION:

```
import java.util.*;

public class Main {

    static String compress(String s) {
        if (s.length() == 0) {
            return "";
        }

        char ch = s.charAt(0);
        int i = 1;
        while (i < s.length() && s.charAt(i) == ch) {
            i++;
        }

        String ros = s.substring(i);
        ros = compress(ros);

        String charCount = i + "";
        return ch + charCount + ros;
    }
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String s = sc.next();

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

Q2. Take as input S, a string. Write a function that toggles the case of all characters in the string. Print the value returned.

Input Format

String

Constraints

Length of string should be between 1 to 1000.

Output Format

String

Sample Input

```
abC
```

Sample Output

```
ABc
```

Explanation

Toggle Case means to change UpperCase character to LowerCase character and vice-versa.

SOLUTION:

```
public static String toggleCase(String str){

      String ans = "";

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

          char ch = str.charAt(i);

          if(ch >= 'a' && ch <= 'z'){
              ans += (char)(ch - 32); //Lower to Upper
          }else if(ch >= 'A' && ch <= 'Z'){
              ans += (char)(ch + 32); //Upper to Lower
          }
      }

      return ans;
  }
```

Q3. John and Mike are both trying to book few tickets available for a movie show. Write a Program using Multithreading to book tickets ensuring required tickets can only be booked if available tickets are more for each of them. **(Creating, Evaluating)**

Shown below is the class having main method. Complete rest of the code.

Prewritten code:

```
import java.util.Scanner;
public class TicketBooking {
    public static void main(String[] args) {
        Scanner tk = new Scanner(System.in);
        int availTickets = tk.nextInt();
        int reqJohn = tk.nextInt();
        int reqMike = tk.nextInt();
        AvailableTicket avlTic = new AvailableTicket(availTickets,reqJohn,reqMike);
        Thread t = new Thread(avlTic);
        Thread tt = new Thread(avlTic);
        t.setName("John");
        tt.setName("Mike");
        t.setPriority(10);
        t.start();
        tt.start();
    }
}
```

Input format:

First line of the input contains available tickets. Second line contains tickets required by John and third line contains tickets required by Mike.

Output format:

Each line of the output contains *currentThread* name (either John or Mike) followed by either ticket booked or not. If ticket is booked then after *currentThread* print ": tickets booked: " and if available tickets is less than required by either of them then print ": not booked.". Print the exact message considering the spaces.

Sample Input:

5

3

2

Sample Output:

John: tickets booked: 3

Mike: tickets booked: 2

Sample Input:

6

4

3

Sample Output:

John: tickets booked: 4

Mike: not booked

SOLUTION:

```
class AvailableTicket extends Thread {
    int available;
    int wantedMike;
    int wantedJohn;
    public AvailableTicket(int avail,int reqJohn,int reqMike) {
        available=avail;
        wantedJohn = reqJohn;
        wantedMike= reqMike;
    }
    public void run() {
        synchronized (this) {
            int wanted;
           String threadName= Thread.currentThread().getName();
           if(threadName=="John")
               wanted=wantedJohn;
           else
```

```
            wanted=wantedMike;
        if (available >= wanted) {
            System.out.print(Thread.currentThread().getName());
            System.out.println(": tickets booked: " + wanted);
            available = available - wanted;
        } else {
            System.out.println(Thread.currentThread().getName() + ": not booked " );
        }
      }
    }
}
```

Q4. Write a program in Java to implement an integer array and perform following operations in form of functions one after another in same sequence as mentioned:

(Applying, Evaluating)

1. Create an integer array having length of five (05) elements.
2. Input all five elements one after another.
3. Find maximum element from the input array.
4. Find minimum element from the input array.
5. Find Subtraction of all elements of the input array consecutively. Subtract first element from second, second element from third and so on. Subtraction starts from index 0 to index 4.
   a. Raise exception “Subtract is greater than equal to Zero”, if subtraction result is in positive or zero.
   b. Raise exception “Subtract is less than Zero”, if subtraction result is zero.

Input format:
Each line of the input contains array of five integers separated with a space.

Constraints:
Entered elements should be greater than 0 and lesser than 10
(1>= Arr[i] >= 9)

Output format:
Each line of the output contains the result
1. Maximum integer of input array,
2. Minimum integer of input array,
3. Subtract all integers in input array as discussed above,
   a. If subtraction result is in zero or positive, then raise exception “Subtract is greater than equal to Zero”, and
   b. If subtraction result is in negative, then raise exception “Subtract is less than Zero”,

Sample Input:
1 2 3 4 5

Sample Output:

```
5
1
-13
java.lang.Exception: Subtract is less than Zero
```

Explanation:

1. Code should able to identify maximum and minimum elements of input array as 5 and 1 shown in above example.
2. Code should able to find subtraction and also able to raise exception as discussed above. For example: If array elements are 1 2 3 4 5 then

    1 - 2 = - 1
    -1 - 3 = - 4
    -4 - 4 = - 8
    -8 - 5 = - 13

Code:

//Prewritten Code

```
import java.util.Scanner;
public class Main
{
public static final MyArray myarr = new MyArray();
public static void main(String[] args)
{
myarr.input();
myarr.max();
myarr.min();
try
{
                myarr.subfn();
          }
        catch (Exception e)
          {
            System.out.println(e);
   }

}
}
```

SOLUTION:

```
class MyArray
{
Scanner sc = new Scanner(System.in);
```

```
public static final int[] Arr = new int[5];


            public void input()
{
for(int i=0; i<5; i++)
{
Arr[i]=sc.nextInt();
}
}

            public void max()
{
int max = 0;
for (int i =0 ;i<5;i++)
{
if (Arr[i] > max)
{
max = Arr[i];
}
}
System.out.println(max);
}
public void min()
{
int min = 10;
for (int i =0 ;i<5;i++)
{
if (Arr[i] < min)
{
min = Arr[i];
}
}
System.out.println(min);
}
public void subfn() throws Exception
{
int sub = Arr[0];
for (int i =1 ;i<5;i++)
{
sub = sub - Arr[i];
}
System.out.println(sub);
if (sub<0)
{
throw new Exception("Subtract is less than Zero");
}
else
{
throw new Exception("Subtract is greater than equal to Zero");
}
}
}
```

Q5. Take as input S, a string. Write a function that does basic string compression. Print the value returned. E.g. for input “aaabbccds” print out a3b2c2d1s1.

Input Format

A single String S

Constraints

1 < = length of String < = 1000

Output Format

The compressed String.

Sample Input

```
aaabbccds
```

Sample Output

```
a3b2c2d1s1
```

Explanation

In the given sample test case 'a' is repeated 3 times consecutively, 'b' is repeated twice, 'c' is repeated twice and 'd and 's' occurred only once.

SOLUTION:

```
import java.util.*;

public class Main {

    static String compress(String s) {
        if (s.length() == 0) {
            return "";
        }

        char ch = s.charAt(0);
        int i = 1;
        while (i < s.length() && s.charAt(i) == ch) {
            i++;
        }

        String ros = s.substring(i);
        ros = compress(ros);

        String charCount = i + "";
        return ch + charCount + ros;
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String s = sc.next();

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

Q6 Given a linked list with n nodes. Find the kth element from last without computing the length of the linked list.

Input Format

First line contains space separated integers representing the node values of the linked list. The list ends when the input comes as '-1'. The next line contains a single integer k.

Constraints

n < 10^5

Output Format

Output a single line containing the node value at the kth element from last.

Sample Input

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

Sample Output

```
4
```

Explanation

The linked list is 1 2 3 4 5 6. -1 is not included in the list. So the third element from the last is 4

Q7. Given an array of patterns containing only I's and D's. I for increasing and D for decreasing. Devise an algorithm to print the minimum number following that pattern. Digits from 1-9 and digits can't repeat.

Input Format

The First Line contains an Integer N, size of the array. Next Line contains N Strings separated by space.

Constraints

1 ≤ T ≤ 100 1 ≤ Length of String ≤ 8

Output Format

Print the minimum number for each String separated by a new Line.

Sample Input

```
4
D I DD II
```

Sample Output

```
21
12
321
123
```

Explanation

For the Given sample case, For a Pattern of 'D' print a decreasing sequence which is 2 1.

Q8. Deepak has a limited amount of money that he can spend on his girlfriend. So he decides to buy two roses for her. Since roses are of varying sizes, their prices are different. Deepak wishes to completely spend that fixed amount of money on buying roses for her.

As he wishes to spend all the money, he should choose a pair of roses whose prices when summed up are equal to the money that he has.

Help Deepak choose such a pair of roses for his girlfriend.

NOTE: If there are multiple solutions print the solution that minimizes the difference between the prices i and j. After each test case, you must print a blank line.

Input Format

The first line indicates the number of test cases T.

Then, in the next line, the number of available roses, N is given.

The next line will have N integers, representing the price of each rose, a rose that costs less than 1000001.

Then there is another line with an integer M, representing how much money Deepak has.

There is a blank line after each test case.

Constraints

1≤ T ≤100

2 ≤ N ≤ 10000

Price[i]<1000001

Output Format

For each test case, you must print the message: ‘Deepak should buy roses whose prices are i and j.’, where i and j are the prices of the roses whose sum is equal do M and i ≤ j. You can consider that it is always possible to find a solution. If there are multiple solutions print the solution that minimizes the difference between the prices i and j.

Sample Input

```
2
2
40 40
80

5
10 2 6 8 4
10
```

Sample Output

```
Deepak should buy roses whose prices are 40 and 40.
Deepak should buy roses whose prices are 4 and 6.
```

Explanation

Find two such kinds of price of roses which has sum up to equal to Deepak's Money.

SOLUTION:

```
public static void roses(int[] arr, int target){

        Arrays.sort(arr); // sort the Array

        int fl = 0;  //To store the leftmost index
       int fr = 0;    //To store the rightmost index

        int left = 0;
        int right = arr.length - 1;

        while (left < right) {

            int sum = arr[left] + arr[right];

            if (sum > target) {
                right--;
            } else if (sum < target) {
                left++;
            } else {
                fl = left;
                fr = right;

                left++;
                right--;
            }

        }

        System.out.println("Deepak should buy roses whose prices are " + fl
+ " and " + fr + ".");

    }
```

Q9. The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate span of stock's price for all N days. You are given an array of length N, where ith element of array denotes the price of a stock on ith. Find the span of stock's price on ith day, for every 1<=i<=N.

A span of a stock's price on a given day, i, is the maximum number of consecutive days before the (i+1)th day, for which stock's price on these days is less than or equal to that on the ith day.

Input Format

First line contains integer N denoting size of the array.

Next line contains N space separated integers denoting the elements of the array.

Constraints

1 <= N <= 10^6

Output Format

Display the array containing stock span values.

Sample Input

```
5
30
35
40
38
35
```

Sample Output

```
1 2 3 1 1 END
```

Explanation

For the given case
for day1 stock span =1

for day2 stock span =2 (as 35>30 so both days are included in it)
for day3 stock span =3 (as 40>35 so 2+1=3)
for day4 stock span =1 (as 38<40 so only that day is included)
for day5 stock span =1 (as 35<38 so only that day is included)
hence output is 1 2 3 1 1 END

SOLUTION:

```
public static int[] StockSpanUsingStacks(int[] prices, Stack<Integer> stack)
throws Exception {

        // span array stores the value of stock span for each day
        int[] span = new int[prices.length];

         // pushing the index of first day
        stack.push(0);

        // span for 1st day will always be 1
        span[0] = 1;

        // Iterating over the prices array
        for (int i = 1; i < prices.length; i++) {

           // poping days from stack when price of the i th day is greater
than the price of day which is on top of stack
            while (stack.size() != 0 && prices[i] > prices[stack.top()]) {
                stack.pop();
            }

            if (stack.size() == 0) {

             // then all the previous prices are smaller than the price on
the  ith day
                span[i] = i + 1;
            } else {

              // the previous highest price will be on the top of the stack
                span[i] = i - stack.top();
            }

              // pushing the index of the price array which is the number of
day in stack
            stack.push(i);
        }
        return span;
        }
```

Q10. Take as input S, a string. Write a function that does basic string compression. Print the value returned. E.g. for input “aaabbccds” print out a3b2c2d1s1.

Input Format

A single String S

Constraints

1 < = length of String < = 1000

Output Format

The compressed String.

Sample Input

```
aaabbccds
```

Sample Output

```
a3b2c2d1s1
```

Explanation

In the given sample test case 'a' is repeated 3 times consecutively, 'b' is repeated twice, 'c' is repeated twice and 'd and 's' occurred only once.

SOLUTION:

```
import java.util.*;

public class Main {

    static String compress(String s) {
        if (s.length() == 0) {
            return "";
        }

        char ch = s.charAt(0);
        int i = 1;
        while (i < s.length() && s.charAt(i) == ch) {
            i++;
        }

        String ros = s.substring(i);
        ros = compress(ros);

        String charCount = i + "";
        return ch + charCount + ros;
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String s = sc.next();

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

Take an input N, the size of array. Take N more inputs and store that in an array. Write a function which returns the maximum value in the array. Print the value returned.

1.It reads a number N.
2.Take Another N numbers as input and store them in an Array.
3.calculate the max value in the array and return that value.

Input Format
First line contains integer n as size of array. Next n lines contains a single integer as element of array.

Constraints
N cannot be Negative. Range of Numbers can be between -1000000000 to 1000000000

Output Format
Print the required output.

Sample Input
4
2
8
6
4
Sample Output
8
Explanation
Arrays= {2, 8, 6, 4} => Max value = 8 .

SOLUTION:

```
import java.util.*;
public class Main {
public static void main(String args[]) {
Scanner sc=new Scanner(System.in);
int n=sc.nextInt();
int arr[]=new int[n];
for(int i=0;i<n;i++){
arr[i]=sc.nextInt();
}
int max=Integer.MIN_VALUE;
for(int i:arr){
max=Math.max(max,i);
}
System.out.println(max);
}
}
```

Q11. Write a program in Java to implement an integer array and perform following operations in form of functions one after another in same sequence as mentioned:

(Applying, Evaluating)

1. Create an integer array having length of five (05) elements.
2. Input all five elements one after another.
3. Find maximum element from the input array.
4. Find minimum element from the input array.
5. Find Subtraction of all elements of the input array consecutively. Subtract first element from second, second element from third and so on. Subtraction starts from index 0 to index 4.
   a. Raise exception "Subtract is greater than equal to Zero", if subtraction result is in positive or zero.
   b. Raise exception "Subtract is less than Zero", if subtraction result is zero.

Input format:
Each line of the input contains array of five integers separated with a space.

Constraints:
Entered elements should be greater than 0 and lesser than 10
(1>= Arr[i] >= 9)

Output format:
Each line of the output contains the result
1. Maximum integer of input array,
2. Minimum integer of input array,
3. Subtract all integers in input array as discussed above,
   a. If subtraction result is in zero or positive, then raise exception "Subtract is greater than equal to Zero", and
   b. If subtraction result is in negative, then raise exception "Subtract is less than Zero",

Sample Input:
```
1 2 3 4 5
```

Sample Output:
```
5
1
-13
```

```
java.lang.Exception: Subtract is less than Zero
```

Explanation:

1. Code should able to identify maximum and minimum elements of input array as 5 and 1 shown in above example.
2. Code should able to find subtraction and also able to raise exception as discussed above. For example: If array elements are 1 2 3 4 5 then

   1 - 2 = - 1
   -1 - 3 = - 4
   -4 - 4 = - 8
   -8 - 5 = - 13

Code:

// Prewritten Code

```
import java.util.Scanner;
public class Main
{
public static final MyArray myarr = new MyArray();
public static void main(String[] args)
{
myarr.input();
myarr.max();
myarr.min();
try
{
                  myarr.subfn();
         }
        catch (Exception e)
          {
           System.out.println(e);
  }

}
}
```

SOLUTION:

```
class MyArray
{
Scanner sc = new Scanner(System.in);
public static final int[] Arr = new int[5];


        public void input()
```

```
{
for(int i=0; i<5; i++)
{
Arr[i]=sc.nextInt();
}
}

        public void max()
{
int max = 0;
for (int i =0 ;i<5;i++)
{
if (Arr[i] > max)
{
max = Arr[i];
}
}
System.out.println(max);
}
public void min()
{
int min = 10;
for (int i =0 ;i<5;i++)
{
if (Arr[i] < min)
{
min = Arr[i];
}
}
System.out.println(min);
}
public void subfn() throws Exception
{
int sub = Arr[0];
for (int i =1 ;i<5;i++)
{
sub = sub - Arr[i];
}
System.out.println(sub);
if (sub<0)
{
throw new Exception("Subtract is less than Zero");
}
else
{
throw new Exception("Subtract is greater than equal to Zero");
}
}
}
```

Q 12 The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate span of stock's price for all N days. You are given an array of length N, where ith element of array denotes the price of a stock on ith. Find the span of stock's price on ith day, for every 1<=i<=N.
A span of a stock's price on a given day, i, is the maximum number of consecutive days before the (i+1)th day, for which stock's price on these days is less than or equal to that on the ith day.

Input Format

First line contains integer N denoting size of the array.
Next line contains N space separated integers denoting the elements of the array.

Constraints

1 <= N <= 10^6

Output Format

Display the array containing stock span values.

Sample Input

```
5
30
35
40
38
35
```

Sample Output

```
1 2 3 1 1 END
```

Explanation

For the given case
for day1 stock span =1
for day2 stock span =2 (as 35>30 so both days are included in it)
for day3 stock span =3 (as 40>35 so 2+1=3)
for day4 stock span =1 (as 38<40 so only that day is included)
for day5 stock span =1 (as 35<38 so only that day is included)
hence output is 1 2 3 1 1 END

SOLUTION:

```
import java.util.*;
public class Main {

		public static void main(String args[]) throws Exception {
				// Your Code Here
				Scanner sc = new Scanner (System.in);
				int n=sc.nextInt();
				int [] arr = new int[n];
				for(int i=0;i<n;i++) {
						arr[i]=sc.nextInt();
```

```
                }

                Stack<Integer> st = new Stack<>();
                int[] ans = new int[arr.length];
                for (int i = 0; i < arr.length; i++) {

                        while (!st.isEmpty() && arr[i] > arr[st.peek()]) {
                                st.pop();
                        }

                        if (st.isEmpty()) {
                                ans[i] = i + 1;

                        } else {
                                ans[i] = i - st.peek();
                        }

                        st.push(i);
                }
                for(int i=0;i<ans.length;i++) {
                        System.out.print(ans[i]+" ");
                }
                System.out.println("END");
        }

}
```

Q13. Take as input S, a string. Write a program that gives the count of substrings of this string which are palindromes and Print the ans.

Input Format

Single line input containing a string

Constraints

Length of string is between 1 to 1000.

Output Format

Integer output showing the number of palindromic substrings.

Sample Input

abc

Sample Output

3

Explanation

For the given sample case , the palindromic substrings of the string abc are "a","b" and "c".So, the ans is 3.

SOLUTION:

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
                        String s = sc.nextLine();
                        int count = 0;
                        for (int i = 0; i < s.length(); i++) {
                                    for (int j = i; j <= s.length(); j++) {
                                                String ans = s.substring(i, j);
                                                if (ispalindrom(ans))
                                                            count = count + 1;
                                    }
                        }
                        System.out.println(count);
            }

            private static boolean ispalindrom(String ans) {
                        int lo = 0;
                        int hi = ans.length() - 1;
                        if (ans.length() == 0) {
                                    return false;
                        } else {
                                    while (lo <= hi) {
                                                if (ans.charAt(lo) != ans.charAt(hi)) {
                                                            return false;
                                                }
                                                lo++;
                                                hi--;
                                    }
                                    return true;
                        }
            }

}
```

Q14. A Good String is a string which contains only vowels (a,e,i,o,u) . Given a string S, print a single positive integer N where N is the length of the longest substring of S that is also a Good String.

Note: The time limit for this problem is 1 second, so you need to be clever in how you compute the substrings.

Input Format

A string 'S'

Constraints

Length of string < 10^5

Output Format

A single positive integer N, where N is the length of the longest sub-string of S that is also a Good String.

Sample Input

```
cbaeicde
```

Sample Output

```
3
```

Explanation

Longest good substring is "aei"

SOLUTION:

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
     Scanner sc = new Scanner(System.in);
                    String str = sc.nextLine();
                    int i = 0;
                    int count = 0;
                    int ans = 0;
                    while (i < str.length()) {
                              char ch = str.charAt(i);
                              if (isVovel(ch)) {
                                        count++;
                              } else {
                                        ans = Math.max(ans, count);
                                        count = 0;
                              }
                              i++;
                    }
                    System.out.println(Math.max(ans, count));

          }

          private static boolean isVovel(char ch) {
                    // TODO Auto-generated method stub
                    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
                              return true;
                    return false;
          }

}
```

Q 15 You are given two integers n and k. Find the greatest integer x, such that, x^k <= n.

Input Format

First line contains number of test cases, T. Next T lines contains integers, n and k.

Constraints

1<=T<=10
1<=N<=10^15
1<=K<=10^4

Output Format

Output the integer x

Sample Input

```
2
10000 1
1000000000000000 10
```

Sample Output

```
10000
31
```

Explanation

For the first test case, for x=10000, 10000^1=10000=n

SOLUTION:

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
       // Your Code Here
       Scanner s = new Scanner(System.in);
                    int t = s.nextInt();
                    for (int i = 1; i <= t; i++) {
                              long n = s.nextLong();
                              int k = s.nextInt();
                              System.out.println(root(n, k));
                    }
          }
          public static int root(long n, int k) {
                    long ans = 0;
```

```
                long lo = 0;
                long hi = n;
                while (lo <= hi) {
                        long mid = lo + (hi - lo) / 2;
                        long check = (long) Math.pow(mid, k);
                        if (check > n) {
                                hi = mid - 1;
                        } else if (check < n) {
                                ans = mid;
                                lo = mid + 1;
                        } else {
                                ans = mid;
                                return (int) ans;
                        }

                }
                return (int) ans;
  }
}
```

Q 16 . Given an array Arr[], Treat each element of the array as the digit and whole array as the number. Implement the next permutation, which rearranges numbers into the numerically next greater permutation of numbers.

If such arrangement is not possible, it must be rearranged as the lowest possible order ie, sorted in an ascending order.

Note: The replacement must be in-place, do not allocate extra memory.

Input Format
The First Line contains the Number of test cases T.
Next Line contains an Integer N, number of digits of the number.
Next Line contains N-space separated integers which are elements of the array 'Arr'.

Constraints
1 <= T <= 100
1 <= N <= 1000
0 <= Ai <= 9

Output Format
Print the Next Permutation for each number separated by a new Line.

Sample Input
```
3
1 2 3
```

Sample Output

1 3 2

Explanation

Possible permutations for {1,2,3} are {1,2,3} , {1,3,2} , {2,1,3} , {2,3,1}, {3,1,2} and {3,2,1}. {1,3,2} is the immediate next permutation after {1,2,3}.
For the second testcase , {3,2,1} is the last configuration so we print the first permutation as its next permutation.

SOLUTION:

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
                        // TODO Auto-generated method stub
                        Scanner sc = new Scanner(System.in);
                        int n = sc.nextInt();
                        int[] arr = new int[n];
                        for (int i = 0; i < n; i++)
                                    arr[i] = sc.nextInt();
                        nextPermutation(arr);
                        for (int i = 0; i < arr.length; i++) {
                                    System.out.print(arr[i] + " ");
                        }

    }
            public static void nextPermutation(int[] arr) {

                        int p = -1;
                        for (int i = arr.length - 2; i >= 0; i--) {
                                    if (arr[i] < arr[i + 1]) {
                                                p = i;
                                                break;
                                    }

                        }
                        if (p == -1) {
                                    Reverse(arr, 0, arr.length - 1);
                                    return;
                        }
                        int q = -1;
                        for (int i = arr.length - 1; i > p; i--) {
                                    if (arr[i] > arr[p]) {
                                                q = i;
                                                break;
                                    }
                        }
                        // swap
                        int t = arr[p];
```

```
                arr[p] = arr[q];
                arr[q] = t;
                Reverse(arr, p + 1, arr.length - 1);

        }

        public static void Reverse(int[] arr, int i, int j) {

                while (i < j) {

                        int t = arr[i];
                        arr[i] = arr[j];
                        arr[j] = t;
                        i++;
                        j--;
                }
    }
}
```

Q 17 .You are given two integers n and k. Find the greatest integer x, such that, x^k <= n.

Input Format

First line contains number of test cases, T. Next T lines contains integers, n and k.

Constraints

1<=T<=10
1<=N<=10^15
1<=K<=10^4

Output Format

Output the integer x

Sample Input

```
2
10000 1
1000000000000000 10
```

Sample Output

```
10000
31
```

Explanation

For the first test case, for x=10000, 10000^1=10000=n

SOLUTION:

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner s = new Scanner(System.in);
                        int t = s.nextInt();
                        for (int i = 1; i <= t; i++) {
                                long n = s.nextLong();
                                int k = s.nextInt();
                                System.out.println(root(n, k));
                        }
                }
                public static int root(long n, int k) {
                        long ans = 0;
                        long lo = 0;
                        long hi = n;
                        while (lo <= hi) {
                                long mid = lo + (hi - lo) / 2;
                                long check = (long) Math.pow(mid, k);
                                if (check > n) {
                                        hi = mid - 1;
                                } else if (check < n) {
                                        ans = mid;
                                        lo = mid + 1;
                                } else {
                                        ans = mid;
                                        return (int) ans;
                                }

                        }
                        return (int) ans;
    }
}
```

Q 18. Given an integer array nums find the squares of each number sorted in non-decreasing order.

Input Format

First line of input contains an integer n representing the length of array n. Next line contains n array elements.

Constraints

1 <= nums.length <= 10^4
-10^4 <= nums[i] <= 10^4
nums is sorted in non-decreasing order.

Output Format

A sorted array representing squares of elements of nums array.

Sample Input
5
-4 -1 0 3 10

Sample Output
0 1 9 16 100

Explanation
After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100]

SOLUTION:

```
import java.util.*;

public class Main {
	public static void main(String args[]) {
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int[] arr = new int[n];
		for (int i = 0; i < n; i++) {
			arr[i] = sc.nextInt();
		}
		Square(arr);
		Arrays.sort(arr);
		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}

	}

	public static void Square(int arr[]) {
		int sum = 0;
		for (int i = 0; i < arr.length; i++) {
			sum = arr[i] * arr[i];
			arr[i] = sum;
		}
	}

}
```

Q 19. Given an array, print the Next Greater Element (NGE) for every element. The Next Greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

Input Format

First line of the input contains a single integer T denoting the number of testcases. First line of each testcase contains an integer N denoting the size of array. Second line of each testcase contains N space seperated integers denoting the array.

Constraints

1 <= T <= 50 1 <= N <= 10^5

Output Format

For each index, print its array element and its next greater element seperated by a comma in a new line.

Sample Input

```
4
11 13 21 3
```

Sample Output

13 21 -1 -1

Explanation

For the first testcase , the next greater element for 11 is 13 , for 13 its 21 and 21 being the largest element of the array does not have a next greater element. Hence we print -1 for 21. 3 is the last element of the array and does not have any greater element on its right. Hence we print -1 for it as well.

SOLUTION:

```
import java.util.*;

public class StockSpan {
	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int[] arr = new int[n];
		for (int i = 0; i < n; i++) {
			arr[i] = sc.nextInt();
		}
		int ans[] = new int[arr.length];

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

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

			while (!st.isEmpty() && arr[i] > arr[st.peek()]) {
				ans[st.peek()] = arr[i];
				st.pop();
			}
			st.push(i);
		}

		while (!st.isEmpty()) {
			ans[st.peek()] = -1;
			st.pop();
```

```
                    }
                    for(int a :ans) {
                              System.out.print(a+" ");
                    }
          }
}
```

Q 20. Take as input S, a string. Write a function that returns the character with maximum frequency. Print the value returned.

Input Format
String

Constraints
A string of length between 1 to 1000.

Output Format
Character

Sample Input
aaabacb
Sample Output
a
Explanation
For the given input string, a appear 4 times. Hence, it is the most frequent character.

SOLUTION:

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
       // Your Code Here
       Scanner sc = new Scanner(System.in);
                String s = sc.nextLine();
                int max = 0;
                char ch = s.charAt(0);
                for (int i = 0; i < s.length(); i++) {
                        int local_max = 0;
                        char local_ch = s.charAt(i);
                        for (int j = i; j < s.length(); j++) {
                                if (s.charAt(i) == s.charAt(j)) {
                                        local_max++;
                                }
```

```
                                        if (local_max > max) {
                                                  ch = local_ch;
                                                  max = local_max;
                                        }

                              }

                    }
                    System.out.println(ch);
          }

}
```

Q21. You are provided n numbers (both +ve and -ve). Numbers are arranged in a circular form. You need to find the maximum sum of consecutive numbers.

Input format:
For each test case, it contains an integer n which is the size of array and next line contains n space separated integers denoting the elements of the array.
Constraints:
1<=n<=1000
|Ai| <= 10000

Output format:

Print the maximum circular sum in a new line.

Sample Input:
7
8 -8 9 -9 10 -11 12

Sample Output:
22

Explanation:
Maximum Circular Sum = 22 (12 + 8 - 8 + 9 - 9 + 10)

Solution:

```
import java.util.*;

public class Main {
          public static void main(String args[]) {
                    Scanner sc = new Scanner(System.in);
                    int n = sc.nextInt();
                    int[] arr = new int[n];
                    for (int i = 0; i < arr.length; i++) {
```

```
				arr[i] = sc.nextInt();
			}
			System.out.println(maxSubarraySumCircular(arr));


	}

	public static int maxSubarraySumCircular(int[] arr) {
			int kad = kadane(arr);
			int sum = sumofarray(arr);
			int revkad = kadane(arr);
			int total = revkad + sum;
			if (total == 0) {
				return kad;
			}
			return Math.max(kad, total);

	}

	public static int kadane(int[] arr) {
			int currentsum = 0;
			int maxsum = Integer.MIN_VALUE;
			for (int i = 0; i < arr.length; i++) {
				if (currentsum < 0)
					currentsum = 0;
				currentsum += arr[i];
				maxsum = Math.max(maxsum, currentsum);
			}
			return maxsum;

	}

	public static int sumofarray(int[] arr) {
			int sum = 0;
			for (int i = 0; i < arr.length; i++) {
				sum += arr[i];
				arr[i] = -arr[i];
			}
			return sum;
	}

}
```

Q22. Take input n numbers, if their sum is more than 100 then throw an Exception that says "**Sum limit exceeded"** Otherwise print their sum.

Input Format:
First line contain size of the array.

Next line is the n integers of the array.

Output: If their sum is less than equal to 100 then print their sum otherwise throw an exception.

Sample Input 1:

```
3

100 200 300
```

Sample Output 1:

```
java.lang.Exception: Sum limit exceeded
```

Sample Input 2:

```
2
10 20
```

Sample Output 2:

```
30
```

SOLUTION:

```
import java.util.Scanner;


public class Main {


    static void getSum(int[] arr) throws Exception
    {
        int sum=0;
        for(int i=0;i<arr.length;i++)
        {
            sum+=arr[i];
        }

        if(sum<=100)
        System.out.print(sum);

        else
        throw new Exception("Sum limit exceeded");
    }

    public static void main(String[] args) {

        int n;
        Scanner sc=new Scanner(System.in);

        n=sc.nextInt();
        int[]arr=new int[n];


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

        try {
            getSum(arr);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Q23.Given an integer array print 1 if any value appears **at least twice** in the array, and print 0 if every element is distinct.

Input format:

First line contain size of the array.

Next line is the n integers of the array.

Output Format

Print 1 if any element occur at least twice and 0 if every element is distinct.

SOLUTION:

```
import java.util.*;
public class Main {

   public static boolean containsDuplicate(int[] nums)
{
       HashMap<Integer,Integer> map = new HashMap<>();
       for (int i = 0; i < nums.length; i++) {
          if (map.containsKey(nums[i])) {
             return true;
          }
          map.put(nums[i],1);
       }
       return false;
  }

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

       int [] arr=new int [n];

       for(int i=0;i<n;i++)
       arr[i]=sc.nextInt();

       if(containsDuplicate(arr))
```

```
        System.out.print("1");

        else
        System.out.print("0");



    }
}
```

Q24. Given an array of **N** integers. Write a program to check whether an arithmetic progression can be formed using all the given elements.

Sample Input 1:

4

0 12 4 8

Sample Output 1: YES

Explanation: Rearrange given array as {0, 4, 8, 12} which forms an arithmetic progression.

Sample Input 1:

4

12 40 11 20

Output: NO

SOLUTION:

```
import java.util.*;
public class Main {
static boolean checkIsAP(int arr[] ,int n)
   {

        HashSet<Integer> set=new HashSet<>();
        int min=Integer.MAX_VALUE;
        int smin=Integer.MAX_VALUE;
        for(int i=0;i<n;i++)
        {
```

```
        set.add(arr[i]);

        if(arr[i]<min)
        {
            smin=min;
            min=arr[i];
        }
        else if(arr[i]<smin)
        {
            smin=arr[i];
        }
    }

    int d=smin-min;
    for(int i=1;i<=n;i++)
    {
        int term=min+(i-1)*d;
        if(!set.contains(term))
        return false;
    }

    return true;
}

public static void main(String args[]) {


   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt();

   int []arr=new int[n];
   for(int i=0;i<n;i++)
   arr[i]=sc.nextInt();
```

```
        if(checkIsAP(arr,n))
        System.out.print("YES");
        else
        System.out.print("NO");
    }
}
```

Q25. Given two strings A and B. Find the characters that are not common in the two strings. If no such character exists return "-1".

Input: A = characters B = alphabets

Output: bclpr

Explanation: The characters 'b','c','l','p','r' are either present in A or B, but not in both.

Input: A = bbbbb B =bb

Output: -1

Solution:

```
import java.util.*;
public class Main {



    static String UncommonChars(String A, String B)
    {
        // code here
        HashSet<Character> set1 = new HashSet<>();
        HashSet<Character> set2 = new HashSet<>();
        ArrayList<Character> list = new ArrayList<>();

        String str = "";

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

```
    char ch = A.charAt(i);
    set1.add(ch);
}

for(int i = 0; i < B.length(); i++)
{
    char ch = B.charAt(i);
    set2.add(ch);
}

for(int i = 0; i < A.length(); i++)
{
    char ch = A.charAt(i);
    if(!set2.contains(ch))
    {
        if(!list.contains(ch))
            list.add(ch);
    }

}

for(int i = 0; i < B.length(); i++)
{
    char ch = B.charAt(i);
    if(!set1.contains(ch))
    {
        if(!list.contains(ch))
            list.add(ch);
    }
}

if(list.size() == 0)
    return "-1";
```

```
         Collections.sort(list);
        for(char ch : list)
        {
            str += ch;
        }

        return str;
    }

    public static void main(String args[]) {
        String str1,str2;
        Scanner sc=new Scanner(System.in);
        str1=sc.nextLine();
        str2=sc.nextLine();

        System.out.print(UncommonChars(str1,str2));
    }
}
```

Q26. Given an array **arr[]** of **n** integers. Check whether it contains a triplet that sums up to zero.

Input Format

First line contain size of the array.

Next line is the n integers of the array.

Output Format

Print 1 if the triplet with 0 exists and 0 if it doesn't exists.

Sample Input
5
0 -1 2 -3 1

Sample Output
1

Explanation
0, -1 and 1 forms a triplet with sum equal to 0.

SOLUTION:

```
import java.util.*;
public class Main
{
   static public boolean findTriplets(int arr[] , int n)
    {
        //add code here.
        boolean found=false;
        for(int i=0;i<n-1;i++){
            HashSet<Integer> hs=new HashSet<>();
            for(int j=i+1;j<n;j++){
                int x=-(arr[i]+arr[j]);
                if(hs.contains(x)){
                    return true;
                }else{
                    hs.add(arr[j]);
                }
            }
        }
        return false;
    }

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

        int [] arr=new int[n];

        for(int i=0;i<n;i++)
        arr[i]=sc.nextInt();

        if(findTriplets(arr,n))
        System.out.println("1");

        else
         System.out.println("0");


    }
}
```

Q27.Given an integer array print 1 if any value appears **at least twice** in the array, and print 0 if every element is distinct.

Input format:

First line contain size of the array.

Next line is the n integers of the array.

Output Format

Print 1 if any element occur at least twice and 0 if every element is distinct.

SOLUTION:

```
import java.util.*;
public class Main {
   public static boolean containsDuplicate(int[] nums)
{
      HashMap<Integer,Integer> map = new HashMap<>();
      for (int i = 0; i < nums.length; i++) {
         if (map.containsKey(nums[i])) {
            return true;
         }
         map.put(nums[i],1);
      }
      return false;
 }

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

      int [] arr=new int [n];

      for(int i=0;i<n;i++)
      arr[i]=sc.nextInt();

      if(containsDuplicate(arr))
      System.out.print("1");

      else
      System.out.print("0");



   }
}
```

Q 28 Given two strings str1 and str2, print the index of the first occurrence of `str2` in str1, or `-1` if str2 is not part of `str1`.

Input Format:

First line contains str1 i.e. bigger string

Second Line contains str2 i.e. smaller string

Output Format:

Print the index of first occurrence of str2 in str1 if not found then print -1

Sample **Input 1:**

sadbutsad

sad

Sample Output 1:

0

Explanation: "sad" occurs at index 0 and 6.

The first occurrence is at index 0, so we return 0.

Sample Input 2:

Hello

Bye

Sample Output 2:

-1

Explanation: “Bye” is not contained in str1

```
import java.util.Scanner;
public class Main {



    static int func(String haystack, String needle) {
        int haylength=haystack.length();
        int needlelength=needle.length();
        if(haylength<needlelength)
            return -1;
        for(int i=0;i<=haystack.length()-needle.length();i++){
            int j=0;
            while(j<needle.length() && haystack.charAt(i+j)==needle.charAt(j))
                j++;
            if(j==needle.length()){
                return i;
            }
        }
        return -1;
    }
    public static void main(String args[]) {
       String str1,str2;
       Scanner sc=new Scanner(System.in);
       str1=sc.nextLine();
       str2=sc.nextLine();

       System.out.print(func(str1,str2));
    }
}
```

Q29. Given two strings A and B. Find the characters that are not common in the two strings. If no such character exists return "-1".

Input: A = characters B = alphabets

Output: bclpr

Explanation: The characters 'b','c','l','p','r' are either present in A or B, but not in both.

Input: A = bbbbb B =bb

Output: -1

Explanation: No uncommon char exists

Solution:

```
import java.util.*;
public class Main {


    static String UncommonChars(String A, String B)
    {
        // code here
        HashSet<Character> set1 = new HashSet<>();
        HashSet<Character> set2 = new HashSet<>();
        ArrayList<Character> list = new ArrayList<>();
```

```
String str = "";

for(int i = 0; i < A.length(); i++)
{
    char ch = A.charAt(i);
    set1.add(ch);
}

for(int i = 0; i < B.length(); i++)
{
    char ch = B.charAt(i);
    set2.add(ch);
}

for(int i = 0; i < A.length(); i++)
{
    char ch = A.charAt(i);
    if(!set2.contains(ch))
    {
        if(!list.contains(ch))
            list.add(ch);
    }

}

for(int i = 0; i < B.length(); i++)
{
    char ch = B.charAt(i);
    if(!set1.contains(ch))
    {
        if(!list.contains(ch))
            list.add(ch);
    }
```

```
        }

        if(list.size() == 0)
            return "-1";

        Collections.sort(list);
       for(char ch : list)
       {
           str += ch;
       }

       return str;
    }

    public static void main(String args[]) {
       String str1,str2;
       Scanner sc=new Scanner(System.in);
       str1=sc.nextLine();
       str2=sc.nextLine();

       System.out.print(UncommonChars(str1,str2));
    }
}
```

Q30 Given an input string s, Check whether it is balanced parenthesis or not.

Balanced parentheses are a grouping of parentheses in a string such that every opening parenthesis has a matching closing parenthesis and vice versa. For example, the following strings have balanced parentheses:
"()" - contains one pair of balanced parentheses
"(()())" - contains two pairs of balanced parentheses
"(())()" - contains two pairs of balanced parentheses
"((()))" - contains one pair of balanced parentheses

If the given String is balanced parenthesis then print 1 otherwise print 0.

Input format:

Accept a string

Output format:

Print the string of the words in reverse order

Sample Input 1:
()()
Sample Output 1:
1

Sample Input 2:
()(
(Sample Output 2:
0

Code:
```
import java.util.*;
class Bal
{
   static boolean isBalanced(String str)
   {
      Stack<Character>st=new Stack<Character>();

      for(int i=0;i<str.length();i++)
      {
         if(str.charAt(i)=='(')
         st.push('(');
         else if(str.charAt(i)==')' && st.empty())
         return false;

         else
         st.pop();


      }

      return st.empty();
   }

   public static void main(String[] args)
   {

      String str;

      Scanner sc=new Scanner(System.in);
      str=sc.next();

      if(isBalanced(str))
      System.out.print("1");
      else
      System.out.print("0");
   }
}
```