Q1. You will be given an integer n, and a threshold, k. For each number from 1 to n, find the maximum value of the Bitwise &, Bitwise | and Bitwise ^, when compared that are greater than i. Consider a value only if the comparison returns less than k. Print the results of the Bitwise &, Bitwise | and Bitwise ^ comparisons on separate lines, in that order.

Function Description

- int n: the highest number to consider
- int k: the result of a comparison must be lower than this number to be considered

Input Format

The input line contains 2 space-separated integers, n and k.

Output Format

Output consists
Print the maximum values for the and, or and xor comparisons, each on a separate line.

Sample Input 1

3 3

Sample Output 2

2
0
2

Explanation

n = 3, k = 3

All possible values of a and b are:

a	b	&	|	^
1	2	0	3	3
1	3	1	3	2
2	3	2	3	1

For the Bitwise & comparison, the maximum is 2.
For the Bitwise | comparison, none of the values is less than k, so the maximum is 0.
For the Bitwise ^ comparison, the maximum value less than k is 2.
The program will print:

Sample Input 2

5 4

Sample Output 2

2
3
3

Explanation

n = 5, k = 4

All possible values of a and b are:

a b	&	|	^
1 2	0	3	3
1 3	1	3	2
1 4	0	5	5
1 5	1	5	4
2 3	2	3	1
2 4	0	6	6
2 5	0	7	7
3 4	0	7	7
3 5	1	7	6
4 5	4	5	1

The maximum possible value of a & b that is also < (k = 4) is 2, so, we print 2 on first line.
The maximum possible value of a l b that is also < (k = 4) is 3, so, we print 3 on second line.
The maximum possible value of a ^ b that is also < (k = 4) is 3, so, we print 3 on second line.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	5 6	7 3	12 15	87 2	36 49
Output	4 5 5	2 0 2	10 14 14	1 0 1	34 48 48

#Solution

```
import java.util.Scanner;

class Main

{

	public static void main(String[] args)

	{

		Scanner inp=new Scanner(System.in);

		int n=inp.nextInt();

		int k=inp.nextInt();


		int and=0,fand=0,or=0,finalor=0,xor=0,fxor=0;

		for (int i=1 ; i<=n ; i++)

		{
```

```
                for (int j=i+1 ; j<=n ; j++)
                {
                        and=i&j;
                        if (and>fand && and<k)
                        {
                                fand=and;
                        }
                        or=i|j;
                        if (or>finalor && or<k)
                        {
                                finalor=or;
                        }
                        xor=i^j;
                        if (xor>fxor && xor<k)
                        {
                                fxor=xor;
                        }
                }
        }
        System.out.println(fand);
        System.out.println(finalor);
        System.out.println(fxor);
    }

}
```

Q.2. Square Pattern

Print a pattern of numbers from 1 to n as shown below.

Input Format

The input consists of single integer n.

Output Format
It will print pattern according to input.

Sample Input 1
2

Sample Output 1
222
212
222

Sample Input 2
5

Sample Output 2
555555555
544444445
543333345
543222345
543212345
543222345
543333345
544444445
555555555

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	6	3	4	1	2
Output	66666666666 65555555556 65444444456 65433333456 65432223456 65432123456 65432223456 65433333456 65444444456 65555555556 66666666666	33333 32223 32123 32223 33333	4444444 4333334 4322234 4321234 4322234 4333334 4444444	1	222 212 222

#Solution

```
import java.util.Scanner;

class Main

{
```

```
	public static void main(String[] args)
	{
		Scanner inp=new Scanner(System.in);
		int n=inp.nextInt();
		for (int i=1 ; i<=(2*n)-1 ; i++)
		{
			int a=n;
			for (int j=1 ; j<=(2*n)-1 ; j++)
			{
				System.out.print(a);
				if (j<i)
				{
					a--;
				}
				if (j>((2*n)-1-i))
				{
					a++;
				}
			}
			System.out.println();
		}
	}
}
```

Q.3. Given an array, consisting of digits, find the frequency of each digit in the given array.

Input Format

Input consists of two lines,

First input consists integer value n,

Second input consists n space separated integer values.

Output Format

Print ten space-separated integers in a single line denoting the frequency of each digit from 0 to 9.

Sample Input 1
10
2 1 1 4 7 2 5 5 0 6

Sample Output 1
1 2 2 0 1 2 1 1 0 0

Explanation 1

In the given string:

- 1,2 and 5 occurs two times.
- 0, 4, 6 and 7 occur one time each.
- The remaining digits 3,8 and 9 don't occur at all.

Sample Input 2
15

12 12 1 2 7 4 6 7 12 98 33 78 12 4 8

Sample Output 2
0 1 1 0 2 0 1 2 1 0

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	5 11 1 1 1 1	7 1 2 3 3 2 1 7	15 1 2 1 4 7 8 9 5 4 2 3 6 7 8 2	7 2 77 99 1 2 66 77	2 9 9
Output	0 4 0 0 0 0 0 0 0 0	0 2 2 2 0 0 0 1 0 0	0 2 3 1 2 1 1 2 2 1	0 1 2 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 2

#Solution

```
import java.util.Scanner;

class Main

{

        public static void main(String[] args)

        {

                Scanner inp=new Scanner(System.in);

                int size = inp.nextInt();

                int[] arr=new int[size];
```

```
            for (int i=0 ; i<size ; i++)
            {
                arr[i]=inp.nextInt();
            }

            for (int i=0 ; i<=9 ; i++)
            {
                int c=0;
                for(int j=0 ; j<arr.length ; j++)
                {
                    if (arr[j]==i)
                    {
                        c++;
                    }
                }
                System.out.print(c+" ");
            }
        }
}
```

Q.4. You are provided with 3 numbers: input1, input2, and input3. Each of these is four-digit numbers within the range >=1000 and <=9999 i.e. 1000 >= input1 <= 9999, 1000 >=input2 <= 9999, 1000 >=input3 <= 9999, You are expected to find the Key using the below formula Key = [SMALLEST digit in the thousands place of all three numbers] [LARGEST digit in the hundreds place of all three numbers] [SMALLEST digit in the tens place of all three numbers] [LARGEST digit in the units place of all three numbers].

Given three numbers, write a JAVA program to find the key using the above-mentioned formula.

Input Format

The input consists of three space-separated integers - input1, input2 and input3 representing the 3 four-digit numbers.

Output Format

Print an integer representing the concatenation of four digits where the first digit represents the smallest digit in the thousands place of all three numbers; the second digit represents the largest digit in the hundreds place of all three numbers; the third digit represents the smallest digit in the tens place of all three numbers and the fourth digit represents the largest digit in the unit place of all three numbers.

Sample Input 1
3521 2452 1352

Sample Output 1
1522

Explanation

Key = [smallest digit in the thousands place of all three numbers] [LARGEST digit in the hundreds place of all three numbers] [smallest digit in the tens place of all three numbers] [LARGEST digit in the units place of all three numbers]

If input1 = 3521, input2=2452, input3-1352, then Key = [1][5][2][2] = 1522.

Sample Input 2
1234 4321 1243

Sample Output 2
1324

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	3212 3425 2349	5783 2341 2153	3421 2423 1234	1111 2222 3333	3521 2452 1352
Output	2419	2743	1424	1313	1522

#Solution

```
import java.util.*;

class Main

{

 public static int fun(int a, int b, int c)

 {

int n = 1, temp = 0;

 for(int i = 0; i < 4; i++)

 {
```

```
int rem = 0, large = -999, small = 999;
if(i % 2 == 0)
{
rem = a % 10;
a = a / 10;
if(rem > large)
large = rem;
rem = b % 10;
b = b / 10;
if(rem > large)
large = rem;
rem = c % 10;
c = c/ 10;
if(rem > large)
large = rem;
temp = (large * n) + temp;
}
else
{
rem = a % 10;
a = a / 10;
if(rem < small)
small = rem;
rem = b % 10;
b = b / 10;
if(rem < small)
small = rem;
rem = c % 10;
c = c/ 10;
if(rem < small)
small = rem;
temp = (small * n) + temp;
```

```
}
n = n * 10;
}
return temp;
}
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
int a = sc.nextInt();
int b = sc.nextInt();
int c = sc.nextInt();
System.out.print(fun(a, b, c));
}
}
```

Q.5. The games development company "ShootingGames" has developed a balloon shooter game. The balloons are arranged in a linear sequence and each balloon has a number associated with it. The numbers on the balloons are in the Fibonacci series. In the game, the player shoots 'k' balloons. The player's score is the sum of numbers on the 'k' balloons, If no balloons are shot then print 0.

Write a java program to generate the player's score.

Input Format

The input consists of an integer - numBalloons representing the total number of balloons shot by the player (k).

Output Format

Output Format Print an integer value representing the player's score. If no balloons are shot then print 0.

Sample Input 1

7

Sample Output 1

20

Sample Input 2

8

Sample Output 2

33

Explanation

The Fibonacci sum is 0+1+1+2+3+5+8=20 Input Format.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	9	10	7	6	4
Output	54	88	20	12	4

#Solution

```
import java.util.Scanner;
class Main
{
 public static int fibo(int n)
 {
 int[] fibs = new int[n];
 fibs[0] = 0;
 fibs[1] = 1;
 int sum=0;
 for(int i=2;i<n;i++)
 {
 fibs[i] = fibs[i-1]+fibs[i-2];
 }
 for(int i=0;i<n;i++)
 {
 sum += fibs[i];
 }
 return sum;
}
public static void main(String[] args)
{
```

```
Scanner s = new Scanner(System.in);

int numBalloons = s.nextInt();

int result = fibo(numBalloons);

System.out.println(result);

}

}
```

Q.6. The children's toy-making company is building cubic-shaped learning toys. The company has a list of N dimensions suggested by its designers but they wish to choose only those dimensions for the toys that are perfect cube numbers. To do this, they need to know the perfect cube numbers present in the list of dimensions.

Write a JAVA program to help the toy manufacturers find the perfect cube numbers present in the list of dimensions. If no such number found print 0.

Input Format

The first line of input consists of an integer - numDimensions representing the total number of dimensions selected by the designers (N).
The second line of input consists of N space-separated integers - dimens1 , dimens2 , ... dimensN-1 representing the value of the dimensions selected by the designers.

Output Format
Print a space separated integer value representing the dimensions that are perfect cube numbers. If there are no perfect cube number print 0.

Sample Input 1
3

27 67 64

Sample Output 1
27 64

Sample Input 2
9

23 1 8 56 27 67 64 125 232

Sample Output 2
1 8 27 64 125

Explanation

The cube numbers are 1, 8, 27, 64, 125.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	4 15 1 64 81	10 1 8 64 125 846 87 45 56 78 62	6 45 78 64 94 56 45	7 55 77 99 66 44 11 22	3 0 0 0
Output	1 64	1 8 64 125	64	0	0

#Solution

```
import java.util.Scanner;
class Main
{
    public static boolean dimens_is_cube(int n)
    {
        int i=1;
        boolean result = false;
        while(i<=n)
        {
            if(i*i*i == n)
            {
                result = true;
                break;
            }
            i++;
        }
        return result;
    }

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

```
                int[] dimens = new int[numDimensions];
                for(int i=0;i<numDimensions;i++)
                {
                        dimens[i] = s.nextInt();
                }
                int perfect_cube = 0;
                for(int i=0;i<numDimensions;i++)
                {
                        if(dimens_is_cube(dimens[i]))
                        {
                                perfect_cube++;
                                System.out.print(dimens[i]+" ");
                        }
                }
                if (perfect_cube==0)
                {
                        System.out.print(perfect_cube);
                }
        }
}
```

Q.7. The Lottery company has a list of N lottery ticket codes. The ticket code is a numeric value. The lottery has to shortlist the runner-up prize winner. The shortlisting strategy involves selecting the code whose frequency in the list is equal to its value. If there are multiple codes where frequency in the list is equal to its value, select the one which has the largest value.

Write a Java Program to select the shortlisted ticket code.

Input Format

The first line of the input consists of an integer - size, representing the size of the list(N).

The second line consists of N space-separated integers - ticket0, ticket1, … ticketN-1.

Output Format

Print an integer representing the code of the shortlisted ticket. If no code exists, output -1.

Sample Input 1
6
1 1 2 3 3 4

Sample Output 1
-1

Explanation

No code exists whose frequency in the list is equal to its value

Sample Input 2
8
2 7 2 3 4 3 2 3

Sample Output 2
3

Explanation

3 occurs 3 times in the list. Hence the output is 3

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	8 3 3 3 4 4 4 4 1	10 2 4 4 4 4 3 3 3 0 1	11 5 5 5 5 5 6 6 6 6 6 6	4 0 2 3 4	8 2 7 2 3 4 3 2 3
Output	4	4	6	-1	3

#Solution

```
import java.util.*;

class Main

{

        public static void sort(int[] number,int n)

        {

                int i,j,a;

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

                {

                        for (j = i + 1; j < n; ++j)

                        {
```

```
                    if (number[i] < number[j])
                    {
                        a = number[i];
                        number[i] = number[j];
                        number[j] = a;
                    }
                }
            }
        }

        public static void frequency(int[] arr,int[] freq,int size)
        {
            int i,j,count;
            for(i=0; i<size; i++)
            {
                count = 1;
                for(j=i+1; j<size; j++)
                {
                    if(arr[i]==arr[j])
                    {
                        count++;
                        freq[j] = 0;
                    }
                }
                if(freq[i] != 0)
                {
                    freq[i] = count;
                }
            }
        }
```

```
    public static void main(String[] args)
    {
        int i, j, a, n,flag =0;
        Scanner s= new Scanner(System.in);
        n=s.nextInt();
        int[] number=new int[n];
        int[] freq=new int[n];
        for (i = 0; i < n; ++i)
        {
            number[i]=s.nextInt();
            freq[i] = -1;
        }
        sort(number,n);
        frequency(number,freq,n);
        for(i=0; i<n; i++)
        {
            if(freq[i] == number[i])
            {
                flag=1;
                System.out.print(number[i]);
                break;
            }
        }
        if(flag==0)
        {
            System.out.print("-1");
        }
    }
}
```

Q.8. An apparel company has a list of sales rates of N items from its website. Each item is labelled with a unique ID from 0 to N-1. The company has a target sales rate, say K. The company wishes to know the two items whose total sales rate is equal to the target sales rate. The list of sales is prepared in such a way that no more than two items' total sales rate will be equal to the target rate.

Write a java program to find the ID of the items whose total sales value is equal to the given target value.

Input Format

The first line of input consists of two space-separated integers - numItems and target, representing the total number of selected items (N) and the selected target sales rate (K). The second line of input consists of N space-separated integers - item0, item1, item2, ... itemN-1, representing the sales rate of the selected items.

Output Format

Print two space-separated integers representing the ID of the items whose total sales rate is equal to the given target rate.

Sample Input 1

5 14
6 5 3 11 10

Sample Output 1

2 3

Explanation

The ID of two items with a total sales rate of 14 is 2 & 3 (3+11=14).

Sample Input 2

5 10
1 2 3 4 6

Sample Output 2

3 4

Explanation

The ID of two items with a total sales rate of 10 is 3 & 4 (4+6=10).

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	5 12 7 6 5 4 3	5 10 7 6 5 2 3	6 20 10 9 8 7 6 11	4 7 1 2 3 4	5 14 6 5 3 11 10
Output	0 2	0 4	1 5	2 3	2 3

#Solution

```
import java.util.* ;
class Main
{
	public static void main (String[] args)
	{
		Scanner sc = new Scanner(System.in) ;
		int n , t ;
		n = sc.nextInt() ;
		t = sc.nextInt() ;
		int arr[] = new int [n] ;
		for(int i = 0 ; i<n ; i++)
		{
			arr[i] = sc.nextInt() ;
		}
		for(int i = 0 ; i<n-1 ; i++)
		{
			for(int j=i+1 ; j<n ; j++)
			{
				if(arr[i]+arr[j] == t)
				{
					System.out.print(i+" "+j);
				}
			}
		}
	}
}
```

Q.9. The apparel company has collected a list of the sales values of the N highest selling brands of products during the festive season. Each brand is identified by a unique ID numbered 0 - (N-1) in the list. From this list, the company wishes to determine the K^{th} largest sales value for a given day.

Write a Java program to help the company determine the sales value of the Kth largest selling product.

Input Format

The first line of input consists of two integers - numBrands and largestSale where the first integer represents the total number of brands selected by the company (N) and the second integer represents the K^{th} value.

The next line consist of N space separated integers - salesValue[0], salesValue[1], ... salesValue[N-1] representing the sales value of the N brands.

Output Format

Print an integer representing the Kth largest sales value in the list for the given day. If no k^{th} largest item available print "Invalid Input".

Sample Input 1

3 4
1 2 3

Sample Output 1

Invalid Input

Explanation

The fourth-largest element in the array is not available. Hence the output is Invalid Input.

Sample Input 2

5 3
45 32 67 21 12

Sample Output 2

32

Explanation

The third-largest element in the array is 32. Hence the output is 32.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	7 2 3 6 9 2 4 8 10	10 9 12 35 69 78 45 15 26 20 70 85	5 1 34 67 23 90 78	7 9 1 2 3 4 5 6 7	1 2 3
Output	9	15	90	Invalid Input	Invalid Input

#Solution

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

```
class Main
{
public static int klargest(int[] arr, int k)
{
	Arrays.sort(arr);
	int targetIndex = arr.length - k;
	return arr[targetIndex];
}
public static void main(String[] args)
{
	int i,n,k;
	Scanner s=new Scanner(System.in);
	n=s.nextInt();
	k=s.nextInt();
	int[] arr=new int[n];
	for (i=0;i<n;i++)
	{
		arr[i]=s.nextInt();
	}
	if(k>n)
	{
		System.out.println("Invalid Input");
		System.exit(0);
	}
	System.out.println(klargest(arr,k));
}
}
```

Q.10. In a gaming hub, N number of players were playing the same type of game. All players got stuck at the pillar level in the game, each with a different score. The owner of the gaming hub announced that players can pass that level if they can break two pillars. Both pillars have their own health points. The trick is to break one pillar at a time that if the

player's current score is multiplied up to a certain point, then it should be equal to the pillar's health. The same trick is to be used for the other pillar. If there is no number that can be multiplied with the player's score to make the score equal to the pillar's health, then that player loses. A player can only break one pillar at a time and if the player is not able to break both the pillars then they will not be able to clear the level.

Write a Java program to find the total number of players who will clear that level of the game.

Input Format
The first line of the input consists of an integer -numPlayers representing the number of players (N).

The second lines consist of N space-separated integers - score1, score2, ..., scoreN, representing the score of each player.

The last line of the input consists of two integers - health1, health2 representing the health of both the pillars respectively.

Output Format
Print the count of the players who will clear that level of the game

Sample Input 1
5
15 5 3 7 9
90 30

Sample Output 1
3

Sample Input 2
5
15 5 3 7 9
135 90

Sample Output 2
4

Explanation
Scores 15, 5, 3, 9 can be multiplied by another number to equal 135 and 90 which will break the pillar. So, the output is 4.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	6 15 5 3 7 9 45 135 90	8 15 5 3 7 9 45 10 20 15 90	7 10 20 30 40 50 60 70 20 10	8 11 15 19 27 29 34 19 32 23 25	5 15 5 3 7 9 135 90
Output	5	3	1	0	4

#Solution

```
import java.util.*;
class Main
{
	public static int fun(int n, int arr[], int x, int y)
	{
		int i, count = 0;
		for(i = 0; i < n; i++)
		{
			if(x % arr[i] == 0 && y % arr[i] == 0)
			count++;
		}
		return count;
	}
	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 x = sc.nextInt();
		int y = sc.nextInt();
		System.out.print(fun(n, arr, x, y));
	}
}
```

Q.11. An e-commerce company wishes to bucketize its products. Each product has a product ID. The product ID is a numeric number. The company has to find the bucket ID from the product ID. The company needs an application that takes the product ID as an input, calculates the smallest and largest permutation with the digits of the product ID, then outputs the sum of these smallest and largest permutations as the bucket ID.

Write a Java program for the company to find the bucket ID for the given product.

Input Format
The input consists of an integer - productID, representing the product ID of the product.

Output Format
Print an integer representing the bucket ID for the given product.

Sample Input 1
556

Sample Output 1
1211

Sample Input 2
734

Sample Output 2
1090

Explanation
The smallest permutation with digits 7, 3, 4 is 347,
The largest permutation with digits 7, 3, 4 is 743,
The sum of 347+743 = 1090. So, the output is 1090.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	8416	1234	431	10101	343
Output	10109	5555	565	11211	767

#Solution

```
import java.util.*;

class Main

{

        public static void main (String[] args)

        {

                Scanner sc = new Scanner(System.in) ;

                int n = sc.nextInt() ;

                int sum , i= 0 , j , temp , p , q , temp1 , k = 0 , t = 0 , fin ;
```

```
int a[] = new int[50] ;
while(n>0)
{
        a[i] = n % 10 ;
        n = n / 10 ;
        i++ ;
}
int count = i ;
for(i = 0 ; i<count ; i++)
{
        for(j = i+1 ; j<count ; j++)
        {
                if(a[i]>a[j])
                {
                        temp = a[i] ;
                        a[i] = a[j] ;
                        a[j] = temp ;
                }
        }
}
for(i = 0 ; i<count ; i++)
        k = 10*k + a[i] ;
for(p = 0 ; p<count ; p++)
{
        for(q = p+1 ; q<count ; q++)
        {
                if(a[p]<a[q])
                {
                        temp1=a[p] ;
                        a[p]=a[q] ;
                        a[q]=temp1 ;
```

```
                        }
                    }
                }
                for (p= 0; p< count; p++)
                    t = 10 * t + a[p] ;
                fin=k+t;
                System.out.println(fin);
                }
}
```

Q.12. Andrew manages a pipe warehouse. He wishes to automate the process of transferring the pipes from the warehouse to the carrier truck. There are N pipes in the warehouse placed vertically along a wall. In the automated system, a drone picks the pipes by length and carries them to the carrier truck. In each turn, the drone moves from left to right to find the pipes whose lengths are greater than the pipe on their left. After finding the pipe, the drone takes the pipe to the carrier truck. The drone repeats this process until it has no more pipes to pick.

Write a Java program to output the list of pipes that will remain in the warehouse after the drone has completed this process.

Input Format
The first line of the input consists of an integer - numOfPipes, representing the number of pipes in the warehouse (N).
The second line consists of N space-separated integers - len[0], len[1], ... len[N-1], representing the length of the pipes.

Output Format
Print space-separated integers representing the list of the remaining pipes in the warehouse.

Sample Input 1
5
3 5 4 8 7

Sample Output 1
3

Sample Input 2
5
3 2 4 6 5

Sample Output 2

3 2

Explanation
In the first turn, the drone picks the pipe with length 4 as 4 > 2. So, the remaining pipes are 3, 2, 6, and 5.
In the next turn, the drone picks the pipe with length 6 as 6 > 2. So, the remaining pipes are 3, 2, and 5.
In the next turn, the drone picks the pipe with length 5 as 5 > 2. So, the remaining pipes are 3 and 2.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Input	5 1 2 3 4 5	5 4 5 2 1 3	7 1 5 2 6 3 7 4	9 9 8 7 6 5 4 3 1 2	6 3 9 2 8 1 7
Output	1	4 2 1	1	9 8 7 6 5 4 3 1	3 2 1

#Solution

```
import java.util.Scanner ;

class Main

{

	public static void main(String[] args)

	{

		Scanner scan = new Scanner(System.in) ;

		int n = scan.nextInt() ;

		int [] pipes = new int[n] ;

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

		{

			pipes[i] = scan.nextInt() ;

		}

		System.out.print(pipes[0]+" ") ;

		int temp = pipes[0];

		for(int i = 1 ; i<n ; i++)

		{

			if(pipes[i]<temp)

			{

				System.out.print(pipes[i]+" ");
```

```
                    temp = pipes[i];
                }
            }
        }
    }
```