me index values in an array:

a prime) is a natural number greater than 1 that
er than 1 and itself. In other words, a prime
greater than 1, whose only two whole-number
first few prime numbers are 2, 3, 5, 7, 11, 13, 17,

nents, you are expected to find the sum of the
on-prime indexes of the array. Note that the
the position (index) of the first array element is
rray element is 1, and so on.
ents are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100},
rime index are 10,20,50,70,90,100 and their sum

ents are {-1, -2, -3, 3, 4, -7}, then the values at
2, 4 and their sum is **1**.
ents are {-4, -2}, the values at the non-prime
is **-6**.

```
        f=0;
        for(int j=2;j<i;j++){
            if(i%j==0){
                f=1;
                break;
            }}
            if(f==1){
                System.out.println(i);
                arr.add(i);
            }
        }}
        if(arr.size()==0){
            return ans;
        }
        int a=0;
        for(int i=0;i<arr.size();i++){
            a=arr.get(i);
            ans+=input1[a];
        }return ans;}}
```

tests.mettl.com is sharing your screen. Stop sharing Hide

Compile and Test Submit Code

Need Help? Contact us: +1 (800) 265-6038 +91 80471-90902

Powered By Mercer mettl

lenovo

es in an array:

natural number greater than 1 that
itself. In other words, a prime
, whose only two whole-number
e numbers are 2, 3, 5, 7, 11, 13, 17,

expected to find the sum of the
xes of the array. Note that the
ndex) of the first array element is
1, and so on.
0, 30, 40, 50, 60, 70, 80, 90, 100},
e 10,20,50,70,90,100 and their sum

, -3, 3, 4, -7}, then the values at
sum is **1**.
}, the values at the non-prime

Compiler: Java - 1.8

```
{

    public int sumOfNonPrimeIndexValues(int[] input1,int input2){
        // Read only region end
        // Write code here...
        int ans=0;
        int f=0;
        ArrayList<Integer>arr=new ArrayList<Integer>();
        if(input2==1){
            arr.add(0);
        }
        if(input2==2){
            arr.add(0);
            arr.add(1);
        }if(input2>2){
            arr.add(0);
            arr.add(1);
        for(int i=2;i<input2;i++){

            f=0;
            for(int j=2;j<i;j++){
                if(i%j==0){
```

tests.mettl.com is sharing your screen. Stop sharing Hide

Compile and Test Submit Code

How to Attempt?

Calculate sum of non-prime index values in an array:

What is a prime number?

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. In other words, a prime number is a whole number greater than 1, whose only two whole-number factors are 1 and itself. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29.

Given an array with 'N' elements, you are expected to find the sum of the values that are present in non-prime indexes of the array. Note that the array index starts with 0 i.e. the position (index) of the first array element is 0, the position of the next array element is 1, and so on.

Example 1: If the array elements are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}, then the values at the non-prime index are 10,20,50,70,90,100 and their sum is **340**.

Example 2: If the array elements are {-1, -2, -3, 3, 4, -7}, then the values at the non-prime index are -1,-2, 4 and their sum is **1**.

Example 3: If the array elements are {-4, -2}, the values at the non-prime index are -4,-2 and their sum is **-6**.

```
public int sumOfNonPrimeIndexValu
    // Read only region end
    // Write code here...
    int ans=0;
    int f=0;
    ArrayList<Integer>arr=new Arr
    if(input2==1){
        arr.add(0);
    }
    if(input2==2){
        arr.add(0);
        arr.add(1);
    }if(input2>2){
        arr.add(0);
        arr.add(1);
    for(int i=2;i<input2;i++){

        f=0;
        for(int j=2;j<i;j++){
            if(i%j==0){
```

tests.mettl.com is sharing your screen. Stop sharing Hide

the range >=1000 and <=9999. i.e.

1000 <= input1 <= 9999

1000 <= input2 <= 9999

1000 <= input3 <= 9999

input4 is a positive integer number.

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) = 12

Example2 - If input4=37 (an ODD number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (3+5+1+5+1+3+5) = 23

Assuming that the 4 numbers are passed to the given function, Complete the function to find and return the PIN.

tests.n

Scanned with CamScanner

```
int a=0;
int b=0;
String a1=Integer.toString(input1);
String b1=Integer.toString(input2);
String c1=Integer.toString(input3);
for(int i=0;i<a1.length();i++){
    String q="";
    q+=a1.charAt(i);
    int w=Integer.valueOf(q);
    if(w%2==0)
        a+=w;
    else
        b+=w;
}
for(int i=0;i<b1.length();i++){
    String q="";
    q+=b1.charAt(i);
    int w=Integer.valueOf(q);
    if(w%2==0)
        a+=w;
    else
        b+=w;
}
```

Scanned with CamScanner

Main.java

```
            q+=b1.charAt(i);
            int w=Integer.valueOf(q);
            if(w%2==0)
                a+=w;
            else
                b+=w;
        }
        for(int i=0;i<c1.length();i++){
            String q="";
            q+=c1.charAt(i);
            int w=Integer.valueOf(q);
            if(w%2==0)
                a+=w;
            else
                b+=w;
        }
        if(input4%2==0)
            return a;
        else
            return b;
    }
}
```

Scanned with CamScanner

You are provided with 4 numbers : input1, input2, input3 and input4 and are expected to find a secret PIN.

The three numbers input1, input2 and input3 are four digit numbers within the range >=1000 and <=9999. i.e.

1000 <= input1 <= 9999

1000 <= input2 <= 9999

1000 <= input3 <= 9999

input4 is a positive integer number.

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3) - (sum of odd digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3) - (sum of even digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) - (3+5+1+5+1+3+5) = -11

Example2 - If input4=37 (an ODD number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (3+5+1+5+1+3+5) - (2+2+4+2+2) = 11

Assuming that the 4 numbers are passed to the given function, Complete the function to find and return the PIN.

Scanned with CamScanner

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3) - (sum of odd digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3) - (sum of even digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) - (3+5+1+5+1+3+5) = -11

Example2 - If input4=37 (an ODD number) and if input1 = 3521,

Scanned with CamScanner

```
11        int a=0;
12        int b=0;
13        String a1=Integer.toString(input1);
14        String b1=Integer.toString(input2);
15        String c1=Integer.toString(input3);
16        for(int i=0;i<a1.length();i++){
17            String p="";
18            p+=a1.charAt(i);
19            int q=Integer.valueOf(p);
20            if(q%2==0){
21                a+=q;
22            }
23            else{
24                b+=q;
25            }
26        }
27        for(int i=0;i<a1.length();i++){
28            String p="";
29            p+=b1.charAt(i);
30            int q=Integer.valueOf(p);
31            if(q%2==0){
```

Scanned with CamScanner

Attempted: 2/2

```
for(int i=0;i<a1.length();i++){
    String p="";
    p+=b1.charAt(i);
    int q=Integer.valueOf(p);
    if(q%2==0){
        a+=q;
    }
    else{
        b+=q;
    }
}
for(int i=0;i<a1.length();i++){
    String p="";
    p+=c1.charAt(i);
    int q=Integer.valueOf(p);
    if(q%2==0){
        a+=q;
    }
    else{
```

Compile and Test

Submit C

Use Custom Input

Scanned with CamScanner

Attempted: 2/2

```
            p+=c1.charAt(i);
            int q=Integer.valueOf(p);
            if(q%2==0){
                a+=q;
            }
            else{
                b+=q;
            }
        }
        if(input4%2==0){
            return a-b;
        }
        else{
            return b-a;
        }


    }
}
```

Use Custom Input

Compile and Test

Submit C

Scanned with CamScanner

How to Attempt?

Find first repeated element in the given array from the tail of the array:
Rajeev works in the data center lab of the survey department. He has been assigned the task of identifying "repeated numbers" in a given set of numbers. He approaches you to help him achieve this.
Given an array of numbers, your task is to return the first number that repeats (appears again) starting from the last index.

For example -
If input1 = {1,2,4,1,2,8} representing the given array, and
input2 = 6 representing the number of elements in the array,
then the result should be 2 which is the first repeated number in the array from the rear end.

Special conditions to be taken care:
Note 1: You should ignore the negative numbers and zeros. The program should consider only non-zero, non-negative numbers from the given array.
Note 2: If no number is repeated then the output should be the last element of the array.
Note 3: If all elements in the array are negative or 0's the output should be 0.

```
1   import j
2   import
3
4   //
5   cl
6   {
7
8       pub
9
10
11
12      }
13  }
```

Use Custom

Scanned with CamScanner

JAVA7 | Compiler: Java - 1.7

```java
        int o=0;
        int q=0;
        int w=0;
        for(int i=input2-1;i>-1;i--){
            if(input1[i]>0){
            o=1;
            for(int j=i-1;j>-1;j--){
                if(input1[i]==input1[j] && j>=q){
                    System.out.println("LL");
                    w=input1[i];
                    q=j;
                }
            }
            }
        }
        System.out.println(w+" "+q);
        if(o==0){
            return 0;
        }
        if(w!=0){
            return w;
        }
```

Scanned with CamScanner

```
            }
        }
        }
    }
    System.out.println(w+" "+q);
    if(o==0){
        return 0;
    }
    if(w!=0){
        return w;
    }
    return input1[input2-1];
}
}
```

Compile and Test

Submit

Scanned with CamScanner

me index values in an array:

a prime) is a natural number greater than 1 that
er than 1 and itself. In other words, a prime
greater than 1, whose only two whole-number
first few prime numbers are 2, 3, 5, 7, 11, 13, 17,

nents, you are expected to find the sum of the
on-prime indexes of the array. Note that the
the position (index) of the first array element is
rray element is 1, and so on.
ents are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100},
rime index are 10,20,50,70,90,100 and their sum

ents are {-1, -2, -3, 3, 4, -7}, then the values at
2, 4 and their sum is **1**.
ents are {-4, -2}, the values at the non-prime
is **-6**.

```
        f=0;
        for(int j=2;j<i;j++){
            if(i%j==0){
                f=1;
                break;
            }}
            if(f==1){
                System.out.println(i);
                arr.add(i);
            }
        }}
        if(arr.size()==0){
            return ans;
        }
        int a=0;
        for(int i=0;i<arr.size();i++){
            a=arr.get(i);
            ans+=input1[a];
        }return ans;}}
```

tests.mettl.com is sharing your screen. Stop sharing Hide

Compile and Test Submit Code

Need Help? Contact us: +1 (800) 265-6038 +91 80471-90902

Powered By Mercer mettl

lenovo

es in an array:

natural number greater than 1 that
itself. In other words, a prime
, whose only two whole-number
e numbers are 2, 3, 5, 7, 11, 13, 17,

expected to find the sum of the
xes of the array. Note that the
ndex) of the first array element is
1, and so on.
0, 30, 40, 50, 60, 70, 80, 90, 100},
e 10,20,50,70,90,100 and their sum

, -3, 3, 4, -7}, then the values at
sum is **1**.
}, the values at the non-prime

Compiler: Java - 1.8

```
{

    public int sumOfNonPrimeIndexValues(int[] input1,int input2){
        // Read only region end
        // Write code here...
        int ans=0;
        int f=0;
        ArrayList<Integer>arr=new ArrayList<Integer>();
        if(input2==1){
            arr.add(0);
        }
        if(input2==2){
            arr.add(0);
            arr.add(1);
        }if(input2>2){
            arr.add(0);
            arr.add(1);
        for(int i=2;i<input2;i++){

            f=0;
            for(int j=2;j<i;j++){
                if(i%j==0){
```

tests.mettl.com is sharing your screen. Stop sharing Hide

Compile and Test Submit Code

How to Attempt?

Calculate sum of non-prime index values in an array:

What is a prime number?

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. In other words, a prime number is a whole number greater than 1, whose only two whole-number factors are 1 and itself. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29.

Given an array with 'N' elements, you are expected to find the sum of the values that are present in non-prime indexes of the array. Note that the array index starts with 0 i.e. the position (index) of the first array element is 0, the position of the next array element is 1, and so on.

Example 1: If the array elements are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}, then the values at the non-prime index are 10,20,50,70,90,100 and their sum is **340**.

Example 2: If the array elements are {-1, -2, -3, 3, 4, -7}, then the values at the non-prime index are -1,-2, 4 and their sum is **1**.

Example 3: If the array elements are {-4, -2}, the values at the non-prime index are -4,-2 and their sum is **-6**.

```
public int sumOfNonPrimeIndexValu
    // Read only region end
    // Write code here...
    int ans=0;
    int f=0;
    ArrayList<Integer>arr=new Arr
    if(input2==1){
        arr.add(0);
    }
    if(input2==2){
        arr.add(0);
        arr.add(1);
    }if(input2>2){
        arr.add(0);
        arr.add(1);
    for(int i=2;i<input2;i++){

        f=0;
        for(int j=2;j<i;j++){
            if(i%j==0){
```

tests.mettl.com is sharing your screen. Stop sharing Hide

the range >=1000 and <=9999. i.e.

1000 <= input1 <= 9999

1000 <= input2 <= 9999

1000 <= input3 <= 9999

input4 is a positive integer number.

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) = 12

Example2 - If input4=37 (an ODD number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (3+5+1+5+1+3+5) = 23

Assuming that the 4 numbers are passed to the given function, Complete the function to find and return the PIN.

tests.n

Scanned with CamScanner

```
int a=0;
int b=0;
String a1=Integer.toString(input1);
String b1=Integer.toString(input2);
String c1=Integer.toString(input3);
for(int i=0;i<a1.length();i++){
    String q="";
    q+=a1.charAt(i);
    int w=Integer.valueOf(q);
    if(w%2==0)
        a+=w;
    else
        b+=w;
}
for(int i=0;i<b1.length();i++){
    String q="";
    q+=b1.charAt(i);
    int w=Integer.valueOf(q);
    if(w%2==0)
        a+=w;
    else
        b+=w;
}
```

Scanned with CamScanner

Main.java

```
            q+=b1.charAt(i);
            int w=Integer.valueOf(q);
            if(w%2==0)
                a+=w;
            else
                b+=w;
        }
        for(int i=0;i<c1.length();i++){
            String q="";
            q+=c1.charAt(i);
            int w=Integer.valueOf(q);
            if(w%2==0)
                a+=w;
            else
                b+=w;
        }
        if(input4%2==0)
            return a;
        else
            return b;
    }
}
```

Scanned with CamScanner

You are provided with 4 numbers : input1, input2, input3 and input4 and are expected to find a secret PIN.

The three numbers input1, input2 and input3 are four digit numbers within the range >=1000 and <=9999. i.e.

1000 <= input1 <= 9999

1000 <= input2 <= 9999

1000 <= input3 <= 9999

input4 is a positive integer number.

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3) - (sum of odd digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3) - (sum of even digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) - (3+5+1+5+1+3+5) = -11

Example2 - If input4=37 (an ODD number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (3+5+1+5+1+3+5) - (2+2+4+2+2) = 11

Assuming that the 4 numbers are passed to the given function, Complete the function to find and return the PIN.

Scanned with CamScanner

If input4 is EVEN, **the PIN** = (sum of even digits in input1 input2 and input3) - (sum of odd digits in input1 input2 and input3)

If input4 is ODD, **the PIN** = (sum of odd digits in input1 input2 and input3) - (sum of even digits in input1 input2 and input3)

Example1 - If input4=38 (an EVEN number) and if input1 = 3521, input2=2452, input3=1352, then PIN = (2+2+4+2+2) - (3+5+1+5+1+3+5) = -11

Example2 - If input4=37 (an ODD number) and if input1 = 3521,

Scanned with CamScanner

```
11        int a=0;
12        int b=0;
13        String a1=Integer.toString(input1);
14        String b1=Integer.toString(input2);
15        String c1=Integer.toString(input3);
16        for(int i=0;i<a1.length();i++){
17            String p="";
18            p+=a1.charAt(i);
19            int q=Integer.valueOf(p);
20            if(q%2==0){
21                a+=q;
22            }
23            else{
24                b+=q;
25            }
26        }
27        for(int i=0;i<a1.length();i++){
28            String p="";
29            p+=b1.charAt(i);
30            int q=Integer.valueOf(p);
31            if(q%2==0){
```

Scanned with CamScanner

Attempted: 2/2

```
for(int i=0;i<a1.length();i++){
    String p="";
    p+=b1.charAt(i);
    int q=Integer.valueOf(p);
    if(q%2==0){
        a+=q;
    }
    else{
        b+=q;
    }
}
for(int i=0;i<a1.length();i++){
    String p="";
    p+=c1.charAt(i);
    int q=Integer.valueOf(p);
    if(q%2==0){
        a+=q;
    }
    else{
```

Compile and Test

Submit C

Use Custom Input

Scanned with CamScanner

Attempted: 2/2

```
            p+=c1.charAt(i);
            int q=Integer.valueOf(p);
            if(q%2==0){
                a+=q;
            }
            else{
                b+=q;
            }
        }
        if(input4%2==0){
            return a-b;
        }
        else{
            return b-a;
        }


    }
}
```

Use Custom Input

Compile and Test

Submit C

Scanned with CamScanner

How to Attempt?

Find first repeated element in the given array from the tail of the array:
Rajeev works in the data center lab of the survey department. He has been assigned the task of identifying "repeated numbers" in a given set of numbers. He approaches you to help him achieve this.
Given an array of numbers, your task is to return the first number that repeats (appears again) starting from the last index.

For example -
If input1 = {1,2,4,1,2,8} representing the given array, and
input2 = 6 representing the number of elements in the array,
then the result should be 2 which is the first repeated number in the array from the rear end.

Special conditions to be taken care:
Note 1: You should ignore the negative numbers and zeros. The program should consider only non-zero, non-negative numbers from the given array.
Note 2: If no number is repeated then the output should be the last element of the array.
Note 3: If all elements in the array are negative or 0's the output should be 0.

```
1   import j
2   import
3
4   //
5   cl
6   {
7
8       pub
9
10
11
12      }
13  }
```

Use Custom

Scanned with CamScanner

JAVA7 | Compiler: Java - 1.7

```java
        int o=0;
        int q=0;
        int w=0;
        for(int i=input2-1;i>-1;i--){
            if(input1[i]>0){
            o=1;
            for(int j=i-1;j>-1;j--){
                if(input1[i]==input1[j] && j>=q){
                    System.out.println("LL");
                    w=input1[i];
                    q=j;
                }
            }
            }
        }
        System.out.println(w+" "+q);
        if(o==0){
            return 0;
        }
        if(w!=0){
            return w;
        }
```

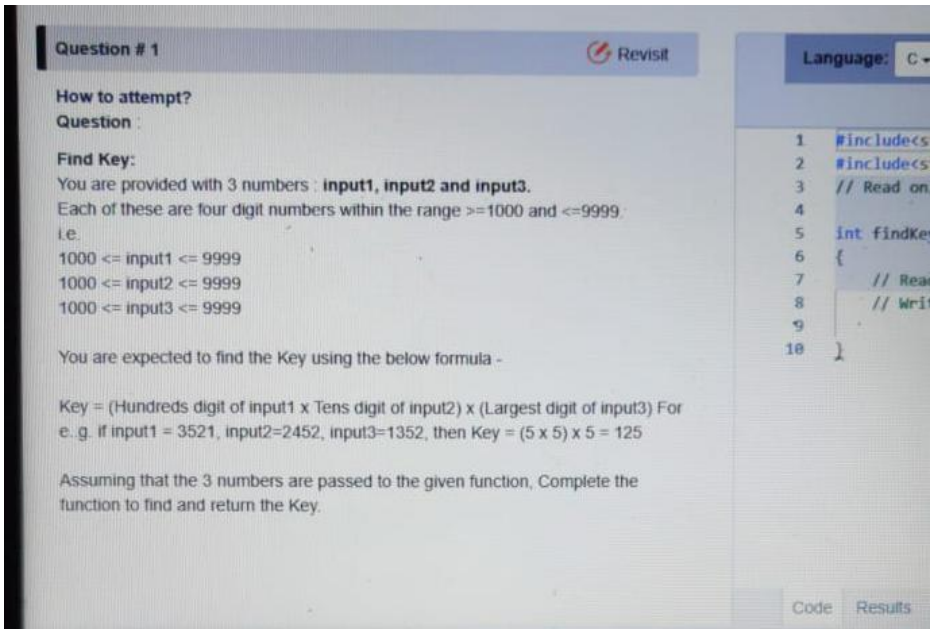
Scanned with CamScanner

```
            }
        }
        }
    }
    System.out.println(w+" "+q);
    if(o==0){
        return 0;
    }
    if(w!=0){
        return w;
    }
    return input1[input2-1];
}
}
```

Compile and Test

Submit

Scanned with CamScanner




```java
public class sample {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc=new Scanner(System.in);
		int input1=sc.nextInt();
		int input2=sc.nextInt();
		int input3=sc.nextInt();
		int in1[]=new int[4];
		String s1=input1+"";
		for(int i=0;i<4;i++)
		{
			in1[i]=Integer.parseInt(s1.charAt(i)+"");
		}
		int in2[]=new int[4];
		String s2=input2+"";
		for(int i=0;i<4;i++)
		{
			in2[i]=Integer.parseInt(s2.charAt(i)+"");
		}
		int in3[]=new int[4];
		String s3=input3+"";
		for(int i=0;i<4;i++)
		{
			in3[i]=Integer.parseInt(s3.charAt(i)+"");
		}
		int key=0;
		Arrays.sort(in3);
		key=in1[1]*in2[2]*in3[3];
		System.out.println(key);
	}
}
```

Question # 1 Revisit

How to attempt?

Question

Find Key:

You are provided with 3 numbers : **input1, input2 and input3.**

Each of these are 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 -

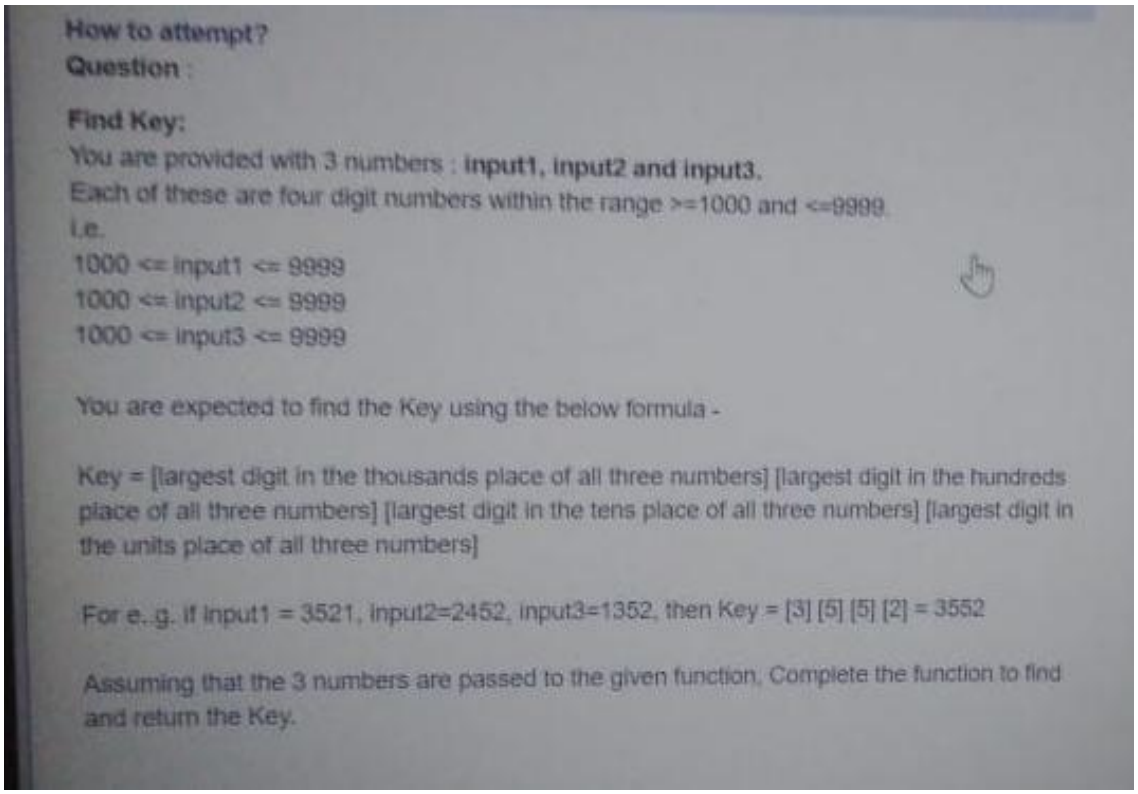
Key = (Sum of Largest digits of all the 3 numbers) + (Sum of Smallest digits of all the 3 numbers) For e. g. if input1 = 3521, input2=2452, input3=1352, then Key = (5+5+5) + (1+2+1) = 19

Assuming that the 3 numbers are passed to the given function. Complete the function to find and return the Key

```
public class sample {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc=new Scanner(System.in);
		int input1=sc.nextInt();
		int input2=sc.nextInt();
		int input3=sc.nextInt();
		int in1[]=new int[4];
		String s1=input1+"";
		for(int i=0;i<4;i++)
		{
			in1[i]=Integer.parseInt(s1.charAt(i)+"");
		}
		int in2[]=new int[4];
		String s2=input2+"";
		for(int i=0;i<4;i++)
		{
			in2[i]=Integer.parseInt(s2.charAt(i)+"");
		}
		int in3[]=new int[4];
		String s3=input3+"";
		for(int i=0;i<4;i++)
		{
			in3[i]=Integer.parseInt(s3.charAt(i)+"");
		}
		int key=0;
		Arrays.sort(in3);
		Arrays.sort(in2);
		Arrays.sort(in1);
		key=(in1[3]+in2[3]+in3[3])+(in1[0]+in2[0]+in3[0]);
```

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



How to attempt?

Question :

Find Key:

You are provided with 3 numbers : **input1, input2 and input3.**

Each of these are 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 = [largest digit in the thousands place of all three numbers] [largest digit in the hundreds place of all three numbers] [largest digit in the tens place of all three numbers] [largest digit in the units place of all three numbers]

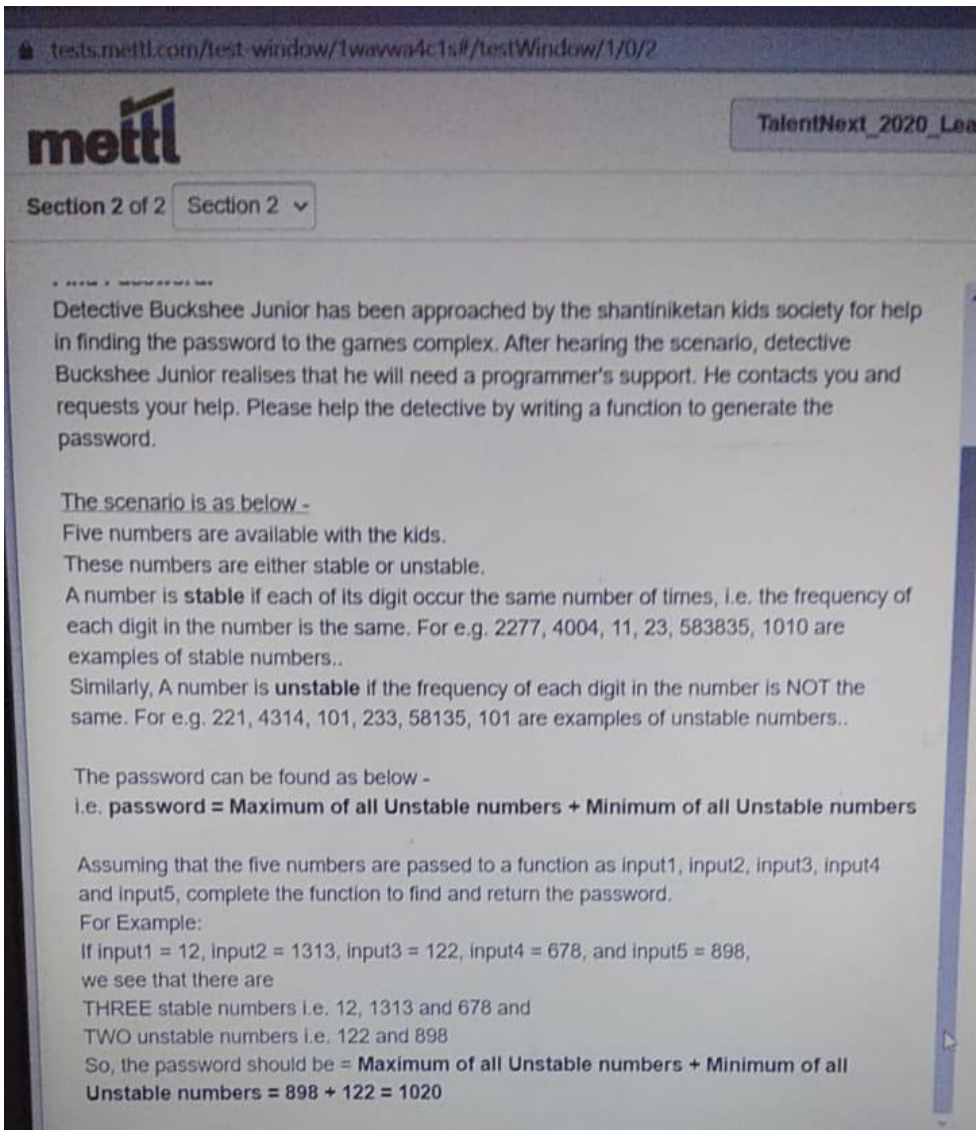
For e. g. if input1 = 3521, input2=2452, input3=1352, then Key = [3] [5] [5] [2] = 3552

Assuming that the 3 numbers are passed to the given function. Complete the function to find and return the Key.

```
public class sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc=new Scanner(System.in);
        int input1=sc.nextInt();
        int input2=sc.nextInt();
        int input3=sc.nextInt();
        int in1[]=new int[4];
        String s1=input1+"";
        for(int i=0;i<4;i++)
        {
            in1[i]=Integer.parseInt(s1.charAt(i)+"");
        }
        int in2[]=new int[4];
        String s2=input2+"";
        for(int i=0;i<4;i++)
        {
            in2[i]=Integer.parseInt(s2.charAt(i)+"");
        }
        int in3[]=new int[4];
        String s3=input3+"";
        for(int i=0;i<4;i++)
        {
            in3[i]=Integer.parseInt(s3.charAt(i)+"");
        }
```

```
            String key="";
            int th1000=in1[0] >in2[0] ? ( in1[0] > in3[0] ? in1[0] : in3[0]) :
(in2[0] > in3[0] ? in2[0] : in3[0]);
            int hun100=in1[1] > in2[1] ? ( in1[1] > in3[1] ? in1[1] : in3[1]) :
(in2[1] > in3[1] ? in2[1] : in3[1]);
            int tens10=in1[2] > in2[2] ? ( in1[2] > in3[2] ? in1[2] : in3[2]) :
(in2[2] > in3[2] ? in2[2] : in3[2]);
            int once1=in1[3] > in2[3] ? ( in1[3] > in3[3] ? in1[3] : in3[3]) :
(in2[3] > in3[3] ? in2[3] : in3[3]);
            key=key+th1000+hun100+tens10+once1;
            int res=Integer.parseInt(key);
            System.out.println(res);
      }
}
```




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

```
// TODO Auto-generated method stub
Scanner sc=new Scanner(System.in);
int input1=sc.nextInt();
int input2=sc.nextInt();
int input3=sc.nextInt();
int input4=sc.nextInt();
int input5=sc.nextInt();
int n1=input1;
int n2=input2;
int n3=input3;
int n4=input4;
int n5=input5;
int in1[]=new int[10];
int in2[]=new int[10];
int in3[]=new int[10];
int in4[]=new int[10];
int in5[]=new int[10];
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
```

```
}
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
	unstable[m]=n2;
	m++;
}
if(stable_check(in3)==1)
{
	stable[l]=n3;
	l++;
}
else
{
	unstable[m]=n3;
	m++;
}
if(stable_check(in4)==1)
{
	stable[l]=n4;
	l++;
}
else
{
	unstable[m]=n4;
	m++;
}
if(stable_check(in5)==1)
{
	stable[l]=n5;
	l++;
}
else
{
	unstable[m]=n5;
	m++;
}
Arrays.sort(unstable);
int max_uns=unstable[4];
int min_uns=0;
for(int i=0;i<5;i++)
{
	if(unstable[i]>0)
	{
		min_uns=unstable[i];
		break;
	}
}
System.out.println(max_uns);
System.out.println(min_uns);
int res=max_uns+min_uns;
```

```
            System.out.println(res);
        }
        static int stable_check(int in[])
        {
            Arrays.sort(in);

            int c=in[9];
            int flag=1;
            for(int i=0;i<10;i++)
            {
                if(in[i]==0||in[i]==c)
                {
                    flag=1;
                }
                else{
                    flag=0;
                    break;
                }
            }
            return flag;
        }
}
```

Question :

Find Password:

Detective Buckshee Junior has been approached by the shantiniketan kids society for help in finding the password to the games complex. After hearing the scenario, detective Buckshee Junior realises that he will need a programmer's support. He contacts you and requests your help. Please help the detective by writing a function to generate the password.

The scenario is as below -

Five numbers are available with the kids.

These numbers are either stable or unstable.

A number is **stable** if each of its digit occur the same number of times, i.e. the frequency of each digit in the number is the same. For e.g. 2277, 4004, 11, 23, 583835, 1010 are examples of stable numbers.

Similarly, A number is **unstable** if the frequency of each digit in the number is NOT the same. For e.g. 221, 4314, 101, 233, 58135, 101 are examples of unstable numbers..

The password can be found as below -

i.e. **password = Maximum of all stable numbers + Minimum of all Unstable numbers**

Assuming that the five numbers are passed to a function as input1, input2, input3, input4 and input5, complete the function to find and return the password.

For Example:

If input1 = 12, input2 = 1313, input3 = 122, input4 = 678, and input5 = 898,

stable numbers are 12, 1313 and 678

unstable numbers are 122 and 898

So, the password should be = **Maximum of all stable numbers + Minimum of all Unstable numbers = 1313 + 122 = 1435**

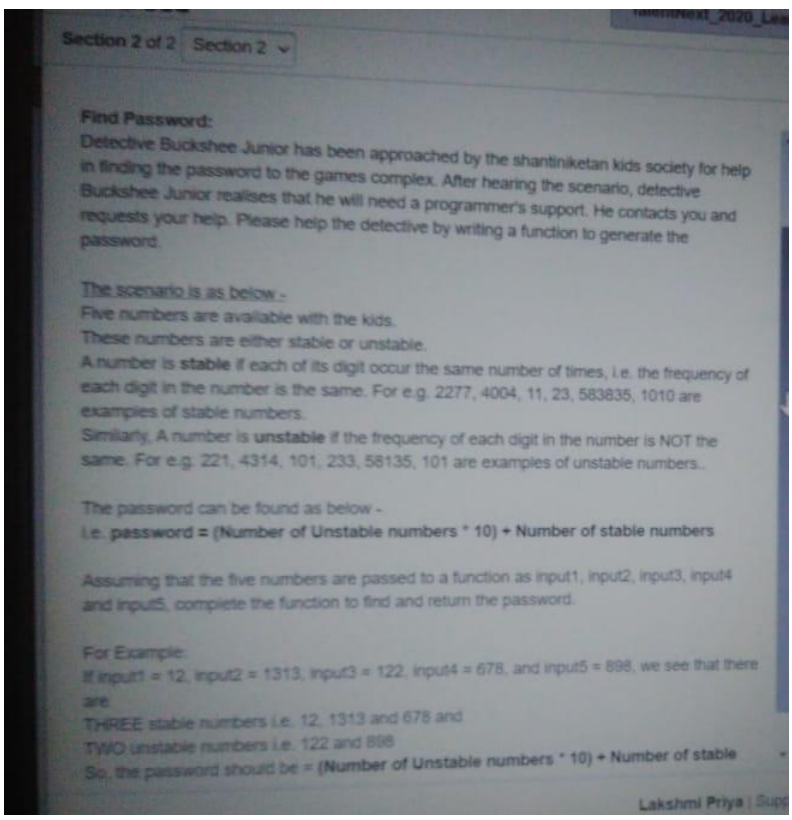
```
public class sample {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
Scanner sc=new Scanner(System.in);
int input1=sc.nextInt();
int input2=sc.nextInt();
int input3=sc.nextInt();
int input4=sc.nextInt();
int input5=sc.nextInt();
int n1=input1;
int n2=input2;
int n3=input3;
int n4=input4;
int n5=input5;
int in1[]=new int[10];
int in2[]=new int[10];
int in3[]=new int[10];
int in4[]=new int[10];
int in5[]=new int[10];
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
}
```

```
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
	unstable[m]=n2;
	m++;
}
if(stable_check(in3)==1)
{
	stable[l]=n3;
	l++;
}
else
{
	unstable[m]=n3;
	m++;
}
if(stable_check(in4)==1)
{
	stable[l]=n4;
	l++;
}
else
{
	unstable[m]=n4;
	m++;
}
if(stable_check(in5)==1)
{
	stable[l]=n5;
	l++;
}
else
{
	unstable[m]=n5;
	m++;
}
Arrays.sort(stable);
Arrays.sort(unstable);
int max_s=stable[4];
int min_uns=0;
for(int i=0;i<5;i++)
{
	if(unstable[i]>0)
	{
		min_uns=unstable[i];
		break;
	}
}
System.out.println(max_s);
System.out.println(min_uns);
int res=max_s+min_uns;
```

```
            System.out.println(res);
        }
        static int stable_check(int in[])
        {
            Arrays.sort(in);

            int c=in[9];
            int flag=1;
            for(int i=0;i<10;i++)
            {
                if(in[i]==0||in[i]==c)
                {
                    flag=1;
                }
                else{
                    flag=0;
                    break;
                }
            }
            return flag;
        }
}
```



Section 2 of 2 Section 2

Find Password:

Detective Buckshee Junior has been approached by the shantiniketan kids society for help in finding the password to the games complex. After hearing the scenario, detective Buckshee Junior realises that he will need a programmer's support. He contacts you and requests your help. Please help the detective by writing a function to generate the password.

The scenario is as below -

Five numbers are available with the kids.

These numbers are either stable or unstable.

A number is **stable** if each of its digit occur the same number of times, i.e. the frequency of each digit in the number is the same. For e.g. 2277, 4004, 11, 23, 583835, 1010 are examples of stable numbers.

Similarly, A number is **unstable** if the frequency of each digit in the number is NOT the same. For e.g. 221, 4314, 101, 233, 58135, 101 are examples of unstable numbers..

The password can be found as below -

i.e. **password = (Number of Unstable numbers * 10) + Number of stable numbers**

Assuming that the five numbers are passed to a function as input1, input2, input3, input4 and input5, complete the function to find and return the password.

For Example:

If input1 = 12, input2 = 1313, input3 = 122, input4 = 678, and input5 = 898, we see that there are

THREE stable numbers i.e. 12, 1313 and 678 and

TWO unstable numbers i.e. 122 and 898

So, the password should be = **(Number of Unstable numbers * 10) + Number of stable**

Lakshmi Priya

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

```
// TODO Auto-generated method stub
Scanner sc=new Scanner(System.in);
int input1=sc.nextInt();
int input2=sc.nextInt();
int input3=sc.nextInt();
int input4=sc.nextInt();
int input5=sc.nextInt();
int n1=input1;
int n2=input2;
int n3=input3;
int n4=input4;
int n5=input5;
int in1[]=new int[10];
int in2[]=new int[10];
int in3[]=new int[10];
int in4[]=new int[10];
int in5[]=new int[10];
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
```

```
}
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
	unstable[m]=n2;
	m++;
}
if(stable_check(in3)==1)
{
	stable[l]=n3;
	l++;
}
else
{
	unstable[m]=n3;
	m++;
}
if(stable_check(in4)==1)
{
	stable[l]=n4;
	l++;
}
else
{
	unstable[m]=n4;
	m++;
}
if(stable_check(in5)==1)
{
	stable[l]=n5;
	l++;
}
else
{
	unstable[m]=n5;
	m++;
}
int S_count=0;
int uns_count=0;
for(int i=0;i<5;i++)
{
	if(stable[i]>0)
	{
		S_count++;
	}
}
for(int i=0;i<5;i++)
{
	if(unstable[i]>0)
	{
		uns_count++;
```

```
                }
            }
            System.out.println(S_count);
            System.out.println(uns_count);
            int res=(S_count*10)+uns_count;
            System.out.println(res);
        }
        static int stable_check(int in[])
        {
            Arrays.sort(in);

            int c=in[9];
            int flag=1;
            for(int i=0;i<10;i++)
            {
                if(in[i]==0||in[i]==c)
                {
                    flag=1;
                }
                else{
                    flag=0;
                    break;
                }
            }
            return flag;
        }
    }
```

The scenario is as below -

Five numbers are available with the kids.

These numbers are either stable or unstable.

A number is **stable** if each of its digit occur the same number of times, i.e. the frequency of each digit in the number is the same. For e.g. 2277, 4004, 11, 23, 583835, 1010 are examples of stable numbers.

Similarly, A number is **unstable** if the frequency of each digit in the number is NOT the same. For e.g. 221, 4314, 101, 233, 58135, 101 are examples of unstable numbers..

The password can be found as below -

i.e. **password = Maximum of all stable numbers + Minimum of all stable numbers**

Assuming that the five numbers are passed to a function as input1, input2, input3, input4 and input5, complete the function to find and return the password.

For Example:

If input1 = 12, input2 = 1313, input3 = 122, input4 = 678, and input5 = 898,

stable numbers are 12, 1313 and 678

unstable numbers are 122 and 898

So, the password should be = **Maximum of all stable numbers + Minimum of all stable numbers = 1313 + 12 = 1325**

Dharani M | Support

mettl

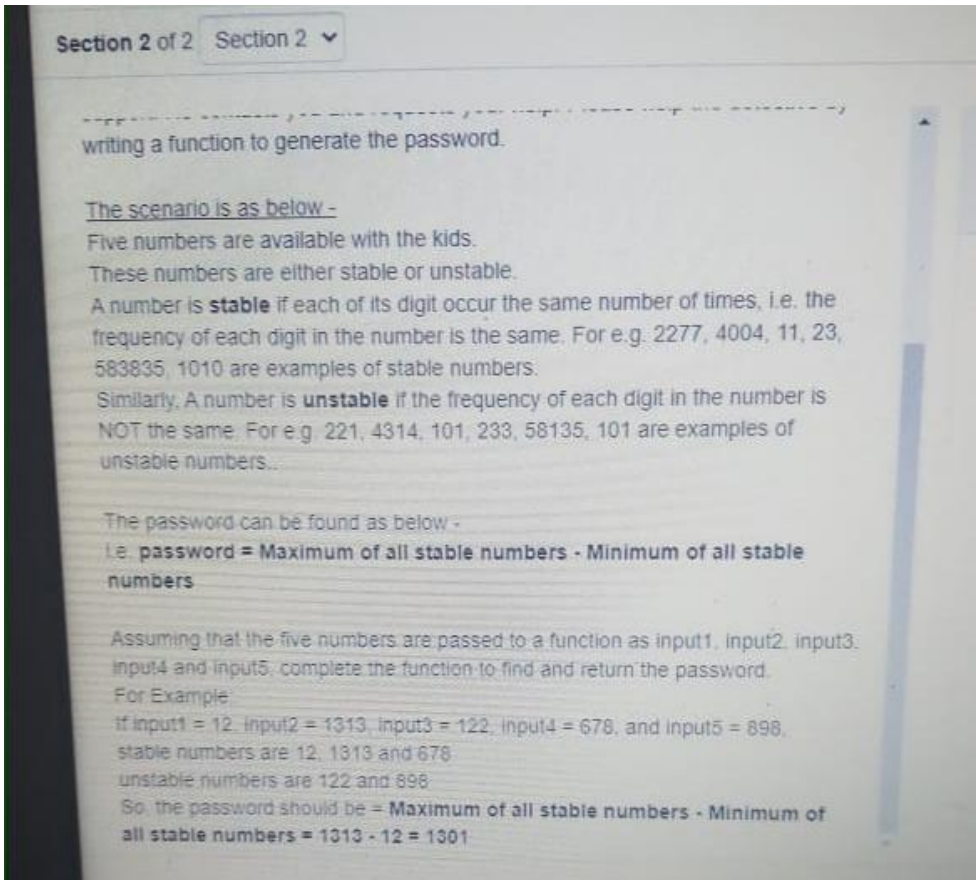
```
public class sample {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc=new Scanner(System.in);
		int input1=sc.nextInt();
		int input2=sc.nextInt();
		int input3=sc.nextInt();
		int input4=sc.nextInt();
		int input5=sc.nextInt();
		int n1=input1;
		int n2=input2;
		int n3=input3;
		int n4=input4;
		int n5=input5;
		int in1[]=new int[10];
		int in2[]=new int[10];
		int in3[]=new int[10];
		int in4[]=new int[10];
```

```
int in5[]=new int[10];
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
}
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
	unstable[m]=n2;
	m++;
}
if(stable_check(in3)==1)
{
	stable[l]=n3;
	l++;
}
```

```
        else
        {
                unstable[m]=n3;
                m++;
        }
        if(stable_check(in4)==1)
        {
                stable[l]=n4;
                l++;
        }
        else
        {
                unstable[m]=n4;
                m++;
        }
        if(stable_check(in5)==1)
        {
                stable[l]=n5;
                l++;
        }
        else
        {
                unstable[m]=n5;
                m++;
        }
        Arrays.sort(stable);
        int max_s=stable[4];
        int min_s=0;
        for(int i=0;i<5;i++)
        {
                if(stable[i]>0)
                {
                        min_s=stable[i];
                        break;
                }
        }
        System.out.println(max_s);
        System.out.println(min_s);
        int res=max_s+min_s;
        System.out.println(res);
}
static int stable_check(int in[])
{
        Arrays.sort(in);

        int c=in[9];
        int flag=1;
        for(int i=0;i<10;i++)
        {
                if(in[i]==0||in[i]==c)
                {
                        flag=1;
                }
                else{
                        flag=0;
```

```
                        break;
                    }
                }
                return flag;
        }
}
```



Section 2 of 2 Section 2

writing a function to generate the password.

The scenario is as below -
Five numbers are available with the kids.
These numbers are either stable or unstable.
A number is **stable** if each of its digit occur the same number of times, i.e. the frequency of each digit in the number is the same. For e.g. 2277, 4004, 11, 23, 583835, 1010 are examples of stable numbers.
Similarly, A number is **unstable** if the frequency of each digit in the number is NOT the same. For e.g. 221, 4314, 101, 233, 58135, 101 are examples of unstable numbers..

The password can be found as below -
i.e. **password = Maximum of all stable numbers - Minimum of all stable numbers**

Assuming that the five numbers are passed to a function as input1, input2, input3, input4 and input5, complete the function to find and return the password.
For Example
If input1 = 12, input2 = 1313, input3 = 122, input4 = 678, and input5 = 898,
stable numbers are 12, 1313 and 678
unstable numbers are 122 and 898
So, the password should be = **Maximum of all stable numbers - Minimum of all stable numbers = 1313 - 12 = 1301**

```
public class sample {
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Scanner sc=new Scanner(System.in);
                int input1=sc.nextInt();
                int input2=sc.nextInt();
                int input3=sc.nextInt();
                int input4=sc.nextInt();
                int input5=sc.nextInt();
                int n1=input1;
                int n2=input2;
                int n3=input3;
                int n4=input4;
                int n5=input5;
                int in1[]=new int[10];
                int in2[]=new int[10];
                int in3[]=new int[10];
                int in4[]=new int[10];
                int in5[]=new int[10];
```

```
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
}
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
	unstable[m]=n2;
	m++;
}
if(stable_check(in3)==1)
{
	stable[l]=n3;
	l++;
}
else
```

```
        {
                unstable[m]=n3;
                m++;
        }
        if(stable_check(in4)==1)
        {
                stable[l]=n4;
                l++;
        }
        else
        {
                unstable[m]=n4;
                m++;
        }
        if(stable_check(in5)==1)
        {
                stable[l]=n5;
                l++;
        }
        else
        {
                unstable[m]=n5;
                m++;
        }
        Arrays.sort(stable);
        int max_s=stable[4];
        int min_s=0;
        for(int i=0;i<5;i++)
        {
                if(stable[i]>0)
                {
                        min_s=stable[i];
                        break;
                }
        }
        System.out.println(max_s);
        System.out.println(min_s);
        int res=max_s-min_s;
        System.out.println(res);
}
static int stable_check(int in[])
{
        Arrays.sort(in);

        int c=in[9];
        int flag=1;
        for(int i=0;i<10;i++)
        {
                if(in[i]==0||in[i]==c)
                {
                        flag=1;
                }
                else{
                        flag=0;
                        break;
```

```
                }
            }
            return flag;
        }
}
```

Question :

Find Password:

Detective Buckshee Junior has been approached by the shantiniketan kids society for h in finding the password to the games complex. After hearing the scenario, detective Buckshee Junior realises that he will need a programmer's support. He contacts you an requests your help. Please help the detective by writing a function to generate the password.

The scenario is as below -

Five numbers are available with the kids.

These numbers are either stable or unstable.

A number is **stable** if each of its digit occur the same number of times, i.e. the frequency each digit in the number is the same. For e.g. 2277, 4004, 11, 23, 583835, 1010 are examples of stable numbers.

Similarly, A number is **unstable** if the frequency of each digit in the number is NOT the same. For e.g. 221, 4314, 101, 233, 58135, 101 are examples of unstable numbers..

The password can be found as below -

i.e. **password = Maximum of all stable numbers - Minimum of all Unstable numbers**

Assuming that the five numbers are passed to a function as input1, input2, input3, input4 and input5, complete the function to find and return the password.

For Example:

If input1 = 12, input2 = 1313, input3 = 122, input4 = 678, and input5 = 898,

stable numbers are 12, 1313 and 678

unstable numbers are 122 and 898

So, the password should be = **Maximum of all stable numbers - Minimum of all Unstable numbers = 1313 - 122 = 1191**

```
public class sample {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc=new Scanner(System.in);
        int input1=sc.nextInt();
        int input2=sc.nextInt();
        int input3=sc.nextInt();
        int input4=sc.nextInt();
        int input5=sc.nextInt();
        int n1=input1;
```

```
int n2=input2;
int n3=input3;
int n4=input4;
int n5=input5;
int in1[]=new int[10];
int in2[]=new int[10];
int in3[]=new int[10];
int in4[]=new int[10];
int in5[]=new int[10];
while(input1>0)
{
	in1[input1%10]++;
	input1/=10;
}
while(input2>0)
{
	in2[input2%10]++;
	input2/=10;
}
while(input3>0)
{
	in3[input3%10]++;
	input3/=10;
}
while(input4>0)
{
	in4[input4%10]++;
	input4/=10;
}
while(input5>0)
{
	in5[input5%10]++;
	input5/=10;
}
int stable[]=new int[5];
int l=0;
int unstable[]=new int[5];
int m=0;
if(stable_check(in1)==1)
{
	stable[l]=n1;
	l++;
}
else
{
	unstable[m]=n1;
	m++;
}
if(stable_check(in2)==1)
{
	stable[l]=n2;
	l++;
}
else
{
```

```
                unstable[m]=n2;
                m++;
        }
        if(stable_check(in3)==1)
        {
                stable[l]=n3;
                l++;
        }
        else
        {
                unstable[m]=n3;
                m++;
        }
        if(stable_check(in4)==1)
        {
                stable[l]=n4;
                l++;
        }
        else
        {
                unstable[m]=n4;
                m++;
        }
        if(stable_check(in5)==1)
        {
                stable[l]=n5;
                l++;
        }
        else
        {
                unstable[m]=n5;
                m++;
        }
        Arrays.sort(stable);
        Arrays.sort(unstable);
        int max_s=stable[4];
        int min_uns=0;
        for(int i=0;i<5;i++)
        {
                if(unstable[i]>0)
                {
                        min_uns=unstable[i];
                        break;
                }
        }
        System.out.println(max_s);
        System.out.println(min_uns);
        int res=max_s-min_uns;
        System.out.println(res);
}
static int stable_check(int in[])
{
        Arrays.sort(in);

        int c=in[9];
```

```
		int flag=1;
		for(int i=0;i<10;i++)
		{
			if(in[i]==0||in[i]==c)
			{
				flag=1;
			}
			else{
				flag=0;
				break;
			}
		}
		return flag;
	}
}
```

CODE 1:

```
import java.util.Scanner;


public class Sample1 {

    private static Scanner key = new
Scanner(System.in);


    public static void main(String[] args) {

        int input1 = key.nextInt();

        int input2 = key.nextInt();

        int input3 = key.nextInt();

        key.close();

    System.out.println(findAnswer(input1,input2,
input3));
```

```
    }

    public static int findAnswer(int input1,
int input2, int input3) {

        if (input1 > 999 && input1 <= 9999 &&
input2 > 999 && input2 <= 9999 && input3 > 999
&& input3 <= 9999) {
            int maxOfInput3 = Integer.MIN_VALUE;
            while (input3 > 0) {
                if (input3 % 10 > maxOfInput3)
                    maxOfInput3 = input3 % 10;
                input3 /= 10;
            }
            return ((input1 / 100) % 10) *
((input2 / 10) % 10) * maxOfInput3;
        }
        return 0;
    }
}
```

CODE 2:

```
import java.util.Scanner;


public class Sample2 {

    private static Scanner key = new
Scanner(System.in);


    // code for sum of minimum digit and
maximum digit of all the three numbers

    public static void main(String[] args) {

        int input1 = key.nextInt();

        int input2 = key.nextInt();

        int input3 = key.nextInt();

        key.close();

    System.out.println(sumOfMinAndMaxDigits(inp
ut1, input2, input3));


    }
```

```
    public static int sumOfMinAndMaxDigits(int
input1, int input2, int input3) {
        if (input1 > 999 && input1 <= 9999 &&
input2 > 999 && input2 <= 9999 && input3 > 999
&& input3 <= 9999) {
            int res = 0, min = 9, max = 0, rem =
0;
            while (input1 > 0) {
                rem = input1 % 10;
                max = (max < rem) ? rem : max;
                min = (min > rem) ? rem : min;
                input1 /= 10;
            }
            res += max + min;
            while (input2 > 0) {
                rem = input2 % 10;
                max = (max < rem) ? rem : max;
                min = (min > rem) ? rem : min;
                input2 /= 10;
```

```
            }
            res += max + min;
            while (input3 > 0) {
                rem = input3 % 10;
                max = (max < rem) ? rem : max;
                min = (min > rem) ? rem : min;
                input3 /= 10;
            }
            res += max + min;
            return res;
        }
        return -1;
    }

}
```

CODE 3:

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

```
public class Sample3 {
    private static Scanner key = new
Scanner(System.in);

    // code for sum of maximum digit of all the
places for three numbers
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        key.close();

    System.out.println(maxSumOfAllPlaces(input1,
input2, input3));

    }

    private static int maxSumOfAllPlaces(int
input1, int input2, int input3) {
```

```
        if (input1 > 999 && input1 <= 9999 &&
input2 > 999 && input2 <= 9999 && input3 > 999
&& input3 <= 9999) {
            int res = 0, pv = 1;
            while (input1 > 0) {
                int rem1, rem2, rem3;
                rem1 = input1 % 10;
                rem2 = input2 % 10;
                rem3 = input3 % 10;
                res = res + pv *
Math.max(Math.max(rem1, rem2), rem3);
                pv *= 10;
                input1 /= 10;
                input2 /= 10;
                input3 /= 10;
            }
            return res;
        }
        return -1;
    }
```

```
}
```

CODE 4:

```
import java.util.Scanner;

public class Sample4 {
    private static Scanner key = new
Scanner(System.in);

    // code for sum of maximum and minimum
number of all unstable number
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
```

```
        key.close();

    System.out.println(maxAndMinSumOfUnstableNu
mber(input1, input2, input3, input4, input5));


    }


    private static int
maxAndMinSumOfUnstableNumber(int input1, int
input2, int input3, int input4, int input5) {
        int[] inputs = { input1, input2, input3,
input4, input5 };
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int unstableCount = 0;
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;


            while (value > 0) {
```

```
                counts[value % 10]++;
                value /= 10;
            }
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
            }
            for (int count : counts) {
                if (count != 0 && count != value)
{
                    isStable = false;
                    break;
                }
            }
            if (!isStable) {
                min = min > input ? input : min;
```

```
                max = max < input ? input : max;
                unstableCount++;
            }
        }
        if (unstableCount > 0)
            return min + max;
        return 0;
    }
}
```

CODE 5:

```
import java.util.Scanner;

public class Sample5 {
    private static Scanner key = new
Scanner(System.in);
```

```
    // code for sum of maximum of stable number
+ minimum number of unstable
    // number
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
        key.close();

    System.out.println(sumOfMaxStableAndMinUnst
able(input1, input2, input3, input4, input5));


    }


    private static int
sumOfMaxStableAndMinUnstable(int input1, int
input2, int input3, int input4, int input5) {
        int[] inputs = { input1, input2, input3,
input4, input5 };
```

```
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int unstableCount = 0;
        int stableCount = 0;
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;

            while (value > 0) {
                counts[value % 10]++;
                value /= 10;
            }
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
```

```
            }
            for (int count : counts) {
                if (count != 0 && count != value)
{
                    isStable = false;
                    break;
                }
            }
            if (isStable) {
                max = max < input ? input : max;
                stableCount++;
            }
            if (!isStable) {

                min = min > input ? input : min;
                unstableCount++;
            }
        }
        if (stableCount > 0 && unstableCount > 0)
```

```
            return min + max;
        if(stableCount==0)
            return min;
        if(unstableCount==0)
            return max;
        return 0;
    }
}
```

CODE 6:

```
import java.util.Scanner;

public class Sample6 {
    private static Scanner key = new
Scanner(System.in);

    // code for  (number of unstable
number)*10+ (number of stable number)
```

```
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
        key.close();

    System.out.println(countOfUnstableAndStable
(input1, input2, input3, input4, input5));


    }


    private static int
countOfUnstableAndStable(int input1, int input2,
int input3, int input4, int input5) {
        int[] inputs = { input1, input2, input3,
input4, input5 };


        int unstableCount = 0;
```

```
        int stableCount = 0;
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;

            while (value > 0) {
                counts[value % 10]++;
                value /= 10;
            }
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
            }
            for (int count : counts) {
```

```
                if (count != 0 && count != value)
{
                    isStable = false;
                    break;
                }
            }
            if (isStable) {
                stableCount++;
            }
            if (!isStable) {
                unstableCount++;
            }
        }

        return unstableCount*10+stableCount;
    }
}
```

CODE 7:

```
import java.util.Scanner;


public class Sample7 {
    private static Scanner key = new
Scanner(System.in);


    // code for sum of maximum and minimum
number of all stable number
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
        key.close();

    System.out.println(maxAndMinSumOfStableNumb
er(input1, input2, input3, input4, input5));
```

```
    }

    private static int
maxAndMinSumOfStableNumber(int input1, int
input2, int input3, int input4, int input5) {
        int[] inputs = { input1, input2, input3,
input4, input5 };
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int stableCount = 0;
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;

            while (value > 0) {
                counts[value % 10]++;
                value /= 10;
            }
```

```
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
            }
            for (int count : counts) {
                if (count != 0 && count != value)
{
                    isStable = false;
                    break;
                }
            }
            if (isStable) {
                min = min > input ? input : min;
                max = max < input ? input : max;
                stableCount++;
            }
```

```
        }
        if (stableCount > 0)
            return max + min;
        return 0;
    }
}
```

CODE 8:

```
import java.util.Scanner;

public class Sample8 {
    private static Scanner key = new
Scanner(System.in);

    // code for subtraction of maximum and
minimum number of all stable number
    public static void main(String[] args) {
        int input1 = key.nextInt();
```

```
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
        key.close();

    System.out.println(maxAndMinSubtractionOfStableNumber(input1, input2, input3, input4, input5));


    }


    private static int maxAndMinSubtractionOfStableNumber(int input1, int input2, int input3, int input4, int input5) {
        int[] inputs = { input1, input2, input3, input4, input5 };
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int stableCount = 0;
```

```
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;

            while (value > 0) {
                counts[value % 10]++;
                value /= 10;
            }
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
            }
            for (int count : counts) {
                if (count != 0 && count != value)
{
```

```
                    isStable = false;
                    break;
                }
            }
            if (isStable) {
                min = min > input ? input : min;
                max = max < input ? input : max;
                stableCount++;
            }
        }
        if (stableCount > 0)
            return  max-min;
        return 0;
    }
}
```

CODE 9:

```java
import java.util.Scanner;


public class Sample9 {
    private static Scanner key = new
Scanner(System.in);


    // code for subtraction of maximum of
stable number - minimum number of unstable
    // number
    public static void main(String[] args) {
        int input1 = key.nextInt();
        int input2 = key.nextInt();
        int input3 = key.nextInt();
        int input4 = key.nextInt();
        int input5 = key.nextInt();
        key.close();

    System.out.println(subtractionOfMaxStableAn
dMinUnstable(input1, input2, input3, input4,
input5));
```

```
    }

    private static int
subtractionOfMaxStableAndMinUnstable(int input1,
int input2, int input3, int input4, int input5)
{
        int[] inputs = { input1, input2, input3,
input4, input5 };
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int unstableCount = 0;
        int stableCount = 0;
        for (int input : inputs) {
            int[] counts = { 0, 0, 0, 0, 0, 0, 0,
0, 0, 0 };
            int value = input;

            while (value > 0) {
                counts[value % 10]++;
```

```
                value /= 10;
            }
            boolean isStable = true;
            for (int count : counts) {
                if (count != 0) {
                    value = count;
                    break;
                }
            }
            for (int count : counts) {
                if (count != 0 && count != value)
{
                    isStable = false;
                    break;
                }
            }
            if (isStable) {
                max = max < input ? input : max;
                stableCount++;
```

```
            }
            if (!isStable) {

                min = min > input ? input : min;
                unstableCount++;
            }
        }
        if (stableCount > 0 && unstableCount > 0)
            return max - min;
        if (stableCount == 0)
            return -min;
        if (unstableCount == 0)
            return max;
        return 0;
    }
}
```