JAVA WEEK 5

Given an array A of positive integers, let S be the sum of the digits of the minimal element of A.

Return 0 if S is odd, otherwise return 1.

Example 1:

Input:

8

34 23 1 24 75 33 54 8

Output:

0

Explanation:

The minimal element is 1, and the sum of those digits is S = 1 which is odd, so the answer is 0.

Example 2:

Input:

5

99 77 33 66 55

Output:

1

Explanation:

The minimal element is 33, and the sum of those digits is S = 3 + 3 = 6 which is even, so the answer is 1.

Constraints:

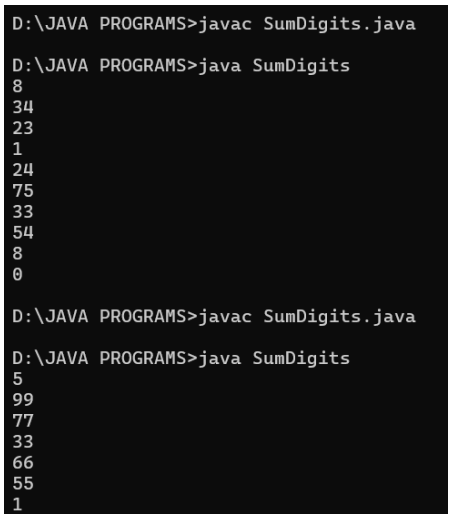
- 1 <= A.length <= 100
- 1 <= A[i] <= 100

CODE:

```
import java.util.*;
class SumDigits{
public static void main(String args[])
{
Scanner obj= new Scanner(System.in);
int n=obj.nextInt();
int sum=0;
int[] arr=new int[n];
for(int i=0;i<n;i++){
arr[i]=obj.nextInt();
}
Arrays.sort(arr);
while (arr[0] > 0) {
sum +=arr[0] % 10;
arr[0] /= 10;
}
```

```
if (sum%2==0){
System.out.println("1");
}else{
System.out.println("0");
}
}
}
```

OUTPUT:



You are provided with a set of numbers (array of numbers).

You have to generate the sum of specific numbers based on its position in the array set provided to you.

This is explained below:

Example 1:

Let us assume the encoded set of numbers given to you is:

input1:5 and input2: {1, 51, 436, 7860, 41236}

Step 1:

Starting from the 0^{th} index of the array pick up digits as per below:

0^{th} index – pick up the units value of the number (in this case is 1).

1^{st} index - pick up the tens value of the number (in this case it is 5).

2^{nd} index - pick up the hundreds value of the number (in this case it is 4).

3^{rd} index - pick up the thousands value of the number (in this case it is 7).

4^{th} index - pick up the ten thousands value of the number (in this case it is 4).

(Continue this for all the elements of the input array).

The array generated from Step 1 will then be – {1, 5, 4, 7, 4}.

Step 2:

Square each number present in the array generated in Step 1.

{1, 25, 16, 49, 16}

Step 3:

Calculate the sum of all elements of the array generated in Step 2 to get the final result. The result will be = 107.

Note:

1) While picking up a number in Step1, if you observe that the number is smaller than the required position then use 0.
2) In the given function, input1[] is the array of numbers and input2 represents the number of elements in input1.

Example 2:

input1: 5 and input1: {1, 5, 423, 310, 61540}

Step 1:

Generating the new array based on position, we get the below array:

{1, 0, 4, 0, 6}

In this case, the value in input1 at index 1 and 3 is less than the value required to be picked up based on position, so we use a 0.

Step 2:

{1, 0, 16, 0, 36}

Step 3:

The final result = 53.

For example:

Input	Result
5 1 51 436 7860 41236	107
5	53

```
import java.util.*;

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

```
            int number = a[i];
            int index = i;
            int digit = 0;
            for (int j = 0; j <= index; j++) {
                digit = number % 10;
                number = number / 10;
            }

            sum=sum+(digit*digit);
        }
        System.out.println(sum);
    }
}
```

```
D:\java>javac DigitSum.java

D:\java>java DigitSum
5
1
51
436
7860
41236
107
```

The program must accept **N** integers and an integer **K** as the input. The program must print every K integers in descending order as the output.

Note: If **N % K != 0**, then sort the final N%K integers in descending order.

Boundary Condition(s):

1 <= N <= 10^4
-99999 <= Array Element Value <= 99999

Input Format:

The first line contains the values of N and K separated by a space.
The second line contains N integers separated by space(s).

Output Format:

The first line contains N integers.

Example Input/Output 1:

Input:

7 3
48 541 23 68 13 41 6

Output:

541 48 23 68 41 13 6

Explanation:

The first three integers are 48 541 23, after sorting in descending order the integers are **541 48 23**.
The second three integers are 68 13 41, after sorting in descending order the integers are **68 41 13**.
The last integer is **6**.
The integers are **541 48 23 68 41 13 6**.
Hence the output is **541 48 23 68 41 13 6**.

```
import java.util.* ;
class DividenSort {
public static void main (String args []){
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int k = sc.nextInt();
int[] a = new int[n];
for (int i = 0 ; i<n ; i++ ){
a[i] = sc.nextInt();
}
for (int i = 0; i < n; i += k) {
int end = Math.min(i + k, n);
for (int j = i; j < end - 1; j++) {
            for (int l = j + 1; l < end; l++) {
               if (a[j] < a[l]) {
                  int temp = a[j];
                  a[j] = a[l];
                  a[l] = temp;
               }
            }
         }
      }

      for (int i = 0; i < n; i++) {
         System.out.print(a[i] + " ");
      }
      sc.close();
   }
}
```

```
E:\>javac DividenSort.java

E:\>java DividenSort
7
3
48
541
23
68
13
41
6
541 48 23 68 41 13 6
E:\>
```