Codility_

Candidate Report: training59CHQH-Y6J

Check out Codility training tasks

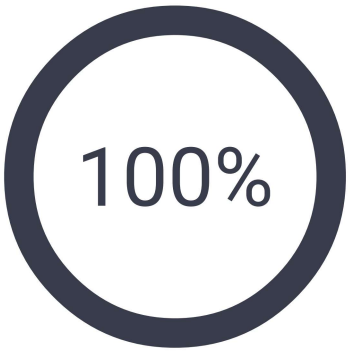
Test Name:

Summary Review (0) Timeline

Tasks summary

Task	Time spent	Score
MinMaxDivision Java 8	18 min	100%

Total score



Tasks Details

Medium

1. MinMaxDivision

Divide array A into K blocks and minimize the largest sum of any block.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are given integers K, M and a non-empty array A consisting of N integers. Every element of the array is not greater than M.

You should divide this array into K blocks of consecutive elements. The size of the block is any integer between 0 and N. Every element of the array should belong to some block.

The sum of the block from X to Y equals A[X] + A[X + 1] + ... + A[Y]. The sum of empty block equals 0.

The *large sum* is the maximal sum of any block.

For example, you are given integers K = 3, M = 5 and array A such that:

```
A[0] = 2
A[1] = 1
A[2] = 5
A[3] = 1
A[4] = 2
A[5] = 2
A[6] = 2
```

The array can be divided, for example, into the following blocks:

- [2, 1, 5, 1, 2, 2, 2], [], [] with a large sum of 15;
- [2], [1, 5, 1, 2], [2, 2] with a large sum of 9;
- [2, 1, 5], [], [1, 2, 2, 2] with a large sum of 8;
- [2, 1], [5, 1], [2, 2, 2] with a large sum of 6.

The goal is to minimize the large sum. In the above example, 6 is the minimal large sum.

Solution

Programming language used:	Java 8
Total time used:	18 minutes
Effective time used:	18 minutes
Notes:	*not defined yet*

Task timeline

18:11:02 18:28:04

Code: 18:28:04 UTC, java, final, score: **100** show code in pop-up

```
1  // you can also use imports, for example:
2  // import java.util.*;
3
4  // you can write to stdout for debugging purposes, e.g.
5  // System.out.println("this is a debug message");
6
7  class Solution {
8      public int solution(int K, int M, int[] A) {
```

Write a function:

```
class Solution { public int solution(int K, int M, int[] A); }
```

that, given integers K, M and a non-empty array A consisting of N integers, returns the minimal large sum.

For example, given K = 3, M = 5 and array A such that:

```
A[0] = 2
A[1] = 1
A[2] = 5
A[3] = 1
A[4] = 2
A[5] = 2
A[6] = 2
```

the function should return 6, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N and K are integers within the range [1..100,000];
- M is an integer within the range [0..10,000];
- each element of array A is an integer within the range [0..M].


```
9           int min = 0;
10          int max = 0;
11          for (int i = 0; i < A.length; i++) {//get the sum
12              max += A[i];
13              min = Math.max(min, A[i]);
14          }
15          int result = max;
16          while (min <= max) {
17              int mid = (min + max) / 2;
18              if (divisionSolvable(mid, K - 1, A)) {
19                  max = mid - 1;
20                  result = mid;
21              } else {
22                  min = mid + 1;
23              }
24          }
25          return result;
26      }
27      private boolean divisionSolvable(int mid, int k, int[]
28          int sum = 0;
29          for (int i = 0; i < a.length; i++) {
30              sum += a[i];
31              if (sum > mid) {
32                  sum = a[i];
33                  k--;
34              }
35              if (k < 0) {
36                  return false;
37              }
38          }
39          return true;
40      }
41  }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N*log(N+M))**

expand all	Example tests	
▶	example example test	✓ OK
expand all	Correctness tests	
▶	extreme_single single elements	✓ OK
▶	extreme_double single and double elements	✓ OK
▶	extreme_min_max maximal / minimal values	✓ OK
▶	simple1 simple tests	✓ OK
▶	simple2 simple tests	✓ OK
▶	tiny_random_ones random values {0, 1}, N = 100	✓ OK
expand all	Performance tests	
▶	small_random_ones random values {0, 1}, N = 100	✓ OK
▶	medium_zeros many zeros and 99 in the middle, length = 15,000	✓ OK
▶		

medium_random random values {1, 100}, N = 20,000	✓ OK
▶ large_random random values {0, ..., MAX_INT}, N = 100,000	✓ OK
▶ large_random_ones random values {0, 1}, N = 100,000	✓ OK
▶ all_the_same all the same values, N = 100,000	✓ OK