Codility_

CodeCheck Report: trainingFH9G9N-5U9

Test Name:

Check out Codility training tasks

Summary   Timeline

Tasks summary

Task	Time spent	Score
MinMaxDivision ⚠ Java 8	44 min	100%

Total score

100%

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;

Solution

Programming language used:	Java 8
Total time used:	44 minutes
Effective time used:	44 minutes
Notes:	*not defined yet*

Task timeline

21:30:30   22:13:58

Code: 22:13:58 UTC, java, final, score: **100**   show code in pop-up

```
// you can also use imports, for example:
// import java.util.*;

```

- [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.

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].

```
// you can write to stdout for debugging purpo
// System.out.println("this is a debug message

class Solution {
    public int solution(int K, int M, int[] A)
        int N = A.length;
        int min = 0;

        int[] prefixSum = new int[N + 1];
        for (int i = 0; i < N; i++) {
            prefixSum[i + 1] = prefixSum[i] +
            min = Math.max(min, A[i]);
        }

        int max = Math.min(prefixSum[N], (N +
        min = Math.max(min, (max + K - 1) / K)

        while (max > min) {
            int mid = (min + max) / 2;
            if (validateDivision(mid, K, pref
                max = mid;
            } else {
                min = mid + 1;
            }
        }

        return min;
    }

    private boolean validateDivision(int grou
        int count = 0;
        int prefixSumLimit = 0;

        for (int i = 1; i < prefixSum.length;
            if (prefixSum[i] > prefixSumLimit)
                count++;
                if (count > K) {
                    return false;
                }

                prefixSumLimit = prefixSum[i -
            }
        }

        return true;
    }
}
```

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