You are given integers K, M and a non-empty zero-indexed 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.

Write a function:

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

that, given integers K, M and a non-empty zero-indexed 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.

Assume that:

* 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].

Complexity:

* expected worst-case time complexity is O(N\*log(N+M));
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.