Problem 1825: Finding MK Average

Problem Information

Difficulty: Hard
Acceptance Rate: 38.56%
Paid Only: No
Tags: Design, Queue, Heap (Priority Queue), Data Stream, Ordered Set

Problem Description

You are given two integers, `m` and `k`, and a stream of integers. You are tasked to implement a data structure that calculates the **MKAverage** for the stream.

The **MKAverage** can be calculated using these steps:

1. If the number of the elements in the stream is less than `m` you should consider the **MKAverage** to be `-1`. Otherwise, copy the last `m` elements of the stream to a separate container. 2. Remove the smallest `k` elements and the largest `k` elements from the container. 3. Calculate the average value for the rest of the elements **rounded down to the nearest integer**.

Implement the `MKAverage` class:

* `MKAverage(int m, int k)` Initializes the **MKAverage** object with an empty stream and the two integers `m` and `k`. * `void addElement(int num)` Inserts a new element `num` into the stream. * `int calculateMKAverage()` Calculates and returns the **MKAverage** for the current stream **rounded down to the nearest integer**.

Example 1:

Input ["MKAverage", "addElement", "addElement", "calculateMKAverage", "addElement", "calculateMKAverage", "addElement", "addElement", "addElement", "calculateMKAverage"] [[3, 1], [3], [1], [], [10], [], [5], [5], [5], []] **Output** [null, null, null, -1, null, 3, null, null, null, 5] **Explanation** MKAverage obj = new MKAverage(3, 1); obj.addElement(3); // current elements are [3] obj.addElement(1); // current elements are [3,1] obj.calculateMKAverage(); // return -1, because m = 3 and only 2 elements exist. obj.addElement(10); // current elements

are [3,1,10] obj.calculateMKAverage(); // The last 3 elements are [3,1,10]. // After removing smallest and largest 1 element the container will be [3]. // The average of [3] equals 3/1 = 3, return 3 obj.addElement(5); // current elements are [3,1,10,5] obj.addElement(5); // current elements are [3,1,10,5,5] obj.addElement(5); // current elements are [3,1,10,5,5,5] obj.calculateMKAverage(); // The last 3 elements are [5,5,5]. // After removing smallest and largest 1 element the container will be [5]. // The average of [5] equals 5/1 = 5, return 5

Constraints:

* `3 <= m <= 105` * `1 < k*2 < m` * `1 <= num <= 105` * At most `105` calls will be made to `addElement` and `calculateMKAverage`.

Code Snippets

C++:

```
class MKAverage {
public:
MKAverage(int m, int k) {

}

void addElement(int num) {

}

int calculateMKAverage() {

}
};

/**
* Your MKAverage object will be instantiated and called as such:
* MKAverage* obj = new MKAverage(m, k);
* obj->addElement(num);
* int param_2 = obj->calculateMKAverage();
*/
```

Java:

```
class MKAverage {

public MKAverage(int m, int k) {

}

public void addElement(int num) {

}

public int calculateMKAverage() {

}
}

/**
* Your MKAverage object will be instantiated and called as such:
* MKAverage obj = new MKAverage(m, k);
* obj.addElement(num);
* int param_2 = obj.calculateMKAverage();
*/
```

Python3:

```
class MKAverage:

def __init__(self, m: int, k: int):


def addElement(self, num: int) -> None:


def calculateMKAverage(self) -> int:



# Your MKAverage object will be instantiated and called as such:
# obj = MKAverage(m, k)
# obj.addElement(num)
# param_2 = obj.calculateMKAverage()
```