meanGroups (Task 3 of 4)
0:05:05

Codewriting

You are given an array of arrays `a`. Your task is to group the arrays `a[i]` by their mean values, so that arrays with equal mean values are in the same group, and arrays with different mean values are in different groups.

Each group should contain a set of indices (`i`, `j`, etc), such that the corresponding arrays (`a[i]`, `a[j]`, etc) all have the same mean. Return the set of groups as an array of arrays, where the indices within each group are sorted in ascending order, and the groups are sorted in ascending order of their minimum element.

Example

For

```
a = [[3, 3, 4, 2],
     [4, 4],
     [4, 0, 3, 3],
     [2, 3],
     [3, 3, 3]]
```

the output should be

```
meanGroups(a) = [[0, 4],
                 [1],
                 [2, 3]]
```

-
  - `mean(a[0]) = (3 + 3 + 4 + 2) / 4 = 3`;
  - `mean(a[1]) = (4 + 4) / 2 = 4`;
  - `mean(a[2]) = (4 + 0 + 3 + 3) / 4 = 2.5`;

    - `mean(a[3]) = (2 + 3) / 2 = 2.5`;
    - `mean(a[4]) = (3 + 3 + 3) / 3 = 3`.
- There are three groups of means: those with mean `2.5`, `3`, and `4`. And they form the following groups:
    - Arrays with indices `0` and `4` form a group with mean `3`;
    - Array with index `1` forms a group with mean `4`;
    - Arrays with indices `2` and `3` form a group with mean `2.5`.

Note that neither

```
meanGroups(a) = [[0, 4],
                 [2, 3],
                 [1]]
```

nor

```
meanGroups(a) = [[0, 4],
                 [1],
                 [3, 2]]
```

- will be considered as a correct answer:
    - In the first case, the minimal element in the array at index `2` is `1`, and it is less then the minimal element in the array at index `1`, which is `2`.
    - In the second case, the array at index `2` is not sorted in ascending order.

For

```
a = [[-5, 2, 3],
     [0, 0],
     [0],
     [-100, 100]]
```

the output should be

```
meanGroups(a) = [[0, 1, 2, 3]]
```

- The mean values of all of the arrays are `0`, so all of them are in the same group.

Input/Output

- **[execution time limit] 20 seconds (swift)**

- **[input] array.array.integer a**
  An array of arrays of integers.
  Guaranteed constraints:
  `1 ≤ a.length ≤ 100`,
  `1 ≤ a[i].length ≤ 100`,
  `-100 ≤ a[i][j] ≤ 100`.
- **[output] array.array.integer**
  An array of arrays, representing the groups of indices.

[Swift] Syntax Tips

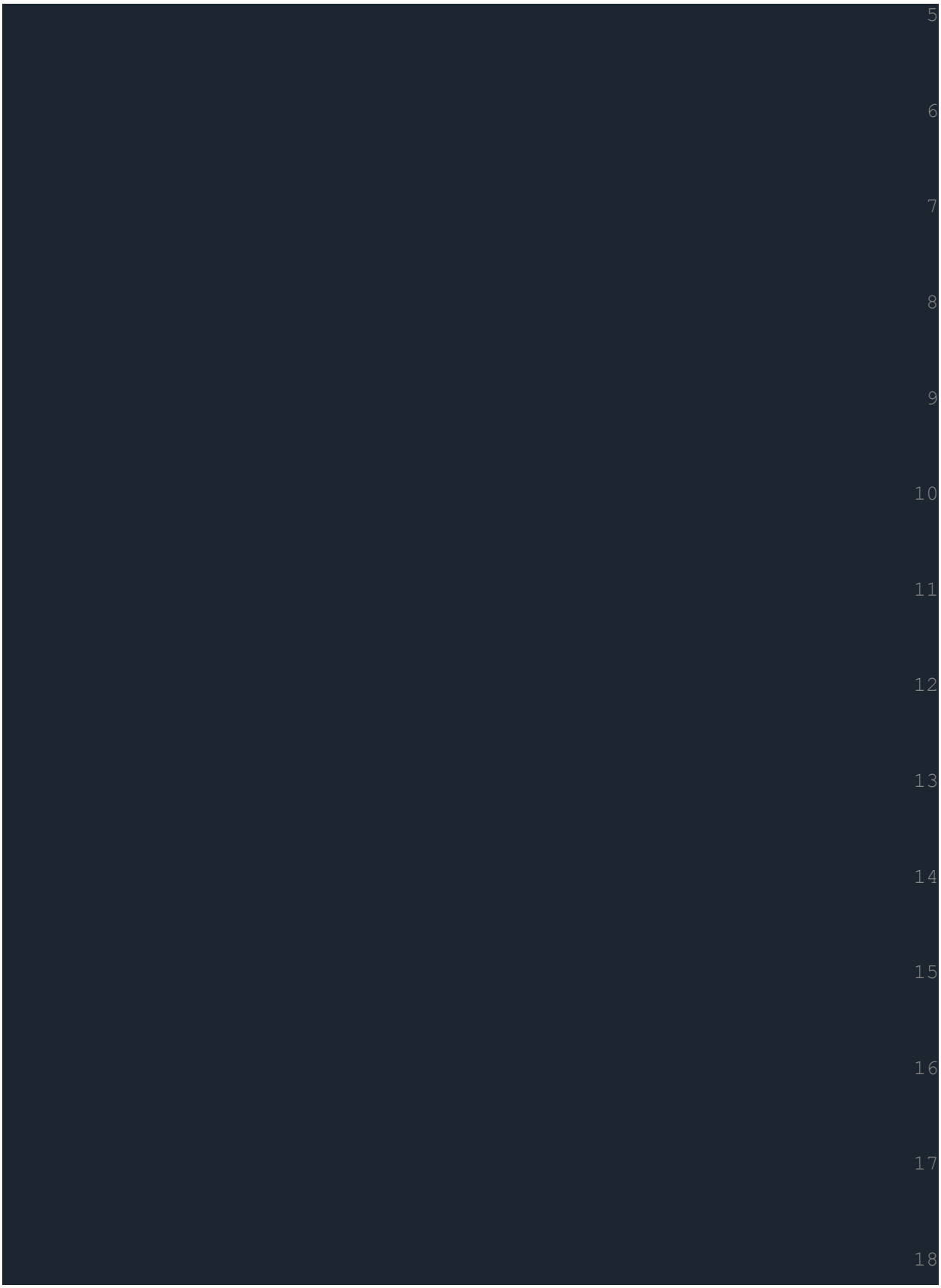
```
// Prints help message to the console
// Returns a string
func helloWorld(name: String) -> String {
    print("This prints to the console when you Run Tests");
    return "Hello, " + name;
}
```

main.swift

Saved

Swift

1
2
3
4



```
19
20
21
22
23
24
25
```

```swift
func meanGroups(a: [[Int]]) -> [[Int]] {

         var dict = [Int:Double]()

        for (index,item) in a.enumerated() {

            let sum = item.reduce(0, +)

            let value = Double(sum)/Double(item.count)

            dict[index] = value

        }
```

```
        let sortedTwo = dict.sorted {

            return $0.value > $1.value

        }



//        print(sortedTwo)

        var sortDict = [Double: [Int]]()

        for item in sortedTwo {

//            print(item)

            if let value = sortDict[item.value] {

                var newArr = value

                newArr.append(item.key)

                sortDict[item.value] = newArr

            }

        }
```

```
        print(sortedTwo)


        return [[0]]


}
```

TESTS

CUSTOM TESTS

RUN TESTS

MORE

0/300

SUBMIT

SKIP

==========

5 minutes remaining!

Wrap up your work and submit.

BACK TO TASKS

isZigzag (Task 1 of 4)

0:03:34

Codewriting

Let's say a triple `(a, b, c)` is a *zigzag* if either `a < b > c` or `a > b < c`.

Given an array of integers `numbers`, your task is to check all the triples of its consecutive elements for being a *zigzag*. More formally, your task is to construct an array of length `numbers.length - 2`, where the `i`th element of the output array equals `1` if the triple `(numbers[i], numbers[i + 1], numbers[i + 2])` is a *zigzag*, and `0` otherwise.

Example

- For `numbers = [1, 2, 1, 3, 4]`, the output should be `isZigzag(numbers) = [1, 1, 0]`.
  - `(numbers[0], numbers[1], numbers[2]) = (1, 2, 1)` **is** a *zigzag*, because `1 < 2 > 1`;
  - `(numbers[1], numbers[2] , numbers[3]) = (2, 1, 3)` **is** a *zigzag*, because `2 > 1 < 3`;
  - `(numbers[2], numbers[3] , numbers[4]) = (1, 3, 4)` **is not** a *zigzag*, because `1 < 3 < 4`;
- For `numbers = [1, 2, 3, 4]`, the output should be `isZigzag(numbers) = [0, 0]`;
  Since all the elements of `numbers` are increasing, there are no *zigzags*.
- For `numbers = [1000000000, 1000000000, 1000000000]`, the output should be `isZigzag(numbers) = [0]`.
  Since all the elements of `numbers` are the same, there are no *zigzags*.

Input/Output

- **[execution time limit] 0.5 seconds (c)**

- **[input] array.integer numbers**
  An array of integers.
  Guaranteed constraints:
  `3 ≤ numbers.length ≤ 100`,
  `1 ≤ numbers[i] ≤ 10^9`.
- **[output] array.integer**
  Return an array, where the `i^th` element equals `1` if the triple `(numbers[i], numbers[i + 1], numbers[i + 2])` is a *zigzag*, and `0` otherwise.

[C] Syntax Tips

```c
// Prints help message to the console
// Returns a string
char * helloWorld(char * name) {
    char * answer = malloc(strlen(name) + 8);
    printf("This prints to the console when you Run Tests");
    strcpy(answer, "Hello, ");
    strcat(answer, name);
    return answer;
}
```

main.c

C

```
1
2
3
4
```

```
// Arrays are already defined with this interface:

// typedef struct arr_##name {
```

```
//   int size;

//   type *arr;

// } arr_##name;

//

// arr_##name alloc_arr_##name(int len) {

//   arr_##name a = {len, len > 0 ? malloc(sizeof(type) * len) : NULL};

//   return a;

// }

//

//

arr_integer isZigzag(arr_integer numbers) {



}
```

TESTS

CUSTOM TESTS

RUN TESTS

MORE

0/300

SUBMIT

SKIP

==========

5 minutes remaining!

Wrap up your work and submit.

BACK TO TASKS

mostFrequentDigits (Task 2 of 4) Submitted

0:02:55

Codewriting

Given an array of integers `a`, your task is to calculate the digits that occur the most number of times in the array. Return the array of these digits in ascending order.

Example

For `a = [25, 2, 3, 57, 38, 41]`, the output should be `mostFrequentDigits(a) = [2, 3, 5]`.

Here are the number of times each digit appears in the array:

```
0 -> 0
1 -> 1
2 -> 2
3 -> 2
4 -> 1
5 -> 2
6 -> 0
7 -> 1
8 -> 1
```

The most number of times any number occurs in the array is `2`, and the digits which appear `2` times are `2`, `3` and `5`. So the answer is `[2, 3, 5]`.

Input/Output

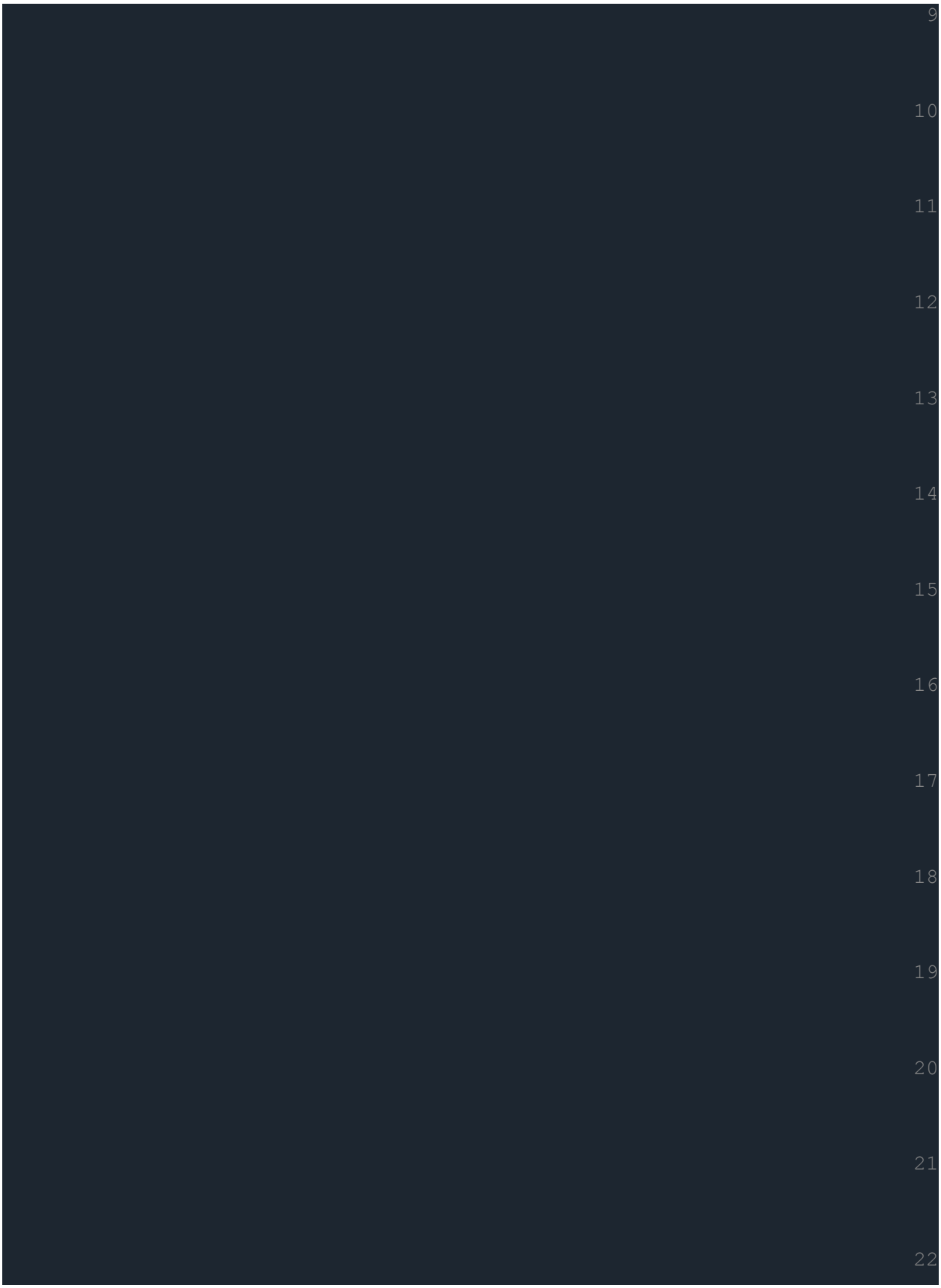
- **[execution time limit] 20 seconds (swift)**
- **[input] array.integer a**
  An array of positive integers.
  Guaranteed constraints:
  $1 \leq a.length \leq 10^3$,
  $1 \leq a[i] < 100$.
- **[output] array.integer**
  The array of most frequently occurring digits, sorted in ascending order.

[Swift] Syntax Tips

```swift
// Prints help message to the console
// Returns a string
func helloWorld(name: String) -> String {
    print("This prints to the console when you Run Tests");
    return "Hello, " + name;
}
```

main.swift

Swift



```
23

24

25

func mostFrequentDigits(a: [Int]) -> [Int] {

        var dict = [Int: Int]()



        func addDict(arr:[Int]) {

            for item in arr {

                if let value = dict[item] {

                    dict[item] = value + 1

                } else {

                    dict[item] = 1

                }
```

```
            }

        }

        for item in a {

            let string = String(item)




            let digits = string.compactMap{ $0.wholeNumberValue } // [1,
2, 3, 4, 5,

6]




            addDict(arr: digits)

        }

        let maxArr = dict.filter({$0.value == dict.values.max()})

        return Array(maxArr.keys).sorted(by: <)
```

```
}
```

TESTS

CUSTOM TESTS

RUN TESTS

MORE

300/300

SUBMIT

NEXT

==========

5 minutes remaining!

Wrap up your work and submit.

BACK TO TASKS

segmentsWithSum (Task 4 of 4)

0:02:34

Codewriting

Given an array of integers `a`, consider all its contiguous subarrays of length `m`. Calculate the number of such subarrays, which contain a pair of integers in it with sum greater than or equal to `k`.

More formally, given the array `a`, your task is to count the number of such indices `0 ≤ i ≤ a.length - m` such that a subarray `[a[i], a[i + 1], ..., a[i + m - 1]]` contains such pair `(a[s], a[t])`, such that:

- `s ≠ t`
- `a[s] + a[t] ≥ k`

Example

- For `a = [2, 4, 2, 7, 1, 6, 1, 1, 1]`, `m = 4`, and `k = 8`, the output should be `segmentsWithSum(a, m, k) = 4`.

Let's consider all subarrays of length `m = 4` and see which of them fit the description conditions:

- Subarray `a[0..3] = [2, 4, 2, 7]` contain a pair `(a[0], a[3])` that have sum greater than or equal `k`: `a[0] + a[3] = 2 + 7 + 9 ≥ 8`. Note, that there are two more such pairs in the subarray: `(a[1], a[3])` and `(a[2], a[3])`. Also note that there is a pair `(a[3], a[3])` having sum `7 + 7 = 14 ≥ 8`, but it shoudn't be taken into account, because it violates condition `s ≠ t`.
- Subarray `a[1..4] = [4, 2, 7, 1]` contains a pair `(a[1], a[3])`, having `a[1] + a[3] = 4 + 7 = 11 ≥ 8`. Note, that there is one more such pair in the subarray: `(a[3], a[4])`.
- Subarray `a[2..5] = [2, 7, 1, 6]` contains a pair `(a[2], a[3])`, having `a[2] + a[3] = 2 + 7 = 9 ≥ 8`. Note, that there are three more such pairs: `(a[2], a[5])`, `(a[3], a[4])`, and `(a[3], a[5])`.
- Subarray `a[3..6] = [7, 1, 6, 1]` contains a pair `(a[3], a[4])` having the sum equal `a[3] + a[4] = 7 + 1 = 8 ≥ 8`. Note, that there is one more such pair in the subarray: `(a[3], a[5])`.
- Subarray `a[4..7] = [1, 6, 1, 1]` doesn't contain any pair having the sum greater than or equal to `k = 8`.
- Subarray `a[5..8] = [6, 1, 1, 1]` doesn't contain any pair having the sum greater than or equal to `k = 8`.

Summary, there are `4` subarrays having a pair with sum greater than or equal to `k = 8`.

- For `a = [2, 3, 3, 3, 4, 3, 2, 1, 2, 4]`, `m = 2`, and `k = 7`, the output should be `segmentsWithSum(a, m, k) = 2`.

There are `2` subarrays satisfying the description conditions: `a[3..4] = [3, 4]` and `a[4..5] = [4, 3]`.

Input/Output

- **[execution time limit] 0.5 seconds (c)**
- **[input] array.integer a**
  The given array of integers.
  Guaranteed constraints:
  $2 \leq a.length \leq 10^5$,
  $0 \leq a[i] \leq 10^9$.
- **[input] integer m**
  The length of subarrays should be considered.
  Guaranteed constraints:
  $2 \leq m \leq a.length$.
- **[input] integer k**
  The requested lower bound for the sum of pair in the subarray.
  Guaranteed constraints:
  $0 \leq k \leq 10^9$.
- **[output] integer**
  The number of such subarrays, which contain a pair of integers with sum greater than or equal to `k`.

[C] Syntax Tips

```
// Prints help message to the console
// Returns a string
```

```c
char * helloWorld(char * name) {
    char * answer = malloc(strlen(name) + 8);
    printf("This prints to the console when you Run Tests");
    strcpy(answer, "Hello, ");
    strcat(answer, name);
    return answer;
}
```

main.c

C

```
// Arrays are already defined with this interface:

// typedef struct arr_##name {

//   int size;

//   type *arr;

// } arr_##name;

//
```

```
// arr_##name alloc_arr_##name(int len) {

//   arr_##name a = {len, len > 0 ? malloc(sizeof(type) * len) : NULL};

//   return a;

// }

//

//

int segmentsWithSum(arr_integer a, int m, int k) {




}
```

TESTS

CUSTOM TESTS

RUN TESTS

MORE

0/300

SUBMIT

BACK TO TASKS