Problem 2090: K Radius Subarray Averages

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

0-indexed

array

nums

of

n

integers, and an integer

k

.

The

k-radius average

for a subarray of

nums

centered

at some index

i

with the

radius

k

is the average of

all

elements in

nums

between the indices

i - k

and

i + k

(

inclusive

). If there are less than

k

elements before

or

after the index

i

, then the

k-radius average

is

-1

.

Build and return

an array

avgs

of length

n

where

avgs[i]

is the

k-radius average

for the subarray centered at index

i

.

The

average

of

x

elements is the sum of the

x

elements divided by

x

, using

integer division

. The integer division truncates toward zero, which means losing its fractional part.

For example, the average of four elements

2

,

3

,

1

, and

5

is

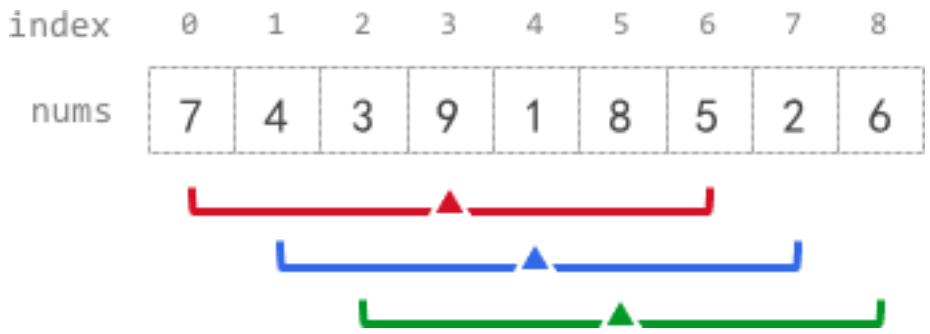
(2 + 3 + 1 + 5) / 4 = 11 / 4 = 2.75

, which truncates to

2

.

Example 1:



Input:

nums = [7,4,3,9,1,8,5,2,6], k = 3

Output:

[-1,-1,-1,5,4,4,-1,-1,-1]

Explanation:

- avg[0], avg[1], and avg[2] are -1 because there are less than k elements

before

each index. - The sum of the subarray centered at index 3 with radius 3 is: 7 + 4 + 3 + 9 + 1 + 8 + 5 = 37. Using

integer division

, avg[3] = 37 / 7 = 5. - For the subarray centered at index 4, avg[4] = (4 + 3 + 9 + 1 + 8 + 5 + 2) / 7 = 4. - For the subarray centered at index 5, avg[5] = (3 + 9 + 1 + 8 + 5 + 2 + 6) / 7 = 4. -

avg[6], avg[7], and avg[8] are -1 because there are less than k elements

after

each index.

Example 2:

Input:

nums = [100000], k = 0

Output:

[100000]

Explanation:

- The sum of the subarray centered at index 0 with radius 0 is: 100000. avg[0] = 100000 / 1 = 100000.

Example 3:

Input:

nums = [8], k = 100000

Output:

[-1]

Explanation:

- avg[0] is -1 because there are less than k elements before and after index 0.

Constraints:

n == nums.length

1 <= n <= 10

5

0 <= nums[i], k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> getAverages(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int[] getAverages(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def getAverages(self, nums: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def getAverages(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var getAverages = function(nums, k) {

};
```

TypeScript:

```
function getAverages(nums: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] GetAverages(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getAverages(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

```
func getAverages(nums []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun getAverages(nums: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func getAverages(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_averages(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def get_averages(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function getAverages($nums, $k) {

}
```

```
}
```

Dart:

```
class Solution {
List<int> getAverages(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def getAverages(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_averages(nums :: [integer], k :: integer) :: [integer]
def get_averages(nums, k) do

end
end
```

Erlang:

```
-spec get_averages(Nums :: [integer()], K :: integer()) -> [integer()].
get_averages(Nums, K) ->
.
```

Racket:

```
(define/contract (get-averages nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> getAverages(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] getAverages(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K Radius Subarray Averages
Difficulty: Medium
Tags: array
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def getAverages(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getAverages(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var getAverages = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function getAverages(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] GetAverages(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: K Radius Subarray Averages
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getAverages(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: K Radius Subarray Averages
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func getAverages(nums []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getAverages(nums: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func getAverages(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: K Radius Subarray Averages
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn get_averages(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def get_averages(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function getAverages($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> getAverages(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def getAverages(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_averages(nums :: [integer], k :: integer) :: [integer]
def get_averages(nums, k) do

end
end
```

Erlang Solution:

```
-spec get_averages(Nums :: [integer()], K :: integer()) -> [integer()].
get_averages(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (get-averages nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```