

Problem 3689: Maximum Total Subarray Value I

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`nums`

of length

`n`

and an integer

`k`

.

You need to choose

exactly

`k`

non-empty

subarrays

`nums[l..r]`

of

nums

. Subarrays may overlap, and the exact same subarray (same

l

and

r

)

can

be chosen more than once.

The

value

of a subarray

`nums[l..r]`

is defined as:

$\max(\text{nums}[l..r]) - \min(\text{nums}[l..r])$

.

The

total value

is the sum of the

values

of all chosen subarrays.

Return the

maximum

possible total value you can achieve.

Example 1:

Input:

nums = [1,3,2], k = 2

Output:

4

Explanation:

One optimal approach is:

Choose

nums[0..1] = [1, 3]

. The maximum is 3 and the minimum is 1, giving a value of

$3 - 1 = 2$

.

Choose

nums[0..2] = [1, 3, 2]

. The maximum is still 3 and the minimum is still 1, so the value is also

$$3 - 1 = 2$$

.

Adding these gives

$$2 + 2 = 4$$

.

Example 2:

Input:

nums = [4,2,5,1], k = 3

Output:

12

Explanation:

One optimal approach is:

Choose

nums[0..3] = [4, 2, 5, 1]

. The maximum is 5 and the minimum is 1, giving a value of

$$5 - 1 = 4$$

.

Choose

nums[0..3] = [4, 2, 5, 1]

. The maximum is 5 and the minimum is 1, so the value is also

4

.

Choose

`nums[2..3] = [5, 1]`

. The maximum is 5 and the minimum is 1, so the value is again

4

.

Adding these gives

$4 + 4 + 4 = 12$

.

Constraints:

$1 \leq n \leq \text{nums.length} \leq 5 * 10$

4

$0 \leq \text{nums}[i] \leq 10$

9

$1 \leq k \leq 10$

5

Code Snippets

C++:

```

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

    }

};

```

Java:

```

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

    }

}

```

Python3:

```

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

```

Python:

```

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

```

JavaScript:

```

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

};

```

TypeScript:

```
function maxTotalValue(nums: number[], k: number): number {  
  
};
```

C#:

```
public class Solution {  
    public long MaxTotalValue(int[] nums, int k) {  
  
    }  
}
```

C:

```
long long maxTotalValue(int* nums, int numsSize, int k) {  
  
}
```

Go:

```
func maxTotalValue(nums []int, k int) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxTotalValue(nums: IntArray, k: Int): Long {  
  
    }  
}
```

Swift:

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

Rust:

```

impl Solution {
  pub fn max_total_value(nums: Vec<i32>, k: i32) -> i64 {

  }
}

```

Ruby:

```

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

end

```

PHP:

```

class Solution {

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

    }

}

```

Dart:

```

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

  }
}

```

Scala:

```

object Solution {
  def maxTotalValue(nums: Array[Int], k: Int): Long = {

  }
}

```

```
}
```

Elixir:

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

  end
end
```

Erlang:

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

Racket:

```
(define/contract (max-total-value nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Total Subarray Value I
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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:
    long long maxTotalValue(vector<int>& nums, int k) {
```

```
}  
};
```

Java Solution:

```
/**  
 * Problem: Maximum Total Subarray Value I  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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 long maxTotalValue(int[] nums, int k) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Maximum Total Subarray Value I  
Difficulty: Medium  
Tags: array, greedy  
  
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 maxTotalValue(self, nums: List[int], k: int) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Maximum Total Subarray Value I
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 maxTotalValue = function(nums, k) {

};

```

TypeScript Solution:

```

/**
 * Problem: Maximum Total Subarray Value I
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 maxTotalValue(nums: number[], k: number): number {

```

```
};
```

C# Solution:

```
/*
 * Problem: Maximum Total Subarray Value I
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 long MaxTotalValue(int[] nums, int k) {

    }
}
```

C Solution:

```
/*
 * Problem: Maximum Total Subarray Value I
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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
 */

long long maxTotalValue(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Total Subarray Value I
// Difficulty: Medium
```

```
// Tags: array, greedy
//
// 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 maxTotalValue(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Maximum Total Subarray Value I
// Difficulty: Medium
// Tags: array, greedy
//
// 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 max_total_value(nums: Vec<i32>, k: i32) -> i64 {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

  }
}
```

Scala Solution:

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

  }
}
```

Elixir Solution:

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

  end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-total-value nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?)
  )
```