Problem 1031: Maximum Sum of Two Non-Overlapping Subarrays

Problem Information

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

Problem Description

Given an integer array

nums

and two integers

firstLen

and

secondLen

, return

the maximum sum of elements in two non-overlapping

subarrays

with lengths

firstLen

and

secondLen

.

The array with length

firstLen

could occur before or after the array with length

secondLen

, but they have to be non-overlapping.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

nums = [0,6,5,2,2,5,1,9,4], firstLen = 1, secondLen = 2

Output:

20

Explanation:

One choice of subarrays is [9] with length 1, and [6,5] with length 2.

Example 2:

Input:

nums = [3,8,1,3,2,1,8,9,0], firstLen = 3, secondLen = 2

Output:

29

Explanation:

One choice of subarrays is [3,8,1] with length 3, and [8,9] with length 2.

Example 3:

Input:

nums = [2,1,5,6,0,9,5,0,3,8], firstLen = 4, secondLen = 3

Output:

31

Explanation:

One choice of subarrays is [5,6,0,9] with length 4, and [0,3,8] with length 3.

Constraints:

1 <= firstLen, secondLen <= 1000

2 <= firstLen + secondLen <= 1000

firstLen + secondLen <= nums.length <= 1000

0 <= nums[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxSumTwoNoOverlap(vector<int>& nums, int firstLen, int secondLen) {

}
};
```

Java:

```
class Solution {
public int maxSumTwoNoOverlap(int[] nums, int firstLen, int secondLen) {

}
}
```

Python3:

```
class Solution:
def maxSumTwoNoOverlap(self, nums: List[int], firstLen: int, secondLen: int)
-> int:
```

Python:

```
class Solution(object):
def maxSumTwoNoOverlap(self, nums, firstLen, secondLen):
"""
:type nums: List[int]
:type firstLen: int
:type secondLen: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} firstLen
* @param {number} secondLen
* @return {number}
*/
var maxSumTwoNoOverlap = function(nums, firstLen, secondLen) {
```

```
};
```

TypeScript:

```
function maxSumTwoNoOverlap(nums: number[], firstLen: number, secondLen:
number): number {

};
```

C#:

```
public class Solution {
public int MaxSumTwoNoOverlap(int[] nums, int firstLen, int secondLen) {

}
}
```

C:

```
int maxSumTwoNoOverlap(int* nums, int numsSize, int firstLen, int secondLen)
{

}
```

Go:

```
func maxSumTwoNoOverlap(nums []int, firstLen int, secondLen int) int {

}
```

Kotlin:

```
class Solution {
fun maxSumTwoNoOverlap(nums: IntArray, firstLen: Int, secondLen: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxSumTwoNoOverlap(_ nums: [Int], _ firstLen: Int, _ secondLen: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_sum_two_no_overlap(nums: Vec<i32>, first_len: i32, second_len:
i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} first_len
# @param {Integer} second_len
# @return {Integer}
def max_sum_two_no_overlap(nums, first_len, second_len)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $firstLen
* @param Integer $secondLen
* @return Integer
*/
function maxSumTwoNoOverlap($nums, $firstLen, $secondLen) {

}
}
```

Dart:

```
class Solution {
int maxSumTwoNoOverlap(List<int> nums, int firstLen, int secondLen) {

}
}
```

Scala:

```
object Solution {
def maxSumTwoNoOverlap(nums: Array[Int], firstLen: Int, secondLen: Int): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sum_two_no_overlap(nums :: [integer], first_len :: integer,
second_len :: integer) :: integer
def max_sum_two_no_overlap(nums, first_len, second_len) do

end
end
```

Erlang:

```
-spec max_sum_two_no_overlap(Nums :: [integer()], FirstLen :: integer(),
SecondLen :: integer()) -> integer().
max_sum_two_no_overlap(Nums, FirstLen, SecondLen) ->
.
```

Racket:

```
(define/contract (max-sum-two-no-overlap nums firstLen secondLen)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxSumTwoNoOverlap(vector<int>& nums, int firstLen, int secondLen) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxSumTwoNoOverlap(int[] nums, int firstLen, int secondLen) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of Two Non-Overlapping Subarrays
Difficulty: Medium
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def maxSumTwoNoOverlap(self, nums, firstLen, secondLen):
"""
:type nums: List[int]
:type firstLen: int
:type secondLen: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} firstLen
* @param {number} secondLen
* @return {number}
*/
var maxSumTwoNoOverlap = function(nums, firstLen, secondLen) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxSumTwoNoOverlap(nums: number[], firstLen: number, secondLen:
number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxSumTwoNoOverlap(int[] nums, int firstLen, int secondLen) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of Two Non-Overlapping Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxSumTwoNoOverlap(int* nums, int numsSize, int firstLen, int secondLen)
{

}
```

Go Solution:

```
// Problem: Maximum Sum of Two Non-Overlapping Subarrays
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxSumTwoNoOverlap(nums []int, firstLen int, secondLen int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSumTwoNoOverlap(nums: IntArray, firstLen: Int, secondLen: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSumTwoNoOverlap(_ nums: [Int], _ firstLen: Int, _ secondLen: Int) ->
Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Maximum Sum of Two Non-Overlapping Subarrays
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_sum_two_no_overlap(nums: Vec<i32>, first_len: i32, second_len:
i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} first_len
# @param {Integer} second_len
# @return {Integer}
def max_sum_two_no_overlap(nums, first_len, second_len)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $firstLen
* @param Integer $secondLen
* @return Integer
*/
```

```
function maxSumTwoNoOverlap($nums, $firstLen, $secondLen) {

}
}
```

Dart Solution:

```
class Solution {
int maxSumTwoNoOverlap(List<int> nums, int firstLen, int secondLen) {

}
}
```

Scala Solution:

```
object Solution {
def maxSumTwoNoOverlap(nums: Array[Int], firstLen: Int, secondLen: Int): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sum_two_no_overlap(nums :: [integer], first_len :: integer,
second_len :: integer) :: integer
def max_sum_two_no_overlap(nums, first_len, second_len) do

end
end
```

Erlang Solution:

```
-spec max_sum_two_no_overlap(Nums :: [integer()], FirstLen :: integer(),
SecondLen :: integer()) -> integer().
max_sum_two_no_overlap(Nums, FirstLen, SecondLen) ->
.
```

Racket Solution:

```
(define/contract (max-sum-two-no-overlap nums firstLen secondLen)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```