Problem 1746: Maximum Subarray Sum After One Operation

Problem Information

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

Problem Description

You are given an integer array

nums

. You must perform

exactly one

operation where you can

replace

one element

nums[i]

with

nums[i] * nums[i]

.

Return

the

maximum

possible subarray sum after

exactly one

operation

. The subarray must be non-empty.

Example 1:

Input:

nums = [2,-1,-4,-3]

Output:

17

Explanation:

You can perform the operation on index 2 (0-indexed) to make nums = [2,-1,

16

,-3]. Now, the maximum subarray sum is 2 + -1 + 16 = 17.

Example 2:

Input:

nums = [1,-1,1,1,-1,-1,1]

Output:

4

Explanation:

You can perform the operation on index 1 (0-indexed) to make nums = [1,

1

,1,1,-1,-1,1]. Now, the maximum subarray sum is 1 + 1 + 1 + 1 = 4.

Constraints:

1 <= nums.length <= 10

5

-10

4

<= nums[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int maxSumAfterOperation(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int maxSumAfterOperation(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxSumAfterOperation(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxSumAfterOperation(int[] nums) {

}
}
```

C:

```
int maxSumAfterOperation(int* nums, int numsSize){

}
```

Go:

```
func maxSumAfterOperation(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun maxSumAfterOperation(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxSumAfterOperation(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_sum_after_operation(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def max_sum_after_operation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxSumAfterOperation($nums) {

}
}
```

Scala:

```
object Solution {
def maxSumAfterOperation(nums: Array[Int]): Int = {

}
}
```

Racket:

```
(define/contract (max-sum-after-operation nums)
(-> (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Subarray Sum After One Operation
* 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 maxSumAfterOperation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Subarray Sum After One Operation
* 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 maxSumAfterOperation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Subarray Sum After One Operation
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 maxSumAfterOperation(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Subarray Sum After One Operation
* 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
* @return {number}
*/
var maxSumAfterOperation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Subarray Sum After One Operation
* 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 maxSumAfterOperation(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Subarray Sum After One Operation
* 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 MaxSumAfterOperation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Subarray Sum After One Operation
* 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 maxSumAfterOperation(int* nums, int numsSize){

}
```

Go Solution:

```
// Problem: Maximum Subarray Sum After One Operation
// 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 maxSumAfterOperation(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSumAfterOperation(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSumAfterOperation(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Subarray Sum After One Operation
// 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_after_operation(nums: Vec<i32>) -> i32 {
}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} nums
# @return {Integer}
def max_sum_after_operation(nums)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxSumAfterOperation($nums) {

}
}
```

Scala Solution:

```scala
object Solution {
def maxSumAfterOperation(nums: Array[Int]): Int = {

}
}
```

Racket Solution:

```racket
(define/contract (max-sum-after-operation nums)
(-> (listof exact-integer?) exact-integer?)

)
```