Problem 53: Maximum Subarray

Problem Information

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

Problem Description

Given an integer array

nums

, find the

subarray

with the largest sum, and return

its sum

.

Example 1:

Input:

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

Output:

6

Explanation:

The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input:

nums = [1]

Output:

1

Explanation:

The subarray [1] has the largest sum 1.

Example 3:

Input:

nums = [5,4,-1,7,8]

Output:

23

Explanation:

The subarray [5,4,-1,7,8] has the largest sum 23.

Constraints:

1 <= nums.length <= 10

5

-10

4

<= nums[i] <= 10

4

Follow up:

If you have figured out the

O(n)

solution, try coding another solution using the

divide and conquer

approach, which is more subtle.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
function maxSubArray($nums) {

}
}
```

Dart:

```
class Solution {
int maxSubArray(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sub_array(nums :: [integer]) :: integer
def max_sub_array(nums) do

end
end
```

Erlang:

```
-spec max_sub_array(Nums :: [integer()]) -> integer().
max_sub_array(Nums) ->
.
```

Racket:

```
(define/contract (max-sub-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Subarray
* 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 maxSubArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Subarray
* 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 maxSubArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Subarray
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 maxSubArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Subarray
* 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 maxSubArray = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Subarray
* 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 maxSubArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Subarray
* 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 MaxSubArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Subarray
* 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 maxSubArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Subarray
// 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 maxSubArray(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Subarray
// 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_sub_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maxSubArray(List<int> nums) {
```

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sub_array(nums :: [integer]) :: integer
def max_sub_array(nums) do

end
end
```

Erlang Solution:

```
-spec max_sub_array(Nums :: [integer()]) -> integer().
max_sub_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (max-sub-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```