Problem 198: House Robber

Problem Information

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

Problem Description

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and

it will automatically contact the police if two adjacent houses were broken into on the same night

.

Given an integer array

nums

representing the amount of money of each house, return

the maximum amount of money you can rob tonight

without alerting the police

.

Example 1:

Input:

nums = [1,2,3,1]

Output:

4

Explanation:

Rob house 1 (money = 1) and then rob house 3 (money = 3). Total amount you can rob = 1 + 3 = 4.

Example 2:

Input:

nums = [2,7,9,3,1]

Output:

12

Explanation:

Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1). Total amount you can rob = 2 + 9 + 1 = 12.

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 400

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (rob nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: House Robber
* Difficulty: Medium
* Tags: array, tree, 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 rob(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: House Robber
* Difficulty: Medium
* Tags: array, tree, 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 rob(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: House Robber
Difficulty: Medium
Tags: array, tree, 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 rob(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: House Robber
* Difficulty: Medium
```

```
* Tags: array, tree, 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 rob = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: House Robber
* Difficulty: Medium
* Tags: array, tree, 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 rob(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: House Robber
* Difficulty: Medium
* Tags: array, tree, 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 Rob(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: House Robber
* Difficulty: Medium
* Tags: array, tree, 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 rob(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: House Robber
// Difficulty: Medium
// Tags: array, tree, 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 rob(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: House Robber
// Difficulty: Medium
// Tags: array, tree, 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 rob(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (rob nums)
(-> (listof exact-integer?) exact-integer?)
)
```