Problem 2560: House Robber IV

Problem Information

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

Problem Description

There are several consecutive houses along a street, each of which has some money inside. There is also a robber, who wants to steal money from the homes, but he

refuses to steal from adjacent homes

.

The

capability

of the robber is the maximum amount of money he steals from one house of all the houses he robbed.

You are given an integer array

nums

representing how much money is stashed in each house. More formally, the

i

th

house from the left has

nums[i]

dollars.

You are also given an integer

k

, representing the

minimum

number of houses the robber will steal from. It is always possible to steal at least

k

houses.

Return

the

minimum

capability of the robber out of all the possible ways to steal at least

k

houses

.

Example 1:

Input:

nums = [2,3,5,9], k = 2

Output:

5

Explanation:

There are three ways to rob at least 2 houses: - Rob the houses at indices 0 and 2. Capability is max(nums[0], nums[2]) = 5. - Rob the houses at indices 0 and 3. Capability is max(nums[0], nums[3]) = 9. - Rob the houses at indices 1 and 3. Capability is max(nums[1], nums[3]) = 9. Therefore, we return min(5, 9, 9) = 5.

Example 2:

Input:

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

Output:

2

Explanation:

There are 7 ways to rob the houses. The way which leads to minimum capability is to rob the house at index 0 and 4. Return max(nums[0], nums[4]) = 2.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= (nums.length + 1)/2

Code Snippets

C++:

```
class Solution {
public:
int minCapability(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int minCapability(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

```
}
```

TypeScript:

```
function minCapability(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinCapability(int[] nums, int k) {

}
}
```

C:

```
int minCapability(int* nums, int numsSize, int k) {

}
```

Go:

```
func minCapability(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-capability nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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 minCapability(vector<int>& nums, int k) {
}
};
```

Java Solution:

```
/**
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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 minCapability(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: House Robber IV
Difficulty: Medium
Tags: array, tree, dp, greedy, search

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 minCapability(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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} k
* @return {number}
*/
var minCapability = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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 minCapability(nums: number[], k: number): number {
};
```

C# Solution:

```
/*
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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 MinCapability(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: House Robber IV
* Difficulty: Medium
* Tags: array, tree, dp, greedy, search
*
* 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 minCapability(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: House Robber IV
// Difficulty: Medium
// Tags: array, tree, dp, greedy, search
//
// 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 minCapability(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: House Robber IV
// Difficulty: Medium
// Tags: array, tree, dp, greedy, search
//
// 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 min_capability(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-capability nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```