Problem 2100: Find Good Days to Rob the Bank

Problem Information

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

Problem Description

You and a gang of thieves are planning on robbing a bank. You are given a

0-indexed

integer array

security

, where

security[i]

is the number of guards on duty on the

i

th

day. The days are numbered starting from

0

. You are also given an integer

time

.

The

i

th

day is a good day to rob the bank if:

There are at least

time

days before and after the

i

th

day,

The number of guards at the bank for the

time

days

before

i

are

non-increasing

, and

The number of guards at the bank for the

time

days

after

i

are

non-decreasing

.

More formally, this means day

i

is a good day to rob the bank if and only if

security[i - time] >= security[i - time + 1] >= ... >= security[i] <= ... <= security[i + time - 1] <= security[i + time]

.

Return

a list of

all

days

(0-indexed)

that are good days to rob the bank

.

The order that the days are returned in does

not

matter.

Example 1:

Input:

security = [5,3,3,3,5,6,2], time = 2

Output:

[2,3]

Explanation:

On day 2, we have security[0] >= security[1] >= security[2] <= security[3] <= security[4]. On day 3, we have security[1] >= security[2] >= security[3] <= security[4] <= security[5]. No other days satisfy this condition, so days 2 and 3 are the only good days to rob the bank.

Example 2:

Input:

security = [1,1,1,1,1], time = 0

Output:

[0,1,2,3,4]

Explanation:

Since time equals 0, every day is a good day to rob the bank, so return every day.

Example 3:

Input:

security = [1,2,3,4,5,6], time = 2

Output:

[]

Explanation:

No day has 2 days before it that have a non-increasing number of guards. Thus, no day is a good day to rob the bank, so return an empty list.

Constraints:

1 <= security.length <= 10

5

0 <= security[i], time <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> goodDaysToRobBank(vector<int>& security, int time) {

}
};
```

Java:

```
class Solution {
public List<Integer> goodDaysToRobBank(int[] security, int time) {

}
}
```

Python3:

```
class Solution:
def goodDaysToRobBank(self, security: List[int], time: int) -> List[int]:
```

Python:

```
class Solution(object):
def goodDaysToRobBank(self, security, time):
"""
:type security: List[int]
:type time: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} security
* @param {number} time
* @return {number[]}
*/
var goodDaysToRobBank = function(security, time) {

};
```

TypeScript:

```
function goodDaysToRobBank(security: number[], time: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> GoodDaysToRobBank(int[] security, int time) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* goodDaysToRobBank(int* security, int securitySize, int time, int*
returnSize) {

}
```

Go:

```
func goodDaysToRobBank(security []int, time int) []int {

}
```

Kotlin:

```
class Solution {
fun goodDaysToRobBank(security: IntArray, time: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func goodDaysToRobBank(_ security: [Int], _ time: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn good_days_to_rob_bank(security: Vec<i32>, time: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} security
# @param {Integer} time
# @return {Integer[]}
```

```
def good_days_to_rob_bank(security, time)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $security
* @param Integer $time
* @return Integer[]
*/
function goodDaysToRobBank($security, $time) {

}
}
```

Dart:

```
class Solution {
List<int> goodDaysToRobBank(List<int> security, int time) {

}
}
```

Scala:

```
object Solution {
def goodDaysToRobBank(security: Array[Int], time: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec good_days_to_rob_bank(security :: [integer], time :: integer) ::
[integer]
def good_days_to_rob_bank(security, time) do

end
```

```
end
```

Erlang:

```
-spec good_days_to_rob_bank(Security :: [integer()], Time :: integer()) ->
[integer()].
good_days_to_rob_bank(Security, Time) ->
.
```

Racket:

```
(define/contract (good-days-to-rob-bank security time)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Good Days to Rob the Bank
* 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:
vector<int> goodDaysToRobBank(vector<int>& security, int time) {

}
};
```

Java Solution:

```
/**
* Problem: Find Good Days to Rob the Bank
```

```
* 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 List<Integer> goodDaysToRobBank(int[] security, int time) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Good Days to Rob the Bank
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 goodDaysToRobBank(self, security: List[int], time: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def goodDaysToRobBank(self, security, time):
"""
:type security: List[int]
:type time: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Good Days to Rob the Bank
* 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[]} security
* @param {number} time
* @return {number[]}
*/
var goodDaysToRobBank = function(security, time) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Good Days to Rob the Bank
* 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 goodDaysToRobBank(security: number[], time: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Good Days to Rob the Bank
* 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 IList<int> GoodDaysToRobBank(int[] security, int time) {

}
}
```

C Solution:

```
/*
* Problem: Find Good Days to Rob the Bank
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* goodDaysToRobBank(int* security, int securitySize, int time, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find Good Days to Rob the Bank
// 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 goodDaysToRobBank(security []int, time int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun goodDaysToRobBank(security: IntArray, time: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func goodDaysToRobBank(_ security: [Int], _ time: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Good Days to Rob the Bank
// 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 good_days_to_rob_bank(security: Vec<i32>, time: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} security
# @param {Integer} time
```

```
# @return {Integer[]}
def good_days_to_rob_bank(security, time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $security
* @param Integer $time
* @return Integer[]
*/
function goodDaysToRobBank($security, $time) {

}
}
```

Dart Solution:

```
class Solution {
List<int> goodDaysToRobBank(List<int> security, int time) {

}
}
```

Scala Solution:

```
object Solution {
def goodDaysToRobBank(security: Array[Int], time: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec good_days_to_rob_bank(security :: [integer], time :: integer) ::
[integer]
def good_days_to_rob_bank(security, time) do
```

```
end
end
```

Erlang Solution:

```
-spec good_days_to_rob_bank(Security :: [integer()], Time :: integer()) ->
[integer()].
good_days_to_rob_bank(Security, Time) ->
.
```

Racket Solution:

```
(define/contract (good-days-to-rob-bank security time)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```