Problem 3449: Maximize the Minimum Game Score

Problem Information

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

Problem Description

You are given an array

points

of size

n

and an integer

m

. There is another array

gameScore

of size

n

, where

gameScore[i]

represents the score achieved at the

i

th

game. Initially,

gameScore[i] == 0

for all

i

.

You start at index -1, which is outside the array (before the first position at index 0). You can make

at most

m

moves. In each move, you can either:

Increase the index by 1 and add

points[i]

to

gameScore[i]

.

Decrease the index by 1 and add

points[i]

to

gameScore[i]

.

Note

that the index must always remain within the bounds of the array after the first move.

Return the

maximum possible minimum

value in

gameScore

after

at most

m

moves.

Example 1:

Input:

points = [2,4], m = 3

Output:

4

Explanation:

Initially, index

i = -1

and

gameScore = [0, 0]

.

Move

Index

gameScore

Increase

i

0

[2, 0]

Increase

i

1

[2, 4]

Decrease

i

0

[4, 4]

The minimum value in

gameScore

is 4, and this is the maximum possible minimum among all configurations. Hence, 4 is the output.

Example 2:

Input:

points = [1,2,3], m = 5

Output:

2

Explanation:

Initially, index

i = -1

and

gameScore = [0, 0, 0]

.

Move

Index

gameScore

Increase

i

0

[1, 0, 0]

Increase

i

1

[1, 2, 0]

Decrease

i

0

[2, 2, 0]

Increase

i

1

[2, 4, 0]

Increase

i

2

[2, 4, 3]

The minimum value in

gameScore

is 2, and this is the maximum possible minimum among all configurations. Hence, 2 is the output.

Constraints:

2 <= n == points.length <= 5 * 10

4

1 <= points[i] <= 10

6

1 <= m <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maxScore(vector<int>& points, int m) {

}
};
```

Java:

```
class Solution {
public long maxScore(int[] points, int m) {

}
}
```

Python3:

```
class Solution:
def maxScore(self, points: List[int], m: int) -> int:
```

Python:

```
class Solution(object):
def maxScore(self, points, m):
"""
:type points: List[int]
:type m: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} points
* @param {number} m
* @return {number}
*/
var maxScore = function(points, m) {

};
```

TypeScript:

```
function maxScore(points: number[], m: number): number {

};
```

C#:

```
public class Solution {
public long MaxScore(int[] points, int m) {

}
}
```

C:

```
long long maxScore(int* points, int pointsSize, int m) {

}
```

Go:

```
func maxScore(points []int, m int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxScore(points: IntArray, m: Int): Long {

}
}
```

Swift:

```
class Solution {
func maxScore(_ points: [Int], _ m: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score(points: Vec<i32>, m: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} points
# @param {Integer} m
# @return {Integer}
def max_score(points, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $points
* @param Integer $m
* @return Integer
*/
function maxScore($points, $m) {

}
}
```

Dart:

```
class Solution {
int maxScore(List<int> points, int m) {

}
}
```

Scala:

```
object Solution {
def maxScore(points: Array[Int], m: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_score(points :: [integer], m :: integer) :: integer
def max_score(points, m) do

end
end
```

Erlang:

```
-spec max_score(Points :: [integer()], M :: integer()) -> integer().
max_score(Points, M) ->
.
```

Racket:

```
(define/contract (max-score points m)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long maxScore(vector<int>& points, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long maxScore(int[] points, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Minimum Game Score
Difficulty: Hard
Tags: array, greedy, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxScore(self, points: List[int], m: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxScore(self, points, m):
"""
:type points: List[int]
:type m: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {number[]} points
* @param {number} m
* @return {number}
*/
var maxScore = function(points, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxScore(points: number[], m: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long MaxScore(int[] points, int m) {

}

}
```

C Solution:

```
/*
* Problem: Maximize the Minimum Game Score
* Difficulty: Hard
* Tags: array, greedy, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

long long maxScore(int* points, int pointsSize, int m) {

}
```

Go Solution:

```
// Problem: Maximize the Minimum Game Score
// Difficulty: Hard
// Tags: array, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxScore(points []int, m int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(points: IntArray, m: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxScore(_ points: [Int], _ m: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize the Minimum Game Score
// Difficulty: Hard
// Tags: array, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_score(points: Vec<i32>, m: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} points
# @param {Integer} m
# @return {Integer}
def max_score(points, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $points
* @param Integer $m
* @return Integer
*/
function maxScore($points, $m) {

}
}
```

Dart Solution:

```
class Solution {
int maxScore(List<int> points, int m) {

}
}
```

Scala Solution:

```
object Solution {
def maxScore(points: Array[Int], m: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score(points :: [integer], m :: integer) :: integer
def max_score(points, m) do

end
end
```

Erlang Solution:

```
-spec max_score(Points :: [integer()], M :: integer()) -> integer().
max_score(Points, M) ->
.
```

Racket Solution:

```
(define/contract (max-score points m)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```