Problem 1014: Best Sightseeing Pair

Problem Information

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

Problem Description

You are given an integer array

values

where values[i] represents the value of the

i

th

sightseeing spot. Two sightseeing spots

i

and

j

have a

distance

j - i

between them.

The score of a pair (

$i < j$

) of sightseeing spots is

$values[i] + values[j] + i - j$

: the sum of the values of the sightseeing spots, minus the distance between them.

Return

the maximum score of a pair of sightseeing spots

.

Example 1:

Input:

values = [8,1,5,2,6]

Output:

11

Explanation:

i = 0, j = 2, values[i] + values[j] + i - j = 8 + 5 + 0 - 2 = 11

Example 2:

Input:

values = [1,2]

Output:

2

Constraints:

2 <= values.length <= 5 * 10

4

1 <= values[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxScoreSightseeingPair(vector<int>& values) {

}
};
```

Java:

```
class Solution {
public int maxScoreSightseeingPair(int[] values) {

}
}
```

Python3:

```
class Solution:
def maxScoreSightseeingPair(self, values: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxScoreSightseeingPair(values: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxScoreSightseeingPair(int[] values) {

}
}
```

C:

```
int maxScoreSightseeingPair(int* values, int valuesSize) {

}
```

Go:

```
func maxScoreSightseeingPair(values []int) int {

}
```

Kotlin:

```
class Solution {
fun maxScoreSightseeingPair(values: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxScoreSightseeingPair(_ values: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score_sightseeing_pair(values: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} values
# @return {Integer}
def max_score_sightseeing_pair(values)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $values
* @return Integer
*/
function maxScoreSightseeingPair($values) {

}
}
```

Dart:

```
class Solution {
int maxScoreSightseeingPair(List<int> values) {

}
```

```
}
```

Scala:

```
object Solution {
def maxScoreSightseeingPair(values: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_score_sightseeing_pair(values :: [integer]) :: integer
def max_score_sightseeing_pair(values) do

end
end
```

Erlang:

```
-spec max_score_sightseeing_pair(Values :: [integer()]) -> integer().
max_score_sightseeing_pair(Values) ->
.
```

Racket:

```
(define/contract (max-score-sightseeing-pair values)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Best Sightseeing Pair
* 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 maxScoreSightseeingPair(vector<int>& values) {

}
};
```

Java Solution:

```
/**
* Problem: Best Sightseeing Pair
* 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 maxScoreSightseeingPair(int[] values) {

}
}
```

Python3 Solution:

```
"""
Problem: Best Sightseeing Pair
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 maxScoreSightseeingPair(self, values: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Best Sightseeing Pair
* 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[]} values
* @return {number}
*/
var maxScoreSightseeingPair = function(values) {

};
```

TypeScript Solution:

```
/**
* Problem: Best Sightseeing Pair
* 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 maxScoreSightseeingPair(values: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Best Sightseeing Pair
* 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 MaxScoreSightseeingPair(int[] values) {

}
}
```

C Solution:

```
/*
* Problem: Best Sightseeing Pair
* 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 maxScoreSightseeingPair(int* values, int valuesSize) {
```

```
}
```

Go Solution:

```
// Problem: Best Sightseeing Pair
// 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 maxScoreSightseeingPair(values []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxScoreSightseeingPair(values: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxScoreSightseeingPair(_ values: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Best Sightseeing Pair
// 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_score_sightseeing_pair(values: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} values
# @return {Integer}
def max_score_sightseeing_pair(values)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $values
* @return Integer
*/
function maxScoreSightseeingPair($values) {

}
}
```

Dart Solution:

```
class Solution {
int maxScoreSightseeingPair(List<int> values) {

}
}
```

Scala Solution:

```
object Solution {
def maxScoreSightseeingPair(values: Array[Int]): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score_sightseeing_pair(values :: [integer]) :: integer
def max_score_sightseeing_pair(values) do

end
end
```

Erlang Solution:

```
-spec max_score_sightseeing_pair(Values :: [integer()]) -> integer().
max_score_sightseeing_pair(Values) ->
.
```

Racket Solution:

```
(define/contract (max-score-sightseeing-pair values)
(-> (listof exact-integer?) exact-integer?)
)
```