Problem 2555: Maximize Win From Two Segments

Problem Information

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

Problem Description

There are some prizes on the

X-axis

. You are given an integer array

prizePositions

that is

sorted in non-decreasing order

, where

prizePositions[i]

is the position of the

i

th

prize. There could be different prizes at the same position on the line. You are also given an integer

k

.

You are allowed to select two segments with integer endpoints. The length of each segment must be

k

. You will collect all prizes whose position falls within at least one of the two selected segments (including the endpoints of the segments). The two selected segments may intersect.

For example if

k = 2

, you can choose segments

[1, 3]

and

[2, 4]

, and you will win any prize

i

that satisfies

1 <= prizePositions[i] <= 3

or

2 <= prizePositions[i] <= 4

.

Return

the

maximum

number of prizes you can win if you choose the two segments optimally

.

Example 1:

Input:

prizePositions = [1,1,2,2,3,3,5], k = 2

Output:

7

Explanation:

In this example, you can win all 7 prizes by selecting two segments [1, 3] and [3, 5].

Example 2:

Input:

prizePositions = [1,2,3,4], k = 0

Output:

2

Explanation:

For this example,

one choice

for the segments is

[3, 3]

and

[4, 4],

and you will be able to get

2

prizes.

Constraints:

1 <= prizePositions.length <= 10

5

1 <= prizePositions[i] <= 10

9

0 <= k <= 10

9

prizePositions

is sorted in non-decreasing order.

Code Snippets

C++:

```
class Solution {
public:
```

```
int maximizeWin(vector<int>& prizePositions, int k) {

}
};
```

Java:

```
class Solution {
public int maximizeWin(int[] prizePositions, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximizeWin(prizePositions: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximizeWin(int[] prizePositions, int k) {

}
}
```

C:

```
int maximizeWin(int* prizePositions, int prizePositionsSize, int k) {

}
```

Go:

```
func maximizeWin(prizePositions []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximizeWin(prizePositions: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} prize_positions
# @param {Integer} k
# @return {Integer}
def maximize_win(prize_positions, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $prizePositions
* @param Integer $k
* @return Integer
*/
function maximizeWin($prizePositions, $k) {

}
}
```

Dart:

```
class Solution {
int maximizeWin(List<int> prizePositions, int k) {

}
}
```

Scala:

```
object Solution {
def maximizeWin(prizePositions: Array[Int], k: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec maximize_win(prize_positions :: [integer], k :: integer) :: integer
def maximize_win(prize_positions, k) do

end
end
```

Erlang:

```
-spec maximize_win(PrizePositions :: [integer()], K :: integer()) ->
integer().
maximize_win(PrizePositions, K) ->
.
```

Racket:

```
(define/contract (maximize-win prizePositions k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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 maximizeWin(vector<int>& prizePositions, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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 maximizeWin(int[] prizePositions, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Win From Two Segments
Difficulty: Medium
Tags: array, dp, sort, 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 maximizeWin(self, prizePositions: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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[]} prizePositions
* @param {number} k
* @return {number}
*/
var maximizeWin = function(prizePositions, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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 maximizeWin(prizePositions: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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 MaximizeWin(int[] prizePositions, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Win From Two Segments
* Difficulty: Medium
* Tags: array, dp, sort, 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 maximizeWin(int* prizePositions, int prizePositionsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximize Win From Two Segments
// Difficulty: Medium
```

```
// Tags: array, dp, sort, 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 maximizeWin(prizePositions []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeWin(prizePositions: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Win From Two Segments
// Difficulty: Medium
// Tags: array, dp, sort, 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 maximize_win(prize_positions: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} prize_positions
# @param {Integer} k
# @return {Integer}
def maximize_win(prize_positions, k)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $prizePositions
* @param Integer $k
* @return Integer
*/
function maximizeWin($prizePositions, $k) {

}
}
```

Dart Solution:

```dart
class Solution {
int maximizeWin(List<int> prizePositions, int k) {

}
}
```

Scala Solution:

```scala
object Solution {
def maximizeWin(prizePositions: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_win(prize_positions :: [integer], k :: integer) :: integer
def maximize_win(prize_positions, k) do

end
end
```

Erlang Solution:

```
-spec maximize_win(PrizePositions :: [integer()], K :: integer()) ->
integer().
maximize_win(PrizePositions, K) ->
.
```

Racket Solution:

```
(define/contract (maximize-win prizePositions k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```