Problem 3281: Maximize Score of Numbers in Ranges

Problem Information

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

Problem Description

You are given an array of integers

start

and an integer

d

, representing

n

intervals

[start[i], start[i] + d]

.

You are asked to choose

n

integers where the

i

th

integer must belong to the

i

th

interval. The

score

of the chosen integers is defined as the

minimum

absolute difference between any two integers that have been chosen.

Return the

maximum

possible score

of the chosen integers.

Example 1:

Input:

start = [6,0,3], d = 2

Output:

4

Explanation:

The maximum possible score can be obtained by choosing integers: 8, 0, and 4. The score of these chosen integers is

min(|8 - 0|, |8 - 4|, |0 - 4|)

which equals 4.

Example 2:

Input:

start = [2,6,13,13], d = 5

Output:

5

Explanation:

The maximum possible score can be obtained by choosing integers: 2, 7, 13, and 18. The score of these chosen integers is

min(|2 - 7|, |2 - 13|, |2 - 18|, |7 - 13|, |7 - 18|, |13 - 18|)

which equals 5.

Constraints:

2 <= start.length <= 10

5

0 <= start[i] <= 10

9

0 <= d <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxPossibleScore(vector<int>& start, int d) {

}
};
```

Java:

```
class Solution {
public int maxPossibleScore(int[] start, int d) {

}
}
```

Python3:

```
class Solution:
def maxPossibleScore(self, start: List[int], d: int) -> int:
```

Python:

```
class Solution(object):
def maxPossibleScore(self, start, d):
"""
:type start: List[int]
:type d: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} start
* @param {number} d
* @return {number}
*/
```

```
var maxPossibleScore = function(start, d) {

};
```

TypeScript:

```
function maxPossibleScore(start: number[], d: number): number {

};
```

C#:

```
public class Solution {
public int MaxPossibleScore(int[] start, int d) {

}
}
```

C:

```
int maxPossibleScore(int* start, int startSize, int d) {

}
```

Go:

```
func maxPossibleScore(start []int, d int) int {

}
```

Kotlin:

```
class Solution {
fun maxPossibleScore(start: IntArray, d: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxPossibleScore(_ start: [Int], _ d: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn max_possible_score(start: Vec<i32>, d: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} start
# @param {Integer} d
# @return {Integer}
def max_possible_score(start, d)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $start
* @param Integer $d
* @return Integer
*/
function maxPossibleScore($start, $d) {

}
}
```

Dart:

```
class Solution {
int maxPossibleScore(List<int> start, int d) {

}
}
```

Scala:

```
object Solution {
def maxPossibleScore(start: Array[Int], d: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_possible_score(start :: [integer], d :: integer) :: integer
def max_possible_score(start, d) do

end
end
```

Erlang:

```
-spec max_possible_score(Start :: [integer()], D :: integer()) -> integer().
max_possible_score(Start, D) ->
.
```

Racket:

```
(define/contract (max-possible-score start d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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:
int maxPossibleScore(vector<int>& start, int d) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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 int maxPossibleScore(int[] start, int d) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Score of Numbers in Ranges
Difficulty: Medium
Tags: array, greedy, sort, 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 maxPossibleScore(self, start: List[int], d: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxPossibleScore(self, start, d):
"""
:type start: List[int]
:type d: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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[]} start
* @param {number} d
* @return {number}
*/
var maxPossibleScore = function(start, d) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maxPossibleScore(start: number[], d: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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 int MaxPossibleScore(int[] start, int d) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Score of Numbers in Ranges
* Difficulty: Medium
* Tags: array, greedy, sort, 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
*/

int maxPossibleScore(int* start, int startSize, int d) {
```

```
}
```

Go Solution:

```
// Problem: Maximize Score of Numbers in Ranges
// Difficulty: Medium
// Tags: array, greedy, sort, 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 maxPossibleScore(start []int, d int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxPossibleScore(start: IntArray, d: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxPossibleScore(_ start: [Int], _ d: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Score of Numbers in Ranges
// Difficulty: Medium
// Tags: array, greedy, sort, 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_possible_score(start: Vec<i32>, d: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} start
# @param {Integer} d
# @return {Integer}
def max_possible_score(start, d)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $start
* @param Integer $d
* @return Integer
*/
function maxPossibleScore($start, $d) {

}
}
```

Dart Solution:

```
class Solution {
int maxPossibleScore(List<int> start, int d) {

}
}
```

Scala Solution:

```
object Solution {
def maxPossibleScore(start: Array[Int], d: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_possible_score(start :: [integer], d :: integer) :: integer
def max_possible_score(start, d) do

end
end
```

Erlang Solution:

```
-spec max_possible_score(Start :: [integer()], D :: integer()) -> integer().
max_possible_score(Start, D) ->
.
```

Racket Solution:

```
(define/contract (max-possible-score start d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```