Problem 475: Heaters

Problem Information

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

Problem Description

Winter is coming! During the contest, your first job is to design a standard heater with a fixed warm radius to warm all the houses.

Every house can be warmed, as long as the house is within the heater's warm radius range.

Given the positions of

houses

and

heaters

on a horizontal line, return

the minimum radius standard of heaters so that those heaters could cover all houses.

Notice

that all the

heaters

follow your radius standard, and the warm radius will be the same.

Example 1:

Input:

houses = [1,2,3], heaters = [2]

Output:

1

Explanation:

The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.

Example 2:

Input:

houses = [1,2,3,4], heaters = [1,4]

Output:

1

Explanation:

The two heaters were placed at positions 1 and 4. We need to use a radius 1 standard, then all the houses can be warmed.

Example 3:

Input:

houses = [1,5], heaters = [2]

Output:

3

Constraints:

1 <= houses.length, heaters.length <= 3 * 10

4

1 <= houses[i], heaters[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int findRadius(vector<int>& houses, vector<int>& heaters) {

}
};
```

Java:

```
class Solution {
public int findRadius(int[] houses, int[] heaters) {

}
}
```

Python3:

```
class Solution:
def findRadius(self, houses: List[int], heaters: List[int]) -> int:
```

Python:

```
class Solution(object):
def findRadius(self, houses, heaters):
"""
:type houses: List[int]
:type heaters: List[int]
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number[]} houses
* @param {number[]} heaters
* @return {number}
*/
var findRadius = function(houses, heaters) {

};
```

TypeScript:

```
function findRadius(houses: number[], heaters: number[]): number {

};
```

C#:

```
public class Solution {
public int FindRadius(int[] houses, int[] heaters) {

}
}
```

C:

```
int findRadius(int* houses, int housesSize, int* heaters, int heatersSize) {

}
```

Go:

```
func findRadius(houses []int, heaters []int) int {

}
```

Kotlin:

```
class Solution {
fun findRadius(houses: IntArray, heaters: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func findRadius(_ houses: [Int], _ heaters: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_radius(houses: Vec<i32>, heaters: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} houses
# @param {Integer[]} heaters
# @return {Integer}
def find_radius(houses, heaters)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $houses
* @param Integer[] $heaters
* @return Integer
*/
function findRadius($houses, $heaters) {

}
```

```
}
```

Dart:

```
class Solution {
int findRadius(List<int> houses, List<int> heaters) {

}
}
```

Scala:

```
object Solution {
def findRadius(houses: Array[Int], heaters: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_radius(houses :: [integer], heaters :: [integer]) :: integer
def find_radius(houses, heaters) do

end
end
```

Erlang:

```
-spec find_radius(Houses :: [integer()], Heaters :: [integer()]) ->
integer().
find_radius(Houses, Heaters) ->
.
```

Racket:

```
(define/contract (find-radius houses heaters)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Heaters
* Difficulty: Medium
* Tags: array, 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 findRadius(vector<int>& houses, vector<int>& heaters) {

}
};
```

Java Solution:

```
/**
* Problem: Heaters
* Difficulty: Medium
* Tags: array, 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 findRadius(int[] houses, int[] heaters) {

}
}
```

Python3 Solution:

```
"""
Problem: Heaters
```

```
Difficulty: Medium
Tags: array, 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 findRadius(self, houses: List[int], heaters: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findRadius(self, houses, heaters):
"""
:type houses: List[int]
:type heaters: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Heaters
* Difficulty: Medium
* Tags: array, 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[]} houses
* @param {number[]} heaters
* @return {number}
*/
var findRadius = function(houses, heaters) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Heaters
* Difficulty: Medium
* Tags: array, 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 findRadius(houses: number[], heaters: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Heaters
* Difficulty: Medium
* Tags: array, 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 FindRadius(int[] houses, int[] heaters) {

}
}
```

C Solution:

```
/*
* Problem: Heaters
```

```
* Difficulty: Medium
* Tags: array, 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 findRadius(int* houses, int housesSize, int* heaters, int heatersSize) {

}
```

Go Solution:

```
// Problem: Heaters
// Difficulty: Medium
// Tags: array, 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 findRadius(houses []int, heaters []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findRadius(houses: IntArray, heaters: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func findRadius(_ houses: [Int], _ heaters: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Heaters
// Difficulty: Medium
// Tags: array, 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 find_radius(houses: Vec<i32>, heaters: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} houses
# @param {Integer[]} heaters
# @return {Integer}
def find_radius(houses, heaters)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $houses
* @param Integer[] $heaters
* @return Integer
*/
function findRadius($houses, $heaters) {

}
}
```

Dart Solution:

```
class Solution {
int findRadius(List<int> houses, List<int> heaters) {

}
}
```

Scala Solution:

```
object Solution {
def findRadius(houses: Array[Int], heaters: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_radius(houses :: [integer], heaters :: [integer]) :: integer
def find_radius(houses, heaters) do

end
end
```

Erlang Solution:

```
-spec find_radius(Houses :: [integer()], Heaters :: [integer()]) ->
integer().
find_radius(Houses, Heaters) ->
.
```

Racket Solution:

```
(define/contract (find-radius houses heaters)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```