Problem 506: Relative Ranks

Problem Information

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

Problem Description

You are given an integer array

score

of size

n

, where

score[i]

is the score of the

i

th

athlete in a competition. All the scores are guaranteed to be

unique

.

The athletes are

placed

based on their scores, where the

1

st

place athlete has the highest score, the

2

nd

place athlete has the

2

nd

highest score, and so on. The placement of each athlete determines their rank:

The

1

st

place athlete's rank is

"Gold Medal"

.

The

2

nd

place athlete's rank is

"Silver Medal"

.

The

3

rd

place athlete's rank is

"Bronze Medal"

.

For the

4

th

place to the

n

th

place athlete, their rank is their placement number (i.e., the

x

th

place athlete's rank is

"x"

).

Return an array

answer

of size

n

where

answer[i]

is the

rank

of the

i

th

athlete.

Example 1:

Input:

score = [5,4,3,2,1]

Output:

["Gold Medal","Silver Medal","Bronze Medal","4","5"]

Explanation:

The placements are [1

st

, 2

nd

, 3

rd

, 4

th

, 5

th

].

Example 2:

Input:

score = [10,3,8,9,4]

Output:

["Gold Medal","5","Bronze Medal","Silver Medal","4"]

Explanation:

The placements are [1

st

, 5

th

, 3

rd

, 2

nd

, 4

th

].

Constraints:

n == score.length

1 <= n <= 10

4

0 <= score[i] <= 10

6

All the values in

score

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findRelativeRanks(vector<int>& score) {

}
};
```

Java:

```
class Solution {
public String[] findRelativeRanks(int[] score) {

}
}
```

Python3:

```
class Solution:
def findRelativeRanks(self, score: List[int]) -> List[str]:
```

Python:

```
class Solution(object):
def findRelativeRanks(self, score):
"""
:type score: List[int]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {number[]} score
* @return {string[]}
*/
var findRelativeRanks = function(score) {
```

```
};
```

TypeScript:

```
function findRelativeRanks(score: number[]): string[] {

};
```

C#:

```
public class Solution {
public string[] FindRelativeRanks(int[] score) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findRelativeRanks(int* score, int scoreSize, int* returnSize) {

}
```

Go:

```
func findRelativeRanks(score []int) []string {

}
```

Kotlin:

```
class Solution {
fun findRelativeRanks(score: IntArray): Array<String> {

}
}
```

Swift:

```
class Solution {
func findRelativeRanks(_ score: [Int]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_relative_ranks(score: Vec<i32>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Integer[]} score
# @return {String[]}
def find_relative_ranks(score)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $score
* @return String[]
*/
function findRelativeRanks($score) {

}
}
```

Dart:

```
class Solution {
List<String> findRelativeRanks(List<int> score) {

}
}
```

Scala:

```
object Solution {
def findRelativeRanks(score: Array[Int]): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_relative_ranks(score :: [integer]) :: [String.t]
def find_relative_ranks(score) do

end
end
```

Erlang:

```
-spec find_relative_ranks(Score :: [integer()]) ->
[unicode:unicode_binary()].
find_relative_ranks(Score) ->
.
```

Racket:

```
(define/contract (find-relative-ranks score)
(-> (listof exact-integer?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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:
vector<string> findRelativeRanks(vector<int>& score) {

}
};
```

Java Solution:

```
/**
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 String[] findRelativeRanks(int[] score) {

}
}
```

Python3 Solution:

```
"""
Problem: Relative Ranks
Difficulty: Easy
Tags: array, sort, queue, heap

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 findRelativeRanks(self, score: List[int]) -> List[str]:
```

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

Python Solution:

```
class Solution(object):
def findRelativeRanks(self, score):
"""
:type score: List[int]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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[]} score
* @return {string[]}
*/
var findRelativeRanks = function(score) {

};
```

TypeScript Solution:

```
/**
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 findRelativeRanks(score: number[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 string[] FindRelativeRanks(int[] score) {

}
}
```

C Solution:

```
/*
* Problem: Relative Ranks
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findRelativeRanks(int* score, int scoreSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Relative Ranks
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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 findRelativeRanks(score []int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findRelativeRanks(score: IntArray): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func findRelativeRanks(_ score: [Int]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Relative Ranks
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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_relative_ranks(score: Vec<i32>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} score
# @return {String[]}
def find_relative_ranks(score)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $score
* @return String[]
*/
function findRelativeRanks($score) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findRelativeRanks(List<int> score) {

}
}
```

Scala Solution:

```
object Solution {
def findRelativeRanks(score: Array[Int]): Array[String] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_relative_ranks(score :: [integer]) :: [String.t]
def find_relative_ranks(score) do

end
end
```

Erlang Solution:

```
-spec find_relative_ranks(Score :: [integer()]) ->
[unicode:unicode_binary()].
find_relative_ranks(Score) ->
.
```

Racket Solution:

```
(define/contract (find-relative-ranks score)
(-> (listof exact-integer?) (listof string?))
)
```