Problem 1395: Count Number of Teams

Problem Information

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

Problem Description

There are

n

soldiers standing in a line. Each soldier is assigned a

unique

rating

value.

You have to form a team of 3 soldiers amongst them under the following rules:

Choose 3 soldiers with index (

i

,

j

,

k

) with rating (

rating[i]

,

rating[j]

,

rating[k]

).

A team is valid if: (

rating[i] < rating[j] < rating[k]

) or (

rating[i] > rating[j] > rating[k]

) where (

0 <= i < j < k < n

).

Return the number of teams you can form given the conditions. (soldiers can be part of multiple teams).

Example 1:

Input:

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

Output:

3

Explanation:

We can form three teams given the conditions. (2,3,4), (5,4,1), (5,3,1).

Example 2:

Input:

rating = [2,1,3]

Output:

0

Explanation:

We can't form any team given the conditions.

Example 3:

Input:

rating = [1,2,3,4]

Output:

4

Constraints:

n == rating.length

3 <= n <= 1000

1 <= rating[i] <= 10

5

All the integers in

rating

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int numTeams(vector<int>& rating) {

}
};
```

Java:

```
class Solution {
public int numTeams(int[] rating) {

}
}
```

Python3:

```
class Solution:
def numTeams(self, rating: List[int]) -> int:
```

Python:

```
class Solution(object):
def numTeams(self, rating):
"""
:type rating: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function numTeams(rating: number[]): number {

};
```

C#:

```
public class Solution {
public int NumTeams(int[] rating) {

}
}
```

C:

```
int numTeams(int* rating, int ratingSize) {

}
```

Go:

```
func numTeams(rating []int) int {

}
```

Kotlin:

```
class Solution {
fun numTeams(rating: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numTeams(_ rating: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_teams(rating: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} rating
# @return {Integer}
def num_teams(rating)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $rating
* @return Integer
*/
function numTeams($rating) {

}
}
```

Dart:

```
class Solution {
int numTeams(List<int> rating) {

}
}
```

Scala:

```
object Solution {
def numTeams(rating: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_teams(rating :: [integer]) :: integer
def num_teams(rating) do

end
end
```

Erlang:

```
-spec num_teams(Rating :: [integer()]) -> integer().
num_teams(Rating) ->
.
```

Racket:

```
(define/contract (num-teams rating)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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 numTeams(vector<int>& rating) {

}
};
```

Java Solution:

```
/**
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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 numTeams(int[] rating) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Number of Teams
Difficulty: Medium
Tags: array, tree, 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 numTeams(self, rating: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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[]} rating
* @return {number}
*/
var numTeams = function(rating) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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 numTeams(rating: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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 NumTeams(int[] rating) {

}
}
```

C Solution:

```
/*
* Problem: Count Number of Teams
* Difficulty: Medium
* Tags: array, tree, 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 numTeams(int* rating, int ratingSize) {

}
```

Go Solution:

```
// Problem: Count Number of Teams
// Difficulty: Medium
// Tags: array, tree, 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 numTeams(rating []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numTeams(rating: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func numTeams(_ rating: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Number of Teams
// Difficulty: Medium
// Tags: array, tree, 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 num_teams(rating: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} rating
# @return {Integer}
def num_teams(rating)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $rating
* @return Integer
*/
function numTeams($rating) {

}
}
```

Dart Solution:

```
class Solution {
int numTeams(List<int> rating) {

}
}
```

Scala Solution:

```
object Solution {
def numTeams(rating: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_teams(rating :: [integer]) :: integer
def num_teams(rating) do

end
end
```

Erlang Solution:

```
-spec num_teams(Rating :: [integer()]) -> integer().
num_teams(Rating) ->
.
```

Racket Solution:

```
(define/contract (num-teams rating)
(-> (listof exact-integer?) exact-integer?)
)
```