Problem 135: Candy

Problem Information

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

Problem Description

There are

n

children standing in a line. Each child is assigned a rating value given in the integer array

ratings

.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

Return

the minimum number of candies you need to have to distribute the candies to the children

.

Example 1:

Input:

ratings = [1,0,2]

Output:

5

Explanation:

You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Example 2:

Input:

ratings = [1,2,2]

Output:

4

Explanation:

You can allocate to the first, second and third child with 1, 2, 1 candies respectively. The third child gets 1 candy because it satisfies the above two conditions.

Constraints:

n == ratings.length

1 <= n <= 2 * 10

4

0 <= ratings[i] <= 2 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int candy(vector<int>& ratings) {

}
};
```

Java:

```
class Solution {
public int candy(int[] ratings) {

}
}
```

Python3:

```
class Solution:
def candy(self, ratings: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function candy(ratings: number[]): number {

};
```

C#:

```
public class Solution {
public int Candy(int[] ratings) {

}
}
```

C:

```
int candy(int* ratings, int ratingsSize) {

}
```

Go:

```
func candy(ratings []int) int {

}
```

Kotlin:

```
class Solution {
fun candy(ratings: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func candy(_ ratings: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn candy(ratings: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} ratings
# @return {Integer}
def candy(ratings)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $ratings
* @return Integer
*/
function candy($ratings) {

}
}
```

Dart:

```
class Solution {
int candy(List<int> ratings) {

}
}
```

Scala:

```
object Solution {
def candy(ratings: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec candy(ratings :: [integer]) :: integer
def candy(ratings) do

end
end
```

Erlang:

```
-spec candy(Ratings :: [integer()]) -> integer().
candy(Ratings) ->
.
```

Racket:

```
(define/contract (candy ratings)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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 candy(vector<int>& ratings) {

}
};
```

Java Solution:

```
/**
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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 candy(int[] ratings) {

}
}
```

Python3 Solution:

```
"""
Problem: Candy
Difficulty: Hard
Tags: array, greedy

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 candy(self, ratings: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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[]} ratings
* @return {number}
*/
var candy = function(ratings) {

};
```

TypeScript Solution:

```
/**
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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 candy(ratings: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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 Candy(int[] ratings) {

}
}
```

C Solution:

```
/*
* Problem: Candy
* Difficulty: Hard
* Tags: array, greedy
*
* 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 candy(int* ratings, int ratingsSize) {

}
```

Go Solution:

```
// Problem: Candy
// Difficulty: Hard
// Tags: array, greedy
//
// 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 candy(ratings []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun candy(ratings: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func candy(_ ratings: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Candy
// Difficulty: Hard
// Tags: array, greedy
//
// 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 candy(ratings: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} ratings
# @return {Integer}
def candy(ratings)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $ratings
* @return Integer
*/
function candy($ratings) {

}
}
```

Dart Solution:

```
class Solution {
int candy(List<int> ratings) {

}
}
```

Scala Solution:

```
object Solution {
def candy(ratings: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec candy(ratings :: [integer]) :: integer
def candy(ratings) do

end
end
```

Erlang Solution:

```
-spec candy(Ratings :: [integer()]) -> integer().
candy(Ratings) ->
.
```

Racket Solution:

```
(define/contract (candy ratings)
(-> (listof exact-integer?) exact-integer?)
)
```