<https://leetcode.com/problems/candy/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

**Attempt 1: 2024-12-01**

**Solution 1: Greedy (10min)**

class Solution {

    public int candy(int[] ratings) {

        int len = ratings.length;

        int[] count = new int[len];

        // Each child must have at least one candy.

        Arrays.fill(count, 1);

        // Left to right scanning

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

        for(int i = 1; i < len; i++) {

            if(ratings[i] > ratings[i - 1]) {

                count[i] = count[i - 1] + 1;

            }

        }

        // Right to left scanning

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

        for(int i = len - 2; i >= 0; i--) {

            if(ratings[i] > ratings[i + 1]) {

                // Wrong statement: test out by

                // ratings = [1,3,4,5,2]

                //count[i] = count[i + 1] + 1;

                count[i] = Math.max(count[i], count[i + 1] + 1);

            }

        }

        int totalCandies = 0;

        for(int c : count) {

            totalCandies += c;

        }

        return totalCandies;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to chatGPT**

**Solution Approach**

To solve this problem, we use a **greedy two-pass approach**:

Left-to-Right Pass:

Ensure each child has more candies than the left neighbor if their rating is higher.

Right-to-Left Pass:

Ensure each child has more candies than the right neighbor if their rating is higher, while retaining the result of the first pass.

class Solution {

public int candy(int[] ratings) {

int n = ratings.length;

int[] candies = new int[n];

// Step 1: Give each child at least one candy

for (int i = 0; i < n; i++) {

candies[i] = 1;

}

// Step 2: Left-to-right pass

for (int i = 1; i < n; i++) {

if (ratings[i] > ratings[i - 1]) {

candies[i] = candies[i - 1] + 1;

}

}

// Step 3: Right-to-left pass

for (int i = n - 2; i >= 0; i--) {

if (ratings[i] > ratings[i + 1]) {

candies[i] = Math.max(candies[i], candies[i + 1] + 1);

}

}

// Step 4: Sum up the candies

int totalCandies = 0;

for (int candy : candies) {

totalCandies += candy;

}

return totalCandies;

}

}