**Question-3**

**Halloween Candy**

The key steps are:

1. It takes input as the number of children N and their ratings.
2. It creates two arrays left[] and right[] to store the candies required for each child based on the rating of left and right neighbors respectively. left[i] and right[i] are initialized to 1.
3. It traverses the ratings array from left to right, comparing ratings[i] with ratings[i-1]. If ratings[i] > ratings[i-1], left[i] is set to left[i-1] + 1. This ensures higher rated child gets 1 more candy than previous child.
4. Similarly, it traverses from right to left comparing ratings[i] and ratings[i+1]. If ratings[i] > ratings[i+1], right[i] is set to right[i+1] + 1.
5. For each child, the maximum of left[i] and right[i] is chosen as the candies required. This is stored in candyDistribution[].
6. The total candies is calculated by summing all elements of candyDistribution[].

The time complexity of this algorithm is O(N) where N is the number of children.

The main operations are:

1. Filling the left[] array by traversing from left to right : O(N)

2. Filling the right[] array by traversing from right to left: O(N)

3. Finding max of left[i] and right[i] and summing to get total candies: O(N)

So overall, there are simple linear traversals and comparisons being done which makes the time complexity O(N).

Specifically:

- Traversing ratings array from left to right: O(N)

- Traversing ratings array from right to left: O(N)

- Comparing and updating left[], right[]: O(N)

- Taking max of left[i], right[i]: O(N)

- Summing final candyDistribution[]: O(N)

So at most, the algorithm does O(N) operations, making the overall time complexity linear i.e. O(N).

**Reference**:

* <https://leetcode.com/problemset/all/>
* <https://www.geeksforgeeks.org/>