Part 1: Chocolate Distribution Algorithm

In order to distribute chocolates to the pupils, we must construct both recursive and iterative functions. Every chocolate has an ID, weight, and price.

Here's a simple approach to distribution:

Iteratively assign chocolates to students, one by one.

Recursively pick a chocolate and assign it to the next student in line.

Part 2: Sorting the Chocolates

After distribution, we should sort the chocolates by weight and price. There are various sorting algorithms to choose from, like Quick Sort, Merge Sort, etc. The choice of sorting algorithm may depend on the expected size of the dataset and whether it is already partially sorted. For educational purposes, let's use Quick Sort for one attribute and Merge Sort for another.

Part 3: Searching for a Specific Chocolate

We need to write a function to search for a chocolate by a specific price or weight. Assuming the chocolates are already sorted by these attributes, a binary search would be the most efficient approach.

Complexity Analysis

For each algorithm, we should analyze the best, average, and worst-case scenarios using Big O notation.

Let's start with the distribution algorithm. Python code for the iterative and recursive distribution functions and include basic test cases. Then, we'll move on to the sorting and searching algorithms with their complexity analysis.

Let’s start with the code for the Chocolate Distribution Algorithm.

The iterative and recursive chocolate distribution functions have been implemented and tested. Both functions distribute chocolates to students until there are no chocolates left to distribute. The result for the provided test cases is the same for both iterative and recursive approaches:

Alice receives a Chocolate(weight=5, price=2, type='Almond', id=002)

Bob receives a Chocolate(weight=7, price=4, type='Peanut Butter', id=005)

Charlie and David do not receive chocolates as there are only two chocolates and four students.

Now implement the sorting algorithm for the chocolates. We'll use **Quick Sort for sorting by weight** and **Merge Sort for sorting by price**. After that, we'll write the search algorithm and perform the complexity analysis for each part. Let's proceed with the sorting algorithms.

The sorting algorithms have been implemented and tested with the provided test cases:

The Quick Sort algorithm sorted the chocolates by weight:

Milk chocolate (3g)

Almond chocolate (5g)

Dark chocolate (6g)

Peanut Butter chocolate (7g)

The Merge Sort algorithm sorted the chocolates by price:

Dark chocolate (1 AED)

Almond chocolate (2 AED)

Milk chocolate (3 AED)

Peanut Butter chocolate (4 AED)

Next, I will implement the search algorithm, which will find a chocolate based on a specified weight or price. Since the lists are already sorted, a binary search will be the most efficient method. After that, we will conduct the complexity analysis for each part. Let's proceed with the search algorithm.

The binary search algorithm has successfully found chocolates based on weight and price:

The chocolate with a weight of 6 grams is a Dark chocolate priced at 1 AED.

The chocolate with a price of 2 AED is an Almond chocolate weighing 5 grams

Now, I will tell some key discuss the complexity analysis for the algorithms used:

1. **Distribution Algorithm (Iterative and Recursive):**

Best, average, and worst-case time complexity: $O(n)$, where n is the number of students. This is because each student is assigned one chocolate, and the process stops when either the chocolates or the students list is exhausted.

Space complexity: O(n) for the resulting distribution map plus the space for the call stack in the recursive solution.

2. **Sorting Algorithm (Quick Sort for Weight and Merge Sort for Price):**

Quick Sort:

Best and average-case time complexity: $O(n\ log\ n)$, where n is the number of chocolates.

Worst-case time complexity: $O(n^2)$, which happens when the pivot selection is poor (e.g., the list is already sorted).

Merge Sort:

• Best, average, and worst-case time complexity: $O(n\ log\ n)$, where n is the number of chocolates.

Space complexity: Both sorting algorithms require $O(n)$ additional space.

3. **Searching Algorithm (Binary Search):**

Best-case time complexity: 0(1), which happens if the middle chocolate is the one we're looking for.

Average and worst-case time complexity:

$O(log\ n)$, where n is the number of chocolates.

Space complexity: $0(1)$, since binary search does not require additional space beyond the input list.