Yes, I'm familiar with the process described on pages 26 and 27 of "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein (CLRS). The exercise involves summing up the products of the cost and time for each operation in the insertion sort algorithm to compute the total running time.

Step-by-Step Breakdown

1. **Cost (c1, c2, etc.):** Represents the constant time (in terms of basic operations) that each statement in the algorithm takes to execute.
2. **Times:** Represents how many times each statement is executed during the run of the algorithm.

We calculate the total time $T(n)$ as the sum of the products of the cost and the number of times each operation is executed:

$$T(n) = c_1 \cdot T_1 + c_2 \cdot T_2 + c_3 \cdot T_3 + \cdots + c_7 \cdot T_7$$

Summing the Costs and Times

Here's the breakdown:

- $c_1 \times n$: The 'for' loop runs n times, so the cost for c_1 is $c_1 \times n$.
- $c_2 \times (n-1)$: The assignment 'let key = arr[j];' runs $n-1$ times.
- $c_3 \times (n-1)$: The assignment 'let mut i = j;' runs $n-1$ times.
- $c_4 \times \sum_{j=2}^{n} t_j$: The 'while' loop's condition is checked $\sum_{j=2}^{n} t_j$ times, where t_j is the number of times the loop executes for each j.
- $c_5 \times \sum_{j=2}^{n} (t_j - 1)$: The statement 'arr[i] = arr[i - 1];' inside the 'while' loop executes $\sum_{j=2}^{n} (t_j - 1)$ times.
- $c_6 \times \sum_{j=2}^{n} (t_j - 1)$: The statement 'i -= 1;' also executes $\sum_{j=2}^{n} (t_j - 1)$ times.
- $c_7 \times (n-1)$: The assignment 'arr[i] = key;' executes $n-1$ times.

Total Running Time

Now, the total running time $T(n)$ is:

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot \sum_{j=2}^{n} t_j + c_5 \cdot \sum_{j=2}^{n} (t_j - 1) + c_6 \cdot \sum_{j=2}^{n} (t_j - 1) + c_7 \cdot (n-1)$$

Simplifying

Let's simplify $T(n)$:

$$T(n) = c_1 \cdot n + (c_2 + c_3 + c_7) \cdot (n-1) + (c_4 \cdot \sum_{j=2}^{n} t_j) + (c_5 + c_6) \cdot \sum_{j=2}^{n} (t_j - 1)$$

Best and Worst Case Analysis

Best Case: The best case occurs when the array is already sorted. Here, the 'while' loop never executes, so $t_j = 1$ for all j, leading to:

$$T(n)_{\text{best}} = c_1 \cdot n + (c_2 + c_3 + c_7) \cdot (n-1) + c_4 \cdot (n-1) + 0$$

This simplifies to:

$$T(n)_{\text{best}} = (c_1 + c_2 + c_3 + c_4 + c_7) \cdot (n-1) + c_1$$

Worst Case: The worst case occurs when the array is sorted in reverse order. Here, the 'while' loop executes the maximum number of times for each j, so $t_j = j$ for all j, leading to:

$$T(n)_{\text{worst}} = c_1 \cdot n + (c_2 + c_3 + c_7) \cdot (n-1) + c_4 \cdot \sum_{j=2}^{n} j + (c_5 + c_6) \cdot \sum_{j=2}^{n} (j - 1)$$

This involves more complex summation but can be simplified with known formulas for summing integers.

This process allows you to estimate the time complexity and understand the cost associated with each operation in the algorithm.