## **Code review:**

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time by comparisons. It is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted output.

A screenshot of a computer program

Description automatically generated

## **Analysis:**

### **Best-Case Analysis:**

**When the input array is already nearly sorted, Insertion Sort requires O(n) time complexity. This is because it can quickly determine that each element is already in its correct position and no further swaps are needed.**

### **Worst-Case Analysis**

When the input array is in reverse order, Insertion Sort requires **O(n²)**time complexity. This is because for each element, it may need to compare and swap with all previous elements*.*

### **Average-Case Analysis**

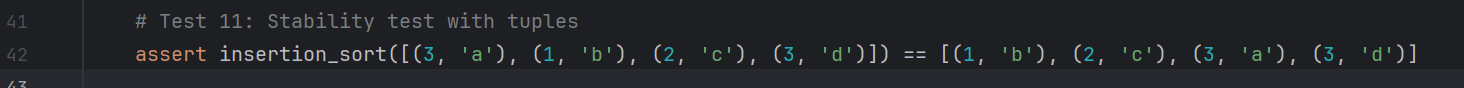
In the average case, the time complexity of Insertion Sort remains O(n²). This scenario involves randomly ordered arrays, where the algorithm performs a number of comparisons and shifts that grow quadratically with the input size. While less efficient than algorithms like QuickSort or MergeSort for large datasets, Insertion Sort can be quite efficient for small datasets or arrays that are already partially sorted.

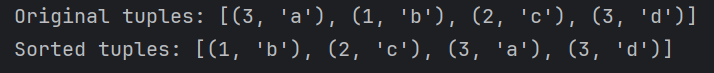
### **Space Complexity Discussion**

Insertion Sort is an in-place sorting algorithm, meaning it sorts the array without needing additional memory proportional to the input size. It only requires a constant amount of extra space for variables such as loop indices. Consequently, the space complexity of Insertion Sort is O(1), indicating constant space usage regardless of the array size.

### **Stability Analysis**

This test ensures that elements with equal keys maintain their original relative order after sorting.





## **Reflection:**

### **Efficiency Discussion:**

Insertion Sort is efficient for small arrays due to its simplicity and low overhead. For small datasets, the difference in performance between O(n) and O(n²) is negligible, making Insertion Sort a viable option. However, for large datasets, Insertion Sort is significantly less efficient than more advanced algorithms like QuickSort or MergeSort, which have better average and worst-case time complexities.

### **Practical Applications:**

* Sorting a small number of records.
* Insertion of new elements into a sorted array.
* Situations where the simplicity and low memory usage of the algorithm are advantageous.

### **Improvements and Variations:**

* Shell Sort - The sorting algorithm compares elements separated by a distance that decreases on each pass. Shell sort has distinctly improved running times in practical work, with two simple variants requiring O(n3/2) and O(n4/3) running time.
* Binary Merge Sort – A variant that uses a binary insertion sort to sort groups of 32 elements, followed by a final sort using merge sort. It combines the speed of insertion sort on small data sets with the speed of merge sort on large data sets.
* Binary Insertion Sort: Uses binary search to find the correct position for the current element, reducing the number of comparisons.