**ASSIGNEMENT-12**

Name –Karthik Thummanapally

Roll-2403A54089

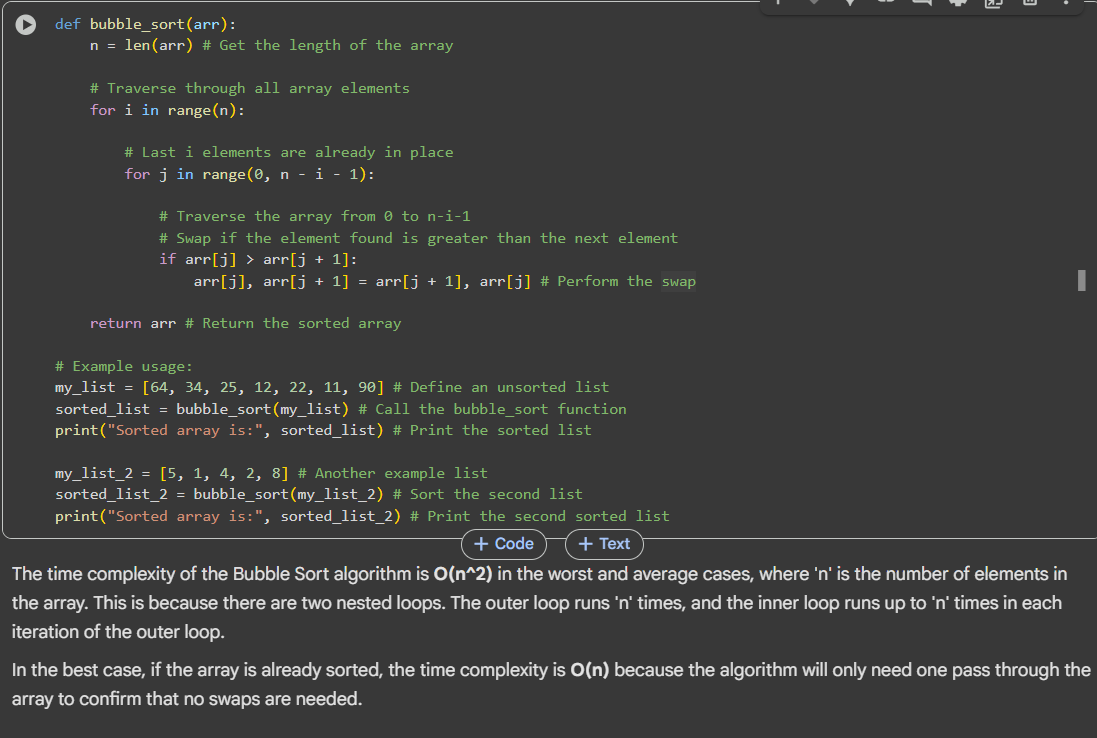
Batch (DS) -03

**Task 1: Implementing Bubble Sort with AI Comments**

**PROMPT:** write a program in python which implements Bubble sort algorithm,provide few examples with inline comments for

every line and provide time complexity analysis in a text block

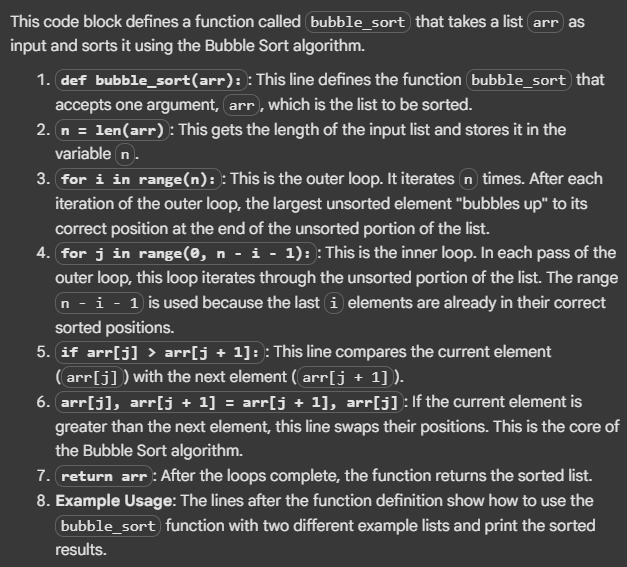
**CODE:**



**OUTPUT:**

****

**EXPLANATION:**

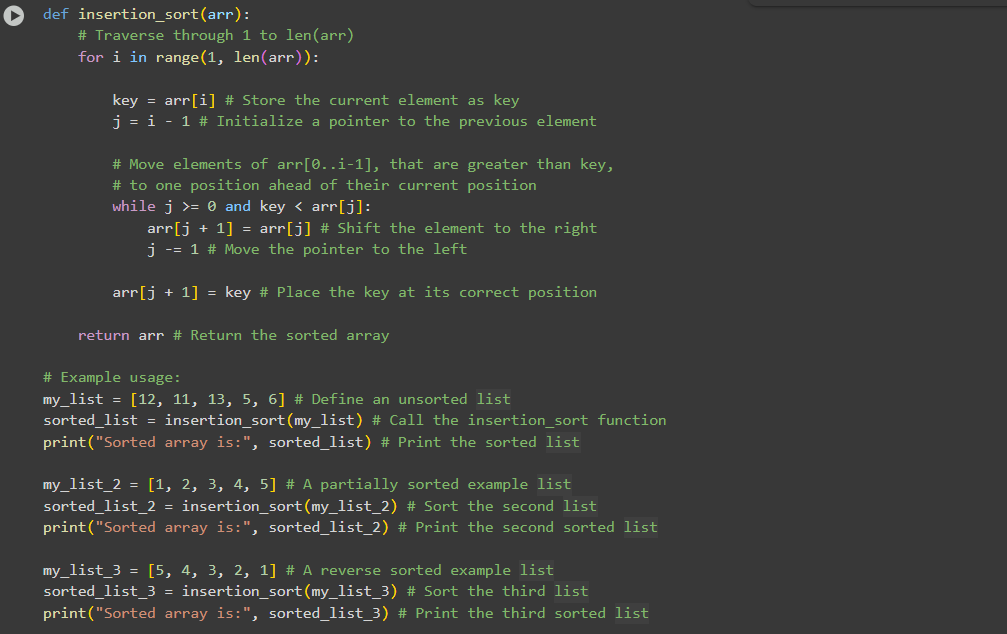
****

**Task 2: Optimizing Bubble Sort → Insertion Sort**

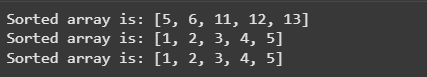
**PROMPT:** Write a program in python which implements bubble sort algorithm,make more efficient code insertion sort for partially sorted arrays,Add inline comments

for every line

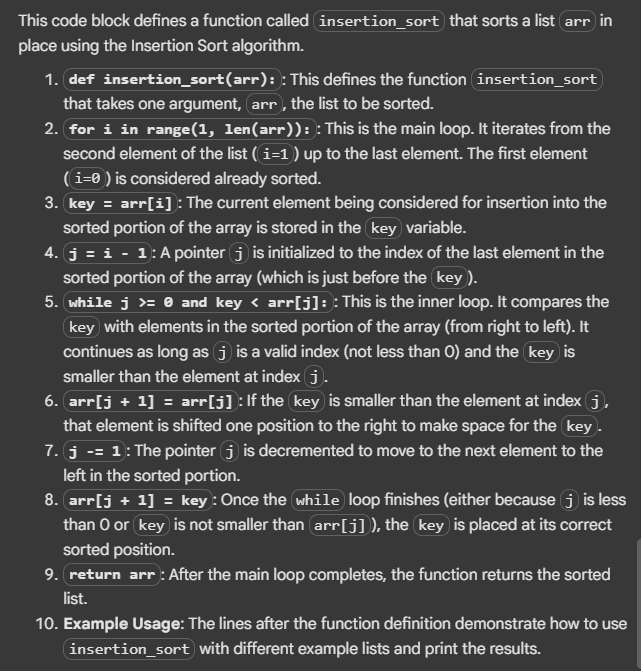
**CODE:**

****

**OUTPUT:**

****

**EXPLANATION:**

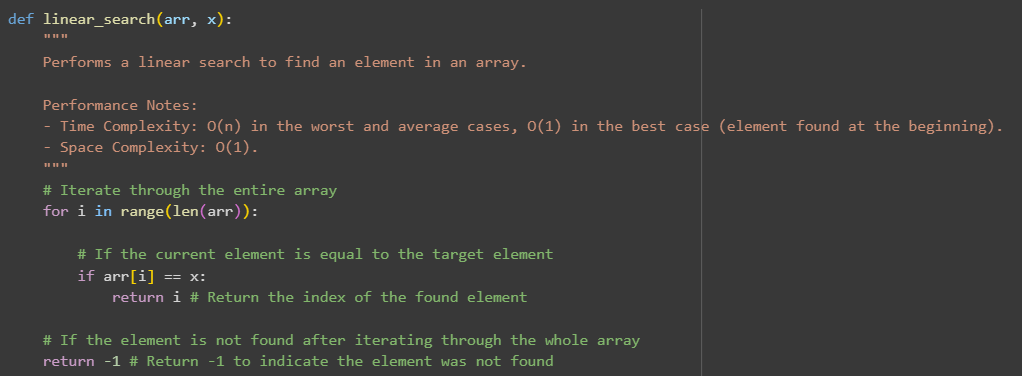
****

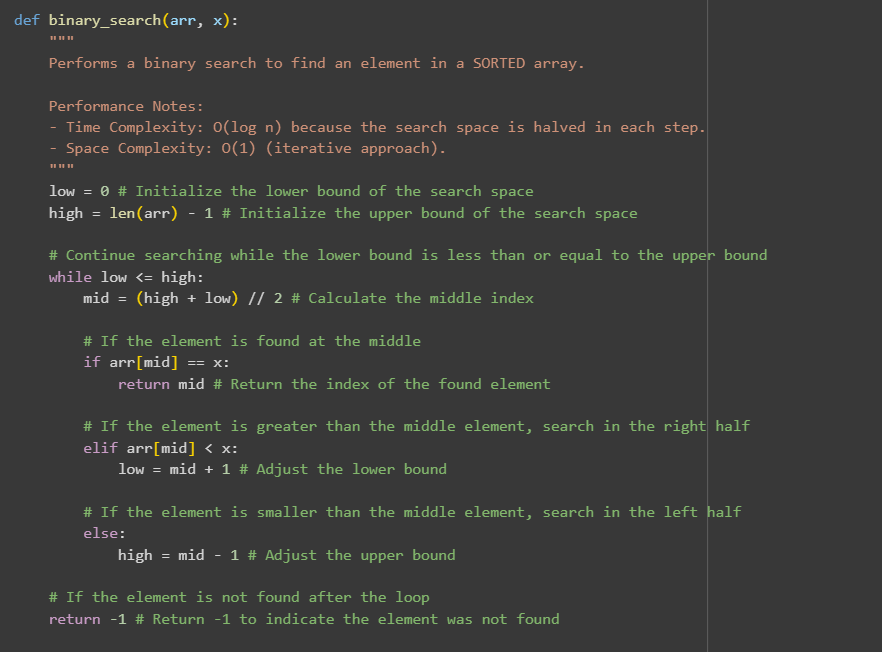
**Task 3: Binary Search vs Linear Search**

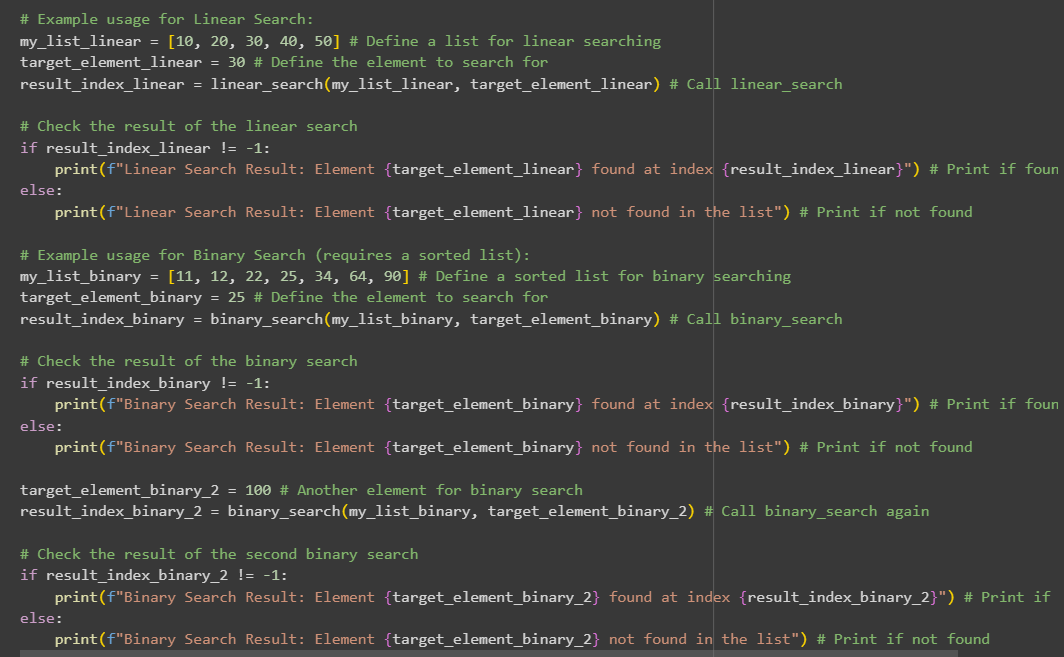
**PROMPT:** write a python code which has bubble sort algorithm and linear search algorithm.Generate docstrings and performance notes for both functions .Provide

inline comments for every line

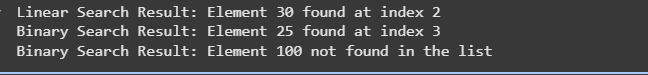
**CODE**:

****

****

****

**OUTPUT:**

****

**EXPLANATION:**

Certainly! This code block contains two functions: linear\_search and binary\_search.

linear\_search(arr, x) Function:

This function performs a linear search to find a target element x in an array arr.

1. for i in range(len(arr)):: It iterates through each element of the array from the beginning to the end.
2. if arr[i] == x:: In each iteration, it checks if the current element arr[i] is equal to the target element x.
3. return i: If the element is found, it returns the index i where the element was found.
4. return -1: If the loop finishes without finding the element, it means the element is not in the array, and the function returns -1.

binary\_search(arr, x) Function:

This function performs a binary search to find a target element x in a *sorted* array arr. Binary search is much faster than linear search for large sorted arrays.

1. low = 0 and high = len(arr) - 1: It initializes two pointers, low and high, to the beginning and end of the array, respectively. These define the current search space.
2. while low <= high:: The search continues as long as the lower bound is less than or equal to the upper bound, meaning there is still a valid search space.
3. mid = (high + low) // 2: In each iteration, it calculates the middle index of the current search space.
4. if arr[mid] == x:: It checks if the element at the middle index arr[mid] is equal to the target element x. If it is, the element is found, and the function returns the middle index mid.
5. elif arr[mid] < x:: If the element at the middle index is less than the target element, it means the target element (if it exists) must be in the right half of the current search space. The lower bound low is updated to mid + 1 to search in the right half.
6. else:: If the element at the middle index is greater than the target element, it means the target element (if it exists) must be in the left half of the current search space. The upper bound high is updated to mid - 1 to search in the left half.
7. return -1: If the while loop finishes without finding the element, it means the element is not in the array, and the function returns -1.

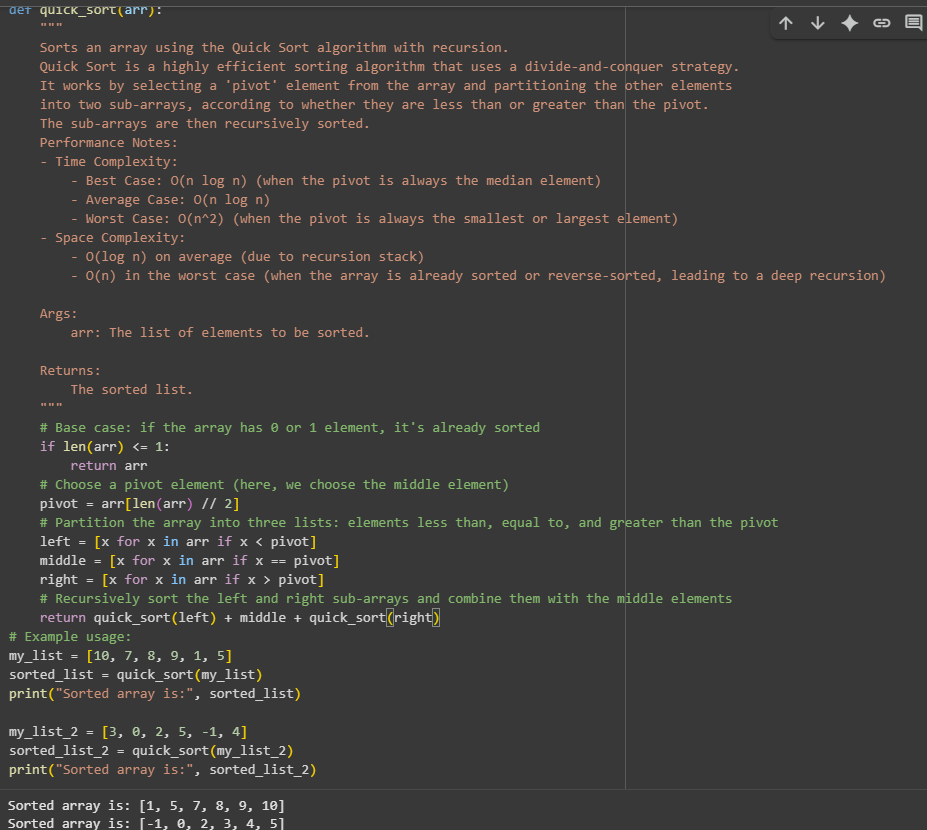
The example usage code at the end demonstrates how to use both functions and print the results.

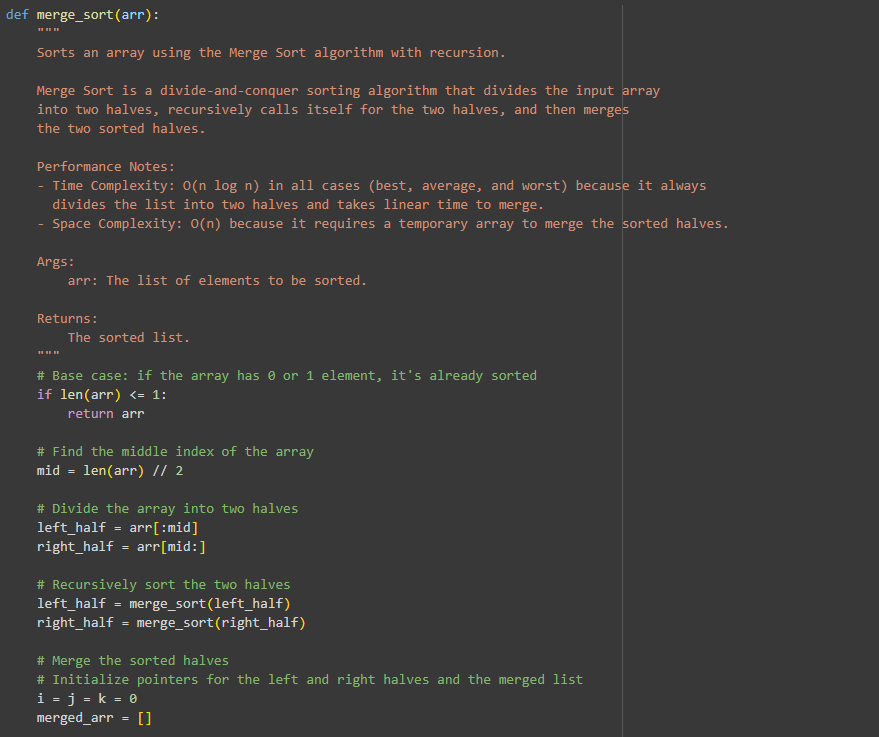
**Task 4: Quick Sort and Merge Sort Comparison**

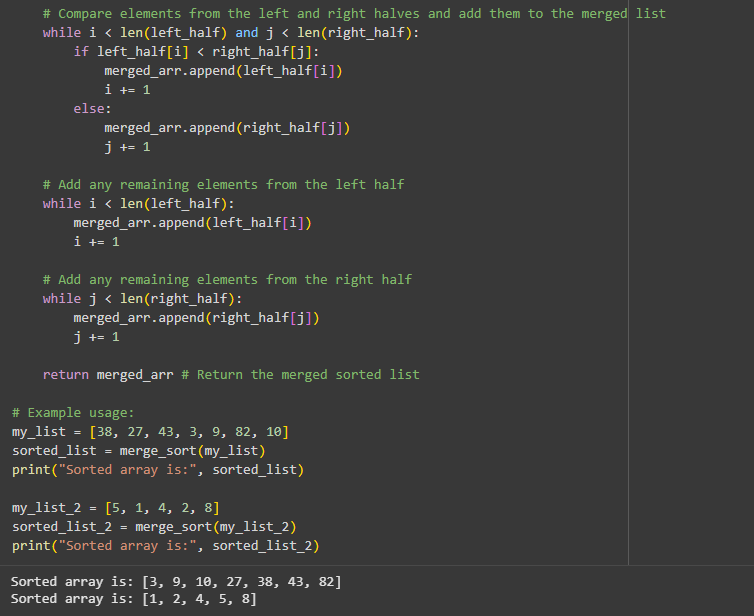
**PROMPT:** write a python code which Implements Quick Sort and Merge Sort using recursion and Compare both algorithms on random, sorted, and reverse-sorted

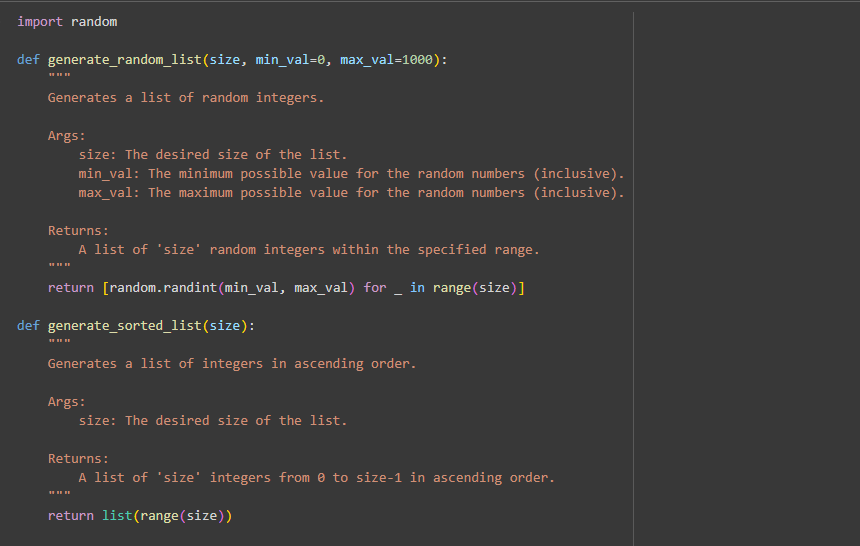
lists. Provide inline comments for every line

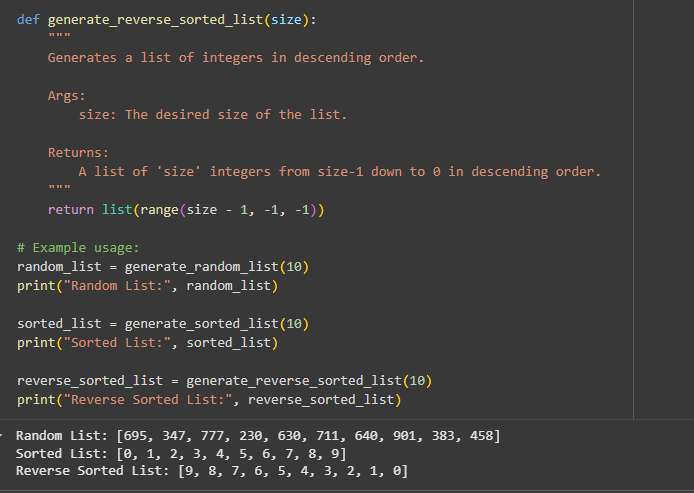
**CODE:**

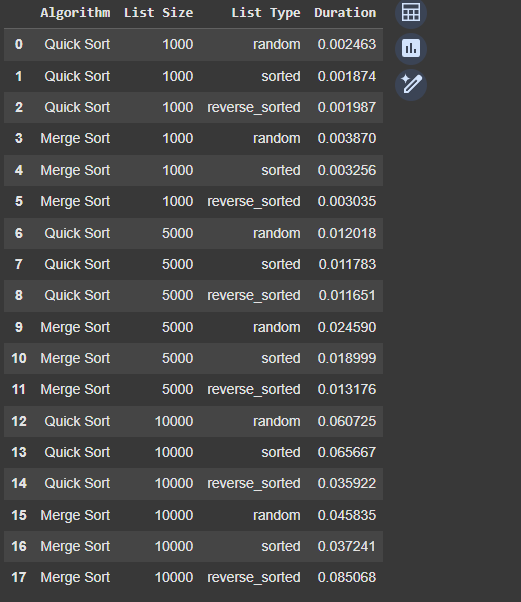


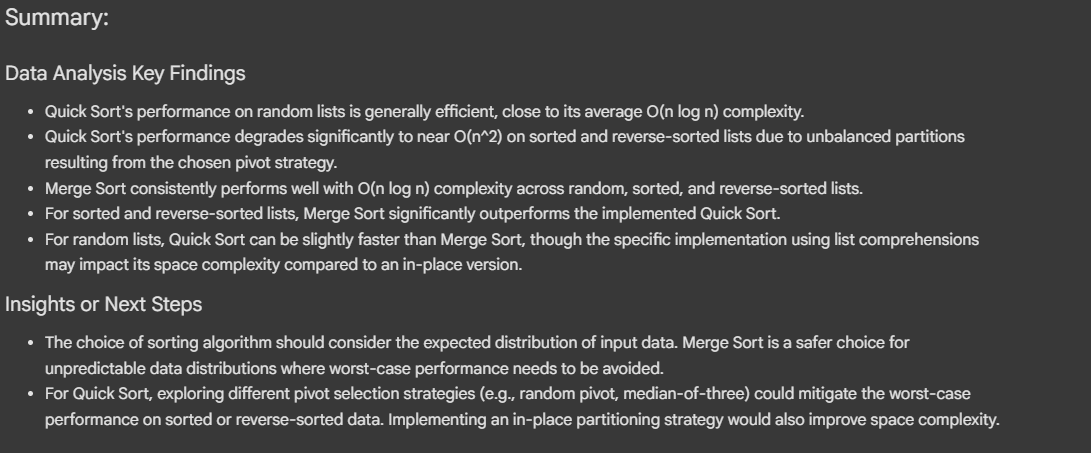
****

****

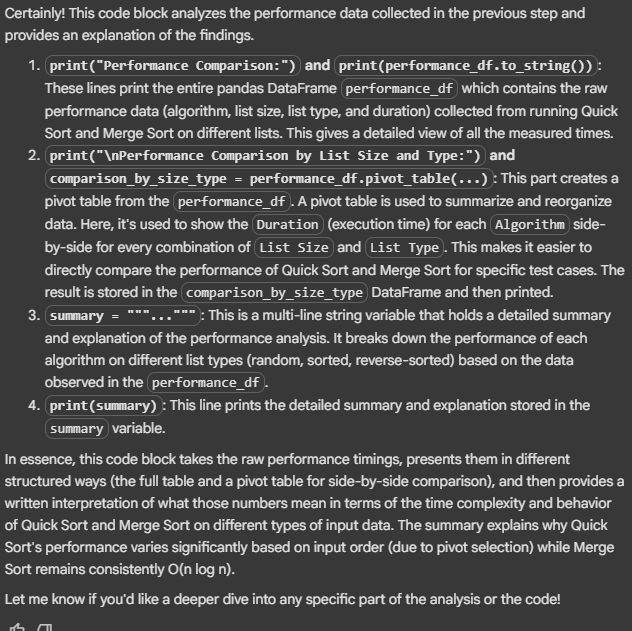
****

****

****

****

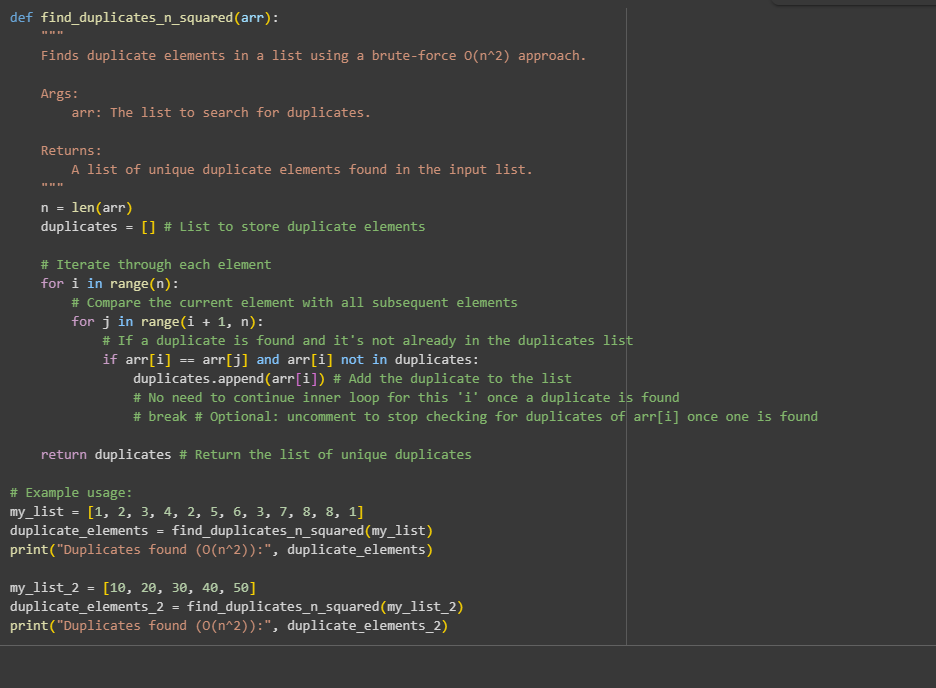
**EXPLANATION:**

****

**Task 5: AI-Suggested Algorithm Optimization**

**PROMPT:** Write a python code which O(n²) duplicate search

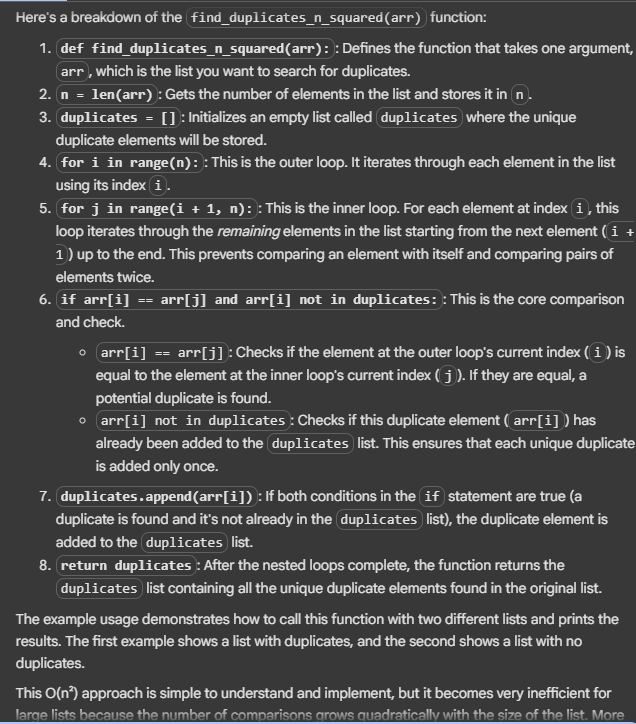
**CODE:**

****

**OUTPUT:**

****

**EXPLANATION:**

****