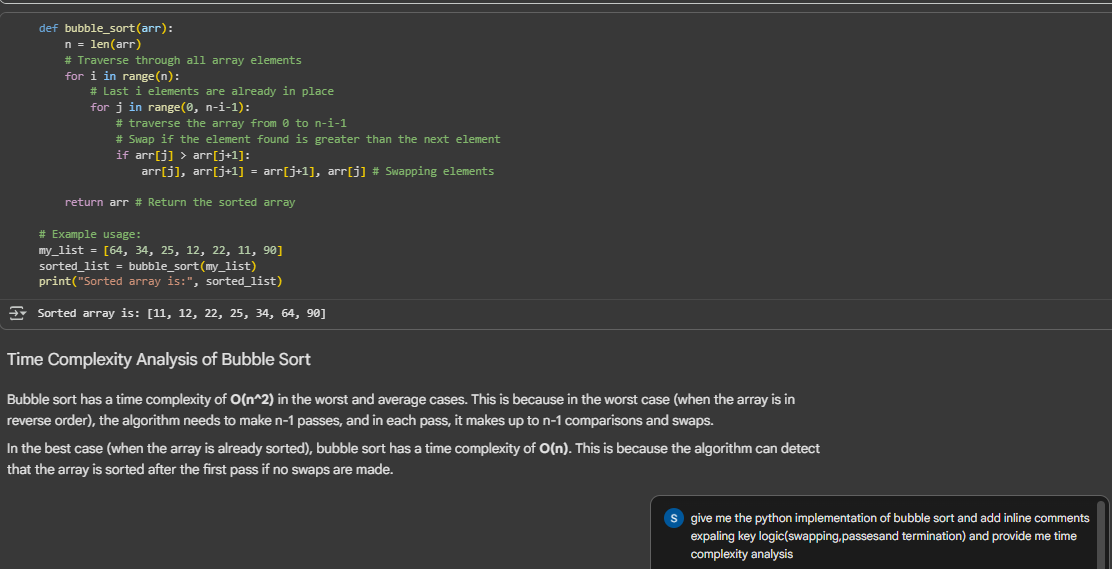
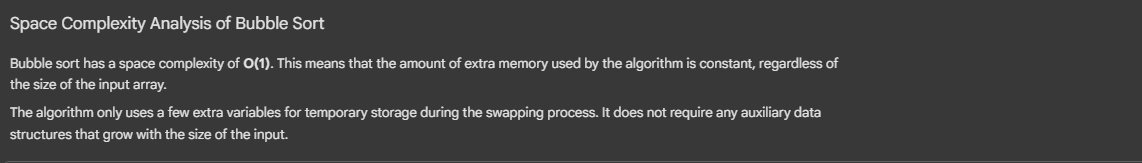
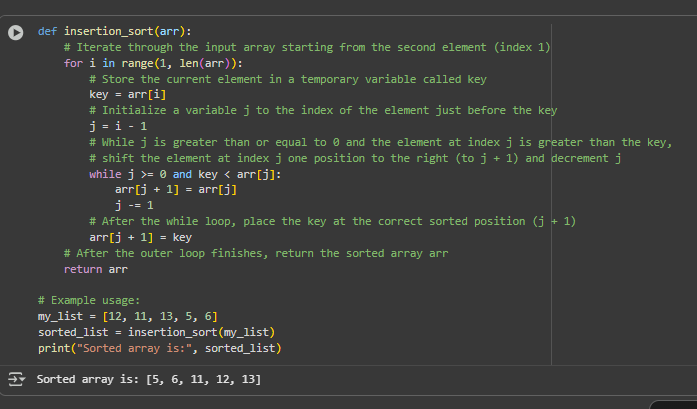
Name:Ch.Rohith

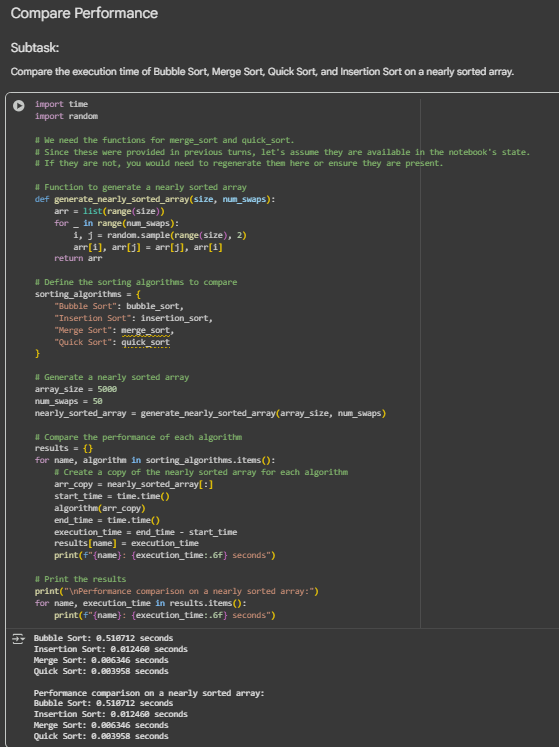
R.no:2403A52178 **Assignment:12.4**

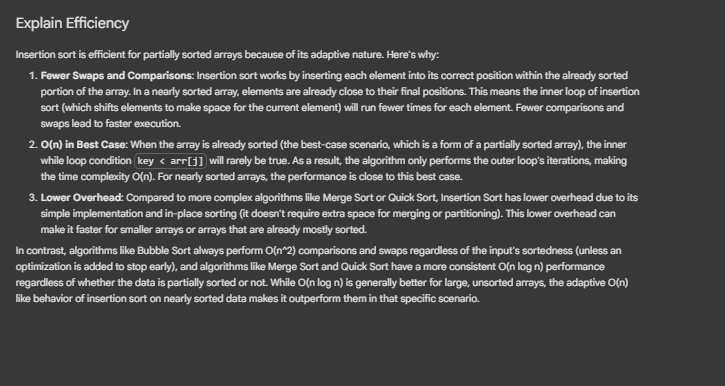
**Task 1**: Implementing Bubble Sort with AI Comments  
• Task: Write a Python implementation of Bubble Sort.  
• Instructions:  
o Students implement Bubble Sort normally.  
o Ask AI to generate inline comments explaining key  
logic (like swapping, passes, and termination).  
o Request AI to provide time complexity analysis.  
• Expected Output:  
o A Bubble Sort implementation with AI-generated  
explanatory comments and complexity analysis



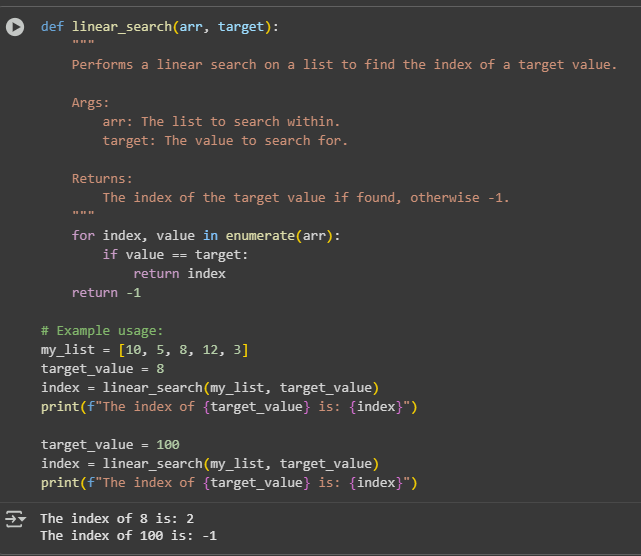
**Task 2**: Optimizing Bubble Sort → Insertion Sort  
• Task: Provide Bubble Sort code to AI and ask it to suggest a  
more efficient algorithm for partially sorted arrays.  
• Instructions:  
o Students implement Bubble Sort first.  
o Ask AI to suggest an alternative (Insertion Sort).  
o Compare performance on nearly sorted input.  
• Expected Output:  
o Two codes (Bubble Sort + Insertion Sort).  
o AI explanation of why Insertion Sort is more efficient for  
partially sorted data.

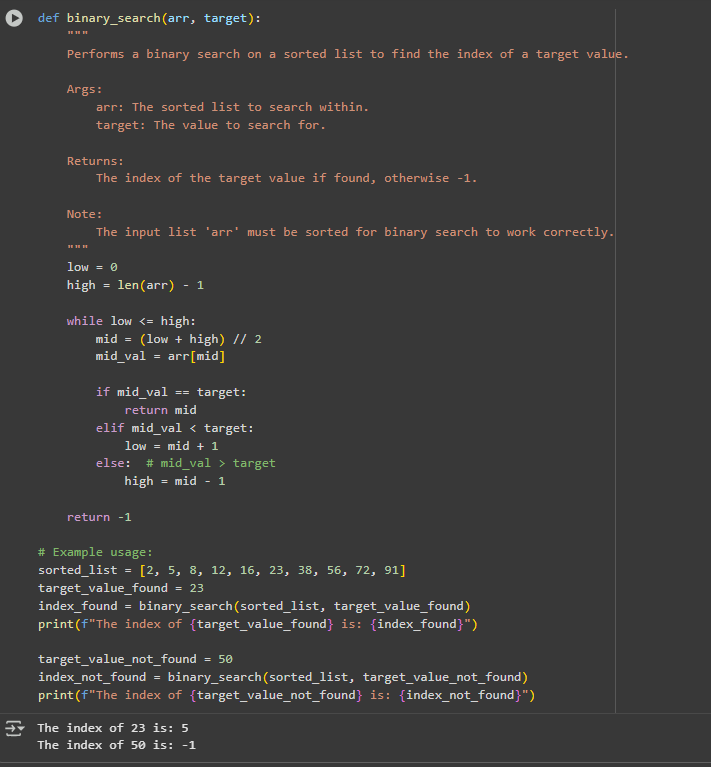


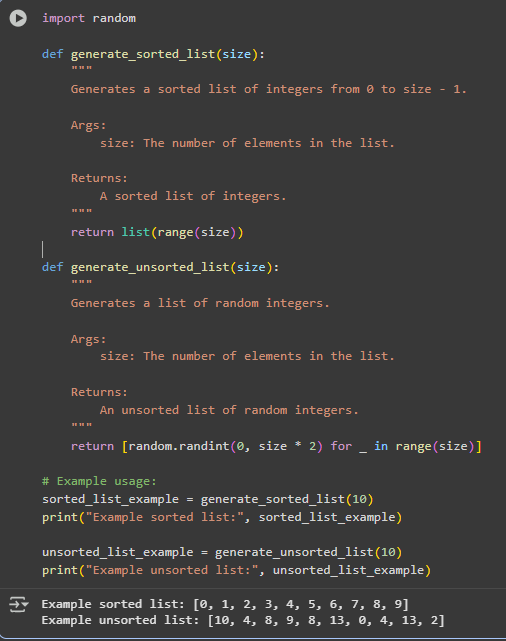


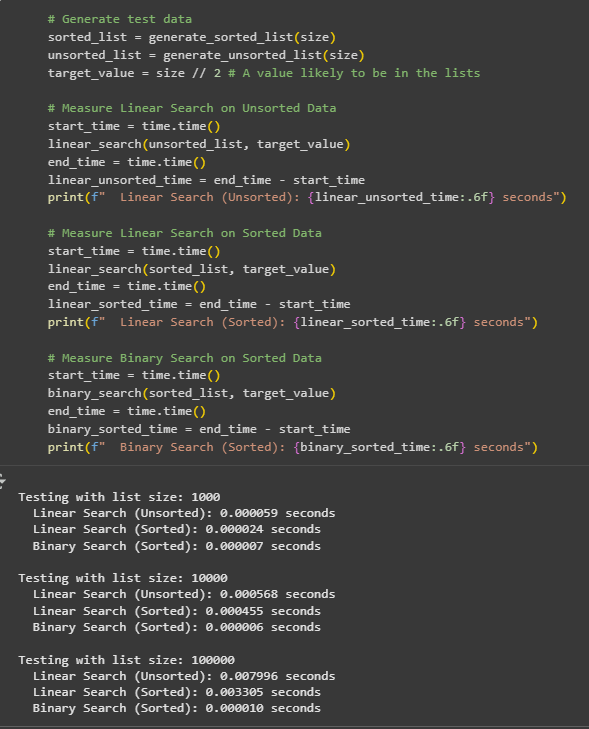


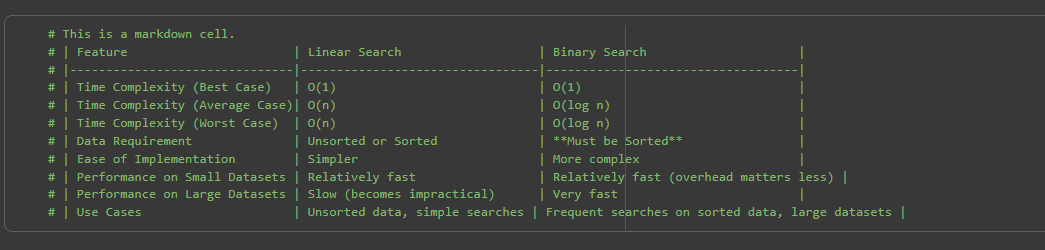
Task 3: Binary Search vs Linear Search  
• Task: Implement both Linear Search and Binary Search.  
• Instructions:  
o Use AI to generate docstrings and performance notes.  
o Test both algorithms on sorted and unsorted data.  
o Ask AI to explain when Binary Search is preferable.  
• Expected Output:  
o Two implementations with docstrings.  
o A student observation table comparing performance  
(Linear vs Binary Search)







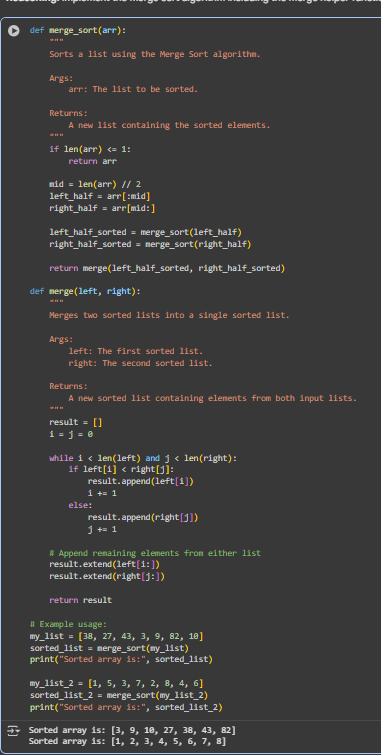




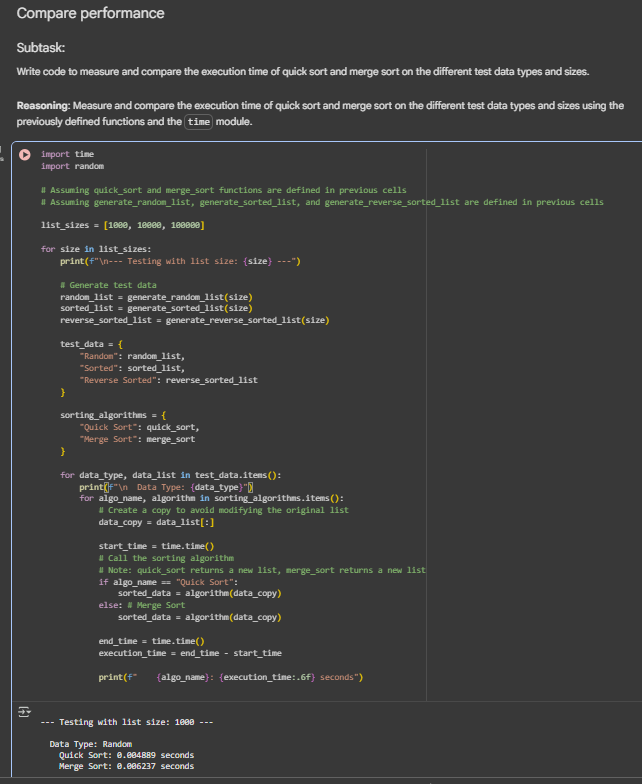
Task 4: Quick Sort and Merge Sort Comparison

• Task: Implement Quick Sort and Merge Sort using recursion.  
• Instructions:  
o Provide AI with partially completed functions for  
recursion.  
o Ask AI to complete the missing logic and add docstrings.  
o Compare both algorithms on random, sorted, and reverse-  
sorted lists.  
• Expected Output:  
o Working Quick Sort and Merge Sort implementations.  
o AI-generated explanation of average, best, and worst-case  
complexities









Task 5: AI-Suggested Algorithm Optimization  
• Task: Give AI a naive algorithm (e.g., O(n²) duplicate search).  
• Instructions:  
o Students write a brute force duplicate-finder.  
o Ask AI to optimize it (e.g., by using sets/dictionaries with  
O(n) time).  
o Compare execution times with large input sizes.  
• Expected Output:  
o Two versions of the same algorithm (brute force +  
optimized).  
o AI explanation of how complexity was improved.

