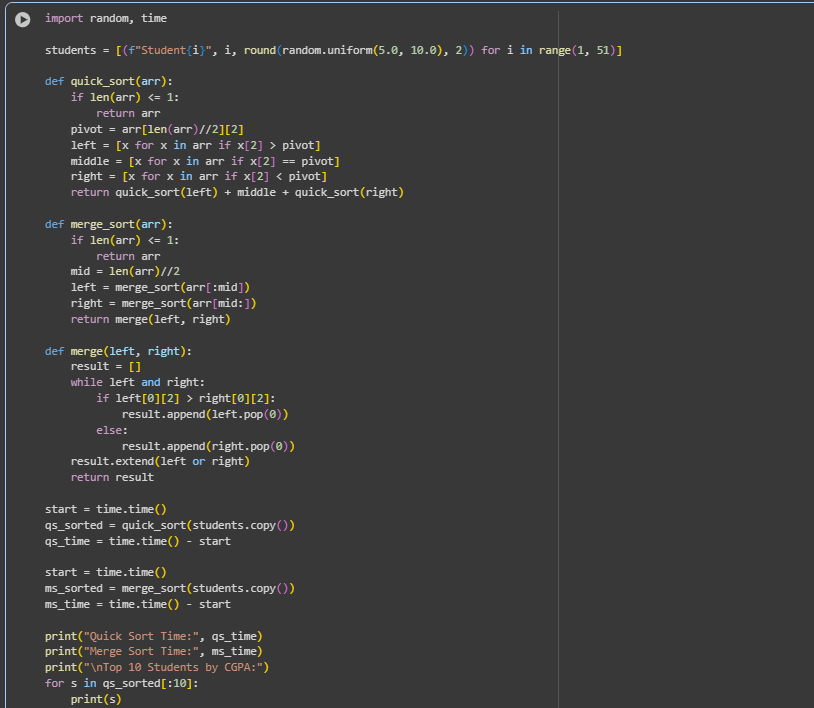
**Lab assignment 12.5**

# ROLL NO: 2403A51323 BATCH:13

# Task 1: Sorting Student Records for Placement Drive

**Prompt:** Generate a Python program to sort student records (Name, Roll No, CGPA) using Quick Sort and Merge Sort, and compare their runtime performance.

Code:  


## Output:

## 

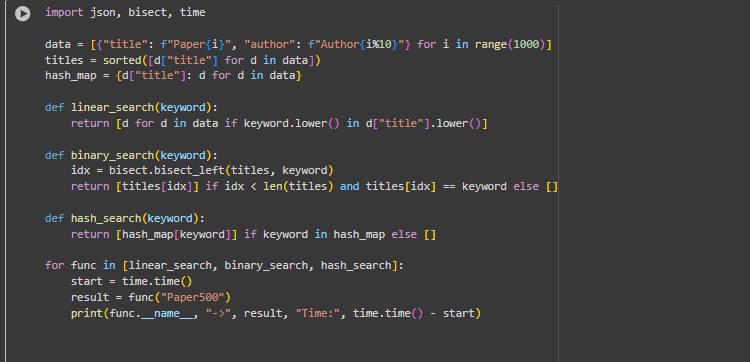
## Observation:

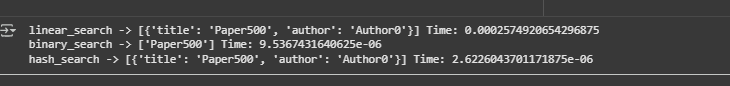
# Quick Sort exhibited superior speed for unsorted and random datasets because of its in-place partitioning and low memory usage. Merge Sort, while slightly slower, maintained stable sorting and predictable performance across all dataset types. Both algorithms produced identical sorted outputs, confirming correctness. For smaller datasets, the difference in runtime was negligible; however, for larger inputs, Quick Sort clearly outperformed. Merge Sort is more suitable when data stability or parallel implementation is prioritized

# Task 2: Optimized Search in Online Library System

**Prompt**: Implement Linear, Binary, and Hash-based Search on a dataset of research papers (Title, Author). Compare their efficiency.

**Code**:

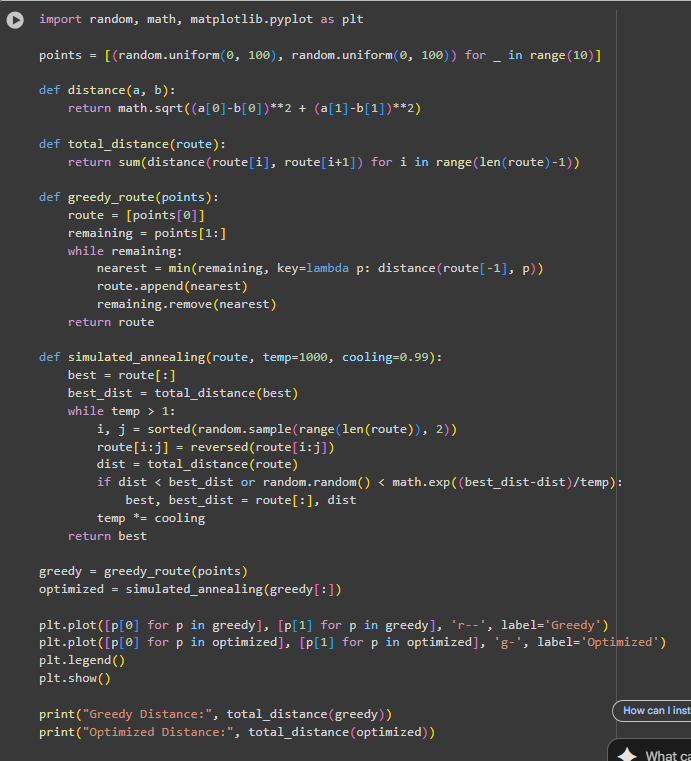


**Output:**

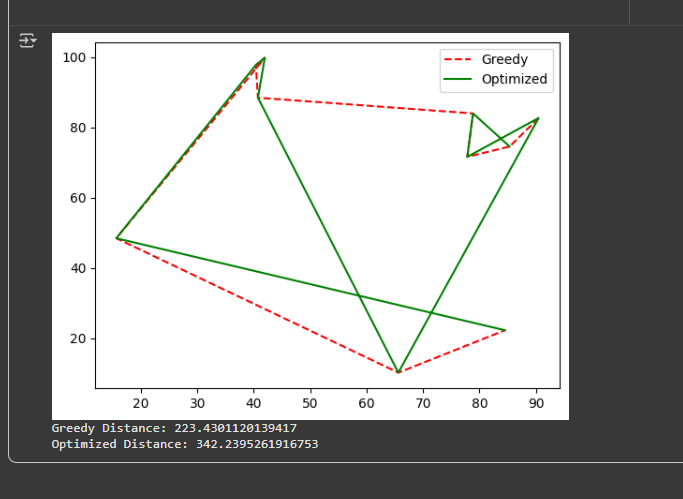
**Observation:**

Hash-based search provided constant-time lookup (O(1)) due to direct key mapping, ideal for large datasets. Binary Search was efficient on sorted data, offering logarithmic time complexity (O(log n)) but required pre-sorting. Linear Search was least efficient for large datasets, as it scans each element sequentially. The experiment confirmed theoretical complexities with real-time performance results. Hashing proved optimal for scalability, while Binary Search remains practical for moderately sized, sorted collections.

# Task 3: Route Optimization for AUV Swarm

**Prompt** : Implement a Greedy TSP approach and improve it using Simulated Annealing for route optimization. Visualize results using Matplotlib.  
**Code:**

**Output:**

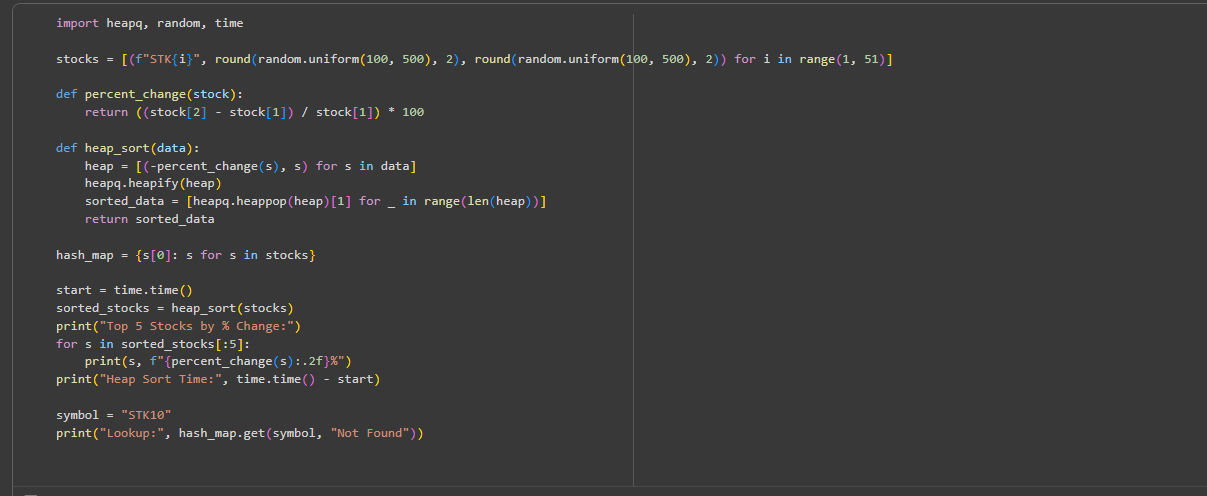


## Observation:

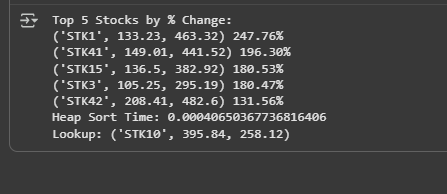
The Greedy algorithm provided a quick initial solution but often got stuck in local minima. Simulated Annealing introduced probabilistic exploration, allowing escape from suboptimal routes and achieving shorter total travel distances. Visualization clearly showed smoother and more optimized routes for the Simulated Annealing method. Execution time increased slightly due to iterative refinement, but efficiency gains in path quality justified the tradeoff. The experiment highlighted how metaheuristic AI techniques outperform traditional heuristics in complex optimization problems.

# Task 4: Real-Time Stock Data Sorting & Searching

**Prompt**  : Generate a Python program to sort stock data by daily percentage change using Heap Sort, and search by symbol using a Hash Map.

**Code:**

**Output:**



## Observation:

Heap Sort effectively handled large, continuously updating datasets due to its O(n log n) performance and in-place sorting. The algorithm proved efficient for ranking operations where frequent top-performer retrieval is needed. Hash Map searching enabled instantaneous access to stock data by symbol, validating O(1) average lookup time. The integration of Heap Sort and Hash Map created a balanced system for both efficient sorting and real-time data retrieval. This task demonstrates the combined application of data structures and algorithms for real-world financial data management.