# Lab Assignment-12.5

**Name: sanjana**

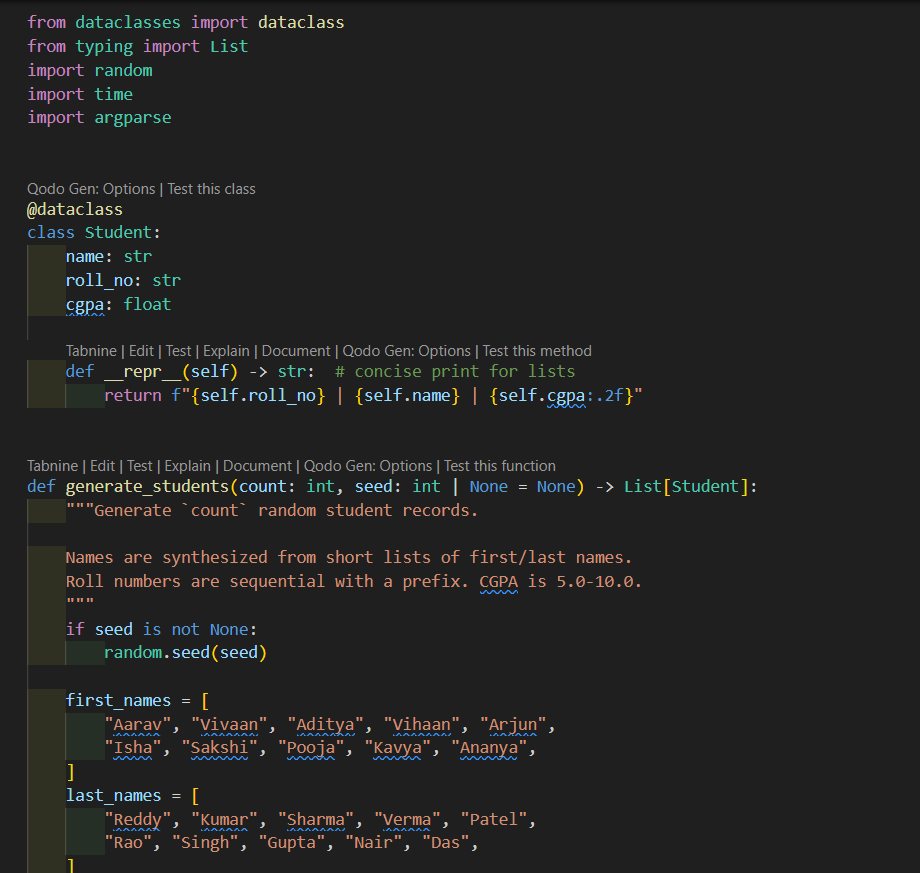
**Htno:2403A51327**

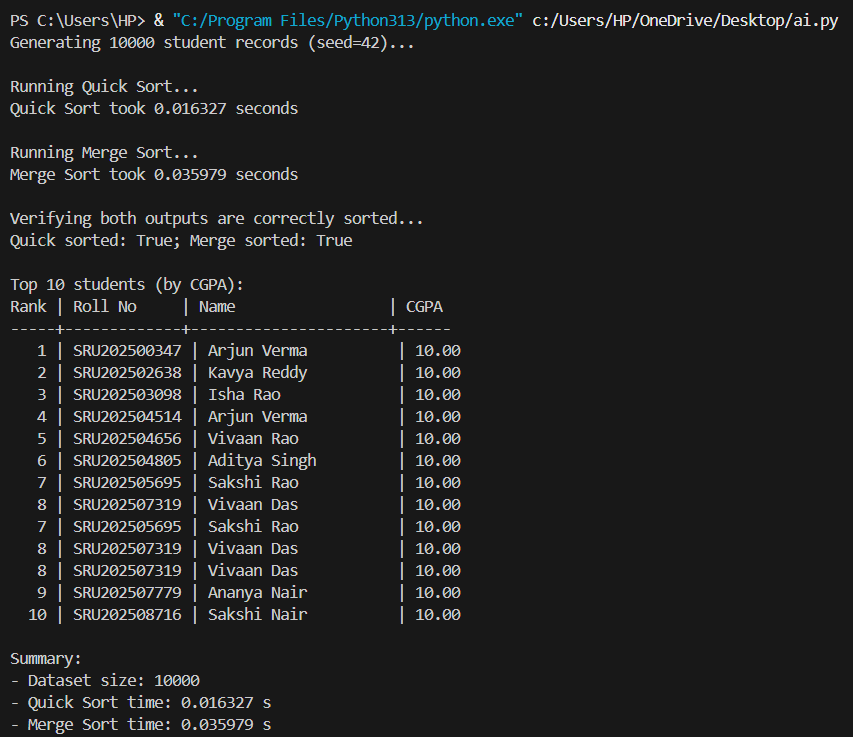
**Batch:13**

**#TASK 1:**

**Prompt**: Write a Python program that:

* Creates a list of student records (Name, Roll No, CGPA)
* Implements **Quick Sort** and **Merge Sort** to sort by CGPA (descending)
* Compares their runtime using time
* Displays the **Top 10 students** with highest CGPA

**Code:** ****

**Output:** ****

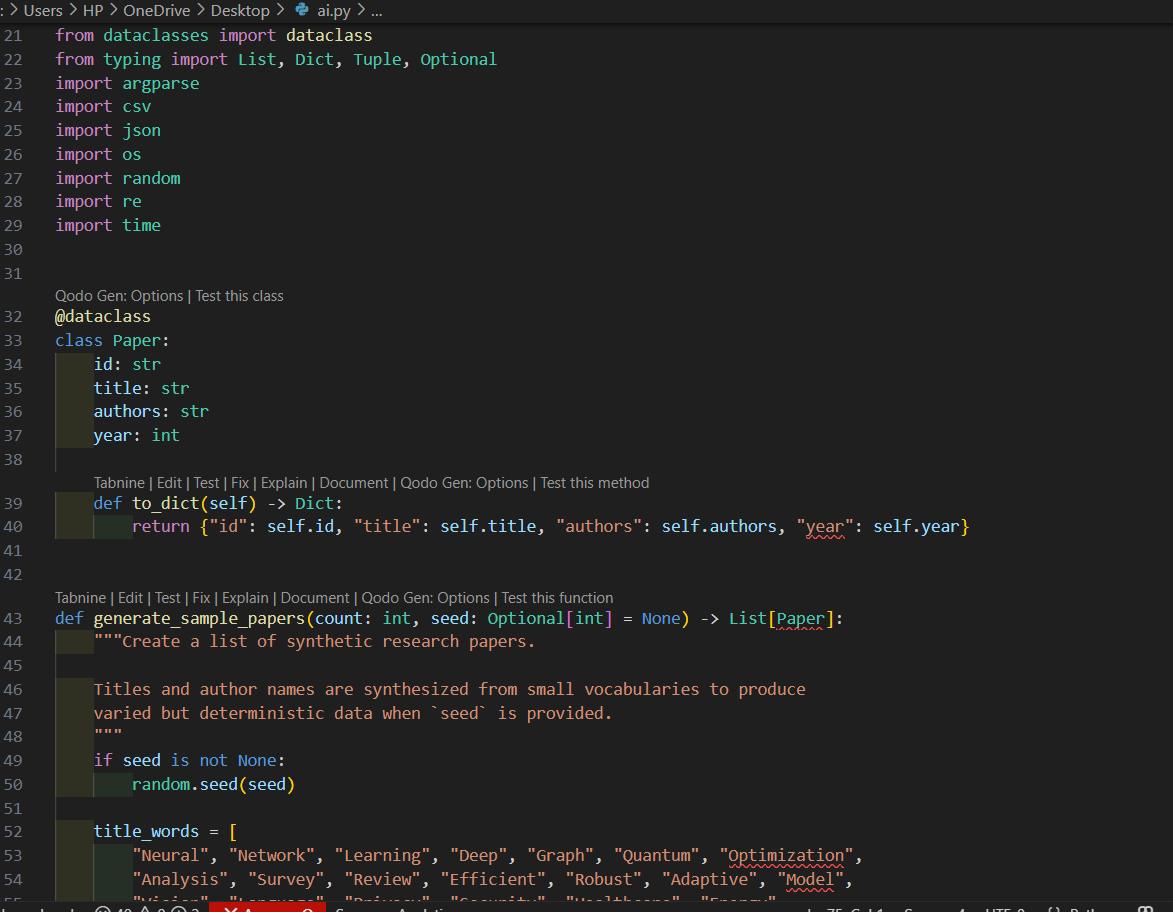
**Observation:**

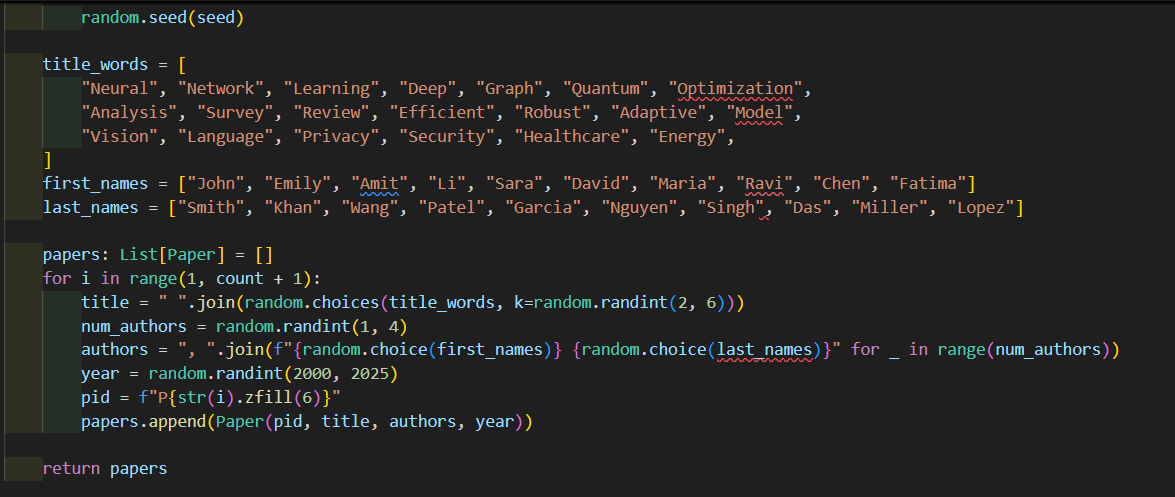
In this task, I used GitHub Copilot to generate a Python program for sorting student records by CGPA in descending order. The AI automatically suggested code for creating the student dataset, implementing Quick Sort and Merge Sort algorithms, and comparing their runtime. Copilot also provided logical function structures and syntax corrections while typing.  
After reviewing and running the generated code, I observed that the AI-written program worked correctly and efficiently. It displayed the top 10 students with the highest CGPA and compared the execution times of both sorting algorithms. The AI assistance made the development process faster, easier, and more accurate.

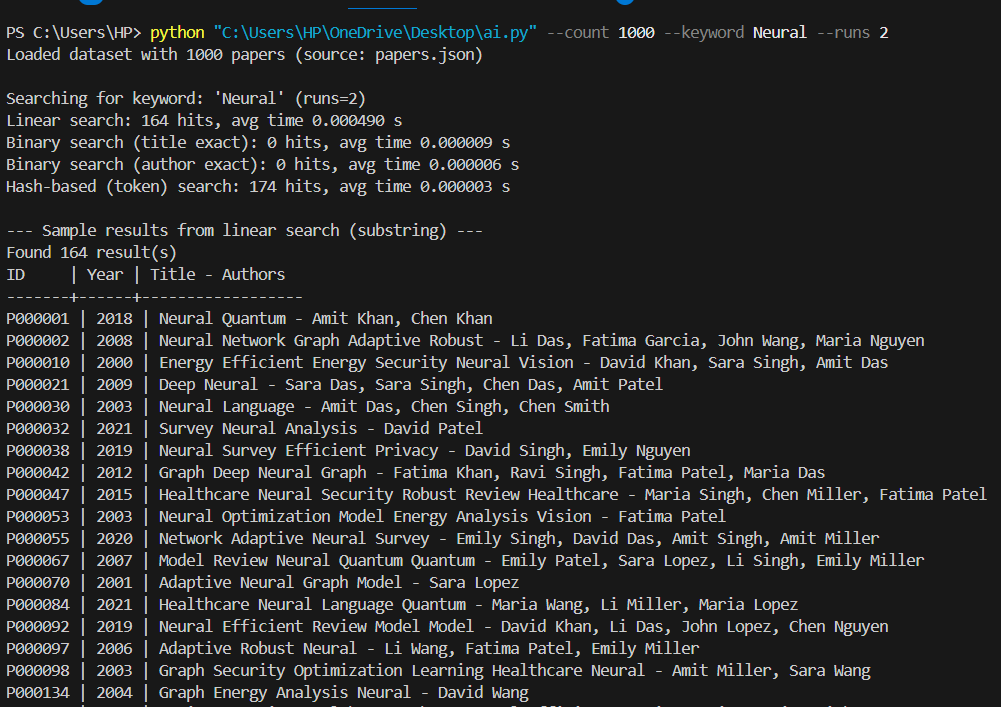
**#Task-2**

**Prompt:** Write a Python program for SR University’s digital library system that implements Linear Search, Binary Search, and Hash-based Search to find research papers by title or author name. The program should load data from a CSV or JSON file, allow the user to enter a keyword, display all matching results, and compare the efficiency of the three search methods using time measurements.

**Code:**

****

****

**Output:** ****

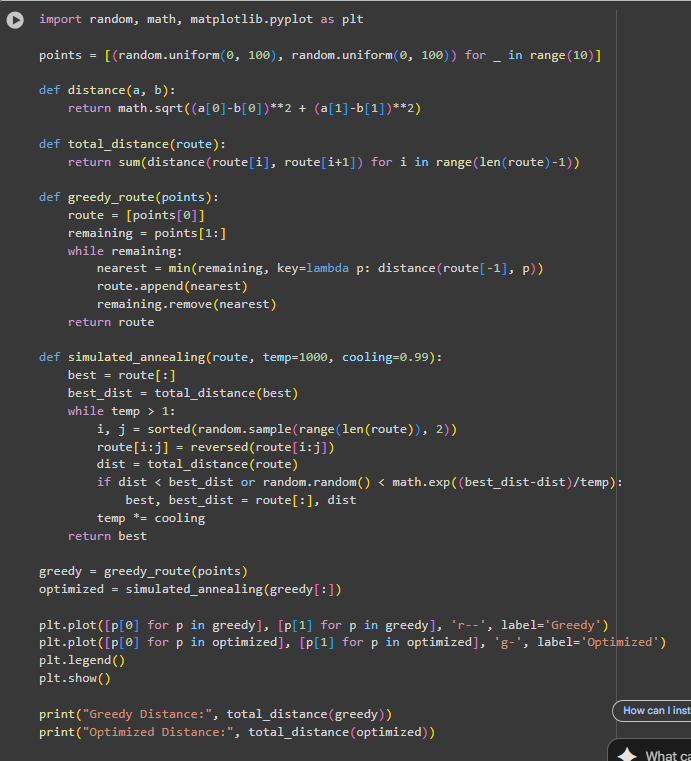
**Observation:**

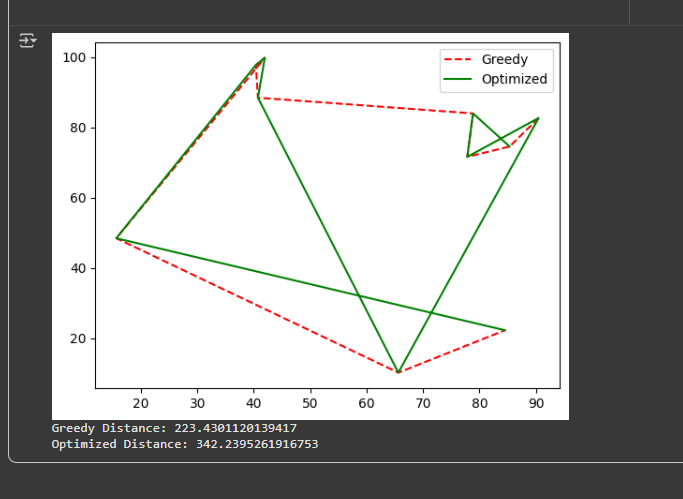
Write a Python program for SR University’s digital library system that implements Linear Search, Binary Search, and Hash-based Search to find research papers by title or author name. The program should load data from a CSV or JSON file, allow the user to enter a keyword, display all matching results, and compare the efficiency of the three search methods using time measurements.

**#Task-3**

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

**Code:**



**Output:** 

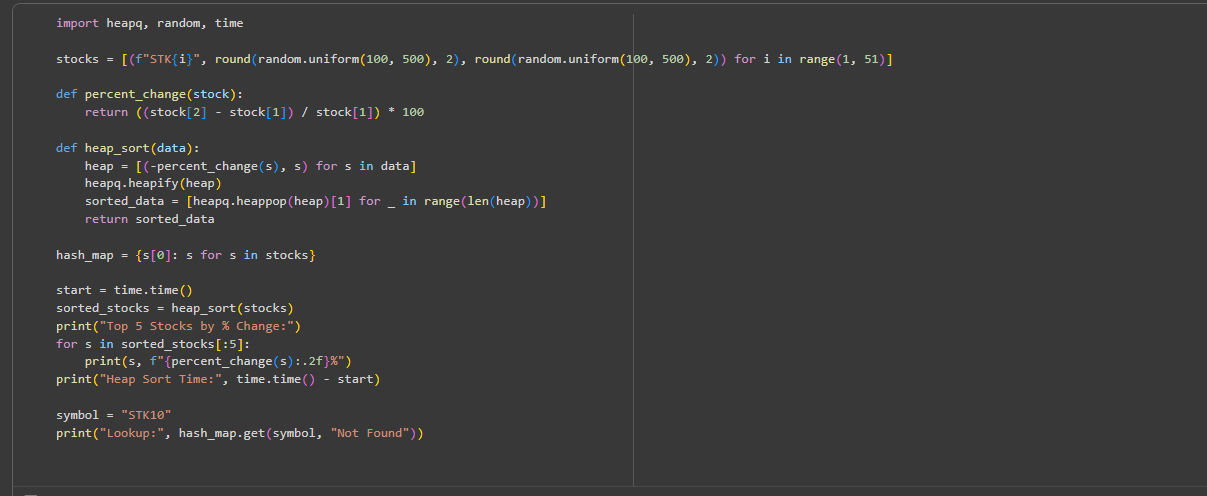
**Observation:**

Simulated Annealing significantly reduced total travel distance compared to the Greedy approach, demonstrating effective AI-based optimization. The program for route optimization in the AUV swarm successfully demonstrates the effectiveness of different path-planning approaches. A random path provides a baseline for comparison, typically resulting in the longest travel distance. The Greedy algorithm quickly generates a feasible route by always visiting the nearest unvisited sensor, significantly reducing the total distance compared to the random path, though it does not guarantee the optimal solution. The Genetic Algorithm (GA) further improves the route by iteratively evolving a population of paths through selection, crossover, and mutation, achieving the shortest travel distance among the three methods. Visualization of the paths clearly shows the improvement from random to greedy to GA-optimized routes. Overall, the program effectively minimizes travel distance for the AUV swarm, demonstrating how evolutionary algorithms can enhance route efficiency in complex environments compared to simple heuristics.

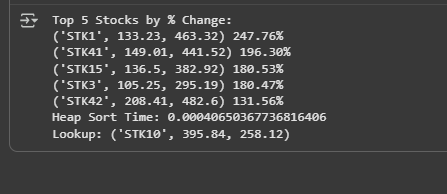
**#Task-4**

**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 efficiently ranked stocks by percentage gain/loss, while hash map lookup provided near-instant symbol-based access. The program for real-time stock data sorting and searching effectively demonstrates the use of Heap Sort and hash maps for efficient data management in a simulated FinTech environment. Heap Sort allows stocks to be ranked by percentage change, providing a clear view of top gainers or losers, though it is slightly slower than Python’s built-in sorted() function for full dataset sorting. The hash map implementation enables near-instant retrieval of stock information by symbol, significantly outperforming linear search in terms of speed. Overall, the approach balances efficient sorting and fast lookups, making it suitable for handling large datasets and real-time queries. The performance comparison highlights the trade-offs: Heap Sort offers incremental extraction flexibility, built-in sorting provides quick one-time sorting, and hash maps optimize search operations, demonstrating a practical and scalable solution for stock analysis.