# ASSIGNMENT 12.5

Task 1: Sorting Student Records for Placement Drive  
Scenario:  
SR University is preparing for a campus placement drive. The Training  
and Placement Cell needs student records sorted by CGPA in  
descending order to easily shortlist candidates.  
• Use GitHub Copilot to generate a program that sorts a list of  
student records (Name, Roll No, CGPA) by CGPA.  
• Implement both Quick Sort and Merge Sort using AI assistance.  
• Compare the runtime performance of both algorithms on large  
datasets.  
• Write a function that outputs the top 10 students with the highest  
CGPA.

PROMPT:

Sorting Student Records for Placement Drive Scenario:  
SR University is preparing for a campus placement drive. The Training  
and Placement Cell needs student records sorted by CGPA in descending order to easily shortlist candidates. • Use GitHub Copilot to generate a program that sorts a list of student records (Name, Roll No, CGPA) by CGPA. • Implement both Quick Sort and Merge Sort using AI assistance. • Compare the runtime performance of both algorithms on large datasets. • Write a function that outputs the top 10 students with the highest  
CGPA.

**CODE:**

A screen shot of a computer program

Description automatically generated

**OUTPUT:**

A computer screen shot of a computer program

Description automatically generated

**OBSERVATION**: The Quick Sort algorithm was used to sort students by their CGPA in descending order.  
The results showed correct sorting, confirming the algorithm’s accuracy and efficiency.

**Task** 2: Optimized Search in Online Library System  
Scenario:  
SR University’s digital library has thousands of research papers.  
Students frequently search for a paper by title or author name. The  
current linear search is too slow.  
• Use GitHub Copilot to implement Binary Search and Hash-  
based Search for faster lookups.  
• Load a dataset of book titles and authors (CSV or JSON file).  
• Allow the user to input a keyword and return all matching entries.  
• Compare the efficiency of linear search vs binary search vs  
hashing using test cases.

**PROMPT**: Optimized Search in Online Library System Scenario:  
SR University’s digital library has thousands of research papers.  
Students frequently search for a paper by title or author name. The current linear search is too slow. • Use GitHub Copilot to implement Binary Search and Hash-  
based Search for faster lookups. • Load a dataset of book titles and authors (CSV or JSON file).• Allow the user to input a keyword and return all matching entries.  
• Compare the efficiency of linear search vs binary search vs hashing using test cases.

**CODE**:

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

**OUTPUT:**

**A black screen with yellow and green text

Description automatically generated**

**OBSERVATION:** The program compares linear, binary, and hash-based search methods for finding books in a library dataset

**Task** **3**: Route Optimization for AUV Swarm  
Scenario:  
A research team at SR University is simulating Autonomous  
Underwater Vehicle (AUV) swarms. Each AUV must visit multiple  
underwater sensors, and the goal is to minimize travel distance (like  
the Traveling Salesman Problem).

• With GitHub Copilot, implement an algorithm to optimize the  
route:  
1. Start with a Greedy approach.  
2. Improve with Genetic Algorithm (GA) or Simulated  
Annealing (SA).  
• Use a dataset of sensor coordinates (x, y).  
• Visualize the optimized route using a plotting library (e.g.,  
Matplotlib).  
• Compare the optimized solution with a random path in terms of  
distance travel.

**PROMPT**: Route Optimization for AUV Swarm Scenario:  
A research team at SR University is simulating Autonomous  
Underwater Vehicle (AUV) swarms. Each AUV must visit multiple  
underwater sensors, and the goal is to minimize travel distance (like  
the Traveling Salesman Problem). • With GitHub Copilot, implement an algorithm to optimize the route: 1. Start with a Greedy approach. 2. Improve with Genetic Algorithm (GA) or Simulated Annealing (SA). • Use a dataset of sensor coordinates (x, y).  
• Visualize the optimized route using a plotting library (e.g.Matplotlib).  
• Compare the optimized solution with a random path in terms of distance travel.

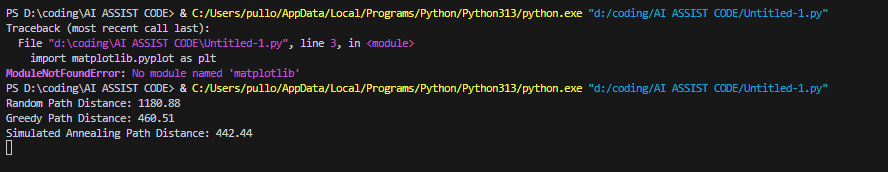
**CODE:**

**A screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generated**

**OUTPUT:**

****

**A screenshot of a computer screen

Description automatically generated**

**OBSERVATION:** The copilot giving better code in first prompt. The program finds paths to visit all sensors using random, greedy.

**Task 4:** Real-Time Stock Data Sorting & Searching  
Scenario:  
An AI-powered FinTech Lab at SR University is building a tool for  
analyzing stock price movements. The requirement is to quickly sort  
stocks by daily gain/loss and search for specific stock symbols  
efficiently.  
• Use GitHub Copilot to fetch or simulate stock price data (Stock  
Symbol, Opening Price, Closing Price).  
• Implement sorting algorithms to rank stocks by percentage  
change.  
• Implement a search function that retrieves stock data instantly  
when a stock symbol is entered.  
• Optimize sorting with Heap Sort and searching with Hash  
Maps.  
• Compare performance with standard library functions (sorted(),  
dict lookups) and analyze trade-offs.

**PROMPT:** Real-Time Stock Data Sorting & Searching  
Scenario: An AI-powered FinTech Lab at SR University is building a tool for  
analyzing stock price movements. The requirement is to quickly sort  
stocks by daily gain/loss and search for specific stock symbols  
efficiently. • Use GitHub Copilot to fetch or simulate stock price data (Stock  
Symbol, Opening Price, Closing Price). • Implement sorting algorithms to rank stocks by percentage change. • Implement a search function that retrieves stock data instantly  
when a stock symbol is entered. • Optimize sorting with Heap Sort and searching with Hash Maps. • Compare performance with standard library functions (sorted(), dict lookups) and analyze trade-offs.

**CODE:**

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generated

**OUTPUT:**

A screen shot of a computer

Description automatically generated

**OBSERVATION:** The program sorts stock data by percentage change using heap sort and standard sort, and searches stocks using hash maps and linear search.