

Assignment12.5

Name:- U.Siddhartha
Hall No:-2403A54122

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.

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows two files: `task1.py` and `task2.py`. `task1.py` is the active file.
- Code Editor:** Displays the following Python code for generating student data and sorting them by CGPA:

```
1  from __future__ import annotations
2  import random
3  import string
4  import time
5  from dataclasses import dataclass
6  from typing import List, Tuple
7
8
9  @dataclass
10 class Student:
11     name: str
12     roll: str
13     cgpa: float
14
15     def __repr__(self) -> str:
16         return f"{self.name} ({self.roll}): {self.cgpa:.2f}"
17
18
19 def generate_random_name(length: int = 6) -> str:
20     return ''.join(random.choices(string.ascii_uppercase, k=length))
21
22
23 def generate_students(n: int, seed: int | None = 0) -> List[Student]:
24     if seed is not None:
25         random.seed(seed)
26     students: List[Student] = []
27     for i in range(1, n + 1):
28         name = generate_random_name()
29         roll = f'R{100000 + i}'
30         # CGPA between 5.0 and 10.0
31         cgpa = round(random.uniform(5.0, 10.0), 2)
32         students.append(Student(name, roll, cgpa))
33
34
35
36 def quicksort_students(arr: List[Student]) -> List[Student]:
37     """
```

- Output Panel:** Shows the output of running `task1.py`, including sorting results, benchmark timings, and next steps.
- Bottom Status Bar:** Shows file statistics (Line 1, Col 1), spaces used (Spaces: 4), encoding (UTF-8), Python version (3.13.7), and system status (ENG IN, 10:09 PM, 28-11-2025).

Output:-

```
task1.py > ... task2.py
1 from future import annotations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\SIDDHARTHA\AI All Lab Assignments & "C:/Users/SIDDHARTHA/AI All Lab Assignments/.venv/Scripts/python.exe" "c:/Users/SIDDHARTHA/AI All Lab Assignments/task1.py"

Benchmarking Quick Sort vs Merge Sort on student records (by CGPA, descending)

Dataset size: 1000
Quick Sort: 0.00020s
Merge Sort: 0.0015s
Top 5 (QuickSort): [SGNGMB (R100395): 9.98, CRGVXB (R100414): 9.98, FJDFTN (R100530): 9.98, UMSND (R100685): 9.98, UHMDMP (R100931): 9.98]

Dataset size: 5000
Quick Sort: 0.0101s
Merge Sort: 0.0119s
Top 5 (QuickSort): [LTMMIT (R101826): 10.00, LWRDMM (R102759): 10.00, IHVRHS (R101079): 9.99, ZLNMTN (R101206): 9.99, VIXUTU (R102748): 9.99]

Dataset size: 20000
Quick Sort: 0.0400s
Merge Sort: 0.0695s
Quick Sort: 0.0101s
Merge Sort: 0.0119s
Top 5 (QuickSort): [LTMMIT (R101826): 10.00, LWRDMM (R102759): 10.00, IHVRHS (R101079): 9.99, ZLNMTN (R101206): 9.99, VIXUTU (R102748): 9.99]

Dataset size: 20000
Quick Sort: 0.0400s
Merge Sort: 0.0695s
Merge Sort: 0.0119s
N (R101206): 9.99, VIXUTU (R102748): 9.99

Key output (from running * task1.py):
• Sample sorting results (small input) — both sorts agree.
• Benchmark timings (example):
  ○ Dataset 1000 — Quick Sort: ~0.0020s, Merge Sort: ~0.0015s
  ○ Dataset 5000 — Quick Sort: ~0.0101s, Merge Sort: ~0.0119s
  ○ Dataset 20000 — Quick Sort: ~0.0400s, Merge Sort: ~0.0695s
• Example Top-5 students printed for each dataset size.

Next steps (pick one)
• I can increase dataset sizes (e.g., 50k, 100k) to see broader performance trends.
• Add timing statistics (multiple runs, mean/stddev) for more reliable comparison.
• Make QuickSort stable (if stability is required) or implement in-place versions to reduce memory.

Would you like me to run larger benchmarks, add repeated trials, or commit these changes?
GPT-5 mini - 1x
> Implement sorting functions (1/3)
task1.py +
Describe what to build next
Agent Agent Auto 3.13.7 (.venv) Go Live 10:10 PM 28-11-2025
```

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.

Input:-

The screenshot shows the VS Code interface with the following details:

- File Structure:** The Explorer sidebar shows a folder named "AI ALL LAB ASSIGNMENTS" containing ".venv", "library_sample.json", "task1.py", and "task2.py".
- Code Editor:** The main editor window displays Python code for "task2.py". It includes imports from `future`, `json`, `random`, `string`, and `typing`. It defines a `@dataclass` named `Paper` with attributes `title` and `author`. A method `to_dict` returns a `Dict[str, str]` with keys "title" and "author". Another method `generate_random_title` generates a title by joining random lowercase letters. A third method `generate_random_author` generates an author name by joining random uppercase letters.
- Terminal:** The terminal at the bottom shows the command `PS C:\Users\SIDDHARTH\AI All Lab Assignments> & "C:/Users/SIDDHARTH/AI All Lab Assignments/.venv/Scripts/python.exe" "c:/Users/SIDDHARTH/AI All Lab Assignments/task2.py"`.
- Chat Panel:** The right-hand Chat panel contains a list of search-related notes and next steps, such as "Search timings (example keywords like 'Data', 'Smith', 'Quantum'):" and "Notes and next steps".
- Bottom Bar:** The status bar shows "Ln 157, Col 1 Spaces: 4 UTF-8 CRLF () Python ENG IN 1016 PM 28-11-2023".

Output:-

The screenshot shows the VS Code interface with the following details:

- File Structure:** The Explorer sidebar shows a folder named "AI ALL LAB ASSIGNMENTS" containing ".venv", "library_sample.json", "task1.py", and "task2.py".
- Code Editor:** The main editor window displays the output of the Python script "task2.py". The output shows results for various keywords: "Quantum", "Analysis", "Data", "Smith", "Quantum", and "Analysis". For each keyword, it compares search times for "Linear", "Hash token", and "Binary-prefix" methods.
- Terminal:** The terminal at the bottom shows the command `PS C:\Users\SIDDHARTH\AI All Lab Assignments> & "C:/Users/SIDDHARTH/AI All Lab Assignments/.venv/Scripts/python.exe" "c:/Users/SIDDHARTH/AI All Lab Assignments/task2.py"`.
- Chat Panel:** The right-hand Chat panel contains a list of search-related notes and next steps, such as "Search timings (example keywords like 'Data', 'Smith', 'Quantum'):" and "Notes and next steps".
- Bottom Bar:** The status bar shows "Ln 157, Col 1 Spaces: 4 UTF-8 CRLF () Python ENG IN 1017 PM 28-11-2023".

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:
 - Start with a **Greedy approach**.
 - 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.

Input:-

The screenshot shows a code editor interface with several files open in the Explorer panel:

- task1.py
- task2.py
- task3.py (selected)
- task4.py
- task3_output
- library_sample.json
- .venv

The task3.py file contains the following Python code:

```
from __future__ import annotations
import math
import random
import os
from dataclasses import dataclass
from typing import List, Tuple

@dataclass
class Sensor:
    x: float
    y: float

def generate_sensors(n: int, bounds: Tuple[float, float] = (0.0, 100.0), seed: int | None = None):
    if seed is not None:
        random.seed(seed)
    sensors: List[Sensor] = []
    lo, hi = bounds
    for _ in range(n):
        sensors.append(Sensor(random.uniform(lo, hi), random.uniform(lo, hi)))
    return sensors

def euclidean(a: Sensor, b: Sensor) -> float:
    return math.hypot(a.x - b.x, a.y - b.y)

def route_distance(route: List[int], sensors: List[Sensor]) -> float:
    # route is a list of indices into sensors; closed tour
    if not route:
        return 0.0
    dist = 0.0
    for i in range(len(route)):
        a = sensors[route[i]]
        b = sensors[route[(i + 1) % len(route)]]
        dist += euclidean(a, b)
```

The Chat panel on the right shows GitHub Copilot's suggestions and tasks:

- 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.
- Defining search functionalities...
- I'll add Task 2 plan to the todo list (update overall TODOs for this workspace).
- Completed: Implement sorting functions (1/7)
- Now I'll create task2.py implementing dataset generation, linear/binary/hash searches, and benchmarking.
- I'll add task2.py with search implementations and benchmarking (including saving a sample JSON). Now applying the patch.
- I'll run task2.py to generate the sample dataset and show benchmark results. I'll call PowerShell with the & operator to handle spaces in paths.
- > Implement AUV route optimization (1/7)
- task3.py
- Describe what to build next
- Agent Auto

Output:-

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `.venv`, `task3_output`, `library_samplejson`, `task1.py`, `task2.py`, `task3.py`, and `task4.py`.
- Terminal:** Displays command-line output for running `task3.py` and `task4.py`, showing route optimization results.
- Chat Panel:** Contains a list of TODO items and notes about implementing search functionalities and sorting functions.
- Bottom Status Bar:** Shows file path (`C:\Users\SIDDHARTHA\AI All Lab Assignments`), line count (Ln 1, Col 1), spaces (Spaces: 4), encoding (UTF-8), and other system information.

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.

Input:-

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files in the "AI ALL LAB ASSIGNMENTS" folder: .venv, task3_output, library.sample.json, task1.py, task2.py, task3.py, and task4.py.
- Code Editor:** Displays the content of task4.py. The code defines a Stock class, implements __repr__ and percent_change methods, generates stock symbols, and creates a list of stocks.
- Performance Insights Panel:** Located on the right, it provides a "CHART" view and a table of "Operation" vs "Time". It includes a section titled "Trade-offs & Analysis" with bullet points about sorting, searching, and when to use specific data structures.
- Terminal:** Shows the command PS C:\Users\SIDDHARTH\AI All Lab Assignments & "C:/Users/SIDDHARTH/AI All Lab Assignments/.venv/Scripts/python.exe" task4.py and the output of the script's performance benchmarks.
- Status Bar:** Shows the current file is task4.py, Python language, and the date/time 28-11-2025.

Output:-

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files in the "AI ALL LAB ASSIGNMENTS" folder: .venv, task3_output, library.sample.json, task1.py, task2.py, task3.py, and task4.py.
- Terminal:** Shows the command PS C:\Users\SIDDHARTH\AI All Lab Assignments & "C:/Users/SIDDHARTH/AI All Lab Assignments/.venv/Scripts/python.exe" task4.py and the output of the script's performance benchmarks for different query sizes (n=100, n=1000, n=10000, n=50000).
- Performance Insights Panel:** Located on the right, it provides a "CHART" view and a table of "Operation" vs "Time". It includes a section titled "Trade-offs & Analysis" with bullet points about sorting, searching, and when to use specific data structures.
- Status Bar:** Shows the current file is task4.py, Python language, and the date/time 28-11-2025.