Assignment12.5

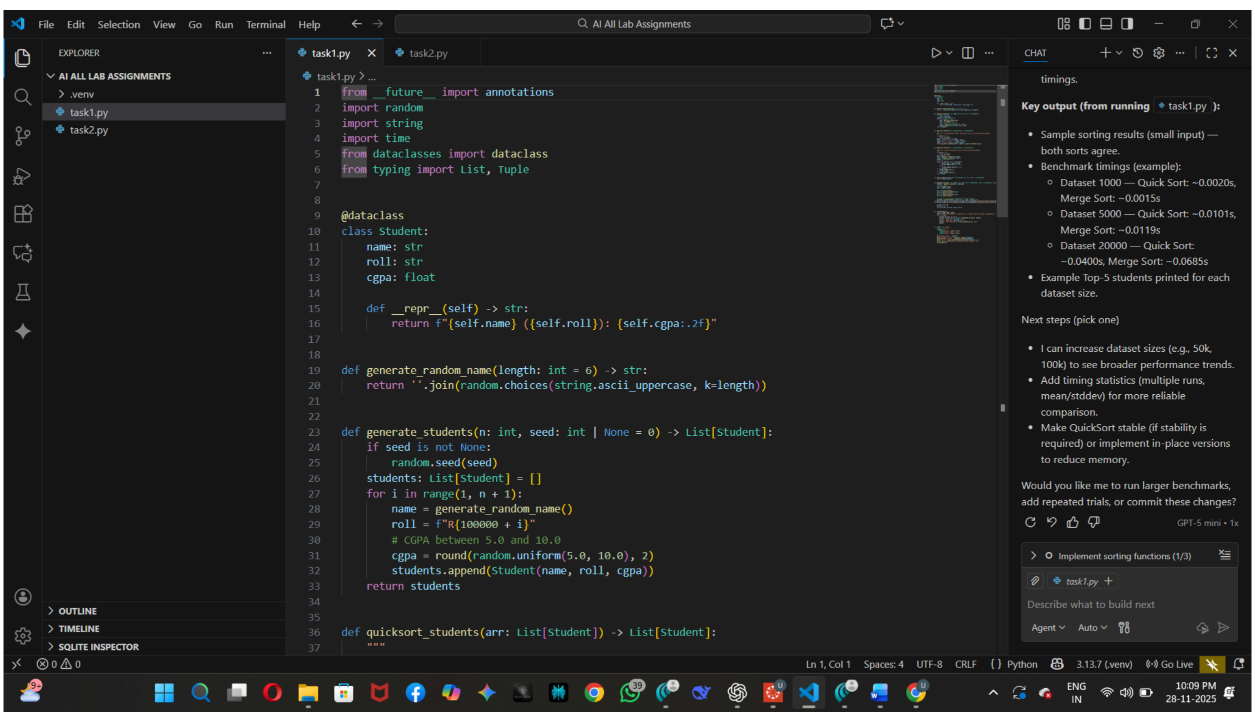
Name:- U.Siddhartha
Hall No:-2403A54122

Task 1: Sorting Student Records for Placement Drive

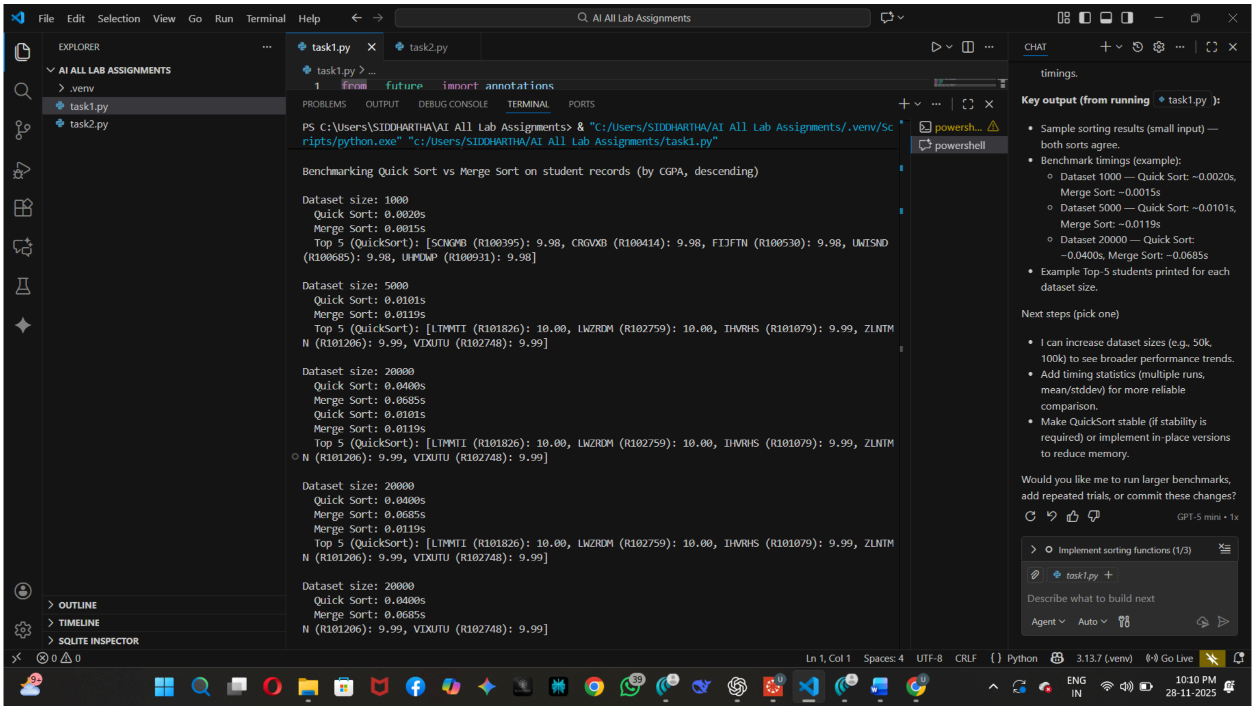
Scenario:
SR University is preparing for a campus placement drive. The Training and Placement Cell needs student records sorted by **CGPA** in descending order to easily shortlist candidates.

- 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.



Output:-

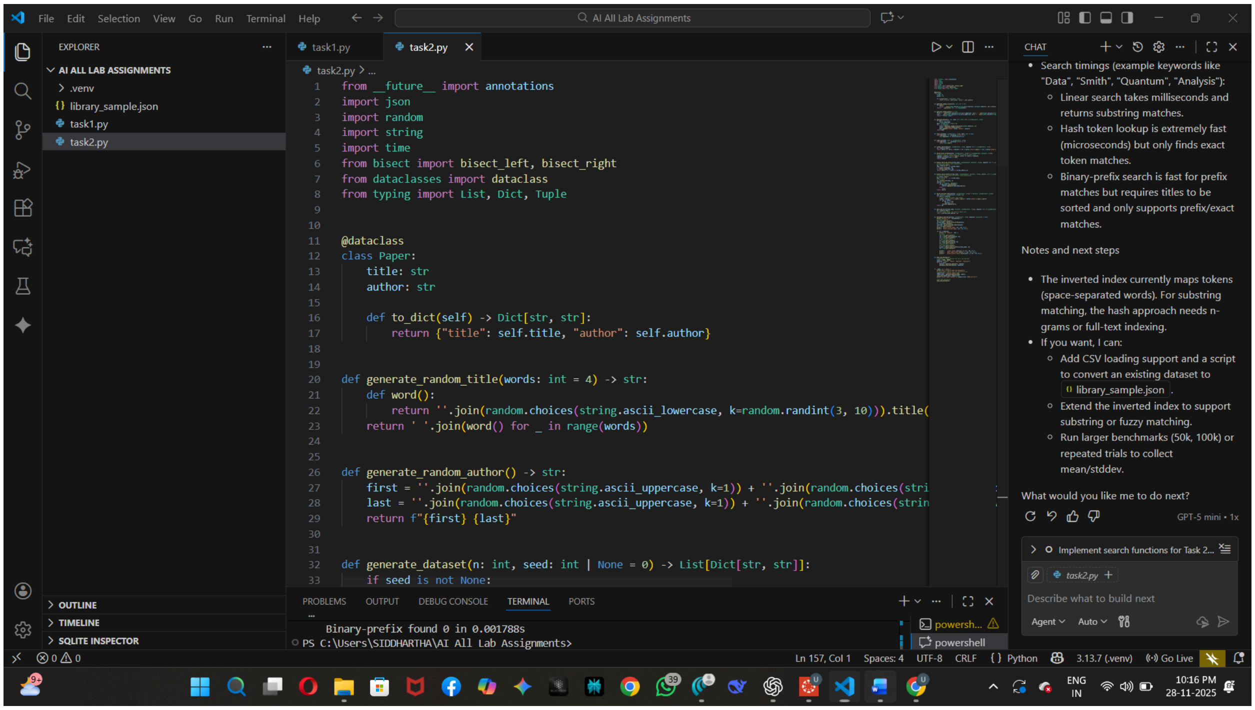


Task 2: Optimized Search in Online Library System

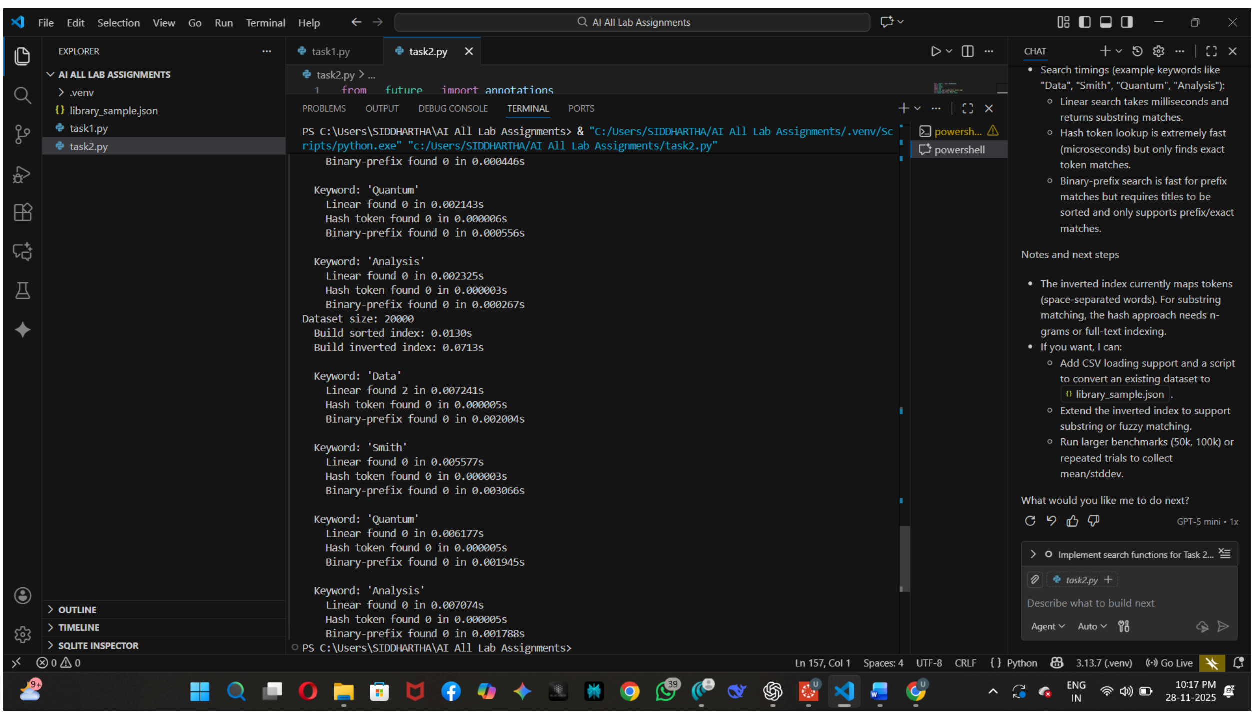
Scenario:
SR University's digital library has thousands of research papers. Students frequently search for a paper by **title or author name**. The current linear search is too slow.

- 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:-


File Edit Selection View Go Run Terminal Help
AI All Lab Assignments
EXPLORER
AI ALL LAB ASSIGNMENTS
.venv
library_sample.json
task1.py
task2.py
task2.py > ...
from __future__ import annotations
import json
import random
import string
import time
from bisect import bisect_left, bisect_right
from dataclasses import dataclass
from typing import List, Dict, Tuple
@dataclass
class Paper:
title: str
author: str
def to_dict(self) -> Dict[str, str]:
return {"title": self.title, "author": self.author}
def generate_random_title(words: int = 4) -> str:
def word():
return ''.join(random.choices(string.ascii_lowercase, k=random.randint(3, 10))).title(
return ' '.join(word() for _ in range(words))
def generate_random_author() -> str:
first = ''.join(random.choices(string.ascii_uppercase, k=1)) + ''.join(random.choices(stri
last = ''.join(random.choices(string.ascii_uppercase, k=1)) + ''.join(random.choices(strin
return f"{first} {last}"
def generate_dataset(n: int, seed: int | None = 0) -> List[Dict[str, str]]:
if seed is not None:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Binary-prefix found 0 in 0.001788s
PS C:\Users\SIDDHARTHA\AI All Lab Assignments>
CHAT
Search timings (example keywords like "Data", "Smith", "Quantum", "Analysis"):
Linear search takes milliseconds and returns substring matches.
Hash token lookup is extremely fast (microseconds) but only finds exact token matches.
Binary-prefix search is fast for prefix matches but requires titles to be sorted and only supports prefix/exact matches.
Notes and next steps
The inverted index currently maps tokens (space-separated words). For substring matching, the hash approach needs n-grams or full-text indexing.
If you want, I can:
Add CSV loading support and a script to convert an existing dataset to library_sample.json.
Extend the inverted index to support substring or fuzzy matching.
Run larger benchmarks (50k, 100k) or repeated trials to collect mean/stddev.
What would you like me to do next?
GPT-5 mini • 1x
Implement search functions for Task 2...
Describe what to build next
Ln 157, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.7 (.venv) Go Live
10:16 PM
28-11-2025

Output:-

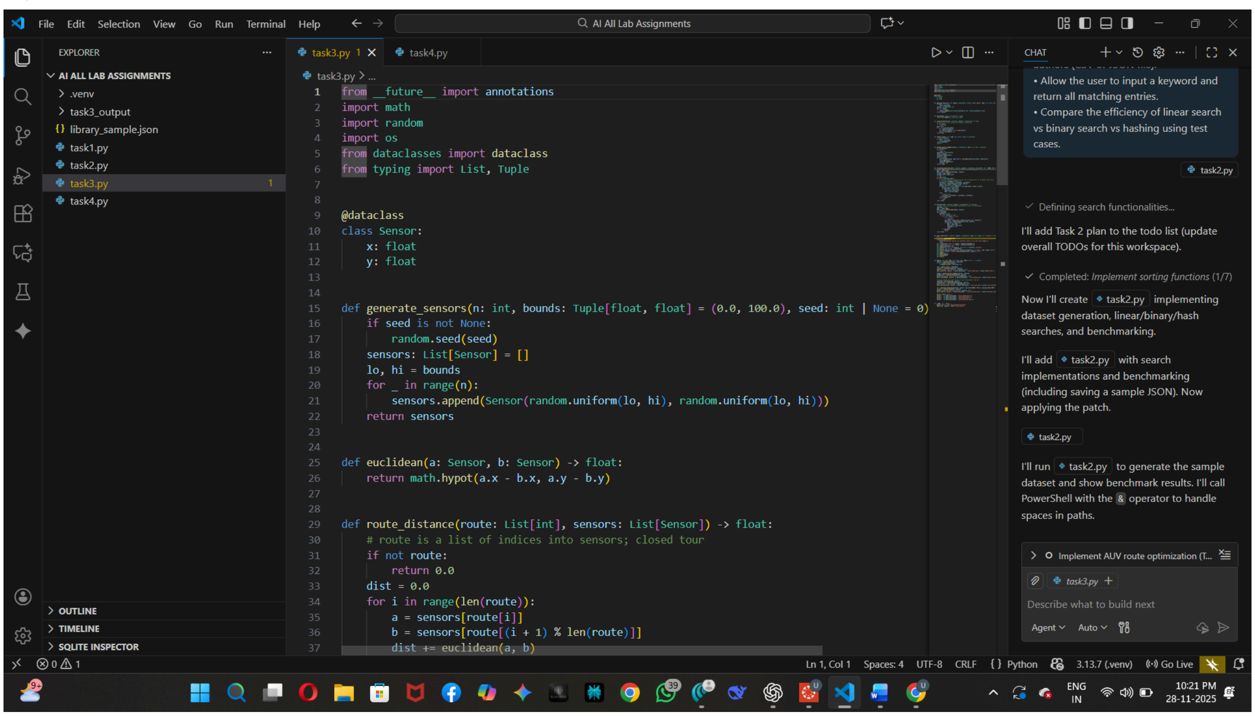

PS C:\Users\SIDDHARTHA\AI All Lab Assignments> & "C:/Users/SIDDHARTHA/AI All Lab Assignments/.venv/Scripts/python.exe" "c:/Users/SIDDHARTHA/AI All Lab Assignments/task2.py"
Binary-prefix found 0 in 0.000446s
Keyword: 'Quantum'
Linear found 0 in 0.002143s
Hash token found 0 in 0.000006s
Binary-prefix found 0 in 0.000556s
Keyword: 'Analysis'
Linear found 0 in 0.002325s
Hash token found 0 in 0.000003s
Binary-prefix found 0 in 0.000267s
Dataset size: 20000
Build sorted index: 0.0130s
Build inverted index: 0.0713s
Keyword: 'Data'
Linear found 2 in 0.007241s
Hash token found 0 in 0.000005s
Binary-prefix found 0 in 0.002004s
Keyword: 'Smith'
Linear found 0 in 0.005577s
Hash token found 0 in 0.000003s
Binary-prefix found 0 in 0.003066s
Keyword: 'Quantum'
Linear found 0 in 0.006177s
Hash token found 0 in 0.000005s
Binary-prefix found 0 in 0.001945s
Keyword: 'Analysis'
Linear found 0 in 0.007074s
Hash token found 0 in 0.000005s
Binary-prefix found 0 in 0.001788s
PS C:\Users\SIDDHARTHA\AI All Lab Assignments>
10:17 PM
28-11-2025

Task 3: Route Optimization for AUV Swarm

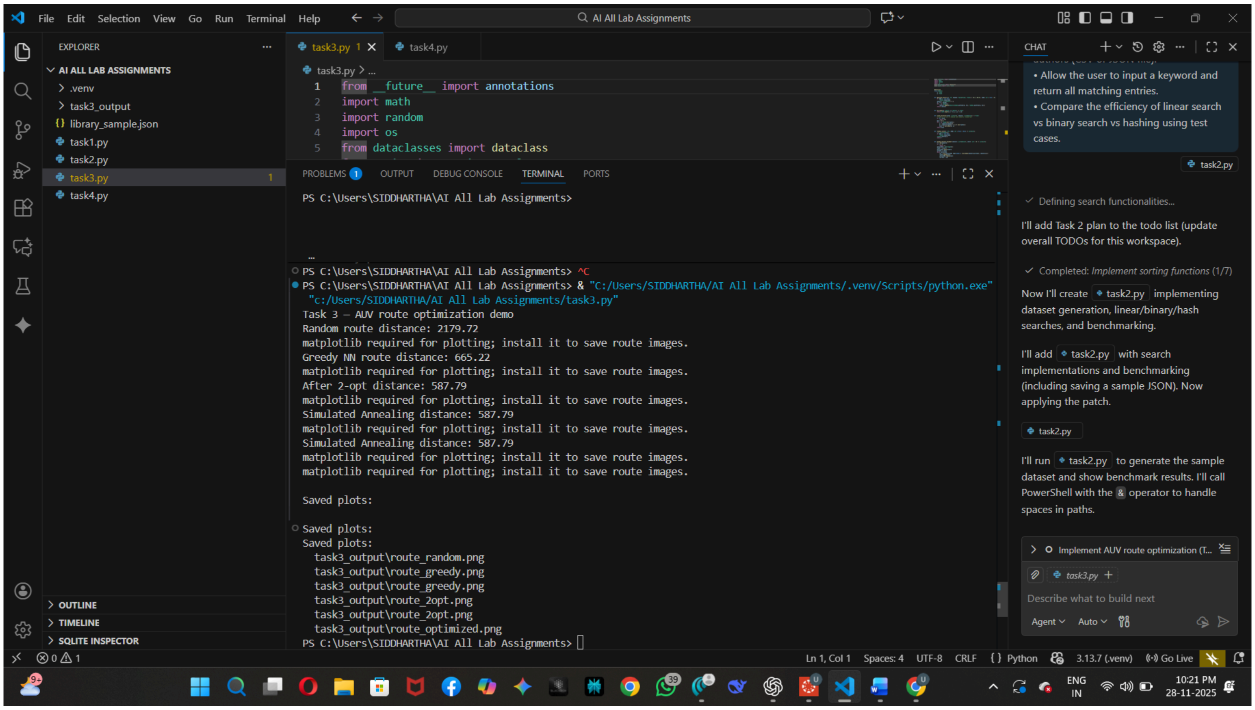
Scenario:
A research team at SR University is simulating **Autonomous Underwater Vehicle (AUV) swarms**. Each AUV must visit multiple underwater sensors, and the goal is to minimize travel distance (like the **Traveling Salesman Problem**).

- 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.

Input:-



Output:-



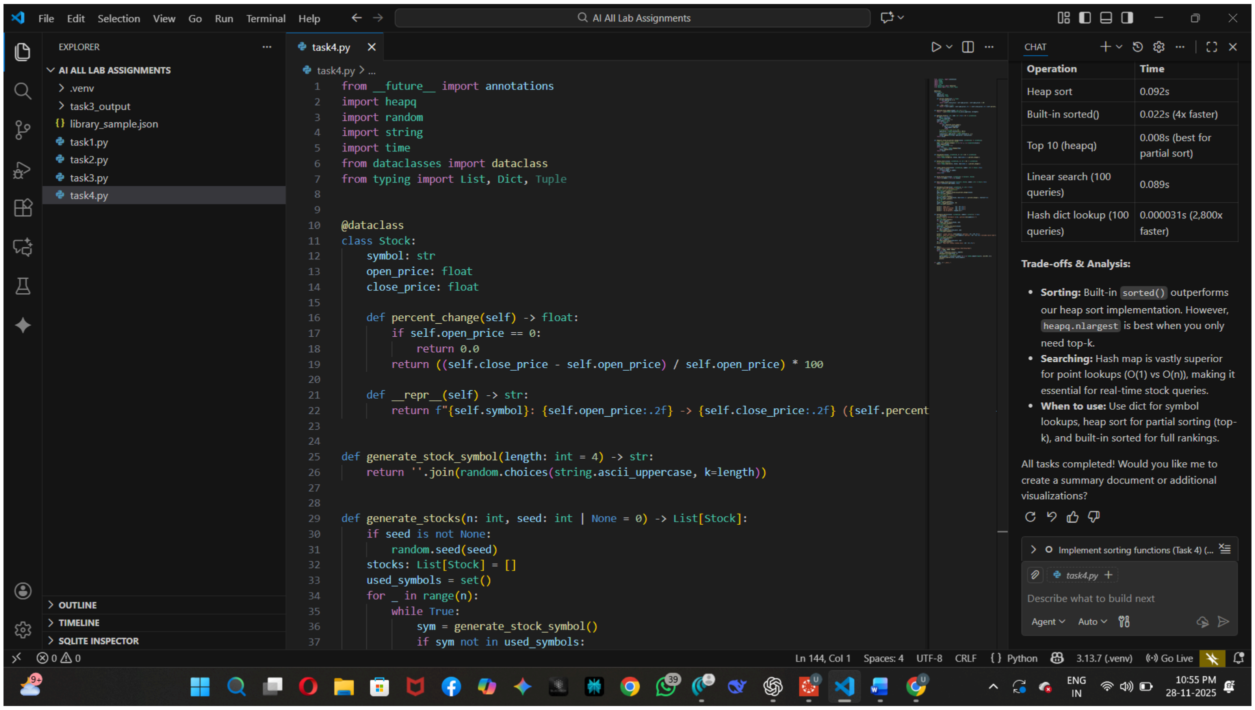
Task 4: Real-Time Stock Data Sorting & Searching

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:-



Output:-