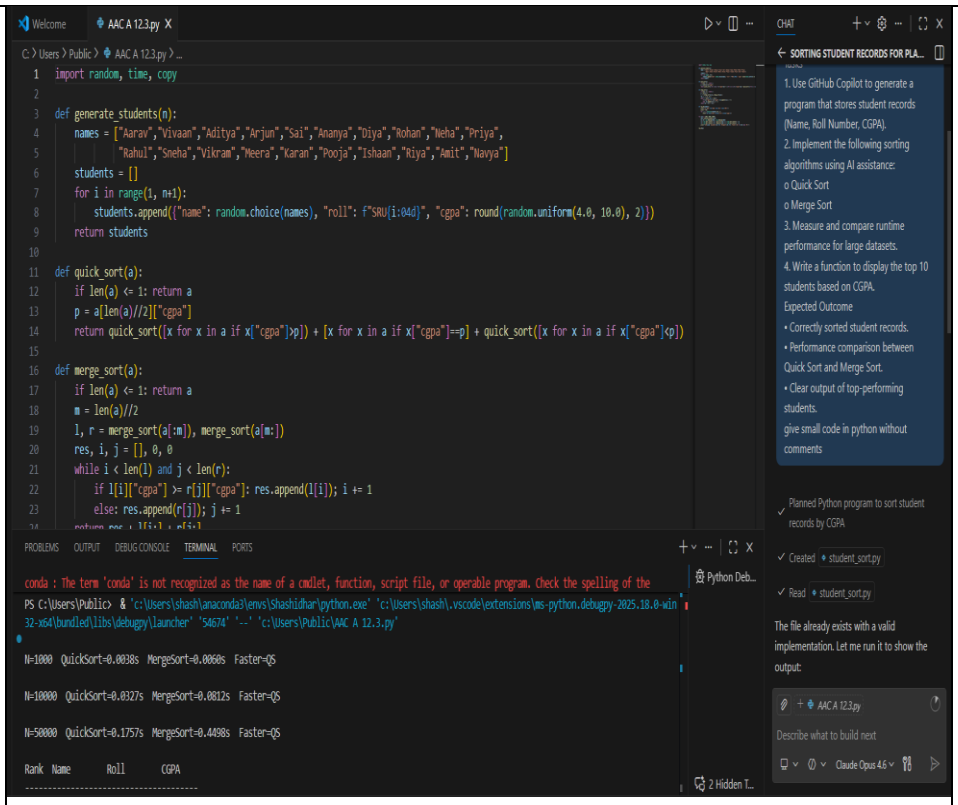
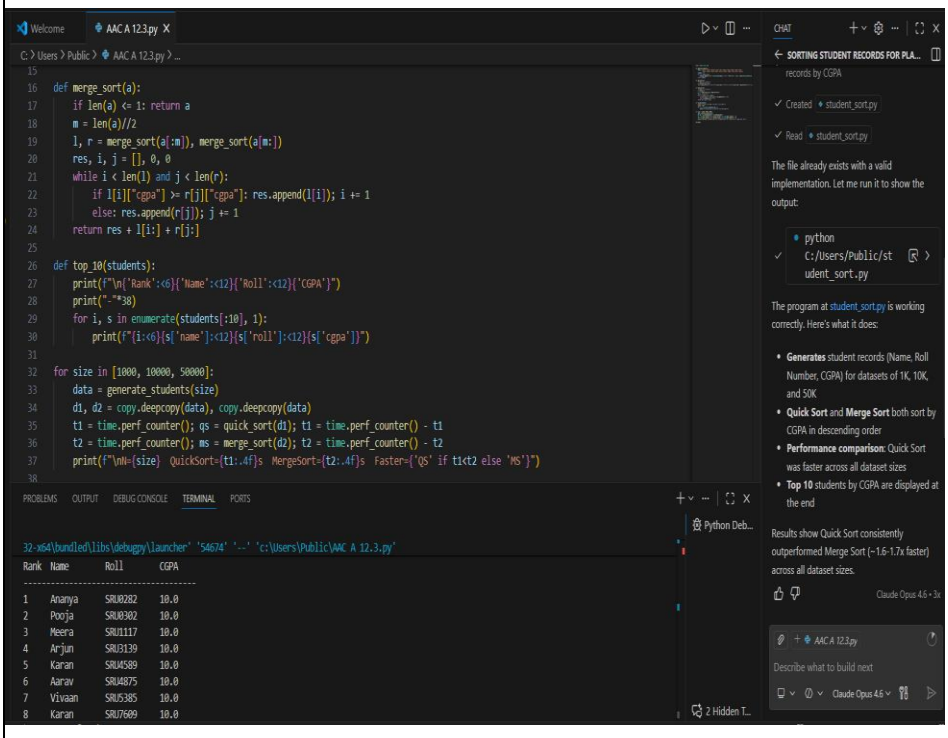
Name:I. Abhinay H.No:2303A51811 Batch:26

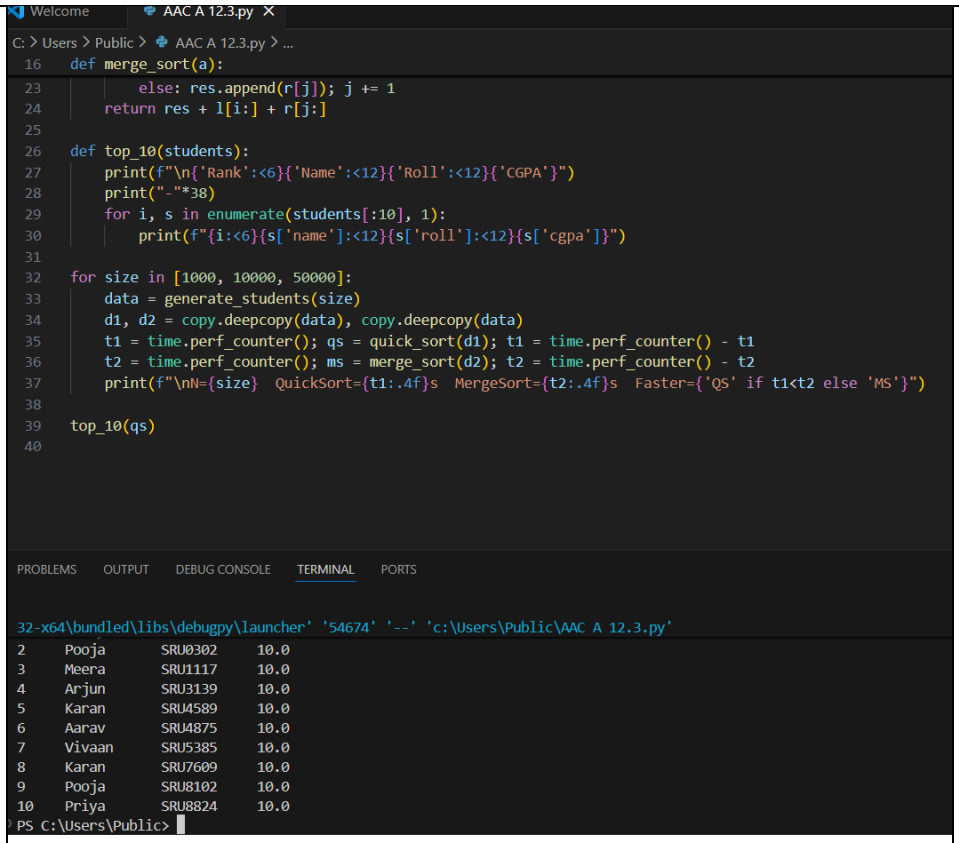
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week6 – Wednusday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:12.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 12: Algorithms with AI Assistance Sorting, Searching, and Algorithm Optimization Using AI Tools** **Lab Objectives**	Week3 - Wednesday

	The objectives of this laboratory exercise are to: • Apply AI-assisted programming techniques to implement sorting and searching algorithms. • Analyze and compare algorithm efficiency using time and space complexity. • Understand how AI tools can suggest optimizations and alternative algorithmic approaches. • Strengthen problem-solving skills through real-world, data-driven scenarios.	
	Learning Outcomes After completing this lab, students will be able to: • Implement and optimize classic algorithms using AI-assisted coding tools. • Compare multiple algorithms for the same problem and justify their selection. • Measure and analyze runtime performance using experimental data. • Critically review and refine AI-generated algorithmic solutions.	
	Task 1: Sorting Student Records for Placement Drive **Scenario** SR University's Training and Placement Cell needs to shortlist candidates efficiently during campus placements. Student records must be sorted by **CGPA in descending order**. **Tasks** 1. Use **GitHub Copilot** to generate a program that stores student records (Name, Roll Number, CGPA). 2. Implement the following sorting algorithms using AI assistance: o Quick Sort o Merge Sort 3. Measure and compare runtime performance for large datasets. 4. Write a function to display the **top 10 students** based on CGPA. **Expected Outcome** • Correctly sorted student records. • Performance comparison between Quick Sort and Merge Sort. • Clear output of top-performing students.	


Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
import random, time, copy
def generate_students(n):
names = ["Aarav","Vivaan","Aditya","Arjun","Sai","Ananya","Diya","Rohan","Neha","Priya",
"Rahul","Sneha","Vikram","Meera","Karan","Pooja","Ishaan","Riya","Amit","Navya"]
students = []
for i in range(1, n+1):
students.append({"name": random.choice(names), "roll": f"SRU{i:04d}", "cgpa": round(random.uniform(4.0, 10.0), 2)})
return students
def quick_sort(a):
if len(a) <= 1: return a
p = a[len(a)//2]["cgpa"]
return quick_sort([x for x in a if x["cgpa"]>p]) + [x for x in a if x["cgpa"]==p] + quick_sort([x for x in a if x["cgpa"]<p])
def merge_sort(a):
if len(a) <= 1: return a
m = len(a)//2
l, r = merge_sort(a[:m]), merge_sort(a[m:])
res, i, j = [], 0, 0
while i < len(l) and j < len(r):
if l[i]["cgpa"] >= r[j]["cgpa"]: res.append(l[i]); i += 1
else: res.append(r[j]); j += 1
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Deb...
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the
PS C:\Users\Public> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win
32-x64\bundled\libs\debugpy\launcher' '54674' '--' 'c:\Users\Public\AAC A 12.3.py'
N=1000 QuickSort=0.0038s MergeSort=0.0060s Faster=QS
N=10000 QuickSort=0.0327s MergeSort=0.0812s Faster=QS
N=50000 QuickSort=0.1757s MergeSort=0.4498s Faster=QS
Rank Name Roll CGPA
2 Hidden T...
CHAT
SORTING STUDENT RECORDS FOR PLA...
1. Use GitHub Copilot to generate a program that stores student records (Name, Roll Number, CGPA).
2. Implement the following sorting algorithms using AI assistance:
o Quick Sort
o Merge Sort
3. Measure and compare runtime performance for large datasets.
4. Write a function to display the top 10 students based on CGPA.
Expected Outcome
• Correctly sorted student records.
• Performance comparison between Quick Sort and Merge Sort.
• Clear output of top-performing students.
give small code in python without comments
Planned Python program to sort student records by CGPA
Created student_sort.py
Read student_sort.py
The file already exists with a valid implementation. Let me run it to show the output:
AAC A 12.3.py
Describe what to build next
Claude Opus 4.6

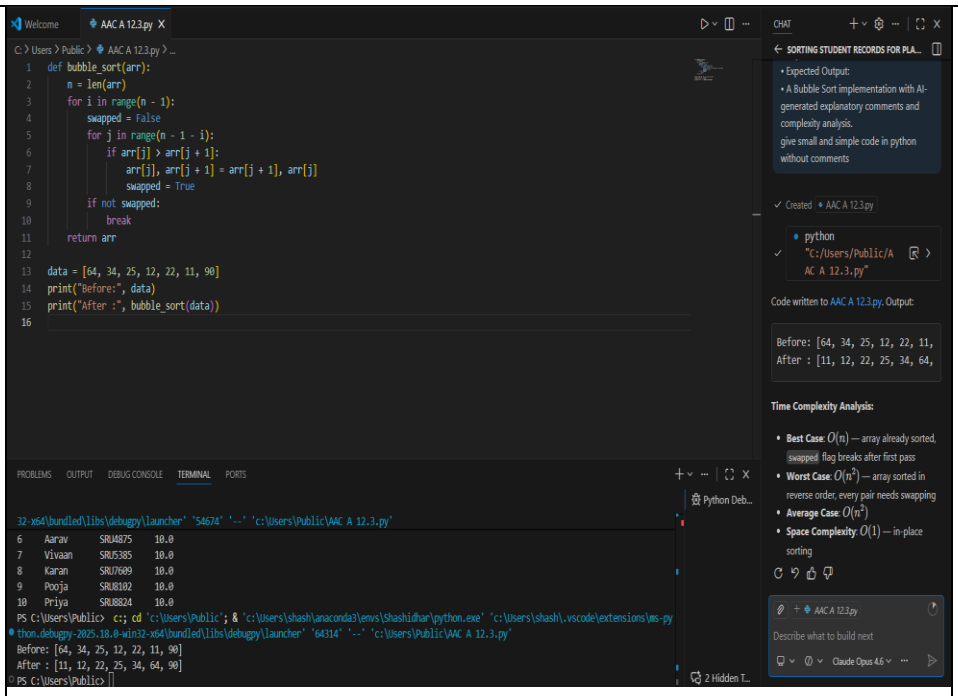

Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
def merge_sort(a):
if len(a) <= 1: return a
m = len(a)//2
l, r = merge_sort(a[:m]), merge_sort(a[m:])
res, i, j = [], 0, 0
while i < len(l) and j < len(r):
if l[i]["cgpa"] >= r[j]["cgpa"]: res.append(l[i]); i += 1
else: res.append(r[j]); j += 1
return res + l[i:] + r[j:]
def top_10(students):
print(f"\n{'Rank':<6}{'Name':<12}{'Roll':<12}{'CGPA'}")
print("-"*38)
for i, s in enumerate(students[:10], 1):
print(f"{i:<6}{s['name']:<12}{s['roll']:<12}{s['cgpa']}")
for size in [1000, 10000, 50000]:
data = generate_students(size)
d1, d2 = copy.deepcopy(data), copy.deepcopy(data)
t1 = time.perf_counter(); qs = quick_sort(d1); t1 = time.perf_counter() - t1
t2 = time.perf_counter(); ms = merge_sort(d2); t2 = time.perf_counter() - t2
print(f"\nN={size} QuickSort={t1:.4f}s MergeSort={t2:.4f}s Faster={'QS' if t1<t2 else 'MS'}")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Deb...
32-x64\bundled\libs\debugpy\launcher' '54674' '--' 'c:\Users\Public\AAC A 12.3.py'
Rank Name Roll CGPA
1 Ananya SRU0282 10.0
2 Pooja SRU0302 10.0
3 Meera SRU1117 10.0
4 Arjun SRU3139 10.0
5 Karan SRU4589 10.0
6 Aarav SRU4875 10.0
7 Vivaan SRU5385 10.0
8 Karan SRU7609 10.0
2 Hidden T...
CHAT
SORTING STUDENT RECORDS FOR PLA...
records by CGPA
Created student_sort.py
Read student_sort.py
The file already exists with a valid implementation. Let me run it to show the output:
python C:/Users/Public/student_sort.py
The program at student_sort.py is working correctly. Here's what it does:
• Generates student records (Name, Roll Number, CGPA) for datasets of 1K, 10K, and 50K
• Quick Sort and Merge Sort both sort by CGPA in descending order
• Performance comparison: Quick Sort was faster across all dataset sizes
• Top 10 students by CGPA are displayed at the end
Results show Quick Sort consistently outperformed Merge Sort (~1.6-1.7x faster) across all dataset sizes.
Claude Opus 4.6 • 3x
AAC A 12.3.py
Describe what to build next
Claude Opus 4.6



```
Welcome  AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
16    def merge_sort(a):
23            else: res.append(r[j]); j += 1
24        return res + l[i:] + r[j:]
25
26    def top_10(students):
27        print(f"\n{'Rank':<6}{'Name':<12}{'Roll':<12}{'CGPA'}")
28        print("-"*38)
29        for i, s in enumerate(students[:10], 1):
30            print(f"{i:<6}{s['name']:<12}{s['roll']:<12}{s['cgpa']}")
31
32    for size in [1000, 10000, 50000]:
33        data = generate_students(size)
34        d1, d2 = copy.deepcopy(data), copy.deepcopy(data)
35        t1 = time.perf_counter(); qs = quick_sort(d1); t1 = time.perf_counter() - t1
36        t2 = time.perf_counter(); ms = merge_sort(d2); t2 = time.perf_counter() - t2
37        print(f"\nN={size}  QuickSort={t1:.4f}s  MergeSort={t2:.4f}s  Faster={'QS' if t1<t2 else 'MS'}")
38
39    top_10(qs)
40

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

32-x64\bundled\libs\debugpy\launcher' '54674' '--' 'c:\Users\Public\AAC A 12.3.py'
2     Pooja       SRU0302     10.0
3     Meera       SRU1117     10.0
4     Arjun       SRU3139     10.0
5     Karan       SRU4589     10.0
6     Aarav       SRU4875     10.0
7     Vivaan      SRU5385     10.0
8     Karan       SRU7609     10.0
9     Pooja       SRU8102     10.0
10    Priya       SRU8824     10.0
PS C:\Users\Public>
```

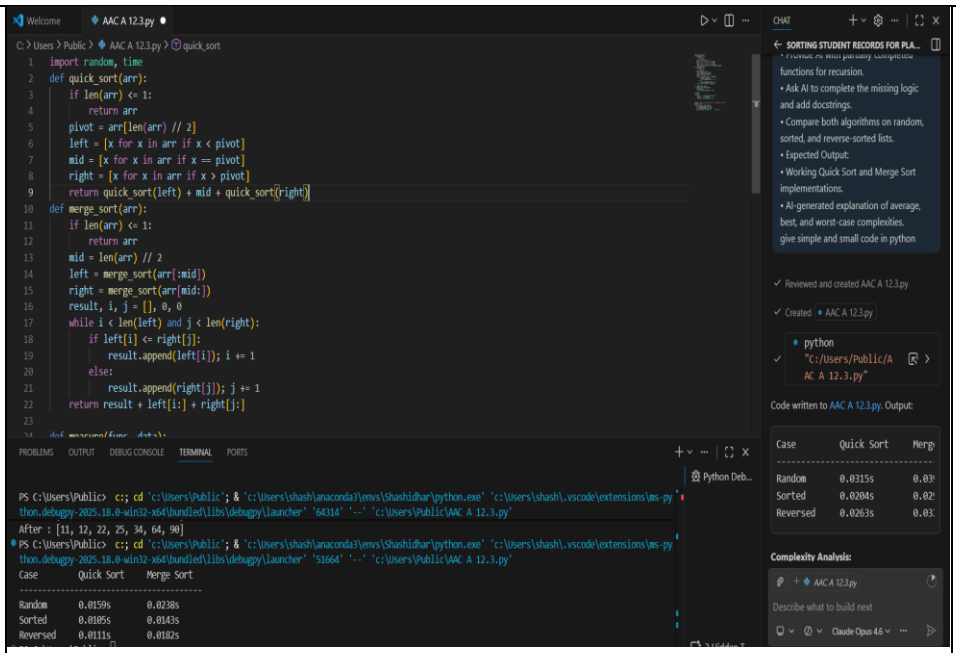
Task 2: Implementing Bubble Sort with AI Comments

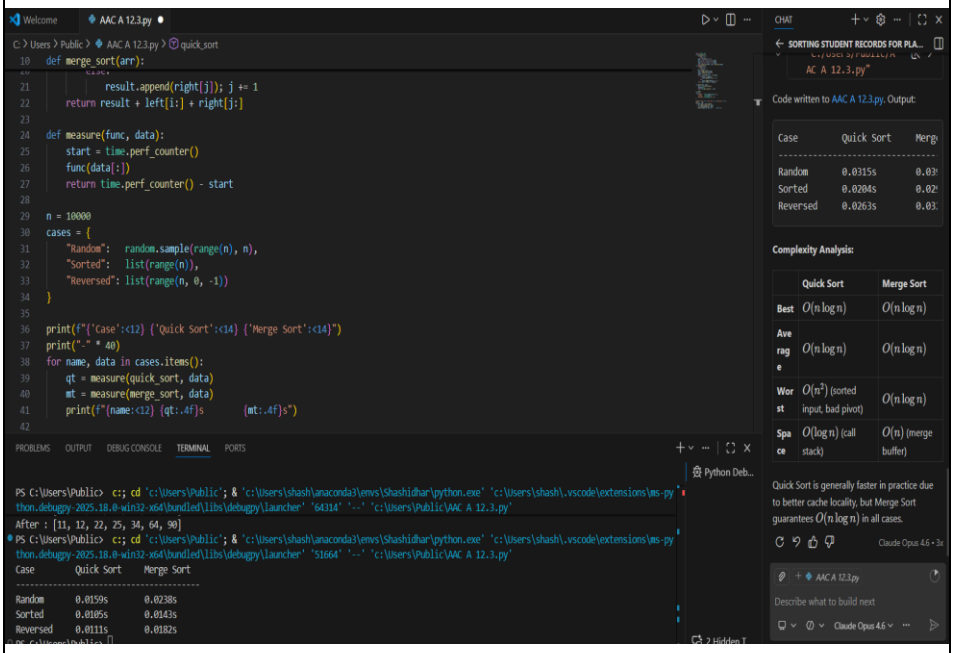
- **Task**: Write a Python implementation of **Bubble Sort**.
- **Instructions**:
  - Students implement Bubble Sort normally.
  - Ask AI to generate **inline comments explaining key logic** (like swapping, passes, and termination).
  - Request AI to provide **time complexity analysis**.
- **Expected Output**:
  - A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis.



Task 3: Quick Sort and Merge Sort Comparison

- **Task:** Implement Quick Sort and Merge Sort using recursion**.**
- **Instructions:**
  - Provide AI with partially completed functions for recursion.
  - Ask AI to complete the missing logic and add docstrings.
  - Compare both algorithms on random, sorted, and reverse-sorted lists.
- **Expected Output:**
  - Working Quick Sort and Merge Sort implementations.
  - AI-generated explanation of average, best, and worst-case complexities.





Task 4 (Real-Time Application – Inventory Management System)

Scenario: A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:

1. Quickly search for a product by ID or name.
2. Sort products by price or quantity for stock analysis.

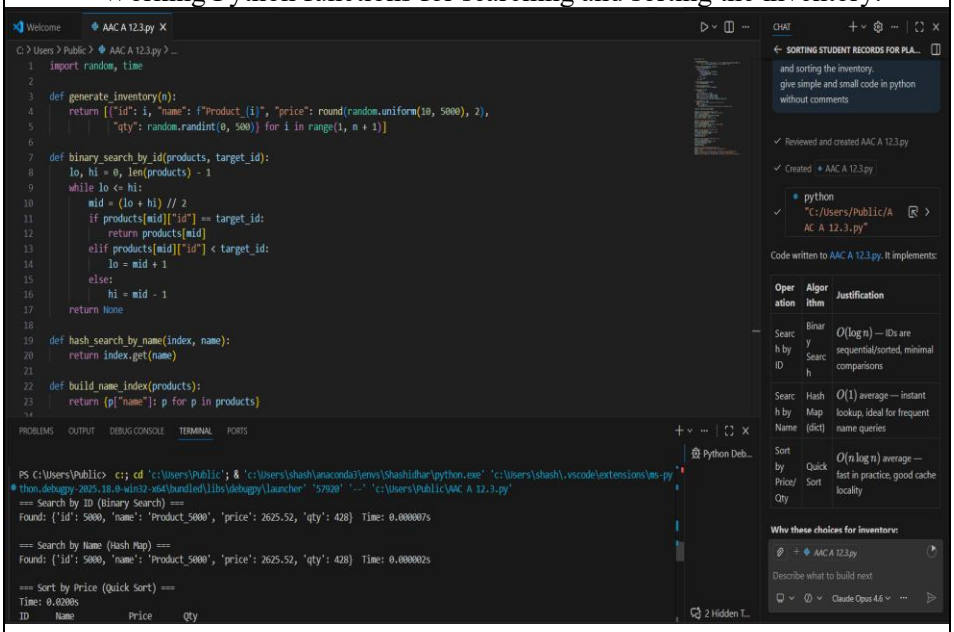
Task:

- Use AI to suggest the most efficient search and sort algorithms for this use case.

- Implement the recommended algorithms in Python.
- Justify the choice based on dataset size, update frequency, and performance requirements.

Expected Output:

- A table mapping operation → recommended algorithm → justification.
- Working Python functions for searching and sorting the inventory.



Welcome | AAC A 12.3.py

C: > Users > Public > AAC A 12.3.py > ...

```python
def build_name_index(products):
    return {p["name"]: p for p in products}

def quick_sort(arr, key):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2][key]
    left = [x for x in arr if x[key] < pivot]
    mid = [x for x in arr if x[key] == pivot]
    right = [x for x in arr if x[key] > pivot]
    return quick_sort(left, key) + mid + quick_sort(right, key)

def display(products, n=10):
    print(f"{'ID':<8}{'Name':<16}{'Price':<12}{'Qty':<6}")
    print("-" * 42)
    for p in products[:n]:
        print(f"{p['id']:<8}{p['name']:<16}{p['price']:<12}{p['qty']:<6}")

inventory = generate_inventory(10000)

print("=== Search by ID (Binary Search) ===")
sorted_by_id = sorted(inventory, key=lambda x: x["id"])
start = time.perf_counter()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public>  c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\p
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Publi
ID       Name            Price       Qty
-------------------------------------------
3623     Product_3623    10.64       206
9538     Product_9538    10.67       14
122      Product_122     11.15       333
2235     Product_2235    11.41       386
6376     Product_6376    11.61       320
5937     Product_5937    11.75       324
3876     Product_3876    11.77       356
```

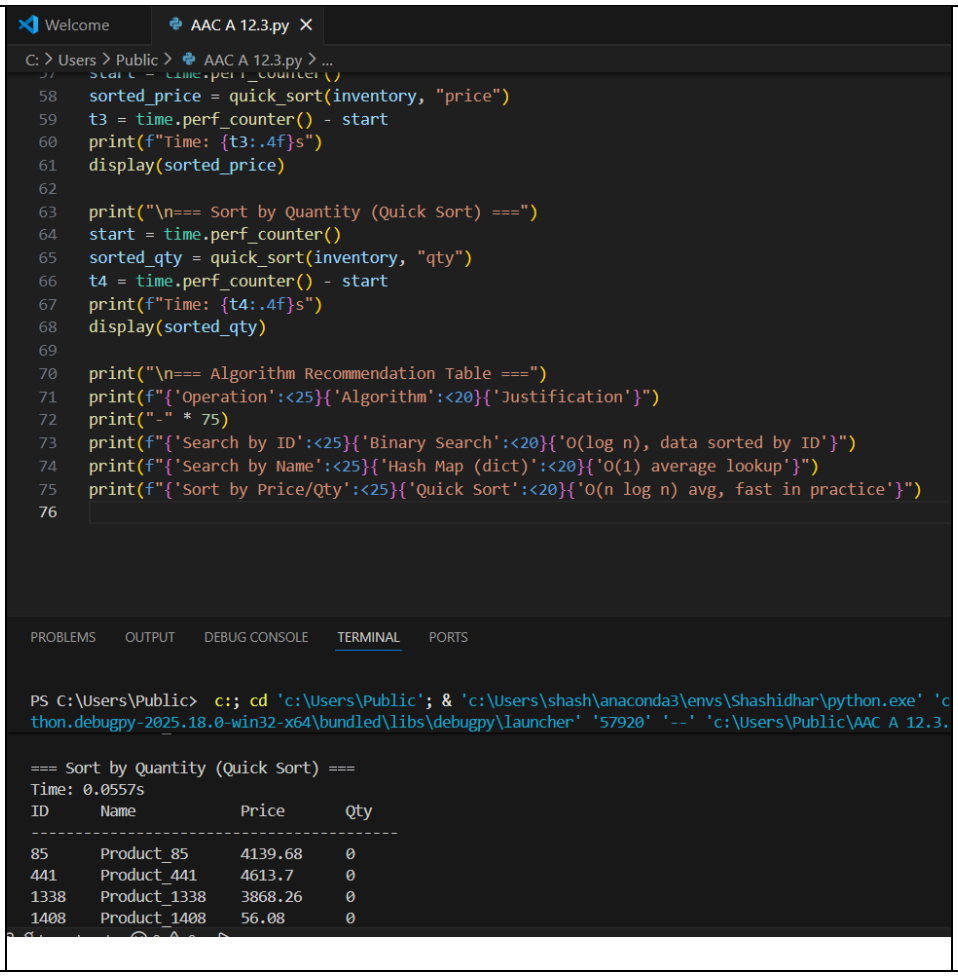
Welcome | AAC A 12.3.py

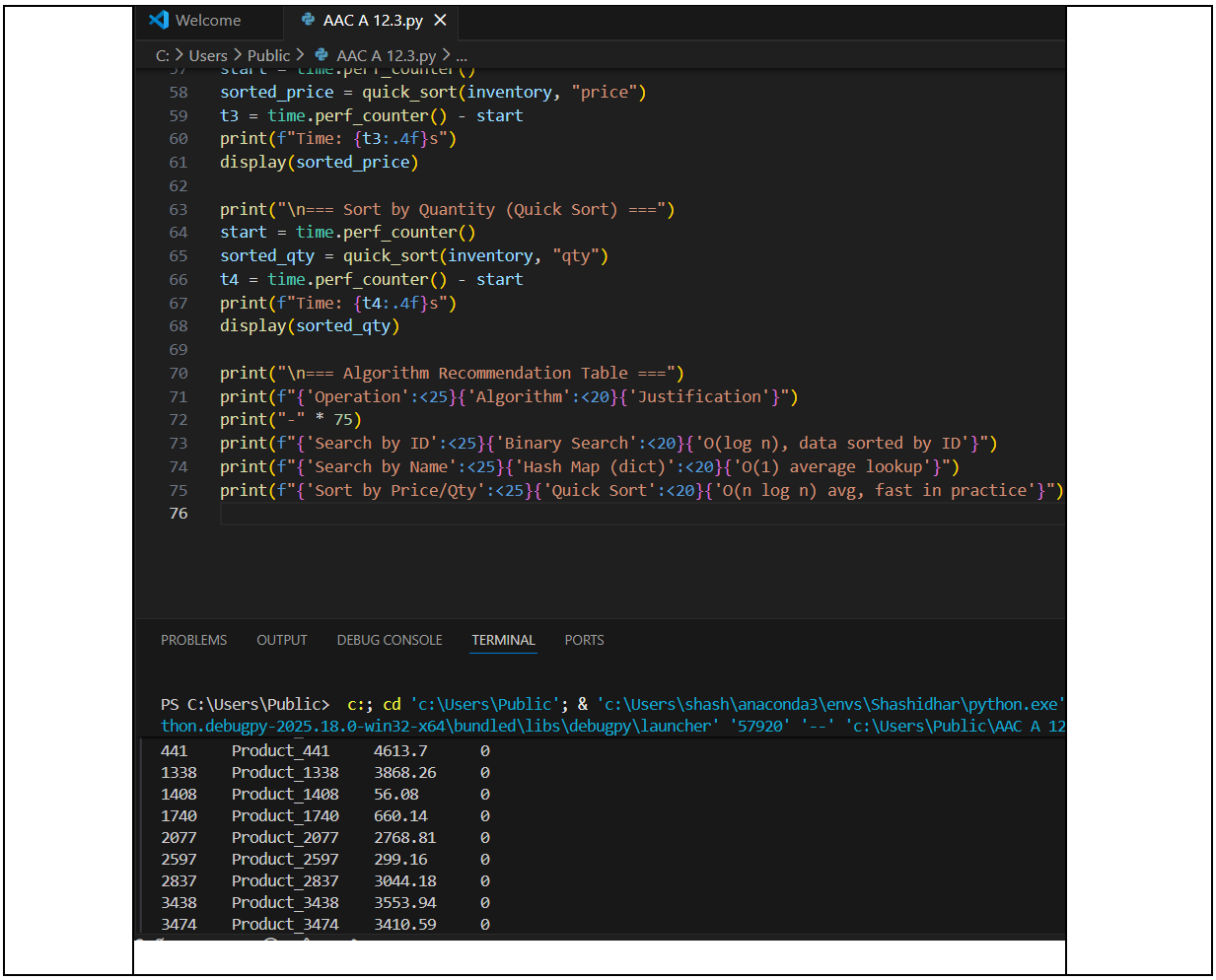
C: > Users > Public > AAC A 12.3.py > ...

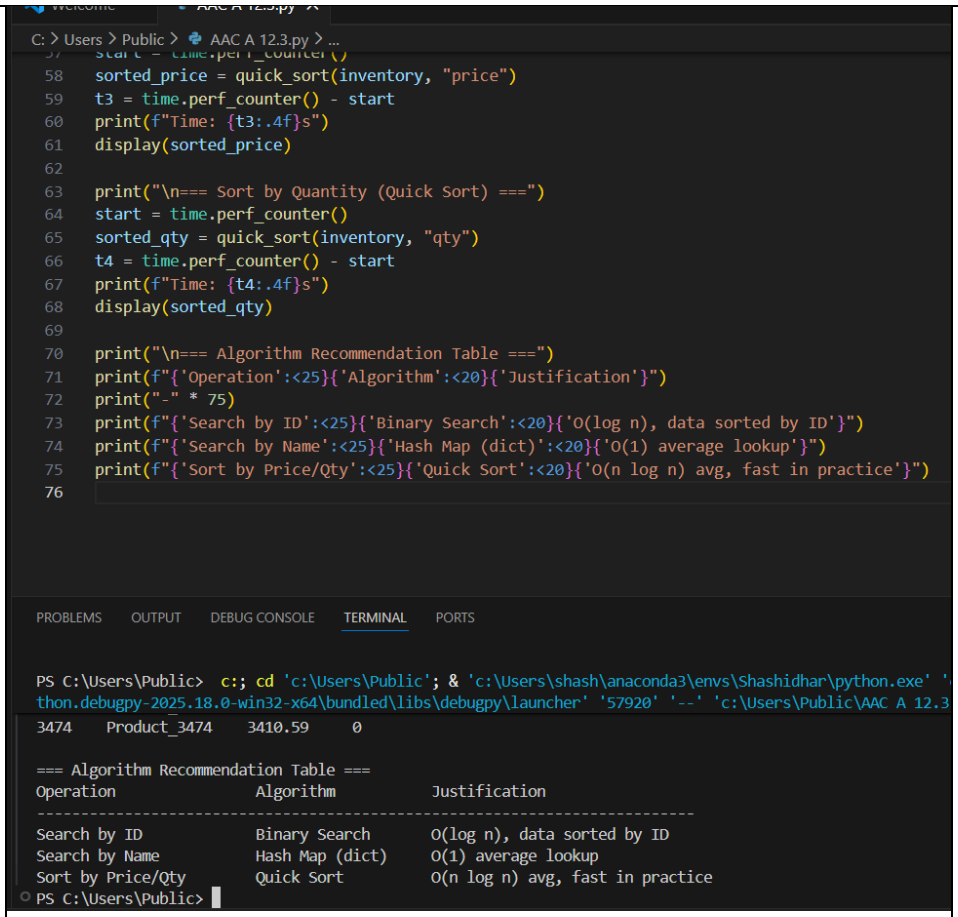
```
42  print("=== Search by ID (Binary Search) ===")
43  sorted_by_id = sorted(inventory, key=lambda x: x["id"])
44  start = time.perf_counter()
45  result = binary_search_by_id(sorted_by_id, 5000)
46  t1 = time.perf_counter() - start
47  print(f"Found: {result}  Time: {t1:.6f}s")
48
49  print("\n=== Search by Name (Hash Map) ===")
50  name_index = build_name_index(inventory)
51  start = time.perf_counter()
52  result = hash_search_by_name(name_index, "Product_5000")
53  t2 = time.perf_counter() - start
54  print(f"Found: {result}  Time: {t2:.6f}s")
55
56  print("\n=== Sort by Price (Quick Sort) ===")
57  start = time.perf_counter()
58  sorted_price = quick_sort(inventory, "price")
59  t3 = time.perf_counter() - start
60  print(f"Time: {t3:.4f}s")
61  display(sorted_price)
62
63  print("\n=== Sort by Quantity (Quick Sort) ===")
64  start = time.perf_counter()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public>  c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anacon
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920'
122     Product_122     11.15       333
2235    Product_2235    11.41       386
6376    Product_6376    11.61       320
5937    Product_5937    11.75       324
3876    Product_3876    11.77       356
6326    Product_6326    12.62       235
8471    Product_8471    13.08       105
2856    Product_2856    13.3        141
```


Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
sorted_price = quick_sort(inventory, "price")
t3 = time.perf_counter() - start
print(f"Time: {t3:.4f}s")
display(sorted_price)
print("\n=== Sort by Quantity (Quick Sort) ===")
start = time.perf_counter()
sorted_qty = quick_sort(inventory, "qty")
t4 = time.perf_counter() - start
print(f"Time: {t4:.4f}s")
display(sorted_qty)
print("\n=== Algorithm Recommendation Table ===")
print(f"{'Operation':<25}{'Algorithm':<20}{'Justification'}")
print("-" * 75)
print(f"{'Search by ID':<25}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
print(f"{'Search by Name':<25}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
print(f"{'Sort by Price/Qty':<25}{'Quick Sort':<20}{'O(n log n) avg, fast in practice'}")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Public\AAC A 12.3.
=== Sort by Quantity (Quick Sort) ===
Time: 0.0557s
ID Name Price Qty
85 Product_85 4139.68 0
441 Product_441 4613.7 0
1338 Product_1338 3868.26 0
1408 Product_1408 56.08 0

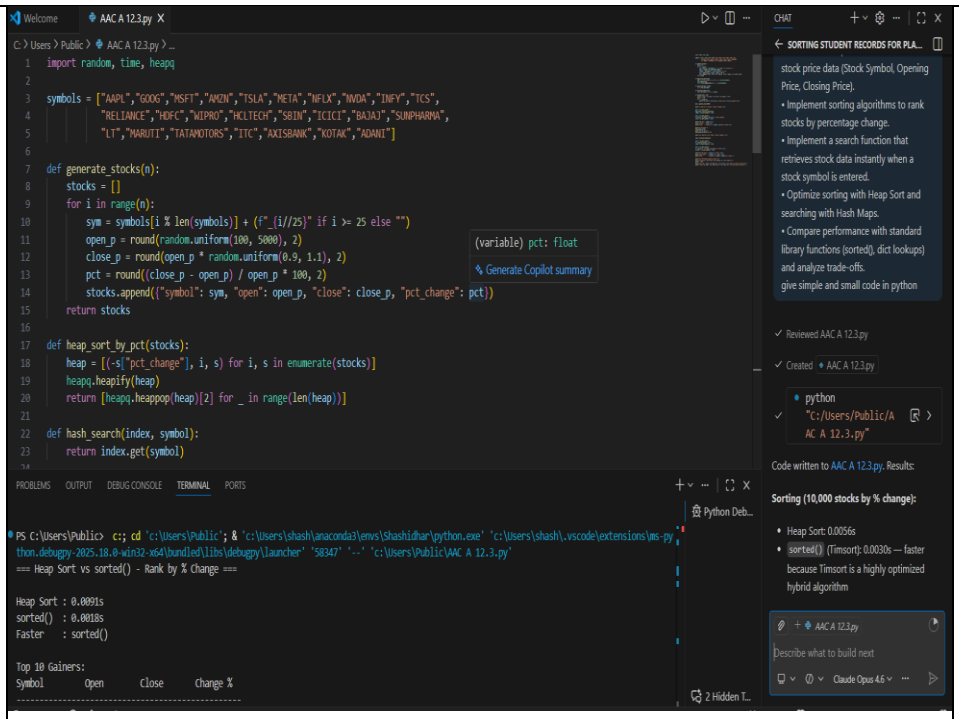

Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
sorted_price = quick_sort(inventory, "price")
t3 = time.perf_counter() - start
print(f"Time: {t3:.4f}s")
display(sorted_price)
print("\n=== Sort by Quantity (Quick Sort) ===")
start = time.perf_counter()
sorted_qty = quick_sort(inventory, "qty")
t4 = time.perf_counter() - start
print(f"Time: {t4:.4f}s")
display(sorted_qty)
print("\n=== Algorithm Recommendation Table ===")
print(f"{'Operation':<25}{'Algorithm':<20}{'Justification'}")
print("-" * 75)
print(f"{'Search by ID':<25}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
print(f"{'Search by Name':<25}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
print(f"{'Sort by Price/Qty':<25}{'Quick Sort':<20}{'O(n log n) avg, fast in practice'}")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Public\AAC A 12
441 Product_441 4613.7 0
1338 Product_1338 3868.26 0
1408 Product_1408 56.08 0
1740 Product_1740 660.14 0
2077 Product_2077 2768.81 0
2597 Product_2597 299.16 0
2837 Product_2837 3044.18 0
3438 Product_3438 3553.94 0
3474 Product_3474 3410.59 0

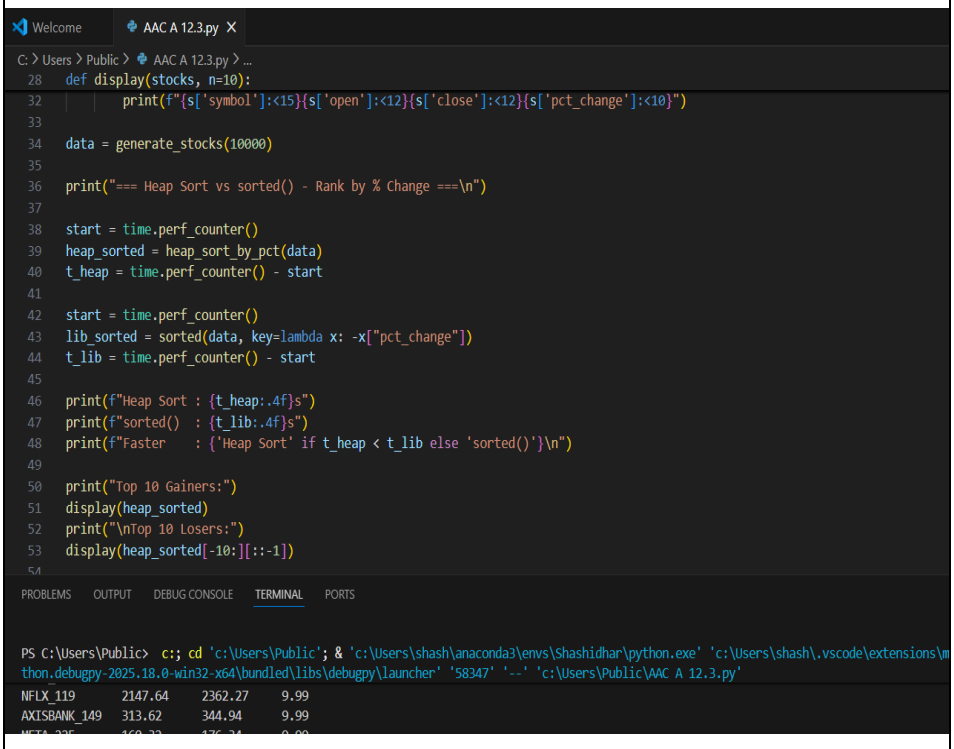


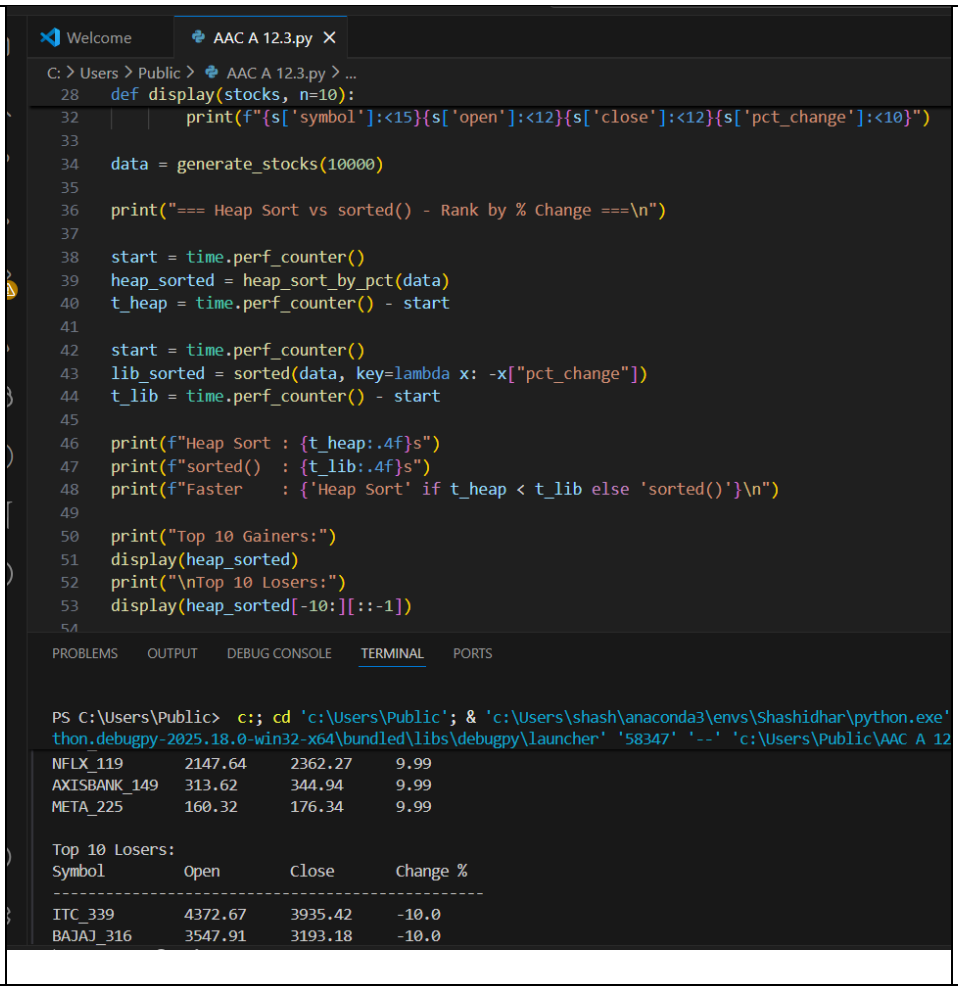
```
C: > Users > Public > AAC A 12.3.py > ...
58  sorted_price = quick_sort(inventory, "price")
59  t3 = time.perf_counter() - start
60  print(f"Time: {t3:.4f}s")
61  display(sorted_price)
62
63  print("\n=== Sort by Quantity (Quick Sort) ===")
64  start = time.perf_counter()
65  sorted_qty = quick_sort(inventory, "qty")
66  t4 = time.perf_counter() - start
67  print(f"Time: {t4:.4f}s")
68  display(sorted_qty)
69
70  print("\n=== Algorithm Recommendation Table ===")
71  print(f"{'Operation':<25}{'Algorithm':<20}{'Justification'}")
72  print("-" * 75)
73  print(f"{'Search by ID':<25}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
74  print(f"{'Search by Name':<25}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
75  print(f"{'Sort by Price/Qty':<25}{'Quick Sort':<20}{'O(n log n) avg, fast in practice'}")
76

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Public>  c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' '
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Public\AAC A 12.3
3474    Product_3474    3410.59     0

=== Algorithm Recommendation Table ===
Operation               Algorithm           Justification
---------------------------------------------------------------------------
Search by ID            Binary Search       O(log n), data sorted by ID
Search by Name          Hash Map (dict)     O(1) average lookup
Sort by Price/Qty       Quick Sort          O(n log n) avg, fast in practice
PS C:\Users\Public>
```

Task 5: 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.


Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
import random, time, heapq
symbols = ["AAPL","GOOG","MSFT","AMZN","TSLA","META","NFLX","NVDA","INFY","TCS",
"RELIANCE","HDFC","WIPRO","HCLTECH","SBIN","ICICI","BAJAJ","SUNPHARMA",
"LT","MARUTI","TATAMOTORS","ITC","AXISBANK","KOTAK","ADANI"]
def generate_stocks(n):
stocks = []
for i in range(n):
sym = symbols[i % len(symbols)] + (f"_{i//25}" if i >= 25 else "")
open_p = round(random.uniform(100, 5000), 2)
close_p = round(open_p * random.uniform(0.9, 1.1), 2)
pct = round((close_p - open_p) / open_p * 100, 2)
stocks.append({"symbol": sym, "open": open_p, "close": close_p, "pct_change": pct})
return stocks
def heap_sort_by_pct(stocks):
heap = [(-s["pct_change"], i, s) for i, s in enumerate(stocks)]
heapq.heapify(heap)
return [heapq.heappop(heap)[2] for _ in range(len(heap))]
def hash_search(index, symbol):
return index.get(symbol)
(variable) pct: float
Generate Copilot summary
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
=== Heap Sort vs sorted() - Rank by % Change ===
Heap Sort : 0.0091s
sorted() : 0.0018s
Faster : sorted()
Top 10 Gainers:
Symbol Open Close Change %
CHAT
SORTING STUDENT RECORDS FOR PLA...
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.
give simple and small code in python
Reviewed AAC A 12.3.py
Created AAC A 12.3.py
python "C:/Users/Public/AAC A 12.3.py"
Code written to AAC A 12.3.py. Results:
Sorting (10,000 stocks by % change):
Heap Sort: 0.0056s
sorted() (Timsort): 0.0030s — faster because Timsort is a highly optimized hybrid algorithm
AAC A 12.3.py
Describe what to build next
Claude Opus 4.6
Python Deb...
2 Hidden T...


Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
def display(stocks, n=10):
print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
data = generate_stocks(10000)
print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
start = time.perf_counter()
heap_sorted = heap_sort_by_pct(data)
t_heap = time.perf_counter() - start
start = time.perf_counter()
lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
t_lib = time.perf_counter() - start
print(f"Heap Sort : {t_heap:.4f}s")
print(f"sorted() : {t_lib:.4f}s")
print(f"Faster : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
print("Top 10 Gainers:")
display(heap_sorted)
print("\nTop 10 Losers:")
display(heap_sorted[-10:][::-1])
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
NFLX_119 2147.64 2362.27 9.99
AXISBANK_149 313.62 344.94 9.99



Welcome | AAC A 12.3.py

C: > Users > Public > AAC A 12.3.py > ...

```
28  def display(stocks, n=10):
32          print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
33
34  data = generate_stocks(10000)
35
36  print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
37
38  start = time.perf_counter()
39  heap_sorted = heap_sort_by_pct(data)
40  t_heap = time.perf_counter() - start
41
42  start = time.perf_counter()
43  lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
44  t_lib = time.perf_counter() - start
45
46  print(f"Heap Sort : {t_heap:.4f}s")
47  print(f"sorted()  : {t_lib:.4f}s")
48  print(f"Faster    : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
49
50  print("Top 10 Gainers:")
51  display(heap_sorted)
52  print("\nTop 10 Losers:")
53  display(heap_sorted[-10:][::-1])
54
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public>  c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12
NFLX_119        2147.64     2362.27     9.99
AXISBANK_149    313.62      344.94      9.99
META_225        160.32      176.34      9.99

Top 10 Losers:
Symbol          Open        Close       Change %
--------------------------------------------------
ITC_339         4372.67     3935.42     -10.0
BAJAJ_316       3547.91     3193.18     -10.0
```

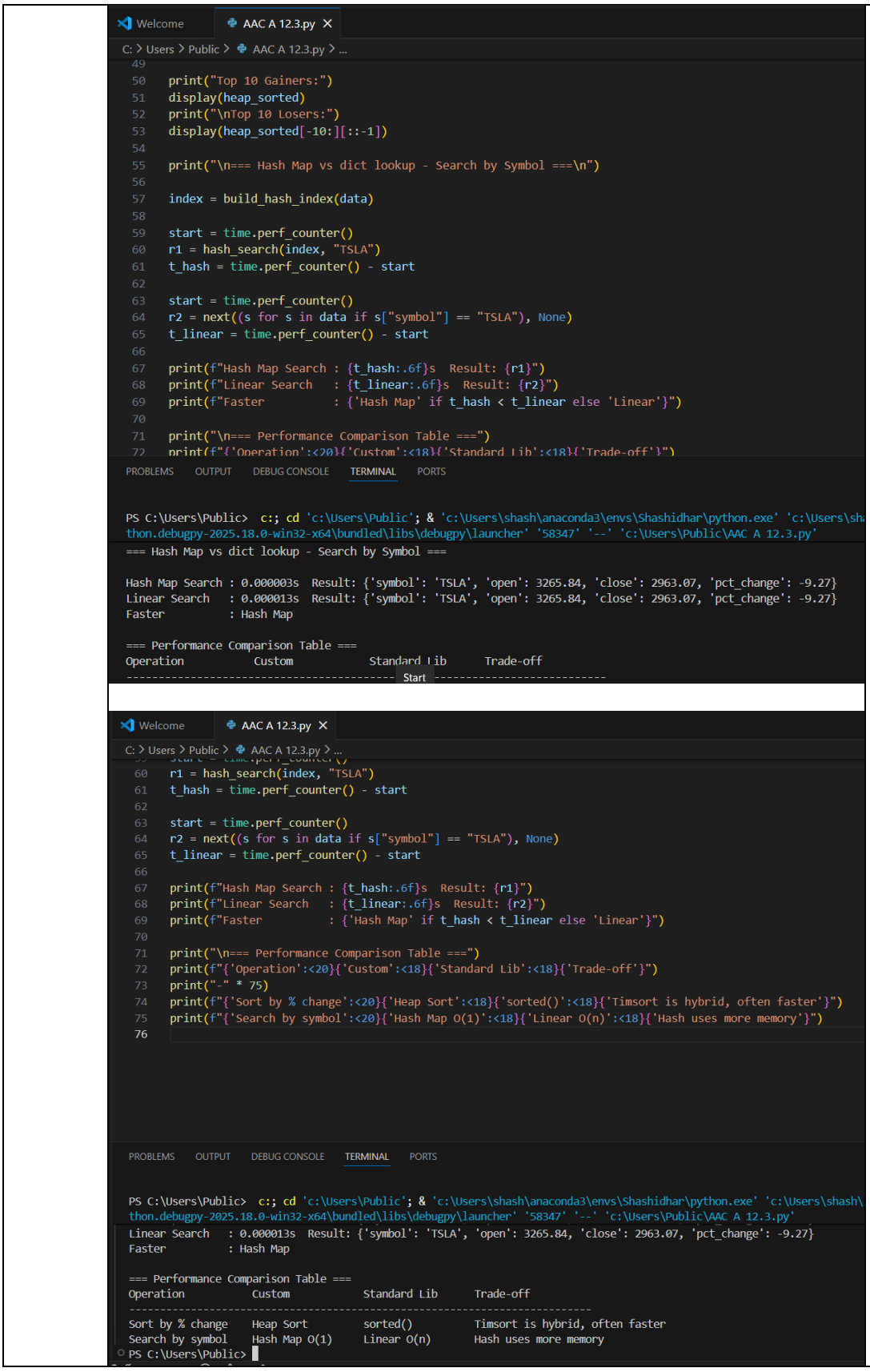
Welcome | AAC A 12.3.py

C: > Users > Public > AAC A 12.3.py > ...

```
28    def display(stocks, n=10):
32            print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
33
34    data = generate_stocks(10000)
35
36    print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
37
38    start = time.perf_counter()
39    heap_sorted = heap_sort_by_pct(data)
40    t_heap = time.perf_counter() - start
41
42    start = time.perf_counter()
43    lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
44    t_lib = time.perf_counter() - start
45
46    print(f"Heap Sort : {t_heap:.4f}s")
47    print(f"sorted()  : {t_lib:.4f}s")
48    print(f"Faster    : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
49
50    print("Top 10 Gainers:")
51    display(heap_sorted)
52    print("\nTop 10 Losers:")
53    display(heap_sorted[-10:][::-1])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public>  c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.
ADANI_258       3536.14     3182.58     -10.0
HCLTECH_254     3068.46     2761.7      -10.0
SUNPHARMA_222   3697.36     3327.72     -10.0
MARUTI_93       1783.28     1604.99     -10.0
WIPRO_334       861.71      775.65      -9.99
ITC_327         2201.07     1981.19     -9.99
GOOG_285        2888.79     2600.15     -9.99
NFLX_256        2957.21     2661.79     -9.99
```


Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
print("Top 10 Gainers:")
display(heap_sorted)
print("\nTop 10 Losers:")
display(heap_sorted[-10:][::-1])
print("\n=== Hash Map vs dict lookup - Search by Symbol ===\n")
index = build_hash_index(data)
start = time.perf_counter()
r1 = hash_search(index, "TSLA")
t_hash = time.perf_counter() - start
start = time.perf_counter()
r2 = next((s for s in data if s["symbol"] == "TSLA"), None)
t_linear = time.perf_counter() - start
print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
print("\n=== Performance Comparison Table ===")
print(f"{'Operation':<20}{'Custom':<18}{'Standard Lib':<18}{'Trade-off'}")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\sh
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
=== Hash Map vs dict lookup - Search by Symbol ===
Hash Map Search : 0.000003s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Linear Search : 0.000013s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Faster : Hash Map
=== Performance Comparison Table ===
Operation Custom Standard Lib Trade-off
Start
Welcome
AAC A 12.3.py
C: > Users > Public > AAC A 12.3.py > ...
r1 = hash_search(index, "TSLA")
t_hash = time.perf_counter() - start
start = time.perf_counter()
r2 = next((s for s in data if s["symbol"] == "TSLA"), None)
t_linear = time.perf_counter() - start
print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
print("\n=== Performance Comparison Table ===")
print(f"{'Operation':<20}{'Custom':<18}{'Standard Lib':<18}{'Trade-off'}")
print("-" * 75)
print(f"{'Sort by % change':<20}{'Heap Sort':<18}{'sorted()':<18}{'Timsort is hybrid, often faster'}")
print(f"{'Search by symbol':<20}{'Hash Map O(1)':<18}{'Linear O(n)':<18}{'Hash uses more memory'}")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
Linear Search : 0.000013s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Faster : Hash Map
=== Performance Comparison Table ===
Operation Custom Standard Lib Trade-off
Sort by % change Heap Sort sorted() Timsort is hybrid, often faster
Search by symbol Hash Map O(1) Linear O(n) Hash uses more memory
PS C:\Users\Public>

	Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.	