

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

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

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

|  |  |  |
|--|--|--|
|  | <p>The objectives of this laboratory exercise are to:</p> <ul style="list-style-type: none"> <li>• Apply AI-assisted programming techniques to implement sorting and searching algorithms.</li> <li>• Analyze and compare algorithm efficiency using time and space complexity.</li> <li>• Understand how AI tools can suggest optimizations and alternative algorithmic approaches.</li> <li>• Strengthen problem-solving skills through real-world, data-driven scenarios.</li> </ul>  |  |
|  | <p><b>Learning Outcomes</b></p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>• Implement and optimize classic algorithms using AI-assisted coding tools.</li> <li>• Compare multiple algorithms for the same problem and justify their selection.</li> <li>• Measure and analyze runtime performance using experimental data.</li> <li>• Critically review and refine AI-generated algorithmic solutions.</li> </ul>   |  |
|  | <p><b>Task 1: Sorting Student Records for Placement Drive</b></p> <p><b>Scenario</b></p> <p>SR University's Training and Placement Cell needs to shortlist candidates efficiently during campus placements. Student records must be sorted by <b>CGPA in descending order</b>.</p> <p><b>Tasks</b></p> <ol style="list-style-type: none"> <li>1. Use <b>GitHub Copilot</b> to generate a program that stores student records (Name, Roll Number, CGPA).</li> <li>2. Implement the following sorting algorithms using AI assistance: <ul style="list-style-type: none"> <li>○ Quick Sort</li> <li>○ Merge Sort</li> </ul> </li> <li>3. Measure and compare runtime performance for large datasets.</li> <li>4. Write a function to display the <b>top 10 students</b> based on CGPA.</li> </ol> <p><b>Expected Outcome</b></p> <ul style="list-style-type: none"> <li>• Correctly sorted student records.</li> <li>• Performance comparison between Quick Sort and Merge Sort.</li> <li>• Clear output of top-performing students.</li> </ul> |  |

```

1 Welcome AAC A 12.3.py X
2 C:\Users\Public> AAC A 12.3.py <...
3 import random, time, copy
4
5 def generate_students(n):
6     names = ["Aarav", "Vivaan", "Aditya", "Arjun", "Sai", "Ananya", "Diya", "Rohan", "Neha", "Priya",
7             "Kahul", "Sneha", "Vikram", "Meera", "Karan", "Pooja", "Ishaan", "Riya", "Mait", "Navya"]
8     students = []
9     for i in range(1, n+1):
10        students.append({"name": random.choice(names), "roll": f"SRU{i:04d}", "cgpa": round(random.uniform(4.0, 10.0), 2)})
11    return students
12
13 def quick_sort(a):
14     if len(a) <= 1: return a
15     p = a[len(a)//2][“cgpa”]
16     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])
17
18 def merge_sort(a):
19     if len(a) <= 1: return a
20     m = len(a)//2
21     l, r = merge_sort(a[:m]), merge_sort(a[m:])
22     res, i, j = [], 0, 0
23     while i < len(l) and j < len(r):
24         if l[i][“cgpa”] > r[j][“cgpa”]: res.append(l[i]); i += 1
25         else: res.append(r[j]); j += 1
26     return res + l[i:] + r[j:]
27
28
29
30
31
32
33
34
35
36
37
38

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the command, or if a command contains spaces, enclose it in quotes.

PS C:\Users\Public> & (C:\Users\shash\anaconda\envs\Shashihaar\python.exe) 'C:\Users\shash\vscode\extensions\ms-python.debug-2025.18.0-win32-x64\bundled\libs\debug\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 |
|------|--------|---------|------|
| 1    | Ananya | SRU0282 | 10.0 |
| 2    | Pooja  | SRU0382 | 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  | SRU7699 | 10.0 |

CHAT + × ⌂ ⌂ X

← SORTING STUDENT RECORDS FOR PL...

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:

Python Deb...

Describe what to build next

Claude Opus 4.6 × 3k

2 Hidden T...

CHAT + × ⌂ ⌂ X

← SORTING STUDENT RECORDS FOR PL...

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 × 3k

Describe what to build next

Claude Opus 4.6 × 3k

```

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':<12}}")
28     print("-*38)
29     for i, s in enumerate(students[:10], 1):
30         print(f"\n{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.

**AAC A 12.3.py**

```

1 def bubble_sort(arr):
2     n = len(arr)
3     for i in range(n - 1):
4         swapped = False
5         for j in range(0, n - 1 - i):
6             if arr[j] > arr[j + 1]:
7                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
8                 swapped = True
9         if not swapped:
10            break
11    return arr
12
13 data = [64, 34, 25, 12, 22, 11, 90]
14 print("Before:", data)
15 print("After :", bubble_sort(data))
16

```

**CHAT**

← SORTING STUDENT RECORDS FOR PLA... X

- Expected Output:
  - A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis.
  - give small and simple code in python without comments

✓ Created AAC A 12.3.py

python C:/Users/Public/AAC A 12.3.py

Code written to AAC A 12.3.py. Output:

Before: [64, 34, 25, 12, 22, 11, 90]  
After : [11, 12, 22, 25, 34, 64]

Time Complexity Analysis:

- Best Case  $O(n)$  — array already sorted, swapped flag breaks after first pass
- Worst Case  $O(n^2)$  — array sorted in reverse order, every pair needs swapping
- Average Case  $O(n^2)$
- Space Complexity  $O(1)$  — in-place sorting

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python Debug...

32-x64(bundled\libs\debug)\launcher" "54674" ... "c:\Users\Public\AAC A 12.3.py"

6 Aarav SRU0875 10.0  
7 Vilvaan SRU5385 10.0  
8 Karan SRU7699 10.0  
9 Pooja SRU8102 10.0  
10 Priya SRU8824 10.0

PS C:\Users\Public> cd ..& "c:\Users\shash\anaconda3\envs\Shashidhar\python.exe" "c:\Users\shash\vscode\extensions\ms-py

thon.debug-2025.18.0-win32-x64\bundled\libs\debug)\launcher" "64314" ... "c:\Users\Public\AAC A 12.3.py"

Before: [64, 34, 25, 12, 22, 11, 90]  
After : [11, 12, 22, 25, 34, 64, 90]

PS C:\Users\Public> [ ]

2 Hidden T...

← AAC A 12.3.py

Describe what to build next

Claude Opus 4.6 v ...

**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.

The screenshot shows two tabs open in VS Code:

- AAC A 12.3.py**: This tab contains Python code for implementing Quick Sort and Merge Sort. The code includes imports for random and arr, and defines functions for quick\_sort and merge\_sort. It uses list comprehensions and loops to partition and merge arrays.
- SORTING STUDENT RECORDS FOR PL...**: This tab displays a complexity analysis table comparing Quick Sort and Merge Sort across different data cases (Random, Sorted, Reversed) and provides a detailed breakdown of their time complexities.

**Complexity Analysis Table (AAC A 12.3.py):**

| Case     | Quick Sort | Merge Sort |
|----------|------------|------------|
| Random   | 0.0315s    | 0.03s      |
| Sorted   | 0.0204s    | 0.02s      |
| Reversed | 0.0263s    | 0.03s      |

**Complexity Analysis Table (AAC A 12.3.py):**

|         | Quick Sort                         | Merge Sort            |
|---------|------------------------------------|-----------------------|
| Best    | $O(n \log n)$                      | $O(n \log n)$         |
| Average | $O(n \log n)$                      | $O(n \log n)$         |
| Worst   | $O(n^2)$ (sorted input, bad pivot) | $O(n \log n)$         |
| Space   | $O(\log n)$ (call stack)           | $O(n)$ (merge buffer) |

#### 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.

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

- Editor:** AAC A 12.3.py
 

```
1 import random, time
2
3 def generate_inventory(n):
4     return [{"id": i, "name": f"Product_{i}"}, {"price": round(random.uniform(10, 5000), 2),
5         "qty": random.randint(0, 500)} for i in range(1, n + 1)]
6
7 def binary_search_by_id(products, target_id):
8     lo, hi = 0, len(products) - 1
9     while lo <= hi:
10         mid = (lo + hi) // 2
11         if products[mid]["id"] == target_id:
12             return products[mid]
13         elif products[mid]["id"] < target_id:
14             lo = mid + 1
15         else:
16             hi = mid - 1
17     return None
18
19 def hash_search_by_name(index, name):
20     return index.get(name)
21
22 def build_name_index(products):
23     return {p["name"]: p for p in products}
```
- Terminal:**

```
PS C:\Users\Public> cd 'C:\Users\Public'; & 'C:\Users\shash\anaconda\envs\shashihar\python.exe' 'C:\Users\shash\vscode\extensions\ms-vscode.cpptools\lib\python\lsp\client.py'
● thon.debugpy-2025.18.0-win32-x64\bundle\libs\vebgepy\launcher' '57928' '--' 'C:\Users\Public\AAC A 12.3.py'
*** Search by ID (Binary Search) ===
Found: {'id': 5000, 'name': 'Product_5000', 'price': 2625.52, 'qty': 428} Time: 0.000007s
*** Search by Name (Hash Map) ===
Found: {'id': 5000, 'name': 'Product_5000', 'price': 2625.52, 'qty': 428} Time: 0.000002s
*** Sort by Price (Quick Sort) ===
Time: 0.0200s
ID      Name      Price      Qty
```
- Right Sidebar:**
  - CHAT:** Chat window with messages about sorting student records.
  - Code Review:** Shows a review of the code with comments like "Reviewed and created AAC A 12.3.py" and "Created AAC A 12.3.py".
  - Code Implementations:** Compares different search methods:
 

| Operation      | Algorithm     | Justification  |
|----------------|---------------|--|
| Search by ID   | Binary Search | $O(\log n)$ — IDs are sequentially/sorted, minimal comparisons   |
| Search by Name | Hash Map      | $O(1)$ average — instant lookup, ideal for frequent name queries |
| Sort by Price  | Quick Sort    | $O(n \log n)$ average — fast in practice, good cache locality    |
  - Why these choices for inventory:** Describes what to build next.

VS Code interface showing a Python script named AAC A 12.3.py.

```
C: > Users > Public > AAC A 12.3.py > ...
22     def build_name_index(products):
23         return {p["name"] : p for p in products}
24
25     def quick_sort(arr, key):
26         if len(arr) <= 1:
27             return arr
28         pivot = arr[len(arr) // 2][key]
29         left = [x for x in arr if x[key] < pivot]
30         mid = [x for x in arr if x[key] == pivot]
31         right = [x for x in arr if x[key] > pivot]
32         return quick_sort(left, key) + mid + quick_sort(right, key)
33
34     def display(products, n=10):
35         print(f"{'ID':<8}{'Name':<16}{'Price':<12}{'Qty':<6}")
36         print("-" * 42)
37         for p in products[:n]:
38             print(f"{p['id']:<8}{p['name']:<16}{p['price']:<12}{p['qty']:<6}")
39
40     inventory = generate_inventory(10000)
41
42     print("== Search by ID (Binary Search) ==")
43     sorted_by_id = sorted(inventory, key=lambda x: x["id"])
44     start = time.perf_counter()

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
```

Terminal output:

```
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\users\Publ...
```

| 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 |

The screenshot shows a terminal window with a dark theme. At the top, there are tabs for 'Welcome' and 'AAC A 12.3.py X'. The main area displays Python code for searching and sorting an inventory. The code includes functions for binary search by ID, hash search by name, quick sort by price, and quick sort by quantity. It also prints the sorted inventory. Below the code, the terminal shows a command to change directory and run a launcher, followed by a table of sorted inventory items.

```
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()
65
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
```

```
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\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 > ...
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()
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("{'Operation':<25}{Algorithm':<20}{'Justification'}")
72     print("-" * 75)
73     print("{'Search by ID':<25}{Binary Search':<20}{'O(log n), data sorted by ID'}")
74     print("{'Search by Name':<25}{Hash Map (dict)':<20}{O(1) average lookup'}")
75     print("{'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' 'c:\Users\Public\AAC A 12.3.py'
==== 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
```

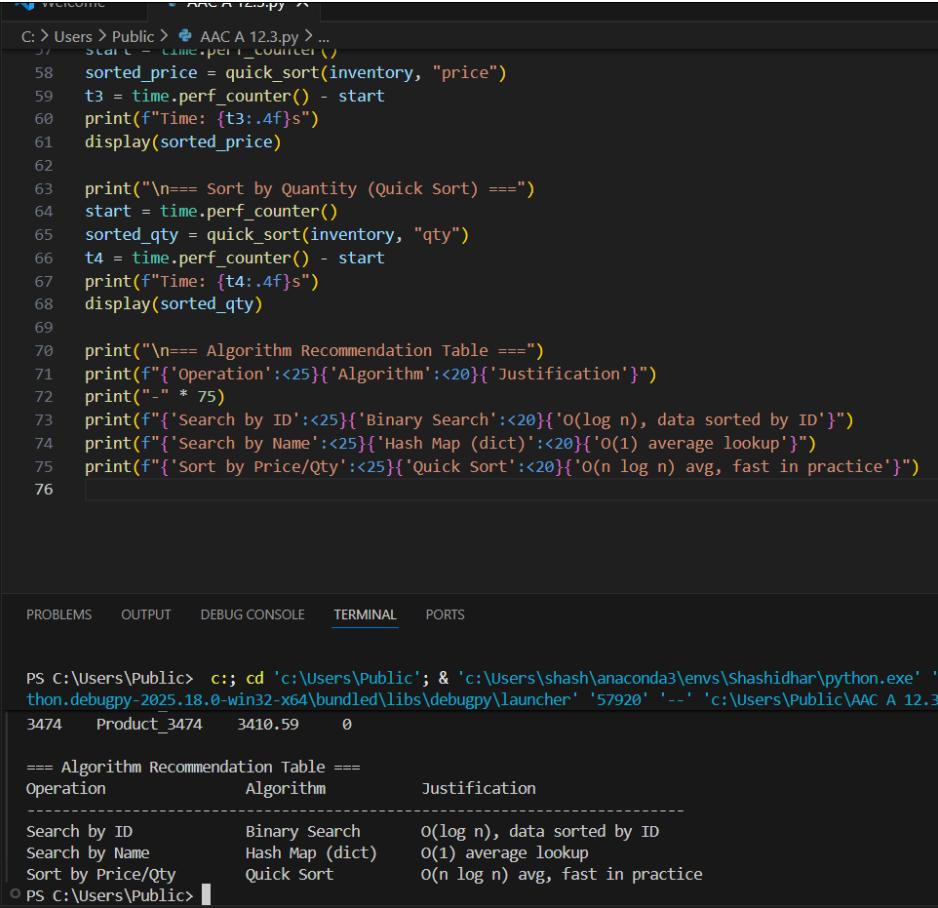
The screenshot shows a code editor interface with a terminal window below it.

**Code Editor (AAC A 12.3.py):**

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

**Terminal:**

```
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.py'
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 > ...
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()
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' 'aac.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

```

**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.

**SORTING STUDENT RECORDS FOR PLU...**

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

PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debug-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 : 0.0081s
sorted() : 0.0088s
Faster   : sorted()

Top 10 Gainers:
Symbol      Open     Close    Change %
-----
```

PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debug-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 ==\n

```
def display(stocks, n=10):
    print(f"[{stocks['symbol'][:15]}[{stocks['open'][:12]}{stocks['close'][:12]}{stocks['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])
```

PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debug-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    |      |
| META_22E     | 160.22   | 176.24  | 9.99    |      |

The screenshot shows a terminal window with the following content:

```
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
```

TERMINAL

```
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
BAJAJI_316   3547.91   3193.18   -10.0
```

The screenshot shows a Jupyter Notebook interface with two cells. The top cell contains Python code for comparing heap sort and library sort performance, and displaying the top 10 gainers and losers. The bottom cell shows the resulting output.

```
C: > Users > Public > AAC A 12.3.py > ...
28 def display(stocks, n=10):
29     print(f'{stocks[0].symbol}: {stocks[0].open:.2f} {stocks[0].close:.2f} {stocks[0].pct_change:.2%}')
30
31     data = generate_stocks(10000)
32
33     print("==> Heap Sort vs sorted() - Rank by % Change ==\n")
34
35     start = time.perf_counter()
36     heap_sorted = heap_sort_by_pct(data)
37     t_heap = time.perf_counter() - start
38
39     start = time.perf_counter()
40     lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
41     t_lib = time.perf_counter() - start
42
43     print(f"Heap Sort : {t_heap:.4f}s")
44     print(f"sorted() : {t_lib:.4f}s")
45     print(f"Faster : {'Heap sort' if t_heap < t_lib else 'sorted()' }\n")
46
47     print("Top 10 Gainers:")
48     display(heap_sorted)
49     print("\nTop 10 Losers:")
50     display(heap_sorted[-10:][::-1])
51
52
53
```

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\bundle\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
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    1684.99    -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
```

```
C:\> Users > Public > AAC A 12.3.py > ...
49
50     print("Top 10 Gainers:")
51     display(heap_sorted)
52     print("\nTop 10 Losers:")
53     display(heap_sorted[-10:][::-1])
54
55     print("\n==== Hash Map vs dict lookup - Search by symbol ===\n")
56
57     index = build_hash_index(data)
58
59     start = time.perf_counter()
60     r1 = hash_search(index, "TSLA")
61     t_hash = time.perf_counter() - start
62
63     start = time.perf_counter()
64     r2 = next(s for s in data if s["symbol"] == "TSLA", None)
65     t_linear = time.perf_counter() - start
66
67     print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
68     print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
69     print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
70
71     print("\n==== Performance Comparison Table ===")
72     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\shashidhar\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 -----
```

---

```
C:\> Users > Public > AAC A 12.3.py > ...
50     r1 = hash_search(index, "TSLA")
51     t_hash = time.perf_counter() - start
52
53     start = time.perf_counter()
54     r2 = next(s for s in data if s["symbol"] == "TSLA", None)
55     t_linear = time.perf_counter() - start
56
57     print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
58     print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
59     print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
60
61     print("\n==== Performance Comparison Table ===")
62     print(f"{'Operation':<20}{{'Custom':<18}{{'Standard Lib':<18}{{'Trade-off'}}")
63     print("-" * 75)
64     print(f"{'sort by % change':<20}{{'Heap sort':<18}{{'sorted()':<18}{{'Timsort is hybrid, often faster'}}")
65     print(f"{'Search by symbol':<20}{{'Hash Map O(1)':<18}{{'Linear O(n)':<18}{{'Hash uses more memory'}}")
66
67     print("\n==== Performance Comparison Table ===")
68     print(f"{'Operation':<20}{{'Custom':<18}{{'Standard Lib':<18}{{'Trade-off'}}")
69     print("-" * 75)
70     print(f"{'sort by % change':<20}{{'Heap sort':<18}{{'sorted()':<18}{{'Timsort is hybrid, often faster'}}")
71     print(f"{'Search by symbol':<20}{{'Hash Map O(1)':<18}{{'Linear O(n)':<18}{{'Hash uses more memory'}}")
72
73     print("\n==== Performance Comparison Table ===")
74     print(f"{'Operation':<20}{{'Custom':<18}{{'Standard Lib':<18}{{'Trade-off'}}")
75     print("-" * 75)
76
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\shashidhar\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
----- Start -----
```

---

```
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>
```

|  |   |  |
|--|---|--|
|  | <p>Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments &amp; code explanation, and output and if required, screenshots.</p> |  |
|--|---|--|