

Name:G.VenkataRamana H.No:2303A51893

Batch:25

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 – Wednesday	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		Week3 - Wednesday

	<p><b>Lab Objectives</b></p> <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>	

**SORTING STUDENT RECORDS FOR PLACEMENT**

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...

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

Describe what to build next

Claude Opus 4.6 v3s

**SORTING STUDENT RECORDS FOR PLACEMENT**

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 v3s

Describe what to build next

Claude Opus 4.6 v3s

The screenshot shows a code editor window with the following Python code:

```

1 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

```

The code includes imports for `copy` and `time`. It defines functions for `merge_sort`, `top_10`, and a loop that generates student data of different sizes, sorts them using both quick sort and merge sort, and prints the results along with their execution times.

Below the code editor is a terminal window showing the output of the script:

```

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.

The screenshot shows a code editor interface with a Python file named 'AAC A 12.3.py'. The code implements a bubble sort algorithm. The terminal below shows the execution of the script with sample data [64, 34, 25, 12, 22, 11, 98]. The output shows the array before and after sorting.

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        swapped = False
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr

data = [64, 34, 25, 12, 22, 11, 98]
print("Before:", data)
print("After :", bubble_sort(data))
```

Terminal Output:

```
PS C:\Users\Public> c:\; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-vscode\python\lib\site-packages\thon\debugpy-2023.1.0-win32-x64\bundle\libs\debugpy\launcher' '64314' '--' 'c:\Users\Public\AAC A 12.3.py'
Before: [64, 34, 25, 12, 22, 11, 98]
After : [11, 12, 22, 25, 34, 64, 98]
```

CHAT

Sorting Student Records for PLA...

- Expected Output:
  - A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis.
- 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, 98]
After : [11, 12, 22, 25, 34, 64, 98]
```

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

Describe what to build next

Claude Opus 4.6

**AAC A 12.3.py**

```

1 import random, time
2 def quick_sort(arr):
3     if len(arr) <= 1:
4         return arr
5     pivot = arr[len(arr)] // 2
6     left = [x for x in arr if x < pivot]
7     mid = [x for x in arr if x == pivot]
8     right = [x for x in arr if x > pivot]
9     return quick_sort(left) + mid + quick_sort(right)
10 def merge_sort(arr):
11     if len(arr) <= 1:
12         return arr
13     mid = len(arr) // 2
14     left = merge_sort(arr[:mid])
15     right = merge_sort(arr[mid:])
16     result, i, j = [], 0, 0
17     while i < len(left) and j < len(right):
18         if left[i] <= right[j]:
19             result.append(left[i]); i += 1
20         else:
21             result.append(right[j]); j += 1
22     return result + left[i:] + right[j:]
23
24 def measure(func, data):
25     start = time.perf_counter()
26     func(data[:])
27     return time.perf_counter() - start
28
29 n = 10000
30 cases = [
31     "Random": random.sample(range(n), n),
32     "Sorted": list(range(n)),
33     "Reversed": list(range(n, 0, -1))
34 ]
35
36 print("{'Case':<12>} {'Quick Sort':<14>} {'Merge Sort':<14>}")
37 print("-" * 40)
38 for name, data in cases.items():
39     qt = measure(quick_sort, data)
40     mt = measure(merge_sort, data)
41     print(f"{name:<12>} {qt:.4f}s" f" {mt:.4f}s")
42

```

PS C:\Users\Public> cd "C:\Users\Public"; & "C:\Users\shash\anaconda3\envs\shash\bin\python.exe" "C:\Users\shash\.vscode\extensions\ms-python.python\\_debug\_2025.18.0-win32-x64\bundled\libs\debug\launcher" "56314" -- "C:\Users\Public\AAC A 12.3.py"

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

PS C:\Users\Public> cd "C:\Users\Public"; & "C:\Users\shash\anaconda3\envs\shash\bin\python.exe" "C:\Users\shash\.vscode\extensions\ms-python.python\\_debug\_2025.18.0-win32-x64\bundled\libs\debug\launcher" "51664" -- "C:\Users\Public\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:**

	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)

Quick Sort is generally faster in practice due to better cache locality, but Merge Sort guarantees  $O(n \log n)$  in all cases.

PS C:\Users\Public> Claude Opus 4.6 + 3r

PS C:\Users\Public> "C:\Users\shash\anaconda3\envs\shash\bin\python.exe" "C:\Users\shash\.vscode\extensions\ms-python.python\\_debug\_2025.18.0-win32-x64\bundled\libs\debug\launcher" "51664" -- "C:\Users\Public\AAC A 12.3.py"

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

PS C:\Users\Public> cd "C:\Users\Public"; & "C:\Users\shash\anaconda3\envs\shash\bin\python.exe" "C:\Users\shash\.vscode\extensions\ms-python.python\\_debug\_2025.18.0-win32-x64\bundled\libs\debug\launcher" "51664" -- "C:\Users\Public\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

## 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 the Visual Studio Code interface with the following details:

- Editor Area:** Displays the code for `AAC A 12.3.py`. The code implements three methods: `generate_inventory`, `binary_search_by_id`, and `hash_search_by_name`.
- Terminal:** Shows the command `PS C:\Users\Public> cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-vscode\python\thon.debug-2025.18.0-win32-x64\bundle\libs\debug\launcher' '57928' -- 'c:\Users\Public\AAC A 12.3.py'` being run, with output indicating successful execution of the script.
- Output Panel:** Shows the results of the script execution, including search operations and sort results.
- Chat Panel:** Contains a message about sorting student records for planning and sorting the inventory.
- Code Review Panel:** Shows a checklist for the code review, with items like "Reviewed and created AAC A 12.3.py" and "Created AAC A 12.3.py".
- Algorithm Analysis Sidebar:** Compares different search and sort operations:
  - Search by ID (Binary Search):**  $O(\log n)$  — IDs are sequential/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.

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
```

The screenshot shows a terminal window with the following content:

```
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']}")
```

TERMINAL

```
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
```

```
C:\> Users > Public > AAC A 12.3.py > ...
58     start = time.perf_counter()
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':<20}{'Algorithm':<20}{'Justification'}")
72     print("-" * 75)
73     print(f"{'Search by ID':<20}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
74     print(f"{'Search by Name':<20}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
75     print(f"{'Sort by Price/Qty':<20}{'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
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.

```

Welcome AAC A 12.3.py X
C:\Users\Public> AAC A 12.3.py ...
1 import random, time, heapq
2
3 symbols = ["AMZN", "GOOG", "MSFT", "AMZN", "TSLA", "META", "NFLX", "IVWD", "INFY", "TCS",
4             "RELIANCE", "HDFC", "WIPRO", "HCLTECH", "SEIN", "ICICITI", "BAJAJ", "SUNPHARMA",
5             "LT", "WARUTI", "TATAMOTORS", "ITC", "AXISBANK", "KOTAK", "ADAN"]
6
7 def generate_stocks(n):
8     stocks = []
9     for i in range(n):
10         sym = symbols[i % len(symbols)] + ("_(i//25)" if i >= 25 else "")
11         open_p = round(random.uniform(100, 5000), 2)
12         close_p = round(open_p * random.uniform(0.9, 1.1), 2)
13         pct = round((close_p - open_p) / open_p * 100, 2)
14         stocks.append({"symbol": sym, "open": open_p, "close": close_p, "pct_change": pct})
15
16     return stocks
17
18 def heap_sort_by_pct(stocks):
19     heap = [(-s["pct_change"], i, s) for i, s in enumerate(stocks)]
20     heapq.heapify(heap)
21     return [heapq.heappop(heap)[2] for _ in range(len(heap))]
22
23 def hash_search(index, symbol):
24     return index.get(symbol)
25

```

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.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 : 0.0001s  
sorted() : 0.0008s  
Faster : sorted()

Top 10 Gainers:

Symbol	Open	Close	Change %

2 Hidden ...

CHAT + ⚡ ...

SORTING STUDENT RECORDS FOR PL...

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(): 0.0030s – faster because TimSort is a highly optimized hybrid algorithm

Describe what to build next

CLAUDIA OPUS 4.5 ...

```

Welcome AAC A 12.3.py X
C:\Users\Public> AAC A 12.3.py ...
28 def display(stocks, n=10):
29     print(f"[{s['symbol']}]<15[{s['open']]<12}{s['close']]<12}{s['pct_change']]<10}")
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

```

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.debug-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'

Symbol	Open	Close	Change %
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 12.3.py'
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
```

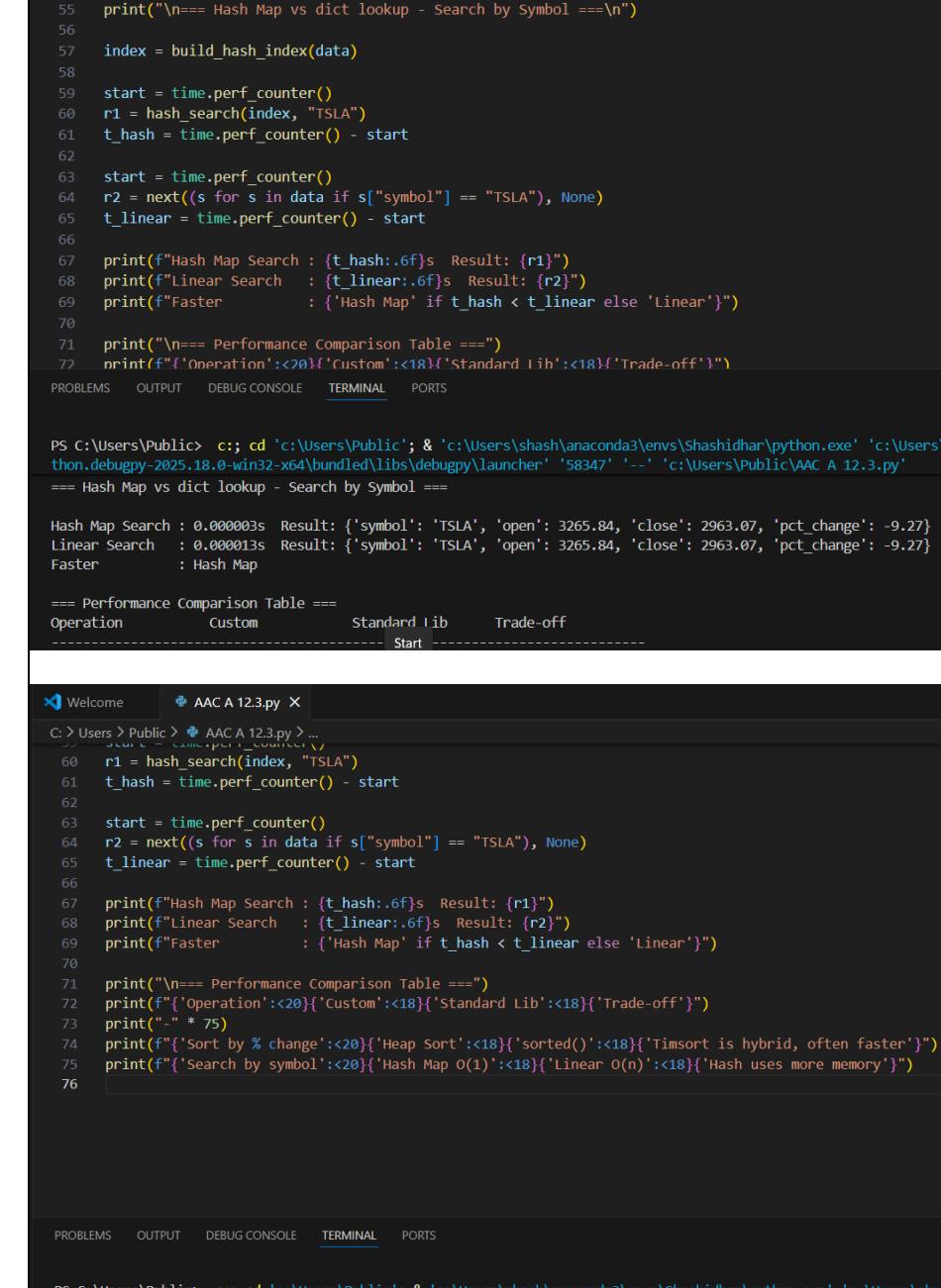
VS Code interface showing a Python script named AAC A 12.3.py.

The code implements a function `display(stocks, n=10)` which prints the top 10 gainers and losers based on percentage change. It uses both heap sort and a standard sorted() function for comparison.

```
C: > Users > Public > AAC A 12.3.py > ...
28 def display(stocks, n=10):
29     print(f'{s[ "symbol"]:<15}{s[ "open"]:<12}{s[ "close"]:<12}{s[ "pct_change"]:<10}')
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
```

The terminal output shows the execution of the script and the resulting sorted list of stocks:

```
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 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 ===")
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\shash\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 -----

```

---

```
C:\Users\Public > AAC A 12.3.py ...
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 ===")
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'}")
73     print("-" * 75)
74     print(f"{'Sort by % change':<20}{'Heap sort':<18}{'sorted()':<18}{'Timsort is hybrid, often faster'}")
75     print(f"{'Search by symbol':<20}{'Hash Map O(1)':<18}{'Linear O(n)':<18}{'Hash uses more memory'}")
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\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
----- 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

```

	<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>	
--	---	--