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

SCourse 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 Mr. G. Kranthi

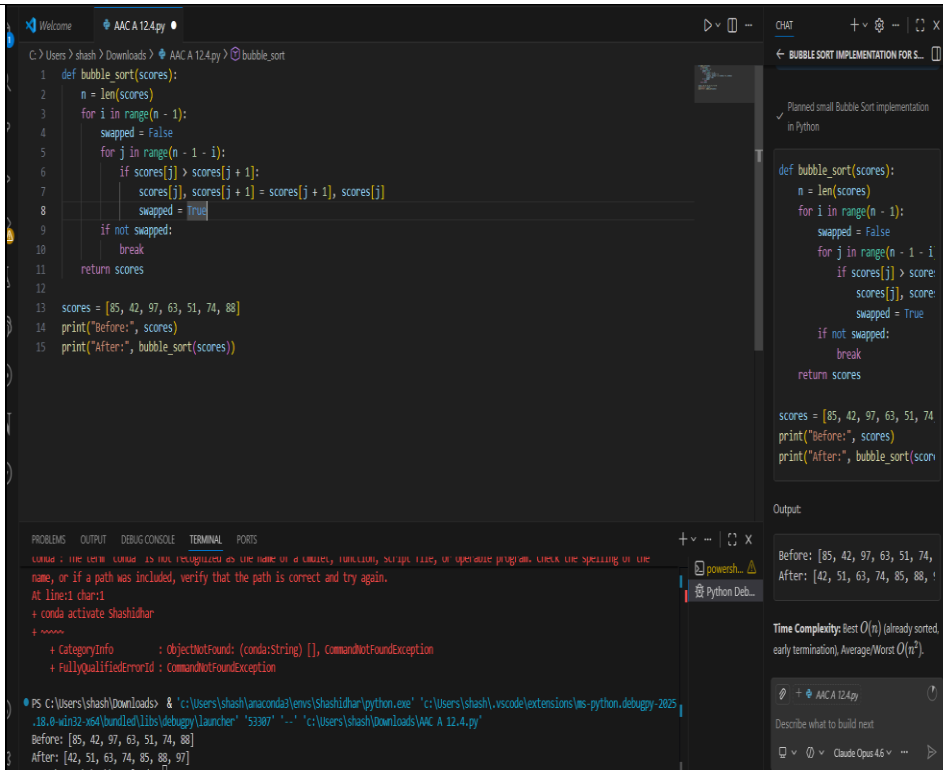
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/I	Regulation	R23
Date and Day of Assignment	Week 6 - Thursday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All Batches

AssignmentNumber:12.4 (Present assignment number)/**24**(Total number of assignments)

Q.No.	Question	ExpectedTime to complete
1	**Lab 12 – Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms**	Week 6

	Lab Objectives • Apply AI-assisted programming to implement and optimize sorting and searching algorithms. • Compare different algorithms in terms of efficiency and use cases. • Understand how AI tools can suggest optimized code and complexity improvements. **Learning Outcome** After completing this assignment, students will be able to: • Implement classical algorithms with AI assistance • Compare algorithm efficiency using real-world scenarios • Understand when optimization is necessary • Critically evaluate AI-generated suggestions instead of blindly accepting them	

	Task 1: Bubble Sort for Ranking Exam Scores
	
	Scenario
	
	You are working on a **college result processing system** where a small list of student scores needs to be sorted after every internal assessment.
	
	Task Description
	
	• Implement **Bubble Sort** in Python to sort a list of student scores.
	• Use an AI tool to:
	  o Insert inline comments explaining key operations such as comparisons, swaps, and iteration passes
	  o Identify early-termination conditions when the list becomes sorted
	  o Provide a brief time complexity analysis
	
	Expected Outcome
	
	• A Bubble Sort implementation with:
	  o AI-generated comments explaining the logic
	  o Clear explanation of best, average, and worst-case complexity
	  o Sample input/output showing sorted scores	



Task 2: Improving Sorting for Nearly Sorted Attendance Records

Scenario

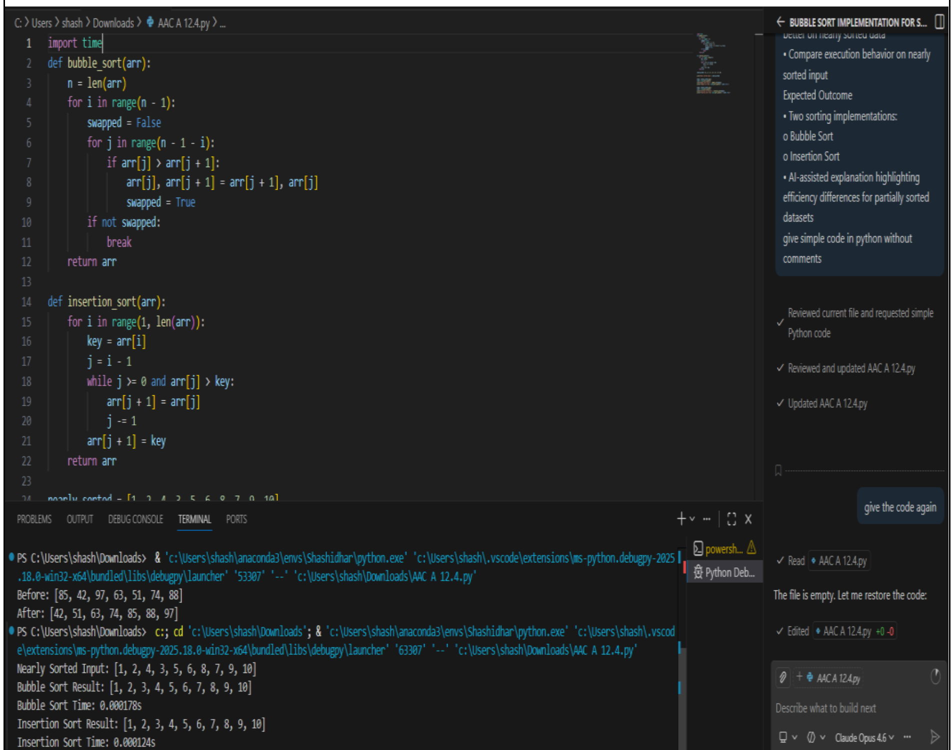
You are maintaining an **attendance system** where student roll numbers are already *almost sorted*, with only a few late updates.

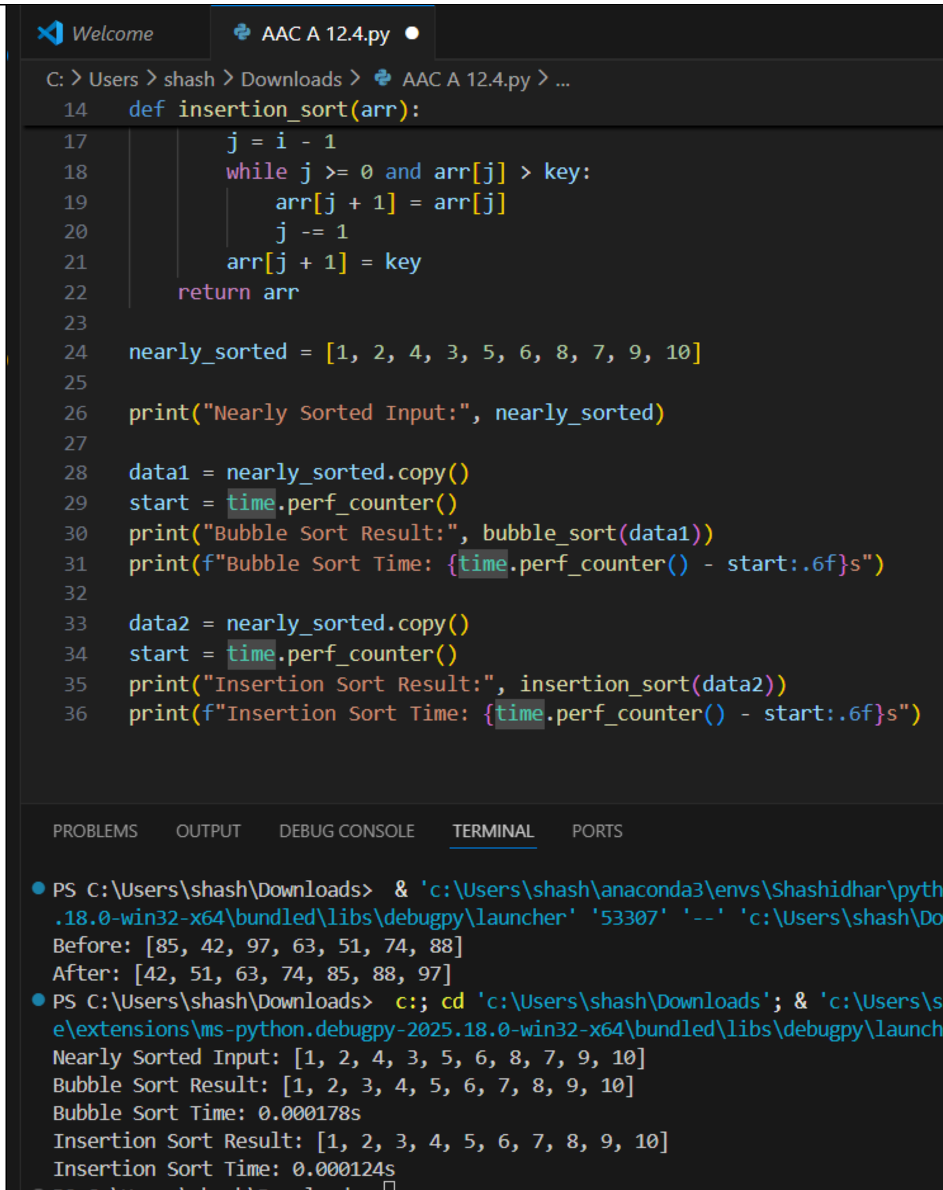
Task Description

- Start with a Bubble Sort implementation.
- Ask AI to:
  - Review the problem and suggest a more suitable sorting algorithm
  - Generate an **Insertion Sort** implementation
  - Explain why Insertion Sort performs better on nearly sorted data
- Compare execution behavior on nearly sorted input

Expected Outcome

- Two sorting implementations:
  - Bubble Sort
  - Insertion Sort
- AI-assisted explanation highlighting efficiency differences for partially sorted datasets





```
Welcome    AAC A 12.4.py
C: > Users > shash > Downloads > AAC A 12.4.py > ...
def insertion_sort(arr):
            j = i - 1
            while j >= 0 and arr[j] > key:
                arr[j + 1] = arr[j]
                j -= 1
            arr[j + 1] = key
        return arr

nearly_sorted = [1, 2, 4, 3, 5, 6, 8, 7, 9, 10]

print("Nearly Sorted Input:", nearly_sorted)

data1 = nearly_sorted.copy()
start = time.perf_counter()
print("Bubble Sort Result:", bubble_sort(data1))
print(f"Bubble Sort Time: {time.perf_counter() - start:.6f}s")

data2 = nearly_sorted.copy()
start = time.perf_counter()
print("Insertion Sort Result:", insertion_sort(data2))
print(f"Insertion Sort Time: {time.perf_counter() - start:.6f}s")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\shash\Downloads>  & 'c:\Users\shash\anaconda3\envs\Shashidhar\pytho
.18.0-win32-x64\bundled\libs\debugpy\launcher' '53307' '--' 'c:\Users\shash\Dow
Before: [85, 42, 97, 63, 51, 74, 88]
After: [42, 51, 63, 74, 85, 88, 97]
PS C:\Users\shash\Downloads>  c:; cd 'c:\Users\shash\Downloads'; & 'c:\Users\sh
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launche
Nearly Sorted Input: [1, 2, 4, 3, 5, 6, 8, 7, 9, 10]
Bubble Sort Result: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Bubble Sort Time: 0.000178s
Insertion Sort Result: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Insertion Sort Time: 0.000124s
PS C:\Users\shash\Downloads>
```

Task 3: Searching Student Records in a Database

Scenario

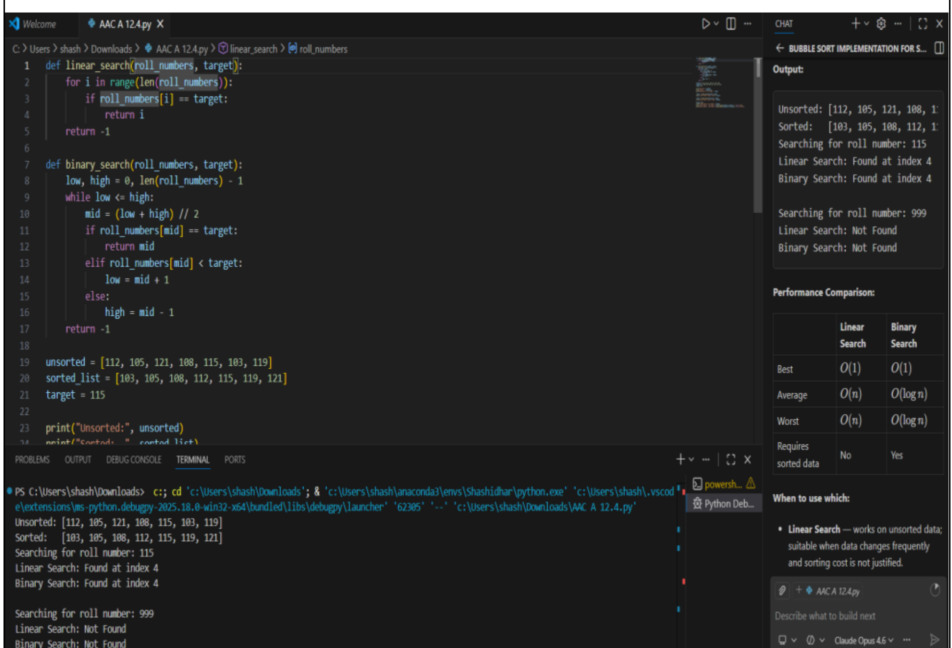
You are developing a **student information portal** where users search for student records by roll number.

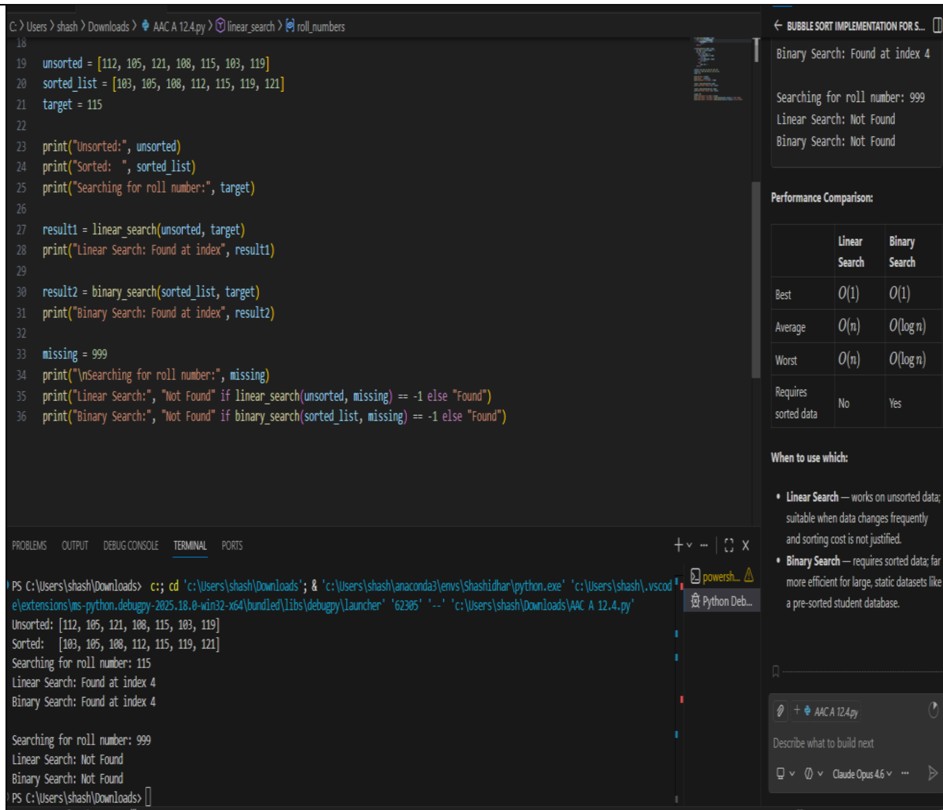
Task Description

- Implement:
    - **Linear Search** for unsorted student data
    - **Binary Search** for sorted student data
- Use AI to:
    - Add docstrings explaining parameters and return values
    - Explain when Binary Search is applicable
    - Highlight performance differences between the two searches

Expected Outcome

- Two working search implementations with docstrings
- AI-generated explanation of:
    - Time complexity
    - Use cases for Linear vs Binary Search
- A short student observation comparing results on sorted vs unsorted lists





Task 4: Choosing Between Quick Sort and Merge Sort for Data Processing

Scenario

You are part of a **data analytics team** that needs to sort large datasets received from different sources (random order, already sorted, and reverse sorted).

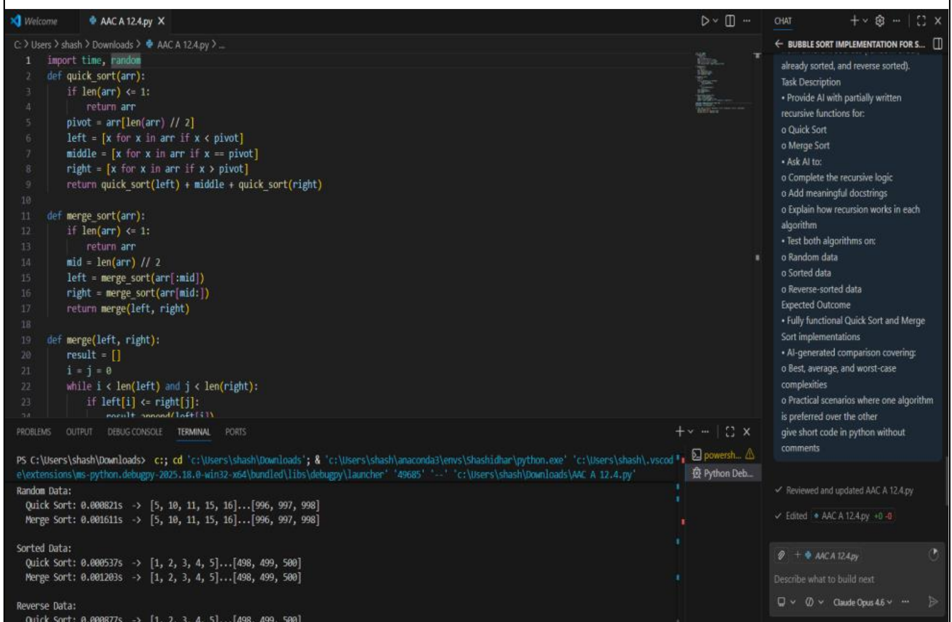
Task Description

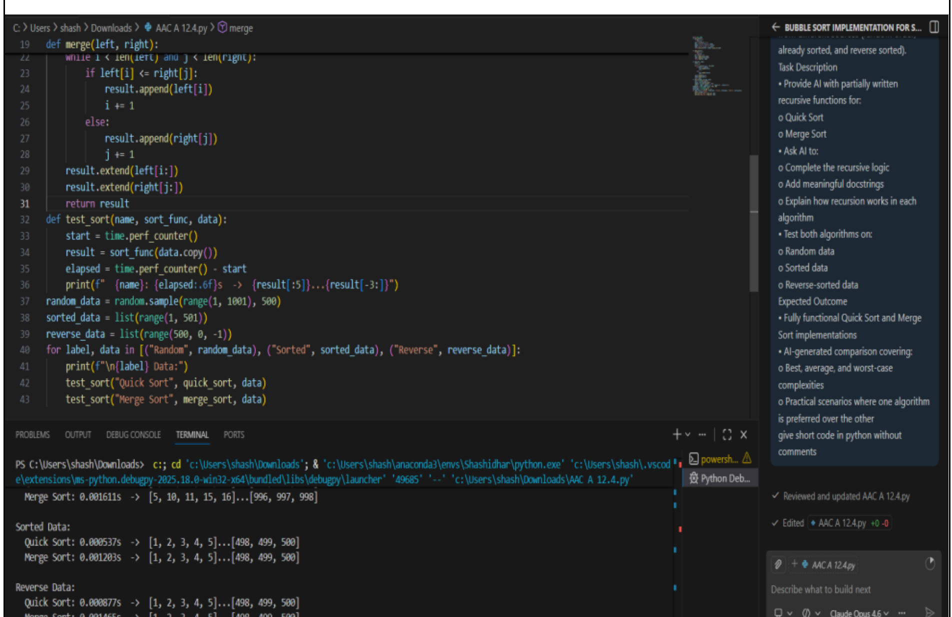
- Provide AI with partially written recursive functions for:
  - **Quick Sort**
  - **Merge Sort**
- Ask AI to:
  - Complete the recursive logic
  - Add meaningful docstrings
  - Explain how recursion works in each algorithm
- Test both algorithms on:
  - Random data

- Sorted data
- Reverse-sorted data

Expected Outcome

- Fully functional Quick Sort and Merge Sort implementations
- AI-generated comparison covering:
  - Best, average, and worst-case complexities
  - Practical scenarios where one algorithm is preferred over the other





Task 5: Optimizing a Duplicate Detection Algorithm

Scenario

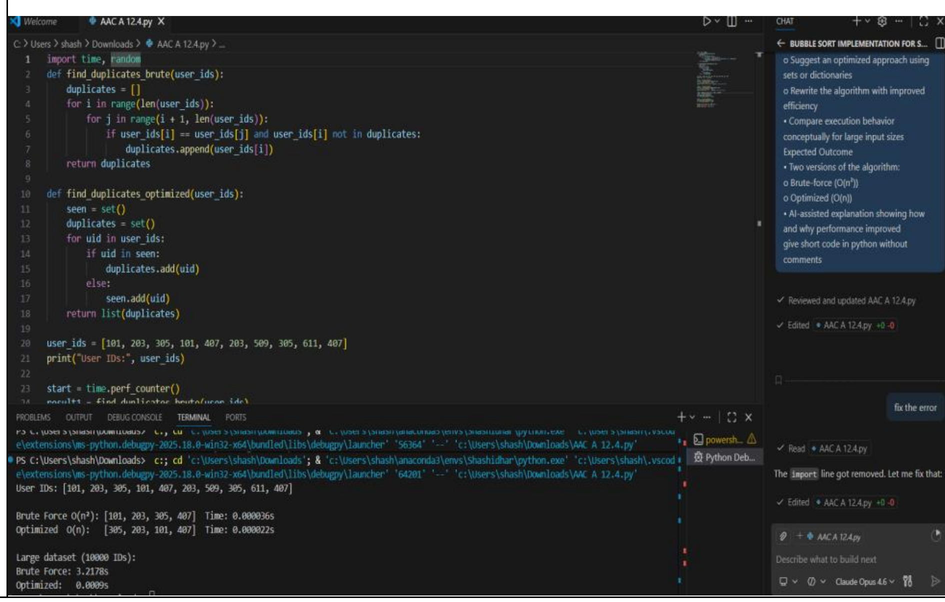
You are building a **data validation module** that must detect duplicate user IDs in a large dataset before importing it into a system.

Task Description

- Write a **naive duplicate detection algorithm** using nested loops.
- Use AI to:
    - Analyze the time complexity
    - Suggest an optimized approach using sets or dictionaries
    - Rewrite the algorithm with improved efficiency
- Compare execution behavior conceptually for large input sizes

Expected Outcome

- Two versions of the algorithm:
    - Brute-force ($O(n^2)$)
    - Optimized ($O(n)$)
- AI-assisted explanation showing how and why performance improved



Welcome | AAC A 12.4.py

C: > Users > shash > Downloads > AAC A 12.4.py > ...

```python

user_ids = [101, 203, 305, 101, 407, 203, 509, 305, 611, 407]
print("User IDs:", user_ids)

start = time.perf_counter()
result1 = find_duplicates_brute(user_ids)
t1 = time.perf_counter() - start
print(f"\nBrute Force O(n²): {result1}  Time: {t1:.6f}s")

start = time.perf_counter()
result2 = find_duplicates_optimized(user_ids)
t2 = time.perf_counter() - start
print(f"Optimized  O(n):    {result2}  Time: {t2:.6f}s")
large = random.choices(range(1, 5001), k=10000)
start = time.perf_counter()
find_duplicates_brute(large)
t1 = time.perf_counter() - start
start = time.perf_counter()
find_duplicates_optimized(large)
t2 = time.perf_counter() - start
print(f"\nLarge dataset (10000 IDs):")
print(f"Brute Force: {t1:.4f}s")
print(f"Optimized:   {t2:.4f}s")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '5636
PS C:\Users\shash\Downloads>  c:; cd 'c:\Users\shash\Downloads'; & 'c:\Users\shash\anac
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '6420
User IDs: [101, 203, 305, 101, 407, 203, 509, 305, 611, 407]

Brute Force O(n²): [101, 203, 305, 407]  Time: 0.000036s
Optimized  O(n):   [305, 203, 101, 407]  Time: 0.000022s

Large dataset (10000 IDs):
Brute Force: 3.2178s
Optimized:   0.0009s
PS C:\Users\shash\Downloads>
```

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