Assignment 12.4 Ai Assisted Coding

Htno:2303a51305

Btno:05

Task 1:

Bubble Sort for Ranking Exam

Scores

Scenario

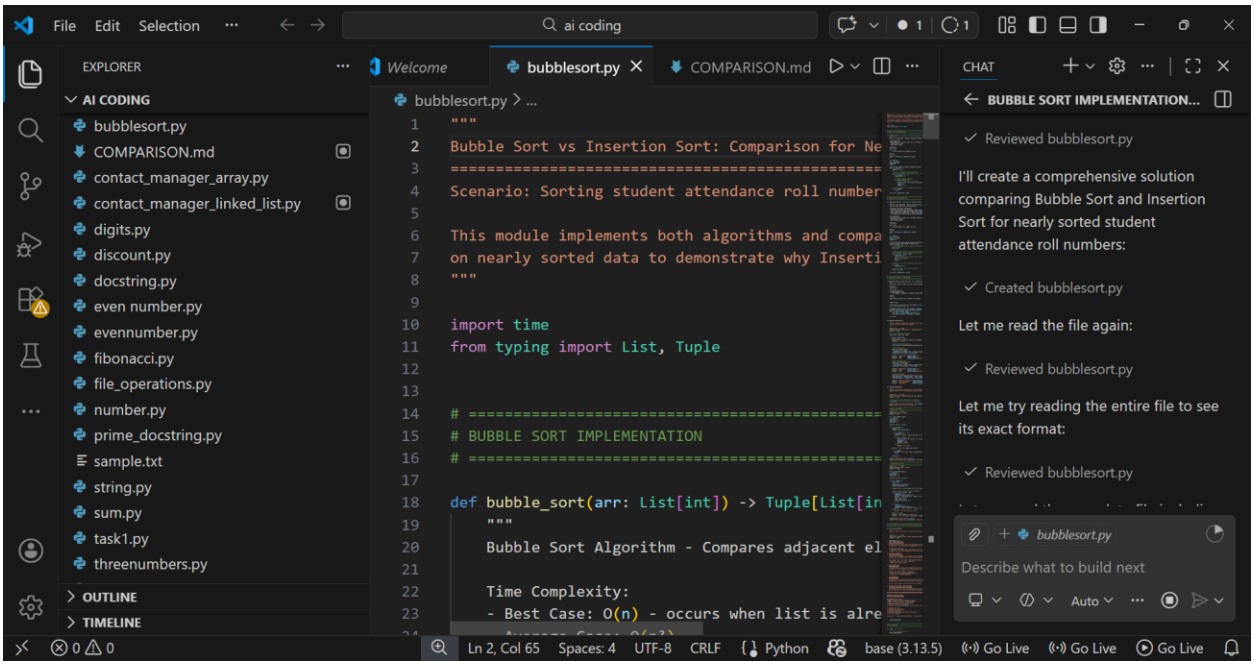
You are working on a college result processing system where a small

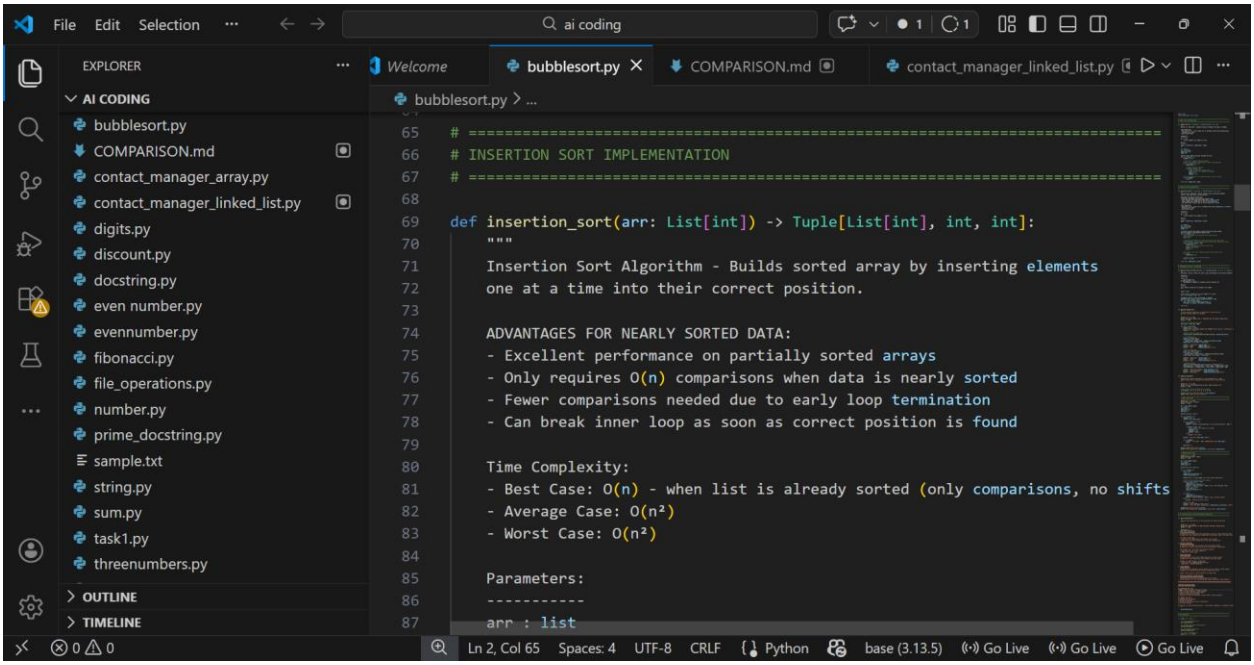
list of student scores needs to be sorted after every internal assessment.

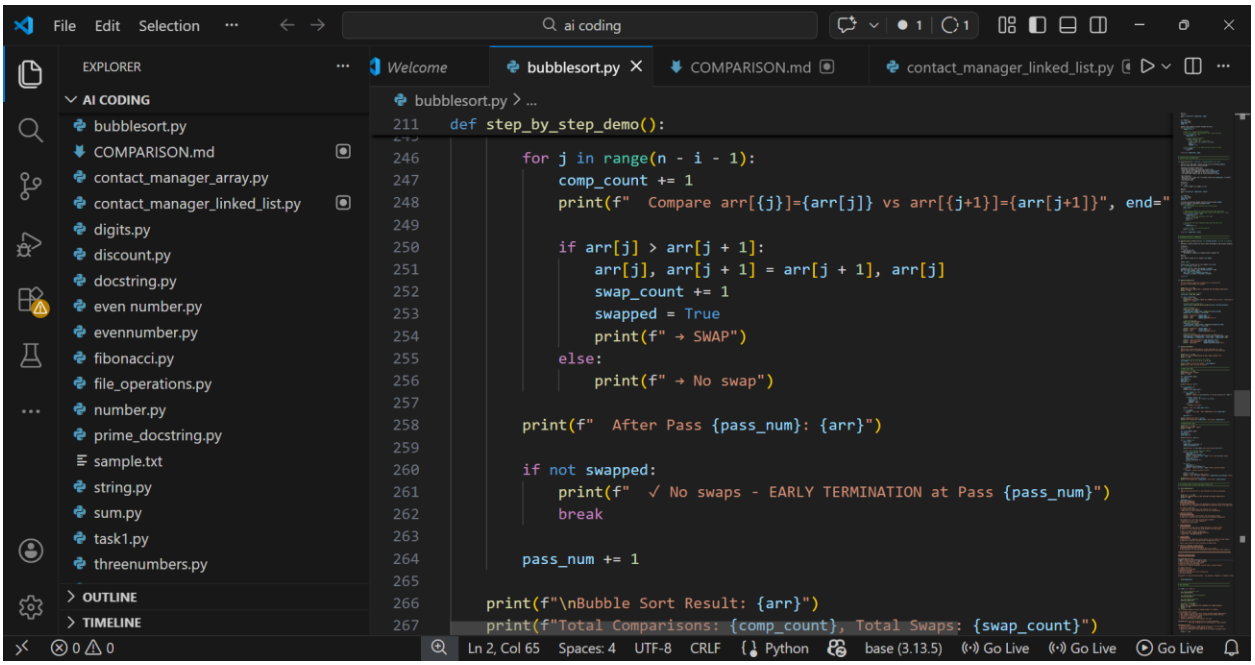
Prompt:

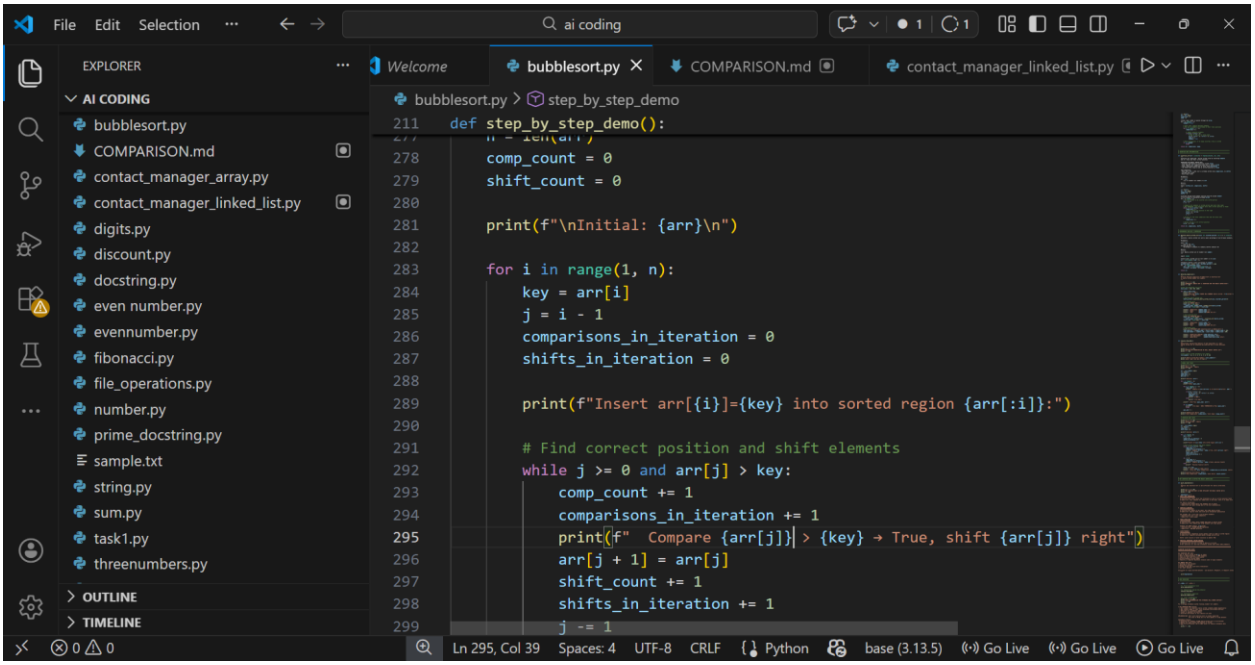
Write a simple Python program for Bubble Sort to sort student exam scores. Add inline comments for comparisons and swaps, include early termination if already sorted, give time complexity (best, average, worst), and show sample input/output.

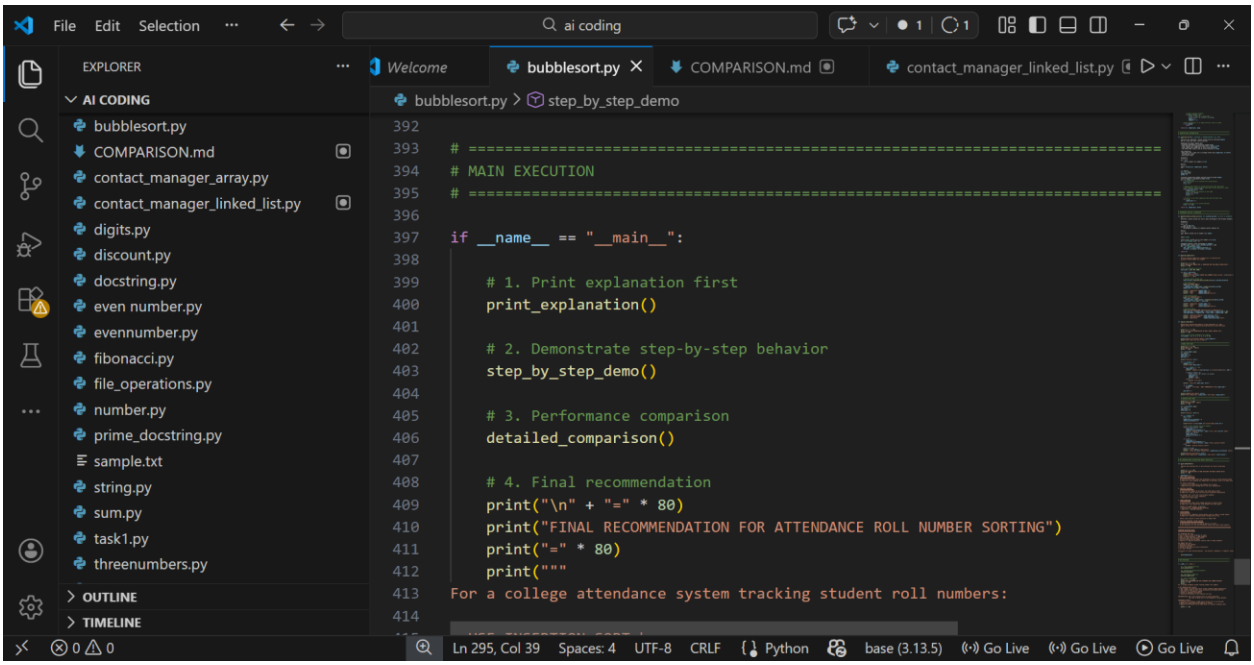
Code:


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
bubblesort.py > ...
"""
Bubble Sort vs Insertion Sort: Comparison for Ne
Scenario: Sorting student attendance roll number
This module implements both algorithms and compa
on nearly sorted data to demonstrate why Inserti
"""
import time
from typing import List, Tuple
# BUBBLE SORT IMPLEMENTATION
def bubble_sort(arr: List[int]) -> Tuple[List[in
"""
Bubble Sort Algorithm - Compares adjacent el
Time Complexity:
- Best Case: O(n) - occurs when list is alre
CHAT
BUBBLE SORT IMPLEMENTATION...
Reviewed bubblesort.py
I'll create a comprehensive solution comparing Bubble Sort and Insertion Sort for nearly sorted student attendance roll numbers:
Created bubblesort.py
Let me read the file again:
Reviewed bubblesort.py
Let me try reading the entire file to see its exact format:
Reviewed bubblesort.py
bubblesort.py
Describe what to build next
Auto
Ln 2, Col 65 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

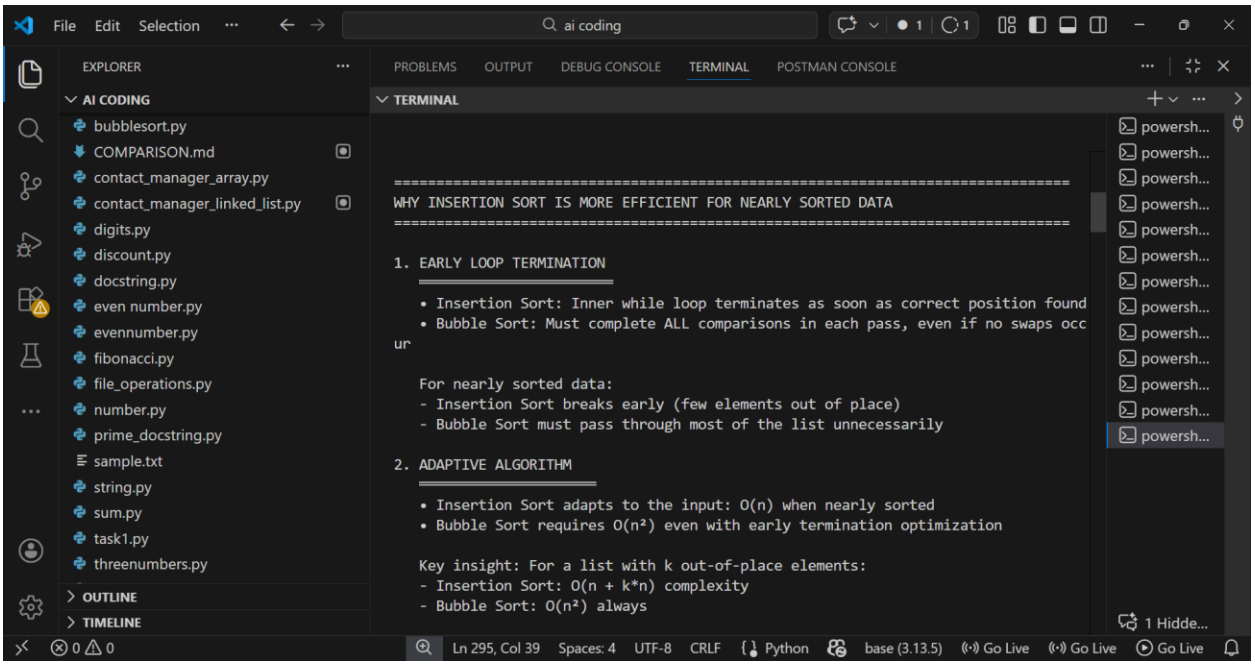

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
# INSERTION SORT IMPLEMENTATION
def insertion_sort(arr: List[int]) -> Tuple[List[int], int, int]:
"""
Insertion Sort Algorithm - Builds sorted array by inserting elements
one at a time into their correct position.
ADVANTAGES FOR NEARLY SORTED DATA:
- Excellent performance on partially sorted arrays
- Only requires O(n) comparisons when data is nearly sorted
- Fewer comparisons needed due to early loop termination
- Can break inner loop as soon as correct position is found
Time Complexity:
- Best Case: O(n) - when list is already sorted (only comparisons, no shifts
- Average Case: O(n²)
- Worst Case: O(n²)
Parameters:
arr : list
Ln 2, Col 65 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

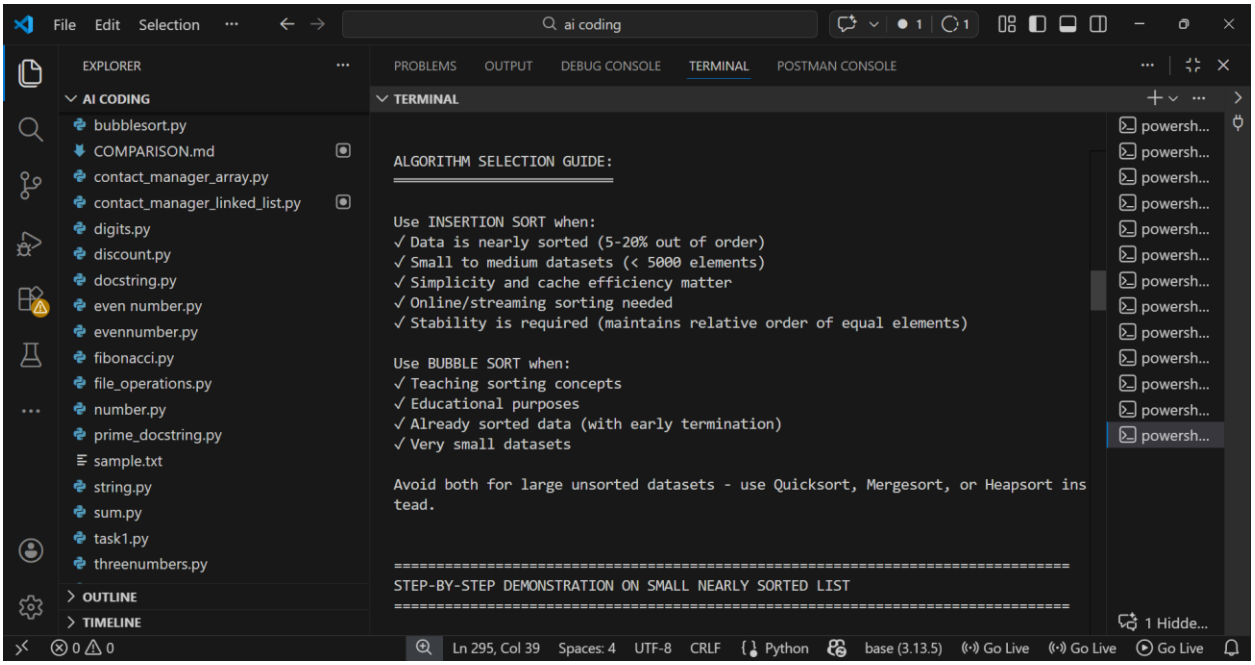

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
211 def step_by_step_demo():
246 for j in range(n - i - 1):
247 comp_count += 1
248 print(f" Compare arr[{j}]={arr[j]} vs arr[{j+1}]={arr[j+1]}", end="
249
250 if arr[j] > arr[j + 1]:
251 arr[j], arr[j + 1] = arr[j + 1], arr[j]
252 swap_count += 1
253 swapped = True
254 print(f" → SWAP")
255 else:
256 print(f" → No swap")
257
258 print(f" After Pass {pass_num}: {arr}")
259
260 if not swapped:
261 print(f" ✓ No swaps - EARLY TERMINATION at Pass {pass_num}")
262 break
263
264 pass_num += 1
265
266 print(f"\nBubble Sort Result: {arr}")
267 print(f"Total Comparisons: {comp_count}, Total Swaps: {swap_count}")
Ln 2, Col 65 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

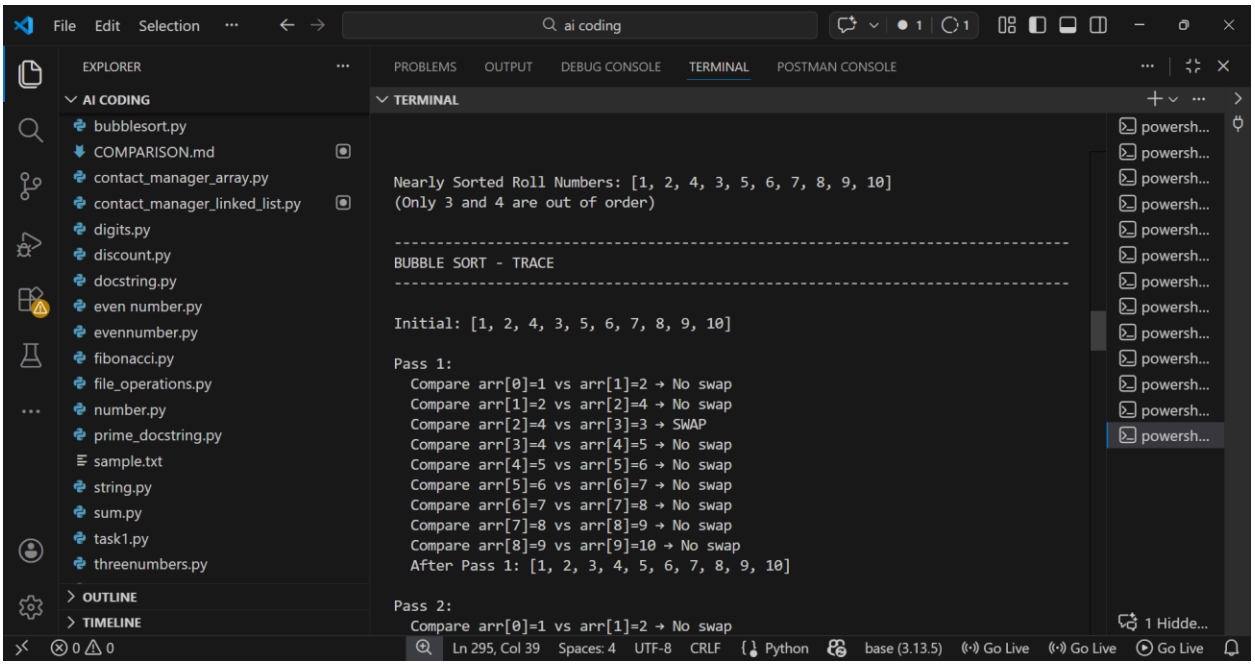

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > step_by_step_demo
211 def step_by_step_demo():
278 comp_count = 0
279 shift_count = 0
280
281 print(f"\nInitial: {arr}\n")
282
283 for i in range(1, n):
284 key = arr[i]
285 j = i - 1
286 comparisons_in_iteration = 0
287 shifts_in_iteration = 0
288
289 print(f"Insert arr[{i}]={key} into sorted region {arr[:i]}:")
290
291 # Find correct position and shift elements
292 while j >= 0 and arr[j] > key:
293 comp_count += 1
294 comparisons_in_iteration += 1
295 print(f" Compare {arr[j]} > {key} → True, shift {arr[j]} right")
296 arr[j + 1] = arr[j]
297 shift_count += 1
298 shifts_in_iteration += 1
299 j -= 1
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

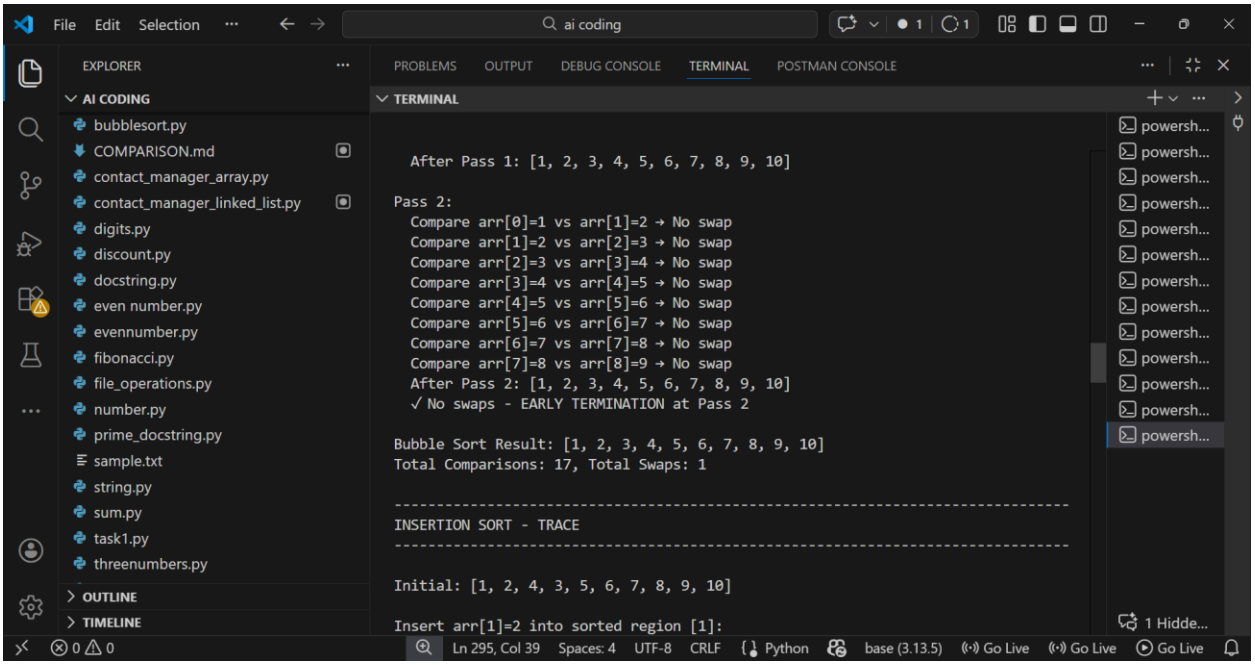

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > step_by_step_demo
# ==
# MAIN EXECUTION
# ==
if __name__ == "__main__":
# 1. Print explanation first
print_explanation()
# 2. Demonstrate step-by-step behavior
step_by_step_demo()
# 3. Performance comparison
detailed_comparison()
# 4. Final recommendation
print("\n" + "=" * 80)
print("FINAL RECOMMENDATION FOR ATTENDANCE ROLL NUMBER SORTING")
print("=" * 80)
print("""
For a college attendance system tracking student roll numbers:
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

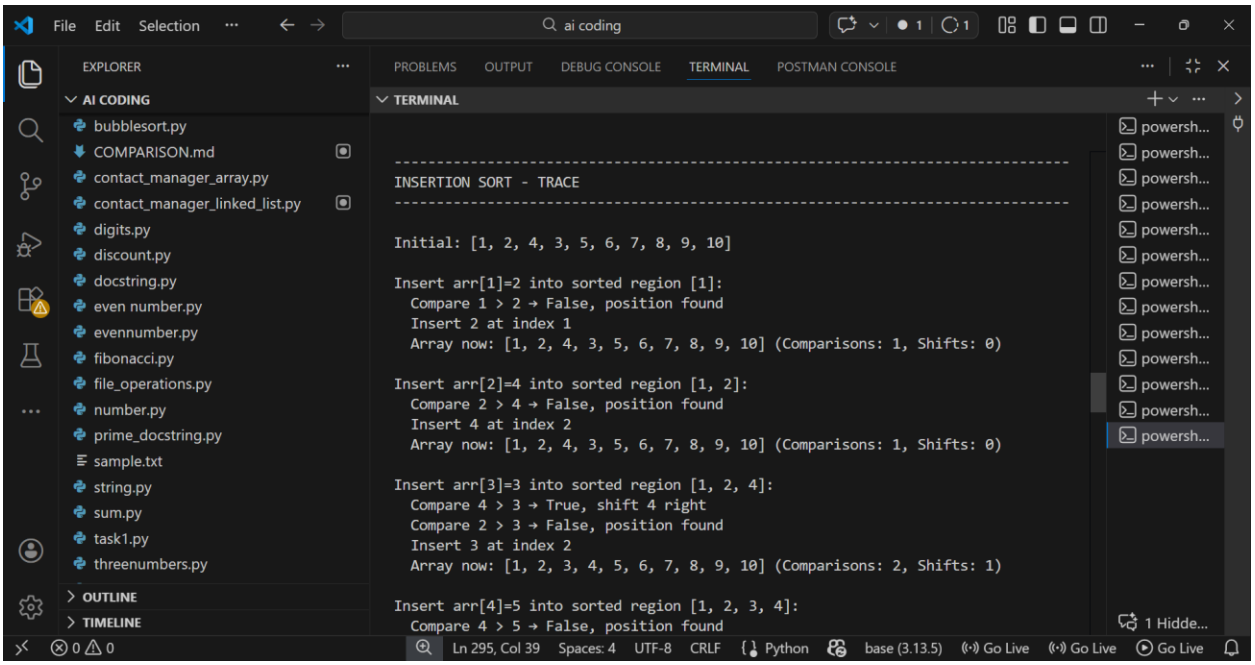
Output:

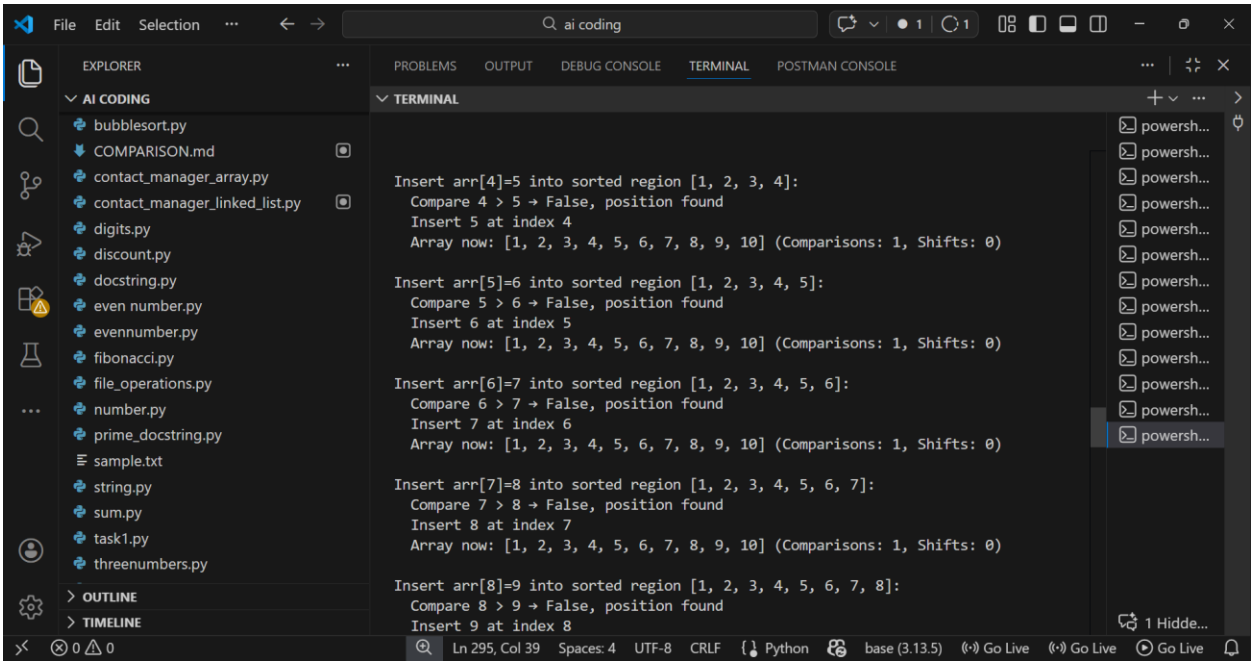

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
powersh...
WHY INSERTION SORT IS MORE EFFICIENT FOR NEARLY SORTED DATA
1. EARLY LOOP TERMINATION
• Insertion Sort: Inner while loop terminates as soon as correct position found
• Bubble Sort: Must complete ALL comparisons in each pass, even if no swaps occ
ur
For nearly sorted data:
- Insertion Sort breaks early (few elements out of place)
- Bubble Sort must pass through most of the list unnecessarily
2. ADAPTIVE ALGORITHM
• Insertion Sort adapts to the input: O(n) when nearly sorted
• Bubble Sort requires O(n²) even with early termination optimization
Key insight: For a list with k out-of-place elements:
- Insertion Sort: O(n + k*n) complexity
- Bubble Sort: O(n²) always
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live

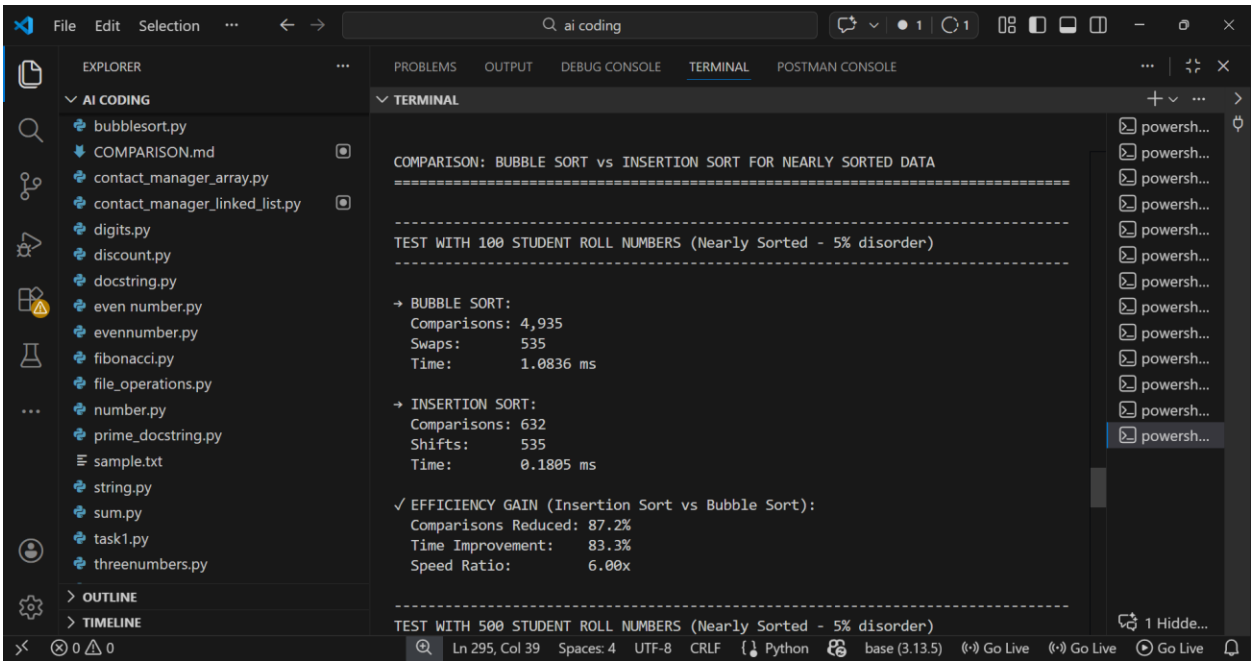

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
powersh...
ALGORITHM SELECTION GUIDE:
Use INSERTION SORT when:
✓ Data is nearly sorted (5-20% out of order)
✓ Small to medium datasets (< 5000 elements)
✓ Simplicity and cache efficiency matter
✓ Online/streaming sorting needed
✓ Stability is required (maintains relative order of equal elements)
Use BUBBLE SORT when:
✓ Teaching sorting concepts
✓ Educational purposes
✓ Already sorted data (with early termination)
✓ Very small datasets
Avoid both for large unsorted datasets - use Quicksort, Mergesort, or Heapsort ins
tead.
STEP-BY-STEP DEMONSTRATION ON SMALL NEARLY SORTED LIST
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
Nearly Sorted Roll Numbers: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10]
(Only 3 and 4 are out of order)
BUBBLE SORT - TRACE
Initial: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10]
Pass 1:
Compare arr[0]=1 vs arr[1]=2 → No swap
Compare arr[1]=2 vs arr[2]=4 → No swap
Compare arr[2]=4 vs arr[3]=3 → SWAP
Compare arr[3]=4 vs arr[4]=5 → No swap
Compare arr[4]=5 vs arr[5]=6 → No swap
Compare arr[5]=6 vs arr[6]=7 → No swap
Compare arr[6]=7 vs arr[7]=8 → No swap
Compare arr[7]=8 vs arr[8]=9 → No swap
Compare arr[8]=9 vs arr[9]=10 → No swap
After Pass 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Pass 2:
Compare arr[0]=1 vs arr[1]=2 → No swap
powersh...
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
After Pass 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Pass 2:
Compare arr[0]=1 vs arr[1]=2 → No swap
Compare arr[1]=2 vs arr[2]=3 → No swap
Compare arr[2]=3 vs arr[3]=4 → No swap
Compare arr[3]=4 vs arr[4]=5 → No swap
Compare arr[4]=5 vs arr[5]=6 → No swap
Compare arr[5]=6 vs arr[6]=7 → No swap
Compare arr[6]=7 vs arr[7]=8 → No swap
Compare arr[7]=8 vs arr[8]=9 → No swap
After Pass 2: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
✓ No swaps - EARLY TERMINATION at Pass 2
Bubble Sort Result: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Total Comparisons: 17, Total Swaps: 1
INSERTION SORT - TRACE
Initial: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10]
Insert arr[1]=2 into sorted region [1]:
powersh...
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
INSERTION SORT - TRACE
Initial: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10]
Insert arr[1]=2 into sorted region [1]:
Compare 1 > 2 → False, position found
Insert 2 at index 1
Array now: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[2]=4 into sorted region [1, 2]:
Compare 2 > 4 → False, position found
Insert 4 at index 2
Array now: [1, 2, 4, 3, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[3]=3 into sorted region [1, 2, 4]:
Compare 4 > 3 → True, shift 4 right
Compare 2 > 3 → False, position found
Insert 3 at index 2
Array now: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (Comparisons: 2, Shifts: 1)
Insert arr[4]=5 into sorted region [1, 2, 3, 4]:
Compare 4 > 5 → False, position found
powersh...
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POSTMAN CONSOLE
TERMINAL
Insert arr[4]=5 into sorted region [1, 2, 3, 4]:
Compare 4 > 5 → False, position found
Insert 5 at index 4
Array now: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[5]=6 into sorted region [1, 2, 3, 4, 5]:
Compare 5 > 6 → False, position found
Insert 6 at index 5
Array now: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[6]=7 into sorted region [1, 2, 3, 4, 5, 6]:
Compare 6 > 7 → False, position found
Insert 7 at index 6
Array now: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[7]=8 into sorted region [1, 2, 3, 4, 5, 6, 7]:
Compare 7 > 8 → False, position found
Insert 8 at index 7
Array now: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (Comparisons: 1, Shifts: 0)
Insert arr[8]=9 into sorted region [1, 2, 3, 4, 5, 6, 7, 8]:
Compare 8 > 9 → False, position found
Insert 9 at index 8
powersh...
1 Hidde...
Ln 295, Col 39 Spaces: 4 UTF-8 CRLF Python base (3.13.5) Go Live Go Live Go Live



Task 3:

Searching Student Records in a

Database

Scenario

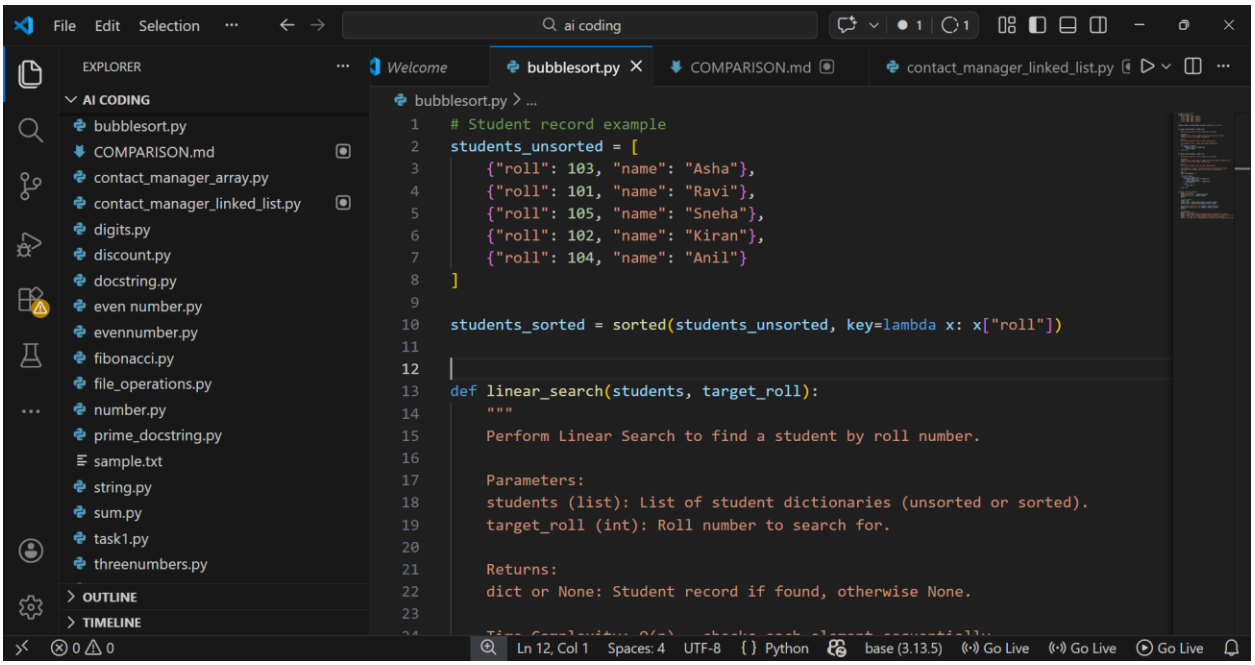
You are developing a student information portal where users search for

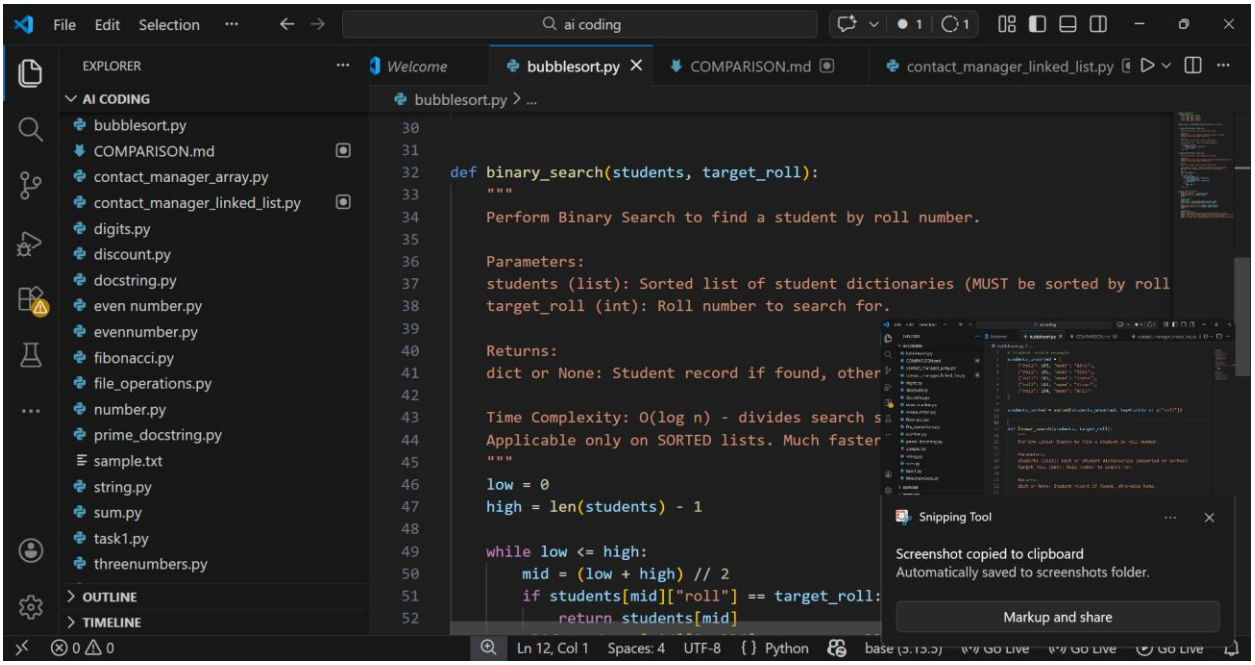
student records by roll number.

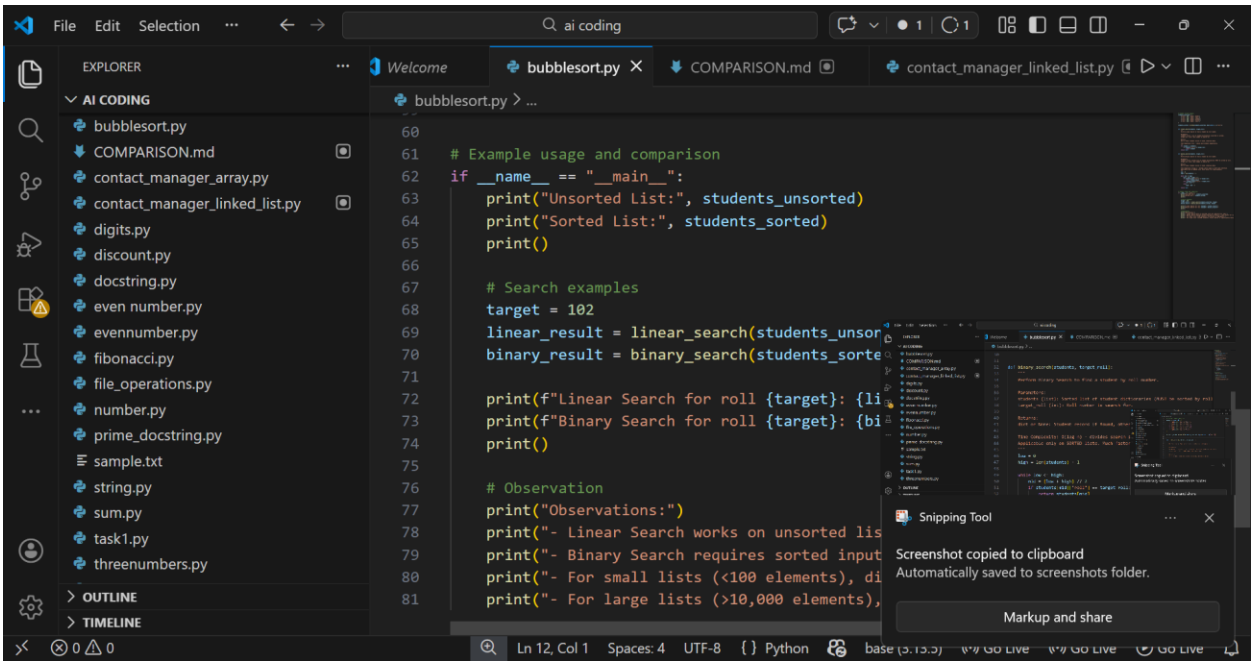
Prompt:

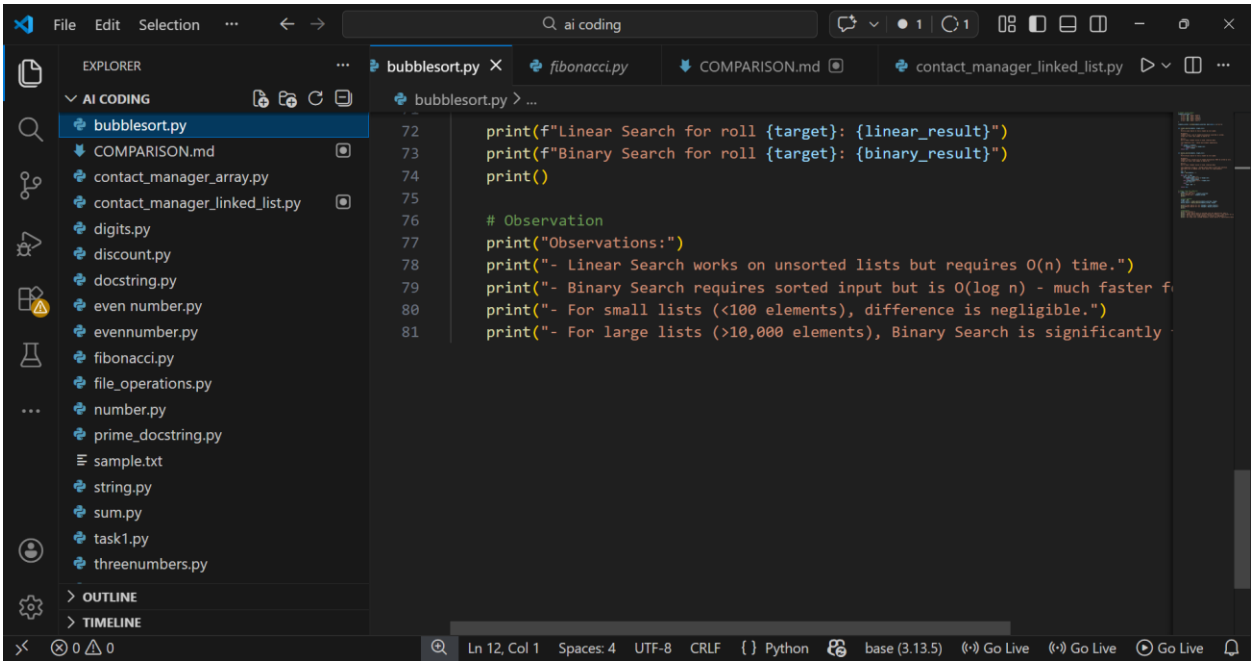
Implement Linear Search and Binary Search in Python to search student records by roll number. Add proper docstrings explaining parameters and return values. Explain when Binary Search is applicable and compare performance differences. Also provide time complexity and a short observation comparing results on sorted vs unsorted lists.

Code:

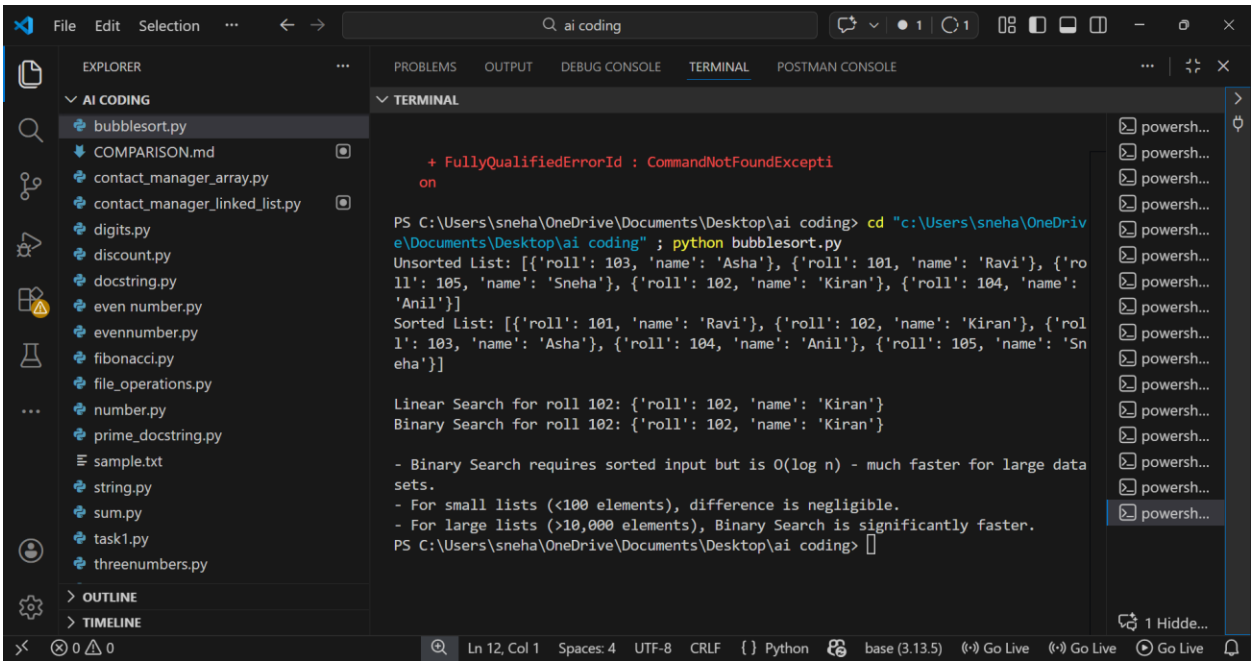

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
# Student record example
students_unsorted = [
{"roll": 103, "name": "Asha"},
{"roll": 101, "name": "Ravi"},
{"roll": 105, "name": "Sneha"},
{"roll": 102, "name": "Kiran"},
{"roll": 104, "name": "Anil"}
]
students_sorted = sorted(students_unsorted, key=lambda x: x["roll"])
def linear_search(students, target_roll):
"""
Perform Linear Search to find a student by roll number.
Parameters:
students (list): List of student dictionaries (unsorted or sorted).
target_roll (int): Roll number to search for.
Returns:
dict or None: Student record if found, otherwise None.
Ln 12, Col 1
Spaces: 4
UTF-8
Python
base (3.13.5)
Go Live
Go Live
Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
def binary_search(students, target_roll):
"""
Perform Binary Search to find a student by roll number.
Parameters:
students (list): Sorted list of student dictionaries (MUST be sorted by roll
target_roll (int): Roll number to search for.
Returns:
dict or None: Student record if found, other
Time Complexity: O(log n) - divides search s
Applicable only on SORTED lists. Much faster
"""
low = 0
high = len(students) - 1
while low <= high:
mid = (low + high) // 2
if students[mid]["roll"] == target_roll:
return students[mid]
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Markup and share
Ln 12, Col 1
Spaces: 4
UTF-8
Python


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
60
61 # Example usage and comparison
62 if __name__ == "__main__":
63 print("Unsorted List:", students_unsorted)
64 print("Sorted List:", students_sorted)
65 print()
66
67 # Search examples
68 target = 102
69 linear_result = linear_search(students_unsor
70 binary_result = binary_search(students_sorte
71
72 print(f"Linear Search for roll {target}: {li
73 print(f"Binary Search for roll {target}: {bi
74 print()
75
76 # Observation
77 print("Observations:")
78 print("- Linear Search works on unsorted lis
79 print("- Binary Search requires sorted input
80 print("- For small lists (<100 elements), di
81 print("- For large lists (>10,000 elements),
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Markup and share
Ln 12, Col 1 Spaces: 4 UTF-8 { } Python


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
72 print(f"Linear Search for roll {target}: {linear_result}")
73 print(f"Binary Search for roll {target}: {binary_result}")
74 print()
75
76 # Observation
77 print("Observations:")
78 print("- Linear Search works on unsorted lists but requires O(n) time.")
79 print("- Binary Search requires sorted input but is O(log n) - much faster f
80 print("- For small lists (<100 elements), difference is negligible.")
81 print("- For large lists (>10,000 elements), Binary Search is significantly
Ln 12, Col 1 Spaces: 4 UTF-8 CRLF { } Python base (3.13.5) Go Live Go Live Go Live

Output:



Explanation:

- **Linear Search** checks each student record one by one, so it works even if the data is unsorted.
- **Binary Search** divides the list into halves repeatedly, but it only works when the student records are sorted by roll number.
- Linear Search has **O(n) time complexity**, meaning it may take longer as the number of records increases.
- Binary Search has **O(log n) time complexity**, making it much faster for large datasets.
- Use Linear Search for small or unsorted data, and Binary Search for large sorted databases where quick searching is needed.

Task 4:

Choosing Between Quick Sort and

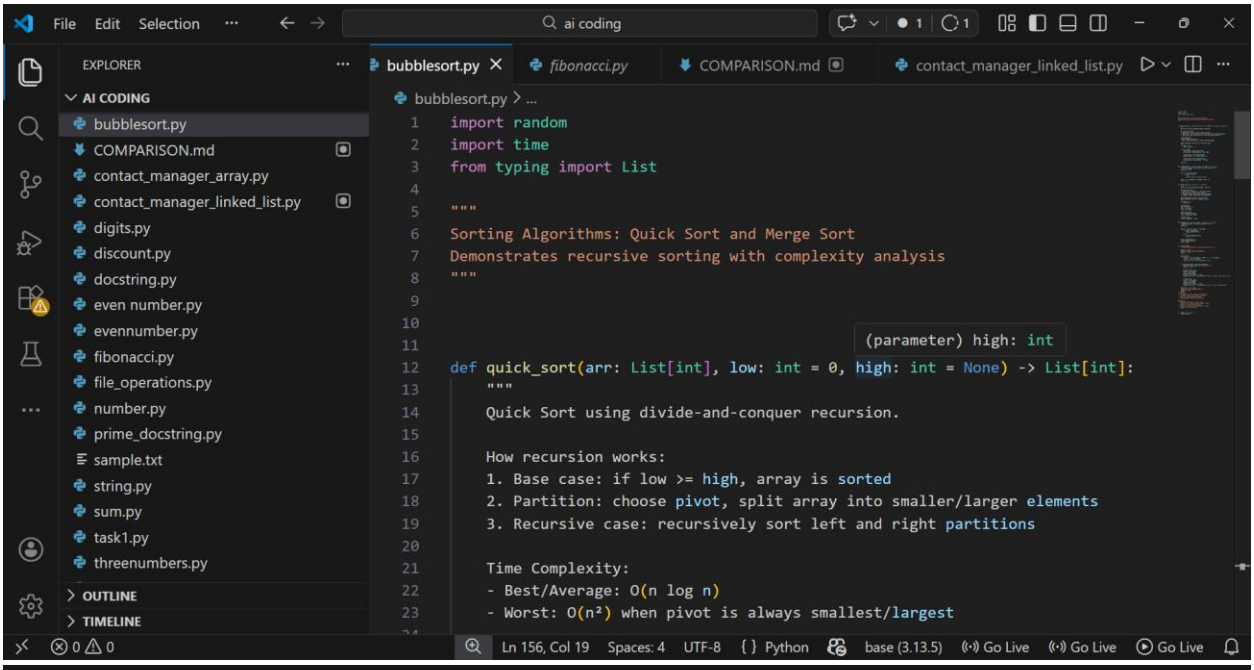
Merge Sort for Data Processing

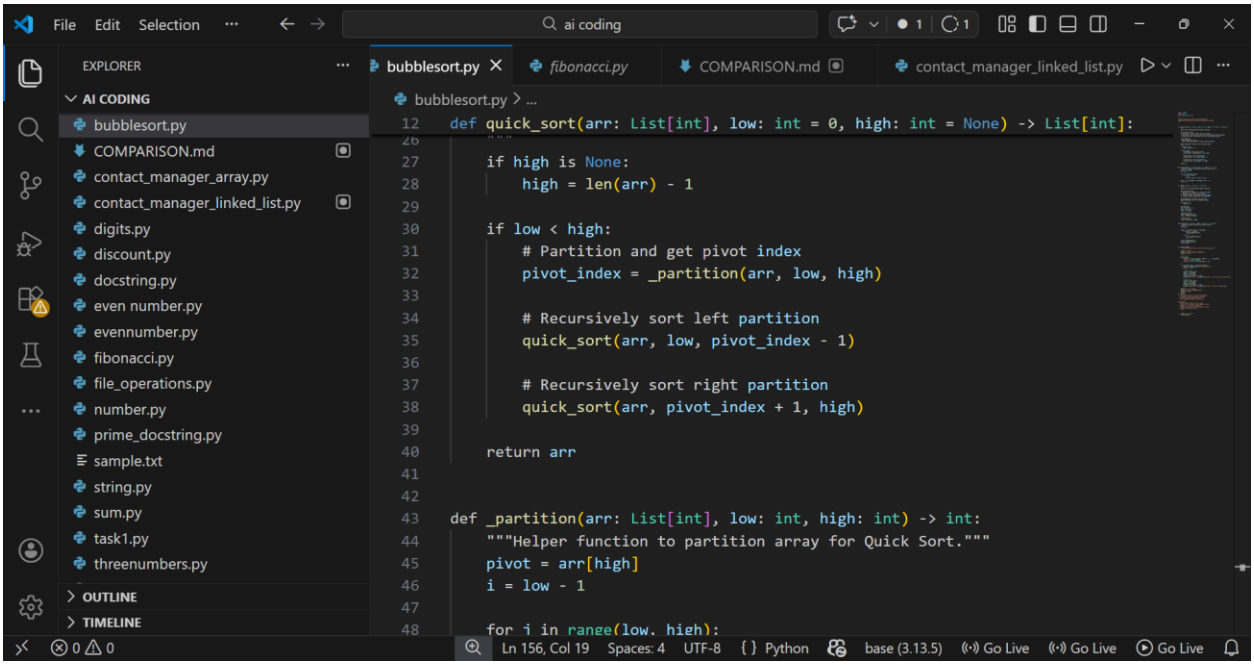
Scenario: You are part of a data analytics team that needs to sort large datasetsreceived from different sources (random order, already sorted, and reverse sorted).

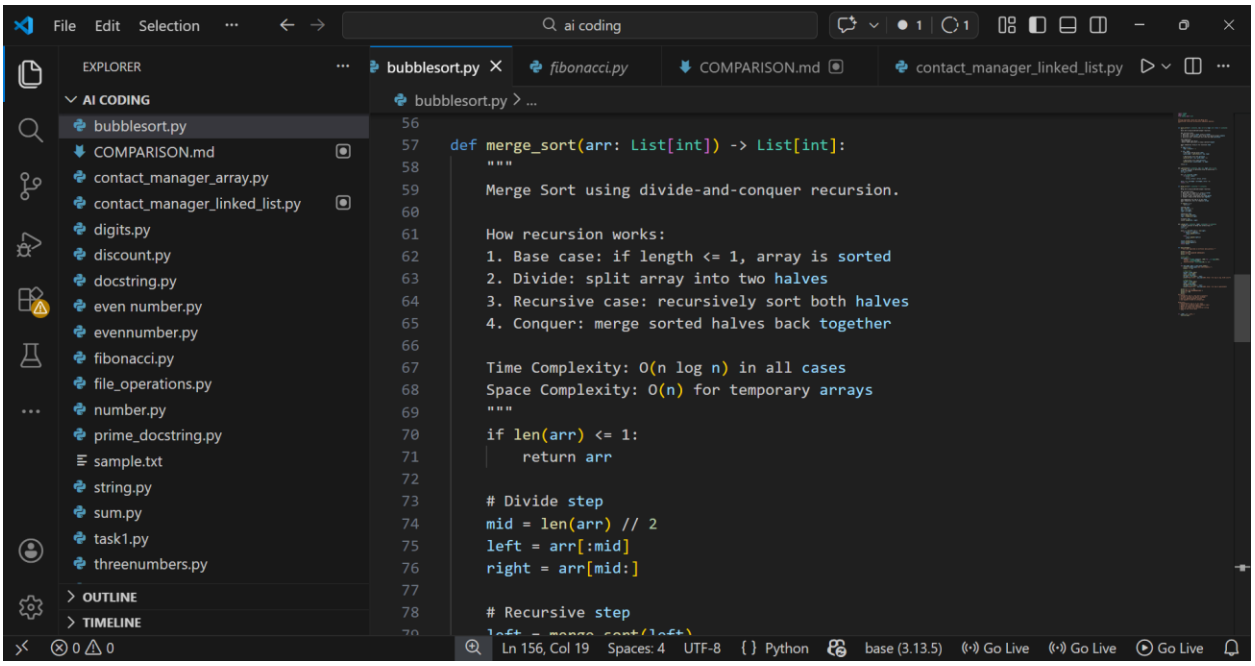
Prompt:

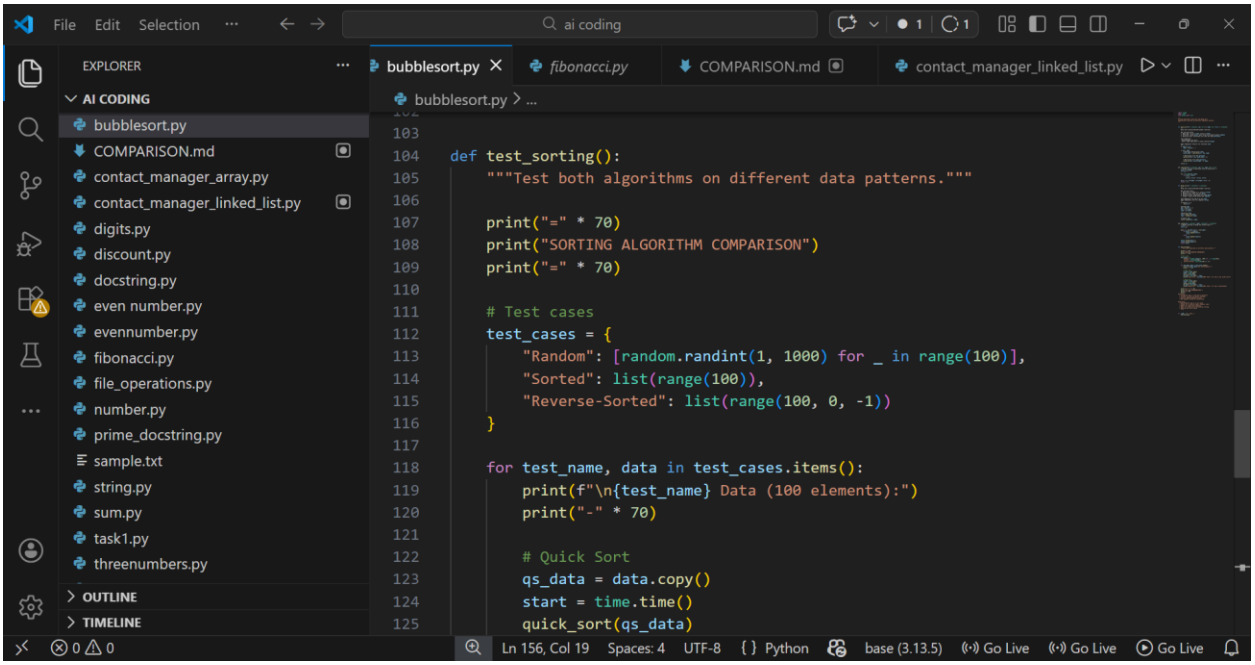
Complete recursive Quick Sort and Merge Sort functions with docstrings. Explain how recursion works, test on random, sorted, and reverse-sorted data, and compare time complexities with practical use cases.

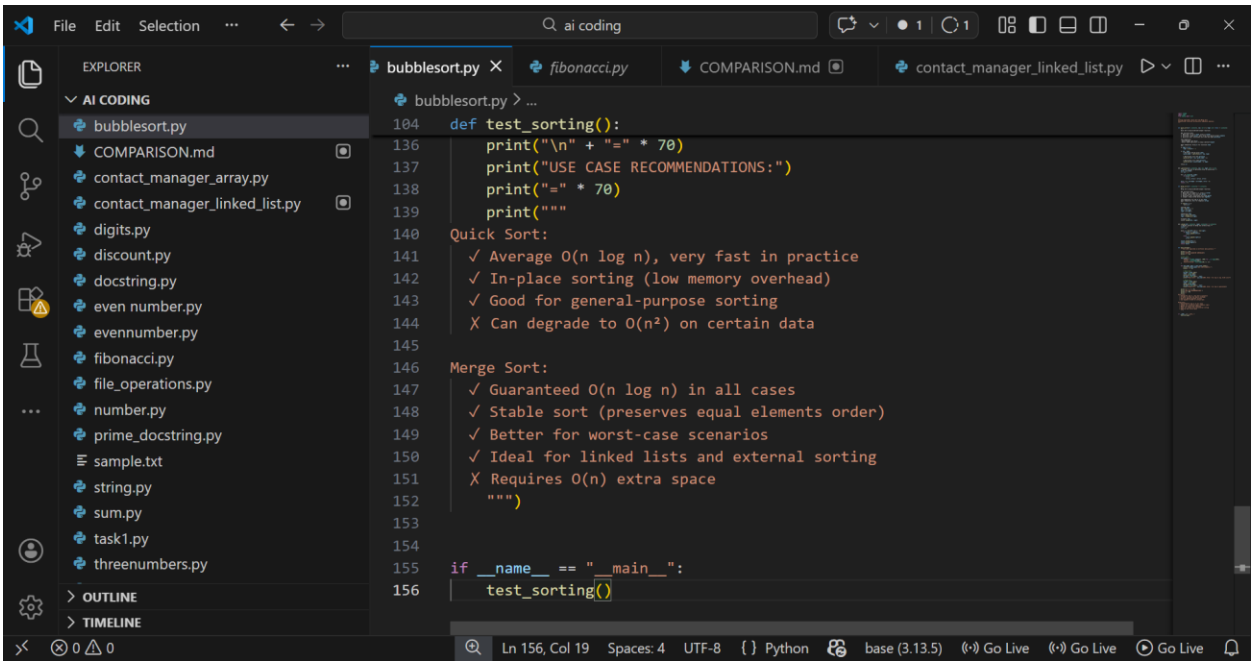
Code:


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
import random
import time
from typing import List
"""
Sorting Algorithms: Quick Sort and Merge Sort
Demonstrates recursive sorting with complexity analysis
"""
(parameter) high: int
def quick_sort(arr: List[int], low: int = 0, high: int = None) -> List[int]:
"""
Quick Sort using divide-and-conquer recursion.
How recursion works:
1. Base case: if low >= high, array is sorted
2. Partition: choose pivot, split array into smaller/larger elements
3. Recursive case: recursively sort left and right partitions
Time Complexity:
- Best/Average: O(n log n)
- Worst: O(n²) when pivot is always smallest/largest
Ln 156, Col 19 Spaces: 4 UTF-8 { } Python base (3.13.5) Go Live Go Live Go Live

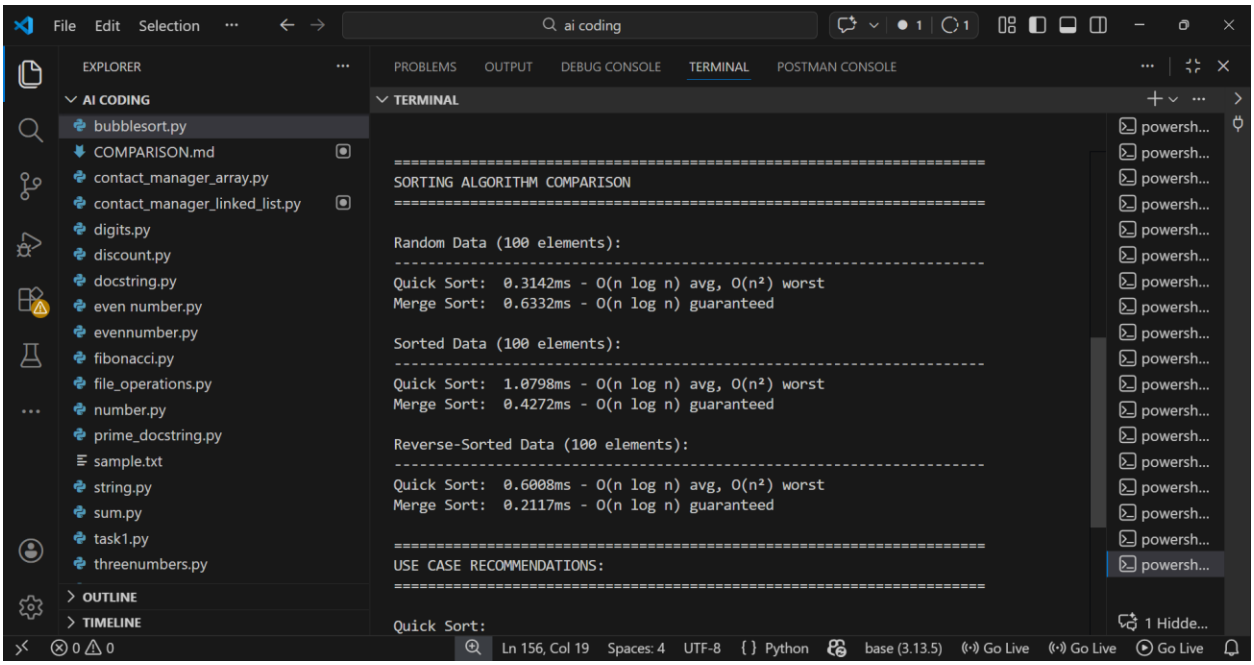

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
def quick_sort(arr: List[int], low: int = 0, high: int = None) -> List[int]:
if high is None:
high = len(arr) - 1
if low < high:
# Partition and get pivot index
pivot_index = _partition(arr, low, high)
# Recursively sort left partition
quick_sort(arr, low, pivot_index - 1)
# Recursively sort right partition
quick_sort(arr, pivot_index + 1, high)
return arr
def _partition(arr: List[int], low: int, high: int) -> int:
"""Helper function to partition array for Quick Sort."""
pivot = arr[high]
i = low - 1
for i in range(low, high):
Ln 156, Col 19 Spaces: 4 UTF-8 { } Python base (3.13.5) Go Live Go Live Go Live

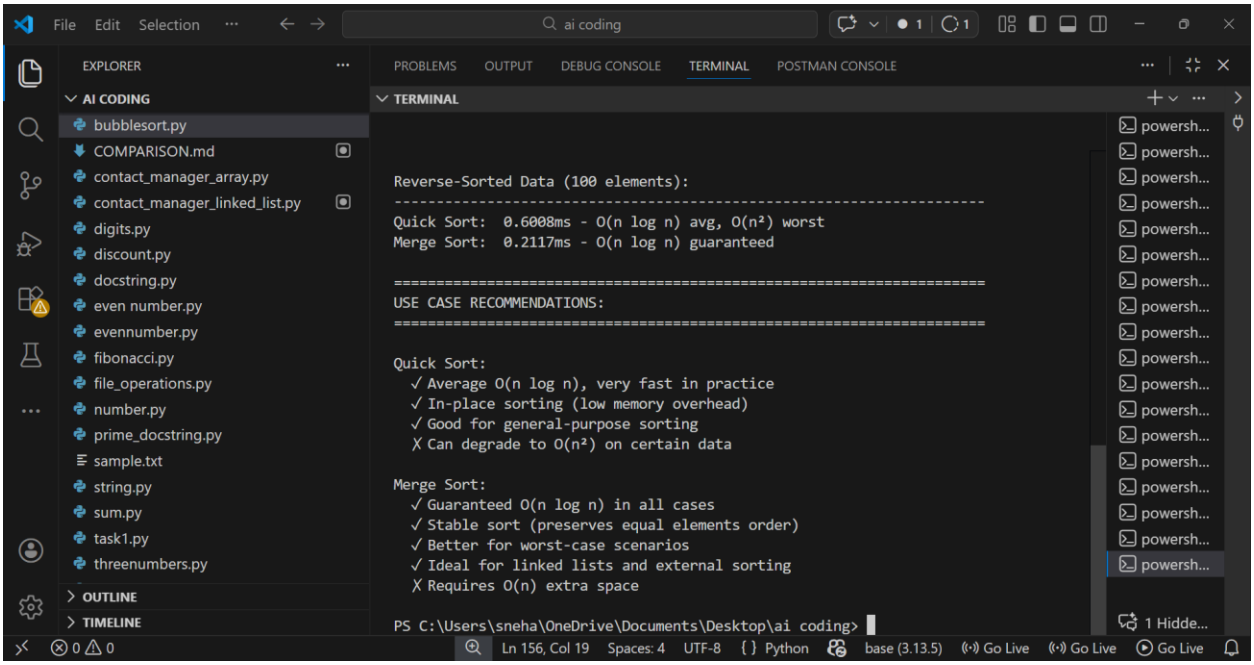

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
def merge_sort(arr: List[int]) -> List[int]:
"""
Merge Sort using divide-and-conquer recursion.
How recursion works:
1. Base case: if length <= 1, array is sorted
2. Divide: split array into two halves
3. Recursive case: recursively sort both halves
4. Conquer: merge sorted halves back together
Time Complexity: O(n log n) in all cases
Space Complexity: O(n) for temporary arrays
"""
if len(arr) <= 1:
return arr
# Divide step
mid = len(arr) // 2
left = arr[:mid]
right = arr[mid:]
# Recursive step
0 0
Ln 156, Col 19 Spaces: 4 UTF-8 { } Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
def test_sorting():
"""Test both algorithms on different data patterns."""
print("=" * 70)
print("SORTING ALGORITHM COMPARISON")
print("=" * 70)
# Test cases
test_cases = {
"Random": [random.randint(1, 1000) for _ in range(100)],
"Sorted": list(range(100)),
"Reverse-Sorted": list(range(100, 0, -1))
}
for test_name, data in test_cases.items():
print(f"\n{test_name} Data (100 elements):")
print("-" * 70)
# Quick Sort
qs_data = data.copy()
start = time.time()
quick_sort(qs_data)
0 0
Ln 156, Col 19 Spaces: 4 UTF-8 { } Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
def test_sorting():
print("\n" + "=" * 70)
print("USE CASE RECOMMENDATIONS:")
print("=" * 70)
print("""
Quick Sort:
✓ Average O(n log n), very fast in practice
✓ In-place sorting (low memory overhead)
✓ Good for general-purpose sorting
✗ Can degrade to O(n²) on certain data
Merge Sort:
✓ Guaranteed O(n log n) in all cases
✓ Stable sort (preserves equal elements order)
✓ Better for worst-case scenarios
✓ Ideal for linked lists and external sorting
✗ Requires O(n) extra space
""")
if __name__ == "__main__":
test_sorting()
Ln 156, Col 19
Spaces: 4
UTF-8
Python
base (3.13.5)
Go Live
Go Live
Go Live

Output:





Explanation:

- **Quick Sort** uses a pivot to divide the dataset into smaller parts and sorts them recursively, making it fast for random data.
- **Merge Sort** divides the dataset into halves and merges them after sorting, ensuring stable and consistent performance.
- Recursion works by repeatedly breaking the problem into smaller subproblems until they become simple to solve.

- Quick Sort is usually faster in practice but can perform poorly on already sorted or reverse-sorted data (worst case).
- Merge Sort guarantees **O(n log n)** performance and is preferred when stable sorting or predictable speed is required.

Task 5:

Optimizing a Duplicate Detection Algorithm

Scenario

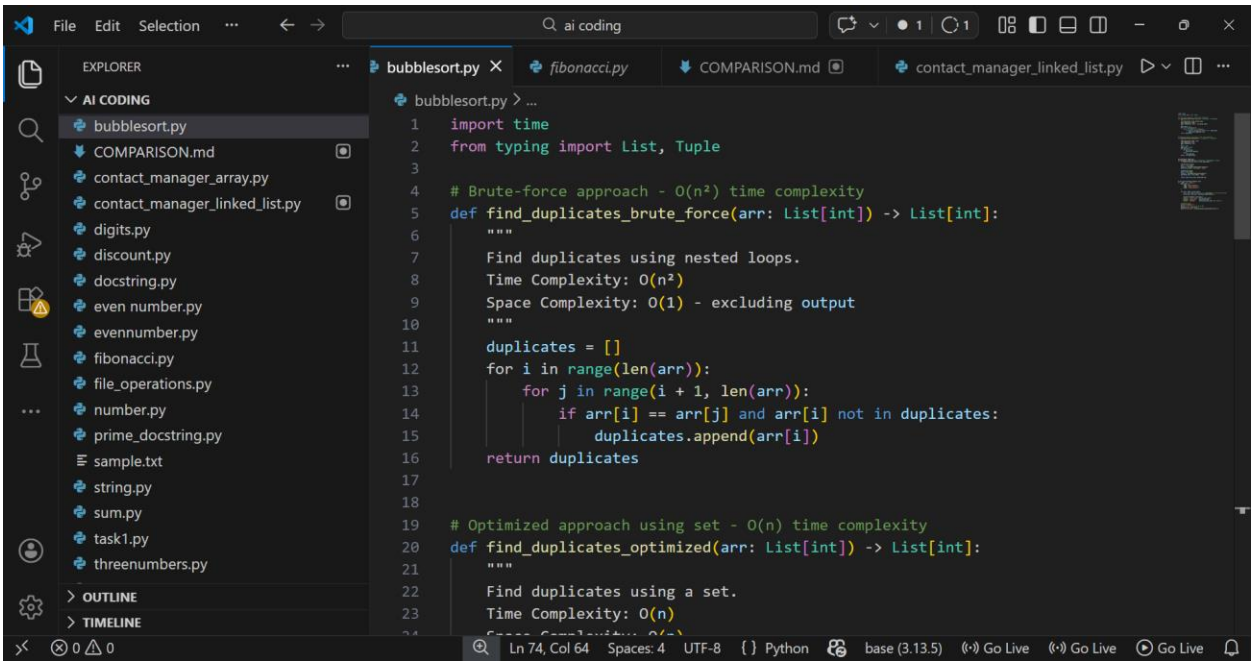
You are building a data validation module that must detect duplicate

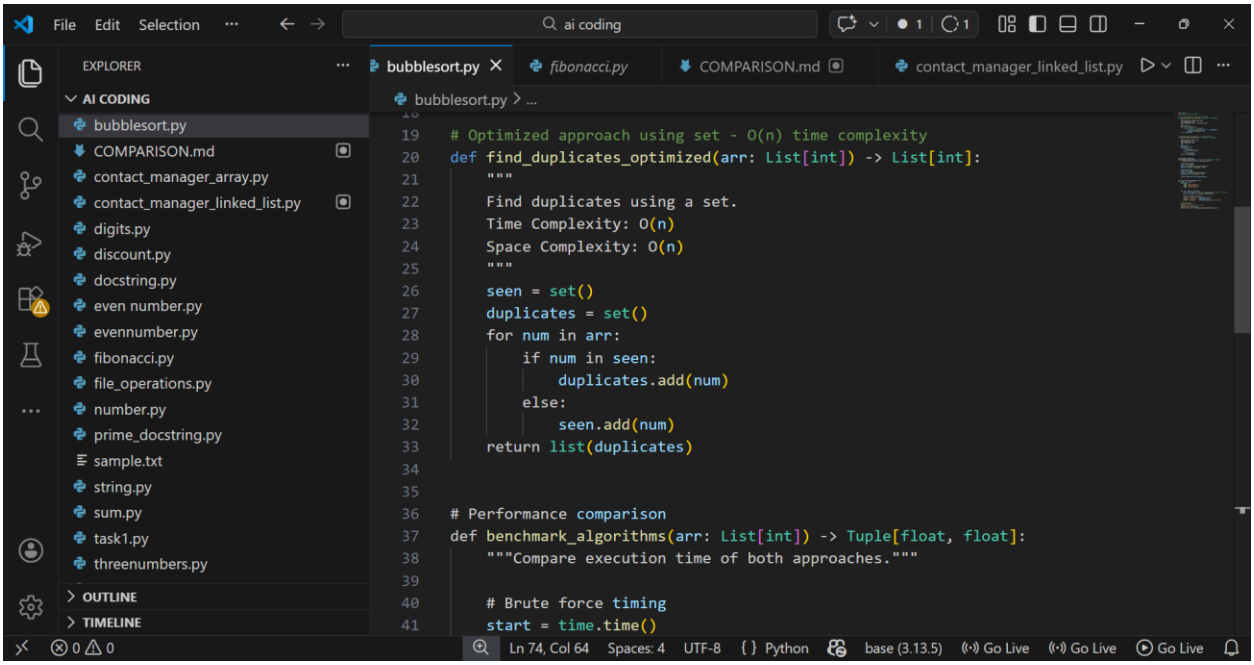
user IDs in a large dataset before importing it into a system.

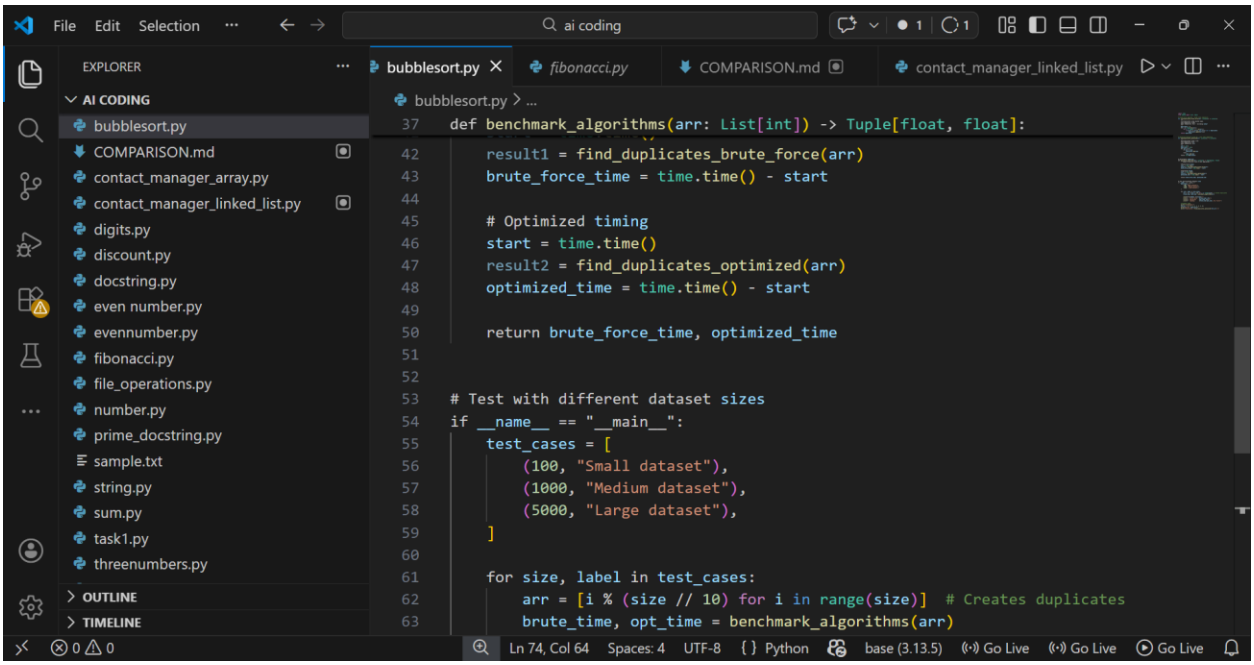
Prompt:

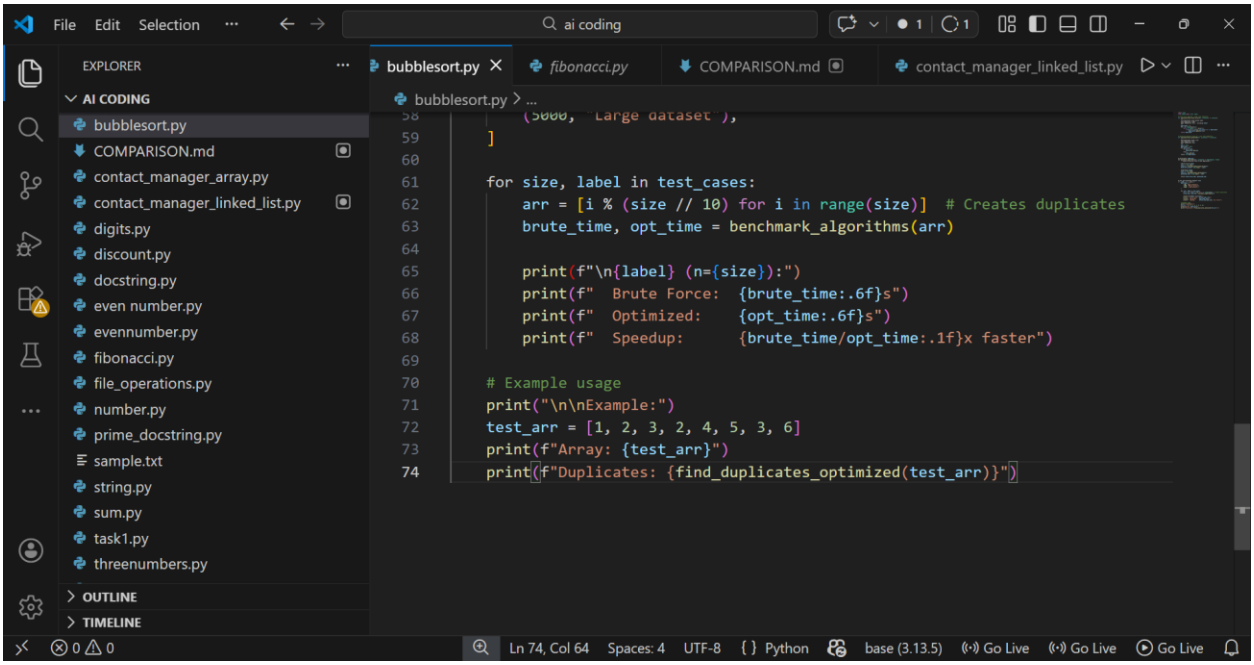
Implement a brute-force duplicate detection algorithm using nested loops, analyze its time complexity, then optimize it using a set or dictionary and compare performance for large datasets.

Code:

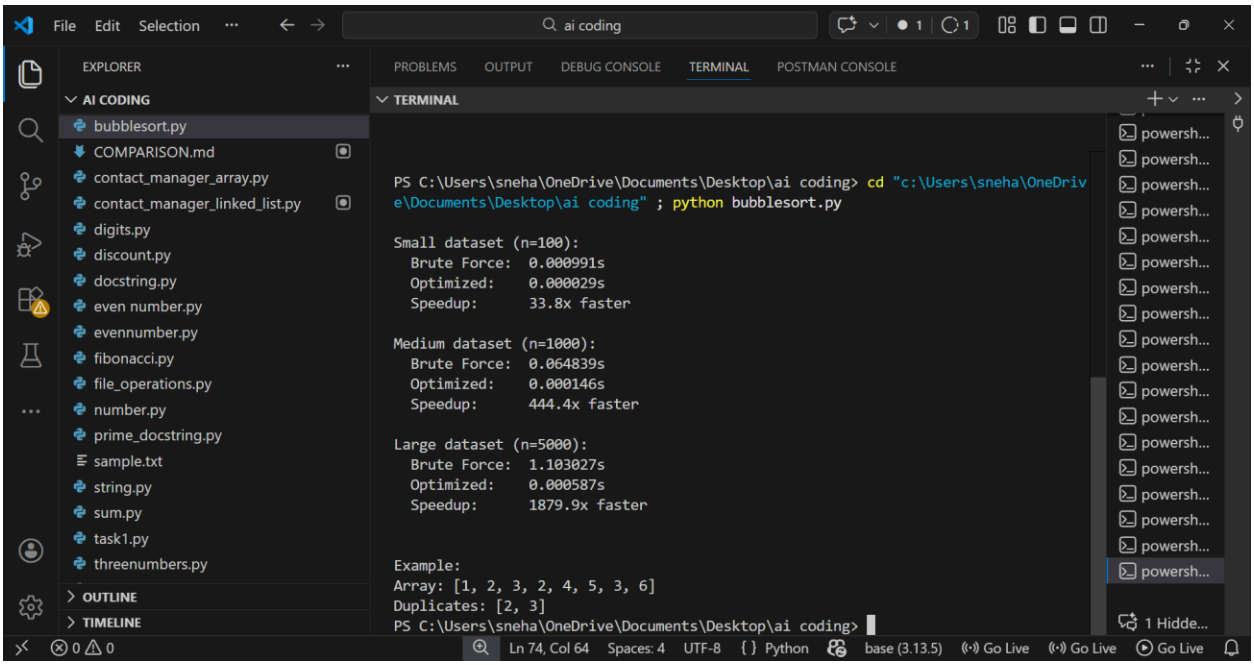

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
import time
from typing import List, Tuple
# Brute-force approach - O(n²) time complexity
def find_duplicates_brute_force(arr: List[int]) -> List[int]:
"""
Find duplicates using nested loops.
Time Complexity: O(n²)
Space Complexity: O(1) - excluding output
"""
duplicates = []
for i in range(len(arr)):
for j in range(i + 1, len(arr)):
if arr[i] == arr[j] and arr[i] not in duplicates:
duplicates.append(arr[i])
return duplicates
# Optimized approach using set - O(n) time complexity
def find_duplicates_optimized(arr: List[int]) -> List[int]:
"""
Find duplicates using a set.
Time Complexity: O(n)
0 0
Ln 74, Col 64
Spaces: 4
UTF-8
Python
base (3.13.5)
Go Live
Go Live
Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
bubblesort.py > ...
# Optimized approach using set - O(n) time complexity
def find_duplicates_optimized(arr: List[int]) -> List[int]:
"""
Find duplicates using a set.
Time Complexity: O(n)
Space Complexity: O(n)
"""
seen = set()
duplicates = set()
for num in arr:
if num in seen:
duplicates.add(num)
else:
seen.add(num)
return list(duplicates)
# Performance comparison
def benchmark_algorithms(arr: List[int]) -> Tuple[float, float]:
"""Compare execution time of both approaches."""
# Brute force timing
start = time.time()
0 0
Ln 74, Col 64
Spaces: 4
UTF-8
Python
base (3.13.5)
Go Live
Go Live
Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
def benchmark_algorithms(arr: List[int]) -> Tuple[float, float]:
result1 = find_duplicates_brute_force(arr)
brute_force_time = time.time() - start
# Optimized timing
start = time.time()
result2 = find_duplicates_optimized(arr)
optimized_time = time.time() - start
return brute_force_time, optimized_time
# Test with different dataset sizes
if __name__ == "__main__":
test_cases = [
(100, "Small dataset"),
(1000, "Medium dataset"),
(5000, "Large dataset"),
]
for size, label in test_cases:
arr = [i % (size // 10) for i in range(size)] # Creates duplicates
brute_time, opt_time = benchmark_algorithms(arr)
Ln 74, Col 64 Spaces: 4 UTF-8 Python base (3.13.5) Go Live Go Live Go Live

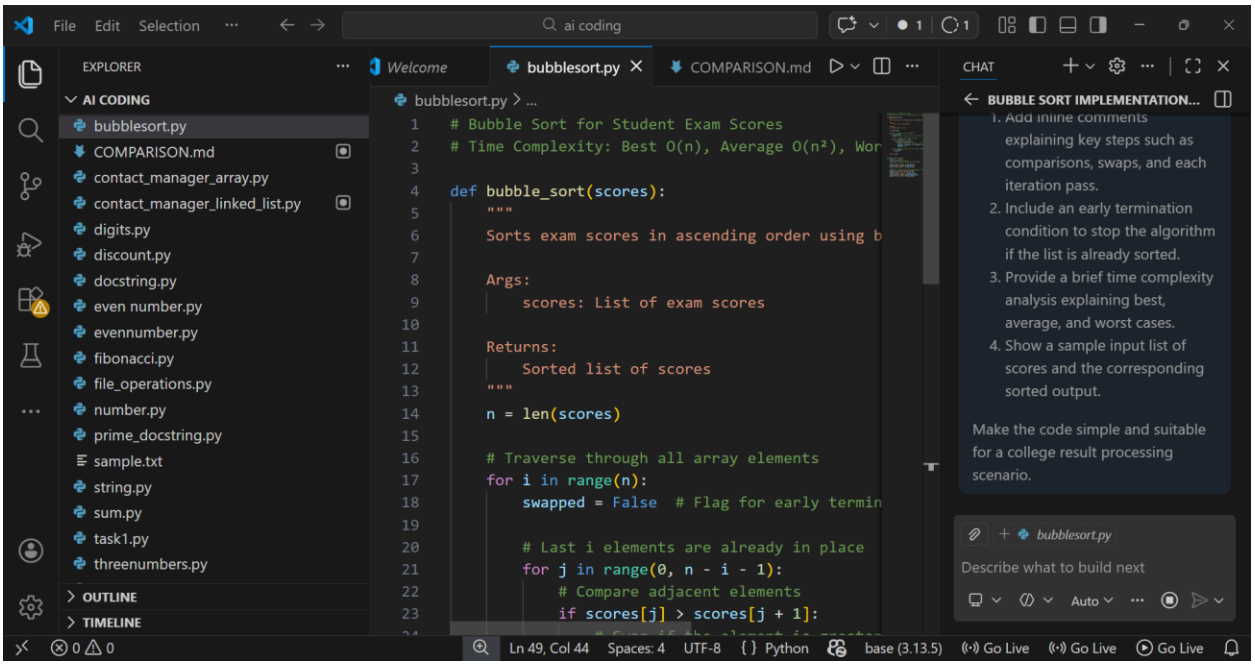

File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
bubblesort.py
fibonacci.py
COMPARISON.md
contact_manager_linked_list.py
(5000, "Large dataset"),
]
for size, label in test_cases:
arr = [i % (size // 10) for i in range(size)] # Creates duplicates
brute_time, opt_time = benchmark_algorithms(arr)
print(f"\n{label} (n={size}):")
print(f" Brute Force: {brute_time:.6f}s")
print(f" Optimized: {opt_time:.6f}s")
print(f" Speedup: {brute_time/opt_time:.1f}x faster")
# Example usage
print("\n\nExample:")
test_arr = [1, 2, 3, 2, 4, 5, 3, 6]
print(f"Array: {test_arr}")
print(f"Duplicates: {find_duplicates_optimized(test_arr)}")
Ln 74, Col 64 Spaces: 4 UTF-8 Python base (3.13.5) Go Live Go Live Go Live

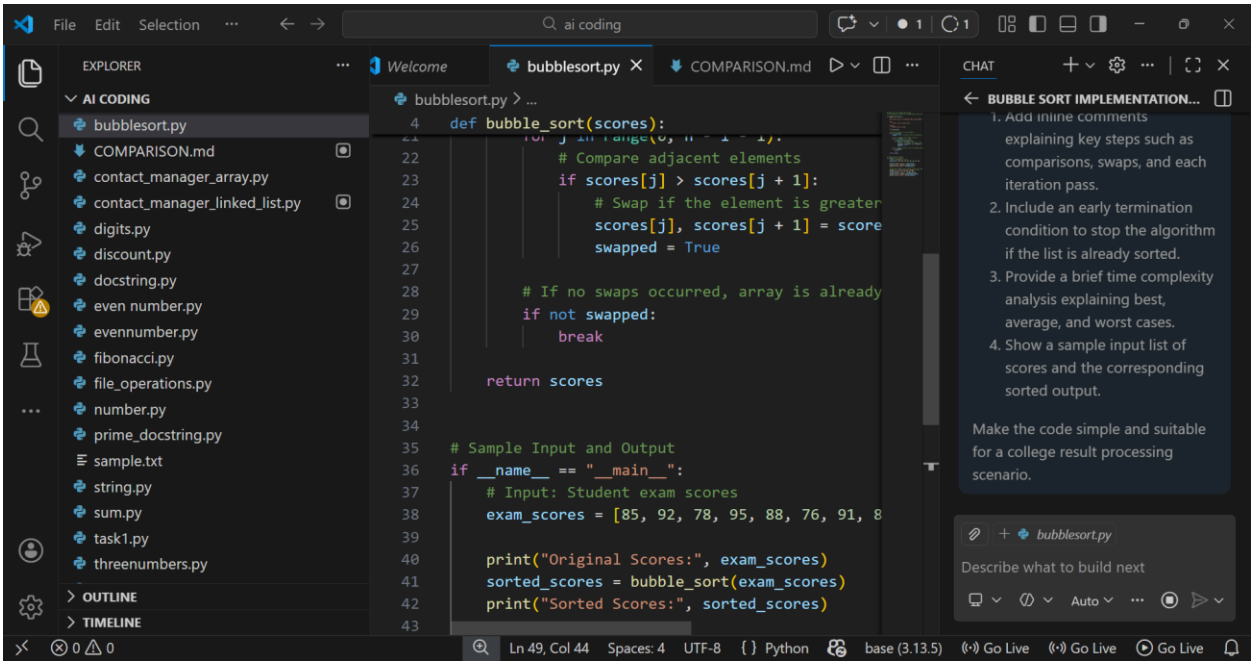
Output:

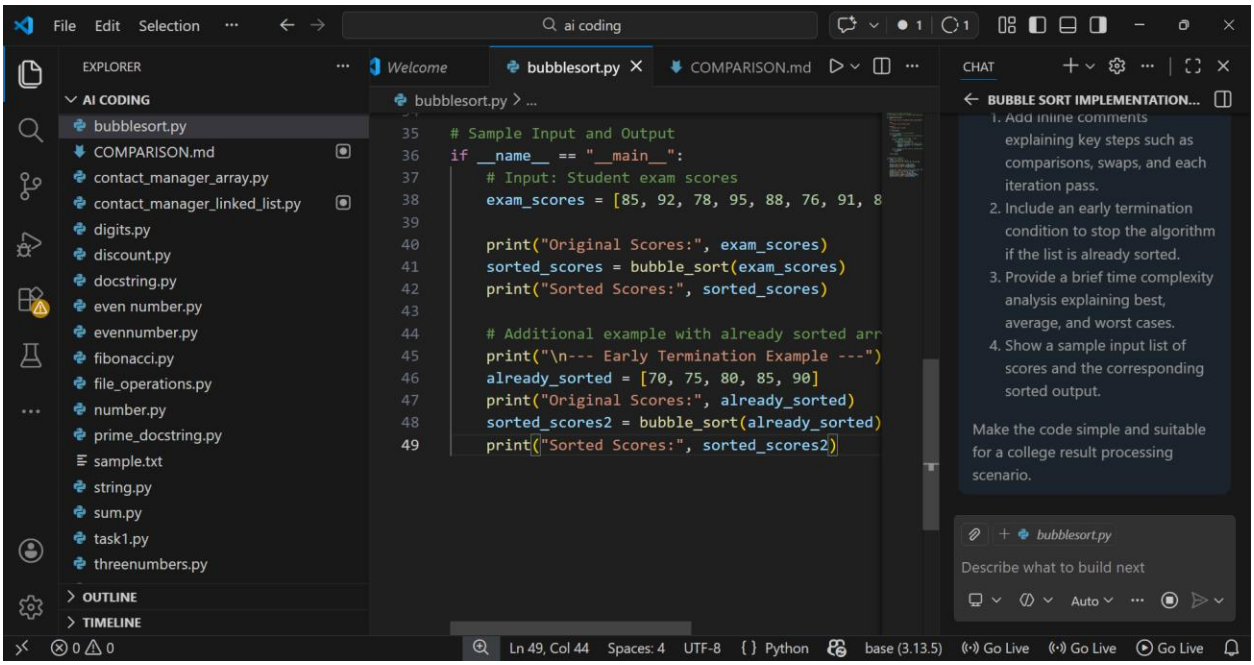


Explanation:

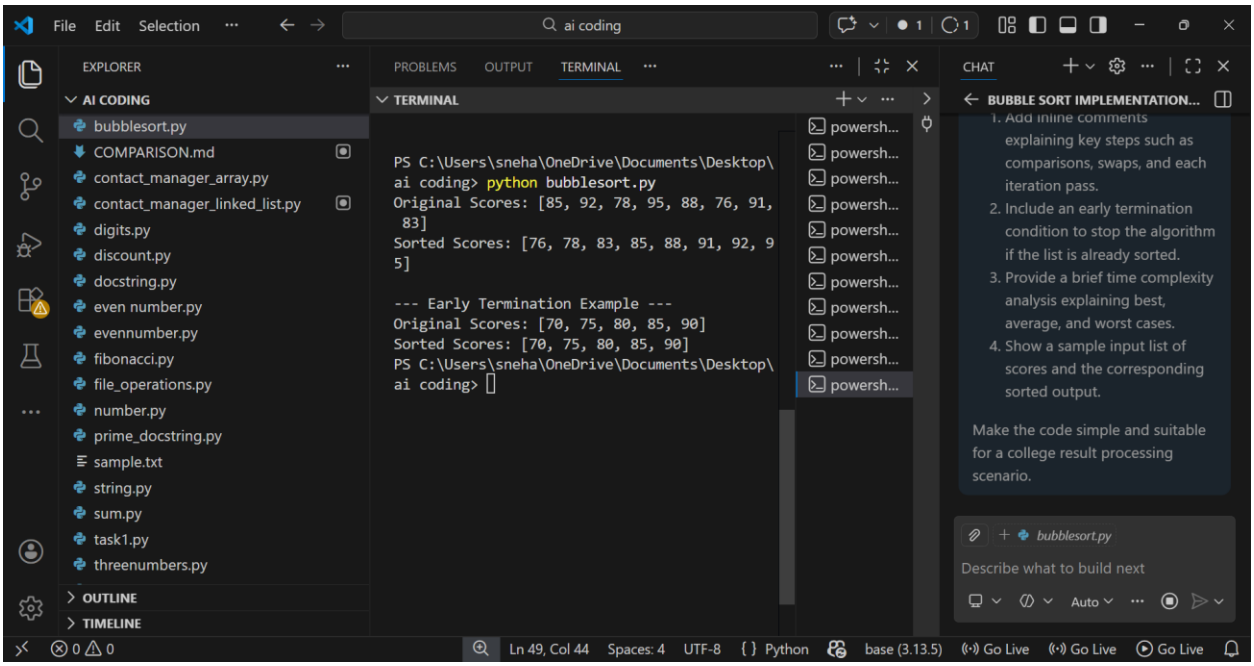
- The brute-force method uses nested loops to compare each user ID with every other ID, resulting in **O(n²)** time complexity.
- This approach becomes slow for large datasets because the number of comparisons grows rapidly.
- The optimized method uses a **set or dictionary** to track seen IDs, reducing the time complexity to **O(n)**.
- Performance improves because each ID is checked only once instead of repeatedly comparing with all others.
- For large inputs, the optimized approach runs much faster and is preferred for real-world data validation systems.


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
bubblesort.py > ...
# Bubble Sort for Student Exam Scores
# Time Complexity: Best O(n), Average O(n²), Wor
def bubble_sort(scores):
"""
Sorts exam scores in ascending order using b
Args:
scores: List of exam scores
Returns:
Sorted list of scores
"""
n = len(scores)
# Traverse through all array elements
for i in range(n):
swapped = False # Flag for early termin
# Last i elements are already in place
for j in range(0, n - i - 1):
# Compare adjacent elements
if scores[j] > scores[j + 1]:
CHAT
BUBBLE SORT IMPLEMENTATION...
1. Add inline comments explaining key steps such as comparisons, swaps, and each iteration pass.
2. Include an early termination condition to stop the algorithm if the list is already sorted.
3. Provide a brief time complexity analysis explaining best, average, and worst cases.
4. Show a sample input list of scores and the corresponding sorted output.
Make the code simple and suitable for a college result processing scenario.
bubblesort.py
Describe what to build next
Auto
Ln 49, Col 44 Spaces: 4 UTF-8 Python base (3.13.5) Go Live Go Live Go Live


File Edit Selection
ai coding
EXPLORER
AI CODING
bubblesort.py
COMPARISON.md
contact_manager_array.py
contact_manager_linked_list.py
digits.py
discount.py
docstring.py
even number.py
evennumber.py
fibonacci.py
file_operations.py
number.py
prime_docstring.py
sample.txt
string.py
sum.py
task1.py
threenumbers.py
OUTLINE
TIMELINE
Welcome
bubblesort.py
COMPARISON.md
bubblesort.py > ...
def bubble_sort(scores):
# Compare adjacent elements
if scores[j] > scores[j + 1]:
# Swap if the element is greater
scores[j], scores[j + 1] = score
swapped = True
# If no swaps occurred, array is already
if not swapped:
break
return scores
# Sample Input and Output
if __name__ == "__main__":
# Input: Student exam scores
exam_scores = [85, 92, 78, 95, 88, 76, 91, 8
print("Original Scores:", exam_scores)
sorted_scores = bubble_sort(exam_scores)
print("Sorted Scores:", sorted_scores)
CHAT
BUBBLE SORT IMPLEMENTATION...
1. Add inline comments explaining key steps such as comparisons, swaps, and each iteration pass.
2. Include an early termination condition to stop the algorithm if the list is already sorted.
3. Provide a brief time complexity analysis explaining best, average, and worst cases.
4. Show a sample input list of scores and the corresponding sorted output.
Make the code simple and suitable for a college result processing scenario.
bubblesort.py
Describe what to build next
Auto
Ln 49, Col 44 Spaces: 4 UTF-8 Python base (3.13.5) Go Live Go Live Go Live



Output:



Explanation:

- Bubble Sort compares adjacent exam scores and swaps them if they are in the wrong order.
- It repeats multiple passes until all scores are arranged correctly.
- After each pass, the largest score moves to its correct position (“bubbles up”).
- If no swaps happen in a pass, the algorithm stops early because the list is already sorted.
- Time complexity: Best — O(n), Average — $O(n^2)$, Worst — $O(n^2)$.

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.

Prompt: