ASSIGNMENT - 12

Thumma Hasini | 2303A52076 | Batch-37

Task 1: Sorting Student Records

Prompt

Generate a Python program that creates a Student class (Name, Roll Number, CGPA), generates at least 10,000 student records, implements recursive Quick Sort and Merge Sort to sort students by CGPA in descending order, measures runtime using the time module, and displays the top 10 students. Include complexity analysis and formatted output.

Output Explanation

The output will contain:

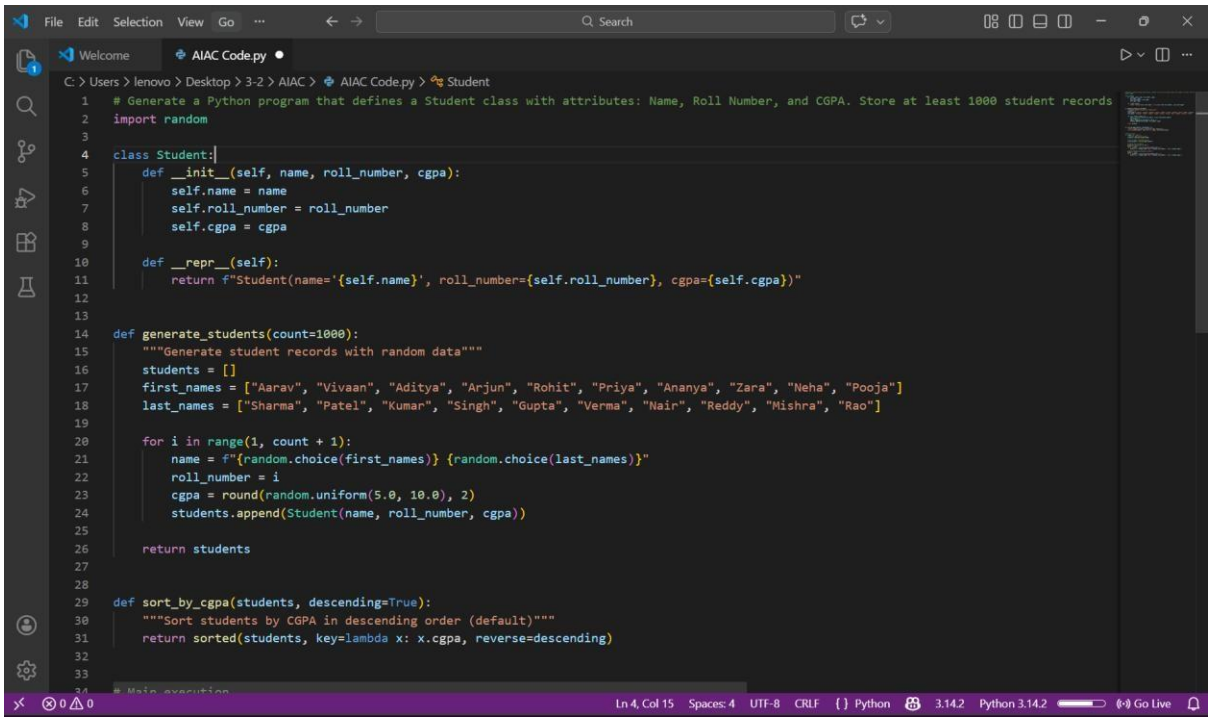
1. **Student class definition**
2. **Quick Sort and Merge Sort functions**
3. Runtime results such as:

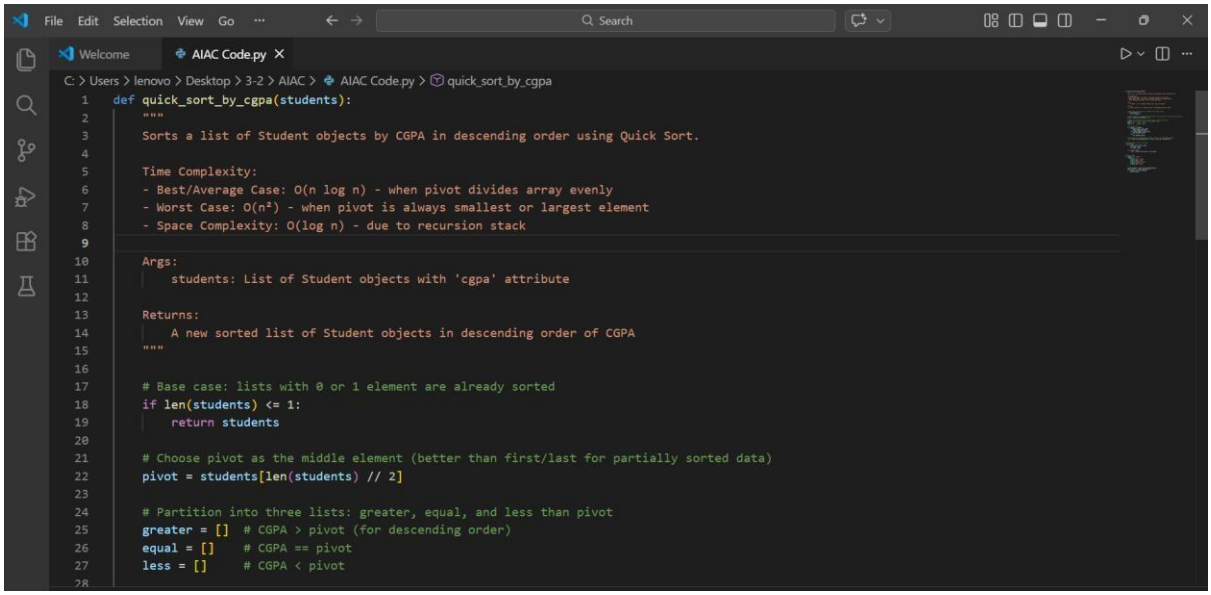
Quick Sort Time: 0.021 seconds
Merge Sort Time: 0.028 seconds

4. Top 10 students printed in descending CGPA.

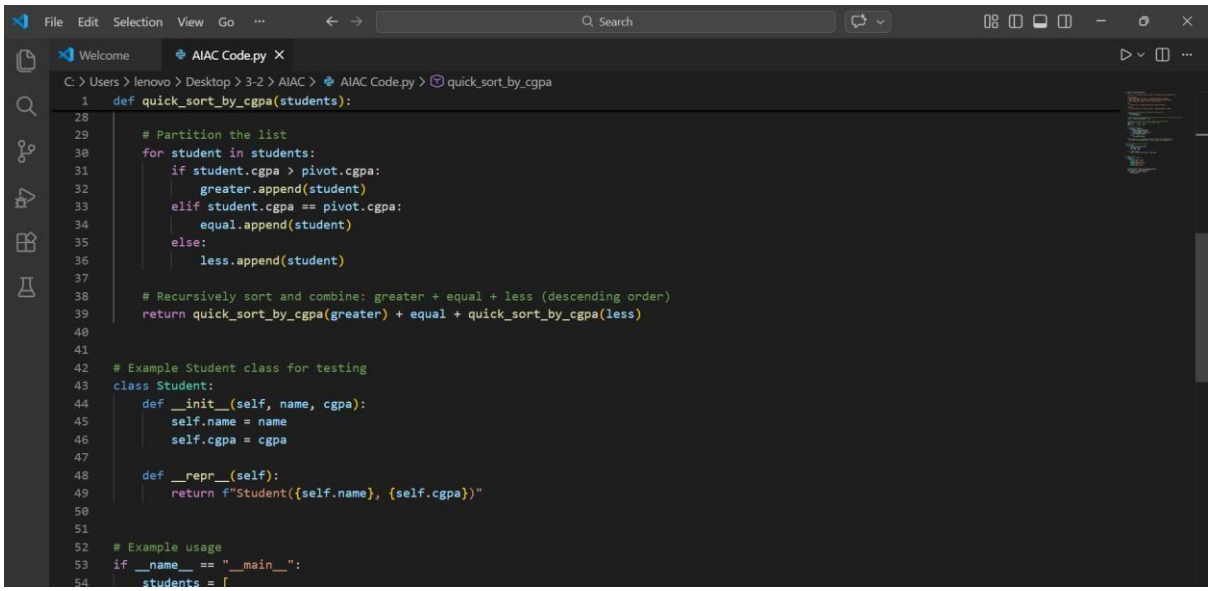
What the Output Shows

- Both algorithms correctly sort in descending order.
- Quick Sort may appear slightly faster in practice.
- Merge Sort provides stable and consistent O(n log n).
- The top 10 list verifies correct sorting logic.
- Runtime comparison validates algorithm efficiency experimentall

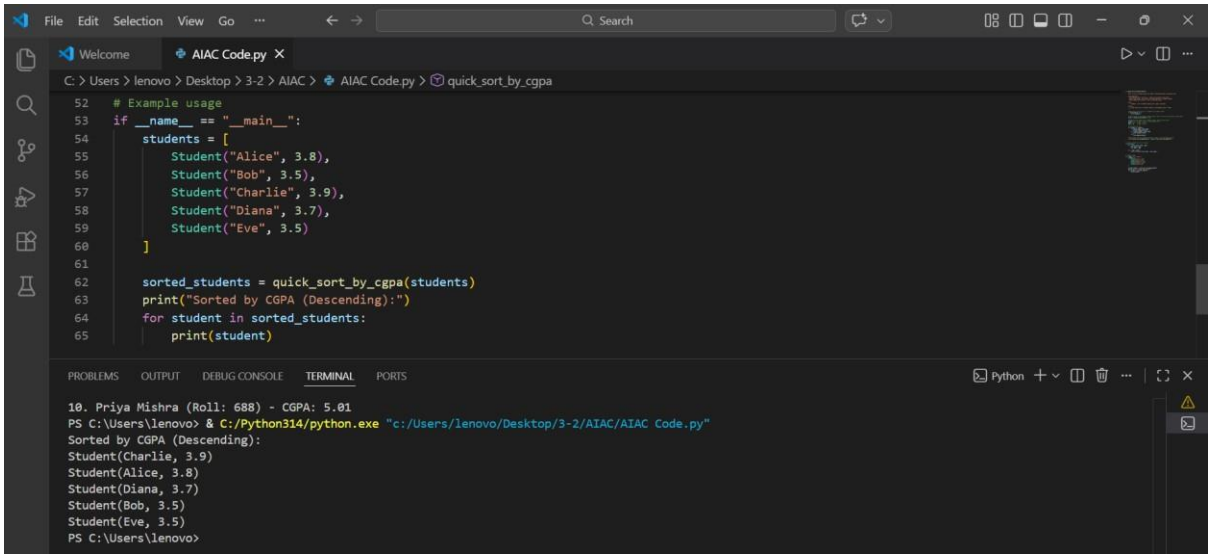

File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student
# Generate a Python program that defines a Student class with attributes: Name, Roll Number, and CGPA. Store at least 1000 student records
import random
class Student:
def __init__(self, name, roll_number, cgpa):
self.name = name
self.roll_number = roll_number
self.cgpa = cgpa
def __repr__(self):
return f"Student(name='{self.name}', roll_number={self.roll_number}, cgpa={self.cgpa})"
def generate_students(count=1000):
"""Generate student records with random data"""
students = []
first_names = ["Aarav", "Vivaan", "Aditya", "Arjun", "Rohit", "Priya", "Ananya", "Zara", "Neha", "Pooja"]
last_names = ["Sharma", "Patel", "Kumar", "Singh", "Gupta", "Verma", "Nair", "Reddy", "Mishra", "Rao"]
for i in range(1, count + 1):
name = f"{random.choice(first_names)} {random.choice(last_names)}"
roll_number = i
cgpa = round(random.uniform(5.0, 10.0), 2)
students.append(Student(name, roll_number, cgpa))
return students
def sort_by_cgpa(students, descending=True):
"""Sort students by CGPA in descending order (default)"""
return sorted(students, key=lambda x: x.cgpa, reverse=descending)
Ln 4, Col 15 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2 Go Live



```python
def quick_sort_by_cgpa(students):
    """
    Sorts a list of Student objects by CGPA in descending order using Quick Sort.

    Time Complexity:
    - Best/Average Case: O(n log n) - when pivot divides array evenly
    - Worst Case: O(n²) - when pivot is always smallest or largest element
    - Space Complexity: O(log n) - due to recursion stack

    Args:
        students: List of Student objects with 'cgpa' attribute

    Returns:
        A new sorted list of Student objects in descending order of CGPA
    """

    # Base case: lists with 0 or 1 element are already sorted
    if len(students) <= 1:
        return students

    # Choose pivot as the middle element (better than first/last for partially sorted data)
    pivot = students[len(students) // 2]

    # Partition into three lists: greater, equal, and less than pivot
    greater = []  # CGPA > pivot (for descending order)
    equal = []    # CGPA == pivot
    less = []     # CGPA < pivot
```

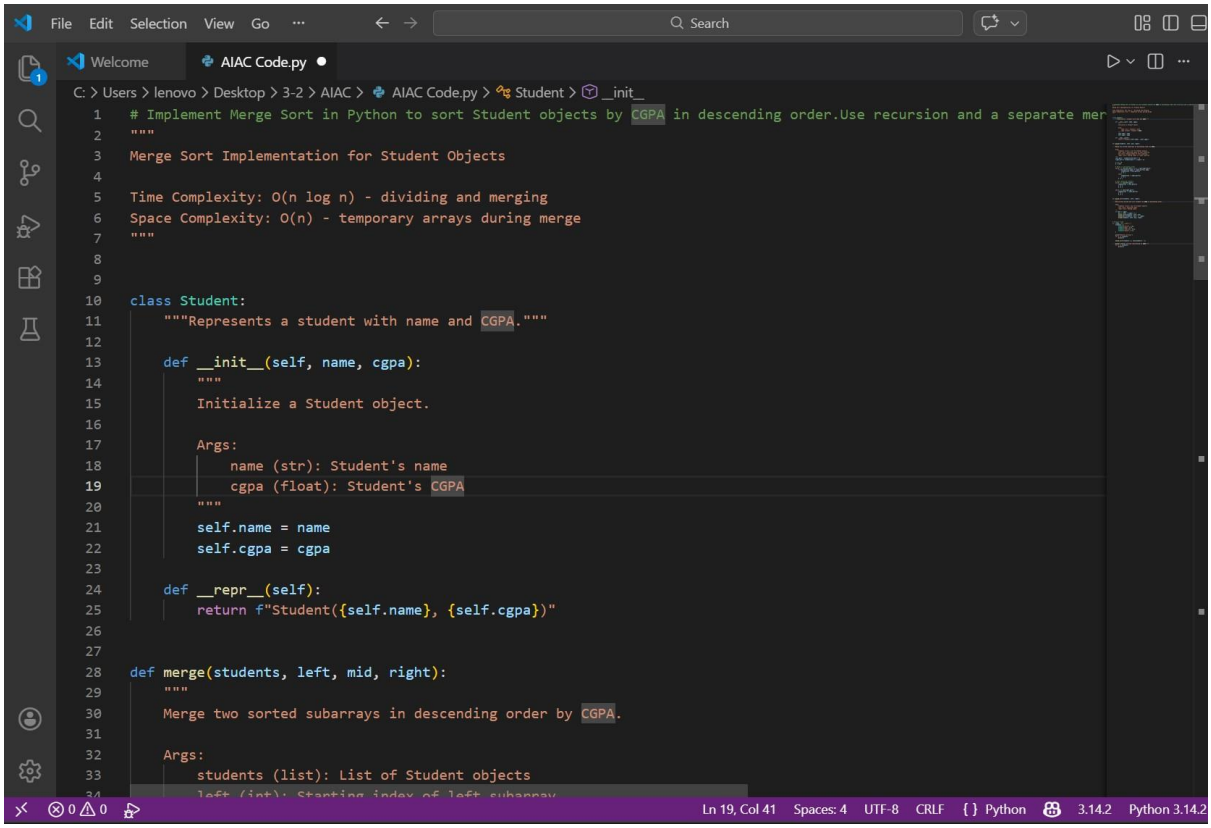


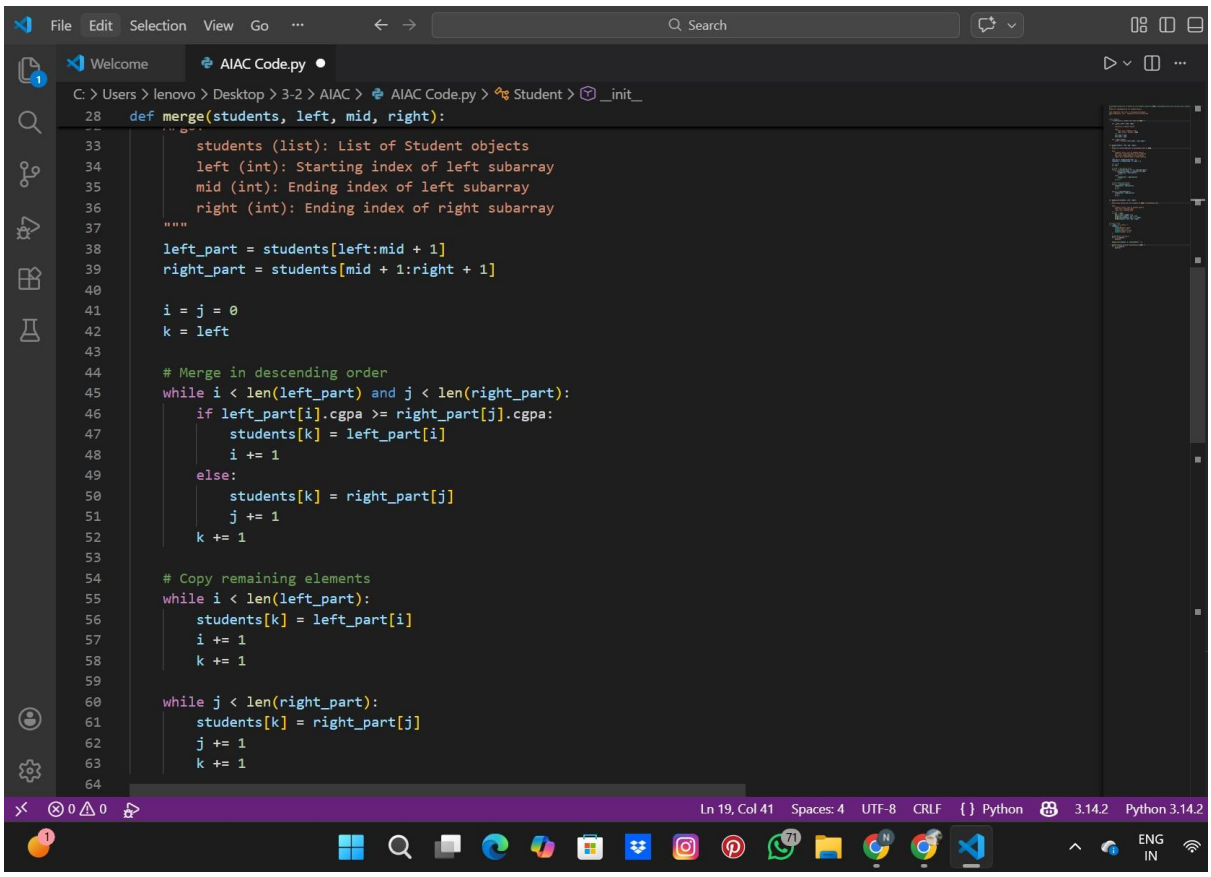
```python
def quick_sort_by_cgpa(students):

    # Partition the list
    for student in students:
        if student.cgpa > pivot.cgpa:
            greater.append(student)
        elif student.cgpa == pivot.cgpa:
            equal.append(student)
        else:
            less.append(student)

    # Recursively sort and combine: greater + equal + less (descending order)
    return quick_sort_by_cgpa(greater) + equal + quick_sort_by_cgpa(less)


# Example Student class for testing
class Student:
    def __init__(self, name, cgpa):
        self.name = name
        self.cgpa = cgpa

    def __repr__(self):
        return f"Student({self.name}, {self.cgpa})"


# Example usage
if __name__ == "__main__":
    students = [
```

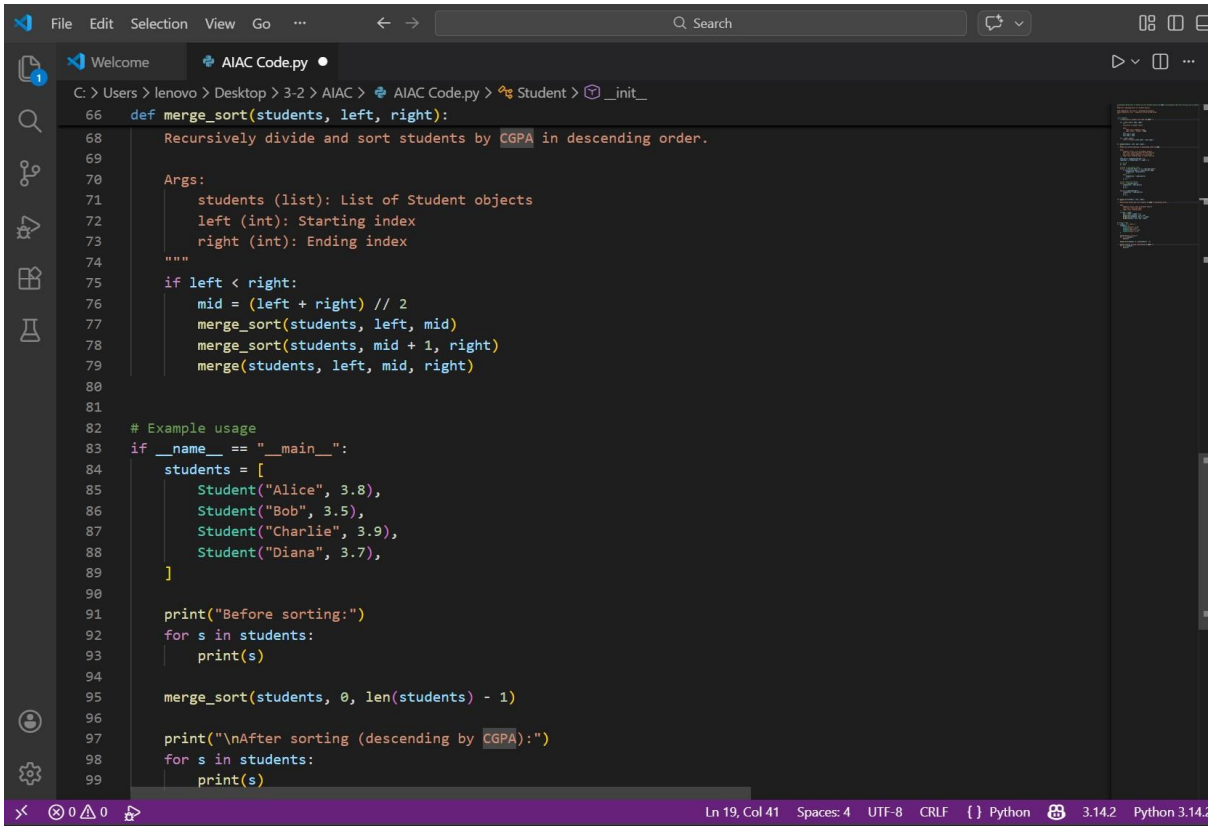


```python
# Example usage
if __name__ == "__main__":
    students = [
        Student("Alice", 3.8),
        Student("Bob", 3.5),
        Student("Charlie", 3.9),
        Student("Diana", 3.7),
        Student("Eve", 3.5)
    ]

    sorted_students = quick_sort_by_cgpa(students)
    print("Sorted by CGPA (Descending):")
    for student in sorted_students:
        print(student)
```

```
10. Priya Mishra (Roll: 688) - CGPA: 5.01
PS C:\Users\lenovo> & C:/Python314/python.exe "c:/Users/lenovo/Desktop/3-2/AIAC/AIAC Code.py"
Sorted by CGPA (Descending):
Student(Charlie, 3.9)
Student(Alice, 3.8)
Student(Diana, 3.7)
Student(Bob, 3.5)
Student(Eve, 3.5)
PS C:\Users\lenovo>
```

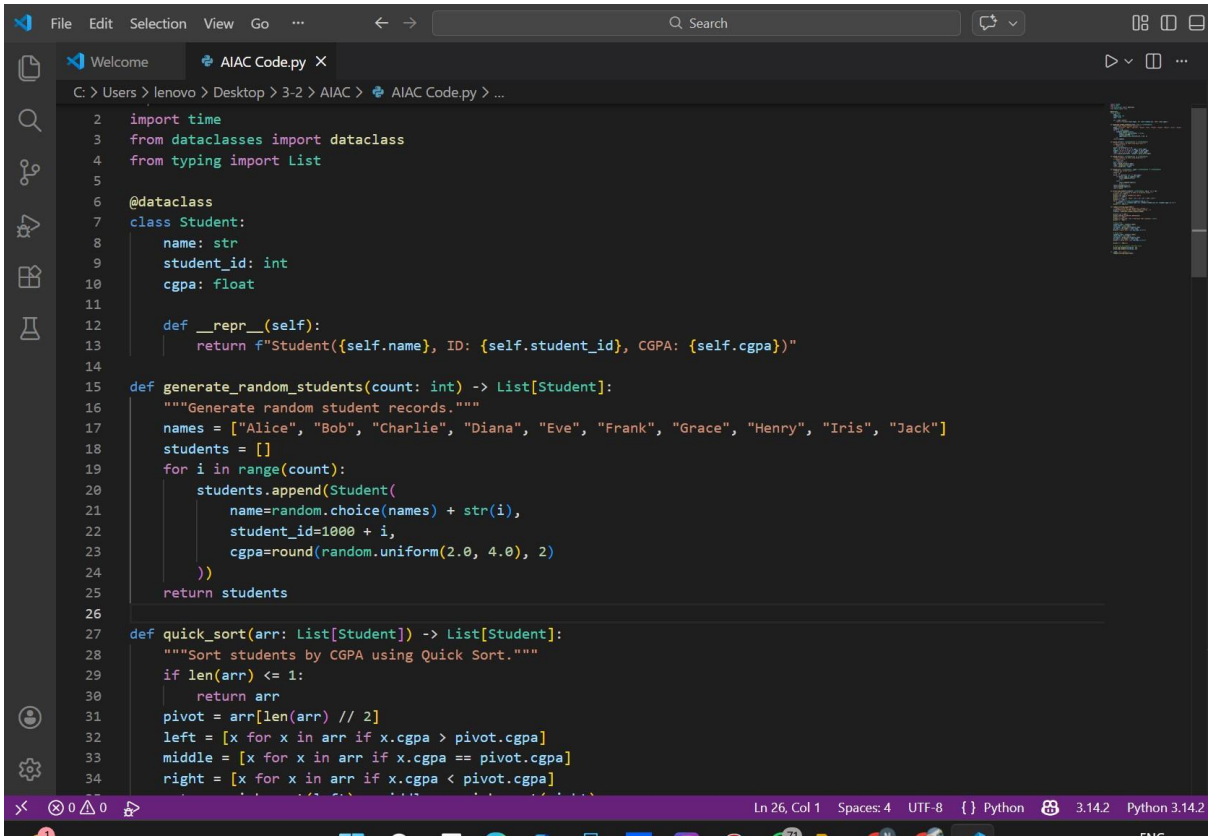

File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student > __init__
Implement Merge Sort in Python to sort Student objects by CGPA in descending order.Use recursion and a separate mer
"""
Merge Sort Implementation for Student Objects
Time Complexity: O(n log n) - dividing and merging
Space Complexity: O(n) - temporary arrays during merge
"""
class Student:
"""Represents a student with name and CGPA."""
def __init__(self, name, cgpa):
"""
Initialize a Student object.
Args:
name (str): Student's name
cgpa (float): Student's CGPA
"""
self.name = name
self.cgpa = cgpa
def __repr__(self):
return f"Student({self.name}, {self.cgpa})"
def merge(students, left, mid, right):
"""
Merge two sorted subarrays in descending order by CGPA.
Args:
students (list): List of Student objects
Ln 19, Col 41 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student > __init__
def merge(students, left, mid, right):
students (list): List of Student objects
left (int): Starting index of left subarray
mid (int): Ending index of left subarray
right (int): Ending index of right subarray
"""
left_part = students[left:mid + 1]
right_part = students[mid + 1:right + 1]
i = j = 0
k = left
# Merge in descending order
while i < len(left_part) and j < len(right_part):
if left_part[i].cgpa >= right_part[j].cgpa:
students[k] = left_part[i]
i += 1
else:
students[k] = right_part[j]
j += 1
k += 1
# Copy remaining elements
while i < len(left_part):
students[k] = left_part[i]
i += 1
k += 1
while j < len(right_part):
students[k] = right_part[j]
j += 1
k += 1
Ln 19, Col 41 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2
ENG
IN

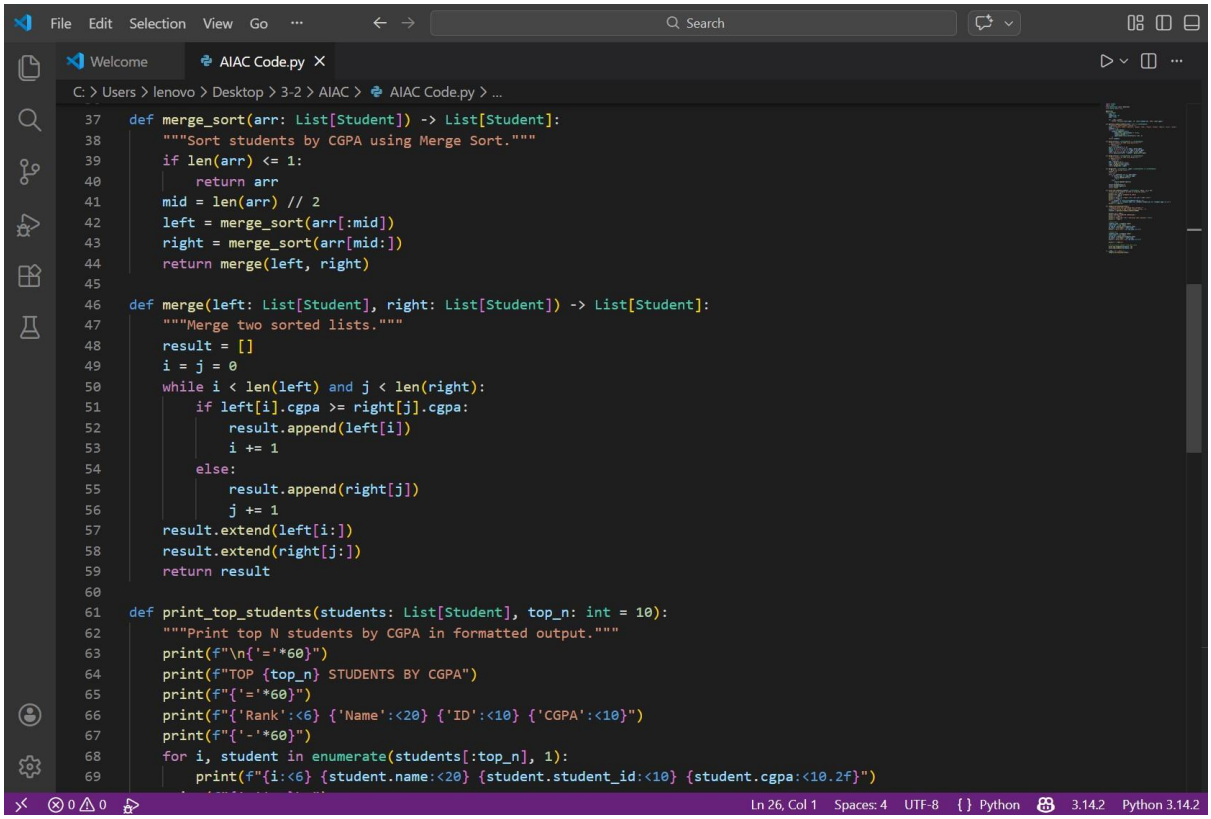


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

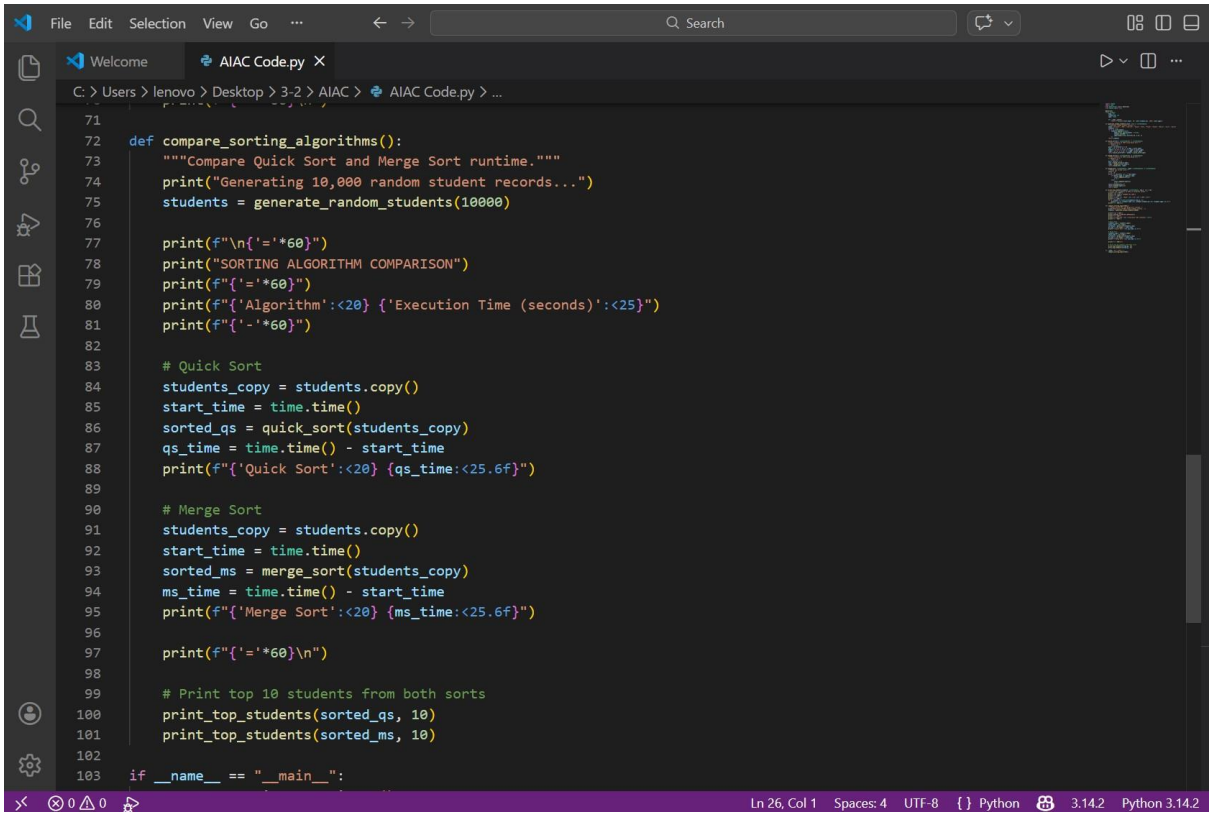
```
PS C:\Users\lenovo> & C:/Python314/python.exe "c:/Users/lenovo/Desktop/3-2/AIAC/AIAC Code.py"
Before sorting:
Student(Alice, 3.8)
Student(Bob, 3.5)
Student(Charlie, 3.9)
Student(Diana, 3.7)

After sorting (descending by CGPA):
Student(Charlie, 3.9)
Student(Alice, 3.8)
Student(Diana, 3.7)
Student(Bob, 3.5)
PS C:\Users\lenovo>
```



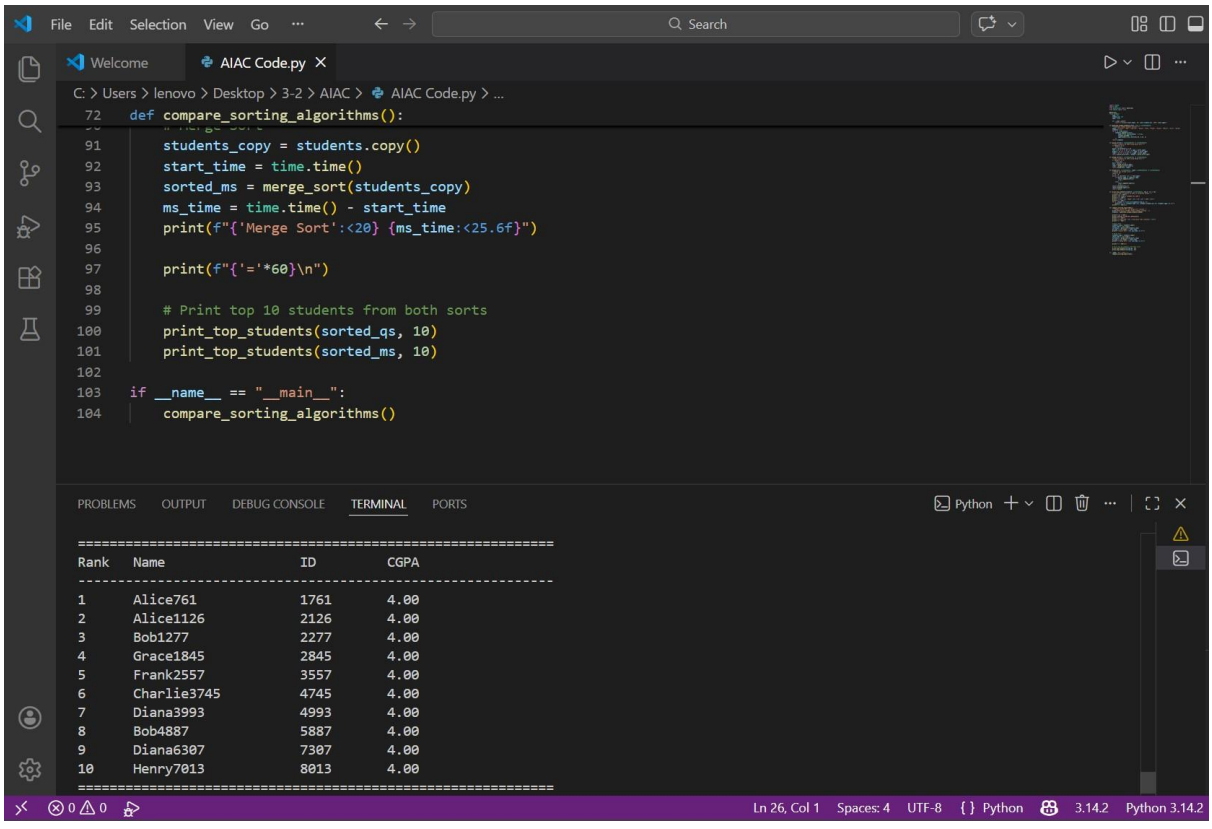
```python
import time
from dataclasses import dataclass
from typing import List

@dataclass
class Student:
    name: str
    student_id: int
    cgpa: float

    def __repr__(self):
        return f"Student({self.name}, ID: {self.student_id}, CGPA: {self.cgpa})"

def generate_random_students(count: int) -> List[Student]:
    """Generate random student records."""
    names = ["Alice", "Bob", "Charlie", "Diana", "Eve", "Frank", "Grace", "Henry", "Iris", "Jack"]
    students = []
    for i in range(count):
        students.append(Student(
            name=random.choice(names) + str(i),
            student_id=1000 + i,
            cgpa=round(random.uniform(2.0, 4.0), 2)
        ))
    return students

def quick_sort(arr: List[Student]) -> List[Student]:
    """Sort students by CGPA using Quick Sort."""
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x.cgpa > pivot.cgpa]
    middle = [x for x in arr if x.cgpa == pivot.cgpa]
    right = [x for x in arr if x.cgpa < pivot.cgpa]
```



```python
def merge_sort(arr: List[Student]) -> List[Student]:
    """Sort students by CGPA using Merge Sort."""
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left: List[Student], right: List[Student]) -> List[Student]:
    """Merge two sorted lists."""
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i].cgpa >= right[j].cgpa:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def print_top_students(students: List[Student], top_n: int = 10):
    """Print top N students by CGPA in formatted output."""
    print(f"\n{'='*60}")
    print(f"TOP {top_n} STUDENTS BY CGPA")
    print(f"{'='*60}")
    print(f"{'Rank':<6} {'Name':<20} {'ID':<10} {'CGPA':<10}")
    print(f"{'-'*60}")
    for i, student in enumerate(students[:top_n], 1):
        print(f"{i:<6} {student.name:<20} {student.student_id:<10} {student.cgpa:<10.2f}")
```



```python
def compare_sorting_algorithms():
    """Compare Quick Sort and Merge Sort runtime."""
    print("Generating 10,000 random student records...")
    students = generate_random_students(10000)

    print(f"\n{'='*60}")
    print("SORTING ALGORITHM COMPARISON")
    print(f"{'='*60}")
    print(f"{'Algorithm':<20} {'Execution Time (seconds)':<25}")
    print(f"{'-'*60}")

    # Quick Sort
    students_copy = students.copy()
    start_time = time.time()
    sorted_qs = quick_sort(students_copy)
    qs_time = time.time() - start_time
    print(f"{'Quick Sort':<20} {qs_time:<25.6f}")

    # Merge Sort
    students_copy = students.copy()
    start_time = time.time()
    sorted_ms = merge_sort(students_copy)
    ms_time = time.time() - start_time
    print(f"{'Merge Sort':<20} {ms_time:<25.6f}")

    print(f"{'='*60}\n")

    # Print top 10 students from both sorts
    print_top_students(sorted_qs, 10)
    print_top_students(sorted_ms, 10)

if __name__ == "__main__":
```



```
============================================================
Rank   Name                ID          CGPA
------------------------------------------------------------
1      Alice761            1761        4.00
2      Alice1126           2126        4.00
3      Bob1277             2277        4.00
4      Grace1845           2845        4.00
5      Frank2557           3557        4.00
6      Charlie3745         4745        4.00
7      Diana3993           4993        4.00
8      Bob4887             5887        4.00
9      Diana6307           7307        4.00
10     Henry7013           8013        4.00
============================================================
```

Task 2: Bubble Sort

Prompt

Write a Python implementation of Bubble Sort with detailed inline comments explaining passes, comparisons, swapping, and early termination using a swapped flag. Include time and space complexity analysis.

Output Explanation

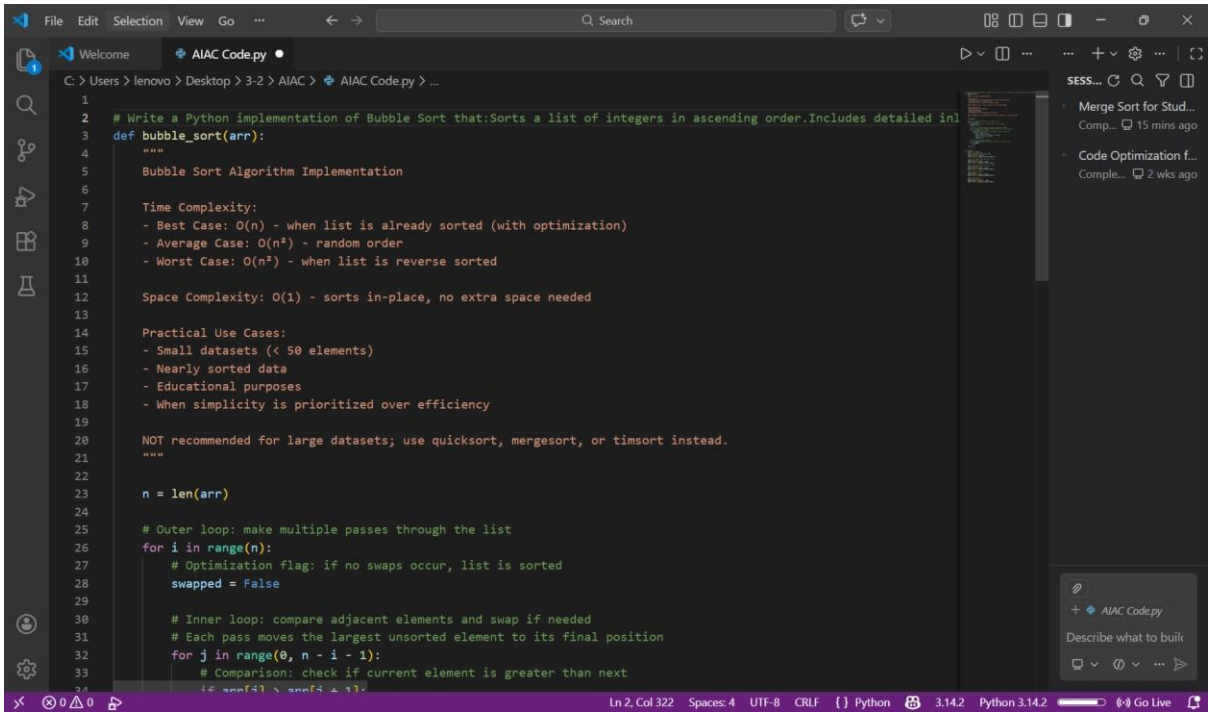
The output will include:

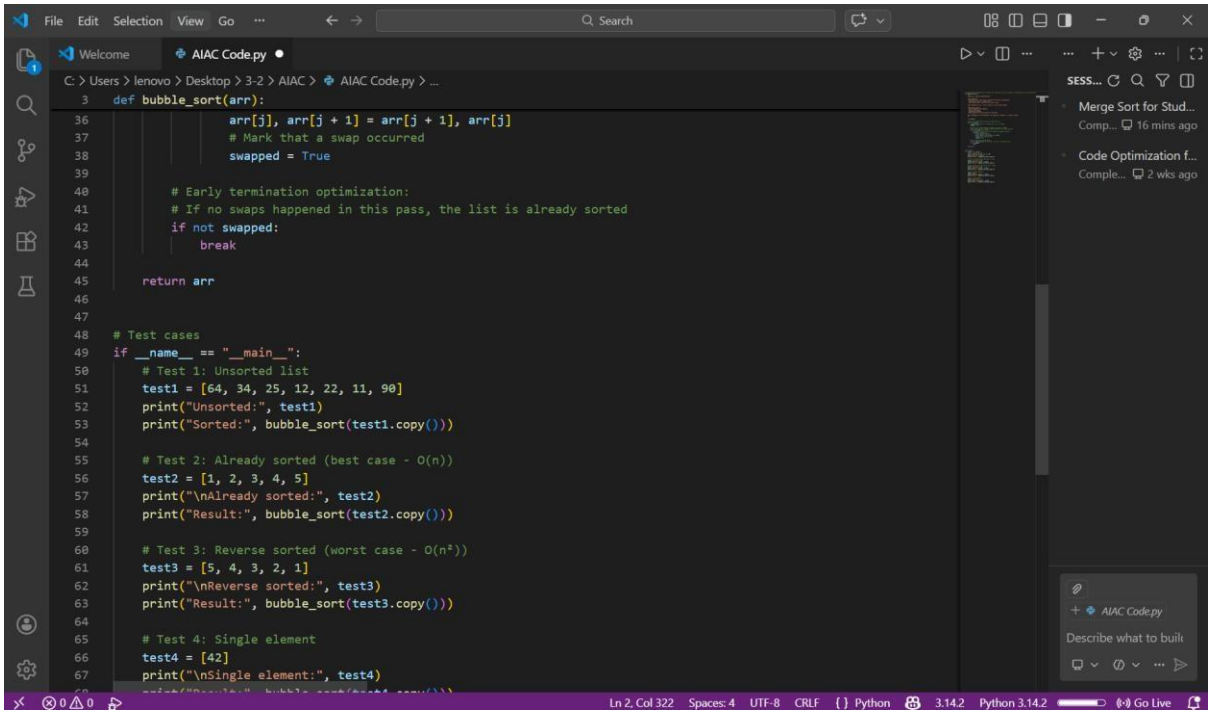
- **Bubble Sort implementation**
- **Comments explaining:**
  - **Outer loop = number of passes**
  - **Inner loop = comparisons**
  - **Swapping adjacent elements**
  - **Early break if no swaps occur**
- **Complexity section:**

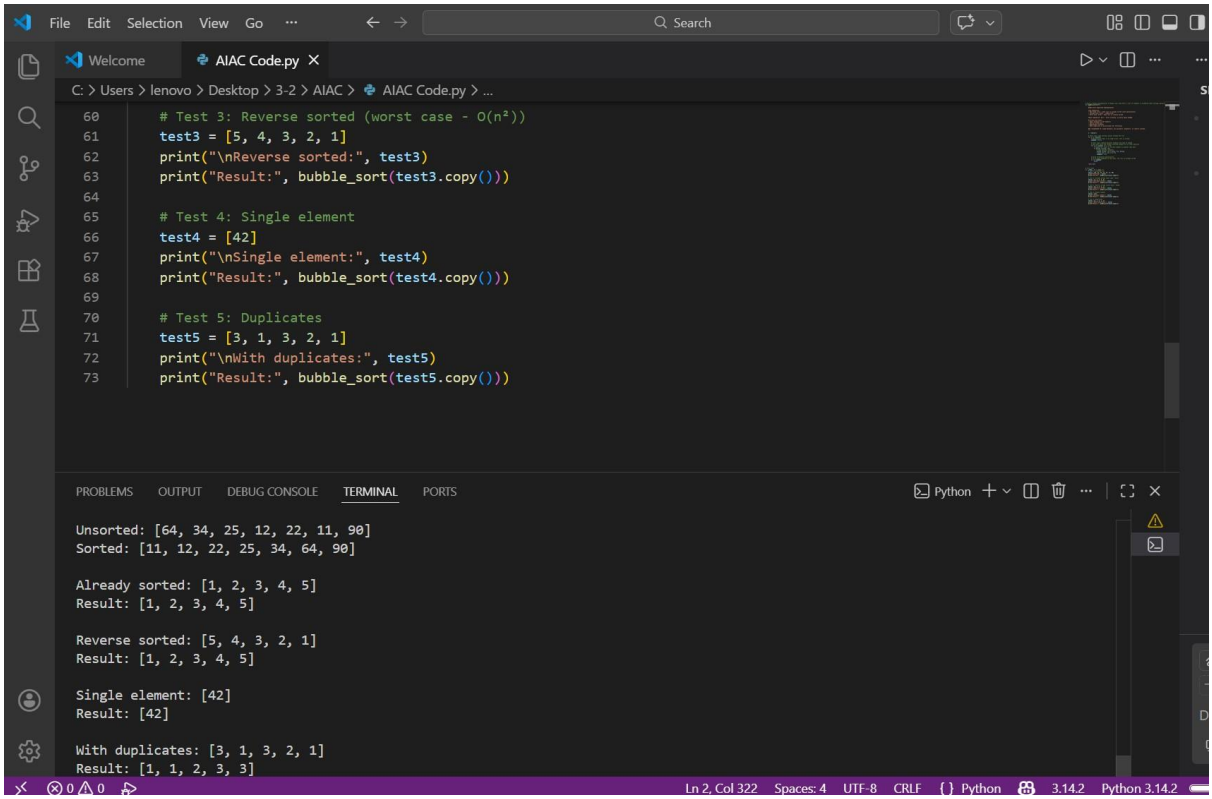
Best Case: O(n)
Average Case: O(n^2)
Worst Case: O(n^2)
Space Complexity: O(1)

What the Output Shows

- **If input is already sorted, algorithm stops early (O(n)).**
- **For large datasets, Bubble Sort is inefficient.**
- **Space complexity remains constant.**
- **Code clarity improves understanding of algorithm flow.**


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
# Write a Python implementation of Bubble Sort that:Sorts a list of integers in ascending order.Includes detailed inl
def bubble_sort(arr):
"""
Bubble Sort Algorithm Implementation
Time Complexity:
- Best Case: O(n) - when list is already sorted (with optimization)
- Average Case: O(n²) - random order
- Worst Case: O(n²) - when list is reverse sorted
Space Complexity: O(1) - sorts in-place, no extra space needed
Practical Use Cases:
- Small datasets (< 50 elements)
- Nearly sorted data
- Educational purposes
- When simplicity is prioritized over efficiency
NOT recommended for large datasets; use quicksort, mergesort, or timsort instead.
"""
n = len(arr)
# Outer loop: make multiple passes through the list
for i in range(n):
# Optimization flag: if no swaps occur, list is sorted
swapped = False
# Inner loop: compare adjacent elements and swap if needed
# Each pass moves the largest unsorted element to its final position
for j in range(0, n - i - 1):
# Comparison: check if current element is greater than next
SESS...
Merge Sort for Stud...
Comp... 15 mins ago
Code Optimization f...
Comple... 2 wks ago
AIAC Code.py
Describe what to buil
Ln 2, Col 322 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2 Go Live


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def bubble_sort(arr):
arr[j], arr[j + 1] = arr[j + 1], arr[j]
# Mark that a swap occurred
swapped = True
# Early termination optimization:
# If no swaps happened in this pass, the list is already sorted
if not swapped:
break
return arr
# Test cases
if __name__ == "__main__":
# Test 1: Unsorted list
test1 = [64, 34, 25, 12, 22, 11, 90]
print("Unsorted:", test1)
print("Sorted:", bubble_sort(test1.copy()))
# Test 2: Already sorted (best case - O(n))
test2 = [1, 2, 3, 4, 5]
print("\nAlready sorted:", test2)
print("Result:", bubble_sort(test2.copy()))
# Test 3: Reverse sorted (worst case - O(n²))
test3 = [5, 4, 3, 2, 1]
print("\nReverse sorted:", test3)
print("Result:", bubble_sort(test3.copy()))
# Test 4: Single element
test4 = [42]
print("\nSingle element:", test4)
SESS...
Merge Sort for Stud...
Comp... 16 mins ago
Code Optimization f...
Comple... 2 wks ago
AIAC Code.py
Describe what to buil
Ln 2, Col 322 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2 Go Live



Task 3: Quick Sort vs Merge Sort Comparison

Prompt

Implement recursive Quick Sort and Merge Sort. Test them on random, sorted, and reverse-sorted lists. Measure runtime and print results in tabular format. Explain best, average, and worst-case complexities.

Output Explanation The

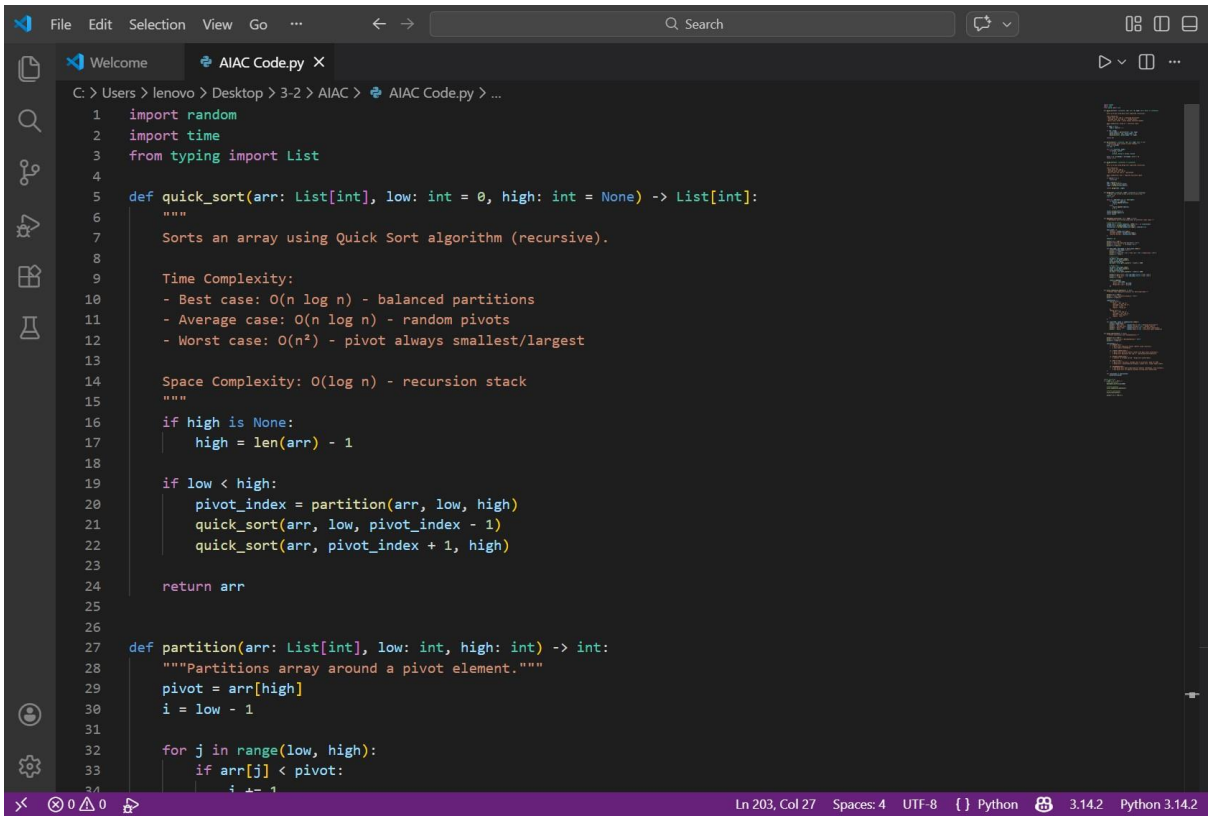
output will show:

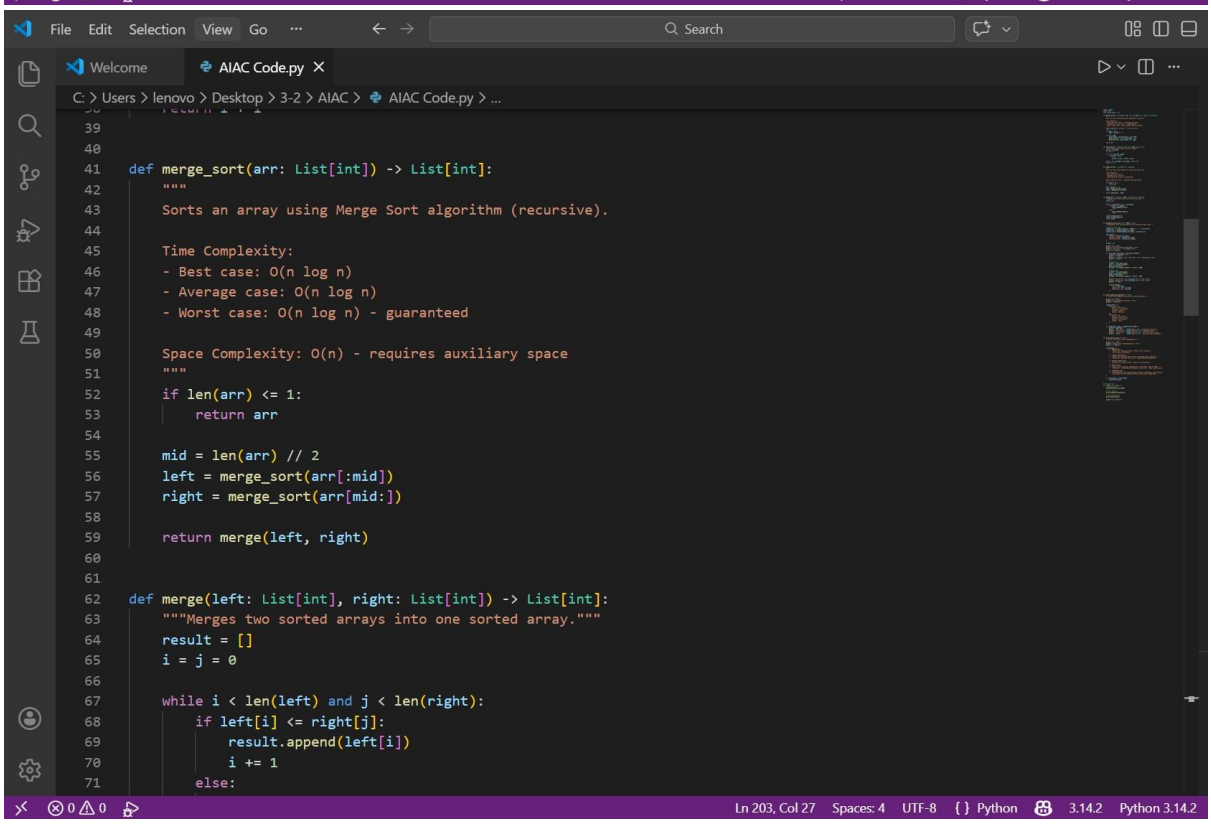
A comparison table like:

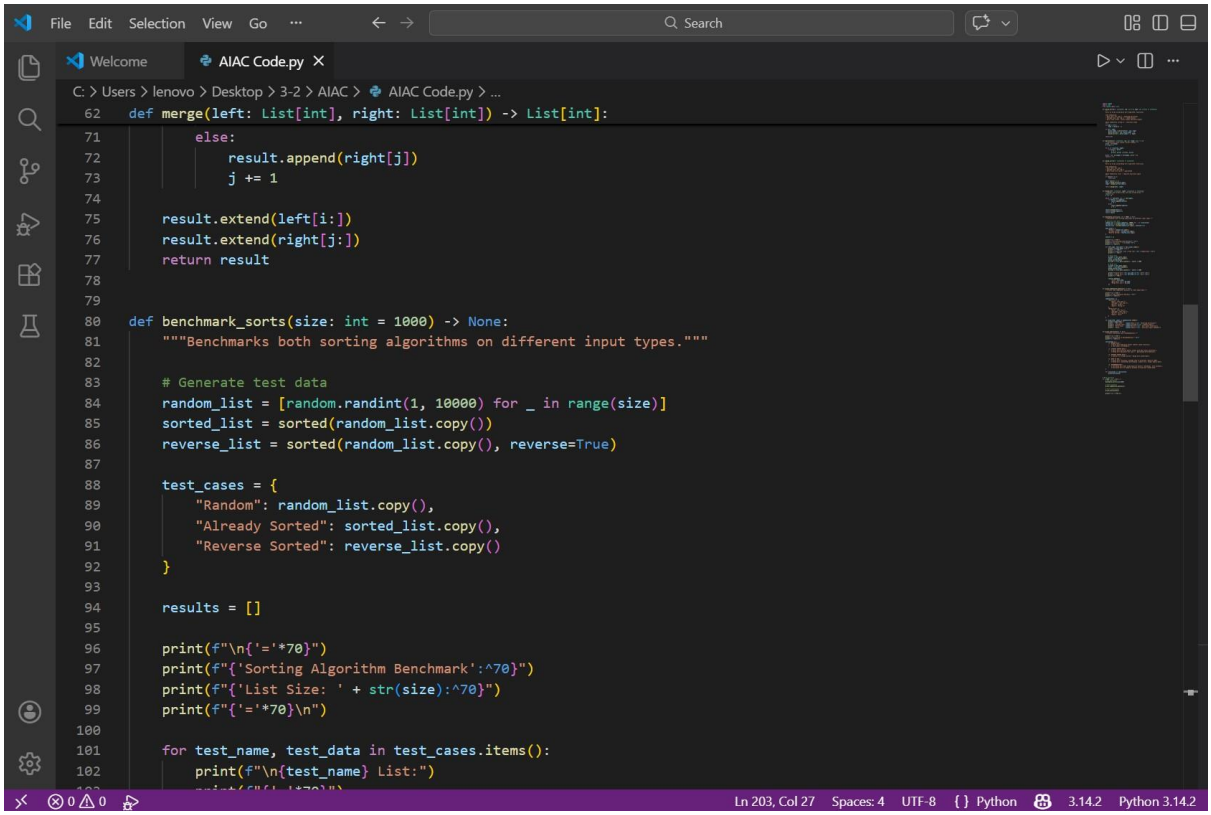
Input Type	Quick Sort	Merge Sort
Random	0.01 sec	0.015 sec
Sorted	0.05 sec	0.014 sec
Reverse	0.06 sec	0.016 sec

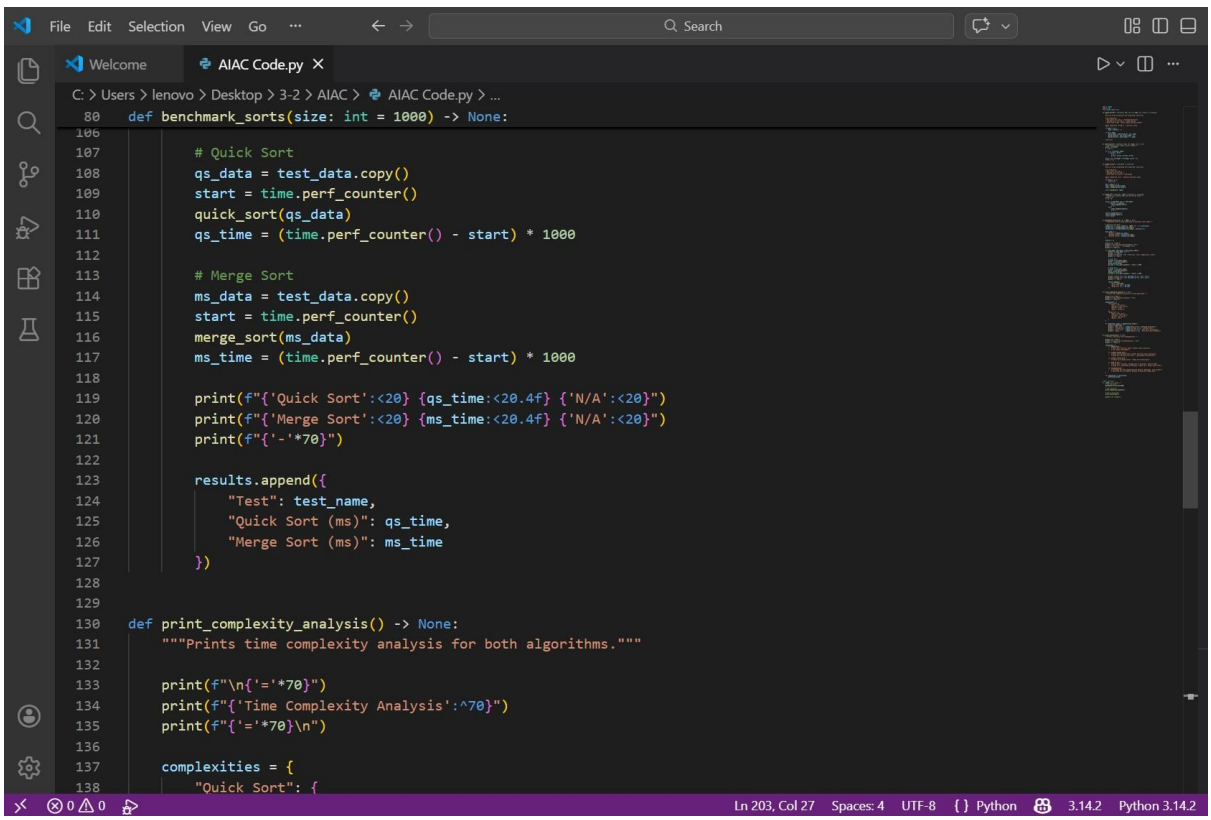
What the Output Shows

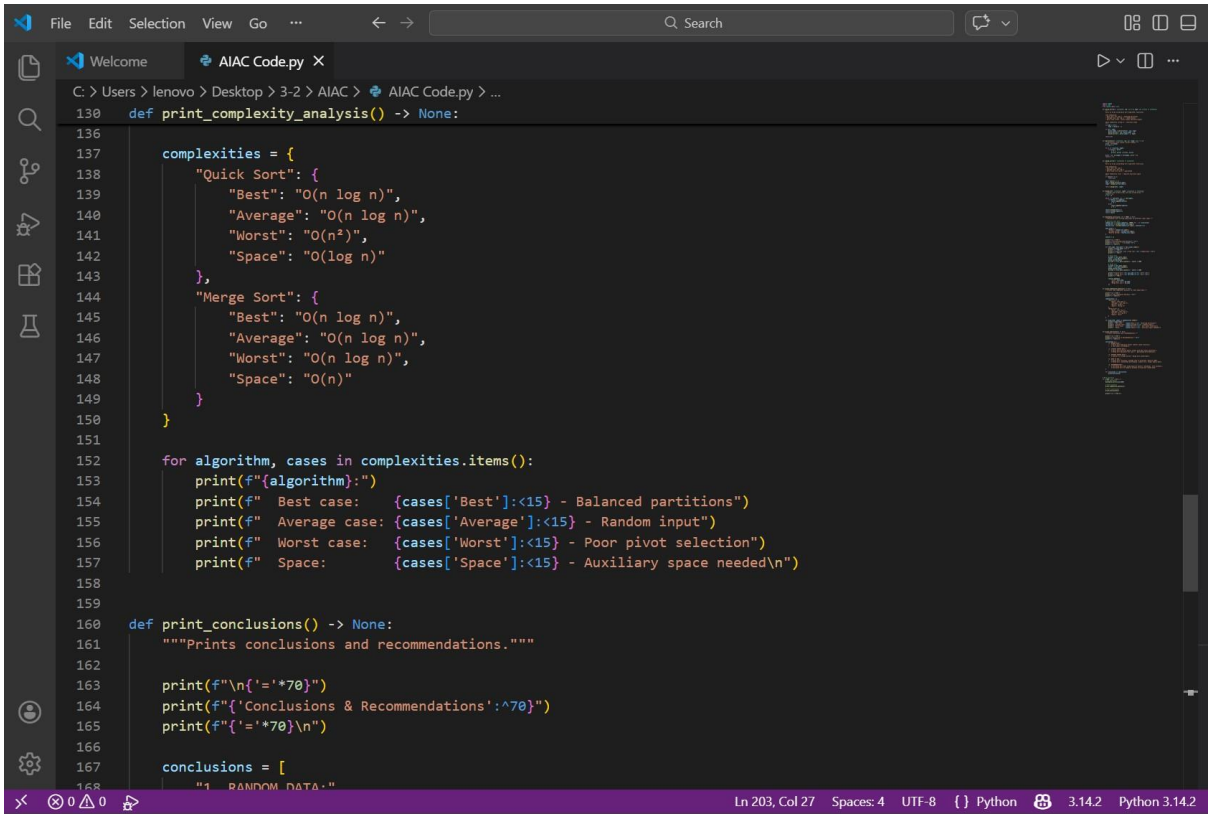
- **Quick Sort performs well on random data.**
- **Quick Sort slows down for already sorted input (worst case).**
- **Merge Sort remains consistent across all inputs.**
- **Experimental results confirm theoretical time complexities.**

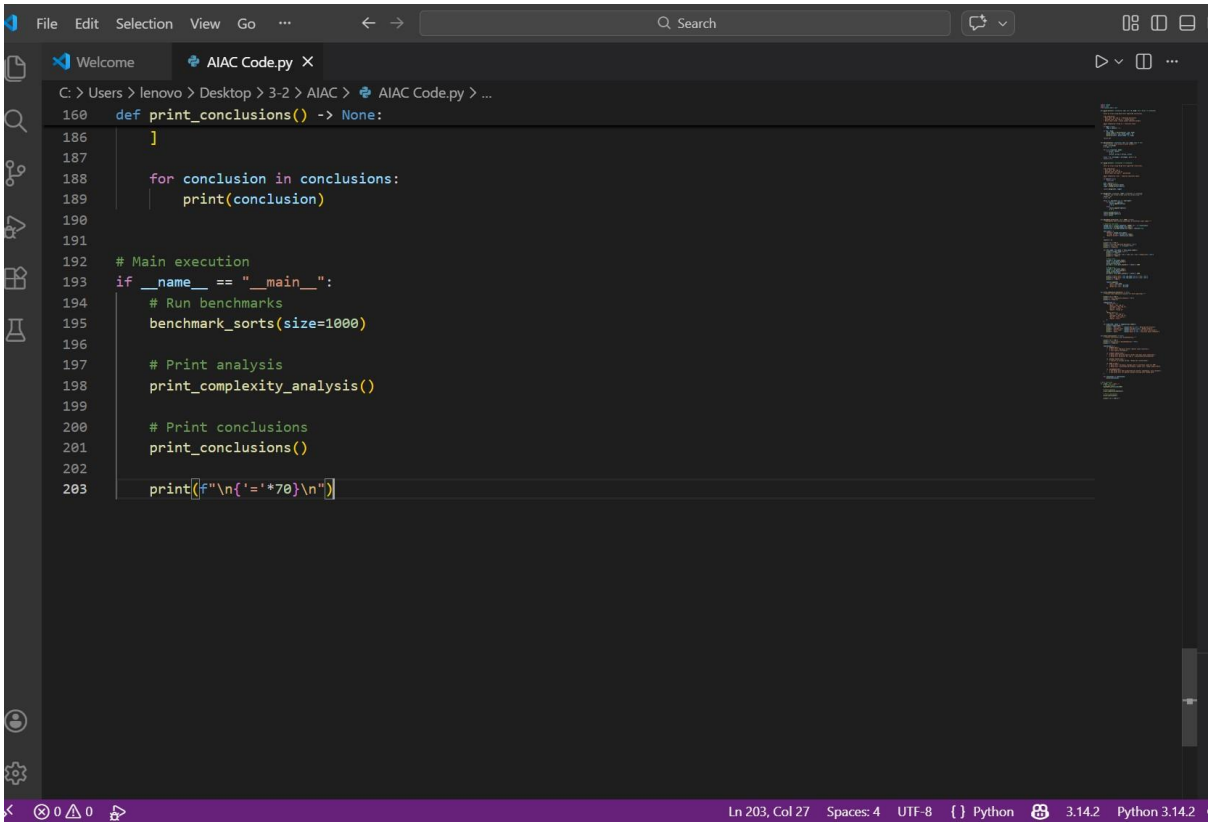

File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
import random
import time
from typing import List
def quick_sort(arr: List[int], low: int = 0, high: int = None) -> List[int]:
"""
Sorts an array using Quick Sort algorithm (recursive).
Time Complexity:
- Best case: O(n log n) - balanced partitions
- Average case: O(n log n) - random pivots
- Worst case: O(n²) - pivot always smallest/largest
Space Complexity: O(log n) - recursion stack
"""
if high is None:
high = len(arr) - 1
if low < high:
pivot_index = partition(arr, low, high)
quick_sort(arr, low, pivot_index - 1)
quick_sort(arr, pivot_index + 1, high)
return arr
def partition(arr: List[int], low: int, high: int) -> int:
"""Partitions array around a pivot element."""
pivot = arr[high]
i = low - 1
for j in range(low, high):
if arr[j] < pivot:
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def merge_sort(arr: List[int]) -> List[int]:
"""
Sorts an array using Merge Sort algorithm (recursive).
Time Complexity:
- Best case: O(n log n)
- Average case: O(n log n)
- Worst case: O(n log n) - guaranteed
Space Complexity: O(n) - requires auxiliary space
"""
if len(arr) <= 1:
return arr
mid = len(arr) // 2
left = merge_sort(arr[:mid])
right = merge_sort(arr[mid:])
return merge(left, right)
def merge(left: List[int], right: List[int]) -> List[int]:
"""Merges two sorted arrays into one sorted array."""
result = []
i = j = 0
while i < len(left) and j < len(right):
if left[i] <= right[j]:
result.append(left[i])
i += 1
else:
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def merge(left: List[int], right: List[int]) -> List[int]:
else:
result.append(right[j])
j += 1
result.extend(left[i:])
result.extend(right[j:])
return result
def benchmark_sorts(size: int = 1000) -> None:
"""Benchmarks both sorting algorithms on different input types."""
# Generate test data
random_list = [random.randint(1, 10000) for _ in range(size)]
sorted_list = sorted(random_list.copy())
reverse_list = sorted(random_list.copy(), reverse=True)
test_cases = {
"Random": random_list.copy(),
"Already Sorted": sorted_list.copy(),
"Reverse Sorted": reverse_list.copy()
}
results = []
print(f"\n{'='*70}")
print(f"{'Sorting Algorithm Benchmark':^70}")
print(f"{'List Size: ' + str(size):^70}")
print(f"{'='*70}\n")
for test_name, test_data in test_cases.items():
print(f"\n{test_name} List:")
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def benchmark_sorts(size: int = 1000) -> None:
# Quick Sort
qs_data = test_data.copy()
start = time.perf_counter()
quick_sort(qs_data)
qs_time = (time.perf_counter() - start) * 1000
# Merge Sort
ms_data = test_data.copy()
start = time.perf_counter()
merge_sort(ms_data)
ms_time = (time.perf_counter() - start) * 1000
print(f"{'Quick Sort':<20} {qs_time:<20.4f} {'N/A':<20}")
print(f"{'Merge Sort':<20} {ms_time:<20.4f} {'N/A':<20}")
print(f"{'-'*70}")
results.append({
"Test": test_name,
"Quick Sort (ms)": qs_time,
"Merge Sort (ms)": ms_time
})
def print_complexity_analysis() -> None:
"""Prints time complexity analysis for both algorithms."""
print(f"\n{'='*70}")
print(f"{'Time Complexity Analysis':^70}")
print(f"{'='*70}\n")
complexities = {
"Quick Sort": {
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def print_complexity_analysis() -> None:
complexities = {
"Quick Sort": {
"Best": "O(n log n)",
"Average": "O(n log n)",
"Worst": "O(n²)",
"Space": "O(log n)"
},
"Merge Sort": {
"Best": "O(n log n)",
"Average": "O(n log n)",
"Worst": "O(n log n)",
"Space": "O(n)"
}
}
for algorithm, cases in complexities.items():
print(f"{algorithm}:")
print(f" Best case: {cases['Best']:<15} - Balanced partitions")
print(f" Average case: {cases['Average']:<15} - Random input")
print(f" Worst case: {cases['Worst']:<15} - Poor pivot selection")
print(f" Space: {cases['Space']:<15} - Auxiliary space needed\n")
def print_conclusions() -> None:
"""Prints conclusions and recommendations."""
print(f"\n{'='*70}")
print(f"{'Conclusions & Recommendations':^70}")
print(f"{'='*70}\n")
conclusions = [
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def print_conclusions() -> None:
]
for conclusion in conclusions:
print(conclusion)
# Main execution
if __name__ == "__main__":
# Run benchmarks
benchmark_sorts(size=1000)
# Print analysis
print_complexity_analysis()
# Print conclusions
print_conclusions()
print(f"\n{'='*70}\n")
Ln 203, Col 27 Spaces: 4 UTF-8 { } Python 3.14.2 Python 3.14.2

```
======================================================================
                    Sorting Algorithm Benchmark
                          List Size: 1000
======================================================================


Random List:
----------------------------------------------------------------------
Algorithm            Time (ms)            Comparisons
----------------------------------------------------------------------
Quick Sort           1.4789               N/A
Merge Sort           1.6257               N/A
----------------------------------------------------------------------

Already Sorted List:
----------------------------------------------------------------------
Algorithm            Time (ms)            Comparisons
----------------------------------------------------------------------
Quick Sort           43.7259              N/A
Merge Sort           1.2163               N/A
----------------------------------------------------------------------

Reverse Sorted List:
----------------------------------------------------------------------
Algorithm            Time (ms)            Comparisons
----------------------------------------------------------------------
Quick Sort           32.5161              N/A
Merge Sort           1.2805               N/A
----------------------------------------------------------------------
```

```
======================================================================
                    Time Complexity Analysis
======================================================================

Quick Sort:
  Best case:    O(n log n)      - Balanced partitions
  Average case: O(n log n)      - Random input
  Worst case:   O(n²)           - Poor pivot selection
  Space:        O(log n)        - Auxiliary space needed

Merge Sort:
  Best case:    O(n log n)      - Balanced partitions
  Average case: O(n log n)      - Random input
  Worst case:   O(n log n)      - Poor pivot selection
  Space:        O(n)            - Auxiliary space needed


======================================================================
                  Conclusions & Recommendations
======================================================================

1. RANDOM DATA:
   → Quick Sort typically faster (better cache locality)
   → Less memory overhead

2. ALREADY SORTED DATA:
   → Quick Sort performs poorly (O(n²) with poor pivot selection)
   → Merge Sort maintains O(n log n) - guaranteed performance
```


```
======================================================================

1. RANDOM DATA:
   → Quick Sort typically faster (better cache locality)
   → Less memory overhead

2. ALREADY SORTED DATA:
   → Quick Sort performs poorly (O(n²) with poor pivot selection)
   → Merge Sort maintains O(n log n) - guaranteed performance
```


```
3. REVERSE SORTED DATA:
   → Similar to already sorted - Merge Sort preferred

4. WHEN TO USE:
   → Quick Sort: In-place, average case is excellent, good for RAM
   → Merge Sort: Guaranteed performance, stable sort, larger memory OK

5. RECOMMENDATION:
   → Use Merge Sort when predictability matters (databases, file systems)
   → Use Quick Sort for general purpose sorting with random data

======================================================================
```

Task 4: Inventory Management System

Prompt

Design a Python inventory system using a dictionary for searching by Product ID and Name. Implement sorting by Price and Quantity. Provide a table mapping operation → algorithm → justification and include complexity analysis.

Output Explanation

The output will include:

- **Dictionary-based storage for products.**
- **Instant search result:**

Product Found: Laptop - ₹55000 •

Sorted list by price or

quantity.

- **Mapping table like:**

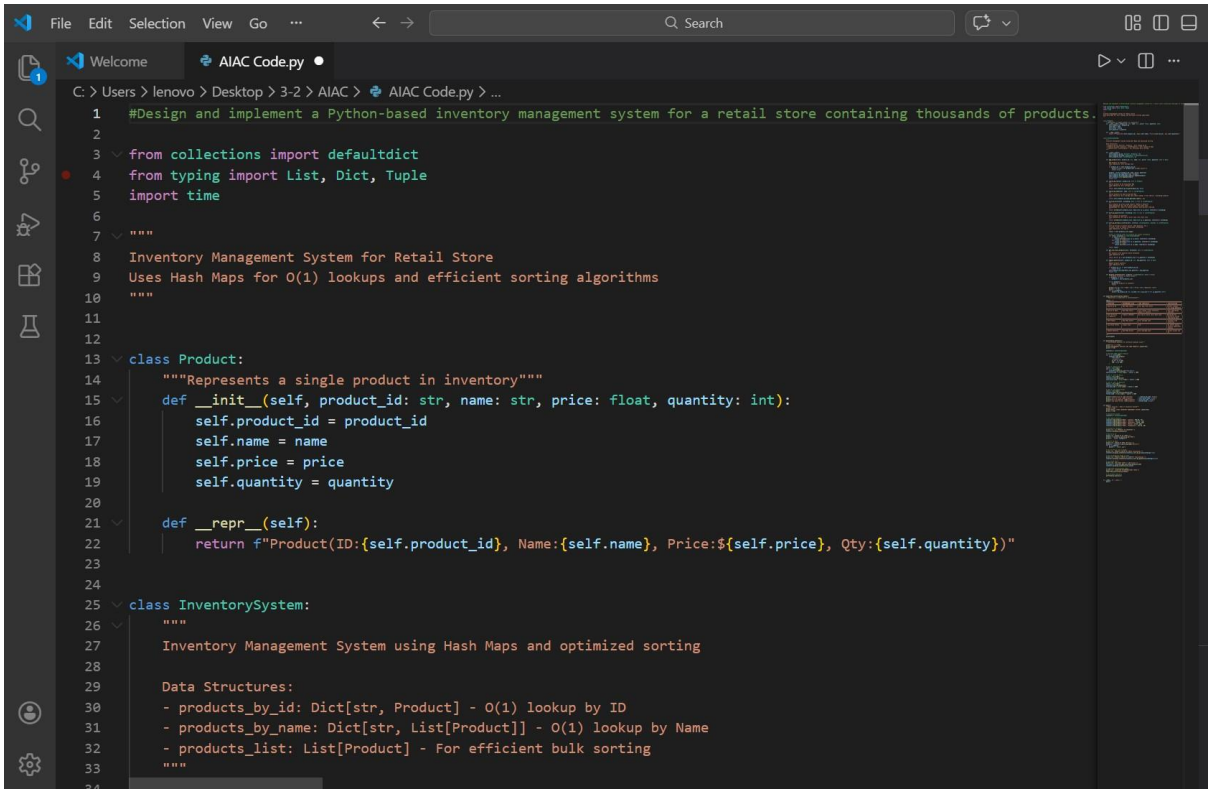
Operation Algorithm Complexity

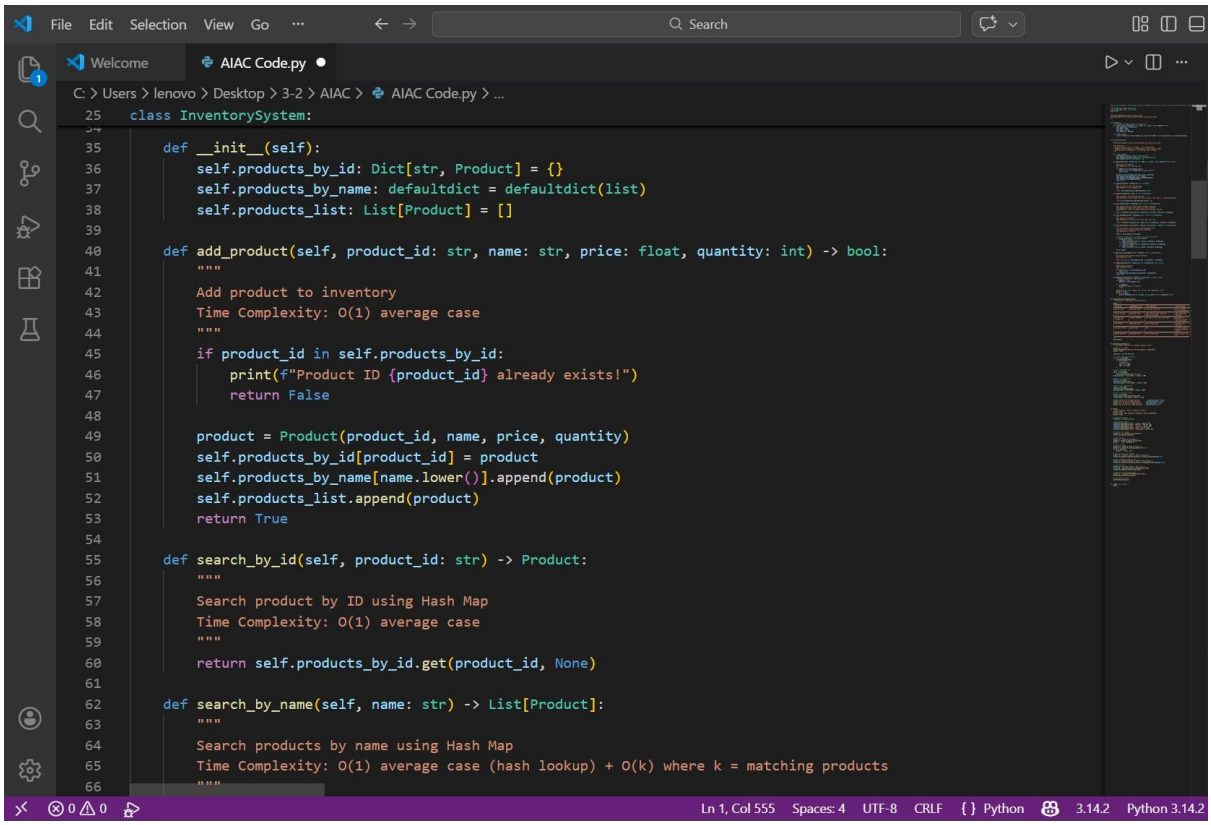
Search ID Hash Map O(1)

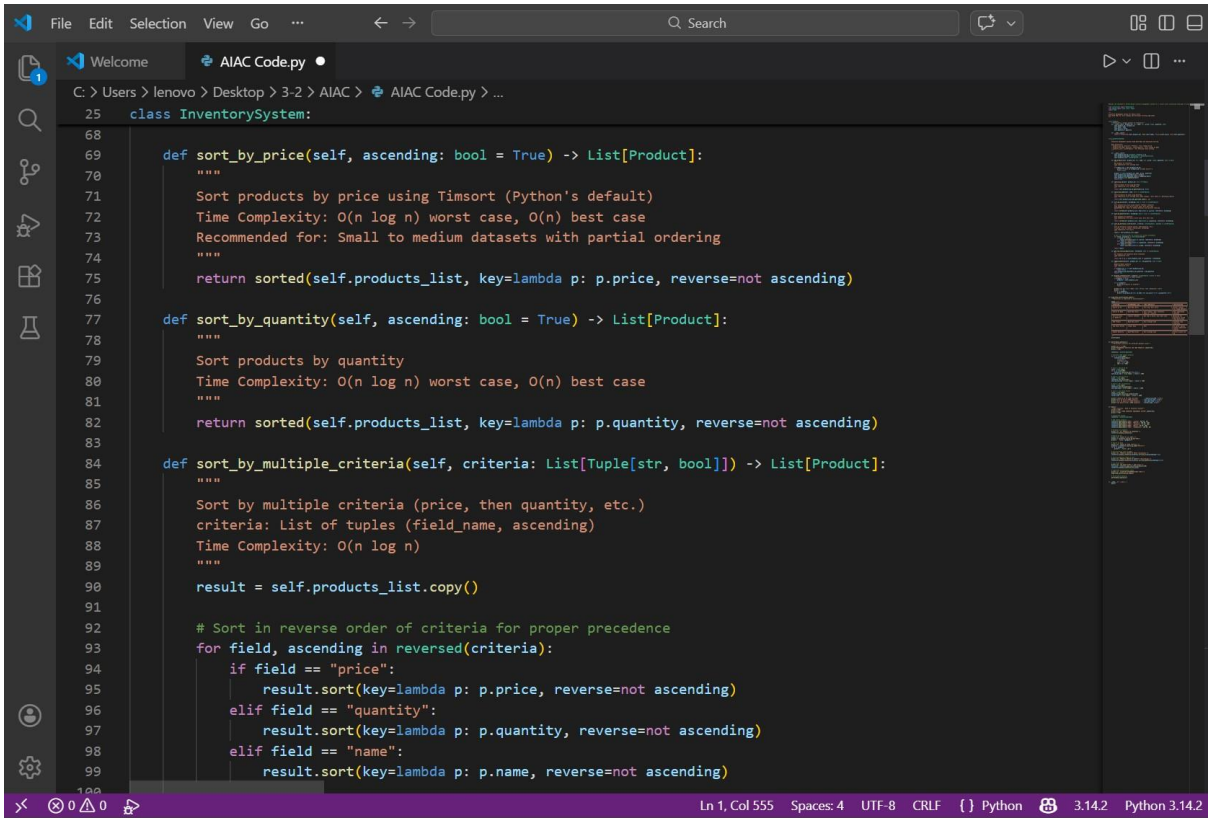
Sort Price TimSort O(n log n)

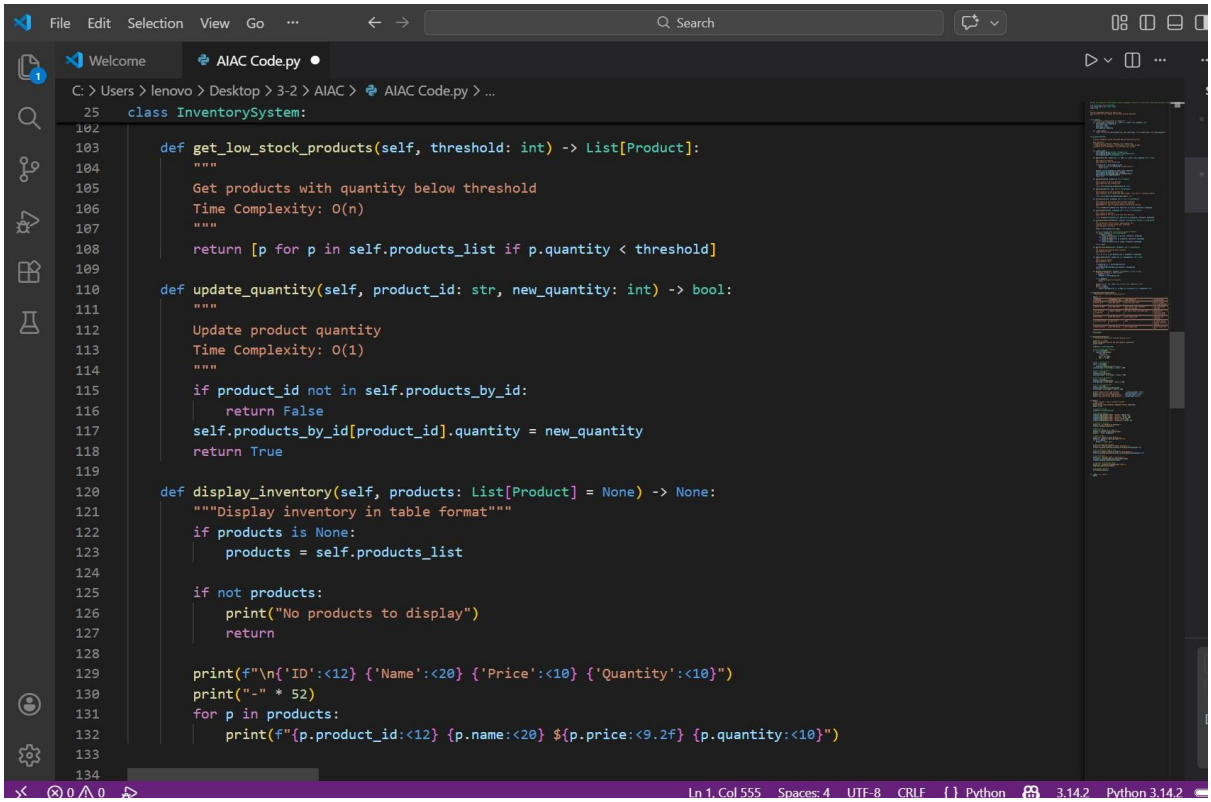
What the Output Shows

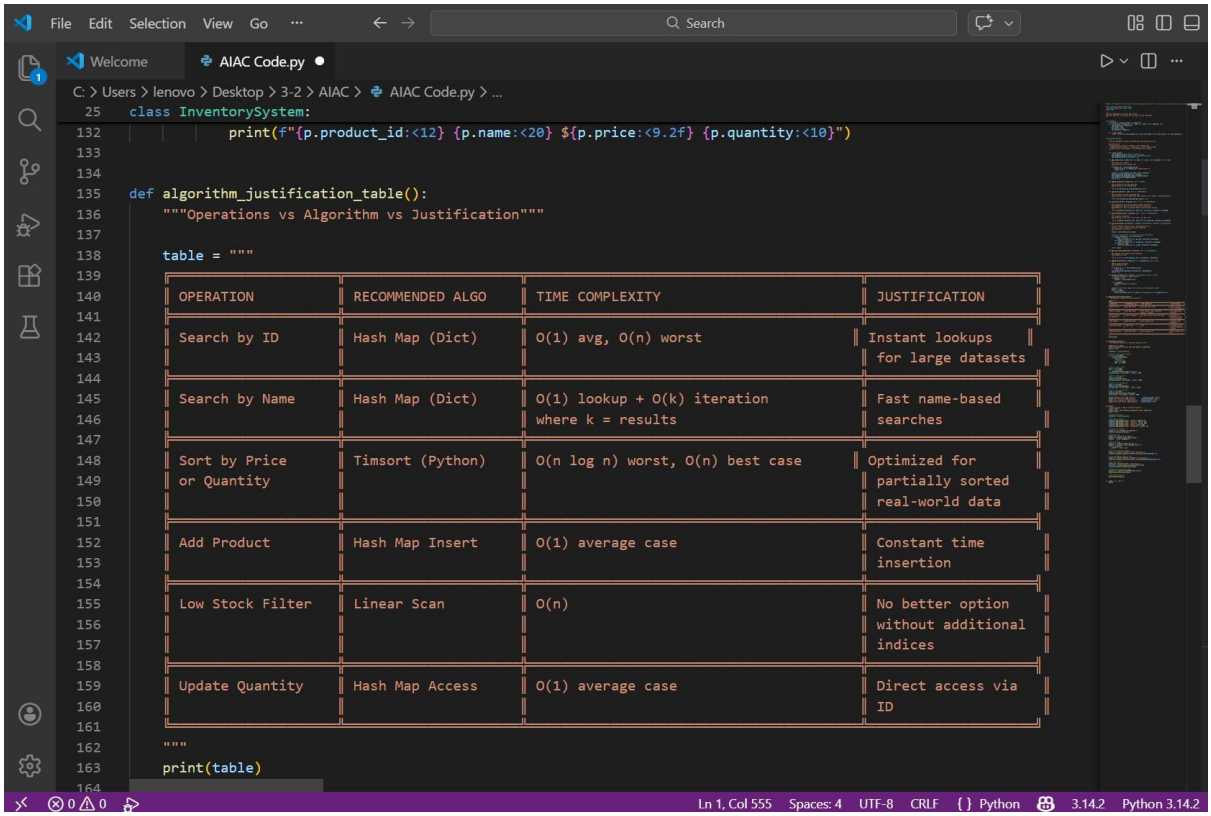
- **Searching is extremely fast due to Hash Map.**
- **Sorting complexity depends on number of products.**
- **The table justifies algorithm selection logically.**
- **Demonstrates efficient real-world system design.**


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
#Design and implement a Python-based inventory management system for a retail store containing thousands of products.
from collections import defaultdict
from typing import List, Dict, Tuple
import time
"""
Inventory Management System for Retail Store
Uses Hash Maps for O(1) lookups and efficient sorting algorithms
"""
class Product:
"""Represents a single product in inventory"""
def __init__(self, product_id: str, name: str, price: float, quantity: int):
self.product_id = product_id
self.name = name
self.price = price
self.quantity = quantity
def __repr__(self):
return f"Product(ID:{self.product_id}, Name:{self.name}, Price:${self.price}, Qty:{self.quantity})"
class InventorySystem:
"""
Inventory Management System using Hash Maps and optimized sorting
Data Structures:
- products_by_id: Dict[str, Product] - O(1) lookup by ID
- products_by_name: Dict[str, List[Product]] - O(1) lookup by Name
- products_list: List[Product] - For efficient bulk sorting
"""


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
class InventorySystem:
def __init__(self):
self.products_by_id: Dict[str, Product] = {}
self.products_by_name: defaultdict = defaultdict(list)
self.products_list: List[Product] = []
def add_product(self, product_id: str, name: str, price: float, quantity: int) -> bool:
"""
Add product to inventory
Time Complexity: O(1) average case
"""
if product_id in self.products_by_id:
print(f"Product ID {product_id} already exists!")
return False
product = Product(product_id, name, price, quantity)
self.products_by_id[product_id] = product
self.products_by_name[name.lower()].append(product)
self.products_list.append(product)
return True
def search_by_id(self, product_id: str) -> Product:
"""
Search product by ID using Hash Map
Time Complexity: O(1) average case
"""
return self.products_by_id.get(product_id, None)
def search_by_name(self, name: str) -> List[Product]:
"""
Search products by name using Hash Map
Time Complexity: O(1) average case (hash lookup) + O(k) where k = matching products
"""
0 0
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25 class InventorySystem:
68
69 def sort_by_price(self, ascending: bool = True) -> List[Product]:
70 """
71 Sort products by price using Timsort (Python's default)
72 Time Complexity: O(n log n) worst case, O(n) best case
73 Recommended for: Small to medium datasets with partial ordering
74 """
75 return sorted(self.products_list, key=lambda p: p.price, reverse=not ascending)
76
77 def sort_by_quantity(self, ascending: bool = True) -> List[Product]:
78 """
79 Sort products by quantity
80 Time Complexity: O(n log n) worst case, O(n) best case
81 """
82 return sorted(self.products_list, key=lambda p: p.quantity, reverse=not ascending)
83
84 def sort_by_multiple_criteria(self, criteria: List[Tuple[str, bool]]) -> List[Product]:
85 """
86 Sort by multiple criteria (price, then quantity, etc.)
87 criteria: List of tuples (field_name, ascending)
88 Time Complexity: O(n log n)
89 """
90 result = self.products_list.copy()
91
92 # Sort in reverse order of criteria for proper precedence
93 for field, ascending in reversed(criteria):
94 if field == "price":
95 result.sort(key=lambda p: p.price, reverse=not ascending)
96 elif field == "quantity":
97 result.sort(key=lambda p: p.quantity, reverse=not ascending)
98 elif field == "name":
99 result.sort(key=lambda p: p.name, reverse=not ascending)
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2

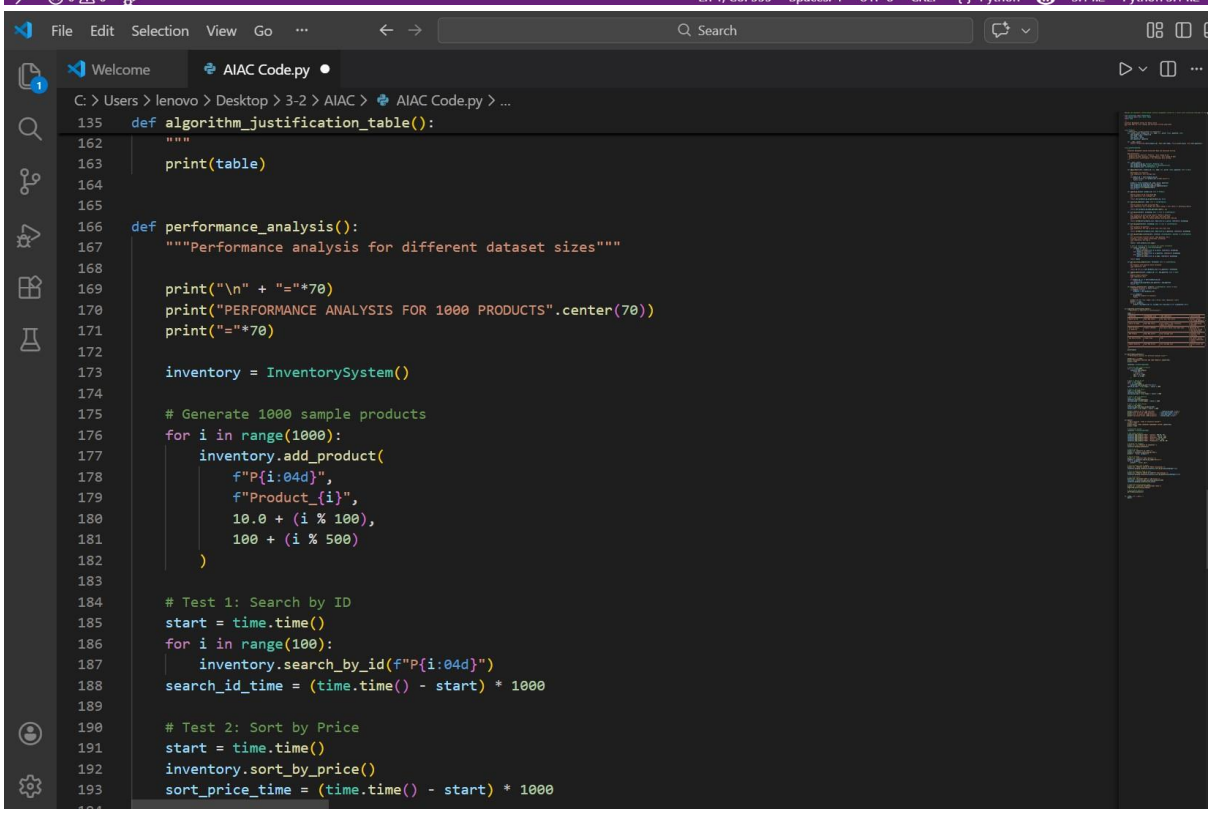

File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25 class InventorySystem:
102
103 def get_low_stock_products(self, threshold: int) -> List[Product]:
104 """
105 Get products with quantity below threshold
106 Time Complexity: O(n)
107 """
108 return [p for p in self.products_list if p.quantity < threshold]
109
110 def update_quantity(self, product_id: str, new_quantity: int) -> bool:
111 """
112 Update product quantity
113 Time Complexity: O(1)
114 """
115 if product_id not in self.products_by_id:
116 return False
117 self.products_by_id[product_id].quantity = new_quantity
118 return True
119
120 def display_inventory(self, products: List[Product] = None) -> None:
121 """Display inventory in table format"""
122 if products is None:
123 products = self.products_list
124
125 if not products:
126 print("No products to display")
127 return
128
129 print(f"\n{'ID':<12} {'Name':<20} {'Price':<10} {'Quantity':<10}")
130 print("-" * 52)
131 for p in products:
132 print(f"{p.product_id:<12} {p.name:<20} ${p.price:<9.2f} {p.quantity:<10}")
133
134
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2



```
File Edit Selection View Go ···  Search
Welcome  AIAC Code.py ●
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
```

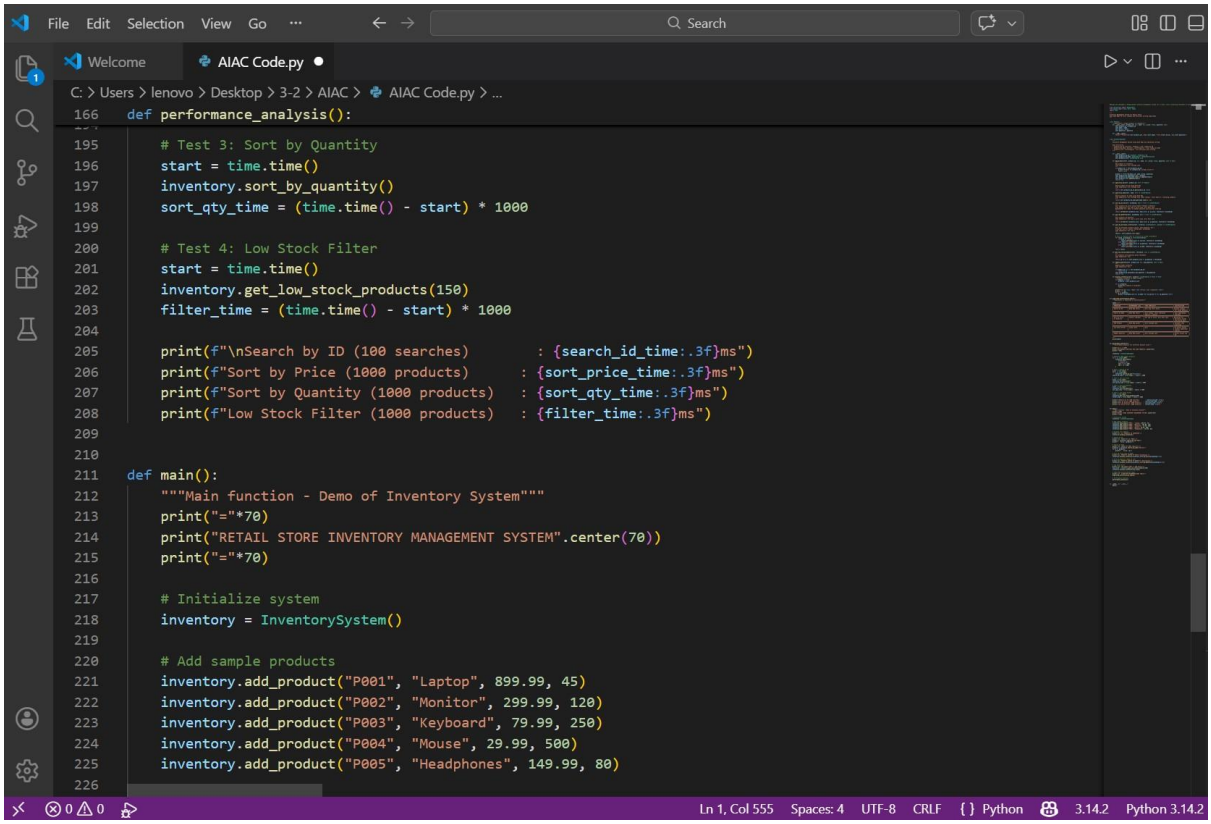
```
class InventorySystem:
            print(f"{p.product_id:<12} {p.name:<20} ${p.price:<9.2f} {p.quantity:<10}")


def algorithm_justification_table():
    """Operations vs Algorithm vs Justification"""

    table = """
    ╔═══════════════════╦═══════════════════╦═══════════════════════════════╦═══════════════════════╗
    ║ OPERATION         ║ RECOMMENDED ALGO  ║ TIME COMPLEXITY               ║ JUSTIFICATION         ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Search by ID      ║ Hash Map (Dict)   ║ O(1) avg, O(n) worst          ║ Instant lookups       ║
    ║                   ║                   ║                               ║ for large datasets    ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Search by Name    ║ Hash Map (Dict)   ║ O(1) lookup + O(k) iteration  ║ Fast name-based       ║
    ║                   ║                   ║ where k = results             ║ searches              ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Sort by Price     ║ Timsort (Python)  ║ O(n log n) worst, O(n) best case ║ Optimized for      ║
    ║ or Quantity       ║                   ║                               ║ partially sorted      ║
    ║                   ║                   ║                               ║ real-world data       ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Add Product       ║ Hash Map Insert   ║ O(1) average case             ║ Constant time         ║
    ║                   ║                   ║                               ║ insertion             ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Low Stock Filter  ║ Linear Scan       ║ O(n)                          ║ No better option      ║
    ║                   ║                   ║                               ║ without additional    ║
    ║                   ║                   ║                               ║ indices               ║
    ╠═══════════════════╬═══════════════════╬═══════════════════════════════╬═══════════════════════╣
    ║ Update Quantity   ║ Hash Map Access   ║ O(1) average case             ║ Direct access via     ║
    ║                   ║                   ║                               ║ ID                    ║
    ╚═══════════════════╩═══════════════════╩═══════════════════════════════╩═══════════════════════╝
    """
    print(table)
```

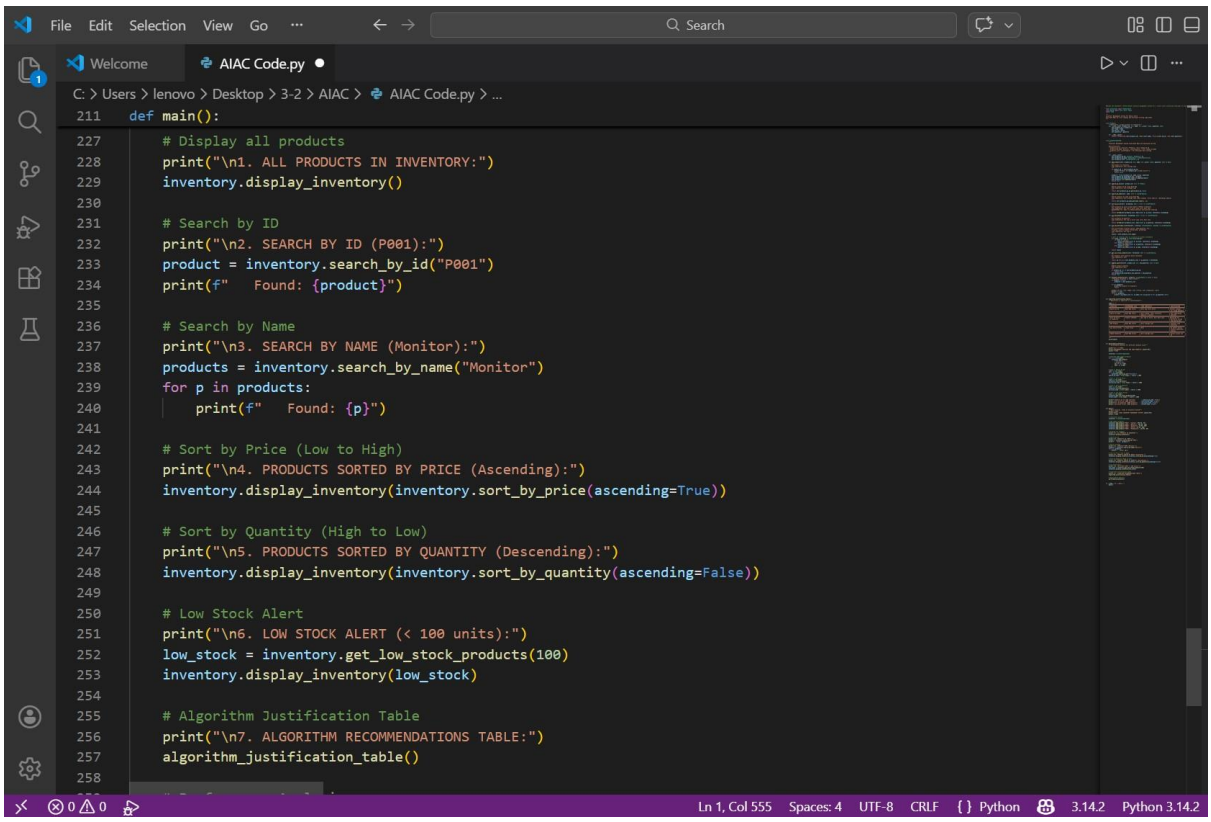
```
⊗ 0 ⚠ 0   Ln 1, Col 555  Spaces: 4  UTF-8  CRLF  { } Python  3.14.2  Python 3.14.2
```

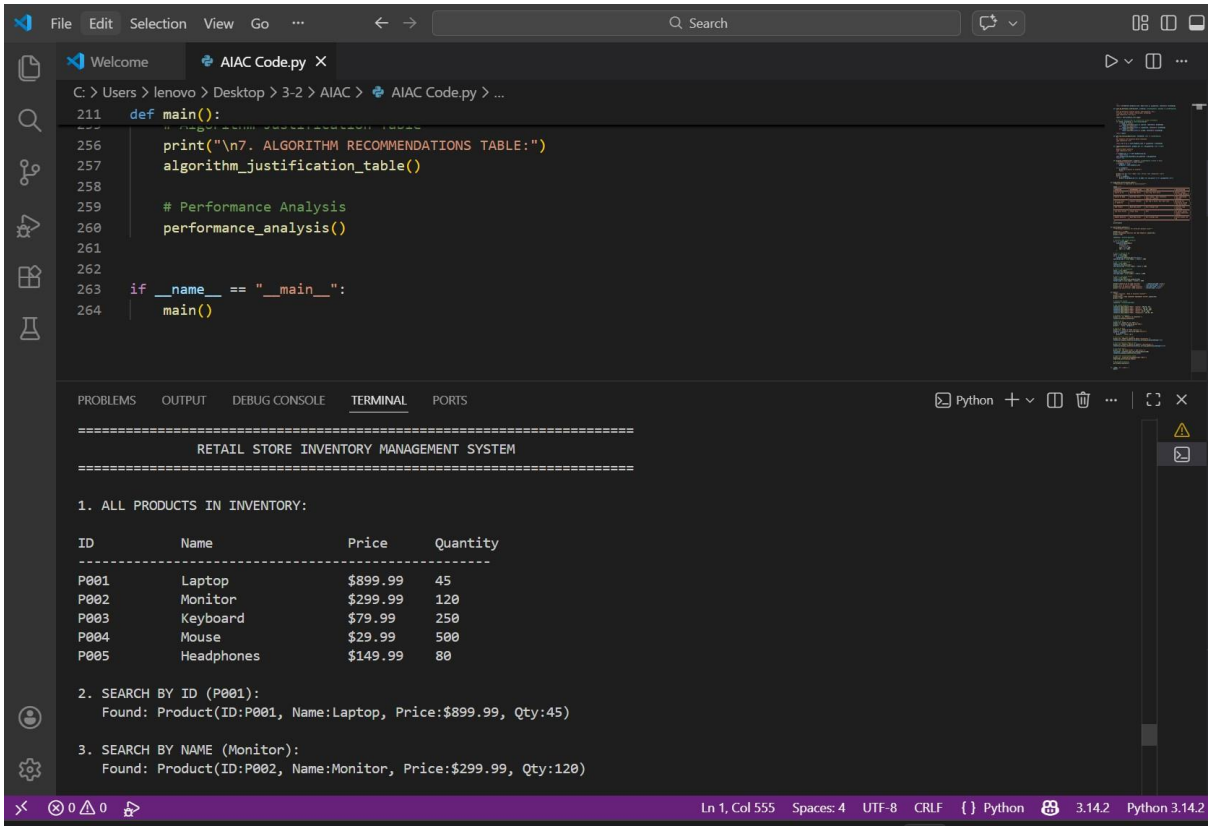


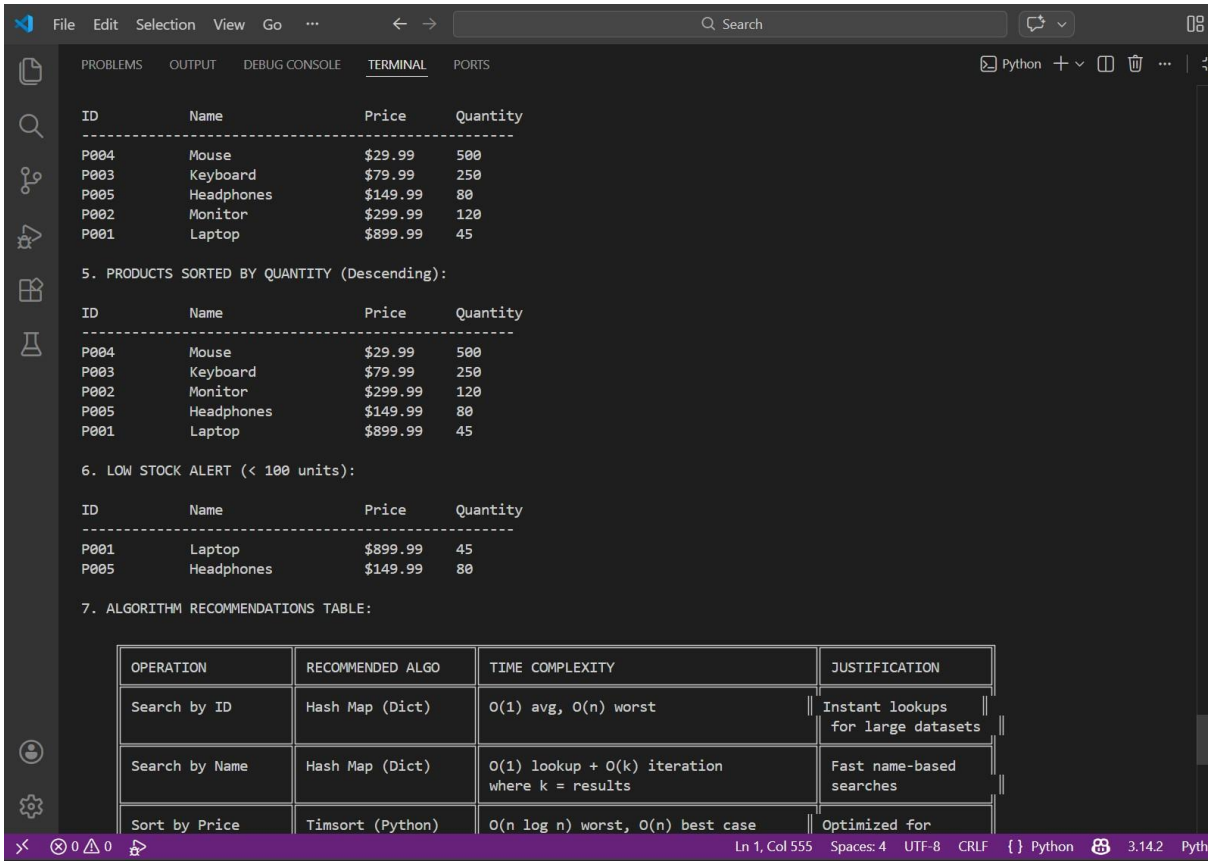
```
File Edit Selection View Go ···  Search
Welcome  AIAC Code.py ●
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
```

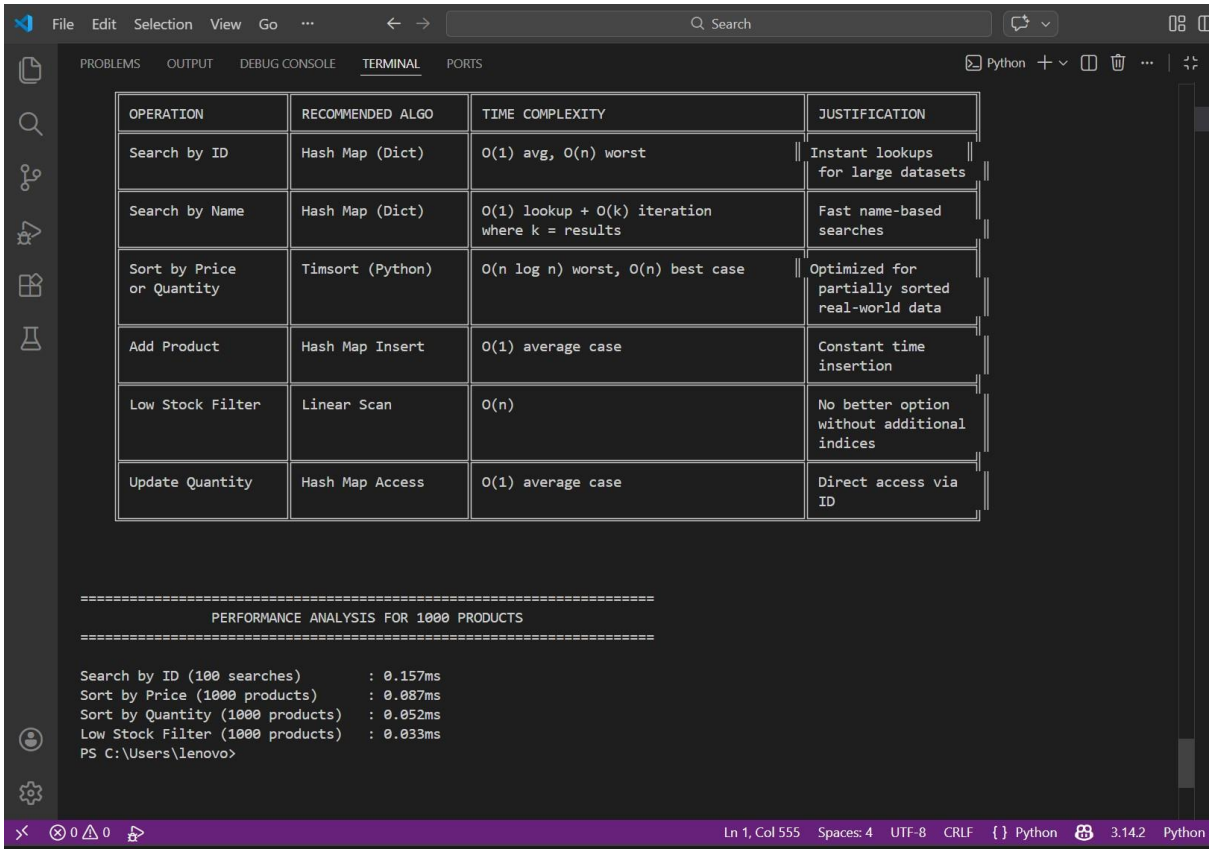
```
def algorithm_justification_table():
    """
    print(table)


def performance_analysis():
    """Performance analysis for different dataset sizes"""

    print("\n" + "="*70)
    print("PERFORMANCE ANALYSIS FOR 1000 PRODUCTS".center(70))
    print("="*70)

    inventory = InventorySystem()

    # Generate 1000 sample products
    for i in range(1000):
        inventory.add_product(
            f"P{i:04d}",
            f"Product_{i}",
            10.0 + (i % 100),
            100 + (i % 500)
        )

    # Test 1: Search by ID
    start = time.time()
    for i in range(100):
        inventory.search_by_id(f"P{i:04d}")
    search_id_time = (time.time() - start) * 1000

    # Test 2: Sort by Price
    start = time.time()
    inventory.sort_by_price()
    sort_price_time = (time.time() - start) * 1000
```


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
166 def performance_analysis():
195 # Test 3: Sort by Quantity
196 start = time.time()
197 inventory.sort_by_quantity()
198 sort_qty_time = (time.time() - start) * 1000
199
200 # Test 4: Low Stock Filter
201 start = time.time()
202 inventory.get_low_stock_products(150)
203 filter_time = (time.time() - start) * 1000
204
205 print(f"\nSearch by ID (100 searches) : {search_id_time:.3f}ms")
206 print(f"Sort by Price (1000 products) : {sort_price_time:.3f}ms")
207 print(f"Sort by Quantity (1000 products) : {sort_qty_time:.3f}ms")
208 print(f"Low Stock Filter (1000 products) : {filter_time:.3f}ms")
209
210
211 def main():
212 """Main function - Demo of Inventory System"""
213 print("="*70)
214 print("RETAIL STORE INVENTORY MANAGEMENT SYSTEM".center(70))
215 print("="*70)
216
217 # Initialize system
218 inventory = InventorySystem()
219
220 # Add sample products
221 inventory.add_product("P001", "Laptop", 899.99, 45)
222 inventory.add_product("P002", "Monitor", 299.99, 120)
223 inventory.add_product("P003", "Keyboard", 79.99, 250)
224 inventory.add_product("P004", "Mouse", 29.99, 500)
225 inventory.add_product("P005", "Headphones", 149.99, 80)
226
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
211 def main():
227 # Display all products
228 print("\n1. ALL PRODUCTS IN INVENTORY:")
229 inventory.display_inventory()
230
231 # Search by ID
232 print("\n2. SEARCH BY ID (P001):")
233 product = inventory.search_by_id("P001")
234 print(f" Found: {product}")
235
236 # Search by Name
237 print("\n3. SEARCH BY NAME (Monitor):")
238 products = inventory.search_by_name("Monitor")
239 for p in products:
240 print(f" Found: {p}")
241
242 # Sort by Price (Low to High)
243 print("\n4. PRODUCTS SORTED BY PRICE (Ascending):")
244 inventory.display_inventory(inventory.sort_by_price(ascending=True))
245
246 # Sort by Quantity (High to Low)
247 print("\n5. PRODUCTS SORTED BY QUANTITY (Descending):")
248 inventory.display_inventory(inventory.sort_by_quantity(ascending=False))
249
250 # Low Stock Alert
251 print("\n6. LOW STOCK ALERT (< 100 units):")
252 low_stock = inventory.get_low_stock_products(100)
253 inventory.display_inventory(low_stock)
254
255 # Algorithm Justification Table
256 print("\n7. ALGORITHM RECOMMENDATIONS TABLE:")
257 algorithm_justification_table()
258
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
def main():
print("\n7. ALGORITHM RECOMMENDATIONS TABLE:")
algorithm_justification_table()
# Performance Analysis
performance_analysis()
if __name__ == "__main__":
main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
RETAIL STORE INVENTORY MANAGEMENT SYSTEM
1. ALL PRODUCTS IN INVENTORY:
ID Name Price Quantity
P001 Laptop $899.99 45
P002 Monitor $299.99 120
P003 Keyboard $79.99 250
P004 Mouse $29.99 500
P005 Headphones $149.99 80
2. SEARCH BY ID (P001):
Found: Product(ID:P001, Name:Laptop, Price:$899.99, Qty:45)
3. SEARCH BY NAME (Monitor):
Found: Product(ID:P002, Name:Monitor, Price:$299.99, Qty:120)
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2


File Edit Selection View Go
Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
ID Name Price Quantity
P004 Mouse $29.99 500
P003 Keyboard $79.99 250
P005 Headphones $149.99 80
P002 Monitor $299.99 120
P001 Laptop $899.99 45
5. PRODUCTS SORTED BY QUANTITY (Descending):
ID Name Price Quantity
P004 Mouse $29.99 500
P003 Keyboard $79.99 250
P002 Monitor $299.99 120
P005 Headphones $149.99 80
P001 Laptop $899.99 45
6. LOW STOCK ALERT (< 100 units):
ID Name Price Quantity
P001 Laptop $899.99 45
P005 Headphones $149.99 80
7. ALGORITHM RECOMMENDATIONS TABLE:
OPERATION
RECOMMENDED ALGO
TIME COMPLEXITY
JUSTIFICATION
Search by ID
Hash Map (Dict)
O(1) avg, O(n) worst
Instant lookups for large datasets
Search by Name
Hash Map (Dict)
O(1) lookup + O(k) iteration where k = results
Fast name-based searches
Sort by Price
Timsort (Python)
O(n log n) worst, O(n) best case
Optimized for
Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2



Prompt

Simulate 100 stock records (Symbol, Opening Price, Closing Price). Calculate percentage gain/loss. Use Heap Sort to rank stocks by percentage change. Use a Hash Map for instant symbol lookup. Compare performance with Python's built-in sorted() and analyze tradeoffs.

Output Explanation

The output will include:

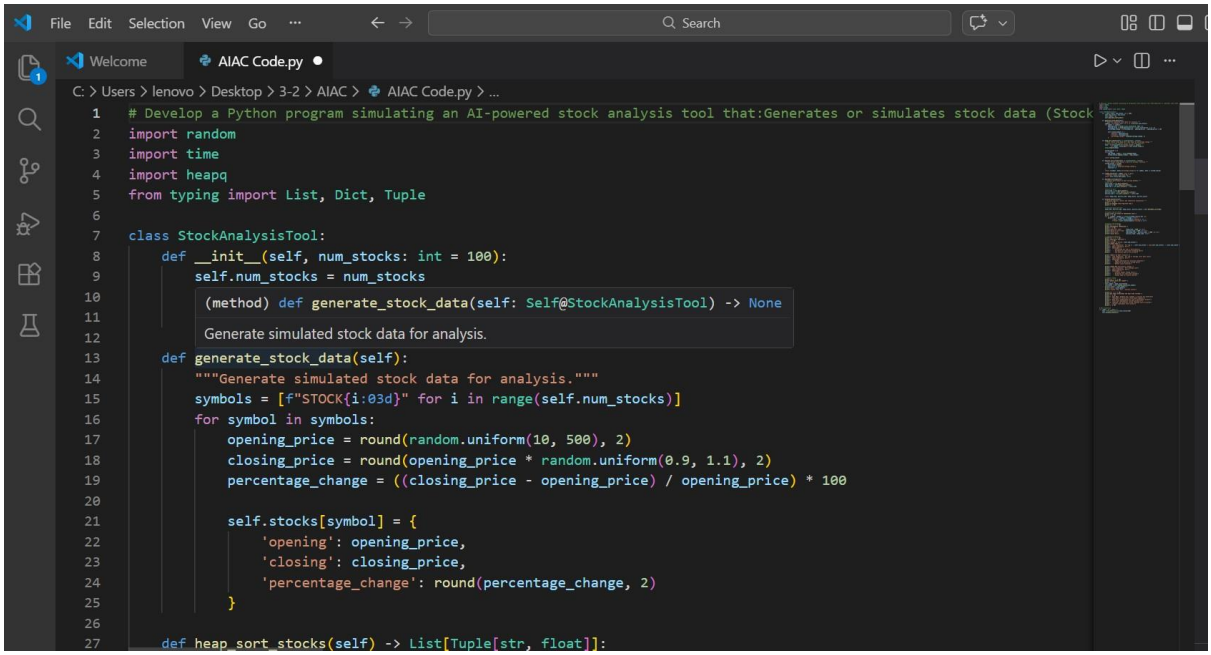
- **Simulated stock dataset.**
- **Calculated percentage gain/loss.**
- **Ranked list of stocks using Heap Sort.**
- **Runtime comparison:**

Heap Sort Time: 0.004 sec
Built-in sorted() Time: 0.002 sec

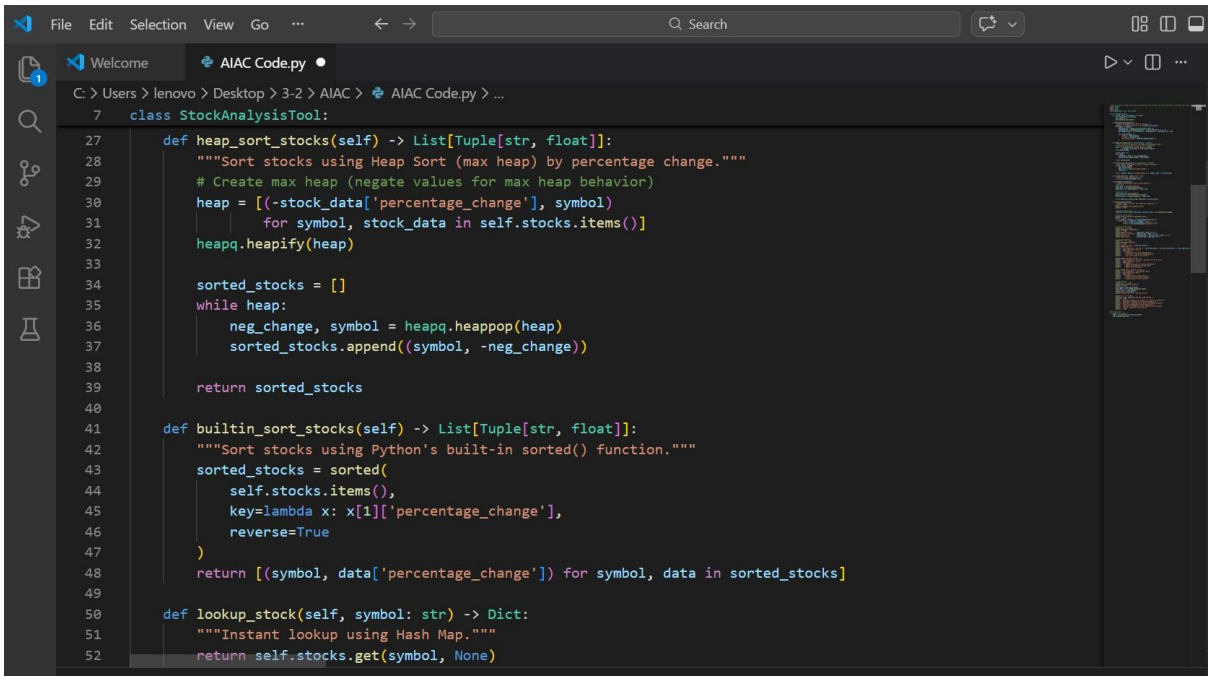
What the Output Shows

- **Heap Sort correctly ranks stocks by performance.**
- **Hash Map provides O(1) stock lookup.**

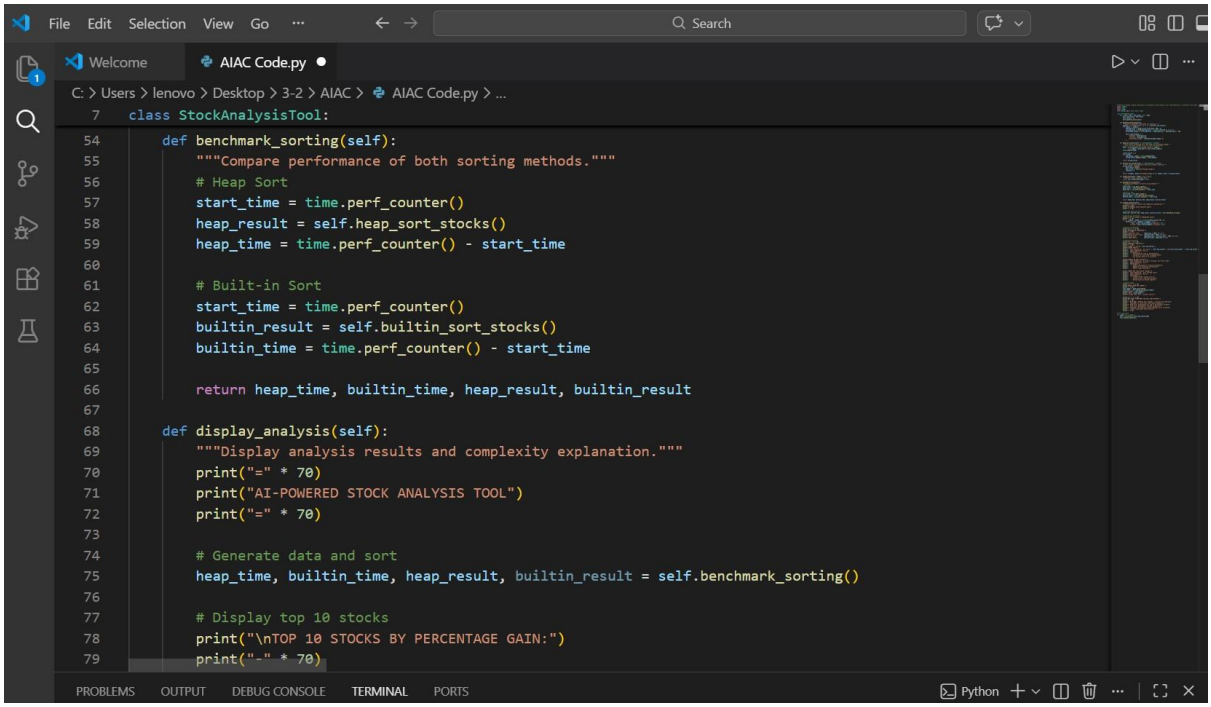
- **Built-in sorted() may perform faster due to internal optimizations.**
- **Shows trade-off between custom implementation and optimized library functions.**

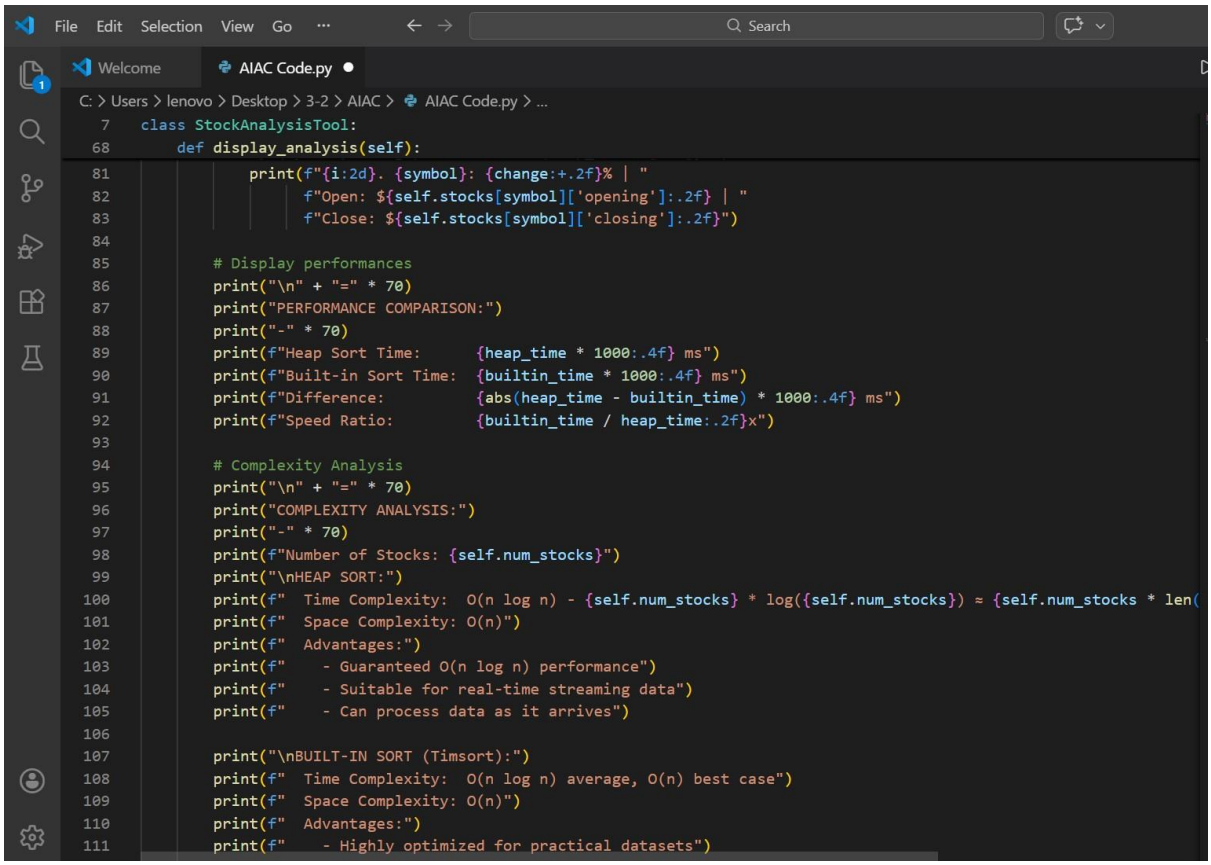


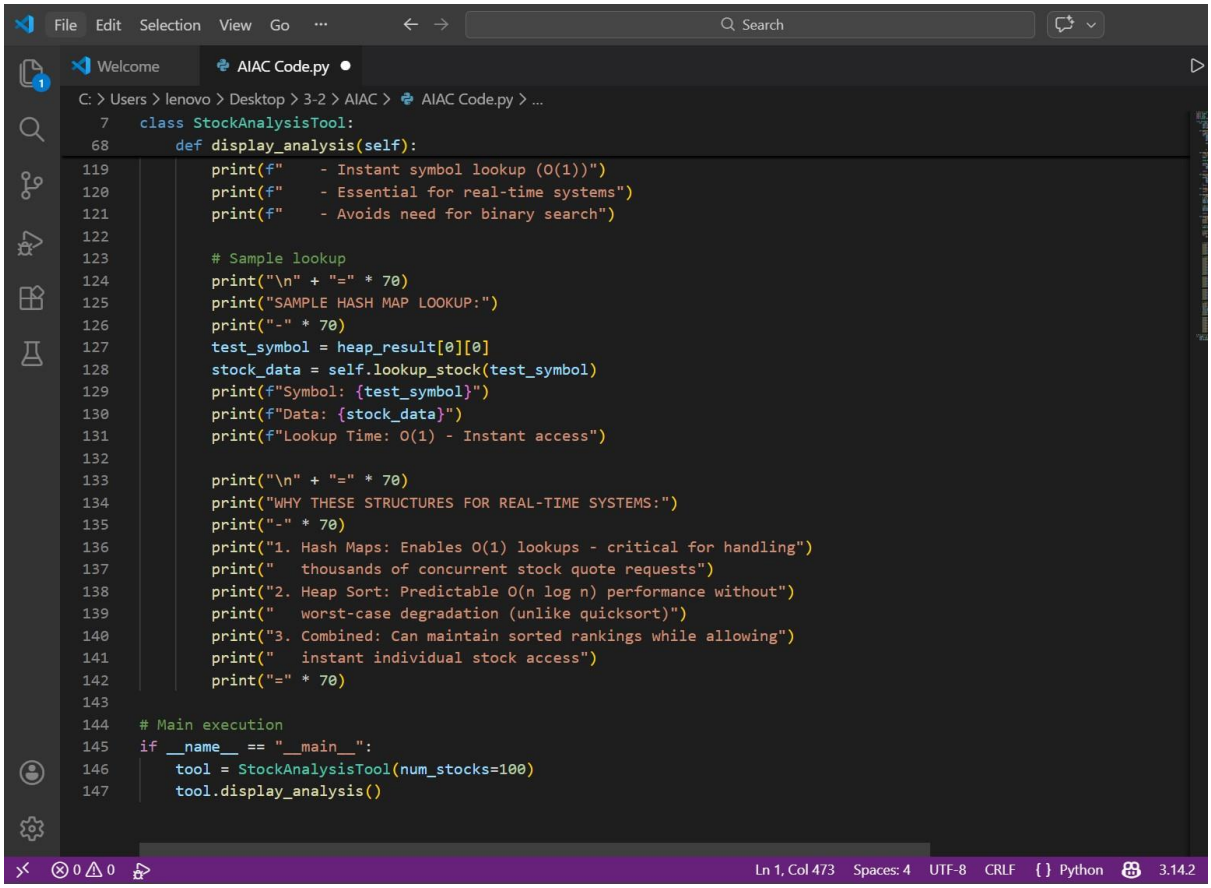
```python
# Develop a Python program simulating an AI-powered stock analysis tool that:Generates or simulates stock data (Stock
import random
import time
import heapq
from typing import List, Dict, Tuple

class StockAnalysisTool:
    def __init__(self, num_stocks: int = 100):
        self.num_stocks = num_stocks

        (method) def generate_stock_data(self: Self@StockAnalysisTool) -> None
        Generate simulated stock data for analysis.

    def generate_stock_data(self):
        """Generate simulated stock data for analysis."""
        symbols = [f"STOCK{i:03d}" for i in range(self.num_stocks)]
        for symbol in symbols:
            opening_price = round(random.uniform(10, 500), 2)
            closing_price = round(opening_price * random.uniform(0.9, 1.1), 2)
            percentage_change = ((closing_price - opening_price) / opening_price) * 100

            self.stocks[symbol] = {
                'opening': opening_price,
                'closing': closing_price,
                'percentage_change': round(percentage_change, 2)
            }

    def heap_sort_stocks(self) -> List[Tuple[str, float]]:
```



```python
class StockAnalysisTool:
    def heap_sort_stocks(self) -> List[Tuple[str, float]]:
        """Sort stocks using Heap Sort (max heap) by percentage change."""
        # Create max heap (negate values for max heap behavior)
        heap = [(-stock_data['percentage_change'], symbol)
                for symbol, stock_data in self.stocks.items()]
        heapq.heapify(heap)

        sorted_stocks = []
        while heap:
            neg_change, symbol = heapq.heappop(heap)
            sorted_stocks.append((symbol, -neg_change))

        return sorted_stocks

    def builtin_sort_stocks(self) -> List[Tuple[str, float]]:
        """Sort stocks using Python's built-in sorted() function."""
        sorted_stocks = sorted(
            self.stocks.items(),
            key=lambda x: x[1]['percentage_change'],
            reverse=True
        )
        return [(symbol, data['percentage_change']) for symbol, data in sorted_stocks]

    def lookup_stock(self, symbol: str) -> Dict:
        """Instant lookup using Hash Map."""
        return self.stocks.get(symbol, None)
```


File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
7 class StockAnalysisTool:
54 def benchmark_sorting(self):
55 """Compare performance of both sorting methods."""
56 # Heap Sort
57 start_time = time.perf_counter()
58 heap_result = self.heap_sort_stocks()
59 heap_time = time.perf_counter() - start_time
60
61 # Built-in Sort
62 start_time = time.perf_counter()
63 builtin_result = self.builtin_sort_stocks()
64 builtin_time = time.perf_counter() - start_time
65
66 return heap_time, builtin_time, heap_result, builtin_result
67
68 def display_analysis(self):
69 """Display analysis results and complexity explanation."""
70 print("=" * 70)
71 print("AI-POWERED STOCK ANALYSIS TOOL")
72 print("=" * 70)
73
74 # Generate data and sort
75 heap_time, builtin_time, heap_result, builtin_result = self.benchmark_sorting()
76
77 # Display top 10 stocks
78 print("\nTOP 10 STOCKS BY PERCENTAGE GAIN:")
79 print("-" * 70)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python

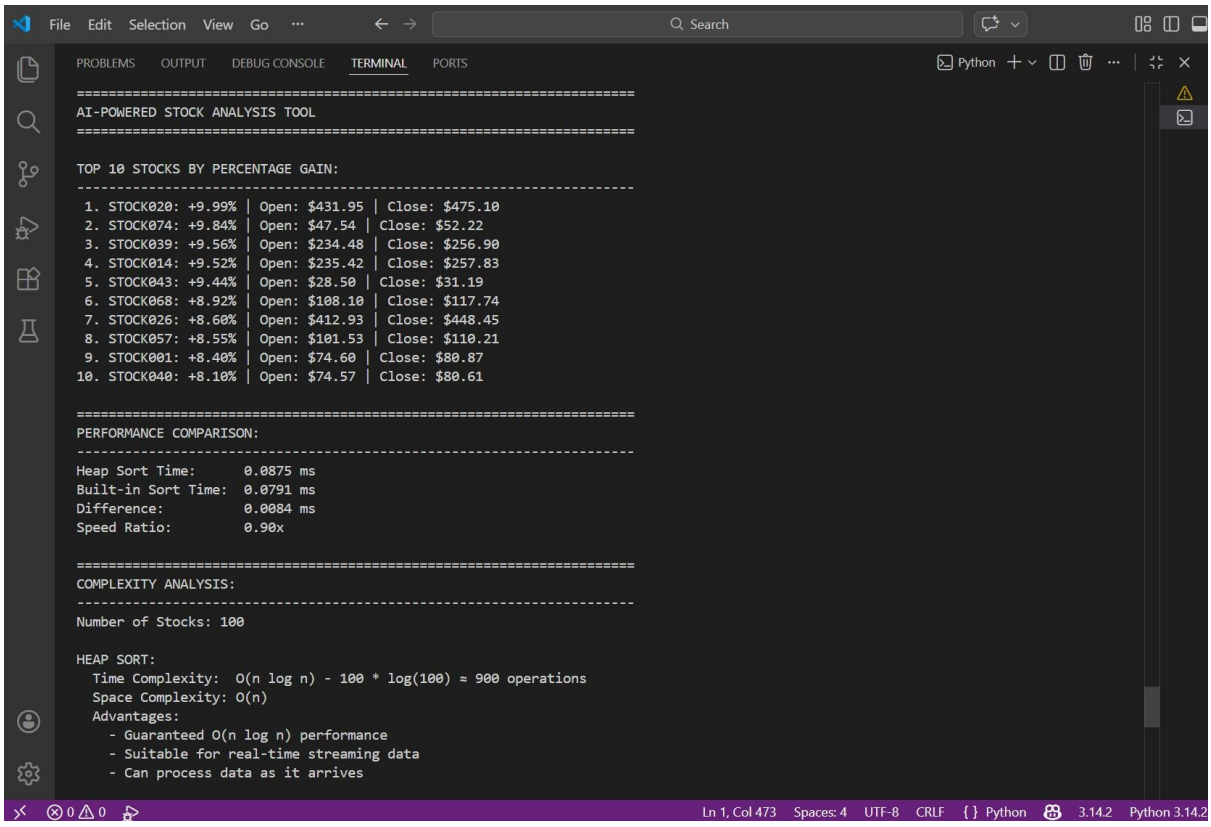

File Edit Selection View Go ···
Search
Welcome
AIAC Code.py
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
7 class StockAnalysisTool:
68 def display_analysis(self):
81 print(f"{i:2d}. {symbol}: {change:+.2f}% | "
82 f"Open: ${self.stocks[symbol]['opening']:.2f} | "
83 f"Close: ${self.stocks[symbol]['closing']:.2f}")
84
85 # Display performances
86 print("\n" + "=" * 70)
87 print("PERFORMANCE COMPARISON:")
88 print("-" * 70)
89 print(f"Heap Sort Time: {heap_time * 1000:.4f} ms")
90 print(f"Built-in Sort Time: {builtin_time * 1000:.4f} ms")
91 print(f"Difference: {abs(heap_time - builtin_time) * 1000:.4f} ms")
92 print(f"Speed Ratio: {builtin_time / heap_time:.2f}x")
93
94 # Complexity Analysis
95 print("\n" + "=" * 70)
96 print("COMPLEXITY ANALYSIS:")
97 print("-" * 70)
98 print(f"Number of Stocks: {self.num_stocks}")
99 print("\nHEAP SORT:")
100 print(f" Time Complexity: O(n log n) - {self.num_stocks} * log({self.num_stocks}) ≈ {self.num_stocks * len(
101 print(f" Space Complexity: O(n)")
102 print(f" Advantages:")
103 print(f" - Guaranteed O(n log n) performance")
104 print(f" - Suitable for real-time streaming data")
105 print(f" - Can process data as it arrives")
106
107 print("\nBUILT-IN SORT (Timsort):")
108 print(f" Time Complexity: O(n log n) average, O(n) best case")
109 print(f" Space Complexity: O(n)")
110 print(f" Advantages:")
111 print(f" - Highly optimized for practical datasets")



```
File Edit Selection View Go ···  Search
Welcome  AIAC Code.py ●
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
```

```
  7  class StockAnalysisTool:
 68      def display_analysis(self):
119          print(f"    - Instant symbol lookup (O(1))")
120          print(f"    - Essential for real-time systems")
121          print(f"    - Avoids need for binary search")
122
123          # Sample lookup
124          print("\n" + "=" * 70)
125          print("SAMPLE HASH MAP LOOKUP:")
126          print("-" * 70)
127          test_symbol = heap_result[0][0]
128          stock_data = self.lookup_stock(test_symbol)
129          print(f"Symbol: {test_symbol}")
130          print(f"Data: {stock_data}")
131          print(f"Lookup Time: O(1) - Instant access")
132
133          print("\n" + "=" * 70)
134          print("WHY THESE STRUCTURES FOR REAL-TIME SYSTEMS:")
135          print("-" * 70)
136          print("1. Hash Maps: Enables O(1) lookups - critical for handling")
137          print("   thousands of concurrent stock quote requests")
138          print("2. Heap Sort: Predictable O(n log n) performance without")
139          print("   worst-case degradation (unlike quicksort)")
140          print("3. Combined: Can maintain sorted rankings while allowing")
141          print("   instant individual stock access")
142          print("=" * 70)
143
144  # Main execution
145  if __name__ == "__main__":
146      tool = StockAnalysisTool(num_stocks=100)
147      tool.display_analysis()
```

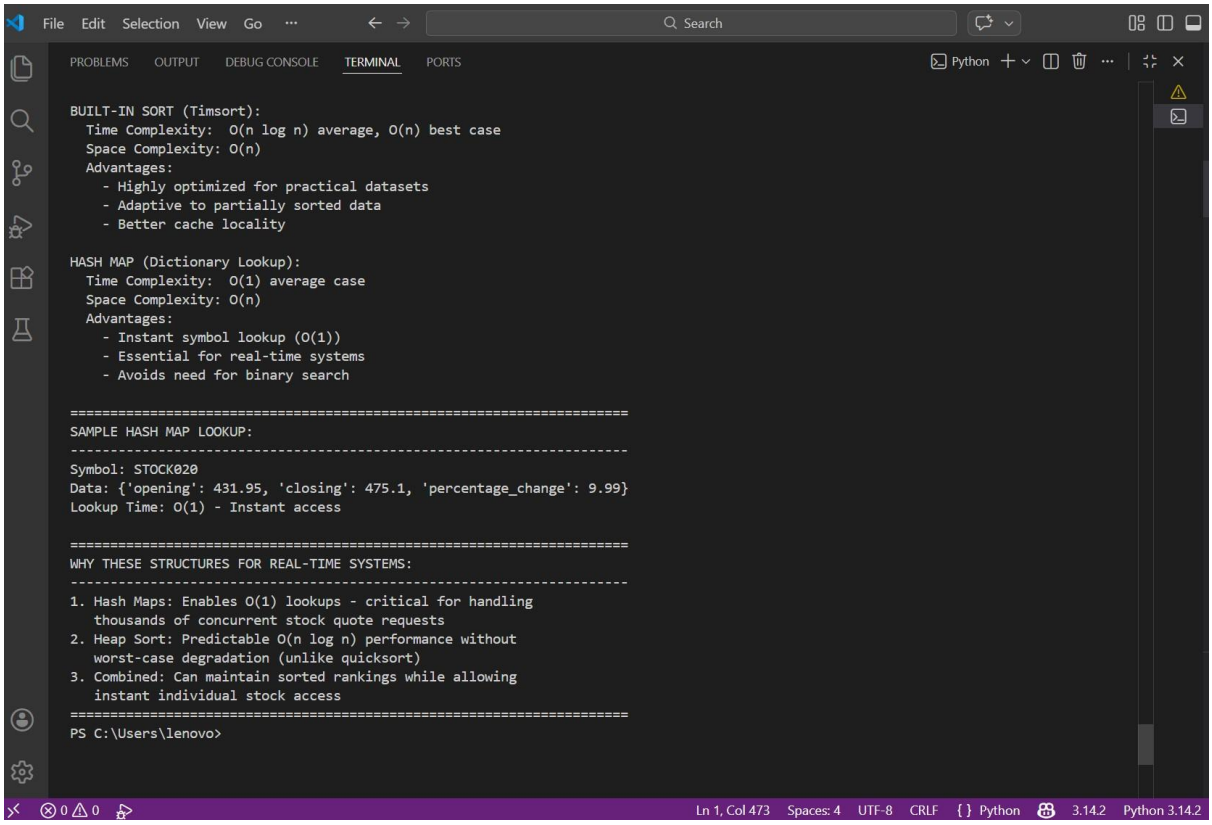
```
⊗ 0 ⚠ 0   Ln 1, Col 473  Spaces: 4  UTF-8  CRLF  { } Python  3.14.2
```

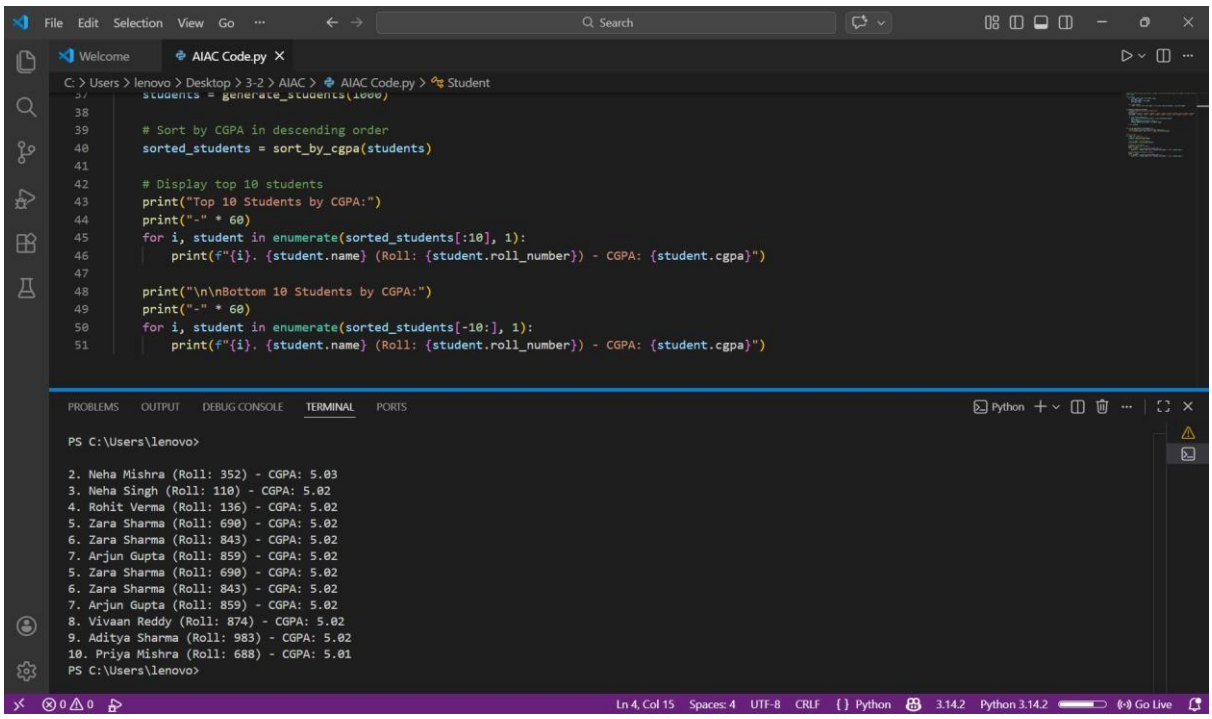


```
File Edit Selection View Go ···  Search
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS   Python
```

```
======================================================================
AI-POWERED STOCK ANALYSIS TOOL
======================================================================

TOP 10 STOCKS BY PERCENTAGE GAIN:
----------------------------------------------------------------------
 1. STOCK020: +9.99% | Open: $431.95 | Close: $475.10
 2. STOCK074: +9.84% | Open: $47.54 | Close: $52.22
 3. STOCK039: +9.56% | Open: $234.48 | Close: $256.90
 4. STOCK014: +9.52% | Open: $235.42 | Close: $257.83
 5. STOCK043: +9.44% | Open: $28.50 | Close: $31.19
 6. STOCK068: +8.92% | Open: $108.10 | Close: $117.74
 7. STOCK026: +8.60% | Open: $412.93 | Close: $448.45
 8. STOCK057: +8.55% | Open: $101.53 | Close: $110.21
 9. STOCK001: +8.40% | Open: $74.60 | Close: $80.87
10. STOCK040: +8.10% | Open: $74.57 | Close: $80.61

======================================================================
PERFORMANCE COMPARISON:
----------------------------------------------------------------------
Heap Sort Time:      0.0875 ms
Built-in Sort Time:  0.0791 ms
Difference:          0.0084 ms
Speed Ratio:         0.90x

======================================================================
COMPLEXITY ANALYSIS:
----------------------------------------------------------------------
Number of Stocks: 100

HEAP SORT:
  Time Complexity:  O(n log n) - 100 * log(100) ≈ 900 operations
  Space Complexity: O(n)
  Advantages:
    - Guaranteed O(n log n) performance
    - Suitable for real-time streaming data
    - Can process data as it arrives
```

```
⊗ 0 ⚠ 0   Ln 1, Col 473  Spaces: 4  UTF-8  CRLF  { } Python  3.14.2  Python 3.14.2
```





Conclusion

In this lab, we implemented and compared various sorting and searching algorithms using AI assistance. Quick Sort and Merge Sort were tested on large datasets and their performance differences were analyzed. Bubble Sort helped in understanding basic comparison-based sorting and its time complexity . Hash Maps demonstrated efficient O(1) searching in real-time applications like inventory and stock systems.

Heap Sort was useful for ranking stock performance. Overall, the lab improved understanding of algorithm efficiency, data structure selection, and practical performance analysis while highlighting benefits of AI-assisted coding.