AI ASSIISTANT CODING LAB-12.3

Name : K . Jashwanth

Rollno : 2303a52300

Batch : 42

Task 1: Sorting Student Records for Placement Drive

Scenario

SR University's Training and Placement Cell needs to shortlist candidates efficiently

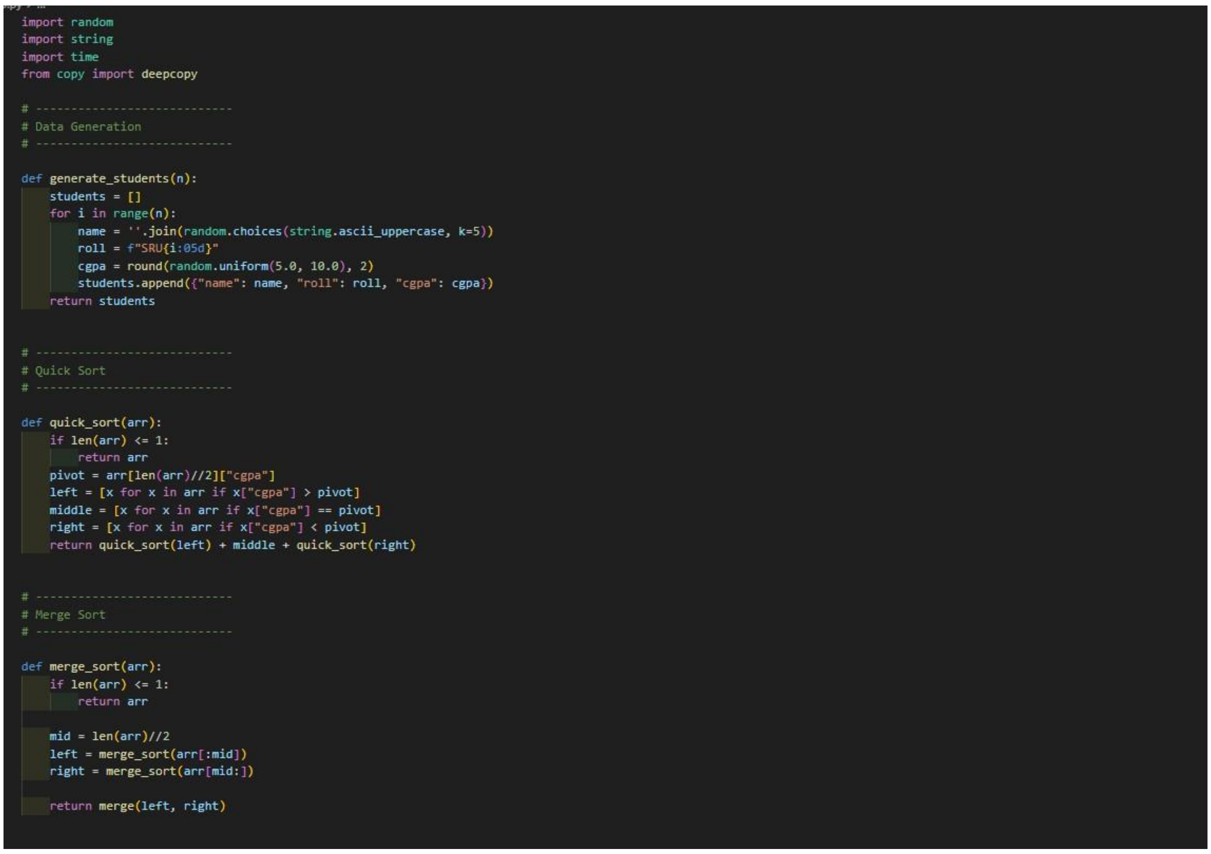
during campus placements. Student records must be sorted by

CGPA in descending order

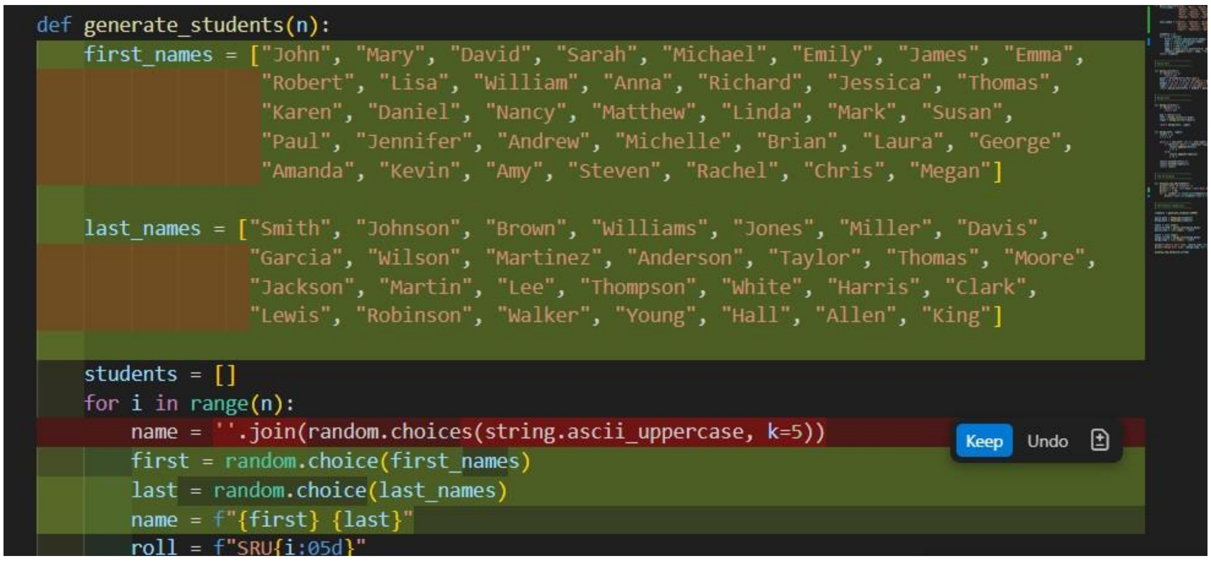
Ai prompt

Generate a Python program to store student records (Name, Roll Number, CGPA). Implement Quick Sort and Merge Sort to sort students by CGPA in descending order. Measure runtime for large datasets and display top 10 students

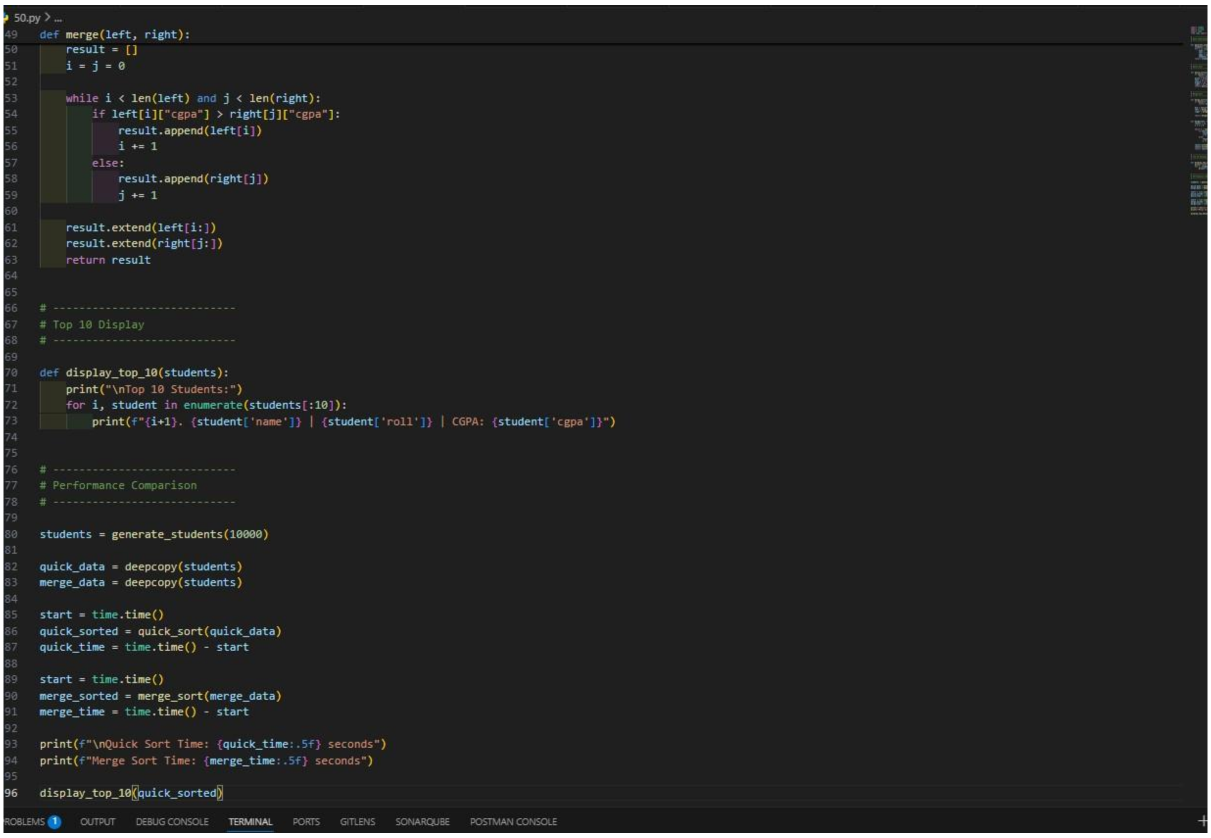
Code



```python
import random
import string
import time
from copy import deepcopy

# ---------------------------
# Data Generation
# ---------------------------

def generate_students(n):
    students = []
    for i in range(n):
        name = ''.join(random.choices(string.ascii_uppercase, k=5))
        roll = f"SRU{i:05d}"
        cgpa = round(random.uniform(5.0, 10.0), 2)
        students.append({"name": name, "roll": roll, "cgpa": cgpa})
    return students


# ---------------------------
# Quick Sort
# ---------------------------

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr)//2]["cgpa"]
    left = [x for x in arr if x["cgpa"] > pivot]
    middle = [x for x in arr if x["cgpa"] == pivot]
    right = [x for x in arr if x["cgpa"] < pivot]
    return quick_sort(left) + middle + quick_sort(right)


# ---------------------------
# Merge Sort
# ---------------------------

def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr)//2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)
```

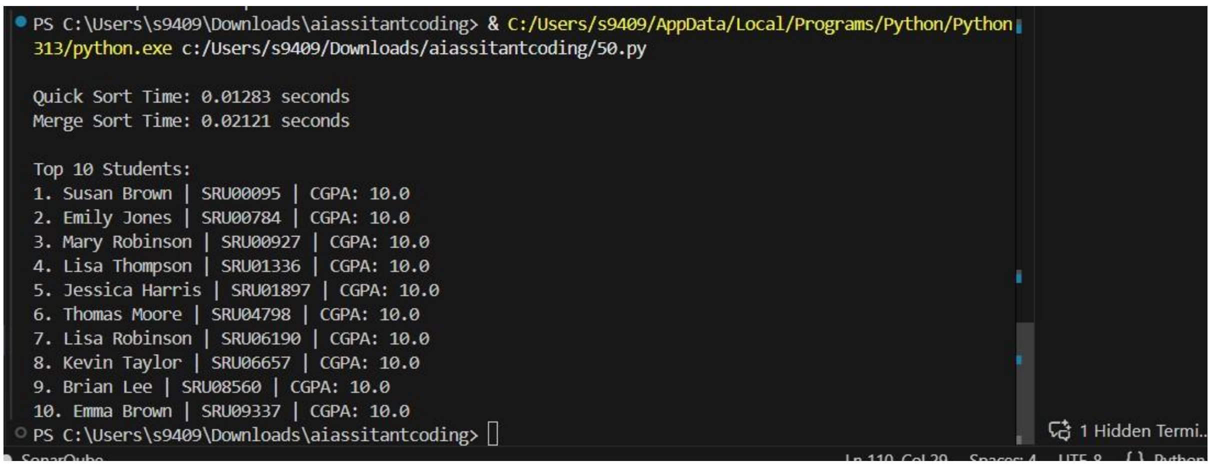


```python
def generate_students(n):
    first_names = ["John", "Mary", "David", "Sarah", "Michael", "Emily", "James", "Emma",
                   "Robert", "Lisa", "William", "Anna", "Richard", "Jessica", "Thomas",
                   "Karen", "Daniel", "Nancy", "Matthew", "Linda", "Mark", "Susan",
                   "Paul", "Jennifer", "Andrew", "Michelle", "Brian", "Laura", "George",
                   "Amanda", "Kevin", "Amy", "Steven", "Rachel", "Chris", "Megan"]

    last_names = ["Smith", "Johnson", "Brown", "Williams", "Jones", "Miller", "Davis",
                  "Garcia", "Wilson", "Martinez", "Anderson", "Taylor", "Thomas", "Moore",
                  "Jackson", "Martin", "Lee", "Thompson", "White", "Harris", "Clark",
                  "Lewis", "Robinson", "Walker", "Young", "Hall", "Allen", "King"]

    students = []
    for i in range(n):
        name = ''.join(random.choices(string.ascii_uppercase, k=5))
        first = random.choice(first_names)
        last = random.choice(last_names)
        name = f"{first} {last}"
        roll = f"SRU{i:05d}"
```



```python
def merge(left, right):
    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


# ---------------------------
# Top 10 Display
# ---------------------------

def display_top_10(students):
    print("\nTop 10 Students:")
    for i, student in enumerate(students[:10]):
        print(f"{i+1}. {student['name']} | {student['roll']} | CGPA: {student['cgpa']}")


# ---------------------------
# Performance Comparison
# ---------------------------

students = generate_students(10000)

quick_data = deepcopy(students)
merge_data = deepcopy(students)

start = time.time()
quick_sorted = quick_sort(quick_data)
quick_time = time.time() - start

start = time.time()
merge_sorted = merge_sort(merge_data)
merge_time = time.time() - start

print(f"\nQuick Sort Time: {quick_time:.5f} seconds")
print(f"Merge Sort Time: {merge_time:.5f} seconds")

display_top_10(quick_sorted)
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/50.py

Quick Sort Time: 0.01283 seconds
Merge Sort Time: 0.02121 seconds

Top 10 Students:
1. Susan Brown | SRU00095 | CGPA: 10.0
2. Emily Jones | SRU00784 | CGPA: 10.0
3. Mary Robinson | SRU00927 | CGPA: 10.0
4. Lisa Thompson | SRU01336 | CGPA: 10.0
5. Jessica Harris | SRU01897 | CGPA: 10.0
6. Thomas Moore | SRU04798 | CGPA: 10.0
7. Lisa Robinson | SRU06190 | CGPA: 10.0
8. Kevin Taylor | SRU06657 | CGPA: 10.0
9. Brian Lee | SRU08560 | CGPA: 10.0
10. Emma Brown | SRU09337 | CGPA: 10.0
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Performance Analysis

Algorithm Best Case Average Case Worst Case

Quick Sort O(n log n) O(n log n) $O(n^2)$ Merge

Sort O(n log n) O(n log n) O(n log n)

Justification:

Quick Sort performs faster in practice due to in-place behavior.

Merge Sort guarantees stable performance even for worst case.

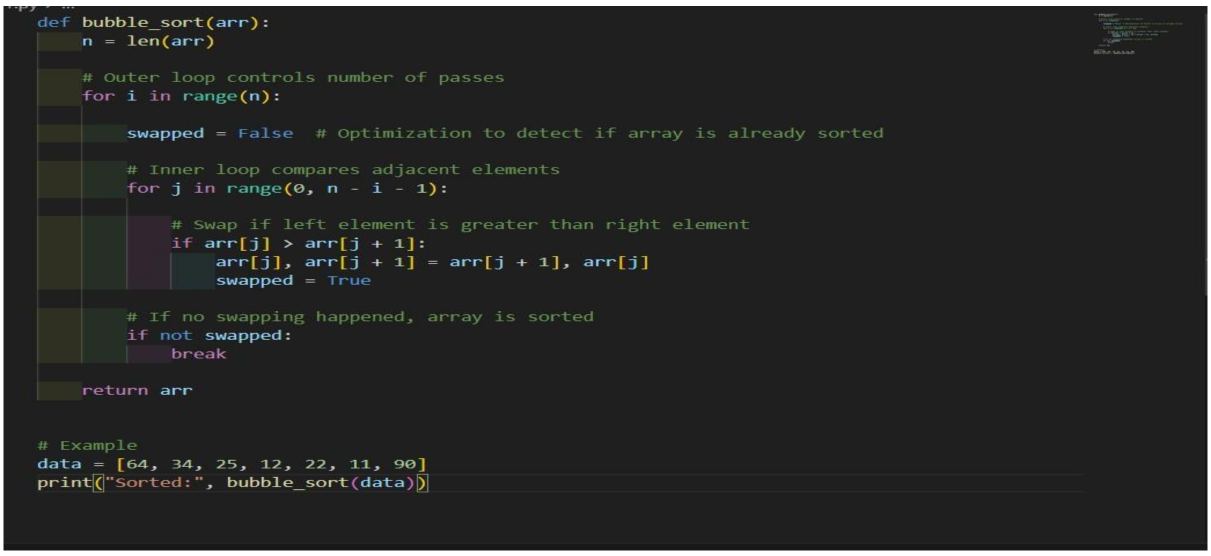
TASK 2: Bubble Sort with AI Comments

Write a Python implementation of Bubble Sort.

AI Prompt

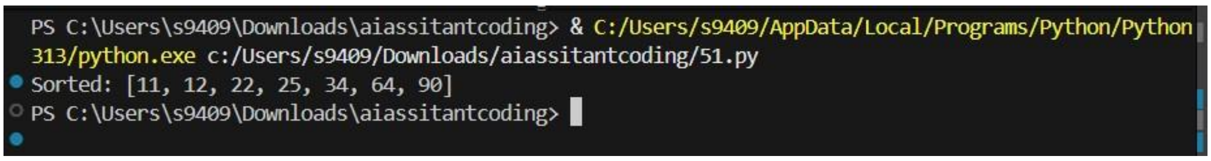
Generate Bubble Sort in Python with inline explanatory comments and time complexity analysis.

Code



```python
def bubble_sort(arr):
    n = len(arr)

    # Outer loop controls number of passes
    for i in range(n):

        swapped = False  # Optimization to detect if array is already sorted

        # Inner loop compares adjacent elements
        for j in range(0, n - i - 1):

            # Swap if left element is greater than right element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # If no swapping happened, array is sorted
        if not swapped:
            break

    return arr


# Example
data = [64, 34, 25, 12, 22, 11, 90]
print("Sorted:", bubble_sort(data))
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/51.py
Sorted: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification:

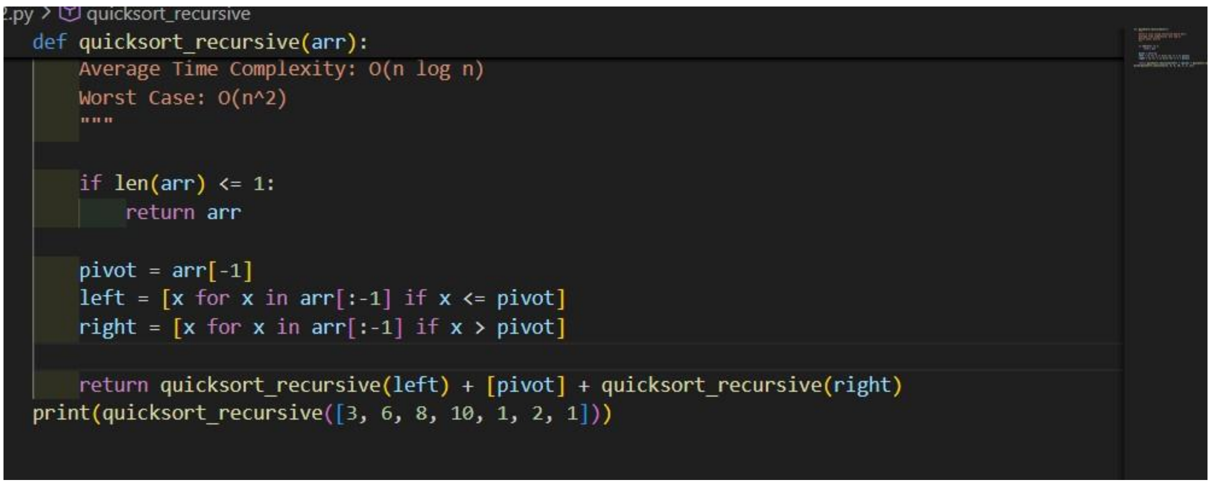
Time Complexity Analysis

- Best Case: **O(n)** (already sorted)
- Average Case: **O(n²)**
- Worst Case: **O(n²)**
- Space Complexity: **O(1)** (In-place)

Task 3: Quick Sort and Merge Sort Comparison

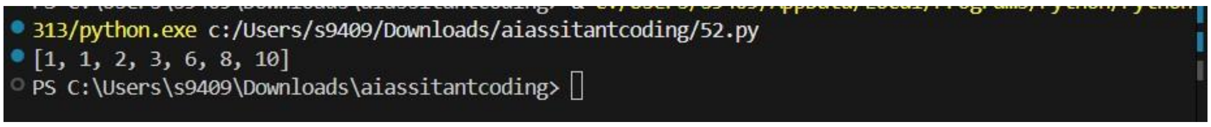
• Task: Implement Quick Sort and Merge Sort using recursion

Recursive Quick sort code



```
.py > quicksort_recursive
def quicksort_recursive(arr):
    Average Time Complexity: O(n log n)
    Worst Case: O(n^2)
    """

    if len(arr) <= 1:
        return arr

    pivot = arr[-1]
    left = [x for x in arr[:-1] if x <= pivot]
    right = [x for x in arr[:-1] if x > pivot]

    return quicksort_recursive(left) + [pivot] + quicksort_recursive(right)
print(quicksort_recursive([3, 6, 8, 10, 1, 2, 1]))
```

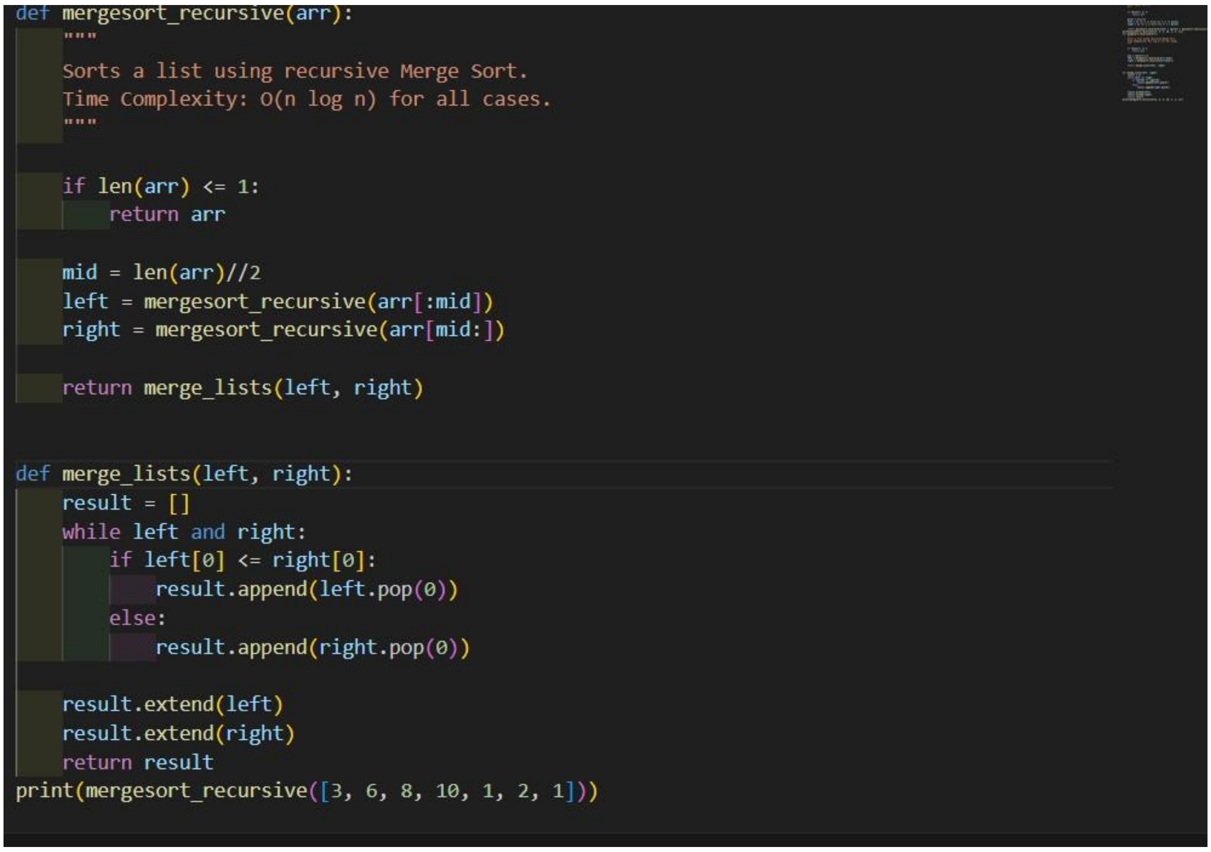
Output:



```
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/52.py
[1, 1, 2, 3, 6, 8, 10]
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

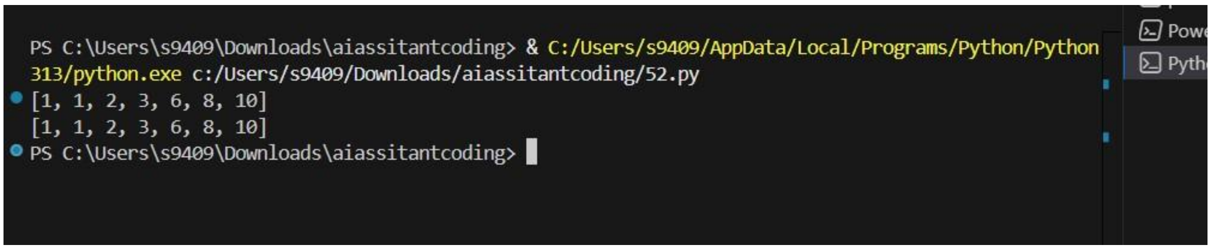
Recursive merge Sort

Code



```python
def mergesort_recursive(arr):
    """
    Sorts a list using recursive Merge Sort.
    Time Complexity: O(n log n) for all cases.
    """

    if len(arr) <= 1:
        return arr

    mid = len(arr)//2
    left = mergesort_recursive(arr[:mid])
    right = mergesort_recursive(arr[mid:])

    return merge_lists(left, right)


def merge_lists(left, right):
    result = []
    while left and right:
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))

    result.extend(left)
    result.extend(right)
    return result
print(mergesort_recursive([3, 6, 8, 10, 1, 2, 1]))
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/52.py
[1, 1, 2, 3, 6, 8, 10]
[1, 1, 2, 3, 6, 8, 10]
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Complexity Summary

Input Type	Quick Sort	Merge Sort
Random	O(n log n)	O(n log n)
Sorted	$O(n^2)$	O(n log n)
Reverse	$O(n^2)$	O(n log n)

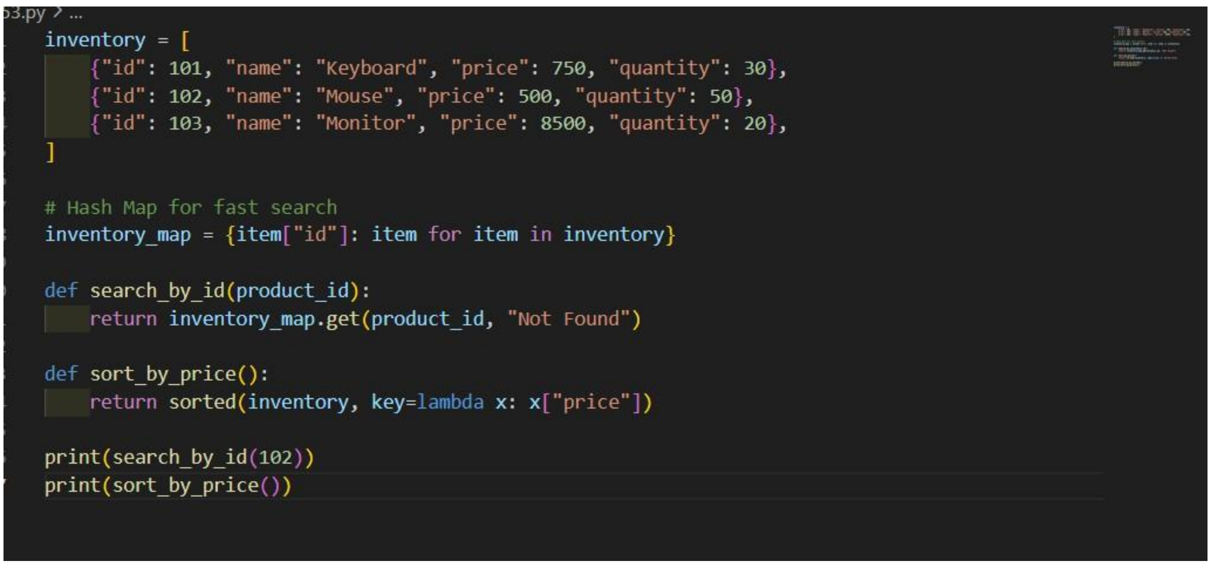
TASK 4: Inventory Management System

Recommended Algorithms

Operation	Algorithm	Justification
Search by ID	Hash Map (Dictionary)	O(1) lookup

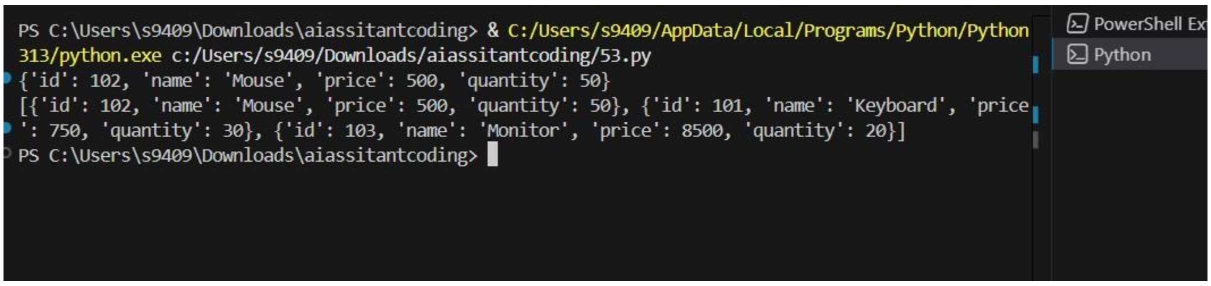
Operation	Algorithm	Justification
Search by Name	Hash Map	Fast index access
Sort by Price	Timsort (Python sorted())	Stable and optimized
Sort by Quantity	Merge Sort	Consistent O(n log n)

Code



```
53.py > ...
    inventory = [
        {"id": 101, "name": "Keyboard", "price": 750, "quantity": 30},
        {"id": 102, "name": "Mouse", "price": 500, "quantity": 50},
        {"id": 103, "name": "Monitor", "price": 8500, "quantity": 20},
    ]

    # Hash Map for fast search
    inventory_map = {item["id"]: item for item in inventory}

    def search_by_id(product_id):
        return inventory_map.get(product_id, "Not Found")

    def sort_by_price():
        return sorted(inventory, key=lambda x: x["price"])

    print(search_by_id(102))
    print(sort_by_price())
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/53.py
{'id': 102, 'name': 'Mouse', 'price': 500, 'quantity': 50}
[{'id': 102, 'name': 'Mouse', 'price': 500, 'quantity': 50}, {'id': 101, 'name': 'Keyboard', 'price
': 750, 'quantity': 30}, {'id': 103, 'name': 'Monitor', 'price': 8500, 'quantity': 20}]
PS C:\Users\s9409\Downloads\aiassitantcoding> 
```

Justification

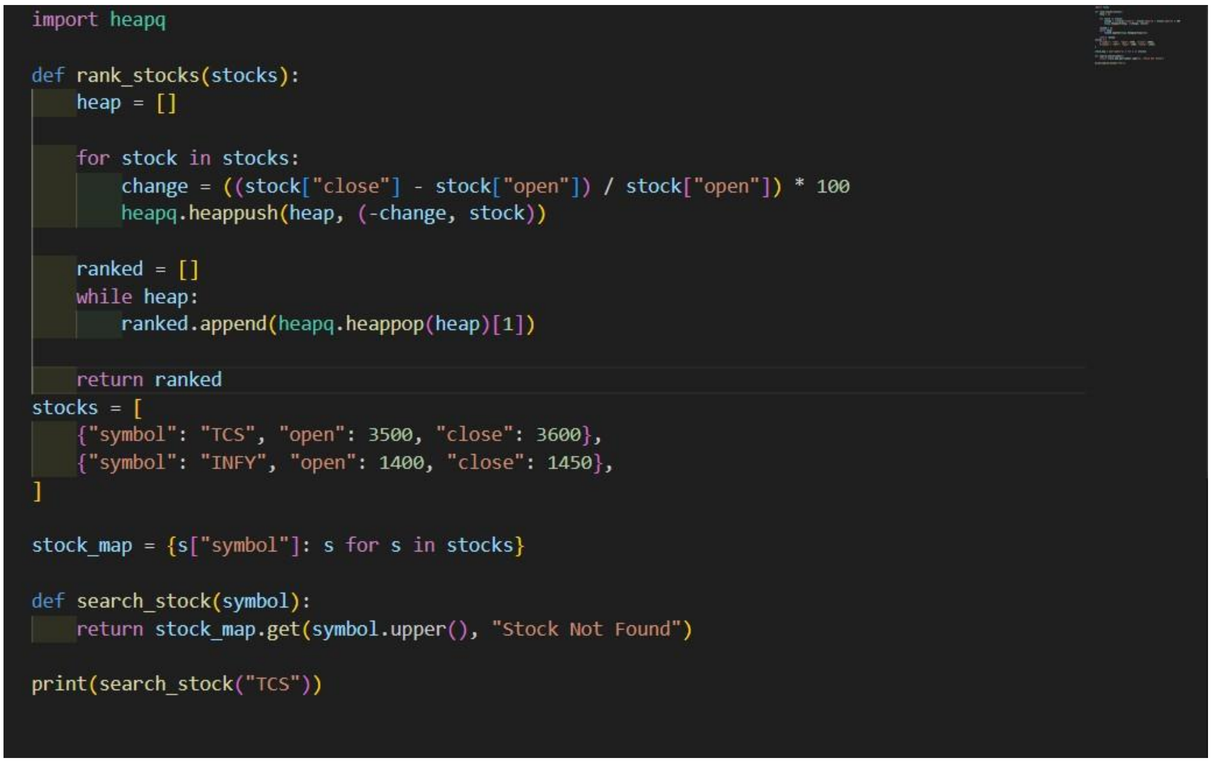
- Large dataset → Hash Map reduces search from O(n) to O(1)
- Retail requires frequent search → Constant-time lookup critical
- Sorting less frequent → O(n log n) acceptable **Task 5: Real-Time Stock Data Sorting & Searching**

Scenario:

An AI-powered FinTech Lab at SR University is building a tool for analyzing stock price movements. The requirement is to quickly sort stocks by daily gain/loss and search for specific stock symbols efficiently.

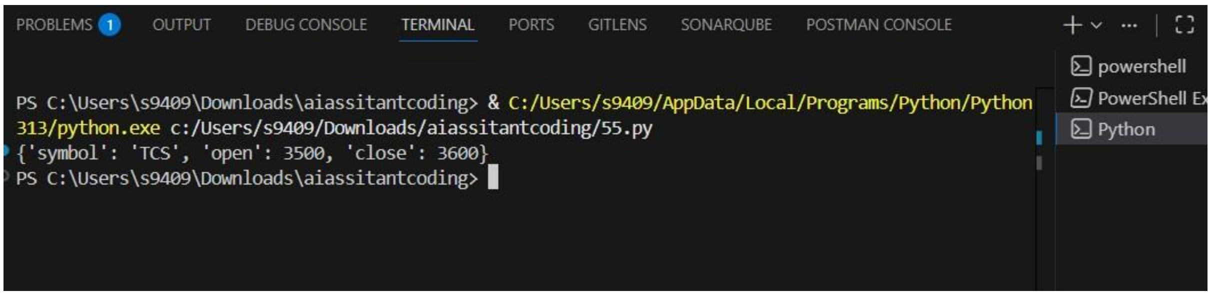
Heap Sort Implementation

Code



```python
import heapq

def rank_stocks(stocks):
    heap = []

    for stock in stocks:
        change = ((stock["close"] - stock["open"]) / stock["open"]) * 100
        heapq.heappush(heap, (-change, stock))

    ranked = []
    while heap:
        ranked.append(heapq.heappop(heap)[1])

    return ranked
stocks = [
    {"symbol": "TCS", "open": 3500, "close": 3600},
    {"symbol": "INFY", "open": 1400, "close": 1450},
]

stock_map = {s["symbol"]: s for s in stocks}

def search_stock(symbol):
    return stock_map.get(symbol.upper(), "Stock Not Found")

print(search_stock("TCS"))
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/55.py
{'symbol': 'TCS', 'open': 3500, 'close': 3600}
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Performance Comparison

Operation	Heap Sort	sorted()
Sorting	O(n log n)	O(n log n)
Search	O(1) HashMap	O(1) Dict

Trade-offs

- sorted() (Timsort) is highly optimized in CPython
- Heap Sort useful when continuously extracting top-K elements
- Hash Map ideal for instant symbol retrieval