2303A51739

Batch : 25

Task 1: Bubble Sort for Ranking Exam

Scores

Scenario

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

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

Task Description

• Implement Bubble Sort in Python to sort a list of student scores.

• Use an AI tool to:

o Insert inline comments explaining key operations such as

comparisons, swaps, and iteration passes

o Identify early-termination conditions when the list

becomes sorted

o Provide a brief time complexity analysis

Expected Outcome

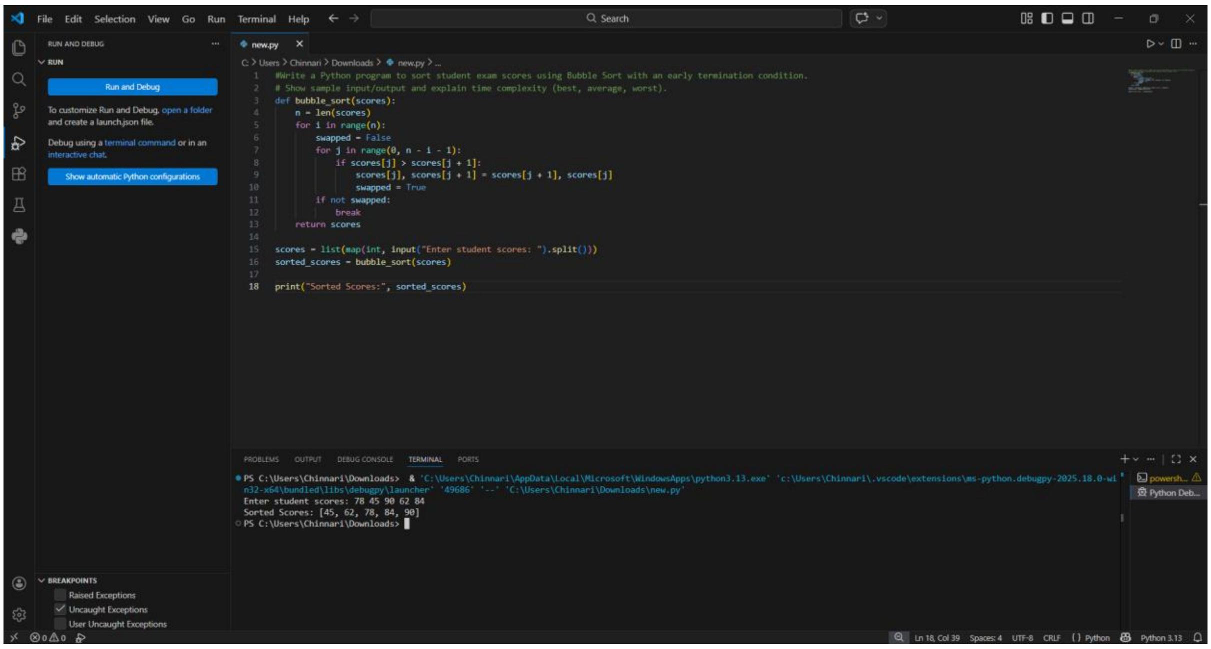
• A Bubble Sort implementation with:

o AI-generated comments explaining the logic

o Clear explanation of best, average, and worst-case

complexity

o Sample input/output showing sorted scores



```python
#Write a Python program to sort student exam scores using Bubble Sort with an early termination condition.
# Show sample input/output and explain time complexity (best, average, worst).
def bubble_sort(scores):
    n = len(scores)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if scores[j] > scores[j + 1]:
                scores[j], scores[j + 1] = scores[j + 1], scores[j]
                swapped = True
        if not swapped:
            break
    return scores

scores = list(map(int, input("Enter student scores: ").split()))
sorted_scores = bubble_sort(scores)

print("Sorted Scores:", sorted_scores)
```

```
PS C:\Users\Chinnari\Downloads>  & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-wi
n32-x64\bundled\libs\debugpy\launcher' '49686' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter student scores: 78 45 90 62 84
Sorted Scores: [45, 62, 78, 84, 90]
PS C:\Users\Chinnari\Downloads>
```

Task 2: Improving Sorting for Nearly Sorted

Attendance Records

Scenario

You are maintaining an attendance system where student roll numbers

are already almost sorted, with only a few late updates.

Task Description

• Start with a Bubble Sort implementation.

• Ask AI to:

o Review the problem and suggest a more suitable sorting

algorithm

o Generate an Insertion Sort implementation

o Explain why Insertion Sort performs better on nearly

sorted data

• Compare execution behavior on nearly sorted input

Expected Outcome

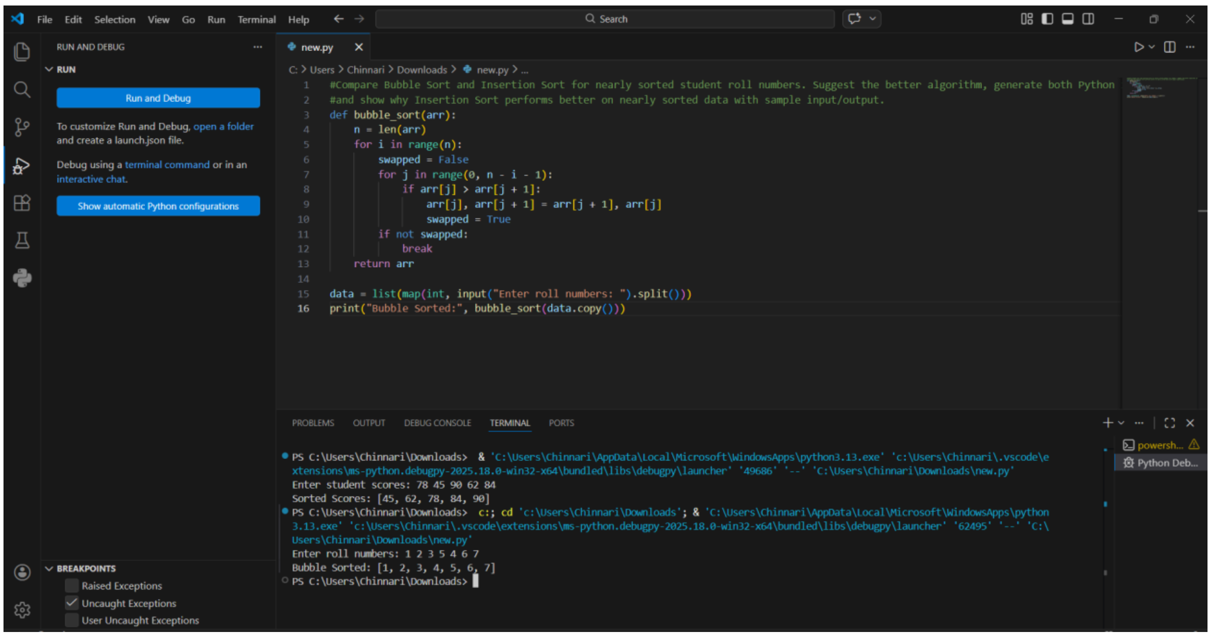
• Two sorting implementations:

o Bubble Sort

o Insertion Sort

• AI-assisted explanation highlighting efficiency differences for

partially sorted datasets give code and promt for this remove comments



```python
#Compare Bubble Sort and Insertion Sort for nearly sorted student roll numbers. Suggest the better algorithm, generate both Python
#and show why Insertion Sort performs better on nearly sorted data with sample input/output.
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr

data = list(map(int, input("Enter roll numbers: ").split()))
print("Bubble Sorted:", bubble_sort(data.copy()))
```

```
PS C:\Users\Chinnari\Downloads>  & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\e
xtensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49686' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter student scores: 78 45 90 62 84
Sorted Scores: [45, 62, 78, 84, 90]
PS C:\Users\Chinnari\Downloads>  c:; cd 'c:\Users\Chinnari\Downloads'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python
3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62495' '--' 'C:\
Users\Chinnari\Downloads\new.py'
Enter roll numbers: 1 2 3 5 4 6 7
Bubble Sorted: [1, 2, 3, 4, 5, 6, 7]
PS C:\Users\Chinnari\Downloads>
```

Task 3: Searching Student Records in a

Database

Scenario

You are developing a student information portal where users search for

student records by roll number.

Task Description

• Implement:

o Linear Search for unsorted student data

o Binary Search for sorted student data

• Use AI to:

o Add docstrings explaining parameters and return values

o Explain when Binary Search is applicable

o Highlight performance differences between the two

searches

Expected Outcome

• Two working search implementations with docstrings

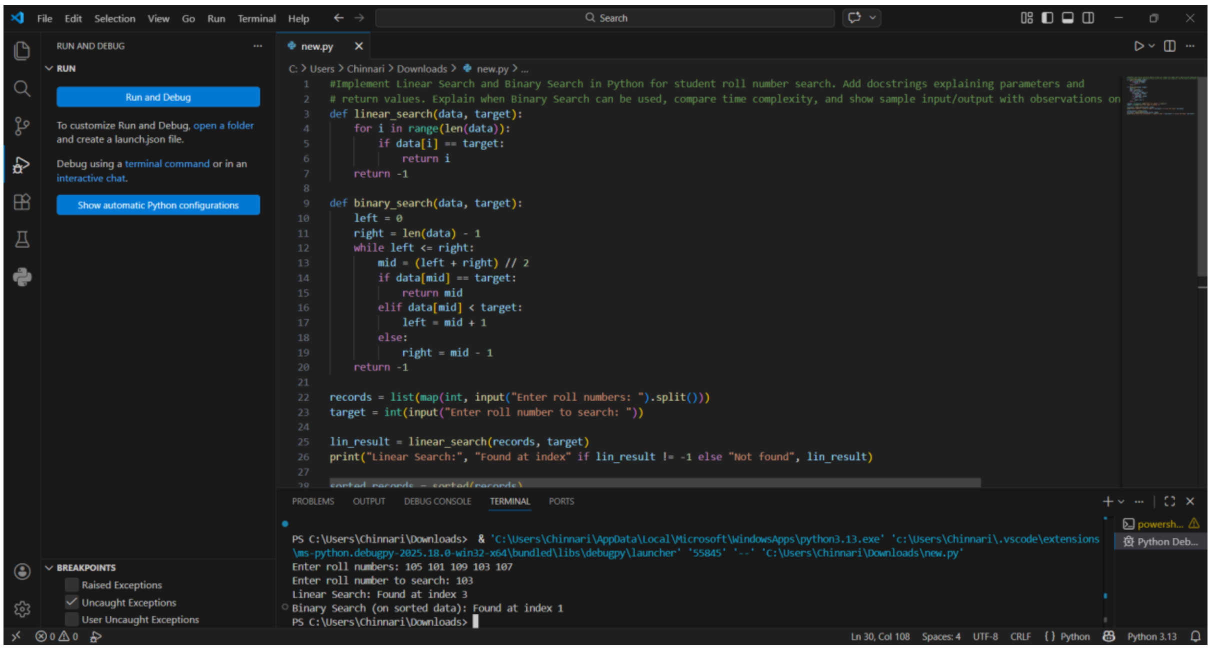
• AI-generated explanation of:

o Time complexity

o Use cases for Linear vs Binary Search

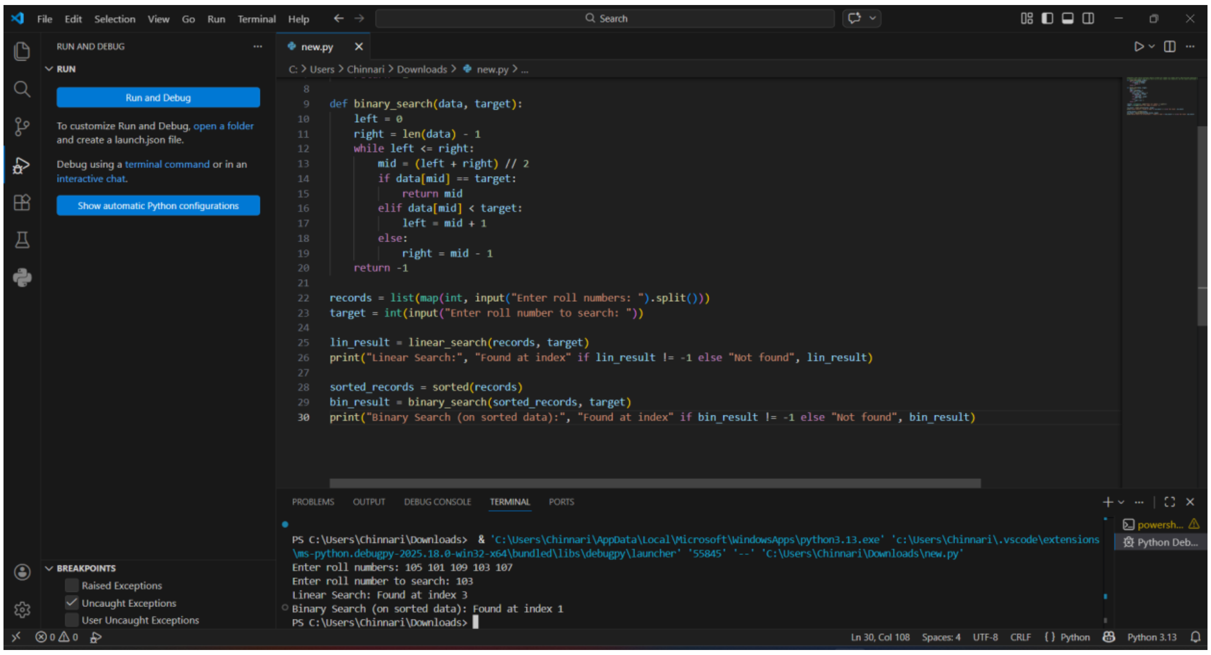
• A short student observation comparing results on sorted vs unsorted lists

give code and promt for this and remove comments



```python
#Implement Linear Search and Binary Search in Python for student roll number search. Add docstrings explaining parameters and
# return values. Explain when Binary Search can be used, compare time complexity, and show sample input/output with observations on
def linear_search(data, target):
    for i in range(len(data)):
        if data[i] == target:
            return i
    return -1

def binary_search(data, target):
    left = 0
    right = len(data) - 1
    while left <= right:
        mid = (left + right) // 2
        if data[mid] == target:
            return mid
        elif data[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

records = list(map(int, input("Enter roll numbers: ").split()))
target = int(input("Enter roll number to search: "))

lin_result = linear_search(records, target)
print("Linear Search:", "Found at index" if lin_result != -1 else "Not found", lin_result)

```

```
PS C:\Users\Chinnari\Downloads>  & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions
\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55845' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter roll numbers: 105 101 109 103 107
Enter roll number to search: 103
Linear Search: Found at index 3
Binary Search (on sorted data): Found at index 1
PS C:\Users\Chinnari\Downloads>
```



```python

def binary_search(data, target):
    left = 0
    right = len(data) - 1
    while left <= right:
        mid = (left + right) // 2
        if data[mid] == target:
            return mid
        elif data[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

records = list(map(int, input("Enter roll numbers: ").split()))
target = int(input("Enter roll number to search: "))

lin_result = linear_search(records, target)
print("Linear Search:", "Found at index" if lin_result != -1 else "Not found", lin_result)

sorted_records = sorted(records)
bin_result = binary_search(sorted_records, target)
print("Binary Search (on sorted data):", "Found at index" if bin_result != -1 else "Not found", bin_result)
```

```
PS C:\Users\Chinnari\Downloads>  & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions
\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55845' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter roll numbers: 105 101 109 103 107
Enter roll number to search: 103
Linear Search: Found at index 3
Binary Search (on sorted data): Found at index 1
PS C:\Users\Chinnari\Downloads>
```

Task 4: Choosing Between Quick Sort and

Merge Sort for Data Processing

Scenario

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

Task Description

• Provide AI with partially written recursive functions for:

o Quick Sort

o Merge Sort

• Ask AI to:

o Complete the recursive logic

o Add meaningful docstrings

o Explain how recursion works in each algorithm

• Test both algorithms on:

o Random data

o Sorted data

o Reverse-sorted data

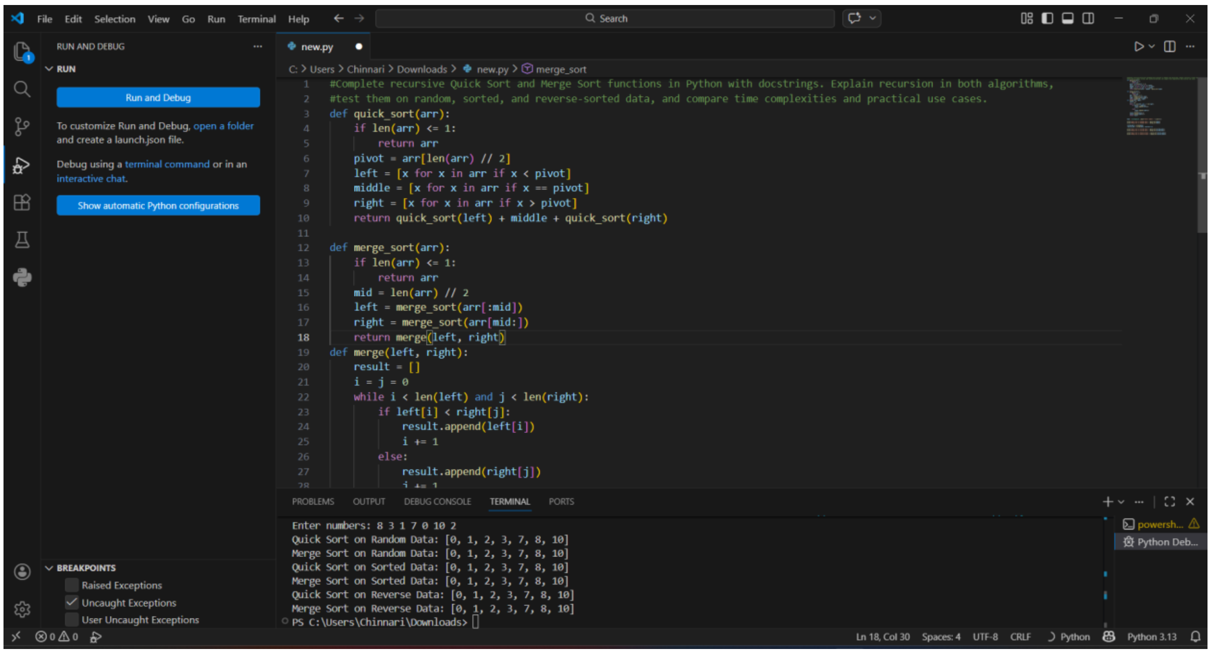
Expected Outcome

• Fully functional Quick Sort and Merge Sort implementations

• AI-generated comparison covering:

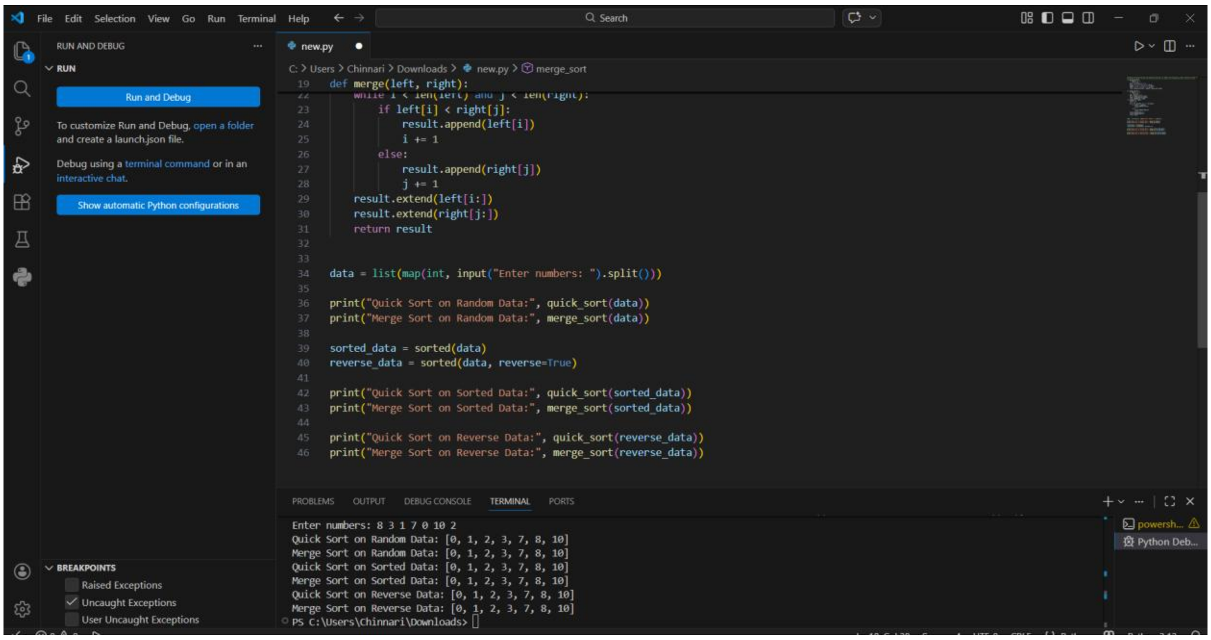
o Best, average, and worst-case complexities

o Practical scenarios where one algorithm is preferred over the other give promt and code for this and remove comments



```python
#Complete recursive Quick Sort and Merge Sort functions in Python with docstrings. Explain recursion in both algorithms,
#test them on random, sorted, and reverse-sorted data, and compare time complexities and practical use cases.
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

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)
def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
```

```
Enter numbers: 8 3 1 7 0 10 2
Quick Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
PS C:\Users\Chinnari\Downloads>
```



```python
def merge(left, right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result


data = list(map(int, input("Enter numbers: ").split()))

print("Quick Sort on Random Data:", quick_sort(data))
print("Merge Sort on Random Data:", merge_sort(data))

sorted_data = sorted(data)
reverse_data = sorted(data, reverse=True)

print("Quick Sort on Sorted Data:", quick_sort(sorted_data))
print("Merge Sort on Sorted Data:", merge_sort(sorted_data))

print("Quick Sort on Reverse Data:", quick_sort(reverse_data))
print("Merge Sort on Reverse Data:", merge_sort(reverse_data))
```

```
Enter numbers: 8 3 1 7 0 10 2
Quick Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
PS C:\Users\Chinnari\Downloads>
```

Task 5: Optimizing a Duplicate Detection

Algorithm

Scenario

You are building a data validation module that must detect duplicate

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

Task Description

• Write a naive duplicate detection algorithm using nested loops.

• Use AI to:

o Analyze the time complexity

o Suggest an optimized approach using sets or dictionaries

o Rewrite the algorithm with improved efficiency

• Compare execution behavior conceptually for large input sizes

Expected Outcome

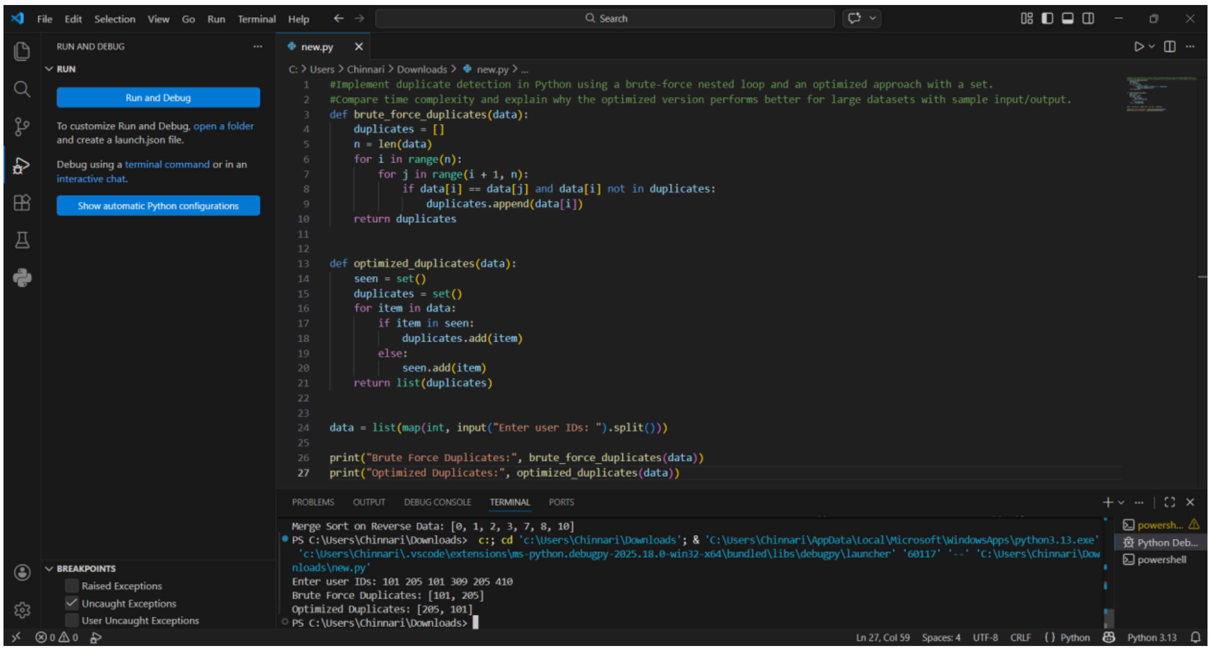
• Two versions of the algorithm:

o Brute-force (O(n²))

o Optimized (O(n))

• AI-assisted explanation showing how and why performance

improved give promt and code for this and remove comments



```python
#Implement duplicate detection in Python using a brute-force nested loop and an optimized approach with a set.
#Compare time complexity and explain why the optimized version performs better for large datasets with sample input/output.
def brute_force_duplicates(data):
    duplicates = []
    n = len(data)
    for i in range(n):
        for j in range(i + 1, n):
            if data[i] == data[j] and data[i] not in duplicates:
                duplicates.append(data[i])
    return duplicates


def optimized_duplicates(data):
    seen = set()
    duplicates = set()
    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)
    return list(duplicates)


data = list(map(int, input("Enter user IDs: ").split()))

print("Brute Force Duplicates:", brute_force_duplicates(data))
print("Optimized Duplicates:", optimized_duplicates(data))
```

```
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
PS C:\Users\Chinnari\Downloads>  c:; cd 'c:\Users\Chinnari\Downloads'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe'
 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60117' '--' 'C:\Users\Chinnari\Dow
nloads\new.py'
Enter user IDs: 101 205 101 309 205 410
Brute Force Duplicates: [101, 205]
Optimized Duplicates: [205, 101]
PS C:\Users\Chinnari\Downloads>
```