ASSIGNMENT 12.4

NAME: K.HITHESH

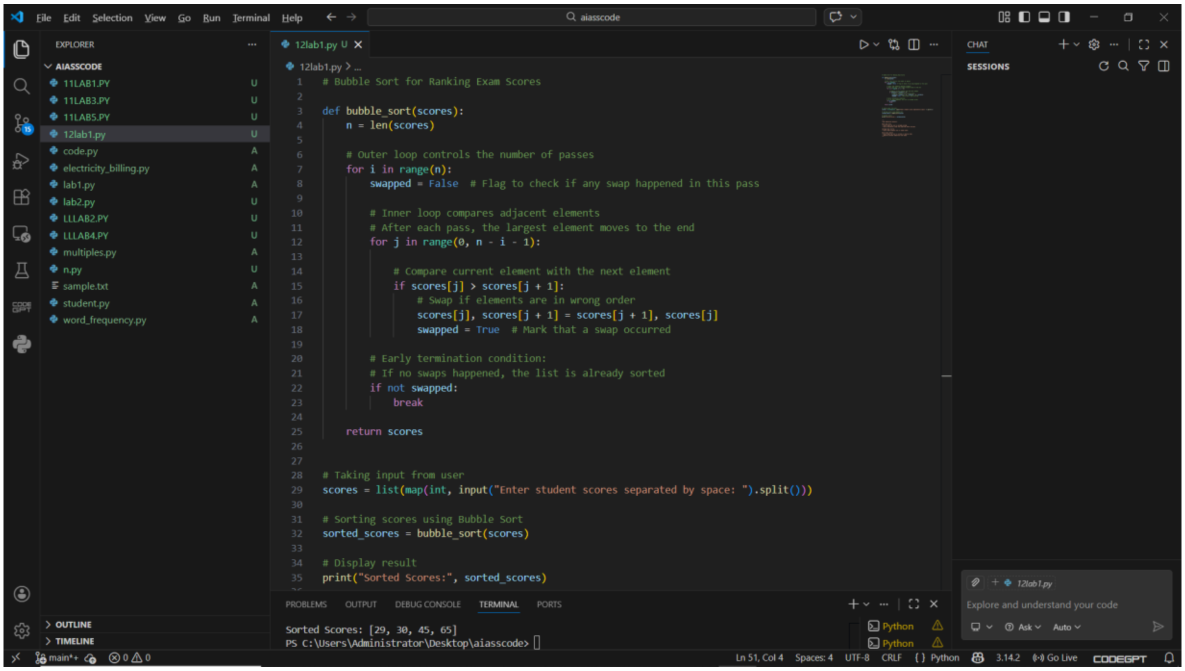
HALLTICKET: 2303A51291

TASK1:

Prompt

Generate a Python program that implements Bubble Sort to sort a list of student exam scores.
Add inline comments explaining comparisons, swaps, and iteration passes.
Include an early termination condition if the list becomes sorted before completing all passes.
Also include a brief time complexity explanation in comments and show sample input/output.

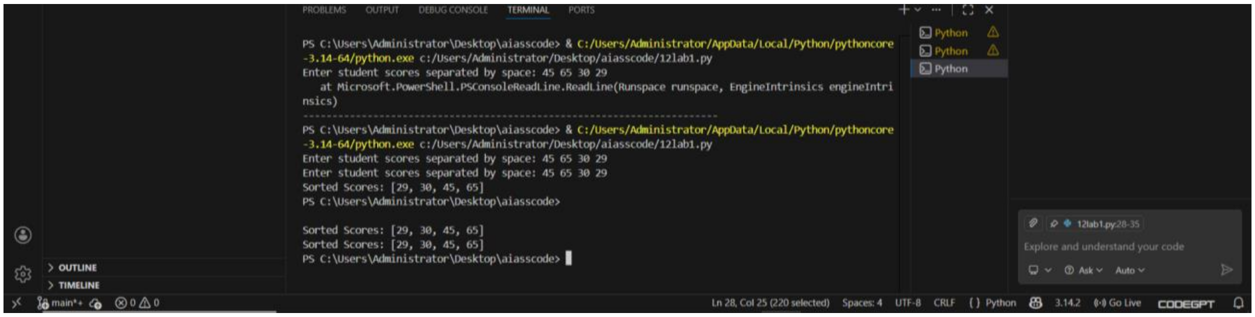
Code:



```python
# Bubble Sort for Ranking Exam Scores

def bubble_sort(scores):
    n = len(scores)

    # Outer loop controls the number of passes
    for i in range(n):
        swapped = False  # Flag to check if any swap happened in this pass

        # Inner loop compares adjacent elements
        # After each pass, the largest element moves to the end
        for j in range(0, n - i - 1):

            # Compare current element with the next element
            if scores[j] > scores[j + 1]:
                # Swap if elements are in wrong order
                scores[j], scores[j + 1] = scores[j + 1], scores[j]
                swapped = True  # Mark that a swap occurred

        # Early termination condition:
        # If no swaps happened, the list is already sorted
        if not swapped:
            break

    return scores


# Taking input from user
scores = list(map(int, input("Enter student scores separated by space: ").split()))

# Sorting scores using Bubble Sort
sorted_scores = bubble_sort(scores)

# Display result
print("Sorted Scores:", sorted_scores)
```

```
Sorted Scores: [29, 30, 45, 65]
PS C:\Users\Administrator\Desktop\aiasscode> 
```

Output:



```
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore
-3.14-64/python.exe c:/Users/Administrator/Desktop/aiasscode/12lab1.py
Enter student scores separated by space: 45 65 30 29
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntri
nsics)
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore
-3.14-64/python.exe c:/Users/Administrator/Desktop/aiasscode/12lab1.py
Enter student scores separated by space: 45 65 30 29
Enter student scores separated by space: 45 65 30 29
Sorted Scores: [29, 30, 45, 65]
PS C:\Users\Administrator\Desktop\aiasscode>

Sorted Scores: [29, 30, 45, 65]
Sorted Scores: [29, 30, 45, 65]
PS C:\Users\Administrator\Desktop\aiasscode> 
```

Explanation:

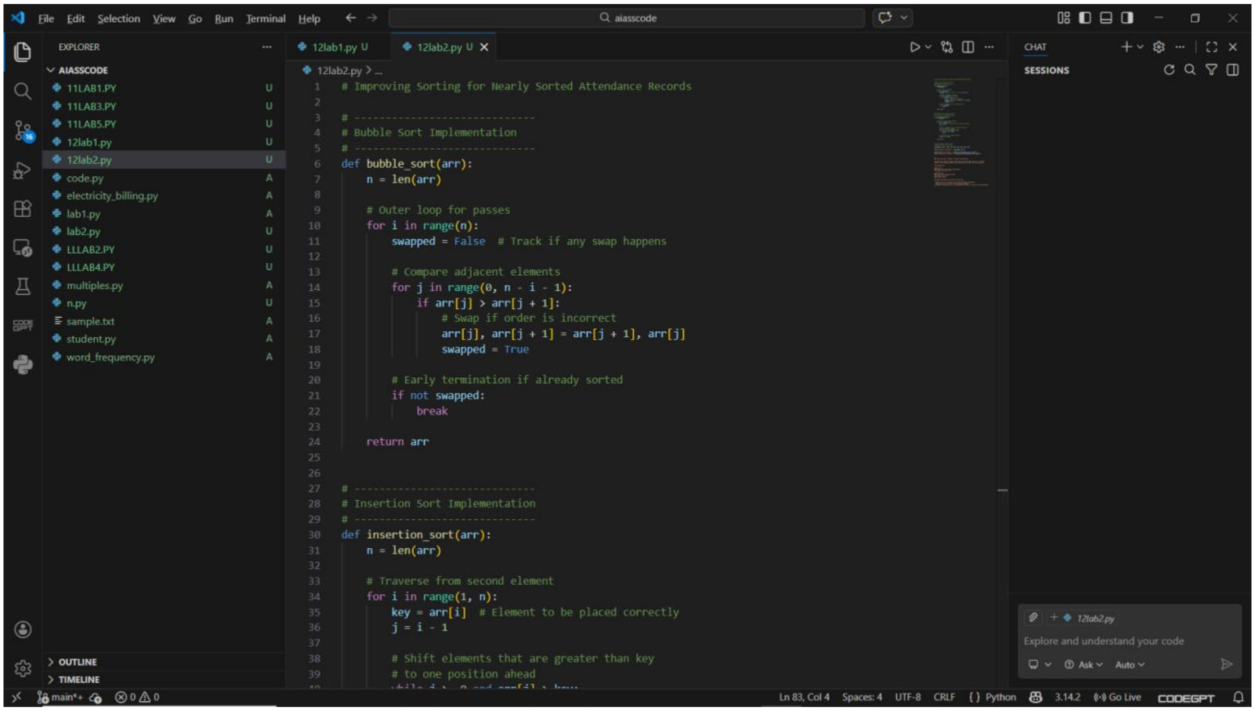
Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. After each pass, the largest element moves to its correct position at the end. Even with early termination, it still performs many comparisons, so it is not very efficient for nearly sorted data.

TASK2:

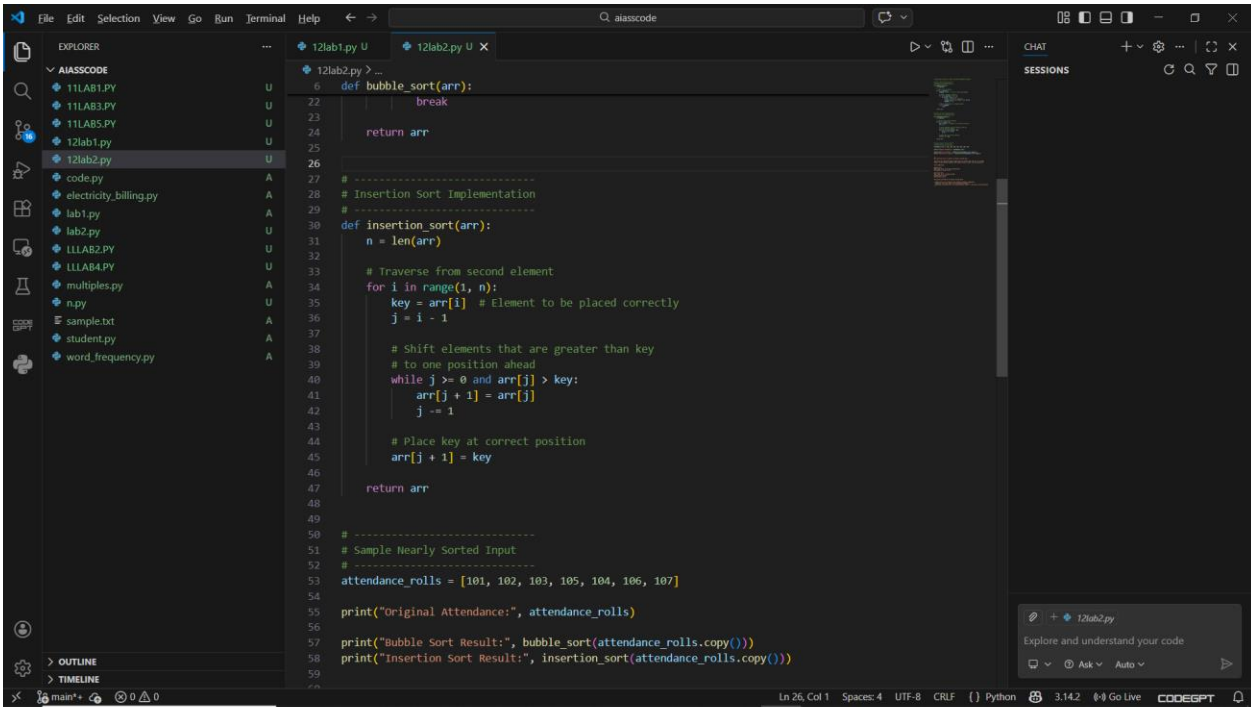
Prompt

I have an attendance system where student roll numbers are already nearly sorted, with only a few late updates.
Start with a Bubble Sort implementation, then review the scenario and suggest a more suitable sorting algorithm.
Generate Python code for both Bubble Sort and Insertion Sort with inline comments.
Explain why Insertion Sort performs better on nearly sorted data and compare execution behavior on nearly sorted input.

Code:

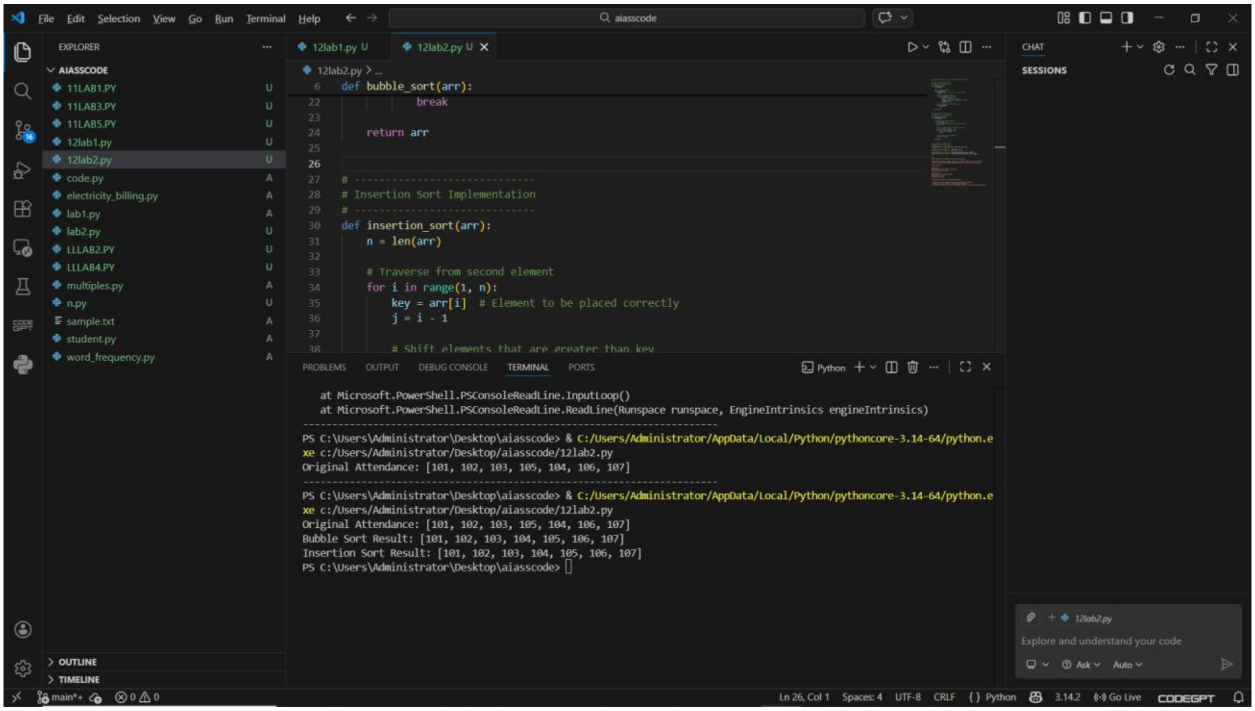


```python
# Improving Sorting for Nearly Sorted Attendance Records

# ----------------------------
# Bubble Sort Implementation
# ----------------------------
def bubble_sort(arr):
    n = len(arr)

    # Outer loop for passes
    for i in range(n):
        swapped = False  # Track if any swap happens

        # Compare adjacent elements
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap if order is incorrect
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # Early termination if already sorted
        if not swapped:
            break

    return arr


# ----------------------------
# Insertion Sort Implementation
# ----------------------------
def insertion_sort(arr):
    n = len(arr)

    # Traverse from second element
    for i in range(1, n):
        key = arr[i]  # Element to be placed correctly
        j = i - 1

        # Shift elements that are greater than key
        # to one position ahead
```



```python
def bubble_sort(arr):
            break

    return arr


# ----------------------------
# Insertion Sort Implementation
# ----------------------------
def insertion_sort(arr):
    n = len(arr)

    # Traverse from second element
    for i in range(1, n):
        key = arr[i]  # Element to be placed correctly
        j = i - 1

        # Shift elements that are greater than key
        # to one position ahead
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1

        # Place key at correct position
        arr[j + 1] = key

    return arr


# ----------------------------
# Sample Nearly Sorted Input
# ----------------------------
attendance_rolls = [101, 102, 103, 105, 104, 106, 107]

print("Original Attendance:", attendance_rolls)

print("Bubble Sort Result:", bubble_sort(attendance_rolls.copy()))
print("Insertion Sort Result:", insertion_sort(attendance_rolls.copy()))
```

Output



File Edit Selection View Go Run Terminal Help
aiasscode
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
main*+
Ln 26, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Go Live CODEGPT

```python
def bubble_sort(arr):
            break

    return arr


# ----------------------------
# Insertion Sort Implementation
# ----------------------------
def insertion_sort(arr):
    n = len(arr)

    # Traverse from second element
    for i in range(1, n):
        key = arr[i]  # Element to be placed correctly
        j = i - 1

        # Shift elements that are greater than key
```

```
   at Microsoft.PowerShell.PSConsoleReadLine.InputLoop()
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/12lab2.py
Original Attendance: [101, 102, 103, 105, 104, 106, 107]
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/12lab2.py
Original Attendance: [101, 102, 103, 105, 104, 106, 107]
Bubble Sort Result: [101, 102, 103, 104, 105, 106, 107]
Insertion Sort Result: [101, 102, 103, 104, 105, 106, 107]
PS C:\Users\Administrator\Desktop\aiasscode> 
```

Explanation:

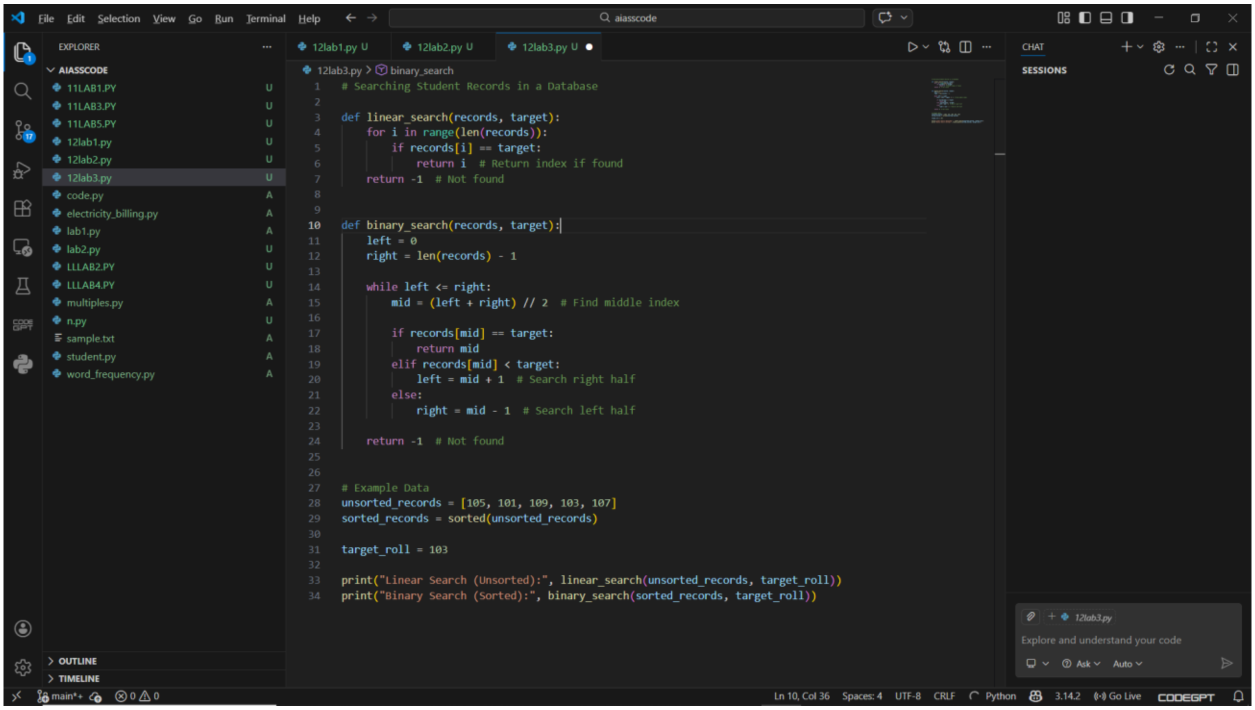
Insertion Sort takes one element at a time and places it in its correct position within the already sorted portion of the list. Since nearly sorted data already has most elements in place, only a few shifts are needed, making it faster and more efficient than Bubble Sort in this scenario

Task3:

Prompt

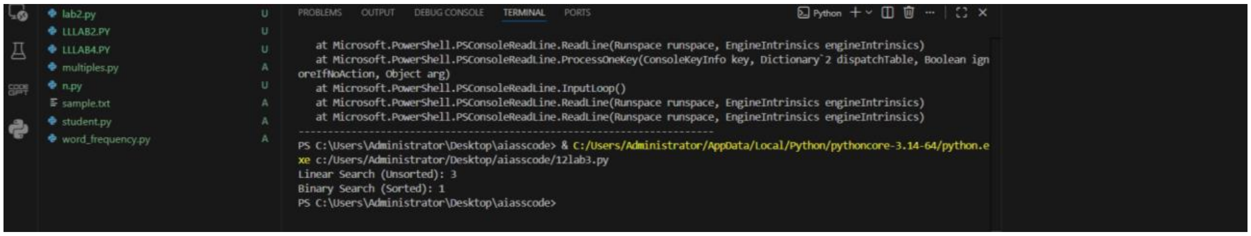
Create a Python program to search student records by roll number.
Implement Linear Search for unsorted data and Binary Search for sorted data.
Add docstrings explaining parameters and return values.
Explain when Binary Search is applicable and compare performance differences between Linear and Binary Search.
Include time complexity and a short observation comparing results on sorted vs unsorted lists.

Code:



```python
# Searching Student Records in a Database

def linear_search(records, target):
    for i in range(len(records)):
        if records[i] == target:
            return i  # Return index if found
    return -1  # Not found


def binary_search(records, target):
    left = 0
    right = len(records) - 1

    while left <= right:
        mid = (left + right) // 2  # Find middle index

        if records[mid] == target:
            return mid
        elif records[mid] < target:
            left = mid + 1  # Search right half
        else:
            right = mid - 1  # Search left half

    return -1  # Not found


# Example Data
unsorted_records = [105, 101, 109, 103, 107]
sorted_records = sorted(unsorted_records)

target_roll = 103

print("Linear Search (Unsorted):", linear_search(unsorted_records, target_roll))
print("Binary Search (Sorted):", binary_search(sorted_records, target_roll))
```

Output:



```
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
   at Microsoft.PowerShell.PSConsoleReadLine.ProcessOneKey(ConsoleKeyInfo key, Dictionary`2 dispatchTable, Boolean ign
oreIfNoAction, Object arg)
   at Microsoft.PowerShell.PSConsoleReadLine.InputLoop()
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/12lab3.py
Linear Search (Unsorted): 3
Binary Search (Sorted): 1
PS C:\Users\Administrator\Desktop\aiasscode>
```

Explanation:

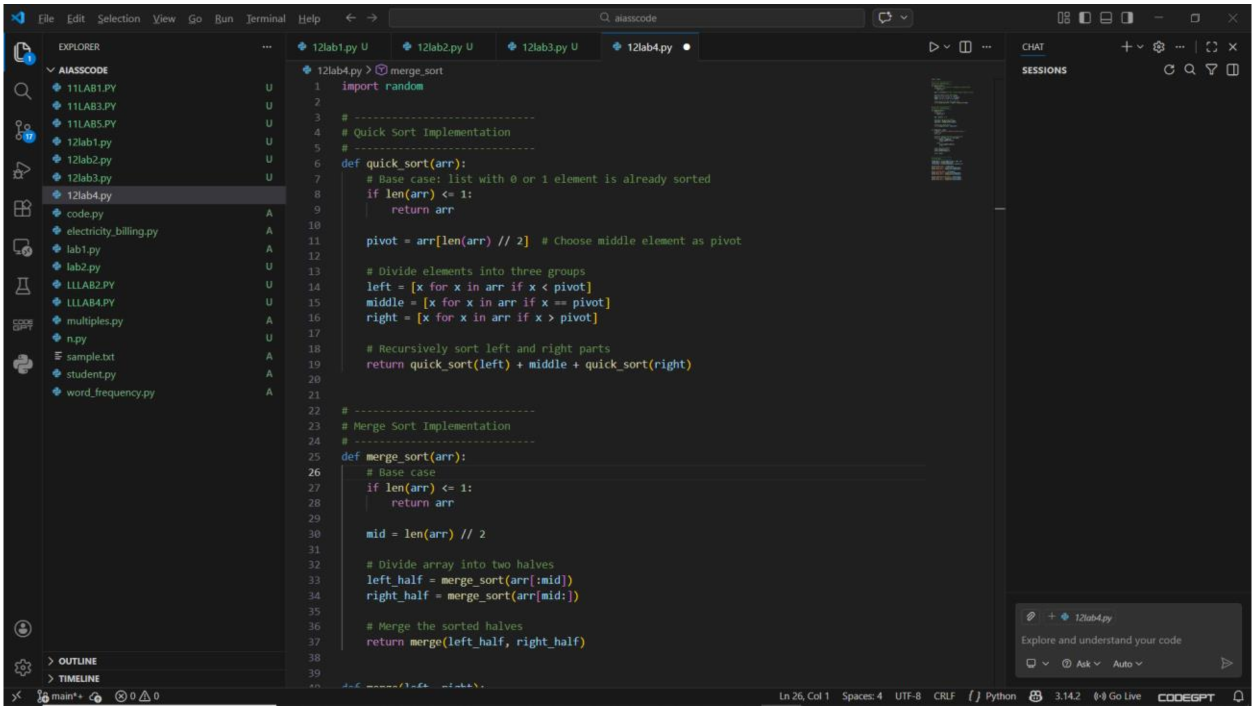
Binary Search is used to find a student roll number in a **sorted list** of records. It works by comparing the target value with the middle element and then repeatedly dividing the search range into halves until the element is found or the search space becomes empty. This method is faster than Linear Search for large datasets because it reduces the number of comparisons significantly (time complexity **O(log n)**)

Task4:

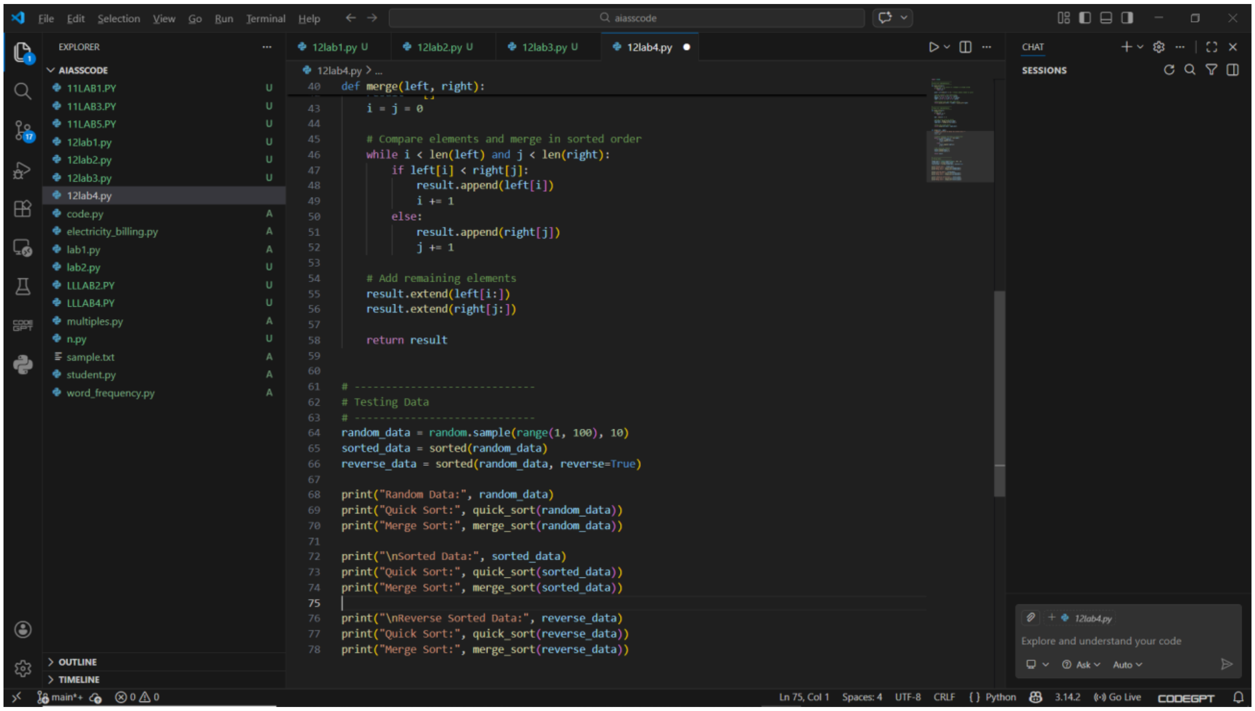
Prompt

I have partially written recursive functions for Quick Sort and Merge Sort.
Complete the recursive logic, add meaningful docstrings explaining parameters and return values, and explain how recursion works in each algorithm.
Then test both algorithms on random data, sorted data, and reverse-sorted data.
Also provide a comparison of time complexities and practical scenarios where one algorithm is preferred over the other

Code:

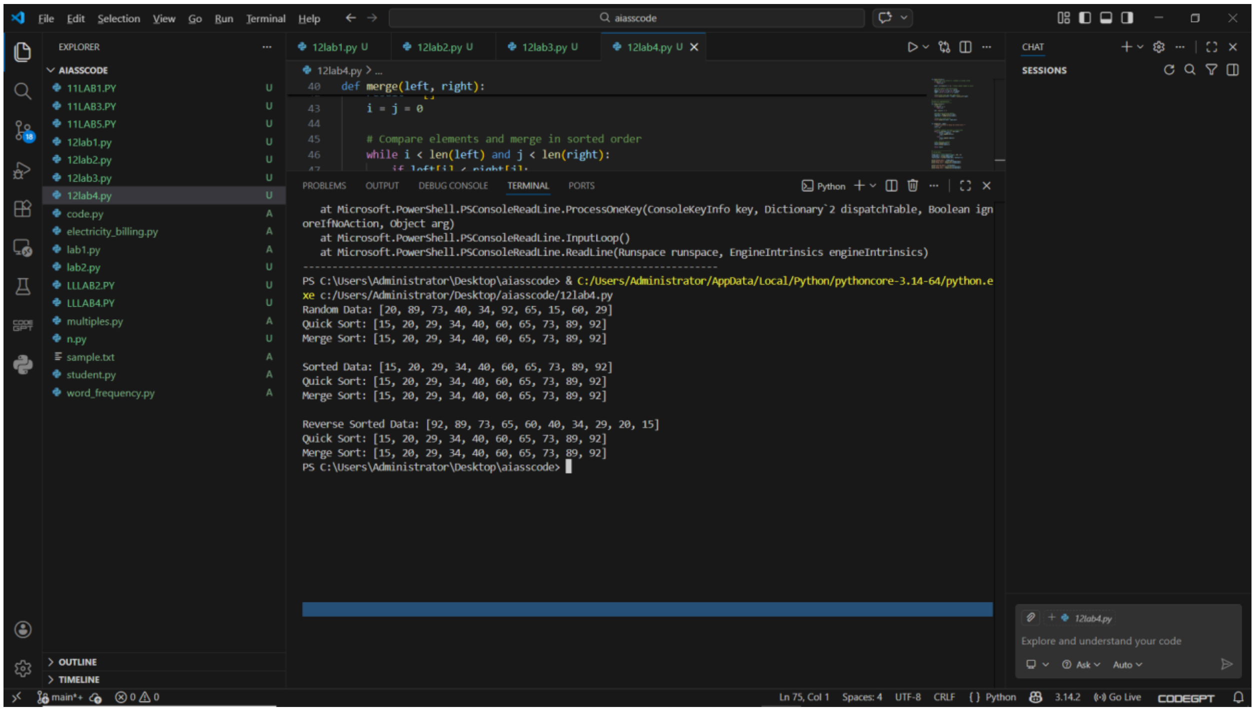


```python
import random

# ----------------------------
# Quick Sort Implementation
# ----------------------------
def quick_sort(arr):
    # Base case: list with 0 or 1 element is already sorted
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2]  # Choose middle element as pivot

    # Divide elements into three groups
    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]

    # Recursively sort left and right parts
    return quick_sort(left) + middle + quick_sort(right)


# ----------------------------
# Merge Sort Implementation
# ----------------------------
def merge_sort(arr):
    # Base case
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2

    # Divide array into two halves
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    # Merge the sorted halves
    return merge(left_half, right_half)


def merge(left, right):
```



```
def merge(left, right):
    i = j = 0

    # Compare elements and merge in sorted order
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # Add remaining elements
    result.extend(left[i:])
    result.extend(right[j:])

    return result


# ----------------------------
# Testing Data
# ----------------------------
random_data = random.sample(range(1, 100), 10)
sorted_data = sorted(random_data)
reverse_data = sorted(random_data, reverse=True)

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

print("\nSorted Data:", sorted_data)
print("Quick Sort:", quick_sort(sorted_data))
print("Merge Sort:", merge_sort(sorted_data))

print("\nReverse Sorted Data:", reverse_data)
print("Quick Sort:", quick_sort(reverse_data))
print("Merge Sort:", merge_sort(reverse_data))
```

Output:



```
def merge(left, right):
    i = j = 0

    # Compare elements and merge in sorted order
    while i < len(left) and j < len(right):
```

```
   at Microsoft.PowerShell.PSConsoleReadLine.ProcessOneKey(ConsoleKeyInfo key, Dictionary`2 dispatchTable, Boolean ign
oreIfNoAction, Object arg)
   at Microsoft.PowerShell.PSConsoleReadLine.InputLoop()
   at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/12lab4.py
Random Data: [20, 89, 73, 40, 34, 92, 65, 15, 60, 29]
Quick Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]
Merge Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]

Sorted Data: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]
Quick Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]
Merge Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]

Reverse Sorted Data: [92, 89, 73, 65, 60, 40, 34, 29, 20, 15]
Quick Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]
Merge Sort: [15, 20, 29, 34, 40, 60, 65, 73, 89, 92]
PS C:\Users\Administrator\Desktop\aiasscode>
```

Explanation:

Quick Sort:
It selects a pivot element, divides the list into smaller parts based on the pivot, and recursively sorts those parts until the list is fully sorted.

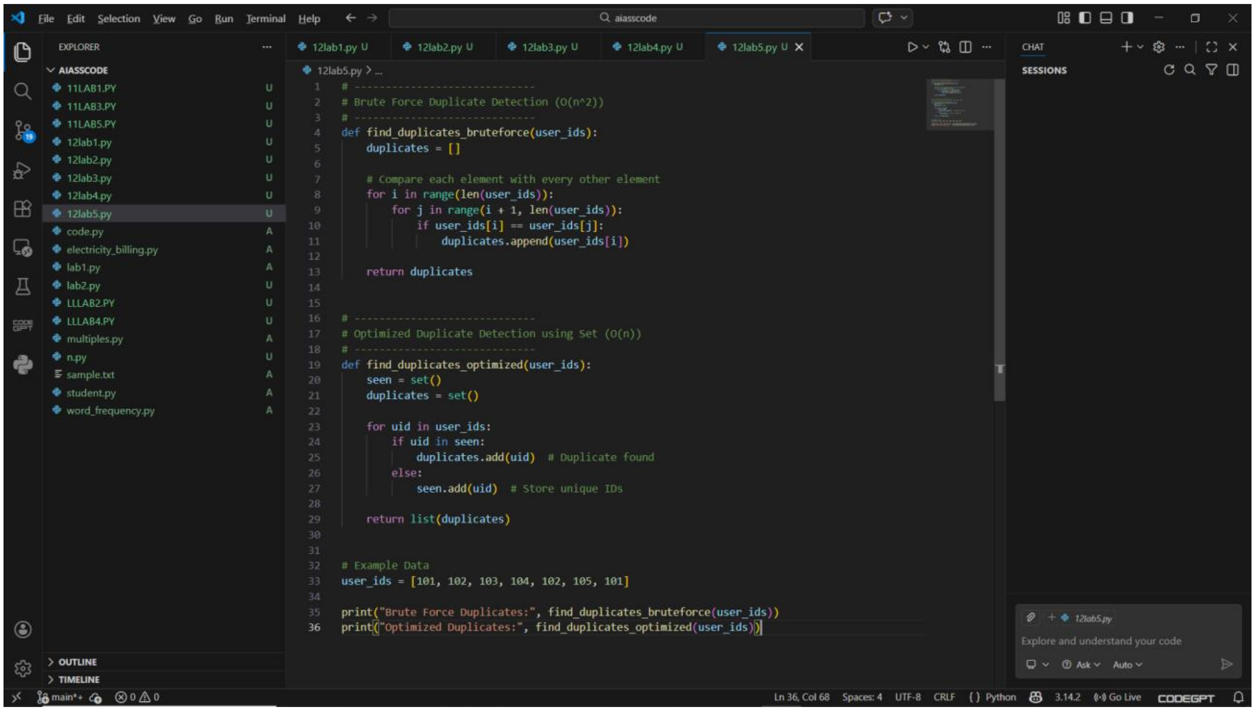
Merge Sort:
It divides the list into halves repeatedly until single elements remain, then merges them back together in sorted order using recursion

Task5:

Prompt

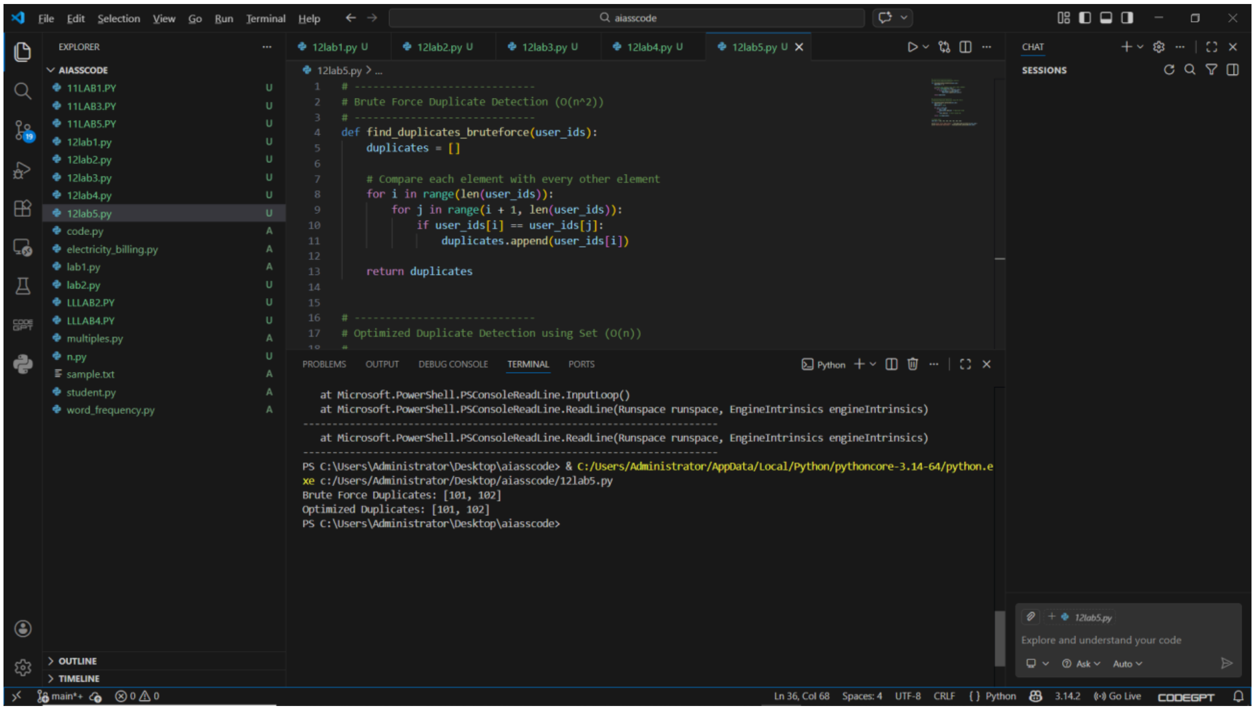
Write a Python program to detect duplicate user IDs in a dataset using a naive nested loop approach.
Analyze the time complexity of this method.
Suggest an optimized approach using sets or dictionaries and rewrite the algorithm with better efficiency.
Compare the execution behavior conceptually for large input sizes and explain how performance improves

Code:



```python
# ----------------------------
# Brute Force Duplicate Detection (O(n^2))
# ----------------------------
def find_duplicates_bruteforce(user_ids):
    duplicates = []

    # Compare each element with every other element
    for i in range(len(user_ids)):
        for j in range(i + 1, len(user_ids)):
            if user_ids[i] == user_ids[j]:
                duplicates.append(user_ids[i])

    return duplicates


# ----------------------------
# Optimized Duplicate Detection using Set (O(n))
# ----------------------------
def find_duplicates_optimized(user_ids):
    seen = set()
    duplicates = set()

    for uid in user_ids:
        if uid in seen:
            duplicates.add(uid)  # Duplicate found
        else:
            seen.add(uid)  # Store unique IDs

    return list(duplicates)


# Example Data
user_ids = [101, 102, 103, 104, 102, 105, 101]

print("Brute Force Duplicates:", find_duplicates_bruteforce(user_ids))
print("Optimized Duplicates:", find_duplicates_optimized(user_ids))
```

Output:



```python
# ----------------------------
# Brute Force Duplicate Detection (O(n^2))
# ----------------------------
def find_duplicates_bruteforce(user_ids):
    duplicates = []

    # Compare each element with every other element
    for i in range(len(user_ids)):
        for j in range(i + 1, len(user_ids)):
            if user_ids[i] == user_ids[j]:
                duplicates.append(user_ids[i])

    return duplicates


# ----------------------------
# Optimized Duplicate Detection using Set (O(n))
```

```
  at Microsoft.PowerShell.PSConsoleReadLine.InputLoop()
  at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
-----------------------------------------------------------------------
  at Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace runspace, EngineIntrinsics engineIntrinsics)
-----------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/12lab5.py
Brute Force Duplicates: [101, 102]
Optimized Duplicates: [101, 102]
PS C:\Users\Administrator\Desktop\aiasscode>
```

Explanation:

The brute-force method uses nested loops, so it compares every pair of user IDs, resulting in **$O(n^2)$** time complexity, which becomes very slow for large datasets.

The optimized method uses a **set** to store seen IDs and check duplicates in constant time, reducing the complexity to **O(n)**. This makes it much faster and more efficient for large inputs