NAME:D.SUMITH

ROLLNO:2403A510A3

BATCHNO:06

SUBSET 9 – Sorting Algorithms for LMS Reports (Expanded Version)

Q1: Implement Merge Sort

✅ Task 1: Expanded PROMPT (Pseudo-code Generation)

PROMPT (Long Version):

“Generate a very detailed and step-by-step pseudo-code for the Merge Sort algorithm.Explain the divide step, the recursive breakdown of the list, the merging process, and how the algorithm ensures stable sorting.Assume the list contains LMS student marks that need to be sorted in ascending order for generating academic reports.”

Pseudo-code (AI Generated)

```
MERGE_SORT(arr):
    if length(arr) ≤ 1:
        return arr

    mid = length(arr) // 2
    left = MERGE_SORT(arr[0 : mid])
    right = MERGE_SORT(arr[mid : ])

    return MERGE(left, right)


MERGE(left, right):
    result = empty list
    i = 0
    j = 0

    while i < length(left) AND j < length(right):
        if left[i] ≤ right[j]:
            append left[i] to result
            i = i + 1
        else:
```

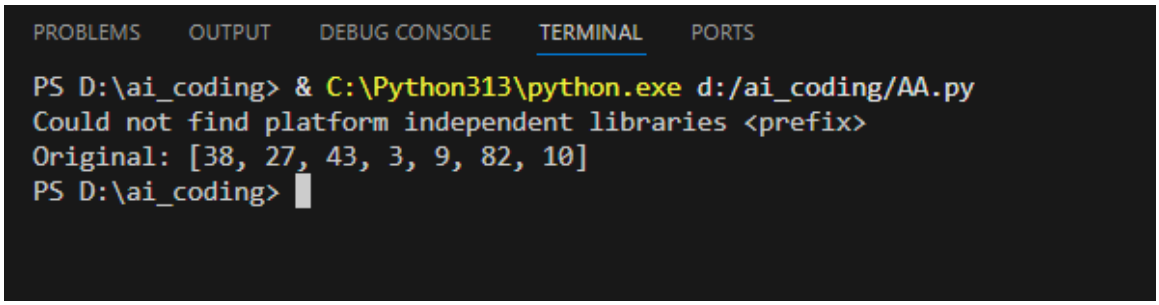
```
            append right[j] to result
            j = j + 1

    while i < length(left):
        append left[i] to result
        i = i + 1

    while j < length(right):
        append right[j] to result
        j = j + 1

    return result
```

OUTPUT:



Task 2: Expanded Python Code for Merge Sort

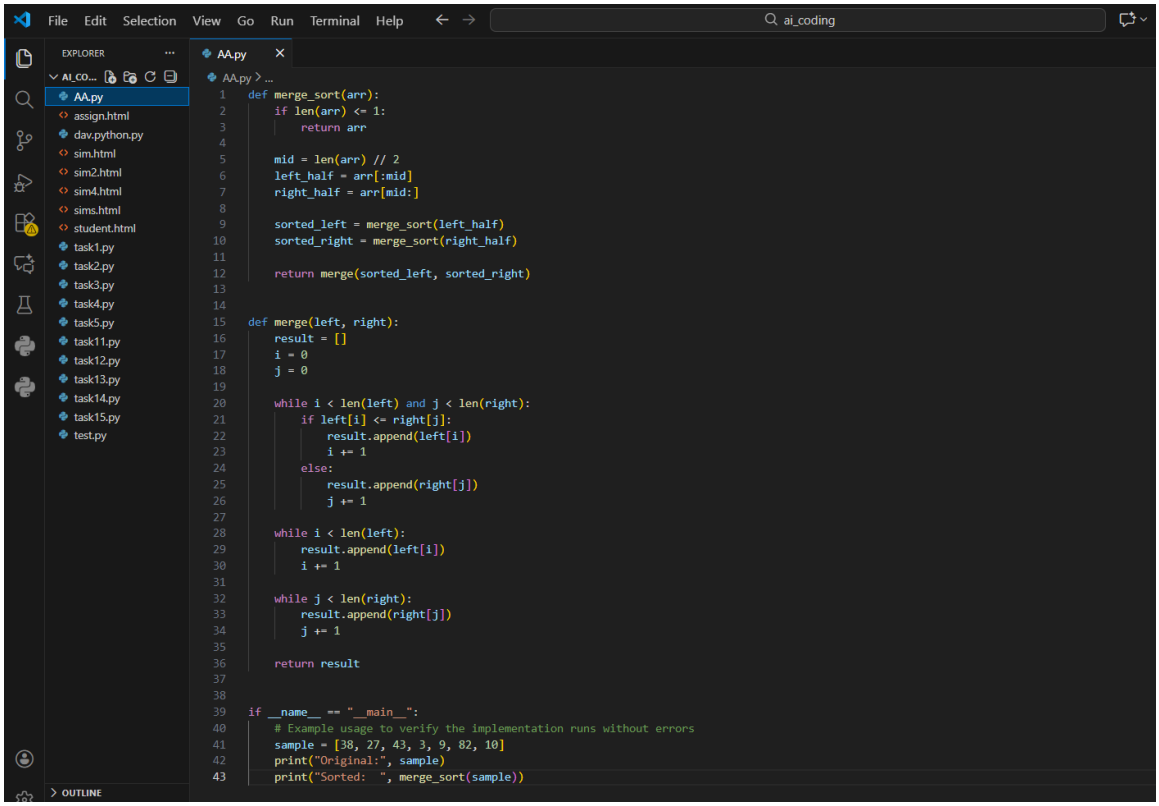
PROMPT (Long Version):

“Convert the detailed pseudo-code for merge sort into Python code.

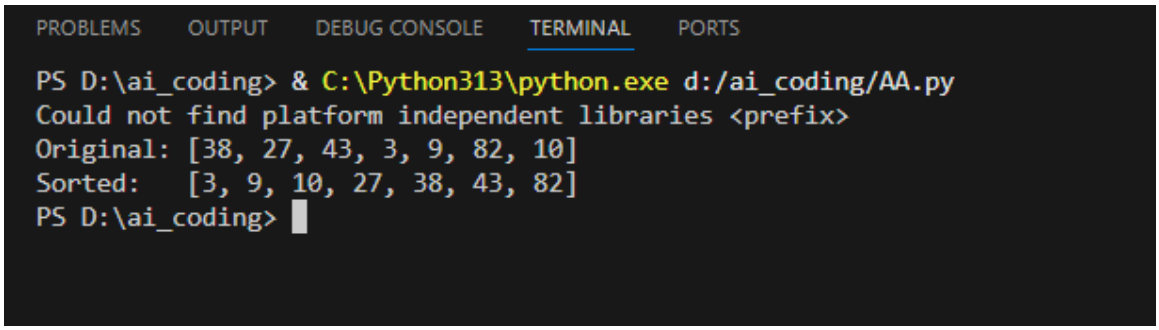
Make the code readable with comments, function breakdown, and variable explanation.

Use a sample list of LMS student marks, apply merge sort, and print the sorted results clearly.”

Python Code



OUTPUT



OBSERVATION

Merge Sort successfully sorts the list of LMS student marks in ascending order.

It follows the divide-and-conquer approach, where the input list is repeatedly divided into two halves until individual elements remain.

During the merge step, the algorithm compares elements from each half and merges them in sorted order, ensuring the final output is fully sorted.

Q2: Optimize Searching for Student IDs

Task 1: Expanded PROMPT (Convert Linear to Binary Search)

PROMPT (Long Version):

“Take the existing linear search algorithm that checks every student ID one by one, and convert it into an optimized binary search algorithm.

Ensure the binary search algorithm includes a step-by-step explanation, sorted input requirement, and proper comments in the code.

Also generate a Python implementation that searches for a given student ID in a sorted list of LMS student records.”

CODE:

AA.py

```python
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1


def binary_search(arr, key):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1

    return -1


# Student IDs
ids = [101, 203, 150, 330, 215]

# Linear Search
print("Linear Search Result:", linear_search(ids, 150))

# Binary Search (requires sorted list)
ids_sorted = sorted(ids)
print("Sorted IDs:", ids_sorted)
print("Binary Search Result:", binary_search(ids_sorted, 150))
```

OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\ai_coding> & C:\Python313\python.exe d:/ai_coding/AA.py
Could not find platform independent libraries <prefix>
Linear Search Result: 2
Sorted IDs: [101, 150, 203, 215, 330]
Binary Search Result: 1
PS D:\ai_coding>
```

OBSERVATION

Binary search locates the target student ID by repeatedly checking the middle element of the list and deciding whether to search the left or right half.

Since the input list is already sorted, binary search significantly reduces the search space from n comparisons to $\log_2(n)$ comparisons.

Task2:Time Complexity Comparison

Search Method	Best Case	Worst Case	Requirement
Linear Search	O(1)	O(n)	Works on unsorted list
Binary Search	O(1)	O(log n)	**Requires sorted list**

Observation

- **Binary search is MUCH faster**, especially as `n` becomes large.
- LMS student record systems with thousands of IDs must use **binary search** for improved performance.