LAB_ASSIGNMENT 12.2

NAME:-ANUSHA PEDDAPELLI

BATCH:-06

COURSE:-AI ASSISTED CODING

• **Task 1:** Use AI to generate a Python program that implements the Merge Sort algorithm.

PROMPT:-

"Write a Python program that implements the Merge Sort algorithm. The program should:

1. Define a function merge_sort(arr) that takes a list of numbers as input and returns the sorted list.
2. Use the divide-and-conquer approach to recursively split the list into halves until single-element lists are reached.
3. Merge the sublists back together in sorted order.
4. Include a main section where the user can input a list of numbers, and the program prints the sorted result.
5. Add comments explaining each step of the algorithm."

CODE:-

```python
12.1_LAB_ASS.py > ...
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0

        # Merge the sorted halves
        while i < len(left_hal  (variable) right_half: Any
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        # Copy remaining elements from left_half
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
```

```python
            k += 1

        # Copy remaining elements from right_half
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

if __name__ == "__main__":
    try:
        arr = list(map(int, input("Enter numbers separated by spaces: ").split()))
        print("Original array:", arr)
        merge_sort(arr)
        print("Sorted array:", arr)
    except ValueError:
        print("Please enter only integers separated by spaces.")
```

OUTPUT:-

```
I/12.1_LAB_ASS.py"
Enter numbers separated by spaces: 2
Original array: [2]
Sorted array: [2]
PS C:\Users\PEDDAPELLI ANUSHA\OneDrive\Desktop\Btech.2nd yr>
```

OBSERVATION:-

- The program asks the user to input a list of numbers.

- It sorts the list using the Merge Sort algorithm.
- The sorted array is displayed.
- Merge Sort divides the array recursively and merges sorted halves, ensuring O(n log n) time complexity.

Task 2: Use AI to create a binary search function that finds a target element in a sorted list.

PROMPT:-

"Write a Python function binary_search(arr, target) that takes a sorted list arr and a target element, and returns the index of the target if found, or -1 if not found. Include comments explaining each step of the algorithm."

CODE:-

```
def binary_search(arr, target):  # Fixed typo: 'ddef' to 'def'
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid  # Target found
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1  # Target not found

if __name__ == "__main__":
    try:
        arr = list(map(int, input("Enter sorted numbers separated by spaces: ").split()))
        target = int(input("Enter the target element to search: "))
        result = binary_search(arr, target)
        if result != -1:
            print(f"Element {target} found at index {result}.")
        else:
            print(f"Element {target} not found in the list.")
    except ValueError:
        print("Please enter only integers separated by spaces.")
```

OUTPUT:-

```
1/12.1.2.LAB_ASS.py"
Enter sorted numbers separated by spaces: 2
Enter the target element to search: 2
Element 2 found at index 0.
PS C:\Users\PEDDAPELLI ANUSHA\OneDrive\Desktop\Btech.2nd yr>
```

OBSERVATION:-

Binary Search works only on sorted lists.

- It repeatedly divides the search interval in half, reducing the number of comparisons.
- Time complexity is O(log n), which is much faster than linear search for large lists.
- If the target is not present, the function returns -1.
- Handles edge cases like empty lists or single-element lists.

Task 3: Use AI to suggest the most efficient search and sort algorithms for this use case.

PROMPT:-

"Suggest the most efficient search and sort algorithms for a use case where we have a large dataset of numbers that needs to be sorted and frequently searched. Explain why each algorithm is suitable, considering time and space complexity."

CODE:-

```python
def merge_sort(arr):
    """Sorts the array using Merge Sort (O(n log n))"""
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    # Merge two sorted halves
    merged = []
    i = j = 0
    while i (variable) left: Any | list ):
        if left[i] < right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged
```

```python
def binary_search(arr, target):
    """Searches for target in a sorted array using Binary Search (O(log n))"""
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1  # target not found

# Example usage
data = [34, 7, 23, 32, 5, 62, 78, 1]
sorted_data = merge_sort(data)
print("Sorted Data:", sorted_data)

target = 23
index = binary_search(sorted_data, target)
if index != -1:
    print(f"Target {target} found at index {index}.")
else:
    print(f"Target {target} not found.")
```

OUTPUT:-

```
Sorted Data: [1, 5, 7, 23, 32, 34, 62, 78]
Target 23 found at index 3.
PS C:\Users\PEDDAPELLI ANUSHA\OneDrive\Desktop\Btech.2nd yr>
```

OBSERVATION:-

- Sorting: For large datasets, Merge Sort or Quick Sort is efficient:
    - Merge Sort: O(n log n) time, stable, good for linked lists or large datasets.
    - Quick Sort: O(n log n) average time, in-place sorting, very fast in practice.
- Searching: If the dataset is sorted, Binary Search is ideal:
    - Time complexity: O(log n).
    - Much faster than linear search for large datasets.