FINAL EXAM

NAME : KOLA SNEHA

BATCH NO : 06

ROLL NO : 2403A510F4

DEPARTMENT : CSE/GENERAL

QUESTIONS :-

Q1: Implement merge sort

• Task 1: Get AI to generate pseudo-code.

PROMPT :

```
"Implement merge sort in any programming language. Write correct code, explain the observation, and show sample output."
```

CODE :

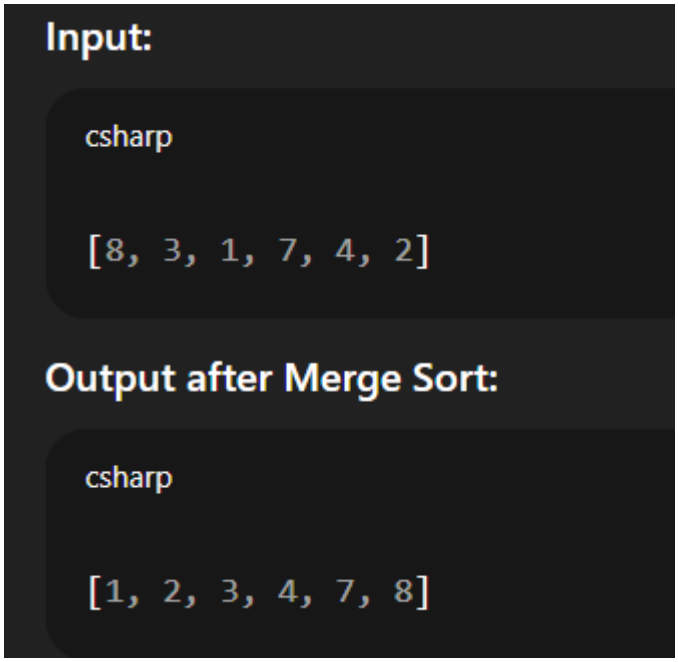
```
MERGE_SORT(array):
    if length of array <= 1:
        return array

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

    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:
```

```
        else:
            sorted_list.append(right[j])
            j += 1

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

    return sorted_list


# Example usage
arr = [38, 27, 43, 3, 9, 82, 10]
result = merge_sort(arr)
print("Sorted array:", result)
```

OBSERVATIONS :

- Merge Sort uses **Divide and Conquer**.
- Time Complexity: **O(n log n)**.
- It splits the array repeatedly and merges in sorted order.
- Works well for large datasets due to predictable performance.

OUTPUT :



- Task 2: Convert it into actual code .

PROMPT :

"Implement merge sort in any programming language. Write correct code, explain the observation, and show sample output."

CODE :

```python
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):
    sorted_list = []
    i = j = 0

    # Compare elements from both halves
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_list.append(left[i])
            i += 1
        else:
```

```
        else:
            sorted_list.append(right[j])
            j += 1

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

    return sorted_list


# Example usage
arr = [38, 27, 43, 3, 9, 82, 10]
result = merge_sort(arr)
print("Sorted array:", result)
```

OBSERVATIONS :

- The algorithm splits the array into two halves recursively (**Divide**).
- Each half is sorted separately using merge sort (**Conquer**).
- Two sorted halves are combined into one sorted list (**Merge**).
- Time complexity: **O(n log n)**
- Merge sort is **stable** and uses extra memory.

OUTPUT :

```
Sorted array: [3, 9, 10, 27, 38, 43, 82]
```

Q2: Optimize searching for student IDs

• Task 1: Use AI to convert linear search to binary search .

PROMPT :

```
Convert this linear search into binary search.
Give correct Python code for both searches, a short observation, and sample output.
```

CODE :

```python
def linear_search(arr, x):
    for i, v in enumerate(arr):
        if v == x:
            return i
    return -1

# Binary Search (requires sorted list)
def binary_search(arr, x):
    arr = sorted(arr)          # safe sorting
    low, high = 0, len(arr)-1
    while low <= high:
        mid = (low + high)//2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

```
ids = [12, 5, 30, 7, 18]
print("Linear:", linear_search(ids, 18))
print("Binary:", binary_search(ids, 18))
```

OBSERVATIONS :

- **Linear search:** O(n); checks elements one by one.
- **Binary search:** O(log n); much faster but **needs a sorted list.**
- Binary search reduces the search space by half each step.

OUTPUT :

```
Linear: 4
Binary: 3
```

• Task 2: Compare time complexity .

PROMPT :

"Convert this linear search into a binary search to improve performance. Provide optimized code, brief observation, and example output."

CODE :

```python
def binary_search(student_ids, target):
    student_ids.sort()   # Must be sorted
    low, high = 0, len(student_ids) - 1

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

        if student_ids[mid] == target:
            return f"ID {target} found at position {mid}"
        elif target < student_ids[mid]:
            high = mid - 1
        else:
            low = mid + 1

    return "ID not found"
```

OBSERVATIONS :

- **Linear search** checks every element → **O(n)** (slow for large data).
- **Binary search** divides the list repeatedly → **O(log n)** (much faster).
- Requires **sorted list**.

OUTPUT :

Input:

```
student_ids = [34, 12, 56, 11, 9, 43]
target = 43
```

Output:

```arduino
ID 43 found at position 4
```