3.4 MERGE SORT

Question:

Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 7 43 86 23 1 8 9. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array.

AIM

To implement Merge Sort in Python and count the number of comparisons made during the sorting process.

ALGORITHM

1. Define a recursive function merge_sort that splits the array into halves.
2. Define a merge function that merges two sorted halves and counts comparisons.
3. Use a global or nonlocal variable to track the number of comparisons.
4. Print the sorted array and the total comparison count.

PROGRAM

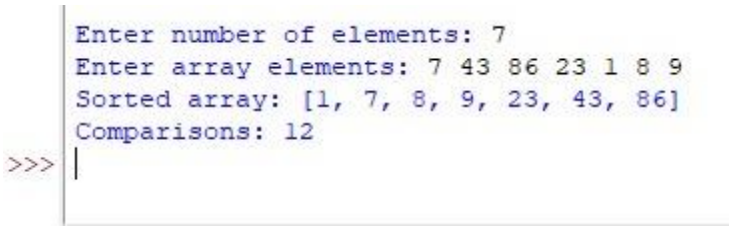
```
comparison_count = 0

def merge_sort_count(arr):
    global comparison_count
    if len(arr) > 1:
        mid = len(arr) // 2
        L = merge_sort_count(arr[:mid])
        R = merge_sort_count(arr[mid:])
        return merge_count(L, R)
    return arr

def merge_count(left, right):
    global comparison_count
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        comparison_count += 1
        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

def run_merge_sort_count():
    global comparison_count
    comparison_count = 0
    N = int(input("Enter number of elements: "))
    arr = list(map(int, input("Enter array elements: ").split()))
    sorted_arr = merge_sort_count(arr)
    print("Sorted array:", sorted_arr)
    print("Comparisons:", comparison_count)

run_merge_sort_count()
```

Input:

7

7 43 86 23 1 8 9

Output:



RESULT:

Thus the program is successfully executed, and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: O(n log n)
- Space Complexity: (n).