ACADEMIC YEAR: 2024-25

Course: Analysis of Algorithm Lab

Course code: CSL401

Year/Sem: SE/IV

Experiment No.: 02
Aim: To implement the Merge Sort Technique using DAC.
Name: SOHAM HEMENDRA RAUT
Roll Number: 24
Date of Performance: 16/01/2025
Date of Submission: 30/01/2025

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission.	10	
Total	**20**	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	5	3	2
Understanding	5	3	2
Journal work and timely submission.	10	8	4

Checked by

Name of Faculty : Mrs. Komal Champanerkar

Signature :

Date :

- **Aim: To implement the Merge Sort Technique using DAC.**
- **Theory:**
- **Merge Sort Technique :**

  Merge Sort is a divide-and-conquer algorithm that works by recursively splitting a large array into smaller sub-arrays until each sub-array contains only one element. Once this division is complete, the algorithm merges the sub-arrays back together in a manner that results in a sorted array.

  Steps to follow in merge sort:

  I) Divide: The array is recursively divided into two halves until each sub-array has only one element.

  II) Conquer: During this merge step, the smallest elements of the two sub-arrays are compared, and they are merged in sorted order. This process continues recursively until all sub-arrays are merged back into a single sorted array.

- **Algorithms (Merge Sort):**

  Step 1: Find the middle index of an array and divide the array into two halves.
  Step 2: Repeat step 1 till each element of an array is separated into a single element.
  Step 3: Compare each element of the two sub-arrays and merge them into a sorted order.
  Step 4: Repeat the process until the entire array is sorted.

- **Complexity:**

  Best Case Analysis: *O(nlogn)*
  Worst Case Analysis: *O(nlogn)*
  Average Case Analysis: *O(nlogn)*

❖ Program: Merge Sort:

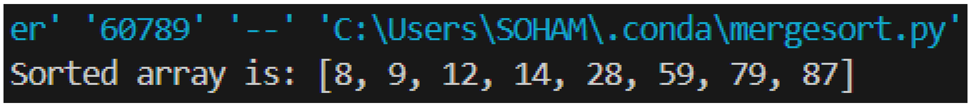
```
def merge_sort(arr, left, right):
    # This will divide the array into halves
    if left < right:
        mid = (left + right) // 2
        # this is wall call the fucntion recursively
        merge_sort(arr, left, mid)       # This will make the left side and so on
        merge_sort(arr, mid + 1, right)           # this will make the right side and
        merge(arr, left, mid, right)

def merge(arr, left, mid, right):
    i = left     # Pointer for the left sub-array
    j = mid + 1 # Pointer for the right sub-array

    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
            i += 1
        else:
            temp = arr[j]           # Store the value to be moved
            for p in range(j, i, -1):
                arr[p] = arr[p - 1]
            arr[i] = temp
            i += 1
            j += 1
            mid += 1
arr = [14, 28, 9, 59, 8, 87, 12, 79]

n = len(arr)
merge_sort(arr, 0, n - 1)
print("Sorted array is:", arr)
```

❖ Output:



```
er' '60789' '--' 'C:\Users\SOHAM\.conda\mergesort.py'
Sorted array is: [8, 9, 12, 14, 28, 59, 79, 87]
```

❖ **Conclusion:** Merge Sort is an efficient sorting algorithm that uses a divide-and-conquer approach. Its time complexity is O(n log n) in all cases, which makes it faster than algorithms with quadratic time complexity. However, Merge Sort has a space complexity of O(n) because it requires additional space for merging the sorted subarrays.