Chandpur Science And Technology University

Lab Report

Course Code : CSE 2201

Course Title : Algorithm Design and Analysis

Experiment no : 02

Experiment Name: Divide and Conquer (Mergesort)

Submitted By:

Name: Turja Chakraborty

ID: B210101001

Program: B.Sc. in CSE

Submitted To:

Name: Mustafizur Rahaman

Lecturer Dept. of CSE

Chandpur science and technology university

Merge Sort & Step Analysis:

1. Objective:

To understand, implement, and analyze the **Merge Sort** algorithm for sorting an array of numbers. The goal is to evaluate its performance both theoretically and practically through step-by-step analysis.

2)Algorithm:

Merge Sort is a **divide-and-conquer** sorting algorithm that works as follows:

- **Divide**: Split the array into two halves.
- **Conquer**: Recursively sort each half.
- **Combine**: Merge the sorted halves to produce the final sorted array.

3) Theoretical Solution of the Given Problem

- **Best Case Time Complexity**: O(n log n)
- **Average Case Time Complexity**: O(n log n)
- **Worst Case Time Complexity**: O(n log n)

- **Space Complexity**: O(n) – due to the temporary arrays used during merging

The time complexity remains consistent because:

- Each recursive level does **O(n)** work for merging.
- There are **log n** levels due to the division into halves.

4. Practical Work

a. Pseudocod

```
MergeSort(arr):

    if length of arr > 1:

        mid = length(arr) // 2

        left = arr[0..mid-1]

        right = arr[mid..end]


        MergeSort(left)

        MergeSort(right)
```

```
    Merge(left, right, arr)


Merge(left, right, arr):
  i = j = k = 0
  while i < length(left) and j < length(right):
    if left[i] <= right[j]:
      arr[k] = left[i]
      i += 1
    else:
      arr[k] = right[j]
      j += 1
    k += 1

  while i < length(left):
    arr[k] = left[i]
    i += 1
    k += 1
```

```
while j < length(right):
    arr[k] = right[j]
    j += 1
    k += 1
```

b. Source Code:

```c
#include <stdio.h>


int stepCount = 0; // Global step counter

void merge(int arr[], int l, int mid, int r) {
    int n1 = mid - l + 1;
    int n2 = r - mid;

    int a[n1];
    int b[n2];

    for (int i = 0; i < n1; i++) {
        a[i] = arr[l + i];
        stepCount++;
    }

    for (int i = 0; i < n2; i++) {
        b[i] = arr[mid + 1 + i];
        stepCount++;
```

```
    }

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        stepCount++;
        if (a[i] <= b[j]) {
            arr[k] = a[i];
            i++;
        } else {
            arr[k] = b[j];
            j++;
        }
        k++;
        stepCount++;
    }

    while (i < n1) {
        arr[k] = a[i];
        i++;
        k++;
        stepCount++; // Assignment
    }

    while (j < n2) {
        arr[k] = b[j];
        j++;
        k++;
        stepCount++; // Assignment
    }
}

void mergesort(int arr[], int l, int r) {
    if (l < r) {
        int mid = l + (r - l) / 2;
```

```c
        mergesort(arr, l, mid);
        mergesort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }
}


int main() {
    int arr[] = {10, 44, 66, 22, 46, 24, 12, 16};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    mergesort(arr, 0, size - 1);

    printf("\n\nSorted array:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n\nTotal steps: %d\n", stepCount);

    return 0;
}
```

Output:

Original array:

10 44 66 22 46 24 12 16

Sorted array:

10 12 16 22 24 44 46 66

Total steps: 64

5. Analysis Table :

Input Size (n)	Number of Comparisons
10	~30
100	~700
1,000	~10,000
10,000	~130,000

Observations:

- Merge Sort consistently shows **O(n log n)** performance regardless of the input's order.
- It is stable and works well for large datasets.
- Requires extra space due to recursive calls and temporary arrays.

Challenges:

- Implementing the merge step carefully to handle edge cases.
- Understanding recursion depth and memory usage.
- Optimizing merge for in-place sorting requires advanced techniques.