



# **MERGE SORT PERFORMANCE ANALYSIS REPORT**

Name – Aman Maggon

SAP ID – 590015673

Batch - 34

Course – Design Analysis Algorithm

Git hub repository link:

[https://github.com/AmanMaggon/DAALAB\\_-AmanMaggon-\\_590015673](https://github.com/AmanMaggon/DAALAB_-AmanMaggon-_590015673)



## 1. Merge Sort Algorithm

Merge Sort is a sorting technique that uses the divide and conquer method.

- **Divide:** Break the array into two smaller halves.
- **Conquer:** Sort each half separately using recursion.
- **Combine:** Merge the two sorted halves into one sorted array.

Time Complexity For Merge Sort

- Best Case  $\rightarrow O(n \log n)$
  - Average Case  $\rightarrow O(n \log n)$
  - Worst Case  $\rightarrow O(n \log n)$
- 

## 2. C Source Code

### INPUT-

```
#include <stdio.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void merge(int arr[], int low, int mid, int high)
```

```
{
```

```
    int n1 = (mid - low) + 1;
```

```
    int n2 = (high - mid);
```

```
    int Left[n1];
```

```
    int Right[n2];
```

```
    for (int i = 0; i < n1; i++)
```

```
    {
```

```
        Left[i] = arr[low + i];
```

```
    }
```

```
    for (int j = 0; j < n2; j++)
```

```
    {
```

```
        Right[j] = arr[mid + 1 + j];
```

```
    }
```

```

int left = 0, right = 0, k = low;
while (left < n1 && right < n2)
{
    if (Left[left] < Right[right])
    {
        arr[k] = Left[left];
        left++;
        k++;
    }
    else
    {
        arr[k] = Right[right];
        right++;
        k++;
    }
}
while (left < n1)
{
    arr[k++] = Left[left++];
}
while (right < n2)
{
    arr[k++] = Right[right++];
}
}

void merge_sort(int arr[], int low, int high)
{
    if (low >= high)
    {
        return;
    }
}

```

```

else
{
    int mid = (high - low) / 2 + low;
    merge_sort(arr, low, mid);
    merge_sort(arr, mid + 1, high);
    merge(arr, low, mid, high);
}
}

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
{
    // Test Case 1 - Small Array
    {
        int arr[] = {87, 34, 21, 56, 12};
        int n = sizeof(arr) / sizeof(arr[0]);
        printf("\n--- Test Case 1 (Small array) ---\n");
        printf("Input: ");
        printArray(arr, n);
        merge_sort(arr, 0, n - 1);
        printf("Output: ");
        printArray(arr, n);
    }
}

```

```
// Test Case 2 - Already Sorted Array
{
    int arr[] = {11, 12, 13, 14, 15, 16, 17};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 2 (Already sorted) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
    printf("Output: ");
    printArray(arr, n);
}
```

```
// Test Case 3 – Reverse sorted array
{
    int arr[] = {9, 8, 7, 6, 5, 4, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 3 (Reverse sorted) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
    printf("Output: ");
    printArray(arr, n);
}
```

```
// Test Case 4 – Array with duplicates
{
    int arr[] = {4, 2, 2, 4, 1, 1, 3, 3, 5, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 4 (Duplicates) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
}
```

```

    printf("Output: ");
    printArray(arr, n);
}

// Test Case 5 – Array with negative numbers
{
    int arr[] = {-5, 10, -2, 7, -9, 0, 3, -1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 5 (With negatives) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
    printf("Output: ");
    printArray(arr, n);
}

// Test Case 6 – Odd-sized array
{
    int arr[] = {11, 3, 7, 2, 9, 14, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 6 (Odd-sized array, n=7) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
    printf("Output: ");
    printArray(arr, n);
}

// Test Case 7 – Even-sized array
{
    int arr[] = {20, 11, 5, 8, 14, 2, 9, 7, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

```

```

printf("\n--- Test Case 7 (Even-sized array, n=10) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

// Test Case 8 – Large random array (n=1000)
{
    int n = 1000;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 10000;
    }
    printf("\n--- Test Case 8 (Large random array, n=1000) ---\n");
    merge_sort(arr, 0, n - 1);
    printf("Output (first 20 elements): ");
    for (int i = 0; i < 20; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Test Case 9 – Very large random array (n=10000)
{
    int n = 10000;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100000;
    }
}

```



```

printf("\n--- Test Case 9 (Very large random array, n=10000) ---\n");
merge_sort(arr, 0, n - 1);
printf("Output (first 20 elements): ");
for (int i = 0; i < 20; i++)
    printf("%d ", arr[i]);
printf("\n");
}

// Test Case 10 – All elements same
{
    int arr[] = {99, 99, 99, 99, 99, 99, 99, 99, 99, 99};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\n--- Test Case 10 (All elements same) ---\n");
    printf("Input: ");
    printArray(arr, n);
    merge_sort(arr, 0, n - 1);
    printf("Output: ");
    printArray(arr, n);
}

return 0;
}

```

## OUTPUT-

```
PS D:\OneDrive\Desktop\DAA LAB> cd "d:\OneDrive\Desktop\DAA LAB\" ; if ($?) { gcc mergesort.c -o mergesort } ; if ($?) { .\mergesort }

--- Test Case 1 (Small array) ---
Input: 87 34 21 56 12
Output: 12 21 34 56 87

--- Test Case 2 (Already sorted) ---
Input: 11 12 13 14 15 16 17
Output: 11 12 13 14 15 16 17

--- Test Case 3 (Reverse sorted) ---
Input: 9 8 7 6 5 4 3
Output: 3 4 5 6 7 8 9

--- Test Case 4 (Duplicates) ---
Input: 4 2 2 4 1 1 3 3 5 5
Output: 1 1 2 2 3 3 4 4 5 5

--- Test Case 5 (With negatives) ---
Input: -5 10 -2 7 -9 0 3 -1
Output: -9 -5 -2 -1 0 3 7 10

--- Test Case 6 (Odd-sized array, n=7) ---
Input: 11 3 7 2 9 14 1
Output: 1 2 3 7 9 11 14

--- Test Case 7 (Even-sized array, n=10) ---
Input: 20 11 5 8 14 2 9 7 3 1
Output: 1 2 3 5 7 8 9 11 14 20

--- Test Case 8 (Large random array, n=1000) ---
Output (first 20 elements): 3 8 21 24 28 28 37 40 40 41 53 53 55 58 67 72 75 80 93 106 ...

--- Test Case 9 (Very large random array, n=10000) ---
Output (first 20 elements): 3 4 8 9 12 20 21 23 24 25 26 28 31 35 37 39 40 42 45 47 ...

--- Test Case 10 (All elements same) ---
Input: 99 99 99 99 99 99 99 99 99 99
Output: 99 99 99 99 99 99 99 99 99 99
```

---

### 3. Test Case Descriptions

#### 1. Already Sorted Array

- **Purpose:** To check if Merge Sort can handle an array that is already sorted without doing extra work.

#### 2. Reverse Sorted Array

- **Purpose:** To test the case when the array is in descending order (worst case).

#### 3. Array with Duplicates

- **Purpose:** To make sure Merge Sort works properly when some numbers are repeated.

#### 4. Array with Negative Numbers

- **Purpose:** To confirm that the algorithm can also sort arrays containing negative values.

#### 5. Single Element Array

- **Purpose:** Edge case to see if Merge Sort works when the array has only one element.

#### 6. Empty Array

- **Purpose:** Edge case to ensure the algorithm does not crash when the input is empty.

#### 7. Array with Both Negatives and Positives

- **Purpose:** To check how Merge Sort deals with a mix of negative and positive numbers.

#### **8. Array with All Equal Elements**

- **Purpose:** To verify that if all numbers are the same, the algorithm still runs correctly and keeps stability.

#### **9. Array with Large Numbers**

- **Purpose:** To test the sorting of very big integer values.

#### **10. Array of Size 2 (Edge Case)**

- **Purpose:** Smallest non-trivial case to check if the algorithm can compare and sort just two numbers.
-

## 4. Plagiarism Report

Page 1

### PLAGIARISM SCAN REPORT

Report Generation Date: 31-08-25

Words: 997

Characters: 8585

Excluded URL : N/A

<b>4%</b> Plagiarism	<b>96%</b> Unique
<b>2</b> Plagiarized Sentences	<b>53</b> Unique Sentences

### Content Checked for Plagiarism

---

#### 1. Merge Sort Algorithm

Merge Sort is a sorting technique that uses the divide and conquer method.

- Divide: Break the array into two smaller halves.
- Conquer: Sort each half separately using recursion.
- Combine: Merge the two sorted halves into one sorted array.

Time Complexity For Merge Sort

- Best Case  $\rightarrow O(n \log n)$
- Average Case  $\rightarrow O(n \log n)$
- Worst Case  $\rightarrow O(n \log n)$

#### 2. C Source Code

INPUT-

```
#include  
#include  
#include
```

```
void merge(int arr[], int low, int mid, int high)  
{  
    int n1 = (mid - low) + 1;
```

```
int n2 = (high - mid);
int Left[n1];
int Right[n2];
for (int i = 0; i < n1; i++)
{
    Left[i] = arr[low + i];
}
for (int j = 0; j < n2; j++)
{
    Right[j] = arr[mid + 1 + j];
}
int left = 0, right = 0, k = low;
while (left < n1 && right < n2)
{
    if (Left[left] < Right[right])
    {
        arr[k] = Left[left];
        left++;
        k++;
    }
    else
    {
        arr[k] = Right[right];
        right++;
        k++;
    }
}
while (left < n1)
{
    arr[k++] = Left[left++];
}
while (right < n2)
{
    arr[k++] = Right[right++];
}
}

void merge_sort(int arr[], int low, int high)
{
    if (low >= high)
    {
        return;
    }
}
```

```
else
{
int mid = (high - low) / 2 + low;
merge_sort(arr, low, mid);
merge_sort(arr, mid + 1, high);
merge(arr, low, mid, high);
}
}

void printArray(int arr[], int n)
{
for (int i = 0; i < n; i++)
{
printf("%d ", arr[i]);
}
printf("\n");
}

int main()
{
// Test Case 1 - Small Array
{
int arr[] = {87, 34, 21, 56, 12};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 1 (Small array) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

// Test Case 2 - Already Sorted Array
{
int arr[] = {11, 12, 13, 14, 15, 16, 17};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 2 (Already sorted) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}
```

```
// Test Case 3 – Reverse sorted array
{
int arr[] = {9, 8, 7, 6, 5, 4, 3};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 3 (Reverse sorted) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

// Test Case 4 – Array with duplicates
{
int arr[] = {4, 2, 2, 4, 1, 1, 3, 3, 5, 5};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 4 (Duplicates) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

// Test Case 5 – Array with negative numbers
{
int arr[] = {-5, 10, -2, 7, -9, 0, 3, -1};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 5 (With negatives) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

// Test Case 6 – Odd-sized array
{
int arr[] = {11, 3, 7, 2, 9, 14, 1};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 6 (Odd-sized array, n=7) ---\n");
printf("Input: ");
```

```

printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

```

```

// Test Case 7 – Even-sized array
{
int arr[] = {20, 11, 5, 8, 14, 2, 9, 7, 3, 1};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 7 (Even-sized array, n=10) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

```

```

// Test Case 8 – Large random array (n=1000)
{
int n = 1000;
int arr[n];
for (int i = 0; i < n; i++)
{
arr[i] = rand() % 10000;
}
printf("\n--- Test Case 8 (Large random array, n=1000) ---\n");
merge_sort(arr, 0, n - 1);
printf("Output (first 20 elements): ");
for (int i = 0; i < 20; i++)
printf("%d ", arr[i]);
printf("_\n");
}

```

```

// Test Case 9 – Very large random array (n=10000)
{
int n = 10000;
int arr[n];
for (int i = 0; i < n; i++)
{
arr[i] = rand() % 100000;
}
printf("\n--- Test Case 9 (Very large random array, n=10000) ---\n");

```



```

merge_sort(arr, 0, n - 1);
printf("Output (first 20 elements): ");
for (int i = 0; i < 20; i++)
printf("%d ", arr[i]);
printf("_\n");
}

// Test Case 10 – All elements same
{
int arr[] = {99, 99, 99, 99, 99, 99, 99, 99, 99, 99};
int n = sizeof(arr) / sizeof(arr[0]);
printf("\n--- Test Case 10 (All elements same) ---\n");
printf("Input: ");
printArray(arr, n);
merge_sort(arr, 0, n - 1);
printf("Output: ");
printArray(arr, n);
}

return 0;
}

```

OUTPUT-

### 3. Test Case Descriptions

#### 1. Already Sorted Array

- Purpose: To check if Merge Sort can handle an array that is already sorted without doing extra work.

#### 2. Reverse Sorted Array

- Purpose: To test the case when the array is in descending order (worst case).

### 3. Array with Duplicates

- Purpose: To make sure Merge Sort works properly when some numbers are repeated.

### 4. Array with Negative Numbers

- Purpose: To confirm that the algorithm can also sort arrays containing negative values.

### 5. Single Element Array

- Purpose: Edge case to see if Merge Sort works when the array has only one element.

### 6. Empty Array

- Purpose: Edge case to ensure the algorithm does not crash when the input is empty.

### 7. Array with Both Negatives and Positives

- Purpose: To check how Merge Sort deals with a mix of negative and positive numbers.

### 8. Array with All Equal Elements

- Purpose: To verify that if all numbers are the same, the algorithm still runs correctly and keeps stability.

### 9. Array with Large Numbers

- Purpose: To test the sorting of very big integer values.

### 10. Array of Size 2 (Edge Case)

- Purpose: Smallest non-trivial case to check if the algorithm can compare and sort just two numbers.

## 4. Plagiarism Report

## Matched Sources :

---

[Brainlybrainly.com](https://brainly.com/question/14613605/) › question › 14613605[FREE] EX 6.1 How many iterations will the following for ...

Let's break it down: The first for loop (`int i = 0; i < 20; i++`) will iterate 20 times. It begins at 0 and increments by 1 up to but not including 20. The second for loop (`int i = 1; i`

**5%**

<https://brainly.com/question/14613605/>