



**Green University of Bangladesh**  
**Department of Computer Science and Engineering (CSE)**  
**Faculty of Sciences and Engineering**  
**Semester: (Spring, Year:2025), B.Sc. in CSE (Day)**

**Lab report #2**  
**Course Title: Algorithm lab**  
**Course Code: CSE 208                      Section:D-5**

**Student Details**

| Name |             | ID        |
|------|-------------|-----------|
| 1.   | MD.SHAJALAL | 223002088 |

**Submission Date                                      : 29-04-2025**  
**Course Teacher's Name                            : Md. Abu Rumman Refat**

[For Teachers use only: **Don't Write Anything inside this box**]

| <b><u>Report Status</u></b> |                        |
|-----------------------------|------------------------|
| <b>Marks: .....</b>         | <b>Signature:.....</b> |
| <b>Comments:.....</b>       | <b>Date:.....</b>      |

**Title Of The Experiment:** Illustrate the step-by-step execution of the Merge Sort algorithm, showing all intermediate steps of division and merging. Implement the Merge Sort algorithm in a programming language of your choice and display the sorted.

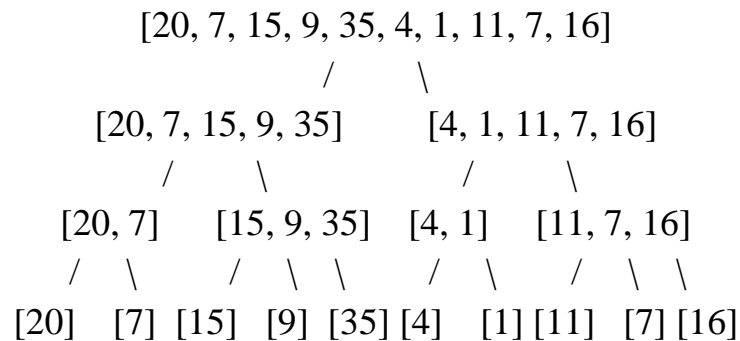
|    |   |    |   |    |   |   |    |   |    |
|----|---|----|---|----|---|---|----|---|----|
| 20 | 7 | 15 | 9 | 35 | 4 | 1 | 11 | 7 | 16 |
|----|---|----|---|----|---|---|----|---|----|

### Objectives :

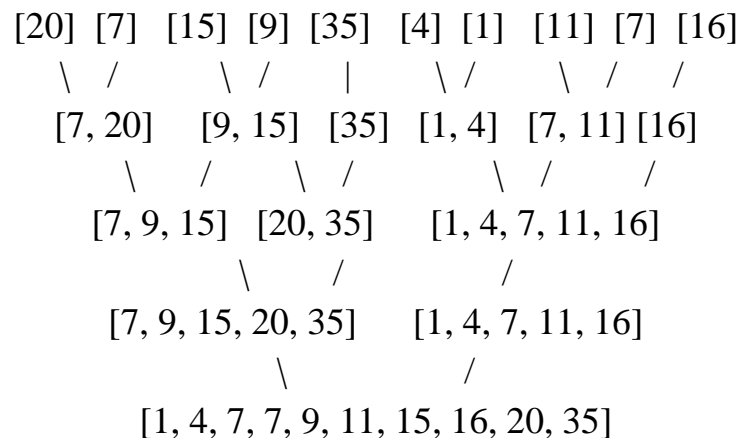
- To understand and visualize the step-by-step execution of the Merge Sort algorithm.
- To implement the Merge Sort algorithm and verify its correctness by sorting a given array.
- To analyze the Merge Algorithm by merging two sorted arrays in different cases and justify the output obtained.

### Step by step way:

Divide part:



Merge part:



### Time Complexity:

**Divide** :  $O(\log n)$

**Merge** :  $O(n)$

**Total** :  $O(n \log n)$

## **Pseudocode:**

MergeSort(arr):

    if length of arr > 1:

        mid = length of arr / 2

        left = arr[0..mid-1]

        right = arr[mid..end]

        MergeSort(left)

        MergeSort(right)

        Merge(arr, left, right)

Merge(left, right):

    create empty array merged[]

    i = 0, j = 0, k = 0

    while i < length of left and j < length of right:

        if left[i] <= right[j]:

            merged[k] = left[i]

            i = i + 1

        else:

            merged[k] = right[j]

            j = j + 1

        k = k + 1

    while i < length of left:

        merged[k] = left[i]

        i = i + 1

        k = k + 1

    while j < length of right:

        merged[k] = right[j]

        j = j + 1

        k = k + 1

    return merged.

**Code:**

```
public class Main {

    static void merge_sort(int arr[], int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            merge_sort(arr, left, mid);
            merge_sort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    static void merge(int arr[], int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int leftArr[] = new int[n1];
        int rightArr[] = new int[n2];

        for (int i = 0; i < n1; i++) {
            leftArr[i] = arr[left + i];
        }
        for (int j = 0; j < n2; j++) {
            rightArr[j] = arr[mid + 1 + j];
        }

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k] = leftArr[i];
                i++;
            } else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArr[i];
```

```

        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

public static void main(String[] args) {
    int arr[] = {2, 5, 1, 3, 7, 6, 9, 10, 8};
    merge_sort(arr, 0, arr.length - 1);

    System.out.print("Sorted Array: ");
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
}
}

```

**Input Array:**

|    |   |    |   |    |   |   |    |   |    |
|----|---|----|---|----|---|---|----|---|----|
| 20 | 7 | 15 | 9 | 35 | 4 | 1 | 11 | 7 | 16 |
|----|---|----|---|----|---|---|----|---|----|

**Output:**

```

"C:\Program Files\Java\jdk-23\bin\java.exe" "
Sorted Array: 1 2 3 5 6 7 8 9 10
Process finished with exit code 0

```

**Problem 02:** To Illustrate the step-by-step execution of the Quick Sort algorithm, showing all intermediate steps of partitioning and recursion. Implement the Quick Sort algorithm in Java (or any programming language of choice) and display the sorted array.

|    |   |    |   |    |   |   |    |   |    |
|----|---|----|---|----|---|---|----|---|----|
| 20 | 7 | 15 | 9 | 35 | 4 | 1 | 11 | 7 | 16 |
|----|---|----|---|----|---|---|----|---|----|

### Objectives :

- Understand and implement Quick Sort: Learn and code the Quick Sort algorithm.
- Visualize partitioning: Demonstrate the array partitioning process around a pivot.
- Analyze performance: Study the time complexity and efficiency of Quick Sort.

### Step by step way:

Initial array:

20 7 15 9 35 4 1 11 7 **16**

Array after partitioning with pivot 16:

7 7 15 9 4 1 **11** [16] 35 20

Array after partitioning with pivot 11:

7 7 9 4 **1** [11] 15 16 35 20

Array after partitioning with pivot 1:

[1] **7** 7 9 4 11 15 16 35 20

Array after partitioning with pivot 7:

1 [7] 7 **9** 4 11 15 16 35 20

Array after partitioning with pivot 9:

1 7 7 [9] **4** 11 15 16 35 20

Array after partitioning with pivot 4:

1 [4] 7 7 9 11 15 16 35 **20**

Sorted array pivot 20:

1 4 7 7 9 11 15 16 [20] 35

### Complexity:

#### Time complexity:

- Best/Average Case:  $O(n \log n)$
- Worst Case:  $O(n^2)$

#### Space Complexity:

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

**Pseudocode:**

QuickSort(arr, left, right):

  if left < right:

    pos = Partition(arr, left, right)

    QuickSort(arr, left, pos - 1)

    QuickSort(arr, pos + 1, right)

Partition(arr, left, right):

  pivot = arr[right]

  k = left - 1

  for i = left to right - 1:

    if arr[i] < pivot:

      k = k + 1

      swap(arr[i], arr[k])

  swap(arr[k + 1], arr[right])

  return k + 1

**Code:**

```
public class Main {
```

```
    static void quick_sort(int arr[], int left, int right) {
```

```
        if (left < right) {
```

```
            int pos = partition(arr, left, right);
```

```
            quick_sort(arr, left, pos - 1);
```

```
            quick_sort(arr, pos + 1, right);
```

```
        }
```

```
    }
```

```
    static int partition(int arr[], int left, int right) {
```

```
        int pivot = arr[right];
```

```
        int k = left - 1;
```

```
        for (int i = left; i < right; i++) {
```

```
            if (arr[i] < pivot) {
```

```
                k++;
```

```
                int temp = arr[i];
```

```
                arr[i] = arr[k];
```

```
                arr[k] = temp;
```

```
            }
```

```

    }

    arr[right] = arr[k + 1];
    arr[k + 1] = pivot;

    return k + 1;
}

public static void main(String[] args) {
    int arr[] = {2, 5, 1, 3, 7, 6, 9, 10, 8};
    int len = arr.length - 1;
    quick_sort(arr, 0, len);

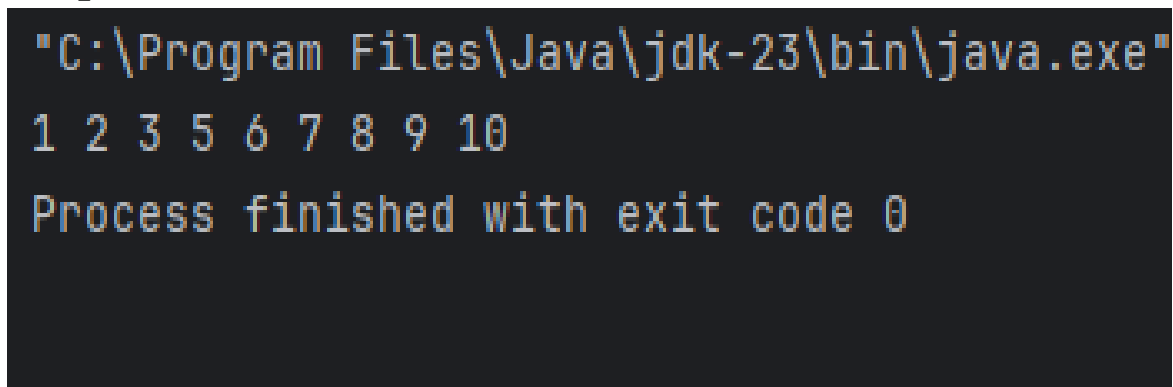
    for (int i = 0; i <= len; i++) {
        System.out.print(arr[i] + " ");
    }
}

```

### Input Array:

|    |   |    |   |    |   |   |    |   |    |
|----|---|----|---|----|---|---|----|---|----|
| 20 | 7 | 15 | 9 | 35 | 4 | 1 | 11 | 7 | 16 |
|----|---|----|---|----|---|---|----|---|----|

### Output:



```

"C:\Program Files\Java\jdk-23\bin\java.exe"
1 2 3 5 6 7 8 9 10
Process finished with exit code 0

```

### Conclusion:

#### Merge Sort:

- A stable, divide-and-conquer algorithm that divides the array into smaller sub-arrays and merges them back in sorted order.
- Time complexity is  $O(n \log n)$ , making it efficient for large datasets and guaranteeing performance.

#### Quick Sort:

- Partitions the array around a pivot and recursively sorts the sub-arrays.
- Time complexity is  $O(n \log n)$  on average but can degrade to  $O(n^2)$  if the pivot selection is poor, making it less stable than Merge Sort in the worst case.