Department of
Computer Science and Engineering

Title: Implement Merge Sort Algorithm

Algorithms Lab

CSE 206

Green University of Bangladesh

1 Objective(s)

- To attain knowledge on divide & conquer algorithm.
- To understand how Merge Sort algorithm works.
- To implement Merge Sort algorithm in Java.

2 Problem analysis

A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. This technique can be divided into the following three parts:

i Divide: This involves breaking the problem into smaller sub-problems.

ii Conquer: Solving the sub-problems.

iii Combine: Combining them to get the desired output.

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two two sub-arrays, and then these sub-arrays into even smaller sub-arrays, until multiple sub-arrays with single element in them are obtained. Now, the idea here is that an array with a single element is already sorted. Finally, merge all these sorted sub-arrays, step by step to form one single sorted array.

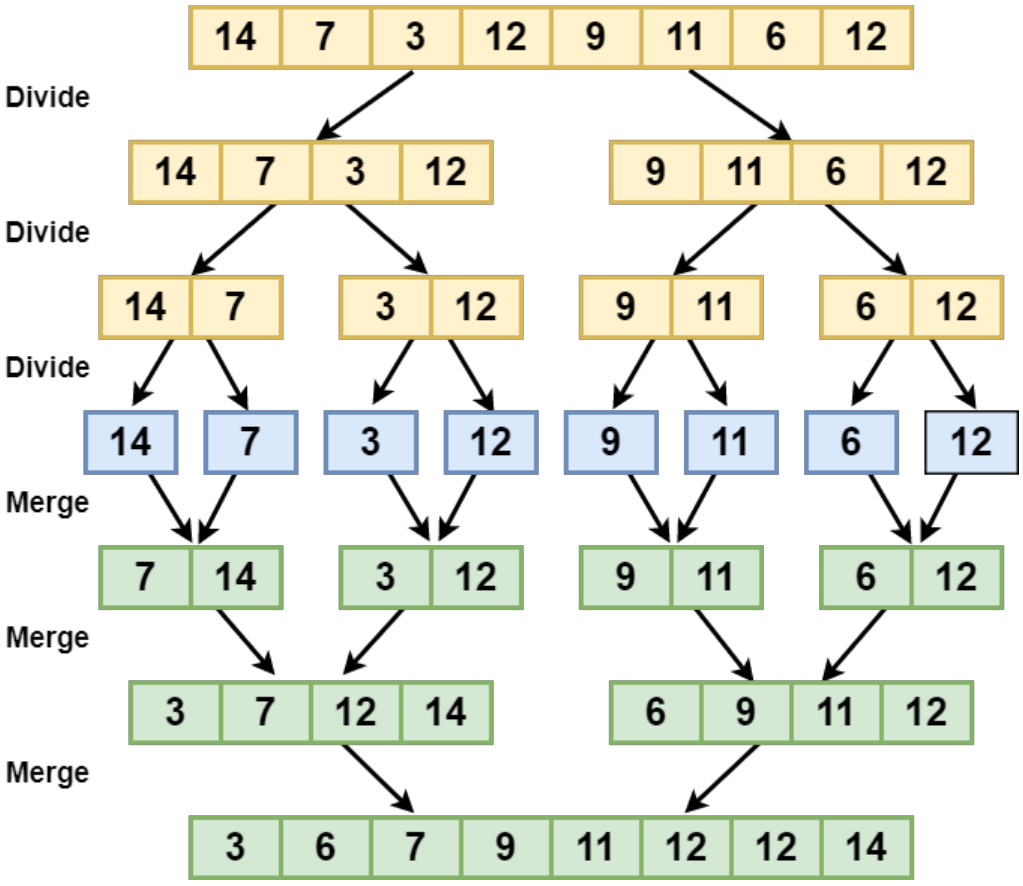

Figure 1: Merge-Sort illustrated

3 Algorithm

Algorithm 1: Merge Sort Algorithm

Input: An array A, index $left$ and $right$ defining sorting range
Output: Sorted array A within index range $left$ and $right$

if $left < right$ **then**
  $mid \leftarrow \frac{left + right}{2}$
  mergesort($A,\ left,\ mid$)
  mergesort($A,\ mid + 1,\ right$)
  merge($A,\ left,\ mid,\ right$)
end

Algorithm 2: Merge algorithm

Input: An array A, two sorted arrays denoted by index $left$ to mid and mid to $right$
Output: Two sorted arrays merged into one sorted array A

$n1 \leftarrow mid - left + 1$
$n2 \leftarrow right - mid$
Create arrays $L[1 \dots n1 + 1]$ and $R[1 \dots n2 + 1]$
for $i \leftarrow 1$ *to* $n1$ **do**
  $L[i] \leftarrow A[left + i - 1]$
end
for $j \leftarrow 1$ *to* $n2$ **do**
  $R[j] \leftarrow A[mid + j]$
end
$L[n1 + 1] \leftarrow \infty$
$R[n2 + 1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow left$ *TO* $right$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] \leftarrow L[i]$
    $i \leftarrow i + 1$
  end
  else
    $A[k] \leftarrow R[j]$
    $j \leftarrow j + 1$
  end
end

4 Implementation in Java

```java
package mergesort;

public class MergeSort {
    void merge(int arr[], int left, int mid, int right) {
        int l = mid - left + 1;
        int r = right - mid;

        int leftArray[] = new int[l];        // New array for Left elements
        int rightArray[] = new int[r];       // New array for right elements

        for (int i = 0; i < l; ++i) {        // Copying elements in leftArray
                leftArray[i] = arr[left + i];
        }

```

```
15          for (int j = 0; j < r; ++j) {        // Copying elements in leftArray
16              rightArray[j] = arr[mid + 1 + j];
17          }
18
19          int i = 0, j = 0;
20          int k = left;
21          while (i < l && j < r) {             // Copying the smaller element in
                array
22              if (leftArray[i] <= rightArray[j]) {
23                  arr[k] = leftArray[i];
24                      i++;
25              } else {
26                  arr[k] = rightArray[j];
27                  j++;
28              }
29                  k++;
30              }
31              while (i < l) {                      // Copying any remainnig left
                    element in array
32                  arr[k] = leftArray[i];
33                  i++;
34                  k++;
35              }
36
37              while (j < r) {                      // Copying any remainnig right
                    element in array
38                  arr[k] = rightArray[j];
39                  j++;
40                  k++;
41              }
42          }
43
44          void sort(int arr[], int left, int right) {
45              if (left < right) {                  // Continue recursion if this
                    condition satisfies
46              int mid = (left + right) / 2;    // Index middle elemnt of array
47              sort(arr, left, mid);            // Recursive call left
48              sort(arr, mid + 1, right);       // Recursive call right
49              merge(arr, left, mid, right);    // Mergin the left and right array
50              }
51          }
52
53          public static void main(String args[]) {
54              int arr[] = {90, 23, 101, 45, 65, 23, 67, 89, 34, 23};
55              MergeSort ob = new MergeSort();      // Creating Object of MergeSort
56              ob.sort(arr, 0, arr.length - 1);     // Calling the method
57
58              // Printing the output
59              System.out.println("Sorted array");
60              for (int i = 0; i < arr.length; i++) {
61                  System.out.println(arr[i] + "");
62              }
63          }
64  }
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Output:
Sorted array
23
23
23
34
45
65
67
89
90
101

6 Discussion & Conclusion

Based on the focused objective(s) to understand about the divide & conquer merge sort algorithm, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Draw the step-by-step solution of Merge Sort for the following array.

20	7	15	9	35	4	1	11	7	16

2. Implement and obtain the output of Merge Sort for the same array.
3. Execute the merge algorithm with the following two arrays and justify the output obtained in each case.

2	6	8	9	1	5	9	11

8	6	2	9	1	5	11	9

8 Lab Exercise (Submit as a report)

- Implement merge sort on a linked list.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.