Fall 2024 – CS 303 Algorithms and Data Structures
Lab3

Objectives:

- Implement merge sort
- Compare the performance of insertion sort and merge sort
- Improve the performance of merge sort by using insertion for sorting short arrays

Problems

1. Implement a method to sort a given array using the merge sort algorithm. You can use the algorithm provided (see Page 2) instead of the algorithm from the textbook.
2. Write a driver program to test the merge sort algorithm for the arrays of varying lengths provided in Canvas.
3. Compare the execution time of merge sort with insertion sort implemented in Lab-2. Make sure you use the same array to compare the performance. Use a table or plot to summarize the results and document your observations and analysis in the report.
4. Based on the performance results obtained in Problem-3, modify the merge sort algorithm such that when the array size gets small enough, you would invoke insertion sort instead of invoking merge sort (hint: you have to change the base condition in the recursion). Instead of hardcoding the array size make it a variable that you can pass as an argument to the merge sort method and test this cutoff value with at least four different values.
5. Test the program for the same array sizes and values. Compare the performance with the original merge sort implementation, plot the execution times, and document the analysis in your lab report.

// NOTE: You have to allocate temp array in the main method and copy the original input array *A* to
// the *temp* array before invoking merge sort in the main method.

```
MERGE-SORT (A, temp, p , r)
        if p < r
                q = |_ (p + r) / 2 _|
                MERGE-SORT (A, temp, p , q)
                MERGE-SORT (A, temp, q + 1, r)
                MERGE (A, temp, p, q, r)
```

//

```
MERGE (A, temp, p, q, r)
// merge A[p..q] with A[q+1..r]
i = p
j = q + 1

// copy A[p..r] to temp[p..r]
for k = p to r
        temp[k] = A[k]
```

```
//merge back to A[p..r]
for k = p to r
        if i > q                // left half empty, copy from the right
                A[k] = temp[j]
                j = j + 1
        else if j > r           // right half empty, copy from the left
                A[k] = temp[i]
                i = i + 1
        else if temp[j] < temp[i]           // copy from the right
                A[k] = temp[j]
                j = j + 1
        else
                A[k] = temp[i]                      // copy from the left
                i = i + 1
```

Note: The above pseudo code assumes that the array indexing is starting from 1. If you are using a programming language that uses array indexing starting from 0, you have to modify the pseudo code accordingly.

Submission:

- You are required to submit a written report (.pdf) and your project file (.zip) to the Canvas
- Homework report must follow the guidelines provided in the sample report uploaded in Canvas. Please include the screenshot of your code and outputs of your code at the end of your report.
- Do not forget to submit DIC Form
- When you create the code to read the file use relative path instead of absolute path

DATA

1000, 2500,5000,10000,25000,50000,100000,250000,500000,1000000

Grading Rubric

Coding	Implementing Algorithms	20 points
	Producing Correct Outputs	20 points
Report	Explaining the algorithms used	10 points
	Displaying the output with a graph or table	20 points
	Comparing the outputs and discussing the time complexity of algorithms	20 points
	Correct submissions of the files (DIC form, Code.zip, report.pdf)	10 points