Objectives of this assignment:

* to explore time complexity and “real time” of a well-known algorithm
* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** YOUR ANSWERS.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

What you need to do: (Use this file to INSERT your answers as indicated below)

1. Implement the ***Merge-Sort*** algorithm to sort an array. (See Appendix for the *Merge-Sort* algorithm)
2. Collect the execution time T(n) as a function of n
3. Plot the functions T(n)/, T(n)/n.log2(n), and T(n)/ as a function of n on three separate graphs.
4. In Module 4, we establish that the running time T(n) of *Merge-Sort* is Θ(n.log(n)). Discuss T(n) in light of the graph you plotted above. Use the prediction techniques learned in M1: Programming Assignment (See Early questions trying to infer the shape)

**Objective**: The objective of this programming assignment is to design and implement in Java the Merge-Sort algorithm presented in the lecture to sort a list of numbers. We are interested in exploring the relationship between the time complexity and the “real time”. For this exploration, you will collect the execution time T(n) as a function of n and plot the functions T(n)/, T(n)/n.log2(n), and T(n)/ on the same graph (*If you cannot see clearly the shape of the plots, feel free to separate plots.*). Try to predict ahead the shapes of T(n)/, T(n)/n.log2(n), and T(n)/ to check whether your plots are correct. Finally, discuss your results.

**Program to implement**

collectData()

Generate an array G of **HUGE** length L (as huge as your language allows) with **random** values capped at some max value (as supported by your chosen language).

for n = 1,000 to L (with step 1,000)

copy in Array A **n** first values from Array G // **(declare Array A only ONCE out of the loop)**

Take current time ***Start*** // We time the sorting of Array A of length n

Merge-Sort(A,0,n-1)

Take current time ***End*** // ***T(n) = End - Start***

Store the value n and the values T(n)/, T(n)/n.log2(n), and T(n)/ in a file **F** where T(n) is the execution time

**Advice:**

**1)** The pseudocode assumes arrays that start with index 1. So, an array A with n elements is an array A[1], A[2]..., A[n-1], A[n]. With most programming languages, an array A with n elements is an array A[0], A[2]..., A[n-1], A[n-1]. When implementing pseudocode that uses some array A with elements, I advise you to declare an array with elements and just ignore (not use) A[0]. This way, you can directly implement the algorithm without worrying about indices changes.

**2)** When plotting, **ignore the first values of n= 1000, 2000, 3000, and 4000**. When a program starts, there will be some overhead execution time not related to the algorithms. That overhead may skew T(n).

**Data Analysis**

Use any plotting software (e.g., Excel) to plot the values T(n)/, T(n)/n.log2(n), and T(n)/ in File F as a function of n. File F is the file produced by the program you implemented. Discuss your results based on the plots. (**Hint**: is T(n) closer to , ), or where K is a constant? See M1: Programming Assignment).

Answer where indicated below. Recall that answers must be well written, documented, justified, and presented to get full credit.

1. (25 points) Implement the ***Merge-Sort*** algorithm to sort an array. (See Appendix for the *Merge-Sort* algorithm)

a) State **here** whether your algorithm works.

b) Insert here a screenshot showing that your implementation sorts correctly an array that contains 10 numbers.

1. (10 points) Collect the execution time T(n) as a function of n. Record the values n, T(n), T(n)/, T(n)/n.log2(n), and T(n)/ in csv (comma-separated-values) file.

Turn in this csv file with your submission

1. (3x15 points) Plot the functions T(n)/, T(n)/n.log2(n), and T(n)/ as a function of n on three separate graphs (15 points per graph)*.*

Insert here the three graphs/plots

1. (20 points) In Module 4, we establish that the running time T(n) of *Merge-Sort* is Θ(n.log(n)).

Discuss here T(n) in light of the graphs you plotted above. Use the prediction techniques learned in M1: Programming Assignment (See Early questions trying to infer the shape of T(n) and determine the asymptotic growth). Discuss whether your plots confirm what we learned in Module M4.

Answer/elaborate/Justify.

**What you need to turn in:**

* Electronic copy of your source program of *collectData* program
* Electronic copy of the csv file recording the values n, T(n), T(n)/, T(n)/n.log2(n), and T(n)/.
* Electronic copy of this file (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

**Grading**

* See points distribution assigned to each task/question

Appendix: Merge-Sort Algorithm.

At this stage, you do NOT need to understand Merge-Sort (It will be presented and explained in Module 4)). Implement Merge-Sort exactly the way it is described below. Replace the infinity value (∞) with 0x0fff ffff.



