COMP 3270 – Algorithms
Programming Assignment 1 – Asymptotic Analysis of Sorting Algorithms
Due: 10/15/2023, 11:59 p.m.

1. Overview

To empirically evaluate 4 sorting algorithms and verify their theoretical upper bound. The sorting algorithms we will evaluate are: merge sort, quick sort, insertion sort, and selection sort. A starter code with helper functions and implementations of 2 algorithms (selection sort and merge sort) has been provided. Your task is to implement the remaining two sorting algorithms (quick sort and insertion sort) and perform the following empirical analyses. You will run a specific algorithm on three different types of arrays - sorted, random, or constant array of a given size n. The provided starter code will generate the input arrays for your analysis. The starter code will take as input three command line arguments in the following order:

1. Type of input array. You have three choices: ‘r’, ‘c’, and ‘s’ that correspond to random, constant and sorted arrays, respectively. If invalid values are given, it will default to a random input array.
2. Size of the array to generate and analyze. It must be a positive number greater than zero. It will default to 10 if an invalid value is provided.
3. Sorting algorithm to use. You have four choices: ‘m’, ‘i’, ‘s’, and ‘q’ that correspond to merge sort, insertion sort, selection sort, and quick sort, respectively. If an invalid value is given, it will default to quick sort.

If an incorrect number of arguments are given or if an integer is not provided for the second command line argument, it will default to running quick sort on a random input array of size 10. For stable computation of timing, the starter code will run each algorithm three times and provides the median of the three runs. For accurate timing, ensure that there are no other processes running on the machine while you conduct your analysis. The output timing will be in nanoseconds.

2. Deliverables:

There are two deliverables for this programming project: (i) the completed code with the two sorting algorithms completed in the functions marked with

"TODO", and (ii) a report with your analysis clearly marked with two sections – "Results" and "Analysis." More details about what is expected to be presented in each section are provided below.

2.1 Results

Run each of the four sorting algorithms on constant, sorted, and random arrays that are powers of 10. For each of the twelve cases, you should record the following:

- N_{min}: the smallest power of 10 array size that takes 20 milliseconds or more per run to sort.
- N_{max}: the largest power of 10 array size that takes 10 minutes or less per run to sort (30 mins for all 3 runs), or the largest input size you can successfully sort if no input took more than 30 minutes total.
- T_{min}: the time to sort an array of size N_{min}.
- T_{max}: the time to sort an array of size N_{max}.

You should report your results in a table. Your table should have 12 rows and 5 columns. Each row must have a label with 2 letters, where the first corresponds to the algorithm (Merge, Quick, Insertion, or Selection) and the second corresponds to the array type (Sorted, Random, or Constant). For example, SS corresponds to Selection sort run on a Sorted array. An example table is shown below.

	$n_{\min}$	$t_{\min}$	$n_{\max}$	$t_{\max}$
SC SS SR				
IC IS IR				
MC MS MR				
QC QS QR				

Note that the goal of this project is to find a N_{min} and N_{max} that are sufficiently far apart that the growth rate of the time complexity can be approximated using the timing and input size ratios. The descriptions of N_{min} and N_{max} given

above are just to give you a reasonable idea of how to find good array sizes; finding the exact value of N_{min} that takes 20 milliseconds to run, for instance, is not critical. You may have to check values of up to 1 billion to find N_{max}.

2.2 Analysis

In this section, you will estimate the complexity of the four algorithms by comparing the ratio between T_{min} and T_{max} to ratios representing the complexity of the algorithm. Specically, you should compute $f(N_{max}) = f(N_{min})$ for f1(n) = n, f2(n) = n ln(n), and f3(n) = n^2. You should round to the nearest integer when computing these ratios. Finally, you should label each experiment according to the ratio T_{max}=T_{min} most resembles.

For example, if you get N_{min}= 10 and N_{max} = 100, your ratios would be:

- $f_1(N_{max})/f_1(N_{min})$ = 100/10
- $f_2(N_{max})/f_2(N_{min})$ = (100*ln(100))/ (10*ln(10))
- $f_3(N_{max})/f_3(N_{min})$ = $100^2/10^2$

You should then label the algorithm based on which of these three ratios T_{max} = T_{min} is closest to. As part of your report, you should create a table that includes the computed ratios as well as the behavior of the algorithm (Linear, n lg n, or Quadratic), across all 12 experiments. An example is given below:

	$t_{\max}/t_{\min}$ n ratio	$n \ln(n)$ ratio	n^2 ratio	Behavior
SC SS SR				
IC IS IR				
MC MS MR				
QC QS QR				

Your report should contain a summary of (1) how your results compare to the theoretical analysis for <u>all four algorithms</u>, and (2) why your results make sense or are surprising. You should spend more time explaining your results when they are unusual or unexpected.