Project 1 – Experimental Analysis of Sorting Algorithms

CS 4120/5120 — 50 pts for 5120/70 for 412
Fall 2021 — Due: Wed, Oct 20, 2021 (NOT Mon, Oct 25)

For this assignment, by sorting we mean sorting in ascending order. Write a program for each of the following sorting algorithms:

a. Bubble sort (the most efficient version that minimizes iterations of the outer loop by tracking whether any swap was made in the last iteration and if so, at what position)
b. Insertion sort
c. Quick sort ([*]one of either randomized or median-of-3 pivot selection)
d. Heap sort
e. 3-way Merge sort (Must make sure your merge algorithm is linear in n)

In each case, try to make your implementation as good as possible. You may use C, C++, Java, or Python, but if you prefer to use a different language, talk to the instructor first.

IMPORTANT NOTE: This is not a hard project, but it absolutely requires careful organization and tracking of *dozens* of details (e.g., potentially, 120 numbers for comparing performance of algorithms). So, it is critical for you to start immediately and plan all details.

Input:

You must minimally first create the following types of input data, of sizes 1000, 2000, 4000, and 8000, and store them in one or more files.

a. random data
b. sorted data
c. reverse (descending) sorted data

Thus, there are at least 4 sizes in the above three categories. Create these 12 sets of input data once and use the same data on each algorithm. The elements of each input list should be in the range of 0 through 1 million. (You have at least 12 "runs" for each program, but those runs can be all in one session, or other combinations.)

Note that you will use mostly standard implementations, but must make sure they run correctly. Although I might probably assume your sorting algorithms sort correctly, if I challenge you, you are expected to demonstrate that they work correctly.

The emphasis of the project is on experimental analysis of running time ***and*** DATA comparisons. You need to track the time ***and*** the number of DATA comparisons for sorting each data set and record them in units that are fine enough (mainly the time) to allow meaningful comparison. Note that your run time tracking must be of sufficiently high resolution or else you might not see much difference in the runtimes (especially on small inputs). (In C++, you need to use *high_resolution_clock* from *chrono*, for example.)

[*] If you complete the entire project with BOTH versions of quicksort, you may get up to 5 bonus points.

Results:

Use the following criteria to compare the complexity of your programs for the above sorting algorithms:

i. Number of comparisons and ii. Execution time

Submit:

1. Submit the three 8000-element data sets.
2. Submit the source code for the sorting programs.
3. Runs showing output (not the sorted output, but the results, namely the running time and the number of comparisons) for each sorting routine for each data set of each type. (There are a minimum of 60 such runs.)
4. A table of time and #comparisons for all algorithms for all sizes for each type of data.
5. Expected/theoretical performance (e.g., theta) for each case.
6. Are the theoretical and practical performance results consistent? Any anomalies and/or surprises?
7. Analysis and Discussion of ***your results*** vis-à-vis expectations.

Items 4-7 should be written in the form of a short paper – can be just a page or two. No major bells and whistles necessary, but you need to make sure it tells a nice story.

Grading Rubric:

CS 4120:

Program code:	35 points
Track time/comparisons	8 points
Data setup	7 points
Runs/collection of results	10 points
Presentation in paper	10 points
Total	**70 points**

CS 5120:

Program code:	20 points
Track time/comparisons	8 points
Data setup	7 points
Runs/collection of results	7 points
Presentation in paper	8 points
Total	**50 points**