Asymptotic Analysis Project

COT 4400, Summer 2022

Due 06/24/2022

1 Overview

This project asks you to evaluate four of the sorting algorithms we have discussed in class on various inputs and relate their theoretical run time to their empirical (actual) behavior. These algorithms are SelectionSort, InsertionSort, MergeSort, and QuickSort. In the interests of time, you have been provided with code that implements all four algorithms and can run them on inputs of varying sizes and types.

2 Running the provided program

Along with this document, you have been provided with source code and a Makefile for the various sorting algorithms and input types to test in this project. This program will run the specified algorithm (SelectionSort, InsertionSort, MergeSort, or QuickSort) on a sorted, random, or constant input array of a given size. I expect that you can use an IDE or the Linux `make` utility in order to compile the code. Once compiled, the program should be run from the command line, as it expects three arguments. If fewer than three arguments are specified, the program will assume default values for any unspecified arguments.

The first of these three arguments represents the size of the input array. The program only accepts sizes between 1 and 1,000,000,000, inclusive (default 10,000). The second argument represents the sorting algorithm you wish to run. Valid algorithms include SelectionSort, InsertionSort, MergeSort, and QuickSort (default MergeSort), or you can abbreviate the algorithms by their first letter (‘s,’ ‘i,’ ‘m,’, or ‘q’). The last argument represents the type of input to sort. Valid input types are sorted, random, and constant (or ‘s,’ ‘r’, ‘c’; default ‘r’), where ‘random’ is an unsorted array, ‘sorted’ is a sorted array (in increasing order), and ‘constant’ is an array where every entry is identical.

In order to improve the timing stability, the algorithm runs the requested sort three times and reports the median of the three timing results to you. In order to get the most accurate timing results, there should be no other processes running on the machine at the same time, but this may not always be possible, especially if you are running the program on a lab machine.

3 Project report

Your project report should be divided into two parts, Results and Analysis. For the Results section, you will need to prepare a data file describing the performance of your algorithms, and in the Analysis portion, you will need to prepare a table describing their complexity, as well as a response to these results.

3.1 Results

Run each of the four sorting algorithms on constant, sorted, and random arrays of different sizes. For each of the twelve cases, you should record the following:

1. n_{min}: the smallest array size that takes 30 milliseconds or more per run to sort;

2. t_{min}: the time to sort an array of size n_{min};

3. n_{max}: the largest array that you sorted

4. t_{max}: the time required to sort n_{max} elements.

When trying to find n_{min} and n_{max}, I recommend starting with an arbitrary input size and multiplying or dividing the array size by something like 10 or 2, as this should quickly get you to arrays that are large or small enough. For n_{max}, I recommend you stop the program if it takes longer than 30 minutes (˜5–10 minutes per call) and try something smaller. Note that for some of the experiments the time will not get close to 30 minutes; just use the largest inputs that you were able to sort in these cases.

For $n_{\min}$ and $t_{\min}$, you do not need to find the smallest array that takes 30 milliseconds to run; anything less than 500 ms (ideally less than 100 ms) is acceptable. Also, you can go below 30 milliseconds, though I would recommend finding a result above 20 ms, as timing values that are too small may have significant errors due to rounding.

You may also use different systems to perform the tests for different algorithm/input combinations, but obviously, you'll want all of the runs for a single experiment (e.g., testing MergeSort on random data) to be on the same machine, as computing a timing ratio with results from different machines is virtually meaningless.

Lastly, increasing your stack size before testing QuickSort can help to sort larger arrays without crashing. If the recursion depth exceeds the maximum stack size, the program will crash due to "stack overflow". Different systems may handle this error differently. For example, the program may halt with no message, or it may "hang" without terminating. You can run `ulimit -s unlimited` in Linux before executing the algorithm to increase the available stack space. For `g++`, the command line options `-Wl,--stack_size,0x20000000` or `-Wl,--stack,0x20000000` *may* work to set the stack size to 1 GB. The stack size can also be changed in many IDEs on Mac and Windows; consult the appropriate documentation. Try to play with it a bit, but ultimately, just use the largest array that you were able to run successfully; an $n_{\max}$ of 100 or 200 million should be sufficient to show how the time complexity increases if you're not able to run the program on an array of size 1 billion.

You should enter your results into a comma-separated value (CSV) file. The CSV file should contain 5 columns and 13 rows. Your first column should label the 12 different experiments, while the first row labels the experiment variables ($n_{\min}$, $t_{\min}$, $n_{\max}$, and $t_{\max}$). Your row labels should include the algorithm name (SelectionSort, InsertionSort, MergeSort, or Quicksort) and input type (Sorted, Random, or Constant). You may abbreviate these labels as S, I, M, Q, and S, R, C. (For example, SS represents your SelectionSort result on a sorted array.) An example table appears below. You may use Excel (or any other software) to prepare your data.

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

3.2 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. Specifically, you should compute $f(n_{\max})/f(n_{\min})$ for $f_1(n) = n$, $f_2(n) = n\ln(n)$, and $f_3(n) = n^2$. 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 one of your experiments resulted in $n_{\min} = 100$ and $n_{\max} = 10,000,000$, your ratios would be:

$$
\begin{aligned}
f_1(n_{\max})/f_1(n_{\min}) &= 10,000,000/100 \\
&= 100,000 \\
f_2(n_{\max})/f_2(n_{\min}) &= 10,000,000\ln(10,000,000)/(100\ln(100)) \\
&= 350,000 \\
f_3(n_{\max})/f_3(n_{\min}) &= 10,000,000^2/100^2 \\
&= 10,000,000,000
\end{aligned}
$$

These three ratios represent the timing increase if the complexity of your code was exactly n, $n\lg n$, or n^2, and you should pick out which of these three ratios is most similar to your actual time increase ($t_{\max}/t_{\min}$). In reality, the growth rate of the timing function will include some lower order terms and may depend on other factors that are hard to model, like caching behavior, so do not expect the numbers to line up exactly.

As part of your report, you should create a chart 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 chart appears 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					

In addition to the table, you should write a brief summary of (1) how your results compare to the theoretical analysis for the four algorithms (below), and (2) why your results make sense or are surprising. You should write a summary *for each experiment*. You should spend more time explaining your results when they are unusual or unexpected.

	Best-case complexity	Average-case complexity	Worst-case complexity
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
InsertionSort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
MergeSort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
QuickSort	$\Omega(n \lg n)$	$\Theta(n \lg n)$	$O(n^2)$

4 Submission

For this project, you should submit a zip archive containing (1) a CSV file containing your results (described in Section 3.1), and (2) your tables and analysis (described in Section 3.2), in PDF format.

Note: This is an individual project. You are not allowed to submit work that has been pulled from the Internet, nor work that has been done by your peers. Your submitted materials will be analyzed for plagiarism. Project 1 will be evaluated out of 50 points:

5 Grading

Data file containing results	15 points
Table with ratios	15 points
Analysis	20 points

Requirements for each portion of the grade are described in Sections 3.1 and 3.2.