Josh Atchley

CS 325

6/26/17

1. Insertion sort runs faster than merge sort for all values of n where n > 1.
   1. F(n) = O(g(n)), because the exponent value of f(n) is less than the exponent value of g(n), and so will grow more slowly than g(n).
   2. F(n) = Ω (g(n)), because g(n)grows in logarithmic time, whereas f(n) grows in linear time. Therefore, g(n) grows more slowly than f(n).
   3. F(n) = O(g(n)), because logn grows more slowly than ln n. This is because smaller exponent values are required to raise 10 (the base of logn) to the value of n, than to raise e (the base of ln n).
   4. F(n) = Θ(g(n)), because only the largest factor is counted when doing asymptotic analysis. Therefore, removing constants and the least significant factors, f(n) simplifies to n2, and g(n) simplifies to n2 as well.
   5. F(n) = Θ(g(n)), because only the largest factor is counted when doing asymptotic analysis. For very large values of n, both logn and become less significant, leaving only the factor of n.
   6. F(n) = O(g(n)), because e < 3, and they both have the same exponential value of n.
   7. F(n) = O(g(n)), because for every value of n >= 1, g(n) will always be double f(n).
   8. F(n) = O(g(n)), because for every value of n >= 1, g(n) will grow exponentially larger than f(n).
   9. F(n) = O(g(n)), because factorial always grows more quickly than exponential time.
   10. F(n) = O(g(n)), because f(n) operates in logarithmic time, whereas g(n) operates in polynomial time (square root can be written as an exponent to the ½)
   11. TODO
   12. TODO
2. See code.
   1. TODO
   2. TODO
   3. TODO
   4. TODO
   5. TODO
   6. EXTRA CREDIT TODO
3. (1 pt) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in 8n2 steps, while merge sort runs in 64nlgn steps. For what values of n does insertion sort run faster than merge sort?

**Note:** lg n is log “base 2” of n or . There is a review of logarithm definitions on page 56. For most calculators you would use the change of base theorem to numerically calculate lgn.

1. (5 pts) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.
   1. f(n) = n0.25; g(n) = n0.5
   2. f(n) = n; g(n) = log2 n
   3. f(n) = log n; g(n) = ln n
   4. f(n) = 1000n2; g(n) = 0.0002n2 – 1000n
   5. f(n) = nlog n; g(n) =*n*
   6. f(n) = en; g(n) = 3n
   7. f(n) = 2n; g(n) = 2n+1
   8. f(n) = 2n; g(n) =
   9. f(n) = 2n; g(n) = n!
   10. f(n) = lgn; g(n) =
2. (4 pts) Let f1 and f2 be asymptotically positive non-decreasing functions. Prove or disprove each of the following conjectures. To disprove give a counter example.
3. If *f1*(*n*) *=* O(*g*(*n*)) and *f2*(*n*) *=* O(*g(n*))then *f1*(*n*)*=* Θ(*f2(n*) ).
4. If *f1*(*n*) *=* O(*g1*(*n*)) and *f2*(*n*) *=* O(*g2*(*n*))then *f1*(*n*)+ *f2*(*n*)*=* O(max{*g1*(*n*)*, g2*(*n*)} )
5. (5 pts) **Merge Sort and Insertion Sort Programs**

Implement merge sort and insertion sort to sort an array/vector of integers. These are very common algorithms and you may modify existing code if you reference it. You may implement the algorithms in the language of your choice, name one program “mergesort” and the other “insertsort”. Your programs should be able to read inputs from a file called “data.txt” where the first value of each line is the number of integers that need to be sorted, followed by the integers.

Example values for data.txt:

4 19 2 5 11

8 1 2 3 4 5 6 1 2

The output will be written to files called “merge.out” and “insert.out”.

For the above example the output would be:

2 5 11 19

1 1 2 2 3 4 5 6

***Submit a copy of all your code files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named data.txt.***

1. (10 pts) **Merge Sort vs Insertion Sort Running time analysis**

The goal of this problem is to compare the experimental running times of the two sorting algorithms.

a) Now that you have proven that your code runs correctly using the data.txt input file, you can modify the code to collect running time data. Instead of reading arrays from a file to sort, you will now generate arrays of size n containing random integer values from 0 to 10,000 and then time how long it takes to sort the arrays. We will not be executing the code that generates the running time data so it does not have to be submitted to TEACH or even execute on flip. Include a “text” copy of the modified code in the written HW submitted in Canvas.

b) Use the system clock to record the running times of each algorithm for n = 1000, 2000, 5000, 10,000, …. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data. If you program in C your algorithm will run faster than if you use python. You will need at least seven values of t (time) greater than 0. If there is variability in the times between runs of the same algorithm you may want to take the average time of several runs for each value of n.

c) For each algorithm plot the running time data you collected on an individual graph with n on the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software. Also plot the data from both algorithms together on a combined graph. Which graphs represent the data best?

d) What type of curve best fits each data set? Again you can use Excel, Matlab, any software or a graphing calculator to calculate a regression equation. Give the equation of the curve that best “fits” the data and draw that curve on the graphs of created in part c).

e) How do your experimental running times compare to the theoretical running times of the algorithms? Remember, the experimental running times were “average case” since the input arrays contained random integers.

**EXTRA CREDIT:** *It was the best of times, it was the worst of times…*

Generate best case and worst case input for both algorithms and repeat the analysis in parts b) to d) above. Discuss your results.