Computer Science 20

Sorting Research Project This is the first of two projects in this course. A project counts the same as a test. Your work is due **Monday, February 28th** by 5 o'clock.

Your mission is to implement **one of these algorithms**: MergeSort, HeapSort, or Quicksort. Then run experiments to answer **two of the questions** below. You have to answer the first one, and you can pick which one of the other questions to answer. Write a 2-5 page paper describing your experiments and your results.

1. *How does the number of array-element comparisons grow as a function of n?* The program should report how many comparisons c of array elements it performs each time it is run. Insert a counter variable in your program, and print out both n and c. Then look at the functional relationship $c = f(n)$.

2. *How well does number of comparisons predict CPU time?* There are a few ways to tackle this question: Suppose theory tells you that Insertion sort uses 100 times more comparisons than your algorithm at particular problem size n. Is it program really 100 times slower? (I have written a Java Insertion Sort program that you can use.) Does increasing by a factor of 100 in comparisons translate to an increase by a factor of 100 in running time? If you plot comparisons vs CPU time, do you get a straight line?

3. *How much difference is there between best case, average case, and worst case performance?* (Heapsort, Quicksort only). Generate inputs of each category, and measure the comparison cost of the algorithm as a function of n. How do these different costs compare?

4. *How important are secondary costs?* The second most common operation in most sorting algorithms is *data movement*, such as swapping two array elements, or assigning an array element to a variable (or vice versa). Run an experiment to count comparisons alone, and then comparisons plus data movement operations. What proportion of the total cost is taken up by comparisons vs data movements.

Here are some details:

- You can download a copy of my Insertion Sort program from

  `www.cs.amherst.edu/ccm/cs20/ISort.java`

- Unless otherwise specified, look at average-case performance. Use `Math.random()` to fill an array with random numbers before sorting. For an average-case analysis, you will need to look at performance over t random trials at each problem size n. Write your program to take n and t as input. For each t, it generates a random array of size n, sorts it, and prints out the performance data.

- Problem 1 asks you to look at the growth rate of comparisons vs problem size. Whichever algorithm you pick, it is $\Theta(n \log n)$ average case. The question is how fast c approaches its asymptotic bound. If you plot n vs $c/\log n$, it should appear to approach a constant c_1 which is the coefficient of the leading term. Does it approach from above or below? How close is it to its asymptote (percentage-wise)?

  Your n values should span at least 4 orders of magnitude. (That is, the max value should be at least 1000x the min value of n). Since the inputs are random, you will need to run several random trials (at least 10) for each input size.

- I expect that most of you will write your program in Java on a Unix system. Next week I will show you the Unix **time** command (for timing programs), and the **R** data analysis program (for creating graphs and charts). If you are not an Amherst Student you need to ask IT about getting an account to use these systems. If you want to write in C, fine. You don't have to use these tools if you have a preferred alternative.

- The CPU timer is not reliable for programs taking less than about a 50th of a second: set n and t so that the programs need at least 0.1 seconds to run, and then take the average of time/trials.

Your Report Your report should contain

- A mention of the algorithm you implemented and how exactly you counted operations. What problem sizes you looked at, and how many trials per problem size. You have to give enough details about your experiment that a reader could replicate it and expect to get the same results.

- Your results – a page or so to answer each question that you looked at. This should be in the form of English sentences, with graphs or tables to illustrate and support your conclusions. You can't just attach a table of numbers and call it a result.

- A description of any problems or questions that might restrict the generality of your results. How consistent and reliable is the cpu timer, for example? How much might another person's implementation of the same sorting algorithm vary from yours?