CPS212 - COMPUTATIONAL STRUCTURES AND ALGORITHMS

Lab #12 - Due: Wednesday, May 6th, at the start of class

Purpose: To compare actual measurements of sorting algorithm performance with theoretical computing times (Big O times) and theoretical space requirements.

Introduction
In this lab, you will be looking at the performance of a number of sorting algorithms, using code furnished by the professor and making the connection between theory and reality.

Use this website (http://webspace.ship.edu/cawell/Sorting/) to understand more clearly the time complexity of certain sort algorithm when dealing different types of arranged data. (This website has a cool animation – which shows the sorting of different arrangements of data using different sorting algorithms.)

Some Notes On Measuring Operation Times and Space Requirements
The larger the dataset the better when gathering sort times.

To measure space utilization, the sorting algorithms have been instrumented with a function call that measures the current extra size of the stack. The "high water mark" of this will be reported. This will give an approximately correct measurement of extra space utilized.

Preparation for Lab
Take some time to recall the time and space complexities of the various sorting algorithms discussed in class and outlined in our textbook. Write down your best understanding of these complexities on the first page of the writeup form.

Detailed Directions

- Today we'll be using only linux – so ssh onto Moses. Create a subdirectory for this lab.

- Make lab12 your working directory, and copy the file `/gc/cs212/lab12_2008/lab12` to your `lab12` directory. This program contains a test driver, plus code for six sorting algorithms. (These sorting algorithms will be identified as "Mystery Algorithm 1" through "Mystery Algorithm 6", and you will need to figure out which is which by experimentation.)
  • Insertion Sort
  • Basic Bubble Sort
  • Quick Sort
  • Simple Selection Sort
  • Heap Sort
  • Merge Sort (internal)

- In the space provided on your writeup form, record the theoretical asymptotic complexity (Big O approximation) for each of the sorting algorithms for the following measurements:
  1. Time complexity for random data - O(n), O(n log n) or $O(n^2)$
  2. Time complexity when sorting data that is already sorted - O(n), O(n log n) or $O(n^2)$
  3. Space complexity for random data - O(1), O(log n) or O(n).
  4. Space complexity when sorting data that is already sorted - O(1), O(log n) or O(n).

  In order to assist in recording these theoretical complexities, see **http://www.math-cs.gordon.edu/courses/cs212/software/sort_algo.pdf** for the actual code used for this lab.

- Run the program. Note that, when it starts up, it first generates random data, and then asks you which sorting algorithm you want to test (number 1... number 6). After you choose an algorithm, it asks you how many items you want to sort. After sorting the items and reporting the results, it

asks if you want to sort the (now sorted) data again. If you say yes, it will perform the sort again and report the results. (Of course, the first time you run a given algorithm, it sorts random data; if you run it again, it starts with already sorted data.)

Once you have done one or two sorts with a given algorithm, you may specify a different number of items to sort using this algorithm (again, starting with random data).

- You must do the following for each algorithm, recording your observations and calculations in the space provided on your writeup form:

  • Test the algorithm for at least five different numbers of items, and report the corresponding times on your writeup form. Because differing sort algorithms take widely-varying amounts of time, the numbers of items you choose to sort may be different for different algorithms, and you may perform a test for one or more numbers of items that you end up not reporting (or even aborting the test if it takes too long.)

  For each sort, record on your writeup form the actual numbers of items sorted for the five cases you choose to report, together with the observed times. Record the values in ascending order of number of items, regardless of the order in which you actually did the tests.

  The numbers you choose to report must conform to the following standards. (This may require some playing around on your part to get appropriate values. Therefore, you may need to run more than five tests before getting values to report that meet the criteria.):

  • The smallest number of items you sort must take *at least* 0.1 second. This is 10 times the clock granularity, and represents a possible error due to clock granularity of no more than 10%. (This degree of potential error is more than desirable, but necessary since the times taken by various algorithms vary so widely, in order to be able to compare algorithms with each other for a common number of items.)

  • The sizes you choose should be powers of two to simplify the algorithm analysis (since log2 of such a size will be an integer). For your information, the following are the relevant powers of two.

n	log2 n
1024	10
2048	11
4096	12
8192	13
16,384	14
32,768	15
65,536	16
131,072	17
262,144	18
524,288	19
1,048,576	20

  • There must be at least one size that is common to all algorithms - i.e. all algorithms are tested for this size. (Finding a size that satisfies this requirement without requiring you to stare at a blinking cursor forever may take a bit of doing, but there is one that works!) Note that you may have to test some algorithm for a size bigger than you would otherwise be required to test, though still not requiring time in excess of 6 minutes.

• Note that, for each sort, you are given the option of repeating the sort on the (already sorted) results of the previous sort. Repeat the sort using the already sorted data for *all* the sizes you report,.

> • *Exception:* you may omit cases where the time on sorted data would exceed 5 minutes - but you must test at least two different cases for any given sort to verify the O(n^2) behavior. (This will happen if you observed, for smaller cases, that the algorithm takes much *longer* on already sorted data than on random data.) Indicate this by "omitted - too large" on your writeup form.
>
> • In other cases - where the algorithm does much better on already sorted data than on random data - you may get a time that is way too small to be accurate - possibly even 0.0. Go ahead and report this - except that if the test reports 0.0 record the time as "< 0.01" - it's certainly not 0 but it's too small to report.

• Also report, for each sort and each number of items that you report on, the amount of extra space required by the sort. Note that you are to do this for both random and already ordered data, provided you do both tests.

• Determine, based on your observations, whether each of the time complexities appears to be O(n), O(n log n) or $O(n^2)$, and whether each of the extra space utilizations appear to be O(1), O(log n) or O(n). Record your conclusions in the appropriate spaces on the writeup form.

When you have completed this work for all six algorithms, you should be able to determine which algorithm is which by comparing observed and predicted behaviors. (*Shh - don't tell the other teams!)* To see how well predicted and actual results compare with each other, perform the following analysis for the random-data case time data for each sort.

> • Observe that the claim that some algorithm has time complexity O(f(n)) implies that the execution time can be approximated by an expression of the form C x f(n) for an appropriate value of C. We can therefore use one value of n to calculate a value for "C", and then use the expression to estimate the time for other values of n.
>
> Example: Suppose that, for a O(log n) sort, we were to observe that for n = 4096 the algorithm took exactly one second. This leads us to approximate "C" as follows:
>
> $C \times (4096 \log_2 4096) \cong 1 \text{ second}$
> $C \times 4096 \times 12 \cong 1 \text{ second}$
> $C \cong 2 \times 10^{-5}$
>
> We now estimate the time for 16,384 items as follows
>
> $\text{Time} \cong C \times (16384 \log_2 16384)$
> $\text{Time} \cong 2 \times 10^{-5} \times 16384 \times 14$
> $\text{Time} \cong 4.6 \text{ seconds}$
>
> (Note the use of the symbol ≅ because we are approximating - also note that two significant figures is plenty! Use of more significant figures suggests more reliability than is there.)
>
> • For each algorithm, use the data from the *middle* size to estimate a value for "C", and then use this, in turn, to estimate the time for the other four sizes. For each estimate, also calculate the percent discrepancy between the estimated time and the actual time. (E.g., for the above example where we estimated a time of 4.6 seconds for n = 16384, if we actually observed 4.8 seconds the discrepancy would be 0.2/4.8 ≅ 4%. Record your estimates and the discrepancy between the estimate and the observed data on your writeup form. If any estimates seem to be *wildly* different from the observed data, get help from the professor before proceeding.

Get the professor to verify the correctness of your identifications and of the computations that justify your conclusions. (Note: credit for this part of the lab requires being able to support your conclusions by sound evidence! Guessing the correct identifications with insufficient data is not acceptable.)

Each member of the team must now individually do the following, and attach it to the writeup form. Answer this question: Under what circumstances would you use each of the algorithms? Be sure to take all of the following into consideration

- the number of items to be sorted
- whether there is significant order already present in the data
- programming effort needed to ensure a correct implementation of the structure

Try to use as much "quantitative proof" as possible.