University of Illinois at Chicago
CS 202: Data Structures and Discrete Mathematics II — Handout 8
Professor Robert H. Sloan — October 28, 2002

Programming project: Lab report timing sorting

1 The Assignment

For this assignment, I would like you to produce a study like the one given in Figure 7.22 of our text, page 295, that compares the time taken by Insertion, Shell, Heap, and Quick sorts on various size inputs.

You may use the coding of the sorting algorithms provided by Weiss in Sort.h. As usual, I will post on the course web page a slightly cleaned up version of that file that uses the STL vector, etc.

I ask you to make only a few changes/additions to simply using Weiss's code:

1. Modify Shellsort to use Sedgewick's increments. You may assume that the number of items to be sorted will never be larger than 1,000,000, so you can just figure out the increments and hard-code them into array or list.

2. Use Weiss's code for "Quicksort." For "Quicksort (optimized)" use the sort routine from `<algorithm>`. Believe me, it has been optimized.

To generate numbers to sort, please use the standard random number generator discussed for our random sentence assignment.

2 Report idea

This assignment is mainly about experimenting, using the scientific method, and presenting your results well, and not so much about coding.

I want a both a nice table and a nice written discussion describing how well the algorithms work in practice on different size inputs.

Your largest size input should be 1,000,000 items. You should determine what your smallest input size is, though it definitely should not be any smaller than 100. It needs to be large enough that you can see the difference between Insertion sort and other, faster sorts. As we discuss next, you don't have the ability to time your code so precisely.

3 Timing code

The function `clock()` in library `<ctime>` returns a long int that gives a time. I believe that the units are microseconds, but am not positive (nor am I positive that the units are the same across operating systems). However, `ctime` defines a constant `CLOCKS_PER_SEC` such that `clock()/CLOCKS_PER_SEC` is a time in seconds.

You should write your program so that it calls clock immediately before and after calling a sort algorithm. Then it will calculate the difference and print out the elapsed time.

Thus your code will look like:

```
long before f= clock();            // get current time (in microseconds?)
sort here
long after = clock();

cout << "Elapsed time is " <<
     (double) (before - after) / (double) CLOCKS_PER_SEC << endl;
```

Note that I believe the accuracy of clock is only 1/60 of a second. Furthermore, it doesn't effectively eliminate all the other things that may be going on on the system, so differences of under 0.1 sec or so may not be meaningful.

However, the time used will get well above 0.1 sec for large enough values of n.

4 Deliverables

Please submit in class on by Wednesday, November 13,a medium-short report that shows timings and discusses the results. Besides a table, please also include a graph.

In the small direction, do not report any times under 0.1 sec; just list them as something like "very fast." In the large direction, you do not have to do any one algorithm run that consumes more than 120 sec of time, so I imagine, for instance, that you will not report an insertion sort time for 1,000,000 items.

The tables and graphs may be hand-drawn, but your written description *must be typewritten/word processed.*

Turn in your code, on paper. We just want to make sure you did the assignment; we will not be grading the code very closely at all.