CSCI-1200 Computer Science II — Fall 2008
Homework 10 — Performance and Order Notation

In this final assignment for CSII, you will carry out a series of tests on the fundamental data structures in the Standard Template Library to evaluate the relative performance of these data structures and solidify your understanding of order notation. *Be sure to read the entire handout before beginning your implementation.*

The Program

You will measure the runtime and the number of pairwise comparisons needed for a few common, moderately compute-intensive operations: sorting, removing duplicates (without changing the overall order), and finding the mode. You will perform these tests on STL `string` objects that can be read from a file or constructed from random sequences of `char`s. The data structure, operation, size of the test (number of strings), source of the input, and output file are specified on the command line. Here are two sample input lines:

```
a.exe vector sort 10000 random 5 out.txt
a.exe vector mode 1000 in.txt out2.txt
```

The first example will generate 10,000 random strings of length 5, use a vector to sort them, and then output the result to the file named "out.txt". The second example will read the first 1,000 strings from the file named "in.txt", use a vector to find the most frequently occurring value (implemented by first sorting the data), and then outputs that string (the mode) to a file named "out2.txt". We provide a starting base of code that performs these operations using the STL `vector` data structure.

You will extend this program to allow other data structures in place of `vector`, including: STL `list`, STL `set` or `map`, STL `priority_queue`, and the `cs2hashset` implementation of hash tables. For extra credit you may test other data structures. You should carefully consider the most efficient way (minimize the running time) to use the data structure to complete the operation. Similarly, for extra credit you may test other common operations. In the provided code base, two fixed length arrays of string objects are used to load and output the data. The data structure specified on the command line is the only additional data structure that is "allowed" in the implementation of the operation. Thus, some combinations of data structure and operation may not be feasible. In these cases include a short writeup in your report explaining why the pairing of data structure and operation is impractical.

Measuring the Performance and # of Comparisons

The provided code demonstrates how the `clock()` function can be used to measure the processing time of the computation. The resolution accuracy of the timing mechanism is system and hardware dependent and may be in seconds, milliseconds, or something else. If the resolution on your system is coarse, you must base your analysis on the measurements from sufficiently large datasets. The program reports the time to load, process, and save the data. The provided code also demonstrates how the number of pairwise comparisons used by the operation can be counted by implementing wrapper functions for the $<$, $>$, and $==$ string comparison functions. Functors are also provided for use with `set`, `map`, `priority_queue`, and `cs2hashset`. Both measurements should inform your analysis of the data structures.

The Report

You will submit the source code for the program described above, but the bulk of the points for this assignment will be awarded for **a complete and well-written report that presents and analyzes the results of your testing**. The basic outline of the report should include: an *introduction* with your initial hypotheses; the *procedure* which includes any nontrivial implementation details; the *data* from your tests with some intermediate analysis; and a *conclusion* that summarizes your findings about the data structures, in what instances the different data structures are most useful, and any surprises you found in the results. Any unanswered questions in your analysis that require further testing, should be described as *future work*.

The Data

Your report should include well-formatted tables of the raw timing data and raw pairwise comparison counts for each category of testing. The data should be well labeled with the operation, data structure, source of input, size of input, etc. You should also include the specs of the hardware on which you performed the tests and any other variables that may be relevant. Here is an example of how an individual data table might be organized:

Sorting randomly ordered, random 5 letter `strings` using STL `vector`

# of strings	load time (sec)	operation time (sec)	output time (sec)	# of comparisons
10000	0.023	0.031	0.089	160,150
20000	0.043	0.067	0.172	339,503
50000	0.115	0.180	0.445	935,492
100000	0.226	0.402	0.918	1,993,096

The Analysis

Following each table you should summarize the conclusions of your testing and derive both the order notation and the approximate constant coefficient on the dominant term for the data. For example, in the table above, the time to both load and output is linear in the # of strings, that is O(n), with approximate coefficients $k_{load} = 2.2 \text{ x } 10^{-6}$ and $k_{output} = 9.0 \text{ x } 10^{-6}$. The number of comparisons and the operation time are both O($n \log n$) with coefficients $k_{num\ comparisons} = 4.0$ and $k_{operation} = 8.0 \text{ x } 10^{-7}$ sec.

Important Note: Since your tests will often include randomness and the true formula for the running time will likely involve additional terms with smaller exponents, your formulas will *not* exactly match the data. Your goal is to determine a reasonable order notation and coefficient that explains the overall trend in the data and provides an upper bound on the computation. Refer to the STL documentation to confirm your findings and discuss any anomalies present in the results.

You should test a variety of input sizes (number of strings), string lengths, and input conditions (randomly ordered data, sorted data, reverse sorted data, data with no duplicates, data with many duplicates, etc.). However, be a good scientist and only vary one parameter at a time and hold all other variables constant. *Important Note:* This is a one week assignment and we do not expect you to thoroughly test all variables on all possible datasets and data structures and operations. Manage your time carefully and select the tests you find most relevant and intriguing. It is more important that you fully analyze the data you collected, rather than simply generate an enormous quantity of data. You may also include a few *select and carefully prepared* graphs or charts summarizing the data and analysis. Your report should be complete and *concise*. Points will be awarded for clarity of presentation. A long document is not necessarily a more thorough study and will not necessarily result in a higher grade.

The report document may be submitted in either plaintext (.txt) or in PDF. No other file formats are allowed. PDF documents may be created with OpenOffice, CutePDF, LaTeX, or a number of other methods. Be sure to proofread and spellcheck your document.

Submission

Do all of your work in a new folder named `hw10` inside of your CSII homeworks directory. Please use the provided template `README.txt` file for notes specific to the program compilation and execution that you want the grader to read. **You must do this assignment on your own, as described in the "Academic Integrity for Homework" handout. If you did discuss the problem or error messages, etc. with anyone, please list their names in your `README.txt` file.** When you are finished please zip up your folder exactly as instructed for the previous assignments and submit it through the course webpage. *Important Note:* Do not include any large test datasets with your submission, because this may easily exceed the submission size. Instead describe any datasets you created, citing the original source of the data as appropriate.