CSCI-1200 Data Structures — Spring 2015
Lab 13 — Order Notation & Performance

In this lab we will carry out a series of tests on the five fundamental data structures in the Standard Template Library we have studied this semester: `vector`, `list`, binary search tree (`set` / `map`), `priority_queue`, and hash table (`unordered_set` / `unordered_map`). We will hypothesize and evaluate the relative performance of these data structures and solidify our understanding of order notation.

For each data structure you will predict and measure the runtime for three different moderately compute-intensive operations: *sorting*, *removing duplicates* (without changing the overall order), and *finding the mode* (most frequently occurring element). You will perform these tests on STL `string` objects that can be read from a file or constructed from random sequences of `char`s.

Checkpoint 1

Before doing any implementation or testing, create a simple table with 5 data structures on one axis and 3 operations on the other axis. In each cell write down the order notation for completing the operation using that datatype. If it is not feasible/sensible to use a particular data structure to complete the specified operation put an X in the box. If two data structures have the same order notation for one operation, predict which one will have a faster running time for large data. We combine set & map (and unordered_set & unordered_map) in this analysis, but be sure to specify which datatype of the two makes the most sense for each operation.

To complete this checkpoint: Present your analysis to one of the TAs and be prepared to explain your answers and an outline of the implementation as necessary.

Checkpoint 2

Next, let's dive into the provided code. We have implemented the 3 operations for the `vector` datatype.

http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/13_order_notation/performance.cpp
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/13_order_notation/test_input.txt
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/13_order_notation/test_sort.txt
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/13_order_notation/test_mode.txt
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/13_order_notation/test_remove_dups.txt

The data structure, operation, source of the input, size & length of input (for randomly generated input), and output file are specified on the command line. Here are a few sample command lines:

```
a.exe vector mode in.txt out.txt
a.exe vector sort random 10000 5 out.txt
a.exe vector remove_dups random 10000 2 out.txt
```

The first example reads the file named `in.txt`, uses 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 `out.txt`. The second 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`. Similarly, the command line option `remove_dups` will remove the duplicate elements from the input (without otherwise changing the order).

Compile and test the provided code on a variety of tests of different sizes for the `vector` datatype for each of the 3 operations. The provided code uses the `clock()` function 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. Record the results in a table like this:

Sorting random 5 letter `strings` using STL `vector`

# of strings	load time (sec)	operation time (sec)	output time (sec)
10000	0.023	0.031	0.089
20000	0.043	0.067	0.172
50000	0.115	0.180	0.445
100000	0.226	0.402	0.918

As the dataset grows, does your predicted order notation match the raw performance numbers? For example, in the table above, the time to both load and output is linear, in the # of strings, that is O(n).

$$\begin{aligned} load\ time(n) &= k_{load} * n \\ output\ time(n) &= k_{output} * n \end{aligned}$$

We can estimate the coefficients from the collected numbers: $k_{load} = 2.2 \text{ x } 10^{-6}$ and $k_{output} = 9.0 \text{ x } 10^{-6}$. The running time for sorting with the STL `vector` sorting algorithm is O($n \log n$):

$$operation\ time(n) = k_{operation} * n \log n$$

with coefficient $k_{operation} = 8.0 \text{ x } 10^{-7}$. Of course these constants will be different on different operating systems, different compilers, and different hardware!

To complete this checkpoint: Show the results of your testing and analysis to one of the TAs. The data should be neat & well organized in order to receive this checkpoint.

Checkpoint 3

For this last checkpoint, form a team of 3 with your labmates. *You will not receive credit for this checkpoint for working alone!* (If the lab section does not have a multiple of 3 students, the grad TA will designate one team with 2 or 4 students. Everyone else must be in a team of 3!)

Working together with your teammates tackle the implementation, testing, and analysis of at least 2 of the 3 operations for at least 2 of the 4 other data structures. We recommend "*pair* or *peer programming*", defined by Wikipedia, http://en.wikipedia.org/wiki/Pair_programming:

> Pair programming (sometimes referred to as peer programming) is an agile software development technique in which two programmers work as a pair together on one workstation. One, the driver, writes code while the other, the observer, pointer, or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently.

In the provided code base, two fixed length arrays of string objects are used to load and output the data. Thus, the load & output times should be similar for all datatypes. The data structure specified on the command line is the only additional data structure that is "allowed" in the implementation of the operation. You should carefully consider the most efficient way (minimize the running time) to use the data structure to complete the operation. *Make sure you debug the output of your implementation by comparing it to the output from the vector datatype!*

To complete this checkpoint: As a team present your results to one of the TAs. Identify the contributions or role each of you played on the team. *It must be a group effort!* What was most surprising about your results? What was the most challenging part of this lab exercise (may be technical or teamwork-related)?