

CSCI-1200 Data Structures — Spring 2018

Lab 14 — Order Notation & Performance

Note: You have been assigned 2 data structures on Submitty (the same way you got your HW4 materials), look for the Lab 10 Data Structures box at the top of the page. In preparing for this lab, you only need to work with one or both of your Data Structures.

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 `chars`.

Checkpoint 0

If you haven't done so already, please complete your Digital Measures course evaluation for Data Structures (your honest & anonymous feedback is very important!). Have the webpage receipt saying "completed" open in your browser to receive credit for Checkpoint 1.

Checkpoint 1

This checkpoint will be given out in lab.

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/spring18/csci1200/labs/14\_order\_notation/performance.cpp  
http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/14\_order\_notation/test\_input.txt  
http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/14\_order\_notation/test\_sort.txt  
http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/14\_order\_notation/test\_mode.txt  
http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/14\_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 $O(n)$.

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

We can estimate the coefficients from the collected numbers: $k_{\text{load}} \approx 2.2 \times 10^{-6}$ and $k_{\text{output}} \approx 9.0 \times 10^{-6}$. The running time for sorting with the STL vector sorting algorithm is $O(n \log_2 n)$:

$$\text{operation time}(n) = k_{\text{operation}} * n \log_2 n$$

with coefficient $k_{\text{operation}} \approx 2.3 \times 10^{-7}$. Of course these constants will be different on different operating systems, different compilers, and different hardware! These constants will allow us to compare data structures / algorithms with the same order notation.

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

Tackle the implementation, testing, and analysis of all 3 of the 3 operations for at least 1 of your assigned data structures.

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!* Compare your experimental results to your answers for Checkpoint 1. If you have time, implement and collect numbers for additional data structures and/or operations.

Near the end of the lab period, present your results to one of the TAs. What was most surprising about your results? What was the most challenging part of this lab exercise?