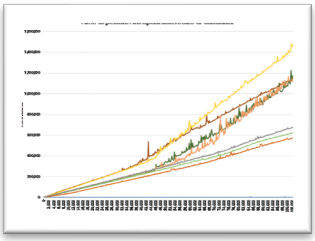
CPSC 131, Data Structures - Spring 2020
Container Review and Analysis Homework

Learning Goals:

- Review of the Sequential, Associated, and Unordered data structures
- Review and practice using the STL's Vector, List, Forward List, Map, and Unordered Map container interfaces
- Reinforce the similarities and differences between containers
- Review and practice efficiency class identification for the fundamental operations of all the data structures covered
    - O(1), $O(\log_2 n)$, O(n), and $O(n^2)$
- Reinforce modern C++ object-oriented programming techniques

Description:

In this homework assignment you are to perform 3 tasks:



1. Complete the starter code given, then compile and execute your program saving the output to a text file. The output will be a table containing the amount of actual elasped time consumed while performing an operation (columns) on a container filled with N elements (rows). This is not the main objective of this assignment, but rather just a required step.
2. Graph the table produced by the program. This too is just a required step towards the main objective.
3. Analyze and summarize the performance of the same operation across all data structures. *This* is the most important objective of the assignment. Explain the information on the graph in terms of the operation's efficiency class. Compare and contrast the operations performed on the different data structures. Be sure to include Big Oh in your explanations. For each operation, provide a concrete, real-world situation that operation is used and then make a selection of the container best suited. Explain why you selected the one you did, and why you did not select the others. Your response should be at least 1 full page (½-inch margins, 10-point Times New Roman font, single spaced, well-formed grammatically correct sentences, etc.). You may include pictures, tables, and graphs, but those are in addition to (doesn't count towards) your 1 full page target. Include your graph in your analysis report.

You are given main.cpp with sections that need to be completed. This program is unusual in that it does not solve a particular problem like previous homework assignments, but enables you to demonstrate you know how to use the STL containers, their interfces, and related iterators. Most of the sections require only a single line of code. For example, insert a grocery item into a container at verious positions (front, back, etc). Other sections require a few more lines of code (about 5 or 6). For example, search for a grocery item with a particular UPC. You've done all of this before, so it should be review. Okay, hash tables (std::unordered_map) may be new but they have an uncanning similarity to binary search trees (std::map).

For example, given the following code fragement that measures the amount of elasped time consumed inserting an element at the back of an unbounded vector data structure:

```
{  // 1a:  Insert at the back of a vector
  std::vector<GroceryItem> dataStructureUnderTest;
  measure( "Vector", "Insert at the back", noop, [&]( const GroceryItem & item ) {

    ////////////////////////// TO-DO (1) ////////////////////////////////
      /// Write the line of code to push "item" at the back of the data structure under test

    ////////////////////// END-TO-DO (1) ////////////////////////////

  } );
}
```

you would write `dataStructureUnderTest.push_back( item );` in the space provided.

The main.cpp starter code (your point of departure) compiles and executes out of the box. The table produced will contain a bunch of zeros because you haven't added your code yet, and that's okay. As you complete a section or two, it's a good idea to make sure it still compiles and executes. If you are having difficulties getting a section to work, remove (or comment out) your "bad" code and move on to the next section. When you have the first round of "easy stuff" finished, cycle back and finish the others. Be sure to submit a program that both compiles and executes with no errors.

Run time tips:

As discussed all semester, when the container holds just a few elements, the efficiency class difference between the data struture operations is not very interesting. It's only when you have large amounts of data in the container that the differences become very interesting.

In summary, this program measures in microseconds the amount of elapsed (wall-clock) time required to perform a data strucure's operation hunderds of thousdands of times, each time slightly changing the container. For example, it measures the amount of time it takes to insert an element into an empty list, then the amount of time to insert an element into a list with only one element, and so on until it measures the amount of time to insert an element into a list with roughly 100,000 elements. Then it repeats for other operations and other data structures. (1,000 microseconds = 1 millisecond, and 1,000,000 microseconds = 1 second)

This all take a considerable amount of time. Expect your program to take up to 45 to 60 minutes to finish execution when using the full sample data set (Grocery_UPC_Database_Sample.dat), depending on the capabilities of your machine. To help you during development, a smaller sample data set (small_sample_input.dat) is also provided, but may still take a minute or two to complete.

The amount of time it takes your program to run is also influenced by how your program is built. A Debug build may run 10 times slower than a Release (optimized) build. You may change Build.sh to produce Release builds by editing Build.sh and changing:

```
CommonOptions="-g3 -O0 -pthread -std=c++17 …
```

to

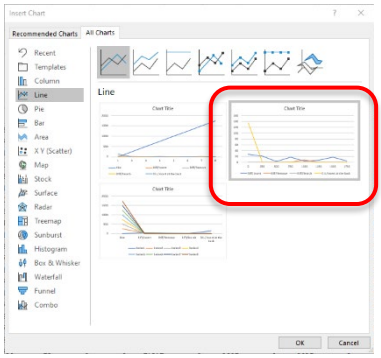
```
CommonOptions="-g0 -O3 -DNDEBUG -pthread -std=c++17 …
```

Graphing tips:

The program outputs tab seperated values (tsv) text that you redirect to a text file at the command line. Open this text file in a spreedsheet program, like Excel. Verify the data is placed in columns with headers and rows. Next, create a line graph within Excel[1] by selecting all the data, including the header row, and then inserting a new line chart. For example, if your data looks like the picture shown to the upper right, select all the cells in the 5 columns and 9 rows.

Size	BST/Inser	BST/Remc	BST/Searc	DLL/Insert
0	27	0	0	135
250	21	0	0	0
500	3	0	0	0
750	17	0	0	1
1000	4	8	0	0
1250	7	0	0	1
1500	17	0	0	0
1750	5	0	0	1

Then from the Insert Tab on the ribbon, open the Insert Chart dialog and pick the line chart option as shown to the lower right.



Of course you may generate the graph however you wish, but it should resemble the graph in the Description above. It's very likely Google Sheets and LibeOffice Calc have similar cpabilities. You may also do it by hand and scan it.

Rules and Constraints:

1. You are to modify only designated TO-DO sections. Do not modify anything outside such designated areas. Designated TO-DO sections are identified with the following comments:

   ```
   ////////////////////////// TO-DO ////////////////////////////////
   ...
   ////////////////////// END-TO-DO ////////////////////////////////
   ```

   Keep these comments and insert your code between them. In this assignment, there are 21 such sections of code you are being asked to complete. You were given the solution to one of them above. All of them are in main.cpp.

2. This assignment **requires** you redirect standard input from a text file and redirect standard output to a text file. See How to build and execute your programs. To run your program, enter something like one of the commands below (pick *one*) at the console:

   ```
   ./project_\(clang++\).exe < Grocery_UPC_Database_Sample.dat | tee output.txt
   ```
   or
   ```
   ./project_\(g++\).exe < Grocery_UPC_Database_Sample.dat | tee output.txt
   ```
   or
   ```
   "./project_(clang++).exe" < Grocery_UPC_Database_Sample.dat | tee output.txt
   ```
   or
   ```
   "./project_(g++).exe" < Grocery_UPC_Database_Sample.dat | tee output.txt
   ```
   or
   ```
   cat Grocery_UPC_Database_Sample.dat | "./project_(clang++).exe" | tee output.txt
   ```
   or
   ```
   cat Grocery_UPC_Database_Sample.dat | "./project_(g++).exe" | tee output.txt
   ```

3. Use Build.sh (potentially modified as described above) to compile and link your program – it employs the correct compile options.

[1] Search Google for "Create a line chart in Excel" for more information, how-tos, and videos.

Deliverable Artifacts:

Provided files	Files to deliver	Comments
main.cpp	1. main.cpp	Modify the file as described above.
GroceryItem.hpp Timer.hpp	2. GroceryItem.hpp 3. Timer.hpp	You should not modify these files. The grading process will overwrite whatever you deliver with the one provided with this assignment. It is important that you deliver complete solutions, so don't omit these files in your delivery.
GroceryItem.cpp	4. GroceryItem.cpp	You should replace the provided file stubs with your (potentially) updated files from the previous assignment.
	5. AnalysisReport.pdf	Your analysis report document including your results graph.
	6. output.txt	Capture your program's output to this text file and include it in your delivery. Failure to deliver this file indicates you could not get your program to execute.
small_sample_input.dat medium_sample_input.dat Grocery_UPC_Database_Sample.dat		Text files to be used as program input. Do not modify these files. They're big and unchanged, so don't include it in your delivery. Start with small_sample_input.dat then medium_sample_input.dat. Once you have a working program, use Grocery_UPC_Database_Sample.dat to generate your final output from which to graph your results
sample_output.txt		Sample output of a working program. Use for reference, your output format may be different. But the contents should match. Do not include this file with your delivery.
sample_graph.xlsx		Sample Excel file with data loaded and graph drawn. Do not include this file with your delivery.