CPSC 131, Data Structures - Spring 2021
# Container Review and Analysis Final Exam

Goals:

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

Description:

You are to perform 3 tasks, each building on the prior:

1. Complete the starter code given. Then compile and execute your program saving the output to a text file. The program's output will be a table containing the amount of actual elapsed 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 program's resulting table several different ways to support the data analysis below. This too is just a required step towards the main objective. Example graphs are shown on the right.

3. Analyze and summarize the 5 items listed below. *This* is your most important objective. For each item listed below provide two to three relevant graphs and then explain the information in the graph in terms of the operation's efficiency class (Big-Oh). Talk to what you see on the graphs and explain _why_ you see it. If the graphs are not what you expected, point that out and explain. Compare and contrast the operations performed on the different data structures. Identify and describe patterns in the graphs, including peaks, trends, and variability. Provide a concrete, real-world example where the operation is used and then select the container best suited. Your selection should be supported by the data collected and graphed. Explain why you selected the one you did, and why you did not select the others. Analyze and summarize these 5 items:

   a. Inserting into a Binary Search Tree versus a Hash Table.

   b. Removing from a Vector at the back versus the front.

   c. Searching a Vector, DLL, SLL, BST, and a Hash Table. (compare all the combinations)

   d. You select #1 – You pick which operations and structures to compare and analyze.

   e. You select #2 – You pick which operations and structures to compare and analyze.
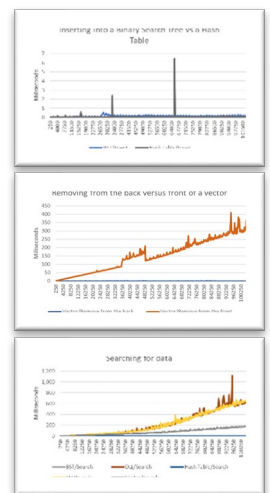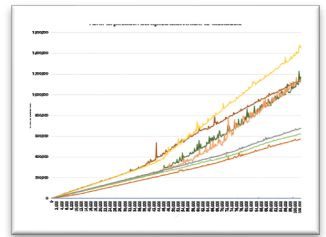
Each item has two to three graphs followed by a ½ page paragraph. Your textual response should be about 2½ full pages of text (½-inch margins, 10-point Times New Roman font, single spaced, well-formed grammatically correct sentences, etc.). Graphs are in addition to (don't count towards) your page target.

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 instead enables you to demonstrate you know how to use the STL containers, their interfaces, and related iterators. Most of the sections require only a single line of code. For example, insert a book into a container at various positions (front, back, etc.). Other sections require a few more lines of code (about 5 or 6). For example, search for a book with a particular ISBN. You've done all this before, so it should be review.

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

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

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

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

  } );
}
```

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

Be sure to submit a program that both compiles and executes with no errors and no warnings. It's better to have an incomplete section than to submit a program that does not compile clean on Tuffix using the Build.sh script.

Run time tips:

As discussed all semester, when the container holds just a few elements, the efficiency class difference between the data structure 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 structure's operation hundreds of thousands 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 20 to 30 minutes to finish execution when using the large sample data set (Open Library Database-Large.dat), depending on the capabilities of your machine. To help you during development, a smaller sample data set (Open Library Database-Small.dat) is also provided but may still take a minute or two to complete.

Graphing tips:

The program outputs tab separated values (tsv) text that you redirect to a text file at the command line. Open this text file in a spreadsheet 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 upper right picture, select all the cells in the 5 columns and 9 rows.

Size	BST/Inser	BST/Remo	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 middle right.

To select which items to graph, filter away the unwanted data series 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 LibreOffice Calc have similar capabilities.
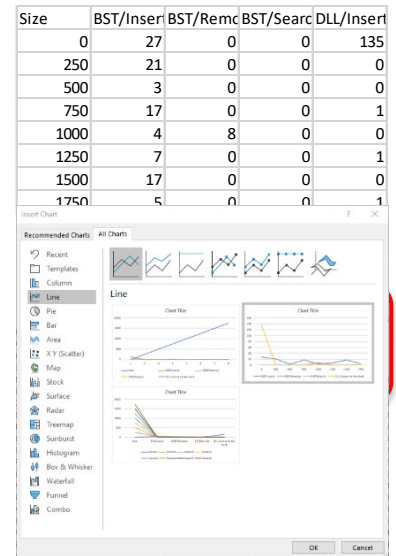
Rules and Constraints:

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 the *I/O Redirection* and *How to Build and Run Your Programs with Build.sh* in Tom's Survival Guide in Canvas. Do not run the program from within an IDE, it may alter the times collected and provide you false data to work with. To run your program, enter something like one of the commands below (pick *one*) at the console:

   ```
   ./project_\(clang++\).exe < "Open Library Database-Large.dat" | tee output.txt
   ```
   or
   ```
   ./project_\(g++\).exe < "Open Library Database-Large.dat" | tee output.txt
   ```
   or
   ```
   "./project_(clang++).exe" < "Open Library Database-Large.dat" | tee output.txt
   ```
   or
   ```
   "./project_(g++).exe" < "Open Library Database-Large.dat" | tee output.txt
   ```

3. Use Build.sh 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 this file as described above. The grading process will detect and discard any changes made outside the designated TO-DO sections, including spacing and formatting.
Book.hpp Timer.hpp	2. Book.hpp 3. Timer.hpp	Don't 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.
Book.cpp (stub)	4. Book.cpp	You should replace the provided file stub with your (potentially) updated files from the previous assignment.
	5. AnalysisReport.pdf	Your analysis report document including your resulting graphs. You may submit a MSWord .docx file instead of the .pdf file if you prefer.
	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.
Open Library Database-Small.dat Open Library Database- Large.dat		Text files to be used as program input. Do not modify these files. They're big and unchanged, so don't include them in your delivery. Start with "Open Library Database-Small.dat". Once you have a working program, use "Open Library Database-Large.dat" to generate your final output from which to graph your results
RegressionTests/ BookTests.cpp CheckResults.hpp		When you're far enough along and ready to have your class tested, then place this folder in your working directory. These tests will be added to your delivery and executed during the grading process.