

CPSC 131, Data Structures – Spring 2022

Container Review and Analysis Final Exam

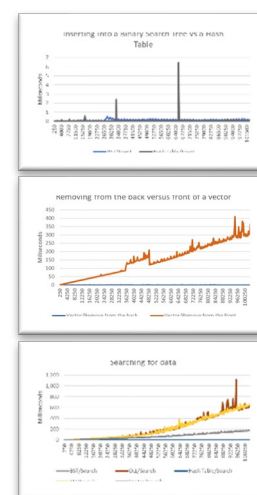
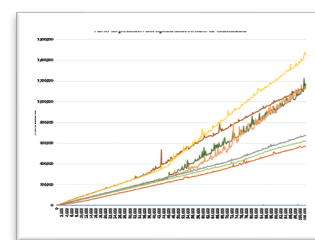
Goals:

- 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 Array, 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
 - $O(1)$, $O(\log_2 n)$, $O(n)$, $O(n \cdot \log_2 n)$, 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 comma-separated value (.csv) 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). Complete this task before moving on to the next.
2. Graph the program's resulting table several different ways to support the data analysis below. Complete this task before moving on to the next. Example graphs are shown on the right.
3. Analyze and summarize the 5 items listed below. 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 Vector at the back versus a Hash Table.
 - b. Removing from a SLL at the back versus a Vector at the front.
 - c. Searching a Vector, DLL, SLL, BST, and a Hash Table. (analyze all 10 pairs)
 - d. You select #1 – You pick which remaining operations and structures to compare and analyze.
 - e. You select #2 – You pick which remaining operations and structures to compare and analyze.



Each item has two to four graphs followed by a ½ page textual analysis response comprised of well-formed grammatically correct sentences. Graphs and real-world usage examples are in addition to (don't count towards) your page count target. Use the provided AnalysisReport.docx with the required document layout, format, margins, font size, paragraph structure, etc. for your answers.

You are given Operations.hpp 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 understand the data structures and 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:

```
struct insert_at_back_of_vector
{
    // Function takes a constant Book as a parameter, inserts that book at the back of a
    // vector, and returns nothing.
    void operator()( const Book & book )
    {
        ////////////////////////////////// TO-DO (1) //////////////////////////////////
        /// Write the lines of code to insert "book" at the back of "my_vector"
        my_vector.push_back( book );
        ////////////////////////////////// END-TO-DO (1) //////////////////////////////////
    }

    std::vector<Book> & my_vector;
};
```

you would write `my_vector.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. Most sections compile clean out of the box, others may require you to provide at least a return statement. A good way to start is to get your program to compile and execute clean, inserting just return statements as necessary, before solving any of the sections.

Run time tips:

As discussed all semester, when the container holds just a few elements, the efficiency class difference between 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 nanoseconds 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 second = 1 billion (1,000,000,000) nanoseconds.

This all takes a considerable amount of time. Expect your program to take up to 15 or 20 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 only, a smaller sample data set (Open Library Database-Small.dat) is also provided but may still take a minute or two to complete.

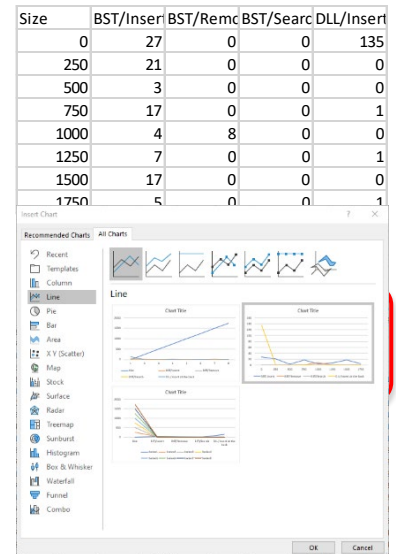
Graphing tips:

The program outputs comma-separated value (csv) text that you redirect to a file at the command line. Open this .csv file in a spreadsheet program, like Excel. Verify the data is placed in columns with headers and rows. Next, create a line graph within Excel¹ 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.

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. LibreOffice Calc² has similar capabilities. Do not upload data to the cloud³ (e.g., Google Drive, One Drive).



Rules and Constraints:

1. You are to modify only designated TO-DO sections. **The grading process will detect and discard any changes made outside the designated TO-DO sections, including spacing and formatting.** Designated TO-DO sections are identified with the following comments:

```

//////////////////// TO-DO (X) //////////////////////
...
//////////////////// END-TO-DO (X) //////////////////////

```

Keep and do not alter these comments. 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 Operations.hpp.

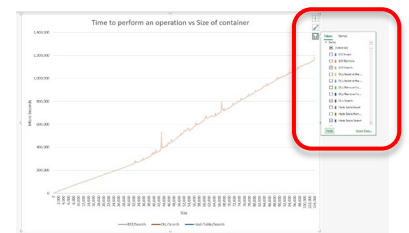
2. This assignment **requires** you redirect standard input from a text file and redirect standard output to a text file. See the [I/O Redirection](#) and [How to Build and Run Your Programs with Build.sh](#) in Tom's Survival Guide in Canvas. Do not run the program from within an IDE, it may alter the times collected and provide you false data to work with. To run your program, enter:

```
program < input_data_file > output.csv
```

where `program` is the name of your executable image file, `input_data_file` is the name of the file containing input data. If the file name contains special characters or spaces, enclose it in double quotes. For example:

```
"./project_clang++.exe" < "Open Library Database-Large.dat" > output.csv
```

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. You must not use One Drive

² View a [Charting Size vs Time](#) video tutorial to create graphs with LibreOffice Calc on Tuffix [here](#).

³ Google Sheets & Docs, and Word & Excel Online produce distinctive results, are easily detectable, and must be avoided.

Deliverable Artifacts:

Provided files	Files to deliver	Comments
Operations.hpp	1. Operations.hpp	Modify this file as described above. The grading process detects and discards all changes made outside the designated TO-DO sections , including spacing and formatting.
Book.cpp	2. Book.cpp	Your (potentially) updated file from the previous assignment.
main.cpp Book.hpp Timer.hpp	3. main.cpp 4. Book.hpp 5. Timer.hpp	Don't modify these files. The grading process overwrites 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.
AnalysisReport.docx	6. AnalysisReport.docx	Your analysis report document including your resulting graphs. Use the provided document as a template to prepare your report. You may submit a .pdf file if you must, but the document format and structure provided must be followed.
sample-output.csv	7. output.csv	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" as the input used to create your final output.csv file.
RegressionTests/ BookTests.cpp CheckResults.hpp OperationsTests.cpp		When you're far enough along and ready to have your work tested, then place this folder in your working directory and rebuild. These tests will be added to your delivery and executed during the grading process. The grading process expects all tests to pass.