**Data Structures (2080C) -- Fall 2018 – Lab 6**

***Topics covered: Sorting***

*Homework due:* ***Monday, Oct 22***

**Objective:**

The objective of this homework is to investigate different sorting algorithms.

**Scenario:**

In this assignment, we are going to be investigating the actual performance of different sorting methods. This needs to be written using C++.

**Requirements:**

1. Create a program that generate an array of sizes n= 10, 100, 500, 5000 and 25000 items. Your program should populate those arrays with randomly generated integers with a value between 0 and the 2n where n is the size of the array. Create an implementation for the following sort operations.
   1. Bubble sort
   2. Insertion sort
   3. Merge-sort
   4. Quicksort
   5. Radix-sort // you probably want to consider storing the values as strings
2. Test each of the sort operations and record the time the sort takes to complete. You should test each on the same unsorted array to get the best comparison. You should do this for each array size (from requirement 1) a minimum of 10 times. Your test should use the chrono library’s [high\_resolution\_clock](http://www.cplusplus.com/reference/chrono/high_resolution_clock/) class. The following example of how to do this in nanoseconds is found on [Stack Overflow](http://stackoverflow.com/questions/3220477/how-to-use-clock-in-c). Only the time in the 5 sort functions should be measured.  
   1. #include <iostream>
   2. #include <chrono>
   3. typedef std::chrono::high\_resolution\_clock Clock;
   4. int main()
   5. {
   6. auto t1 = Clock::now();
   7. auto t2 = Clock::now();
   8. std::cout << "Delta t2-t1: "
   9. << std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count()
   10. << " nanoseconds" << std::endl;
   11. }

Use the output of this to make a table similar to (these numbers are completely fabricated) the following with the values showing the average of all your runs for this test type:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 10 | 100 | 500 | 5000 | 25000 |
| Bubble sort | 10 ms | 1000 ms | 25004 ms | 2500312 ms | > 5 minutes |
| Insertion sort | 11 ms | 708 ms | 45646 ms | 98984 ms | 98798797 ms |
| Merge-sort | 50 ms | 652 ms | 44646 ms | Ran out of memory | Ran out of memory |
| Quicksort | 42 ms | 753 ms | 88544 ms | 990090 ms | Ran out of memory |
| Radix-sort | 512 ms | 1101 ms | 2002 ms | 77757 ms | 7747474 ms |

If any 1 test run takes longer than 5 minutes, you may discontinue that test and record that the time took longer than 5 minutes. Likewise, should any test crash due to running out of memory, record that as well. Graph this data and explain how well or poorly it matches your expectations for performance given the known Big O notation for the given sort algorithms. Include what you expected for time for each of the array sizes based on the results for array size of 10.

1

**Submission:**

Submit all source code files and any required data files in a zip file. Include a write up as a PDF including:

* Table and graph from requirement 2 along with your writeup of the results from that requirement.

Submission should be submitted via BlackBoard.

**Grading:**

1. 20% - Requirement 1 is completed and working
2. 40% - Code for Requirement 2 is completed and working
3. 30% - Analysis for Requirement 2 explains your results including any unexpected results.
4. 10% - Code is well formatted, well commented and follows a reasonable style.

If program fails to compile, the grade will be limited to a max grade of 50%.