COSC 320 - Advanced Data Structures and Algorithm Analysis
Lab 1

Dr. Joe Anderson

Due: 8 Februrary 2018

1 Objectives

In this lab you will focus on the following objectives:

1. Review dynamic memory and array usage in `c++`
2. Explore empirical tests for program efficiency
3. Compare theoretical algorithm analysis with practical implementations

2 Tasks

1. Put your code in a folder called "Lab-1". This folder will be zipped and turned in at the end.
2. Write a function to take an array parameter and sort it in place with your own implementation of the Bubble Sort. Include the following in the output:
   (a) The number of swaps that are made during the sorting process
   (b) The time it takes for the sorting to happen using the standard library utilities (requires compiler argument `-std=c++11`). One example of how to do this:

```
#include<chrono>

...

// The "auto" type determines the correct type at compile-time
auto start = std::chrono::system_clock::now();
myFunction(); // replace with your sorting alorithm
auto end = std::chrono::system_clock::now();

std::chrono::duration<double> elapsed_seconds = end-start;
std::time_t end_time = std::chrono::system_clock::to_time_t(end);

std::cout << "finished at " << std::ctime(&end_time)
          << "elapsed time: " << elapsed_seconds.count() << "s\n";
```

3. Also write a function to validate that a given integer array is in sorted order.
4. Test your sorting algorithm on different sized arrays.

(a) Use some small (≈ 100 elements) and some large ($\approx 1,000,000$ elements) arrays, and various sizes in between

(b) Use some arrays that are already sorted

(c) Use some arrays that are sorted backwards

(d) Use some arrays that contain many duplicate elements

(e) Use some arrays that are randomly generated

5. Include a `Makefile` to build your code.

6. Include a `README` file to document your code, any interesting design choices you made, and answer the following questions:

   (a) How does the timing scale with the number of elements in the array? The size of the elements?

   (b) What is the time complexity of your sorting algorithm, in terms of the array size?

   (c) How does your algorithm perform on an array that is already sorted? Sorted in reverse? Random order? Give specific benchmarks resulting from your code.

   (d) How could the code be improved in terms of usability, efficiency, and robustness?

3 Submission

All submitted labs must compile with `g++` and run on the COSC Linux environment.

Upload your project files to MyClasses in a single `.zip` file.

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.