CSCI 311 - Algorithms and Data Structures
Project 1

Assignment Date: Feb 12, 2025
Due Date: 11:59pm on Mar 14, 2025
Grace Date: 11:59pm on Mar 17, 2025

For this project you will be implementing, timing, and analyzing several sorting algorithms. There is very little direction regarding how to design or structure this project. These decisions are left to you. Make sure to **start early** and to **stay organized**. This project is worth 100 points.

Solutions to the written portion of this project should be submitted as a PDF file on Canvas. Make sure to justify your answers. **C++ code should also be submitted on Canvas as well.**

All code should be written in a single file named **project_1.cpp**. Tests and experiments can be included as separate functions or in the main function directly. However, you must include an option to run a simple small ltest in isolation (see Section 2 Number 1). Also submit all the related files (the graphs and the excel sheet files). All vectors should be of integer type. Remember, coding style and comments matter. Poor style or a lack of comments may cost you points! (up to 10%)

What to Submit on Canvas (Reminder)

A) Project_1.cpp
B) a pdf file for the answers to the written parts
C) a folder that contains all the related files you will have generated for project 1. (Specifically, the pictures for the graphs and the excel sheets you will generate for project 1, etc. See "sample plotData related files.zip" for reference.)
D) Put A, B, and C parts into a single zip file and submit it.

There will be time in lab to discuss these problems in small groups and I highly encourage you to collaborate with one another outside of class. However, you must write up your own solutions **independently** of one another. Do not post solutions in any way. Also, please include a list of the people you work with in a comment at the top of your project_1.cpp file.

Good luck!

1 Coding Questions

For the following questions, make sure to follow the pseudocode discussed in class.

1. (5 pts) Implement a function that checks whether or not a vector is sorted.
2. (10 pts) Implement bubble sort.
3. (11 pts) Implement insertion sort.
4. (11 pts) Implement selection sort.
5. (13 pts) Implement quicksort.

2 Testing and Experiments

1. (10 pts) Write a function that runs each of the sorting algorithms on ten random vectors of length 100, collects the run times, verifies that the sorting was successful, and prints out the minimum, mean, standard deviation, and maximum of the run times. Example output for a single sort might look something like this:

   ```
   ************************
   Bubble sort on 10 vectors of length 100
   Sorting successful
   Minimum: 1 sec; Mean: 1 sec; Standard deviation: 1 sec; Maximum: 1 sec
   ************************
   ```

   This should be what is run by default after making your programming and running "./a.out" inthe terminal.

2. (10 pts) Collect worst-, average-, and best-case run time data associated with each algorithm.

   (a) Pick at least five vector lengths ranging from small to large. These sizes should cover a range that begins to show asymptotic growth of the run time of these sorting algorithms. The lengths 10, 100, 1000, 5000, and 10000 are probably a good place to start though this will depend somewhat on your machine.

   (b) For each length and each type of vector (worst-, average-, and best-case), generate 50 vectors.

   (c) Sort these vectors using each algorithm and collect the run times. You may want to save these values in a separate output file.

   (d) In the end, you should have 12 (3×4) sets of times, one for each case/algorithm pair. Each set should include a total of 250 (5×50) run times, 50 times for each vector size.

   (e) To use plotData (described in the next section) this data should be stored in three CSV files, one for worst-, average-, and best-case times. Each line in these files should be in the form of algorithm, size, and time where algorithm is one of bubble, insertion, selection, or quick. For example, several lines of a best_case_times.csv file might look something like this: (see the sample excel file for more examples)

   ```
   bubble,10,7e-07
   insertion,10,6e-07
   selection,100,2.67e-05
   quick,10000,0.0143179
   ```

   Collecting this data took about 284 seconds on my machine. I recommend testing your code with a smaller number of repeats on shorter vectors to make sure that everything is working as expected before running all tests.

3 Written Questions

1. (18 pts) For each algorithm, determine the worst-, average-, and best-case asymptotic run times (1 pt each). In addition, describe what kind of vectors (i.e. sorted, reversed, or something else) achieve these bounds (1 pt each).

Algorithm	Worst-case	Average-case	Best-case
Bubble sort			
Insertion sort			
Selection sort			
Quicksort			

2. (5 pts) Generate three graphs, one for best-, average-, and worst-case scenarios, using the data collected in your experiments and the plotData executable provided on Canvas. plotData takes three arguments:

   (a) --data is the CSV file holding the relevant data.

   (b) --case should be Best, Worst, or Average.

   (c) --save is the file where the resulting figure will be saved.

   For example,
   `./plotData --data best.csv --case Best --save best_fig`
   generates figures for best-case run times in the file best.csv and saves the figure as best_fig.png.

   Note: If you want to run plotData on the server or the lab machine, you will need to execute the following command "pip install matplotlib"
   Note: If you want to run it on your own computer, you will need to
   1) install current version of python (python.org)
   2) open the command line windows and run "pip install matplotlib"

3. (7 pts) For each of the graphs generated in (2), what do you see? Do the curves follow the patterns that we expect? Do certain algorithms seem faster than others in different cases? Are there any surprises? Be specific and refer back to your answers to question 3.1.

4 Potentially Useful Code Snippets

For the code snippets below to work, you must include time.h, math.h, and chrono.

```
#include  <time.h>
#include <math.h>
#include <chrono>
```

This line sets the random seed. You may want to include it as the first line in your main.

```
srand(time(NULL));
```

This function generates a vector of random integers in a specified range.

```
/*****************************************************************************
 * Generate a vector of random integers in a given range. The ends           *
 * of this range are inclusive.                                              *
 * size - int - the number of integers in the resulting vector               *
 * low, high - int - the range from which to draw random integers (inclusive) *
 * return - vector<int> - a vector of random integers                        *
 *****************************************************************************/
vector<int> randomVector(int size, int low, int high){
  vector<int> v(size, 0);
  for (int i = 0; i < size; i++){
    v[i] = rand() % (high - low +1) + low;
  }
  return v;
}
```

This function calculates sample standard deviation.

```
/*************************************************************
 * Calculate the sample standard deviation of a vector of doubles *
 * v - const vector<double> - a vector of doubles              *
 * return - double - the sample standard deviation of v        *
 *************************************************************/
double sampleSD(const vector<double> v){
  double mean = 0;
  for (int i = 0; i < v.size(); i++){
    mean += v[i];
  }
  mean = mean / v.size();
  double sd = 0;
  for (int i = 0; i < v.size(); i++){
    sd += (v[i]-mean)*(v[i]-mean);
  }
  sd = sd / (v.size()-1);
  return sqrt(sd);
}
```

This code snippet, borrowed in large part from lab 1, shows one way to time functions in C++.

```
chrono::high_resolution_clock::time_point start;
chrono::high_resolution_clock::time_point end;
cout << "Sieve of Erathosthenes on primes below 100,000" << endl;
start = chrono::high_resolution_clock::now();
int sum = sieveOfErathosthenes(100000);
end = chrono::high_resolution_clock::now();
double elapsed = chrono::duration_cast<chrono::duration<double>>(end - start).count();
cout << "Sum: " << sum << "; Elapsed time: " << elapsed << endl;
```

5 Notes on plotData

If you want to use “plotData” on the server or lab machine, do the following on the terminal first.

pip install matplotlib

If the above command generates errors for you, then you also need to do this

pip install –user tornado

If “./plotData …” command freezes your machine, then use the following command instead.

python3 plotData.py …