# Project A2

**Due Date: March 6** (Note change from syllabus)

## Purpose

There are two main objectives for this project: One is to use large data sets, and the second is to perform analysis using such data. In your previous courses, the input may have been in the form of "toy" or synthetic data sets. Now you will use some real data to better see the power and limitations of algorithms and the programming language.

## Problem

In a GIS (Geographic Information System), all data is related in terms of a reference location on the globe. Thus, no matter what the GIS is being used for, the underlying data set refers to topographic data. One such data set is called a Digital Elevation Model, or DEM (so clever, eh?), which stores elevation values (as floats) in a rectangular grid. The file is indexed by the $x$ and $y$ positions relative to the latitude and longitude coordinates that position the grid to a location on the earth. These coordinates are given in the header information, but all you will need to do is index the grid and not worry about latitude and longitude. DEM files are used to compute all manner of statistics. You will do just that, as well as compare some timing data.

## Input

The program should prompt for a DEM filename. You may assume that the input files contain the proper data; no error checking of their contents is needed. However, you are not guaranteed that the filename itself is valid, so an error check for that is in order.

The files will contain a DEM in the ESRI ASCII Grid format (see The Google for more info). The file has a header that describes the data. The first two lines of the data file contain two integers, denoting the $c$ columns and $r$ rows of data, respectively. The file contains several more lines of descriptive data, some of which you won't need and some that you may. This header information is then followed by $r$ rows of $c$ floating point values representing terrain elevations. A sample input file is available on the course web page. Expect large files.

Once this is input, your program should ask the user for which row ('r') or column ('c') to process as well as a file name for the output. This process should repeat until the user quits ('q'):

```
Enter file name -> franconia.dem.grd
Enter 'r' (row), 'c' (column), or 'q' (quit) -> r
Enter row (1-1200) -> 830
Enter output file name -> row830.dat
```

—- your output goes here (see below) —-

```
Enter 'r' (row), 'c' (column), or 'q' (quit) -> q
All done!
```

## Output

### Part 1

For each iteration of user input, the program should display an **aligned** and **labeled** table with the following rows of statistics:

Number of data points in the row/column
Highest elevation in the row/column
Lowest elevation in the row/column
Mean elevation in the row/column
Median elevation (oh, oh) of the row/column
Sorting time (bubble or selection) in milliseconds/microseconds
Sorting time (qsort) in milliseconds/microseconds

Floating point values should be shown with two decimal places. Timing values should be in milliseconds or microseconds, whichever makes more sense given two decimal places. Values should be aligned by the decimal point. In addition, the $x$ (row) or $y$ (column) coordinate and the elevation of each point in the row/column should be stored in the file specified by the user. This will be used to graph the profile.

To find the median, all of the values must be in sorted order[1]. To do this, you will call a built-in sort method (see below) as well as using Bubble or Selection sort (you can choose which you want). Thus, you will sort the same data twice; this is for comparison purposes. When calculating this, **do not** count the time taken for any other statements besides the sort.

### Part 2

Using LaTeX[2] or a word processor, type up a short (one or two pages, including graphs) report describing the time efficiency of each sort relative to the size of the input. Most importantly, write an analysis comparing the efficiency statistics with the theoretical Big O figures of the two sorting routines. Be clear as to which sort is generating the timing data! Furthermore, you must test with more data files than with those supplied. One way to do this is to generate random files. Another way is to simply cut-n-paste from the given files to create new ones. A suite of files of different sizes and/or data patterns is needed to produce enough data to write a comprehensive report. Use one or more graphs to augment your argument to convince me of the efficiency of each of the sorting methods.

As part of this report, show a line graph of the row/column that was used to generate the statistics. This should be a profile of the DEM data, and should look like recognizable terrain.

## Specifics

- For this project, you should call the C++ `qsort()` routine. Implement the Bubble/Selection sort yourself, using available code (you do not have to reinvent sorting routines). Both sorts work on arrays, but the method calls might be slightly different. How you compare the sorting routines is up to you, but you should test each sort with each input file at minimum. How you test is up to you, and should be made clear in the report.
- Make your life easy: After reading the file header information, read the rest of the numeric data into a `float` array.

[1] Well, we'll see about that later in the semester.
[2] Uh, what??

- The program must be written in C++. You can use any compiler/IDE.
- Students sometimes have a tendency (from Python??) to read all data as strings. I **strongly discourage** reading the floating point data as strings. If you insist on reading the data as strings, note that various operating systems treat character data differently. Be sure to test on Linux using the `g++` compiler if you are reading all of the data as strings.
- Timing of the code segments can be done as described in class, using the `clock()` function in `<time.h>`.
- Design a good program using OOP principles. Make appropriate objects. Points will be awarded for good code reuse.
- Follow good programming practices. Use good identifier names, comment all variables and methods, and do not use global variables. As before, be sure to have an introductory comment describing the program.
- The report should be clear, neat, have no spelling errors, and use correct grammar. AI can help out here, but it's you who has to have the correct data, graphs, and interpretation.

## Notes

- Be sure your program reads the data file correctly before continuing!
- Work on smaller files and/or generate a random set of values to test your program's functionality. Remember that Step 1 is to have a working solution; once it works, then you can test with larger data sets and correct any problems you may find.
- Leave yourself adequate time to write and properly format your report. A sloppy report will adversely affect your grade. The report is worth 25% of the grade.
- Zip your code together along with your report using the same naming convention as in the previous project, as in `gousieA2.zip`. **Zip only your code! I do not want either of the `__MACOSX` or `cmake-build-debug` folders!** Figure out how to zip up only your `.cpp` and `.h` files (and your report), not the entire folder!
- Turn in the project via Canvas before 11:59:59 PM on the due date. This is the Friday before break, so you will want to submit earlier!

*The generation of random numbers is too important to be left to chance.*
– Robert R. Coveyou