CSCI 1320 Computer Science I: Engineering Applications – Fall 2018
Instructor: Zagrodzki
Assignment 9: Due Sunday, Nov 18, by 6pm

Sorting Arrays

In this assignment you will get to experiment with arrays in C++. You will implement a sorting algorithm and use functions defined with appropriate kind of parameters.

You will need to create 4 versions of your program, in a incremental fashion. You should call your files:

1. `a9<lastNameFirstName>A.cpp`
2. `a9<lastNameFirstName>B.cpp`
3. `a9<lastNameFirstName>C.cpp`.
4. `a9<lastNameFirstName>D.cpp`

Your program needs to accomplish the following:

A. Generating Random Arrays

Generates 2 arrays of random integers between 0 and 1000. Both arrays need to have length of *N*. Set N = 10 for now. Declare the arrays inside of main() and call them *arrayA* and *arrayB*. Then have a separate function `randArrays` that fills two arrays with random numbers. `randArrays` should be called with the two arrays from main.

B. Sorting Arrays

Sort the first array (call it *arrayA*) in ascending order. Sort the second array (call it *arrayB*) in descending order.

Have a function `sortArr` that takes in an array, array length, and a boolean value to indicate whether to sort in ascending or descending order. Function `sortArr` needs to call another function, call it `swap2`, for performing the array element swap operations required for sorting.

You should call `sortArr` twice from `main` to sort each array.

Many sorting algorithms exist. A very simple one (though quite inefficient) is the Bubble Sort algorithm. The Bubble Sort implementation is given below and can be used as starter code:

```
for(int i = length-1; i>0; i--)
{
     for(int j = 0; j < i; j++){
          if(arr[j]>arr[j+1]){
               temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
          }
     }
}// Ascending or descending implementation?
```

C. Writing contents of 2 arrays into 1

Declare a new array of length *2N* inside of main and call it *arrayC.* Have a function `arrayMerge` that gets called with 4 arguments (the 3 arrays and N). Copy the elements of *arrayA* into first half of *arrayC* and elements of *arrayB* into second half of *arrayC*.

D. Performance

Set N = 10e3. Inside of main, add timing code to measure the amount of time it takes to run the whole program starting with `randArrays` and ending with `arrayMerge`. Print the total execution time to the command prompt. If you have a local compilation environment setup, try running your program locally vs on JupyterHub. Is one faster than the other?

You can use the following technique for measuring execution time (make sure to include the ctime library):

```
int startTime, endTime;
startTime = clock();
// code you want to time here
endTime = clock();
double execTime = (double)(endTime-startTime)/CLOCKS_PER_SEC;
```

Submitting the assignment:

Zip all the .cpp files together and submit the resulting *.zip* file through Moodle as Assignment 9 by due date. You do not need to submit any executable files.

NOTE: if your C++ file does not compile with standard g++ compiler in Jupyter Hub you will get a zero on the C++ portion of the assignment (no partial credit). This will be the case with all C++ assignments going forward.