Project 5: `Sorting`

The sorting algorithms:

For this project you will code and analyze 3 sorting algorithms:
SelectionSort (https://en.wikipedia.org/wiki/Selection_sort **)**
InsertionSort (https://en.wikipedia.org/wiki/Insertion_sort)
MergeSort (https://en.wikipedia.org/wiki/Merge_sort **)**
Your sorting algorithms may (or may not) use helper functions as you see fit.
Here is the prototype for these functions, you must conform to the interface (the full interface can be found on blackboard).

```
/**
 @post Sorts an array in ascending order using the selection sort algorithm.
 @param a the array to sort
 @param size the number of elements in a
 */
void selectionSort(int a[], size_t size);

/**
 @post Sorts an array in ascending order using insertion sort.
 @param a the array to sort
 @param size of the array
 */
void insertionSort(int a[], size_t size);

/**
 @post Sorts the elements in a range of a array.
 @param a the array with the elements to sort
 @param from the first position in the range to sort
 @param to the last position in the range to sort (included)
 */
void mergeSort(int a[], int from, int to);
```

The testing functions:

You will also implement a **testing function** to analyze the behavior of these sorting algorithms. The testing function will do the following:

- Sort a copy of the array 10 times and measure the runtime
- Return the average runtime over the 10 runs

Because selectionSort and insertionSort take the same parameters, we can use a single testing function for both, and pass a pointer to the sorting function we want to use as a parameter (namely selectionSort or insertionSort).
MergeSort, on the other hand, takes a third parameter, so we must write a separate testing function for it.
Note: these two functions essentially do the same thing. Once you get the first one right, you only need to tweak it to obtain the other.

The prototypes for the testing functions are as follows:

```
/**
 @post Sorts values in ascending order and averages its runtime over 10 runs
 @param sortingFunction the function used to sort the array
 @param values the array to sort
 @param size of the array to sort
 @return the average runtime in microseconds
 */
double sortTest(void (*sortingFunction)(int a[], size_t size), int values[],
size_t size)
```

```
/**
 @post Sorts values in ascending order using mergeSort and averages its
runtime over 10 runs
 @param values the array to sort
 @param size of the array to sort
 @return the average runtime in microseconds
 */
double mergeSortTest(int values[], size_t size)
```

The data:

You will run the testing function on the following data:

- An array of randomly generated numbers
- An array that is already sorted in ascending order
- An array that is sorted in reverse (descending) order
- An array that is sorted **except** for the last 10 numbers which are random
- An array of **few** randomly generated numbers.

These functions' prototypes are:

```
/**
 @post Populates values with randomly generated numbers in range size
 @param values the array to populate
 @param size of the array to be populated
 */
void generateRandomArray(int values[], size_t size);

/**
 @post Populates values with integers in ascending order
 @param values the array to populate
 @param size of the array to be populated
 */
void generateAscendingArray(int values[], size_t size);

/**
 @post Populates values with integers in descending order
 @param values the array to populate
 @param size of the array to be populated
 */
void generateDescendingArray(int values[], size_t size);

/**
 @post Populates values with integers in ascending order except for the last
10 that are randomly generated
 @param values the array to populate
 @param size of the array to be populated
 */
void generateLastTenRandomArray(int values[], size_t size);

/**
 @post Populates values with integers in randomly generated in range size/10
 @param values the array to populate
 @param size of the array to be populated
 */void generateFewRandomArray(int values[], size_t size);
```

IMPLEMENTATION TIPS:

Timing the runtime:

To time your sort algorithms you can use the C++11 library chrono (you don't have to, if you prefer to use something else it is fine, as long as you measure runtime in microseconds).
To use chrono:

```
#include <chrono>
using namespace std::chrono;
```

To measure the runtime of some function, you must obtain a timestamp right before you call that function and another one right after. To do so with chrono, you can use `high_resolution_clock().now();`
For example:

```
auto start_time = high_resolution_clock().now();
```

Once you have the start and end times, you can subtract start from end to obtain the duration. With chrono you can do it as follows:

```
duration<float, std::micro> runt_time = duration_cast<microseconds>(end -
start);
```

Finally, you are measuring this each time you run the sorting algorithm (10 times), and you want to take the average. So you must save this time interval in a sum variable that you can average at the end. With chrono you can use the count() function to extract the actual interval.

```
runtime_sum += runt_time.count();
```

Generating random numbers:

You can do this too in many ways. Here is a suggestion:
First generated a seed using a timestamp, otherwise you will always generate the same "random" sequence (not so random after all).

```
srand(static_cast<unsigned>(time(0)));
```

Now you can use the rand() function to generate a random number, and don't forget that we are using the size of the array to set the range in which the random numbers will be generated.

```
some_random_variable = rand() % size;
```

Pointers to functions as parameters:

You can pass a pointer to a function as parameter to another function. Since our test functions will do the same thing except run a different sorting algorithm, we can pass a pointer to the sorting function as a parameter. The syntax is as specified above in the function prototypes.

```
double sortTest(void (*sortingFunction)(int a[], size_t size), int values[],
size_t size)
```

In the body of sortTet, you simply make a function call using the name of the parameter:

```
sortingFunction(sorted_values, size);
```

When calling sortTest (this would happen in main), you pass the name of the actual sorting function you want to run the test on, but you must pass it by reference, for example:

```
sortTest(&selectionSort, values, ARRAY_SIZE)
```

Implementation and Testing:

I am sure that by now you are a pro at **INCREMENTAL** implementation and testing… but I will say it again.

Do this step by step!!!

1. Implement one sorting algorithm (selectionSort or insertionSort).
2. Test it on any array, print it out an make sure its sorting correctly.
3. Write and test the array populating functions (ONE BY ONE), print out the array contents and make sure you are generating the desired sequences.
4. Write and test the sortTest function. Run it with the sorting algorithm you implemented, print things out to observe behavior and make sure you are doing things correctly.
5. Now that you are correctly populating the arrays and correctly running the sortTest, implement the other sorting algorithm (selectionSort or insertionSort).
6. Test it with sortTest.
7. Implement and test mergeSort.
8. Implement and test mergeSortTest (this is just tweak on sortTest)

Usage:

To run your tests you will write your own main() function that will make calls to the array populating functions and then call the test functions on each type of array. You should run tests for each sorting algorithm on each type of data. Do this for arrays of size 100, 1,000, 10,000, 100,000

Print out the averages and observe. What are the differences? Is mergeSort always better? Is there some data on which other sorting algorithms perform better than mergeSort. When does the size of the array become unmanageable and for which algorithm?

Submission:

All your functions will live in a single file called **SortingTest.cpp,** you will **submit this file only**.
The interface provided on BlackBoard under CourseMaterials/Project4. is called **SortingTest.hpp**, you will not submit this but make sure to include it in your SortingTest.cpp

```
#include “SortingTests.hpp"
```

Your project must be submitted on Gradescope. The due date is Tuesday November 27 by NOON (12pm). No late submissions will be accepted.

I strongly encourage you to **START EARLY!!!!!**

Have Fun!!!!!