Programming Assignment #5
Incentive Date: Apr 16th
Due Date: Apr 19th
Sorting

Description:
For this programming assignment, you will implement four different sorting algorithms: Bubble Sort, Heap Sort, Merge Sort, and QuickSort. Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. Heap sort builds a heap structure and repeatedly extracts the minimum element. Merge sort divides the array, sorts subarrays, and merges them. QuickSort partitions the array into smaller sections recursively, sorting each partition independently.

starter.zip (Starter Code):
The following is a brief description of the starter code provided to you:

- **sort.cpp**
  - **void bubbleSort(int* arr, int size)**
  - **void heapSort(int* arr, int size)**
    - You can use the STL priority queue (which uses a heap) to implement the heap sort.
  - **void mergeSort(int* arr, int start, int end)**
  - **void quickSort(int* arr, int start, int end)**
  - You will implement the respective sorting algorithms for each of these functions. The ***arr*** passed in as the parameter must contain the final sorted sequence of numbers.
  - The initial call to mergeSort and quickSort will be using "start" as 0 and "end" as size - 1 (where size is the number of elements in arr)
- **main.cpp**
  - You can use this file to test each of your sorting functions.

Contract (TASK):
Your task is to implement the bubble, heap, merge, and quick sort functions. You have been provided with their non-functional definitions. **You must implement these function definitions in the given sort.cpp file**, by replacing the current non-functional definitions with your functional implementations.

[Advisory]

- **There are several ways to pick a pivot for quicksort such as just choosing the last element in the array or using the median of three strategy. You can use any method in your implementation, however, note that the median of three tends to perform better.**
  - **If you choose to select a random element as a pivot, please do not use the srand() function within your code to set the seed as this will be set by the test script.**

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<PA5>.zip** you must submit the following file(s) to Canvas:

- **sort.cpp** (containing the functional definitions of the bubble, heap, merge, and quick sort)
- **PA5_Report.pdf** (report details mentioned below)

Do not use angular brackets in the name of the submission folder.

Grading:
Coding (60 points)
The different sorting algorithms will be tested using an automatic testing infrastructure. You will be provided with the testing infrastructure which you can use to evaluate your implementations.

The grading rubric is as follows:

- Bubble Sort: 15 points
- Heap Sort: 15 points
- Merge Sort: 15 points
- Quick Sort: 15 points

Each of these functions will be timed to ensure you implement them in the right manner. You will be penalized 5 points if you don't meet the timing expectations for each of the sorting algorithms.
Some of the timings recorded by our machines for the algorithms are as follows:

- Bubble Sort
  - Mac M1: 19s
  - WSL: 43s
- Heap Sort
  - Mac M1: 0.56s
  - WSL: 3.89s
- Merge Sort
  - Mac M1: 0.16s
  - WSL: 0.44s
- Quick Sort
  - Mac M1: 0.12s
  - WSL: 0.63s

Report (20 points)
You will write a brief report that includes theoretical analysis, a description of your experiments, and discussion of your results. At a minimum, your report should include the following sections:

- **Introduction**: In this section, you should describe the objective of this assignment.
- **Theoretical Analysis**: In this section, you should provide an analysis of the complexity of each sorting algorithm on the different input types. What do you expect the complexity should be for each algorithm and input type?

- **Experimental Setup:** In this section, you should provide a description of your experimental setup, which includes but is not limited to
  - How did you generate the test inputs? What input sizes did you test? Why?
  - How many times did you repeat each experiment?
- **Experimental Results:** In this section, you should compare the performance (running time) of the sort operation for the four different sorting algorithms to one another and to their theoretical complexity.
  - Make a plot showing the running time (y-axis) vs. the number of elements (x-axis). You must use some electronic tool (matlab, gnuplot, excel, . . .) to create the plot – hand-written plots will NOT be accepted.
    - Suggested data points for number of elements are 10, 100, 1000, 10000, 100000, 1000000
      - You don't need to go as far as 1000000 for bubble sort
  - Provide a discussion of your results, which includes but is not limited to:
    - Which of the sorting algorithms performs the best and why? Does it depend on the input? Why or why not?
  - To what extent does the theoretical analysis agree with the experimental results? Attempt to understand and explain any discrepancies you note.

testInfrastructure.zip (Test Methodology):
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:

- **test_sort.cpp**: This cpp file defines all the test functions that have been mentioned in the grading rubric above. It also runs these tests and outputs your final score.
- **test.py**: This file compiles test_sort.cpp in order to produce the executable. It then runs this executable which will indicate your score (as well as the test cases you have passed or failed).

Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.

How to test your program natively (run your own test cases):
You can use main.cpp provided in the starter.zip folder to test your sorting functions. You can then test the functionality of your methods by

calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests

How to use testInfrastructure:
To check your score using testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your sort.cpp file into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your sort.cpp file into the testInfrastructure directory. **Ensure that the makefile in this directory is the one given to you originally in the testInfrastructure.zip folder and not in the starter.zip folder.** In your terminal:
    - Run **make** in order to compile the program (which will compile test_sort.cpp and create an executable named ./test_sort).
    - Next, run **./test_sort** on your terminal to actually run the tests and output your score.