CSCI 2720

Project 5 – **Sorting Algorithms**

Due: Friday, April 30, 2021 @11:59:59 PM

For this project, you may choose to work individually or form a team of 2 with a fellow student.

You will need to implement insertion sort, selection sort, merge sort, heapsort and quicksort in C++ and write a short analysis report. Please note that not all sorting algorithms have been explicitly discussed in our class. You may need to look up and study some of these sorting algorithms.

For quicksort, you may freely decide how the pivot is chosen. However, you must specify your pivoting/partitioning strategy in your report. All of the sorting algorithms need to **keep track of the number of comparisons between the elements that are being sorted**. The sorting algorithms should sort a set of integers in **ascending order**.

Additional Requirements

1. The program must include a makefile with a "*compile*" directive that compiles your source files into an executable file called "**main**". The makefile should also have a "*clean*" directive that cleans compiled artifacts. Please note that no skeleton source files are provided for this project. Please create your own C++ source files from scratch.

2. When the program is executed (with input filename as a command line argument as shown on the next page), it should prompt the user for the sorting algorithm to be used.

3. After printing the results of a sorting algorithm to the screen, please exit the program. In other words, do not prompt the user for another sorting algorithm.

4. In each of the sorting functions, you should embed statements to keep track of the total number of comparisons between the elements that are being sorted (e.g., assuming there's an input array, then only count comparisons such as "array[i] < array[j]", but comparisons such as "i < length" in the condition of a loop do not count). You can use a counting variable of type long. After sorting is completed, you should print this count to the screen.

5. You will be given 3 input files for testing.
    a. ordered.txt – contains integers placed in ascending order from 0 to 9999
    b. random.txt – contains the 10000 integers arranged randomly
    c. reverse.txt – contains integers placed in reverse order from 9999 to 0

6. Write a short analysis report named "**report.txt**" in plain text format. Your report should include an explanation of how quicksort is implemented (pivoting/partitioning strategy) and answers for the following questions:
    a. Provide the total number of comparisons used by each sorting algorithm
        i. for ordered.txt as input.
        ii. for random.txt as input.
        iii. for reverse.txt as input.
    b. Did you use extra memory space or other data structures other than the input array? If so, explain where and why.
    c. Explain which sorting algorithms work the best in what situations based on your experimental results.

Note: Example output has **not shown** the complete list of sorted numbers. However, your program must print the complete list of numbers. The number of comparisons provided in the example output is **not accurate**. It is there just to show the expected output format.

Example Output:

```
./main ordered.txt
(i)insertion sort (s)selection sort (m)merge sort (h)heapsort (q)quicksort
Enter the algorithm: i
0 1 2 3 4  ..... 9999
#insertion sort comparisons: 12345

./main ordered.txt
(i)insertion sort (s)selection sort (m)merge sort (h)heapsort (q)quicksort
Enter the algorithm: m
0 1 2 3 4  ..... 9999
#merge sort comparisons: 12345

./main random.txt
(i)insertion sort (s)selection sort (m)merge sort (h)heapsort (q)quicksort
Enter the algorithm: h
0 1 2 3 4  ..... 9999
#heapsort comparisons: 12345

./main reversed.txt
(i)insertion sort (s)selection sort (m)merge sort (h)heapsort (q)quicksort
Enter the algorithm: q
0 1 2 3 4  ..... 9999
#quicksort comparisons: 12345
```

Grading Rubric

Projects that do not compile will receive a grade of 0.
See course syllabus for late submission evaluation.

Grade Item	**Grade**
insertion sort	15
selection sort	15
merge sort	15
heapsort	15
quicksort	15
report	20
readme, makefile & other misc. requirements	5
Total	**100**

Submission Instructions

Name your project directory **lastname_project5** (if working individually) or **lastname1_lastname2_project5** (if working in a team). Submit your project 5 directory to **csci-2720c** on odin.

For those working in a team, **only one team member should submit** on behalf of the team.

If you are off campus, you must use UGA VPN (remote.uga.edu) before you can ssh into *odin.cs.uga.edu* using your MyID and password.

Your project 5 directory should contain the following files:

1. Source files of your program
2. The 3 input files: ordered.txt, random.txt, and reverse.txt
3. The "report.txt" file containing your analysis report
4. The README file to tell us how to compile and execute your program. Include your name and UGA ID# (and your teammate's info if working in a team) on the top of this file. **Also describe the contributions of each team member if working in a team.**
5. The makefile