CSCI 2720

Assignment-5

Sorting Algorithms

Due: November 21, Sunday 11:59 PM

For this project, you will implement Selection sort, Merge sort, Heap sort and Quicksort in C++ and you will write a 1 to 3 page analysis report. **For quicksort, you should provide two implementations**. One implementation should use the first element of the array as the pivot. The second implementation should use a random pivot. All of your sorting algorithms need to keep track of the number of comparisons used by the sorting algorithm. **The sorting algorithms should sort a set of integers in the ascending order.**

Start early on this assignment as you need to submit a report along with your code. Note that most of the code is already provided in the slide and you are free to use that code (just cite it in your read me file). The only thing you need to do is to calculate the number of comparisons each algorithm is making (something similar to what was explained in the lecture). For comparisons, you need to count when you compare two 'data elements' not the comparison for 'i' in a for loop.

<u>Additional Requirements</u>

1. You can do this assignment by writing your own classes with required data members and methods. You have full freedom on how you want to build your classes.
2. The program must include a makefile that compiles your source files into an executable file called "main". The makefile should also have a clean directive to clean compiled artifacts. You will require a separate "Sorting.cpp" file for implementing all four sorting algorithms.
3. As the program is run (with input filename as a command line argument as shown below), it should ask for which sorting algorithm to be used for sorting. Sorted results should then be generated using that algorithm for sorting.
4. Unlike the previous assignments, after printing the results of a sort, exit the program automatically. In other words, <u>**do not**</u> ask to enter the algorithm again in a loop.
5. In each of the sorting functions, you should embed statements to count the total number of comparisons used by the function for sorting. You can use a type long count variable to count comparisons. After sorting is completed, you should print this count exactly as shown in the example output.
6. For this assignment you need to do two sets of experiments and prepare a report which will include results, plots and discussion from these two experiments. The first experiment is explained in points 7 and 8 where you just need to find out the number of comparisons for different sorting algorithms for different types of input files. The second experiment is explained in point 9 and 10 where you need to draw a plot between size vs no. of comparisons for different sorting algorithms and then verify that the plots obtained by your experiments match with the

theoretical result. You need to prepare a report summarizing all your results, plots, discussion and conclusion from these two experiments.

7. **You will be given 3 input files for testing the first experiment. Check eLC contents page for these files.**
    - **a.** Ordered file - Containing integers placed in ascending order from 0 to 9999
    - **b.** Random file – Containing 10000 integers arranged in a random fashion
    - **c.** Reversed file – Containing integers placed in reverse order from 9999 to 0

8. You can use the same code snippet for reading input from the files as in the previous assignments. Write a well-organized, 1 to 3 page report of your work (you can exceed this if you need to). Your report must be submitted as a PDF file with your name in the middle of the cover page. Submit your report on elc and your code on odin. Your report must include an explanation of the experiments you performed for each algorithm. In conclusion answer the following questions :

    a. Provide the total number of comparisons used by each one of the algorithms and explain whether they align with the Big O time complexity of each algorithm. In case of quick sort, compare and comment about the number of comparisons and complexity with the other quicksort implementation.

    i. For Ordered file as input.
    ii. For Random file as input.
    iii. For Reversed file as input.

    Please use the following table format to summarize the content for point a.

Algorithm	Input Type	# comparisons	Comments about time complexity and # comparisons
...	...	...	...

    After this table, also answer these questions below:

    b. Did you use extra memory space or other data structures other than the input array? If so, explain where and why?

    c. Explain what sorting algorithms work best in what situations based on your experimental results.

Here is a small example that shows what to expect when you run these algorithms on the input files. The size of input is 10,000 so for an algorithm with N^2 complexity N^2 is going to be 10000000 for such an algorithm. Expect 8-9 digits in the number of comparisons value. Similarly for an algorithm with NlogN complexity NlogN will be 40,000. So expect 5-6 digits in the number of comparisons that you will get. For an NlogN algorithm, if you are getting 8-9 digits in comparisons that will indicate a problem in your comparison calculation.

9. After completing the above program and report, you will do the next set of experiments where you will draw plots (n , input size vs no. of comparisons) for sorting algorithms and verify that the experimental results match with the theoretical results. You have complete freedom in how you want to implement this part of the assignment. You can create a new main, or add an function to your sorting.cpp class or add an function in the main.cpp that implements this part of the assignment.Your code should use the algorithms created in Sorting.cpp to find additional insights about the 5 different sorting algorithms. For this part of the assignment you are not reading inputs from the text files, but instead you will be generating the inputs within the program itself and it should calculate the comparison values. Your code should be able to run the five algorithms (mentioned below) with different sizes of the inputs and give the number of comparisons. You can use these comparison values to draw the plots. (This is for the grading purpose) provide an interface in your implementation where user can specify which sorting algorithm they want to use and the size of the input that they wish to test. Your program should then generate an array with n number of **random values** (with n being the number specified by the user) and then it should sort those values using the sorting algorithm selected by the user. The number of comparisons should also be printed. **Please specify in your README file how to compile, run and use your implementation because it is mostly up to you for how you want to implement this part of the assignment. Also note there is no sample output for this part of the assignment.**

10. You are now going to use implementation in point 9 to create multiple different plots for the report mentioned above. Keep track of your experiments in a table formatted like the following and include this on your report:

Size of input

Algorithm	10	100	500	1000	10000	20000	50000	100000
Selection								
Merge								
Heap								
QuickSort-fp								
QuickSort-rp								

Note: **I recommend running each algorithm multiple times and taking the average result for each input size to use in this table and for your plots. You can also use more values of n to make your plot look smoother. However, in the report you just need to show the results for the above table.**

a. You should use some sort of graphing software like Microsoft Excel or Google Sheets to generate plots that compare the size of input n to the number of comparisons for the 5 sorting algorithms. You should have 5 separate plots(one for each sorting algorithm) with n on the x-axis and number of comparisons on the y-axis. Include these plots in your report.

b. You will then provide some discussion about your results. Compare the theoretical result with your experimental result for each algorithm. Does your experimental result coincide with the Big-O of that specific algorithm? Describe why there may be some inconsistencies between your plot and what the theoretical plot looks like.

Your report must be in PDF format. Submit the report on the submission link provided on the eLC under the assignment section.

Note: Example output has not shown the complete list of numbers. However, you must print the complete list of numbers in your program. The number of comparisons shown in sample output below may be slightly different than the numbers that you get, however make sure that you at least get the same number of digits so you know that it is in the correct big-O.

quick-sort-fp stands for quick sort with first element as the pivot. quick-sort-rp stands for quick sort with random element as the pivot.

Sample Output 1 (ordered.txt):

```
./main ordered.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: s
1 2 3 4 5 ………….. 9999
     #Selection-sort comparisons:  49995000

./main ordered.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: m
1 2 3 4 5 ……………. 9999
        #Merge-sort comparisons: 69008

./main ordered.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: h
1 2 3 4 5 ……………. 9999
```

```
        #Heap-sort comparisons: 244576

./main ordered.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: q
1 2 3 4 5 ……………. 9999
        #Quick-sort-fp comparisons: 49995000

./main ordered.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: r
1 2 3 4 5 ……………. 9999
        #Quick-sort-rp comparisons: 161020
```

Sample Output 2 (random.txt):

```
./main random.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: s
1 2 3 4 5 ……………. 9999
        #Selection-sort comparisons: 49995000

./main random.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: m
1 2 3 4 5 ……………. 9999
        #Merge-sort comparisons: 120414

./main random.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: h
1 2 3 4 5 ……………. 9999
        #Heap-sort comparisons: 235440

./main random.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: q
```

```
1 2 3 4 5 ……………. 9999
        #Quick-sort-fp comparisons: 159534

./main random.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: r
1 2 3 4 5 ……………. 9999
        #Quick-sort-rp comparisons: 167559
```

Sample Output 3 (reverse.txt):

```
./main reversed.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: s
1 2 3 4 5 ……………. 9999
        #Selection-sort comparison: 49995000

./main reversed.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: m
1 2 3 4 5 ……………. 9999
        #Merge-sort comparison: 64608

./main reversed.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: h
1 2 3 4 5 ……………. 9999
        #Heap-sort comparison: 226720

./main reversed.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: q
1 2 3 4 5 ……………. 9999
        #Quick-sort-fp comparison: 49995000

./main reversed.txt
selection-sort (s)   merge-sort (m)  heap-sort (h)   quick-sort-fp
(q)  quick-sort-rp (r)
Enter the algorithm: r
```

```
1 2 3 4 5 ……………. 9999
        #Quick-sort-rp comparison: 165518
```

Note: Make sure to quit the program after printing the number of comparisons after each sort. In other words, do not ask to enter the algorithm again in a loop.

Grading Rubric

Grade Item	**Grade**
Selection Sort	10%
Merge Sort	10%
Heap Sort	10%
Quick Sort	10%
Report	50%
Readme, Comments, makefile & Spec conformity	10%
Total	**100%**

Compiling the Program (Same as previous assignments):

A Makefile should compile your code into a program called "main".

Your program should run with the following command syntax:

```
./main <input file name>
```

Commands to run and compile the code should be documented in the Readme File clearly.

Code that fails to compile or the code that compiles but fails to run will receive a grade of a zero.

Late Submission Policy:

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Students unable to submit a project due to a serious illness or other emergency should contact the instructor as soon as possible before a project's deadline. Based upon the circumstances, the instructor will decide an appropriate course of action.

Submission Notes:

You need to submit all your code to odin like in the previous assignments. But you should submit the report on eLC under the assignment section:

- **Report in PDF format (submit on eLC).**
- **Your program**
- **Makefile**
- **3 input files**
- **Readme file with all necessary descriptions.**

Odin Login Instructions

This semester we are switching from Nike to a new server called Odin. Unlike with Nike, to use Odin, you must use the UGA VPN first. Please visit remote.uga.edu if you haven't already and install the VPN client for your OS. Be sure to have an SSH client like PuTTY or MobaXterm and use that to connect. The server's name is odin.cs.uga.edu and you will log in with your MyID credentials. For those that have used Nike before, the login process is almost the same, except that nike.cs.uga.edu is replaced by odin.cs.uga.edu, and, as stated before, you'll log in with your MyID credentials instead of your regular Nike credentials.

Group Submissions:

****Note: Assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a readme file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the readme file.**

If you are working in a group of two, your submission folder must be named as below:

```
LastName1_LastName2_assignment5
```

When you are submitting the folder you should use the submit command as below:

```
$ submit LastName1_LastName2_assignment5 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

If you are working in a group, you must **submit the assignment from the Odin account of only one of the members** in the group.

Individual Submissions:

If you are working individually, your submission folder must be named as below:

```
LastName_assignment5
```

Submission command should be used as below:

```
$ submit LastName_assignment5 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.

Note: Make sure you submit your assignment to the right folder to avoid grading delays.

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

You can check your current working directory by typing:

```
$ pwd
```

For example if your assignment is in the following directory,

```
/documents/csci2720/assignments/LastName_assignment5
```

Before executing the submit command, change to the assignment's immediate parent directory:

```
$ cd /documents/csci2720/assignments/
```

Then execute the submit command:

```
$ submit LastName_assignment5 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

To check if your submission command was valid, there should be a receipt file.

```
$ ls LastName_assignment5
// should return rec##### like 'rec54321'
```

Selecting the Odin Submission Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Destination
CRN 17504	M 10:20 AM	csci-2720a
CRN 42634	M 04:10 PM	csci-2720c