CPI 220-FALL 2018 ASSIGNMENT -2 POINTS -100

What is this homework about?

We have seen five sorting algorithms in the lecture slides. In this homework, we try to develop a custom sorting method (let's call it customSort), that sorts a stack of Integers. More specifically, you will develop the algorithm using the two main operations of the stack, which are push and pop. Once the algorithm is implemented, you will analyze it's running time by sorting arrays of various lengths (10, 100, 1000, 10000, 100000) given as input files. In the homework1, you were given code templates to fill-in. For this homework, you will need to implement all the code (along with the client) by yourself. Parallelly, implement quicksort and compare it's running time with the running time of your new sorting algorithm.

Implementation…

You should implement the following classes with specifications as follows,

1. quicksort: should have the following methods (for description follow lecture slides)
    a. void sort(Comparable[] a, int low, int high)
    b. void sort(Comparable[] a)
    c. void exch(Comparable[] a, int i, int j)
    d. boolean less(Comparable a, Comparable b)
    e. int partition(Comparable[] a, int low, int high)
2. stack (fixed size array implementation) (for description follow lecture slides)
    a. boolean isEmpty()
    b. void push(Item it)
    c. Item pop()
3. customSort (follow either your own logic or the logic given below to implement this)
    // To do
4. Client
    a. write a method to read the input files into an array.
    b. Write a method to store the sorted arrays into an output file.
    c. Write a method to know if the contents of two files (eg. output file and expectedOutput file) are exactly same
    d. Write a method to compute and print the time taken by a sorting algorithm in nano second.
    e. Write a method to populate a stack using the contents of the input files.
    f. Write a method to store the elements of the stack after sorting into an output file.
    g. In the main method, use the above methods and
        i. Test both your sorting methods for fixed array A = {9, 9, 10, 3, 6, 2, 1, 1, 7, 2}; For testing the customSort implementation, you should save the array as a stack using the above implemented method.
        ii. Test both algorithms for the arrays formed by taking the input from the files, arrayLength_10.txt …. arrayLength_100000.txt You can test as

follows, first read the input file into an array (or stack), sort it and store in a file called “output_100.txt”, compare the content of this output file with that of the expected_output files. You only need to test this for arrays of lengths 100 and 1000 (i.e input files arrayLength_100.txt and arrayLength_1000.txt).

iii. Write code to print the compute time for these two sorting algorithms for arrays various lengths (compute the times for arrays of all lengths).

Logic for the customSort (look at the demo first and then read the description)…

Maintain another stack and call it “helperStack”. Initially keep it empty, keep popping elements from the original stack and place it on the helperStack. We need to maintain the helperStack always sorted. To do this, pop elements from the original stack and push them on to the helperStack. If pushing an item (let’s call this item *it*) violates the “always sorted” condition of the helperStack, pop the elements from the helperStack and push them on to the original stack. Push until the top most element of the original stack is smaller than *it.* Now push *it* on to the helperStack and repeat. Once the whole original stack is exhausted, you will have the sorted stack in the helperStack.

A demonstrative video is given below,

https://www.youtube.com/watch?v=Jhaf7G_sO3k

To Start With…

- First implement the file access methods (file read method, write method and compare method). These are independent of other parts of the project, so this is a good place to start.
- Now implement the quickSort class.
- Now implement the main method of the client file and test the quickSort method implemented.
- You can now implement the method used to compute time and test it.
- Finally implement the customSort class, and the corresponding test statements to test this class in the main method of the client class.

Grading…

- Code with all the required classes and methods implemented without errors – 20 points
- Quicksort working as expected – 10 points
- customSort working as expected – 20 points
- file read, write compare methods working correctly – 20 points

- time method working well – 10 points
- Running time analysis document (see deliverables) – 20 points

Deliverables…

- Deliver the class files, the input and output files, and the analysis.pdf in a zipped folder named, LASTNAME_FIRSTNAME_CLASSID.zip.
- The analysis document should contain the following parts.
    - Plot a graph for the running time of QuickSort and CustomSort for arrays of various lengths.
    - How do you compare the times taken by the quickSort and the customSort, which one is better, what is the order of running time of the custom Sort (use empirical analysis)?

Plagiarism Policies…

- This work should be individual, so do not collaborate with others.
- You are allowed to discuss your program logic with your classmates, however you are not allowed to copy any code parts from anyone.
- The only place where you can copy the code from is the lecture slides. This code is officially uploaded by the instructor and so feel free to use bits and parts of it for your homework.
- You are not allowed to copy the code from the internet, however you can search on the internet to figure out the logic for the program.
- In case of doubt, email the instructor (vijetha.gattupalli@asu.edu)

Late Submission Policies…

- You will be imposed a penalty of **10% late submission per day for the first 5 days after the deadline.** No submissions will be accepted after this late submission deadline.
- The original deadline for this homework is **16th October 11:59 PM**. With the late submission policy, the **late submission deadline is 21th October 11:59 PM.**
- Email me in advance (if there is a valid reason) if you want me to extend the deadline for you without a penalty.