Assignment 2: Analysis of Algorithms, Sorting

In this assignment, you are forbidden from using standard APIs that would otherwise implement **data structures or sort functions or etc**. Any data structure used must be programmed on your own and submitted with the assignment. **Do not use java.util.Collections or the C++ STL or etc. unless otherwise stated.**

You will push all of your solutions to the Assignment 2 repository through the Github Classroom. Organize your repository in such a way that it contains:

Root – Assignment 1 directory

- Problem 1 directory
  - Problem 1 code…
  - Problem 1 PDF…
- Problem 2 directory
  - Problem 2 code…
  - Problem 2 PDF…
- …etc. for all numbered problems in this assignment set.

Note: We have removed the requirement for a main file.

Note: Do not include binaries / executables / bin folders / etc. Only upload the .java / .cpp / .py / .h / etc only.

Note: You are graded on directory structure.

There is no restriction on the programming language that you use.

For the problems, you will be implementing your code as an API through which the graders will make function calls.

Include a README.md file for the GitHub repository that has your Name as the title and a description of what is inside the repository, generally.

!! INVITATION LINK: https://classroom.github.com/a/q8wGWa8u !!

For anyone new to Github, we encourage you to use **Github Desktop** as a simple GUI for interacting: https://help.github.com/desktop/guides/getting-started-with-github-desktop/

For people who want to be more advanced and up to industry standards, you can use the Git command line:

https://git-scm.com/downloads

A very barebones introduction to the command line (very readable, I recommend it):

http://rogerdudler.github.io/git-guide/

1. **Sorting - 1**
    a. In the lecture, we found that our compare-based sorting algorithms Mergesort and Quicksort are often bounded by $O(Nlog_2(N))$, where N is the size of the input array. In this problem, we want you to implement a sorting algorithm that does not compare elements in an array to each other. Instead, group digits by their order in a numeric sequence to sort them. Take for example: [12 9001 5]. Continually re-order these by digit sequence until they end up sorted. Preserve the list order when the digits of multiple entries are equal. Example below: (note that your algorithm does not have to work like this exactly, but needs to avoid **comparing elements of the array to each other** and must sort by ranking digit sequences):
        i. [12 9001 5] --> Initial input
        ii. [900**1** 1**2** **5**] → Sorted by 1,2,5
        iii. [90**0**1 5 **1**2] -> Sorted by 0,0,1
        iv. [9**0**01 5 12] -> Sorted by (0,0,0)
        v. [5 12 **9**001] -> Sorted by (0,0,9)
    b. For this problem, **include only a main file that contains the function for this sort.** The sort will be called Problem1Sort, and is called with Problem1Sort(int[] a, int arraySize).
        i. The first parameter int[] a is an array to be sorted.
        ii. The second parameter, int arraySize, is the size of the array int[] a.
    c. In a separate PDF, give an analysis of the run-time of your algorithm. You may show the proof however you like, either through explanation or mathematics. <u>If you are not using math, your explanation should be convincing and short, with exact pointers to the cost of operations</u>. **<u>You need to include the exact code</u>** and highlight where the bottleneck is – i.e. which operation is taking the longest in the sort and what is its Big-Oh notational runtime? **PDF requires the code and the explanation**.
    d. Is your sort stable? Why? Give this answer in the PDF and show where it remains stable in your code snippet.
    e. Update your sorting algorithm to use constant extra space. Highlight this change in the code snippet and show why it uses constant extra space compared to the old one.
        i. Depending on whether your algorithm uses extra space, answer the below depending on your implementation:
            1. If your algorithm already uses constant extra space, what **major** change to the algorithm would using O(n) extra space provide?
            2. If your algorithm did not use constant extra space, what **major** change to the algorithm does using O(1) extra space provide?
2. **Sorting -2**
    a. Implement an in-place merge sort algorithm that does not use O(N) extra space to merge the two halves. Your requirements:
        i. For this problem, **include only a main file that contains the function for this sort.** The sort will be called mergeSort, and is called with mergeSort(int[] a, int arraySize).
            1. The first parameter int[] a is an array to be sorted.
            2. The second parameter, int arraySize, is the size of the array int[] a.

b. Produce a code snippet in a new PDF, Problem 2.pdf. Write an explanation (or mathematically prove) the complexity of your in-place mergesort for the best and worst case. Highlight, in the code, which operations cause the bottleneck for in-place mergesort in the best and worst case.
    i. Is the sort stable? Highlight in the code segment and explain why.

3. Sorting – 3

a. Implement 3-way Quicksort for a linked list. Define these methods for the class:
    i. LinkedList(); -- constructor
    ii. add(int a); -- Adds a number to the linked list
    iii. printList(); -- Prints the current list contents to console as comma separated values ("1,2,3"
    iv. quicksort(); -- Runs 3-way quicksort.
b. Your requirements for quicksort:
    i. It must shuffle the linked list. **Note: there is no restriction on how you shuffle the list. It is up to you.** We will test runtime for different insertions to validate the shuffle.
    ii. It must print out every comparison and what is being compared.
    iii. It should print out every swap.
    iv. It should print out the current list after finishing a partition (pivot is placed in final location).
    v. It should initially use median of 3 selection for the pivot in the first partition.
    vi. For this problem, **include only the class files / code files. No main function file is needed.**

4. Problem Solving - In this problem set, include a single main file that contains functions that can be called from a main function. For this problem, you are allowed to use external sort functions and data structures from the STL or Java.util.etc.

a. Assume you have an array of size N containing integers. Your client asks you to order the array in terms of local maxima and local minima. Example: {a b c d e …}, a>=b, b=<a & b<=c, c>=d, d=<c & d=<e, e>=… etc. Program an algorithm that orders an array of random numbers into a set of maxima and minima.
    i. Example: [8 9 0 3 1]
    ii. Your function should be called with Problem4A(int[] a, int arraySize) from the main file.
        1. The first parameter int[] a is an array to be sorted.
        2. The second parameter, int arraySize, is the size of the array int[] a.
    iii. Include a PDF: Problem 4.pdf. Highlight the code snippet for your algorithm. What is the overall complexity? Give an explanation or a proof. Explanation requires references to the code with highlights of specific lines.
    iv. Is there an linear time solution for this problem? Why or why not? Include your answer in Problem 4.pdf.