Lab 05 Specification – More Exercise to practice implementing Recursive Sorting algorithms and Tree Data Structure.
50 points

Due by: 04/22/2021 2:00 PM (Individual/Team-based)

Lab Goals

- Practice analysis and implementing Recursive Sorting algorithms in a programming language of your choice (Python or Java).
- Think about how to solve the sorting problem by applying different approaches.
- Practicing working on Tree Data Structure using an Array or a List.

Learning Assignment

If not done previously, it is strongly recommended to read all of the relevant ”GitHub Guides”, available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading assignment is useful to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, it is also recommended to do the reading assignment from the section of the course textbook outlined below:

- **Sedgewick Chapter 02, 2.2, 2.3, 2.4**

Assignment Details

Now that we have discussed some basics of recursive sorting algorithms (Quick and MergeSort) and analyzed them together in the last few lectures, it is now time to do it practically. In this lab, students will practice developing a variety of algorithms to retain the knowledge from the class discussions so far. This also includes developing one or more code files to implement a variation of MergeSort algorithm, implementing recursive procedures using a programming language such as Java or Python. At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials.

The Professor proofread the document more than once, if there is an error in the document, it will be much appreciated if the student(s) can communicate that to the Professor. The class will be then informed as soon as possible regarding the error in the document. Additionally, it is highly recommended that students will reach out to the Professor in advance of the lab submission with any questions. Waiting till the last minute will minimize the student chances to get proper assistance from the Professor and the Technical Leader(s).

This lab can be completed either individually or as a team-based work. Each team can have a maximum of 3 members. Each member of the class has a choice to continue working with their same team from the previous lab or join a new team or change to an individual submission. Please make sure to communicate to the Professor if you like any changes to be done from your submission for the last lab. Those who had not communicated any changes, I expect you to follow the submit the work the same way as lab-4. It is expected that all members of the team are contributing equally to the completion of the lab. The contribution from every group member is vital to maximizing your performance in this course.

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done by the member(s) of the team. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details outside their team. It is acceptable to have a healthy discussion with other teams. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other team(s) program(s) and algorithm(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually/team, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc $\cdots$

Part 1 - MergeSort Algorithm (30 points)

Let us suppose that the patient details such as patient id's are given and the requirement is to sort the patients in ascending order based on their id's in ascending order.

1. A starter code in the files named `ms.py` and `MS.java` is provided in the repository. The starter code contains the implementation to generate the input dataset using random sampling. It is **not complete** and as it stands doesn't sort the input dataset.
2. Complete the implementation of `main_sort` and `partition` methods. Once these two methods are completed, the starter code should be able to display the original unsorted data and the output sorted data on the console.

PS next page ...

3. The algorithms for MergeSort and merge procedure are provided below:

Algorithm - Merge(A, p, m, r)

Input: an n-element un-sorted array A of integer values, a lower bound p of the array A, a midpoint m, and a pivot r in the array A.

Output: an n-element merged array A of integer values.

```
$n_1 \leftarrow m - p$
$n_2 \leftarrow r - m$
Initialize Two Arrays L of size $n_1 + 1$ and R of size $n_2 + 1$
**for** i = 0 to $n_1$ **do**
    L[i] $\leftarrow$ A[p+i]
**end for**
**for** j = 0 to $n_2$ **do**
    R[j] $\leftarrow$ A[m+j]
**end for**
$L[n_1 + 1] \leftarrow \infty$ and $R[n_2 + 1] \leftarrow \infty$
Initialize $i, j \leftarrow 0$
**for** k = p to r **do**
    **if** L[i] $\leq$ R[j] **then**
        A[k] $\leftarrow$ L[i]
        i $\leftarrow$ i+1
    **else**
        A[k] $\leftarrow$ R[j]
        j $\leftarrow$ j+1
    **end if**
**end for**
```

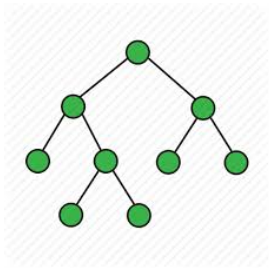
Algorithm - MergeSort(A, p, r)

Input: an n-element un-sorted array A of integer values, a lower bound p of the array A, and a pivot r in the array A.

Output: an n-element sorted array A of integer values.

```
**if** p $<$ r **then**
    $m \leftarrow$ Floor $(p + r)/2$
    MergeSort(A, p, m)
    MergeSort(A, m+1, r)
    Merge(A, p, m, r)
**end if**
```

Part 02 - A Simple Exercise to develop a series of features in a Tree data structure. (20 points)



We had seen in detail the different terminologies connected to the tree data structure and binary trees. Complete the requirements outlined below:

1. For simplicity, a starter-code in `trees.py` `Trees.java` are provided in the lab repository.
2. Analyze the code provided. This code generates an array of numbers and prompts the user to pick a node to explore.
3. Develop the methods called `getChildren and getParent` based on in-class discussion. This is the starting point for all other implementations in this section. It is required to have a solid understanding of how the parent and children are identified. There is some hint in the last page. But, look at only after trying to figure the formula yourself.
4. Develop a new method called `getAncestors`. This method should accept an array/list and the node to be explored as the inputs. Add the required logic to output all the ancestors. You may implement a recursive solution by creating a recursiveParent method by using a similar logic that you implemented in the getParent method. Alternatively, you may think through and implement the same functionality using a mathematical formula. This will be little bit more challenging than the recursive solution.
5. Develop a new method called `getDescendents`. This method should accept an array/list and the node to be explored as the inputs. Add the required logic to output all the descendents. You may implement a recursive solution by creating a recursiveChildren method by using a similar logic that you implemented in the getChildren method. Alternatively, you may think through and implement the same functionality using a mathematical formula. This will be little bit more challenging than the recursive solution.
6. Develop a new method called `findDepth`. This method should accept an array/list and the node to be explored as the inputs. Add the required logic to output the depth of the node in the context of the tree.
7. Develop a new method called `findHeight`. This method should accept an array/list and the node to be explored as the inputs. Add the required logic to output the height of the node in the context of the tree.
8. A simple validation should be implemented. If there are no ancestors or descendents, this should be identified and displayed on the console. It is required to do a slight extension to the starter-code to implement this part.
9. Please make sure to read all the comments in the starter-code so as to make sure to understand the requirements fully.
10. **Optional:** Develop a new method called `isBalanced` to identify if a given tree is balanced or unbalanced. Add the required logic to output the status of the tree. True for balanced and False for unbalanced.

Part 03 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. Part 1: “ms.py’ or “MS.java’file.
2. Part 2: “trees.py’ or “Trees.java’file.
3. A signed honor code file, named `honor-code.txt`.
4. To reiterate, it is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits may be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your GitHub repository will lead to receiving no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student’s submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If a student needs any clarification on their lab grade, it is strongly recommended to talk to the Professor. The lab grade may be changed if deemed appropriate.

HINT

- (2 * node) + 1 = left child
- (2 * node) + 2 = right child
- parent = Math.ceil(node/2) - 1