Lab 04 Specification – A Hand-on Exercise to practice implementing and analysis Recursive Sorting algorithms
50 points

Due by: 04/15/2021 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.
- Analyze sorting algorithms by taking a deeper look at the comparisons and swap procedures.

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**

Assignment Details

Now that we have discussed some basics of recursive sorting algorithms (Quick and Merge sort) 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 sorting algorithms, 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 proof-read the document more than once, if there is an error in the document, it will be much appreciated if 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 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 last lab. Those who had not communicated any changes, I expect you to follow the submit the work the same way as lab-3. 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 maximize your individual 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 01 - Quick Sort Algorithm (20 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 `qs.py` and `QS.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 don't have the capability to 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.
3. The algorithms for quick sort and partition procedure are provided below:

   Algorithm - Partition(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.

   $x \leftarrow A[r]$
   $i \leftarrow p - 1$
   for j = p to r-1 **do**
   if A[j] $\leq$ x **then**
   $i \leftarrow i + 1$
   swap A[i] and A[j]
   end if
   end for
   swap A[i+1] and A[r]
   return i+1

Algorithm - QuickSort(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**
  $q \leftarrow$ Partition(A,p,r)
  QuickSort(A, p, q-1)
  QuickSort(A, q+1, r)
end if

Part 02 - Randomized Quick Sort Algorithm (20 points)

A variation of the Quick Sort is the Randomized Quick Sort. A huge performance metric of the Quick Sort algorithm is the pivot selection. If the pivot is fixed to the last element as in the original Quick Sort, then this may lead to a **0 : n-1** split, which then leads to a O(n^2) run time in the worst case. A randomized quick sort may take this equation out of the picture. The general idea is to simply randomize the pivot selection. The Randomized Quick Sort algorithm and the pseudocode is provided on the next page for easy access.

Characteristics of Randomized Quick Sort:

- Assume all elements are distinct.
- Partition around a random element.
- Consequently, all splits (1: n-1, 2: n-2, ... , n-1: n)
- Randomization is a general tool to improve algorithms with bad worst-case but good average case complexity.

Algorithm - Randomized Partition(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.

$i \leftarrow Random(p, r)$
swap A[r] and A[i]
return Partition(A, p, r)

Note: Partition algorithm is provided above.

Algorithm - Randomized QuickSort(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
   q ← Randomized-Partition(A,p,r)
   Randomized-QuickSort(A, p, q-1)
   Randomized-QuickSort(A, q+1, r)
end if
```

Expected Running Time:

- Randomizing pivot selection
- Splits are going to be better than 0: n-1
- Take average for running time using different pivot selection.
  $O(n \times log(n))$

After thinking through the pseudocode provided above, implement the requirements outlined below:

1. First, read through the high-level outline of the randomized partition and randomized quick sort algorithm provided above. The challenge here is to think through a given algorithm and traslate it to implementation. A huge learning objective here is to find out the limitations of Quick Sort through practical activity. It is highly recommended to discuss and brainstorm ideas with your peers, the TL(s), and the Professor on how to implement this.
2. Implement the algorithm proposed above using Java or Python programming language. For simplicity, a starter code-named `RQS.java` and `RQS.py` is provided. The input of the program is automatically generated within the starter-code using a randomized sequencing. The output of the program should be generated using the ascending order format. Make necessary code modifications to this starter-code file. Read through the comments in the code file for additional information on the structure of the code.

Part 03 - Quick Sort Analysis. (5 points)

Answer the following question to record your understanding of the basic ideology of Quick Sort in the file named `quick-analysis.md`. Let us suppose that there are n elements in the un-sorted array. Answer the following:

- q1: What is the worst, best, and average case running time of the Quick Sort algorithm.
- q2: What is the base case for Quick Sort to break?
- q3: What is the recursive case for Quick Sort?
- q4: How many recursive calls are made at each step in QS main algorithm?
- q5: Does QS partition always create equal splits?

Part 04 - Merge Sort Analysis (5 points)

Answer the following question to record your understanding of the basic ideology of Merge Sort in the file named `merge-analysis.md`. Let us suppose that there are n elements in the un-sorted array. Answer the following:

- q1: How is merge sort different from quick sort?
- q2: What is the split ratio in merge sort?
- q3: What is the worst-case/average-case/best-case running time of Merge Sort?
- q4: Why is the worst case running time of Merge sort O(n log n) always?
- q5: Why does Merge Sort use a static tree in the recursion process? It is worth noting that the Quick Sort use a dynamic tree.

Part 05 - 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

PS next page ...

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: "josephus.tex' and "josephus.pdf' files.
2. Part 2: "StackSorting.java' or "stack-sorting.py', and "sorting.tex', "sorting.pdf' files.
3. Part 3: "Hop.java' or "hop.py' files.
4. A signed honor code file, named `honor-code.txt`.
5. 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.