Lab 03 Specification – A Hand-on Exercise to practice Algorithm Implementation
50 points

Due by: 03/25/2021 2:00 PM (Individual/Team-based)

Lab Goals

- Practice integrating Stack and Queue ADT 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.
- Implement different sorting algorithms using a programming language such as Java or Python.

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 01, section 1.3 Stacks, Queues, Chapter 02, 2.1**

Assignment Details

Now that we have discussed some basics of sorting algorithms 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, integrating Stack and Queue ADT 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).

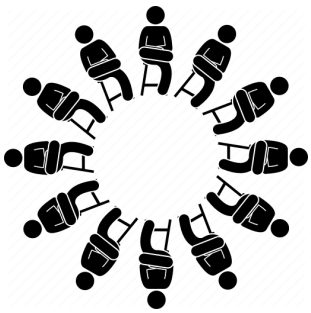
This lab can be completed either individually or as a team-based work. Each team can have a maximum of 3 members. Please make sure to complete the team-list document (shared in Slack) to provide the general details about your team. If you are completing the work individually, then you don't need to fill out the form. 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 - A Variation to Josephus Problem (15 points)



The Josephus problem and a solution to the problem using Queue ADT was discussed in class in detail in lesson 4.

The **original** Josephus problem is defined below: A group of n people are standing in a circle, numbered consecutively clockwise from 1 to n. Starting with person no. 2, we remove every other person, proceeding clockwise. For example, if n = 6, the people are removed in the order 2, 4, 6, 3, 1, and the last person remaining is no. 5. Let $j(n)$ denote the last person remaining. Find some simple way to compute $j(n)$ for any positive integer $n > 1$.

A variation of the **original** Josephus problem is defined below: A group of n people are standing in a circle, numbered consecutively clockwise from 1 to n. Starting with person no. m, we remove the person located at the next m^{th} position in the circle, proceeding clockwise. For example, if m = 3 and n = 8, the people are removed in the order 3, 6, 1, 5, 2, 8, 4 and the last person remaining is no. 7. Let $j(n)$ denote the last person remaining. Find some simple way to compute $j(n)$ for any positive integer $m > 1$ and $n > 1$.

It is important to learn how to understand a problem definition? To have a good understanding of the problem is the starting point to formulate the solution to the problem. So start thinking towards understanding this variation to Josephus problem.

1. For simplicity, a starter code with a file named `josephus.tex` is provided in the lab repository. The starter code has the original formal algorithm discussed and practiced during class discussions.
2. A latex file may be developed using the Overleaf website link provided below:

   `https://www.overleaf.com/login`
3. Please note: The Overleaf website may prompt to register and log in before providing the options to compile latex files and generate pdf files. This should be a straight forward process. If there are any questions, students are encouraged to ask the Technical Leader(s) and the Professor.
4. Make necessary modification(s) to the file named `josephus.tex` to include the steps to compute the survivor in the Josephus circle. The computation should include step by step integration of the Queue ADT to the solution.

5. The algorithm should be developed in a formal manner using a similar style as we did in the class examples. Please refer back to the lecture slides and your class notes to look at the examples discussed in class.
6. It may be too tempting to try to propose a solution that is of linear asymptotic running time. However, it is acceptable to submit a solution that is in-between linear and quadratic solution.
7. **Hint: m** is the key for the running time. What is the running time if **m** is small? and what is the running time if **m** is extremely large and approaching **n**?
8. It is required as part of this lab submission to compile the latex files and generate a PDF version of the file. The PDF file should be named as `josephus.pdf` and uploaded to the repository. Both the **tex** and **pdf** files will be used for grading purposes.

Part 02 - Sorting using Stack (20 points)

In our last class, we discussed an elementary sorting algorithm known as Insertion sort. In this part, let us solve the sorting problem using a Stack data structure. Let us suppose that you are given with a list of un-sorted weight(s) of different athletes stored in a primary Stack data structure. The requirement is to sort the weights stored in the Stack in an ascending order. Assume that we are also provided with access to a temporary Stack data structures. The access to both Stack ADTs is restricted to only the following supporting operations:

1. push
2. pop
3. isEmpty()
4. peek

Start thinking towards understanding this variation of the sorting problem.

1. For simplicity, a starter code with a file named `sorting.tex` is provided in the lab repository. To give an example, the starter code has the original formal insertion sorting algorithm discussed and practiced during class discussions.
2. A latex file may be developed using the Overleaf website link provided below:

   `https://www.overleaf.com/login`
3. Please note: The Overleaf website may prompt to register and log in before providing the options to compile latex files and generate pdf files. This should be a straight forward process. If there are any questions, students are encouraged to ask the Technical Leader(s) and the Professor.
4. Make necessary modification(s) to the file named `sorting.tex` to include the steps to formulate your algorithmic solution to the sorting problem.

5. The algorithm should be developed in a formal manner using a similar style as we did in the class examples. Please refer back to the lecture slides and your class notes to look at the examples of sorting algorithms discussed in class.

6. Analyze the algorithm proposed above and add a summary file named `sorting-analysis`. This file can be PDF or markdown format. In this file, analyze the running time of the algorithm and provide an asymptotic upper bound for the running time.

7. Implement the algorithm proposed above using Java or Python programming language. For simplicity, a starter code-named `StackSorting.java` and `stack-sorting.py` are 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.

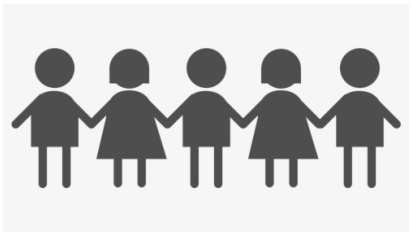
8. A sample code file named `StackUsage.java` and `stack-usage.java` is provided in the repository. The sample code may be used as a tool to learn how to use the built-in Stack ADT in Java and Python respectively.

9. For more details on the requirements for this program, please look at the comments in the source code file.

10. It is required as part of this lab submission for students to compile the latex files and generate a PDF version of the file. The PDF file should be named as `sorting.pdf` and uploaded to the repository. Both the **tex** and **pdf** files will be used for grading purposes.

11. Both the algorithm and the Java/Python implementation is worth 10 points each.

Part 03 - An implementation of the Hot Potato Problem. (15 points)



Write a Java or Python program that implements a `Hot Potato` program using a series of requirements outlined below.

1. The problem definition is provided below:

   Children line up in a circle and pass an item from neighbor to neighbor as fast as they can. At a certain point in the game, the action is stopped and the child who has the item (the potato) is removed from the circle. Play continues until only one child is left. How do you find the winning position for a child, if there are N number of children?

   Approach: We discussed this in detail during our lecture through the classic Josephus problem. You should take a similar approach to implement this simulator.

2. Implement the algorithm step by step and the program should output the array S.

3. The starter code file is provided inside the lab repository in a file named, `HOP.java` and `hop.py` are provided. Here HOP refers to the Hot Potato problem.

4. A sample code file named `QueueUsage.java` and `queue-usage.py` is provided in the repository. The sample code may be used as a tool to learn how to use the built-in Queue ADT in Java and Python respectively.

5. Implement the **findWinner** method using the algorithm that was designed in Part-1. The fine-level details on how to implement the method are purposefully left out so as to challenge the students to think about the implementation on their own.

6. For more details on the requirements for this program, please look at the comments in the source code file.

7. To validate the correctness of the output of the program, please pick an example from the slides and/or your notes and validate the correctness of the program. Please note, in the Josephus example discussed in slides, m = 2.

Part 04 - 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.