Lab 02 Specification – A Hand-on Exercise to integrate Data Structures into Algorithm Implementation.
50 points

Lab Goals

- Refresh your ability to implement ADTs and Data Structures.
- Think about how to solve algorithmic challenges with the use of ADT's.
- Integrate Data Structures to algorithm implementation using a programming language such as Java.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

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, Stacks and Queues**

Assignment Details

Now that we have discussed some basics of algorithms and analyzed algorithms with data structures 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 series of algorithms using a programming language such as Java.

The lab has two parts. The first part is required to be completed by the end of the lab session. The second part is required to be completed within a one week time period. Please make sure to push your code file(s) and commit by the deadline provided in each of the following section(s).

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. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.

2. Submitting a copy of the other's program(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, students maximize the learning and increase the chances to do well in other assessments such as skill tests, exams, etc.

At any duration during and/or after the lab, students are recommended to team up with the Professor and the Technical Leader(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).

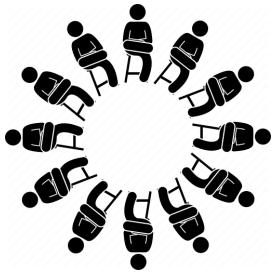
Section 1: In-Lab Exercise (25 points)

Attendance is in general a required component in computer science class(es) and laboratory session(s). By attending these sessions regularly, students get an opportunity to interact with the Professor and the Technical Leaders(s). This interaction helps maximize the student's learning experience. Attendance is taken in the form of Mastery Quizze(s) and by checking the commit log of the student's In-Lab exercise submission. No student is allowed to take the Mastery Quizze(s) outside the laboratory room unless prior arrangements are done with the Professor. It is against the class honor code to violate the policy outlined above.

Part 01 - Attendance Mastery Quiz (5 points)

The Mastery Quiz is graded based on a Credit/No-credit basis. Students are required to complete the Mastery Quiz only in the laboratory room, which is in **Alden 109** before 3:15 PM. Submissions after 3:15 PM is not acceptable. Students who had completed the Mastery Quiz according to the policy outlined above will automatically receive (5) points towards their lab grade. The link to the Mastery Quiz is provided below:
`https://forms.gle/Y3r6k6i9FdcG9fXr5`

Part 02 - A Simple Exercise to integrate ADT's to an algorithmic problem. (15 points)



Students are required to complete this part only in the laboratory room, which is in **Alden 109**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is late submission accepted for this part, according to the late policy outlined in the course syllabus.

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 forumalate 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. An alternative approach is to install latex on your laptop using the web resources below:
   (a) **MAC**: `https://www.latex-project.org/get/`
   (b) **Ubuntu**: `https://milq.github.io/install-latex-ubuntu-debian/`
   (c) **Windows**: `https://www.youtube.com/watch?v=oI8W4MvFo1M`
5. An easier way to do Latex development is through Overleaf. To do the development without the internet, it is best to install latex on your laptop.
6. There are also apparently options to install latex as a plugin in popular text editors such as Atom. This is not something that had been tried out by the Professor yet. Hence, it is up to the students to try out this option.
7. **ADD** the following statement to all your submission files including the file named `josephus.tex`.

   This work is mine unless otherwise cited - Student Name
8. 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.
9. 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.
10. It may be too tempting to try to propose a solution that is of linear asymptotic running time. However, it is accepted to submit a solution that is in-between linear and quadratic solution.
11. **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**?
12. 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 `josephus.pdf` and uploaded to the repository. Both the **tex** and **pdf** files will be used for grading purposes.

Part 03 - Pair Discussion (5 points)

Students are required to complete this part only in the laboratory room, which is in **Alden 109**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is no late submission accepted for this part unless prior arrangements are done with the Professor.

1. Pair up with at least one of your peers.
2. Talk to your peer and discuss at least one application, in which, applying the solution to the Josephus problem matters.
3. Please note, we already looked at one such application in the Hot Potato problem. The pair should identify a different application where applying the Josephus problem solution is useful.
4. Each person in the pair should discuss their own idea and write a short summary to explain their idea. The other person should critic the idea of their peer by providing them with at least one strong and weak point related to their peer's idea.
5. The summary should also include the original idea and the critic points provided by the peer.
6. Include the name of your peer in the summary file.
7. Each person in the pair should then include their discussion points in the summary document. If the team is unable to come up with an alternative solution, then this should be indicated as the discussion results.
8. Name the summary file as **pd-summary.pdf**.
9. Here **pd** refers to pair discussion. In order to pass the submission requirements, the file should be generated using a markdown (or) PDF format. **ADD** the following statement to all your code files including the file named `pd-summary.pdf` or pd-summary markdown file.

This work is mine unless otherwise cited - Student Name

After completing all the parts in Section 01, it is acceptable for students to start working on Section 2 remotely. Although it is strongly recommended to be in the laboratory room till the end of the lab session, and work on the lab along with other colleagues.

Section 2: Take Home Exercise (25 points)

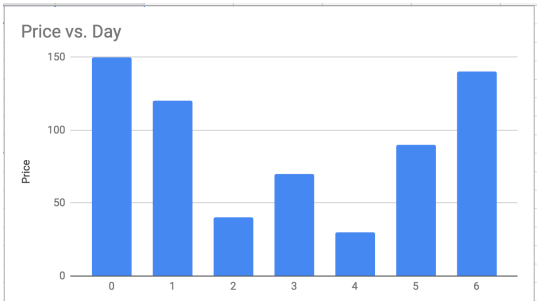
Students are recommended to get started with this part in the laboratory room, by discussing ideas and clarifying with the Professor and the Technical Leader(s). All the part(s) in this section is due in **one week** time. It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. The deadline for submitting the part(s) in this section is **7th Feb 2020, 8:00 AM**. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

Part 01 - A Time Series problem implementation (15 points)

Implement the Time Series problem using a series of requirements outlined below.

1. The problem definition is provided below:

   The span S_i of a stock's price on a certain day i is the maximum number of consecutive days (up to the current day) the price of the stock has been less than or equal to its price on day i. Let us suppose we are given with the stock prices for different days. Identify the span S for those days.



   Span[0,1,2,3,4,5,6] = {1,1,1,2,1,4,6}

2. Please refer to lesson 3 and 4 and your notes for more details from class discussion related to this problem.

3. In order to easily access the Time Series algorithm Version 1 is provided below:

   Algorithm - ComputeSpan(P)

   Input: an n-element array P of numbers such that P[i] is the price of the stock on a day i.

   Output: an n-element array S of numbers such that S[i] is the span of the stock on a day i.

```
1:  for i = 0 to n − 1 do
2:      h ← 0
3:      done ← false
4:      while (h = i) or done) do
5:          if P[i − h] <= P[i] then
6:              h ← h + 1
7:          else
8:              done ← true
9:          end if
10:     end while
11:     S[i] ← h
12: end for
13: return S
```

4. The starter code file is provided inside the lab repository in a file named, `TSA1.java`. Here TSA refers to Time Series Algorithm.

5. **ADD** the following statement to all your code files including the file named `TSA1.java`.

   This work is mine unless otherwise cited - Student Name

6. Implement the **computeSpan** method using the approach 1 algorithm provided above. The 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.

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

8. 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.

9. In order to easily access the Time Series algorithm Version 2 is provided below:

   Algorithm - ComputeSpan(P)

   Input: an n-element array P of numbers such that P[i] is the price of the stock on a day i.

   Output: an n-element array S of numbers such that S[i] is the span of the stock on a day i.

```
1:  Initialize an empty stack D
2:  Initialize an array S
3:  for i = 0 to n − 1 do
4:      h ← 0
5:      done ← false
6:      while not (D is empty() or done) do
7:          if P[i] >= P[D.top()] then
8:              D.pop()
9:          else
10:             done ← true
11:         end if
12:     end while
13:     if D is empty() then
14:         h ← −1
15:     else
16:         h ← D.top()
17:     end if
18:     S[i] ← i − h
19:     D.push(i)
20: end for
21: return S
```

10. The starter code file is provided inside the lab repository in a file named, `TSA2.java`. Here TSA refers to Time Series Algorithm.

11. **ADD** the following statement to all your code files including the file named `TSA2.java`.

    This work is mine unless otherwise cited - Student Name

12. Implement the **computeSpan** method using the approach 2 algorithm provided above by integrating the Stack ADT to the program. The 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.

13. A sample code file named `StackUsage.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.

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

15. 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.

Part 02 - An implementation of the Hot Potato Problem. (10 points)

Write a Java 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 for implementing 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`. Here HOP refers to the Hot Potato problem.

4. **ADD** the following statement to all your code files including the file named `HOP.java`.

   This work is mine unless otherwise cited - Student Name

5. A sample code file named `QueueUsage.java` 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.

6. Implement the **findWinner** method using the algorithm that was designed in Part-2 of Section-1. The 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.

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

8. 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.

Submission Details

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

1. Commented source code from the “josephus.tex” program.

2. Commented source code from the “josephus.pdf” program.

3. Commented source code from the “TSA1.java” program.

4. Commented source code from the “TSA2.java” program.
5. Commented source code from the “HOP.java” program.
6. A document containing the peer discussion points, named `pd-summary.pdf` or pd-summary markdown file.
7. It is highly important, for you to meet the honor code standards provided by the college and to ensure that the submission is made before the deadline. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine unless otherwise cited.” in all your deliverables such as source code and PDF files. In your summary file, please make sure to include your name and your partner’s name.

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.