DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 352: Data Structure and Algorithms
Winter 2023
ASSIGNMENT 2
Due date and time: Thursday, February 23rd by 11:59 pm

Written Questions (50 marks):

Q.1 Consider the following scenario: you have a machine hall containing three pegs named **A**, **B**, and **C**. Each peg can hold a number of discs, but a disc with a larger diameter can never be placed on top of a disc with a smaller diameter and all discs have different diameters. If there are *n* discs, then the discs are numbered from 1 to *n*, where 1 is the smallest disc and *n* is the largest disk. There exists a robot arm that can move exactly one disc at a time from peg **A** to **B** or **B** to **A**, or from **B** to **C** or **C** to **B**. The robot arm cannot do any other move. Initially, all *n* discs are on peg **A** stacked by increasing diameters from top to bottom, with the bottommost one of the largest diameter. The goal is to move all discs from peg **A** to **C** by using only the robot arm based on the rules and restrictions described above (**Hint**: This problem is a more restricted version of the famous Tower of Hanoi problem). You have to submit the following deliverables:

a) Well documented pseudocode of a **recursive** algorithm that solves this problem and prints a protocol of all disc movements, e.g., "disc 7 moved from **A** to **B**".
b) A hand-run of your algorithm for $n = 3$.
c) Formulation and calculation of its complexity using the big-Oh notation. Justify your answers.

Q2. Consider the problem of generating all the possible permutations of length n. For example, the permutations of length 3 are: {1,2,3}, {2,1,3},{2,3,1}, {1,3,2}, {3,1,2}, {3,2,1}. In this question, you will provide:

a) Well documented pseudocode of a **recursive** algorithm that computes all the permutations of size n. Calculate the time complexity of your algorithm using the big-Oh notation. Show all calculations.
b) Well documented pseudocode of a **non-recursive** algorithm that computes all the permutations of size n. The only data structure allowed is a **queue**. Any other memory usage should be O(1). Calculate the time complexity of your algorithm using the big-Oh notation. Show all calculations.

Q3. Answer the following questions:

a) In this assignment, you will develop a well-documented pseudocode that generates all possible subsets of a given set *T* (i.e. power set of *T*) containing *n* elements with the following requirements: your solution must be **non-recursive,** and must use a **stack** and a **queue** to solve the problem. For example: if T = {2, 4, 7, 9} then your algorithm would generate: {}, {2}, {4}, {7}, {9}, {2,4}, {2,7}, {2,9}, {4,7}, {4,9}, {7,9}, …etc. (**Note**: your algorithm's output needs not be in this order).
b) Calculate the time complexity of your algorithm using the big-Oh notation. Show all calculations.

Q4. Rank the following functions in non-decreasing order (≤) according to their big-Oh complexities and justify your ranking:

$n^4 + (\log n)^2$, $\log\log n$, $\sqrt{n}$, $n! + n$, $\frac{n}{2}$, $\binom{n}{2}$, 2^n, $n \log n$, n^n, $2^{\log n}$, $2^{n!} + n^2$, 2^{2^n}

Programming Questions (50 marks)*:*

Q4. In this programming assignment, you will expand and implement the pseudo code discussed in class for an arithmetic calculator that uses two different stacks. The expansion will involve inclusion of more binary operators and the parentheses pairs (possibly nested ones). You will implement your code in Java. Your *arithmetic calculators* must read lines of text from a text file, where each line contains a **syntactically correct** arithmetic expression. Your output file must repeat the input line and print the computed result on the next line. Your calculators must support the following operators on integers and observe the standard operator precedence as shown in the following (1 to 6: 1 is the highest and 6 is the lowest. Same precedence operators are evaluated from left to right).

1. Parentheses (possibly nested ones): (,)

Binary operators:

2. power function: x^y.
3. operators: *, /
4. operators: +, -
5. operators: >, ≥, ≤, <
6. operators: ==, !=

As part of this programming assignment, you will do the following:

a) Design and submit the pseudo-code of the arithmetic calculator.
b) Implement a Java program. In your program, **you will implement and use your own array-based stack (and not the built-in Java stack) of dynamic size based on the linearly incremental strategy discussed in class** (refer to the code segments provided on your slides and the textbook).
c) Briefly explain the time and memory complexity of your calculator. You can write your answer in a separate file and submit it together with the other submissions.
d) Provide test logs for the Java program for at least 20 different and sufficiently complex arithmetic expressions that use all types of operators (including parentheses) in varying combinations.

Submit all your answers to written questions in PDF format only. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into two separates .zip files (one for written questions and one for Programming) and submitted via Moodle (Refer to the course outline for more details on submission guidelines).