**Department Computer Science and Software Engineering**
**Concordia University**

# COMP 352: Data Structures and Algorithms
# Fall 2020 - Assignment 2

**Due date and time: Tuesday October 20th, 2020 by midnight**

**Written Questions (50 marks): Please read carefully: You must submit the answers to all the questions below. However, only one or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.**

## Question 1

**a.** What is the big-O ($O(n)$) and big-Omega ($\Omega(n)$) time complexity for the following algorithm in terms of n? Show all necessary steps:

```
Algorithm MyMagic(A, n)
 Input: Array of integer containing n elements
 Output: Possibly modified Array A
  done ← true
  j ← 0
  while j ≤ n - 2 do
    if A[j] > A[j + 1] then
      swap(A[j], A[j + 1])
      done:= false
    j ← j + 1
  end while
  j ← n - 1
  while j ≥ 1 do
    if A[j] < A[j - 1] then
      swap(A[j - 1], A[j])
      done:= false
    j ← j - 1
  end while
 if ¬ done
  MyMagic (A, n)
 else
  return A
```

**b.** Document the hand-run of MyMagic for array A = (9,3,11,5,2). What is the resulting A?

**c.** Describe the functionality of MyMagic. What can be asserted about its result given an arbitrary integer array A as input?

**d.** Can the runtime of MyMagic be improved easily?

**e.** What type of recursion is used in MyMagic? Is MyMagic tail-recursive?

**Question 2**

For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) is θ(g(n)). For each pair, determine which relationship is correct. Justify your answer.

| | | |
|---|---|---|
| i) | $f(n) = 10^5 n \log n + n^3$ | $g(n) = \log n$ |
| ii) | $f(n) = 2\log n^2$ | $g(n) = (\log n)^2$ |
| iii) | $f(n) = \log n^2 + n^3$ | $g(n) = \log n + 5$ |
| iv) | $f(n) = n\sqrt{n} + \log n$ | $g(n) = \log n^2$ |
| v) | $f(n) = 2^n + 10^n$ | $g(n) = 10n^2$ |
| vi) | $f(n) = n!$ | $g(n) = n^n$ |
| vii) | $f(n) = \log^2 n$ | $g(n) = \log n$ |
| viii) | $f(n) = n$ | $g(n) = \log^2 n$ |
| ix) | $f(n) = \sqrt{n}$ | $g(n) = \log n$ |
| x) | $f(n) = 2^n$ | $g(n) = 3^n$ |
| xi) | $f(n) = 2^n$ | $g(n) = n^n$ |

**Programming Questions (50 marks):**

In this programming part you are asked to implement a game called ***MagicBoard.***

*MagicBoard* is a 1-player game using a chess-like board consisting of $d \times d$ squares with $d$ between 5 and 20, and each square contains an integer between 0 and $d$ -1. The rules of the game are simple. The circle marks the start position on the board. The integer ($n$) in the circled square indicates that the circle can move $n$ squares on the board. In one step the circle can move either $n$ squares east or west or north or south. At each step the chosen direction is fixed. The circle cannot move off the board. The only legal start positions are the four corner squares. The board must contain exactly one goal square with zero as value that can be located anywhere on the board, but its position cannot be identical to the start position.

| | | | | |
|---|---|---|---|---|
| ④ | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | 3 | 1 | 2 | 0 |

For instance, in the above example the circle can move either 4 squares east or south. All other moves are illegal. The objective of the game is to move the circle to the goal square containing the zero value. In the configuration given below, you can solve the game by making the following sequence of steps:

1. Step
Move south

| | | | | |
|---|---|---|---|---|
| ④ | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | 3 | 1 | 2 | 0 |

2. Step
Move east

| | | | | |
|---|---|---|---|---|
| 4 | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| ③ | 3 | 1 | 2 | 0 |

3. Step
Move west

| | | | | |
|---|---|---|---|---|
| 4 | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | 3 | 1 | ② | 0 |

4. Step
Move east

| | | | | |
|---|---|---|---|---|
| 4 | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | ③ | 1 | 2 | 0 |

Final position

| | | | | |
|---|---|---|---|---|
| 4 | 2 | 1 | 3 | 1 |
| 2 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | 3 | 1 | 2 | ⓪ |

Although the given example is solvable, actually with more than one solution, some board configurations may be unsolvable, i.e., the goal square cannot be reached from the given start position. Below is a simple example.

| | | | | |
|---|---|---|---|---|
| ① | 4 | 1 | 3 | 1 |
| 4 | 3 | 2 | 1 | 4 |
| 3 | 2 | 3 | 1 | 4 |
| 1 | 3 | 4 | 2 | 3 |
| 3 | 4 | 1 | 2 | 0 |

In this configuration, the circle can only move to its two adjacent squares with value 4. If it moves to its eastern neighbor, it will bounce south-north between two 4's, and if it moves to its southern neighbor it will bounce east-west between two 4's.

## Requirements:

**1.** In this programming assignment, you will develop **two versions** of the *MagicBoard* game. For both versions you have to design an algorithm in pseudo code and develop a corresponding Java implementation**.**
   - The first version will be completely based on recursion. No iterations are allowed.
   - The second one will be iterative and based on a stack, queue, list, or vector.

2. Your solution takes a legal start position of the circle along with a $d \times d$ array of square values. Your solution should return *true* if it is possible to solve the game from the start position and *false* if it is impossible. Your solution should work for any $d \times d$ board with $d$ between 5 and 20, and all squares should contain a random integer value between 1 and $d$-1 with the exception of the goal square that contains the only zero value. You are not allowed to modify the values on the board at any time.

   a) Briefly explain the time and memory complexity for both versions of your game. You can write your answer in a separate file and submit it together with the other programming submissions.
   b) For the first version of your solution describe the type of recursion used in your implementation. Does the particular type of recursion have an impact on the time and memory

complexity? Justify your answer. Also explain whether a tail-recursive version is possible. If yes, you can earn bonus marks by submitting such a version.

c) For the second part of your solution, justify why you choose that particular data structure (e.g. why you choose a stack and not a queue, etc.). Explain whether your chosen data structures have an impact on the time and memory complexity.

d) For each version provide test logs for at least 20 different game configurations. They must be sufficiently complete to show that your solution works for various board dimensions and square values.

e) Explain how one can detect unsolvable board configurations and whether there exists a way to speed up the execution time.

For each version you are required to submit your pseudo code, the corresponding fully commented Java source files, the compiled files (.class files), the log and text files.

You will need to submit both the pseudo code and the Java program, together with your experimental results. Keep in mind that Java code is **not** pseudo code.

**The written part must be done individually (no groups are permitted). The programming part can be done individually or in groups of two students (with all members from the same section).**

**For the written questions, submit all your answers in PDF (no scans of handwriting; this will result in your answer being discarded) or text formats only. Please be concise and brief (less than ¼ of a page for each question) in your answers. Submit the assignment under Theory Assignment 2 directory in EAS or the correct Dropbox/folder in Moodle (depending on your section).**

**For the Java programs, you must submit the source files together with the compiled files. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via Moodle/EAS under Programming 2 directory or under the correct Dropbox/folder. You must upload at most one file (even if working in a team; please read below). In specific, here is what you need to do:**

1) Create **one** zip file, containing the necessary files (.java and .html). Please name your file following this convention:
If the work is done by 1 student: Your file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID number.
If the work is done by 2 students: The zip file should be called *a#_studentID1_studentID2*, where # is the number of the assignment *studentID1* and *studentID2* are the student ID numbers of each student.

2) If working in a group, only one of the team members can submit the programming part. Do not upload 2 copies.

**Very Important:** Again, the assignment must be submitted in the right folder of the assignments. Depending on your section, you will either upload to Moodle or EAS (your instructor will indicate which one to use). **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

⇨ Additionally, for the programming part of the assignment, a demo is required (please refer to the courser outline for full details). The marker will inform you about the demo times. **Please notice that failing to demo your assignment will result in zero mark regardless of your submission.** If working in a team, both members of the team must be present during the demo.