# Comp 251: Assignment 2

Answers must be returned online by March 1st (11:59:59pm), 2025.

## General instructions (Read carefully!)

- **Important:** All of the work you submit must be done by only you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. For Comp251, we will use software that compares programs for evidence of similar code. This software is very effective and it is able to identify similarities in the code even if you change the name of your variables and the position of your functions. The time that you will spend modifying your code, would be better invested in creating an original solution.

  Please don't copy. We want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

  Never look at another assignment solution, whether it is on paper or on the computer screen. Never share your assignment solution with another student. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web, or get code from a private tutor, that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in CS involve students who have never met, and who just happened to find the same solution online, or work with the same tutor. If you find a solution, someone else will too. The easiest way to avoid plagiarism is to only discuss a piece of work with the Comp251 TAs, the CS Help Centre TAs, or the COMP 251 instructors.

- Your solution must be submitted electronically on ed-Lessons.

- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments (i.e. as a comment at the beginning of your java source file) the names of the people with whom you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators". If asked, you should be able to orally explain your solution to a member of the course staff. At the end of this document, you will find a check-list of the behaviours/actions that are allowed during the development of this assignment.

- This assignment is due on March $1^{th}$ at 11h59:59 pm. It is your responsibility to guarantee that your assignment is submitted on time. We do not cover technical issues or unexpected difficulties you may encounter. Last minute submissions are at your own risk.

- This assignment includes a programming component, which counts for 100% of the grade.

- Multiple submissions are allowed before the deadline. We will only grade the last submitted file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue.

- Late submissions can be submitted for 24 hours after the deadline, and will receive a flat penalty of 20%. We will not accept any submission more than 24 hours after the deadline. The submission site will be closed, and there will be no exceptions, except medical.
- In exceptional circumstances, we can grant a small extension of the deadline (e.g. 24h) for medical reasons only.
- Violation of any of the rules above may result in penalties or even absence of grading. If anything is unclear, it is up to you to clarify it by asking either directly the course staff during office hours, by email at (`comp251.cs@mcgill.ca`) or on our discussion board (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on one of the communication channels used in the course. It is your responsibility to monitor MyCourses and the discussion board for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time. In particular, we will not answer individual emails about the assignment that are sent the day of the deadline.

**Programming component**

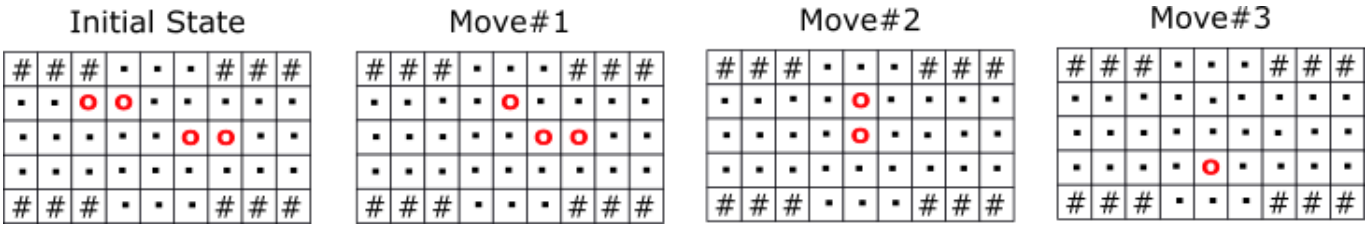
- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will result in an automatic 0.**
- Public tests cases are available on ed-Lessons. You can run them on your code at any time. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging, private set of test cases. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the discussion board. Do not include it in your submission.
- Your code should be properly commented and indented.
- **Do not change or alter the name of the files you must submit, or the method headers in these files**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded.
- **Do not add any package or import statement that is not already provided**
- Please submit only the individual files requested.
- **You will automatically get 0 if the files you submitted on ed-Lessons do not compile, since you can ensure yourself that they do. Note that public test cases do not cover every situation and your code may crash when tested on a method that is not checked by the public tests. This is why you need to add your own test cases and compile and run your code from command line on linux.**

# Homework

**Exercise 1** (60 points). ***Complete Search***

Let's play a fun game :). We will provide a two-dimensional board with some holes on it, where each hole can contain a ball. In each turn during the game, you will only be allowed to jump (horizontally or vertically) over an adjacent ball into the empty hole next to it. Once the jump is performed, the ball which was jumped over is removed from the board. The idea of the game is to apply the minimum number of moves (i.e., jumps) to end in a state with a minimum number of balls such that no further moves can be made. Note: there could be cases where balls are dispersed and cannot cancel each other.

Let's analyze one example to make sure that the game is clear. Figure 1 shows the initial state of the game, and a sequence of moves (i.e., horizontal or vertical jumps over adjacent balls) that allows for a state (i.e., the last state) with the minimum number of balls in the board. In particular, we performed three moves to end with one ball on the board. Please notice that in this representation, "." (a period) denotes an empty hole, and "o" (small vowel o) a ball, and "#" (number sign) indicates a part of the board without a hole. For this question, you can safely assume that all the boards always have the same shape ($5 \times 9$).




For this question, your task is to develop an algorithm (i.e., a complete search one) that determines the minimum number of balls that remains in the board after applying (recursively) the above-mentioned movement. You will also need to report the minimum number of moves required to reach that number of balls. You will need to submit your `A2_Q1.java` source file to the `Assignment 2 => Q1 - Complete Search` lesson in Ed-Lessons. Please complete the function `game` which receive a 2-D array of Strings representing a board (as the one shown in Figure 1). The function `game` must return an array of integers (`int[]`) of two positions, where the first position (i.e., index 0) reports the minimum number of balls that remains in the board and the second position (i.e., index 1) reports the minimum number of moves performed to reach that number of balls.

**Exercise 2** (50 points). ***Divide and Conquer***

Given the requirements for limiting contacts between students during the COVID pandemic, the University designed a new plan to design a seating plan at final exams. The goal was to ensure that students to only be in close contact with students whose ID number is relatively close to theirs. They proposed the following protocol that students arriving at the exam room should follow. When a student arrives at the exam room, they join the queue of students who arrived before

them (i.e., they stand at the back of the queue). Once the queue is complete (i.e., all the Comp 251 students registered to take the course arrive to the queue), the University must guarantee that the students in the queue appear in increasing order based on their student ID. This ensures that students are only in close contact with other students with a similar ID number. However, the Department of Computer Science raised a concern about the efficiency of the proposed strategy. One instructor (yes, it was David) suggested that the efficiency of the strategy can be understood by computing the total number of successively swapping pairs of **adjacent students** in the queue. Furthermore, the instructor suggested that the Comp251 students will be more than happy (aren't you?) to create a Java program (based on a divide and conquer approach) to compute this number.

You will need to submit your `A2_Q2.java` source file to the `Assignment 2 => Q2 - Divide and Conquer` lesson in Ed-Lessons. Please complete the function `num_swaps` which receive a 1-D array of integers representing the queue of students. The function `num_swaps` must return a number representing the total number of successively swapping pairs of **adjacent students** in the queue to guarantee that the queue appear in increasing order based on their student ID.

Let's see one example to make sure that the exercise is clear.

SAMPLE CALL 1:

```
num_swaps([3,1,2])
```

SAMPLE OUTPUT 1:
2

NOTES CALL 1: Please notice that the numbers 3 and 1 (which are stored in the **consecutive** indexes of the array 0 and 1) need to be swapped. This swap will create the array $\{1, 3, 2\}$. Then, the numbers 3 and 2 (which are stored in the **consecutive** indexes of the array 1 and 2) need to be swapped to produce the array $\{1, 2, 3\}$. We required two swaps to produce a sorted array.

**Exercise 3** (60 points)**.** ***Dynamic Programming***

During Winter'25, I (i.e., David) registered the Staff class called **Learn to Run 2**. This course is "designed for those who are at the beginning of their running journey or who are returning to running after an injury. This class is tailored for beginners who can currently run for 10 minutes consecutively but haven't reached the 3-5km mark yet"[1]. The training in this class happens in the stairs of the Mt Royal mountain and it consists of a slow and gradual build-up of endurance with walk-climb intervals. In particular, we walk-climb a certain number of stairs steps, rests for ten seconds, then walk-climb again, rests for ten seconds again, and so on. We start our training at street level in the stairs close to the McMed building (where we have our Comp251 lectures). The coach (i.e., the instructor of the class) give us a sequence of numbers $(s_1, s_2, \ldots, s_m)$ telling us how many stair steps we must take before the first break, before the second break and so on. We are allowed to decide if we want to climb up or climb down those stairs steps, but we will always guarantee that our training is legal. One training is legal if and only if:

- The training starts and end at the street level.
- We only use existing stairs steps (i.e., we can never be below the street level or above the

[1]The information is taken from: https://recreation.mcgill.ca/staff-programming

number of stair steps in Mt Royal). Please notice that the stairs in Mt Royal (where we train) has a maximum of one thousand stair steps.

Given that I have my Comp251 class in McMed just after my **Learn to Run 2** training, and as a personal constraint, I also need to guarantee that I stay as close as possible (at any time of the training) to the street level.

As a Comp251 student, you have been commissioned to help me determine when should I go up and when down in my training. In particular, please complete the function `directions` in the `A2_Q3.java` source file. The function `directions` receives as parameter an array $S = \{s_1, s_2, \ldots, s_m\}$ of positive integers storing the number of stair steps that I have to walk in. `directions` must return a String made of the concatenation of two possible characters ("U" for up, and "D" for down), where the $i^{th}$ character indicates if I should climb up or down at the $i^{th}$ stage. Please return the String "IMPOSSIBLE" if no legal solution exists.

Let's see some examples to make sure that the exercise is clear.

SAMPLE CALL 1:

```
directions([20,20,20,20])
```

SAMPLE OUTPUT 1:
`"UDUD"`

NOTES CALL 1: If the number of stair steps are $S = \{20, 20, 20, 20\}$, I have two possible legal solutions. In the first one, I climb up 20, up 20, down 20, down 20(i.e., "UUDD"). In the second one, I climb up 20, down 20, up 20, down 20(i.e., "UDUD"). Both solutions are legal; however, the second one is better because it only goes 20 stair steps away from the starting point, whereas the first one goes 40 stair steps away from the starting point (please remember that I have class just after the training and I do not want to be too far from McMed building).

SAMPLE CALL 2:

```
directions([3,2,5,3,1,2])
```

SAMPLE OUTPUT 2:
`"UUDUDD'`

SAMPLE CALL 3:

```
directions([3,4,2,1,6,4,5])
```

SAMPLE OUTPUT 3:
`"IMPOSSIBLE'`

**Exercise 4** (30 points)**.** ***Greedy***

In this exercise, you will plan your homework with a greedy algorithm. The input is a list of

homework defined by two arrays: deadlines and weights (the relative importance of the homework towards your final grade). These arrays have the same size and they contain integers between 1 and 100. The index of each entry in the arrays represents a single homework, for example, `Homework 2` is defined as a homework with deadline `deadlines[2]` and weight `weights[2]`. Each homework takes exactly one hour to complete.

Your task is to output a `homeworkPlan`: an array of length equal to the last deadline. Each entry in the array represents a one-hour timeslot, starting at 0 and ending at 'last deadline - 1'. For each time slot, `homeworkPlan` indicates the homework which you plan to do during that slot. You can only complete a single homework in one 1-hour slot. The homeworks are due at the beginning of a time slot, in other words if an assignment's deadline is `x`, then the last time slot when you can do it is `x - 1`. For example, if the homework is due at t=14, then you can complete it before or during the slot t=13. If your solution plans to do `Homework 2` first, then you should have `homeworkPlan[0]=2` in the output. Note that sometimes you will be given too much homework to complete in time, and that is okay.

Your homework plan should maximize the sum of the weights of completed assignments.

To organize your schedule, we give you a class `HW_Sched.java`, which defines an `Assignment` object, with a number (its index in the input array), a weight and a deadline.

The input arrays are unsorted. As part of the greedy algorithm, the template we provide sorts the homeworks using Java's `Collections.sort()`. This sort function uses Java's `compare()` method, which takes two objects as input, compares them, and outputs the order they should appear in. The template will ask you to override this `compare()` method, which will alter the way in which Assignments will be ordered. You have to determine what comparison criterion you want to use. Given two assignments A1 and A2, the method should output:

- 0, if the two items are equivalent
- 1, if a1 should appear after a2 in the sorted list
- -1, if a2 should appear after a1 in the sorted list

The `compare`[2] method (called by `Collections.sort()`) should be the only tool you use to re-organize lists and arrays in this problem. You will then implement the rest of the `SelectAssignments()` method.

# What To Submit?

Attached to this assignment are java template files. You have to submit only this java files. Please DO NOT zip (or rar) your files, and do not submit any other files.

# Where To Submit?

You need to submit your assignment in ed - Lessons. Please note that you do not need to submit anything to myCourses.

[2]For more information about the compare function, visit `https://www.geeksforgeeks.org/how-compare-method-works-in-java/`

## When To Submit?

Please do not wait until the last minute to submit your assignment. You never know what could go wrong during the last moment. Please also remember that you are allowed to have multiple submission. Then, submit your partial work early and you will be able to upload updated versions later (as far as they are submitted before the deadline).

## How will this assignment be graded?

Each student will receive an overall score for this assignment. This score is the combination of the passed open and private test cases for the questions of this assignment. The open cases correspond to the examples given in this document plus other examples. These cases will be run with-in your submissions and you will receive automated test results (i.e., the autograder output) for them. You MUST guarantee that your code passes these cases. In general, the private test cases are inputs that you have not seen and they will test the correctness of your algorithm on those inputs once the deadline of the assignment is over; however, for this assignment you will have information about the status (i.e., if it passed or not) of your test. Please notice that not all the test cases have the same weight.