Comp 251: Assignment 2

Answers must be returned online by March 9th (11:59:59pm), 2023.

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 9^{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, and an optional long answer component designed to prepare you for the exams. This component will not be graded, but a solution guide will be published.

- 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 (`cs251-winter@cs.mcgill.ca`) or on the discussion board 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 (30 points)**.** ***Complete Search***

Please help me!!!!!! My kid is wining every game that we play together. Let me try to give you some context. My kid just invented a new game (guess what, YES, using Lego pieces) with the following rules.

- The game involves two players (my kid and myself).
- The game is played in a 5-row equilateral triangle game board containing 15 holes (5 holes per side of the triangle).
- At the beginning of the game, there is one open hole and 14 holes occupied by Lego pieces. My kid has assigned a numerical value to each piece depending on the size of the Lego piece (exactly as he did in the Q3 of our previous assignment).
- Since I am very bad in the game, my kid always allows me start the game (I will always do the first move). After, we alternate turns.
- A valid move consist of picking up a Lego piece to jump it over an adjacent Lego piece, landing on an empty hole adjacent to the jumped piece. The Lego piece that is jumped over is removed from the board. Jumps must be in straight lines (horizontally or diagonally).
- The score of a move is the product of the two Lego pieces involved in the jump. The total score of a player is the sum of the scores of their jumps.
- The game ends when a player has no possible jumps to make.
- The objective of the game is to maximize the difference of your score minus your opponent's score (that's right, my kid isn't happy to just win the game, he wants to humiliate me in the process). For example, my kid would prefer to win with a score of 200 to David's score of 120 (with a differential of 80), than to win with a score of 2000 to David's score of 1990 (with a smaller differential of only 10). Similarly, I want to win by as much as possible over my kid.

Figure 1 shows some of the possible first moves in the game created by my kid. In particular, the red arrow shows a move worth 42 points and the blue arrow shows a move worth 12 points. The green, cyan and purple arrows show possible moves that my kid can perform (depending on the state of the board for his move). Hint: Hopefully the figure also shows you the recursion tree to traverse :). Please notice that a value of zero (0) represents a hole while the rest of the values represent the sizes of the Lego pieces. These sizes are integers between 1 and 100, inclusive. Also, notice that it is possible for two different Lego pieces to have the same value (because they have the same size). You can safely assume that all the input (initial) boards start with exactly one hole (i.e., out of the 15 initial holes, it is guaranteed that exactly one is 0). Just for your reference, the solution/answer for the problem show in Figure 1 is 21 (**Hint: Write out the first few**

moves of this example with pen and paper before coding your solution. This will help you to better understand the recursion tree and hopefully require less debugging). You can find other examples in the auto-grader.

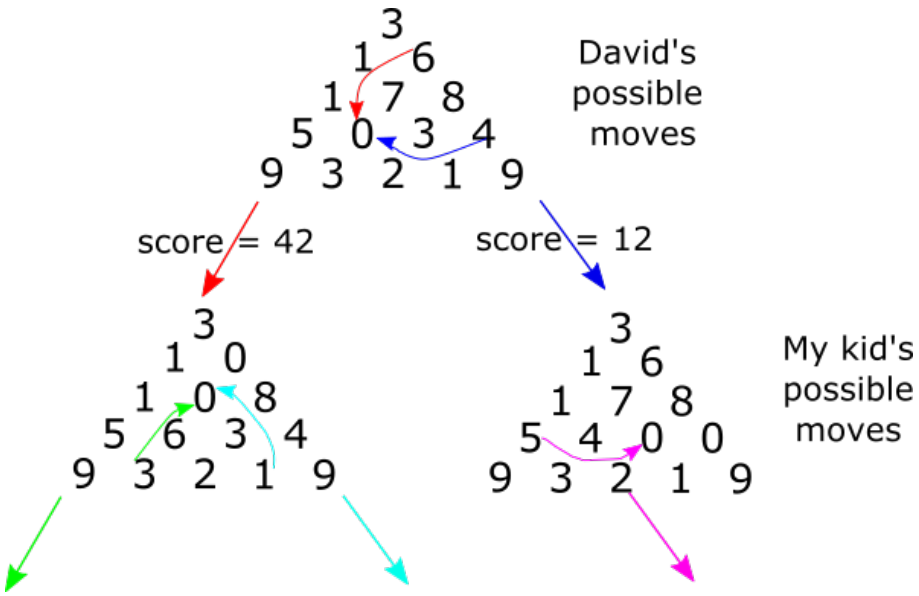

Figure 1: Some possible moves

Please help me!!!!!! I have lost all the games that I have played with my kid and I am losing his respect. For this question, your task is to develop an algorithm (i.e., a complete search one) that outputs the maximum value of my score minus the score of my kid at the end of the game. Please remember that both of us want to win, so we are both playing optimally. 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_recursion` which receives a 2-D array of `int` representing a board, aligned to the left. For example, the starting board in the Figure 1 is represented as follows `{{3,-1,-1,-1,-1},{1,6,-1,-1,-1}, {1,7,8,-1,-1}, {5,0,3,4,-1}, {9,3,2,1,9}}`. Please notice that the values of `-1` represent unavailable holes (i.e., not part of the board). The function `game_recursion` must return one integer (`int`) reporting the maximum possible difference between my score and the one of my kid.

Exercise 2 (20 points plus 5 bonus points)**.** ***Dynamic Programming (+ Greedy)***

For this part of the assignment, we will solve a very common day-to-day problem. What is the smallest number of coins required to dispense exactly an amount that is owed? This question can always be solved by using a Dynamic Programming algorithm; however, for certain coin systems (as the Canadian), this question can be answered quicker by running a greedy approach. This greedy approach involves repeatedly choosing a coin of the largest denomination that is less than or equal to the amount still owed, until the amount owed reaches zero. As a Comp251 student, you have been commissioned to: Given a coin system $C = \{c_1, c_2, \ldots, c_n\}$, determine whether a greedy algorithm will always give an optimal solution or not. For this problem, we will work under the following restrictions:

- The number of denominations n in the coin system is greater than 1 and less than 101 (i.e., $1 < n < 101$).
- The n denominations are integers $c_1, c_2, \ldots, c_n$, where $1 = c_1 < c_2 < \cdots < c_n < 10^6$

For this question, you will need to look for a counter-example that proves whether the greedy algorithm fails or not. To do that you have to implement i) a dynamic programming solution to the problem and a ii) greedy approach. Then, if for a specific value, both approaches give different numbers of coins, then you have found a counterexample and your function needs to return the smallest value within all the counterexamples; otherwise, your function returns -1. You will not have to do this test for the set of all possible owed values (which is infinite) because you can take advantage of the useful fact that if a counter-example exists, this example is less than the sum of the two largest denominations of coins in your system. I have to stress here that we are expecting from you a greedy and dynamic-programming implementations (this question is about that), other solutions to the problem that do not implement these two approaches will not be accepted (even if they pass all the test cases). Also, randomly returning a -1 (i.e., hardcode your solution to match some answers) will result in a zero.

Let me give you some examples such that you can test your solution (there are more test cases available in our autograder).

Example 1:
Input: `denominations{1,2,4,8}`
Output: -1

Example 2:
Input: `denominations{1,5,8}`
Output: 10

Example 3:
Input: `denominations{1,5,10,25,100,200}`
Output: -1

You will need to submit your `A2_Q2.java` source file to the `Assignment 2 => Q2 - DP` lesson in Ed-Lessons. Please code your solutions in the function `change` which receives as parameter an array on sorted integers corresponding to the denominations of your coins and returns a `int` reporting if a greedy approach is optimal or not for that specific coin system.

To pass as many test cases as possible, try to optimize the time and space complexity of your code.

Exercise 3 (20 points)**.** ***Greedy***

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

You will then implement the rest of the `SelectAssignments()` method.

Exercise 4 (30 points)**.** ***Divide and Conquer***

Let me tell you something that happened on Tuesday (Feb 7th). I was waiting for my bus in order to go to campus to teach my Comp251 lecture. The lecture that day was about divide and conquer algorithms (what a coincidence). There are three buses that stop at my bus stop. In particular, bus 202 arrives at 9:00 am, bus 250 arrives at 9:01 am and bus 251 arrives at 9:02 am (Ohhh, I just noticed that the numbers of the buses coincide with the codes of our Comp courses, what a coincidence again). Just for your information, the bus that brings me downtown is the 251 route (I know that you didn't expect that :P). When I arrived at the bus stop, there was already a long line of people waiting to take one of the three buses. Please note that given the amount of snow in the pedestrian path, there was only space for **one** line. At 9:00 am, bus 202 arrived and chaos was created. People taking that bus had to step out of the line to get into the bus (remember, there is no space to pass because the snow and there were people in line waiting for the other two buses). The full process of getting into the three buses took almost 11 minutes and it created a traffic jam, delayed the bus system, made the passengers late for work. However most importantly, it gave me an idea for the assignment.

I spent the full commute thinking of a better way to organize people and make the process more efficient. A better strategy would be to guarantee that all the people taking bus 202 (the first bus arriving) should be in the beginning of the line, the ones taking bus 250 should be in the middle and the rest, taking bus 251, should be in the end. I need to guarantee that the order of the people in that new line respects the order of arrival to the line, but also to give priority to people who should board first {e.g., people with a disability, seniors, expectant mothers, children,

and obviously computer science professors}). To solve the problem, I plan to give a unique priority number to each person in the line (based on the bus to take, the order of arrival to the line and other considerations). Since there is no space to step out of the line (remember the snow in the pedestrian path), to get the line organized, only passengers standing next to each other can swap places. In other words, the new arrangement must be accomplished only by successively swapping pairs of consecutive passengers in the line. Based on those conditions, I need your help to **compute the minimum number of swaps required to get the passengers in increasing order by passenger priority**.

Since this is a common problem at many bus stops, you need to create a solution that works with the following restrictions:

- The number of passengers in the line will be no more than one million.
- Each passenger priority number will be one integer between 1 and 10^9 (inclusive). Please remember that no passenger priority number will appear more than once.

Let's now see an example to make sure that the exercise is clear. Given the following list of priority numbers `{3,1,2}`, the minimum number of swaps needed to get the passengers in increasing order is 2.

You will need to submit your `A2_Q4.java` source file to the `Assignment 2 => Q4 - Divide and Conquer` lesson in Ed-Lessons. Please code your solutions in the function `swaps` which receives as parameter an array of unique integers corresponding to the priority passenger numbers and returns a `double` representing the minimum number of swaps required to arrange the passengers in increasing order by passenger priority. Hint: Why do you think we are returning a `double` instead of an `int`?

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 review the tutorial 2 if you still have questions about how to do that (or attend office hours). Please note that you do not need to submit anything to myCourses.

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.

Student Code of Conduct Assignment Checklist

The instructor provides this checklist with each assignment. The instructor checks the boxes to items that will be permitted to occur in this assignment. If an item is not checked or not present in the list, then that item is not allowed. The instructor may edit this list for their case. A student cannot assume they can do something if it is not listed in this checklist, it is the responsibility of the student to ask the professor (not the TA).

Instructor's checklist of permitted student activities for an assignment:

Understanding the assignment:

- ☒ Read assignment with your classmates
- ☒ Discuss the meaning of the assignment with your classmates
- ☒ Consult the notes, slides, textbook, and the links to websites provided by the professor(s) and TA(s) with your classmates (do not visit other websites)
- ☐ Use flowcharts when discussing the assignment with classmates.
- ☒ Ask the professor(s) and TA(s) for clarification on assignment meaning and coding ideas.
- ☐ Discuss solution use code
- ☐ Discuss solution use pseudo-code
- ☐ Discuss solution use diagrams
- ☐ Can discuss the meaning of the assignment with tutors and other people outside of the course.
- ☐ Look for partial solutions in public repositories

Doing the assignment:

- Writing
  - ☒ Write the solution code on your own
  - ☒ Write your name at the top of every source file with the date
  - ☒ Provide references to copied code as comments in the source code (e.g. teacher's notes). Please notice that you are not allowed to copy code from the internet.
  - ☐ Copied code is not permitted at all, even with references
  - ☐ Permitted to store partial solutions in a public repository
- Debugging
  - ☒ Debug the code on your own
  - ☒ Debugging code with the professor
  - ☒ Debugging code with the TA
  - ☒ Debugging code with the help desk
  - ☒ Debugging code with the Internet. Please notice that this is allowed to debug syntax errors, no logic errors.
  - ☐ You can debug code with a classmate
  - ☐ You can debug code with a tutor or other people outside of the course
- Validation
  - ☒ Share test cases with your classmates
- Internet

☒ Visit stack-overflow (or similar). Please notice that this is allowed only to debug syntax errors, no logic errors.
☐ Visit Chegg (or similar)

- Collaboration
  - ☐ Show your code with classmates
  - ☐ Sharing partial solutions with other people in the class
  - ☐ Can post code screenshots on the course discussion board
  - ☒ Can show code to help desk

Submitting and cleaning up after the assignment:

- ☐ Backup your code to a public repository/service like github without the express written permission from the professor (this is not plagiarism, but it may not be permitted)
- ☐ Let people peek at your files
- ☐ Share your files with anyone
- ☐ ZIP your files and upload to the submission box
- ☒ Treat your work as private
- ☐ Make public the solutions to an assignment
- ☐ Discuss solutions in a public forum after the assignment is completed