EECS 2011 Z, Winter 2023 – Assignment 4

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of the course materials, and the course staff, that you consulted**. You must submit a PDF file and Java files (if any) with the required filename. The PDF file could be generated using LaTeX(recommended), Microsoft Word (saved as PDF, **not** the `.docx` file), or other editor/tools. The submission must be **typed**. Handwritten submissions are **not** accepted.

Due April 8, 2023, 10:0[illegible]; required file(s): a4sol.pdf

Answer each question complete[illegible]ur claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. [illegible]ically correct that are hard to understand will not receive full marks. Mark values for each question a[illegible]uare brackets]. Your submitted file does **not** need to include/repeat the questions — just write your solutions.

The assignment must be completed individually.

1. **[10]** We define the **golden element** of a collection of n elements to be the $\lceil 0.618 \times n \rceil$-th smallest element of the collection (note: $\lceil x \rceil$ means the ceiling integer of x). For example, if the collection is `[22, 72, 14, 14, 92, 80, 35, 22]`, then $n = 8$ and $\lceil 0.618 \times n \rceil = 5$, therefore the golden element is the 5th smallest element in the collection which is 35. Note that duplicate elements are possible and each duplicate counts in the ranking — that's why 35 is the 5th smallest element rather than the 3rd.

   Consider the following abstract data type that we will call a "`GOLDEN-SET`."

   Objects: A collection of elements that can be compared with each other. Duplicate elements are allowed (this is why we use the term "collection" instead of "set").

   Operations:

   - `INSERT`(S, x): Add element x to the collection S.
   - `FIND-GOLD`(S): Return the golden element of collection S without removing it.
   - `DIG-GOLD`(S): Remove and return the golden element of collection S.

   Requirements: Let n be the size of S,

   - `INSERT`(S, x) must have worst-case runtime $\mathcal{O}(\log n)$.
   - `FIND-GOLD`(S) must have worst-case runtime $\mathcal{O}(1)$.
   - `DIG-GOLD`(S) must have worst-case runtime $\mathcal{O}(\log n)$.

   In your PDF, describe your implementation of the `GOLDEN-SET` by completing the following parts.

   (a) Describe what data structures you use to implement the `GOLDEN-SET`. You may use multiple data structures that we learned in this course, and/or multiple instances of the same data structure. State clearly what are the structural properties/invariants that must be satisfied by these data structures (e.g., "the size of something must be something").

   (b) For each of the above operations, provide the following information:

   - A brief, high-level description of how the algorithm works for this operation.
   - If the operation modifies the data (i.e., `INSERT` and `DIG-GOLD`), describe how the algorithm maintains the structural property/invariant that you defined in Part (a).
   - Analyze the worst-case runtime of the operation.
   - Provide the pseudocode of the operation.

   Note: Please do **not** repeat algorithms or runtime analyses from the lectures — just refer to known results as needed.
   Hint: Consider using the heap data structure, and consider using more than one heaps.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In this assignment, we have a programming exercise for which you will be asked to write some code and submit it. You may also be asked to include a write-up about your code in the PDF file that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. All code submissions will be checked for plagiarism at the end of the term. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

2. **[10]** Imagine you are work N floors. One day, when you enter the elevator, you notice something different: most of the butt gone, with only the following three buttons left:

   - Button "UP": a butt here U is a positive integer (for example, "Up by 3"). By pushing this button, you will go u reach the top floor (the N-th floor), whichever comes first.
   - Button "DOWN": a y D" where D is a positive integer (for example, "Down by 21"). By pushing this button, rs, or until you reach the ground floor (the 1st floor), whichever comes first.
   - Button "JUMP": a button saying "JUMP 2x", which will bring you to the floor number that is **twice** of the current one, or hit the top floor. For example, if you are currently on floor 12, then pushing the JUMP button will bring you to floor 24, or to the top floor if the top floor is less than 24.

You are currently at floor X and you want to go to floor Y. Being an EECS 2011 trained computer scientist, you immediately take out your laptop to write a program that can tell you the **minimum number of button pushes** needed to take you from floor X to floor Y. Below is the specification of the program:

Input:

- The values of N, U, D, X, and Y.

Output:

- Return the minimum number of button pushes that can take you from floor X to floor Y.
- If it is impossible to go from floor X to floor Y with any sequence of button pushes, return -1.

Assumptions:

- X and Y are different from each other.

Requirements:

- There are a few test cases provided in the `main` function of the starter code. Each call to your function must be able to return the correct answer within **1 second**.
- You are **not** allowed to add any imports to the starter code, which means if you need a class such as a queue, a stack, a linked list, or a tree, you will need to implement them from scratch inside `A4Q2.java`.

Submission: Your submission will include the following parts:

- `A4Q2.java`, which includes the complete implementation of the `minimumButtonPushes()` function as well as any other classes that you add.
- In your PDF file (a4sol.pdf), provide a high-level description of your algorithm, which includes the following information:
  (a) What graph is constructed to solve this problem? What do the vertices and edges represent?
  (b) Is the graph directed or undirected?
  (c) What graph algorithm is used to find the answer? Why is this algorithm appropriate for this problem?
  (d) What is the worst-case runtime of your algorithm? Note that the runtime should be in terms of the given input values of the problem, i.e., N, U, D, X, or Y.

As usual, you may setup the Java project using any programming IDE of your choice and you are fully responsible for using the IDE you choose properly. Be careful: sometimes the IDE automatically inserts lines of code such as the package declaration and imports. Make sure to double-check your code before submission to remove any unnecessary those extra lines since they may cause all test cases to fail in auto-testing.

The marking of this question will be 50% based on the correctness of the code in `A4Q2.java` (passed test cases in auto-testing), and 50% based on the written answers provided in the PDF file.

Note: Even though the question does not ask for it, printing out the actual sequence of button pushes will provide you with interesting insights into the problem and will potentially help with debugging.