CS 245: Lab 1
Due: 11:59 PM Monday, Jan 27

1. You play sabacc for money, and you like to keep track of the number of credits you've won/lost. You only win/lose integer credit values. A positive value like 10 means that you won 10 credits. A negative value like -5 means that you lost five credits.

   You have a *sorted* integer array listing all of the numbers of credits that you've won/lost, with no repeats. (That is, if you've lost 8 credits during several sabacc games, you just have -8 listed once in the array.). You'd like to write a method to let you know at which index a new number should be inserted, as you play additional games.

   This method, `findInsertIndex`, has two parameters: `sortedArr`, which is the sorted array of integers, and `target` which is the integer value you'd like to insert. You've definitely played at least one game of sabacc, so you know that `sortedArr.length` $\geq$ `1`. The method `findInsertIndex` will return the index at which `target` should be inserted into the array `sortedArr` in order for the array to remain sorted. If the value of `target` already appears in the array, then it should simply return the value of the index at which it appears.

   For example, suppose that `sortedArr` is [1, 2, 5, 9, 11, 15].

   (a) If `target` is 7, the array with 7 in it would be [1, 2, 5, 7, 9, 11, 15], so the number 7 is at index 3.

   (b) If `target` is -5, the array with -5 in itwould be [-5, 1, 2, 5, 9, 11, 15], so the number -5 is at index 0.

   (c) If `target` is 5, the array with 5 in it would be [1, 2, 5, 9, 11, 15] (remember, no repeats) so the number 5 is at index 2

   (d) If `target` is 20, the array with 20 in it would be [1, 2, 5, 9, 11, 15, 20], so the number 20 is at index 6.

   For this problem, you do not need to actually insert `target` into `sortedArray`, just return the index at which it should be inserted as described above. *Do not step through the elements of the array one-by-one to solve this problem!* Instead, use the more clever and faster method discussed in class.

Submission:

- For the coding portion:
  - If you don't have a GitHub account please create one: `https://github.com/`. Then please go to `https://classroom.github.com/classrooms/127782556-cs245-veomett-`

and find your name on the list of available names in order to link your GitHub account. The starter code for Problem 1 is here: `https://classroom.github.com/a/OpsdXWfL`

  – Use the starter code available at the GitHub link; you need only edit the `findInsertIndex` method in the InsertIndex.java file on your own computer. If you wish, you may edit the `main` method as well. *Do not edit any other files, or the file structure.*

  – Make sure to commit and push often, both for your own sanity and also so that the teaching staff can see your progress, know that you have academic integrity, and help you more easily when you have questions.

- Be sure to answer the questions in the main method comments. You can write your answers there, in comments. Be brief but complete in your answers.

- All of your solutions must be your work (no copy/paste/edit), although you may look to other sources if you get stumped. For your own learning benefit, put in a good solid couple of hours of effort before deciding that you're truly stumped! If any part of your solution is derived from another work (such as StackOverflow, ChatGPT, a friend in the class, etc) **you must cite that work**, include a discussion about what you used, and include a link to the source.

- Submit your repository link on Canvas.

Grading:

1. 25 points

   (a) test cases: 12 points total (there will be 6 total test cases, each for +2 points. Four of the test cases are already available to you).

   (b) questions: 4 points total (+2 for correctly answering each of the two questions asked in the comments)

   (c) style: 9 points total(+5 for correctly implementing the faster method described in class, +2 for good coding style (like appropriate variable names), +2 for javadoc documentation of your method)