CS 245: Lab 2
Due: 11:59 PM Monday, Feb 5

1. For this lab, you will write an algorithm that will find the maximum sum of a contiguous sub-array within a given array. This question (and its solution!) were discussed in our first lecture. Here is a recap:

   - Suppose you have an array `arr` with n numbers in it. (For this assignment, those numbers will be integers)
   - You want to find a contiguous segment of the array whose sum is as large as possible.

   We are trying to maximize

   $$\sum_{i=t_1}^{t_2} \texttt{arr[i]}$$

   Note: Here you are allowed to have an *empty sum*. An empty sum has value 0.

   To complete this Lab, you will write a method `maxSubArray` which will return the maximum sum of a contiguous segment of an array. Here are a few examples:

   (a) Suppose that `arr` is [4, -5, 3, 2, -4, 8, -3]. Then the maximum sum of a contiguous subarray is 9 (= 3+2+(-4)+8)

   (b) Suppose that `arr` is [-2, -1]. Then the maximum sum of a contiguous sub-array is 0 (not summing any entries . . . the empty sum).

   (c) Suppose that `arr` is [3, 2, 5]. Then the maximum sum of a contiguous sub-array is 10 (=3+2+5).

   For this problem, you can assume that `arr` has at least one entry (that is, `arr.length` is at least 1). *Please implement the fast algorithm that we discussed in class.* In particular, your solution should be $O(n)$ in order to receive full credit.

Submission:

- For the coding portion:
  - The starter code for Problem 1 is here: `https://classroom.github.com/a/tr6MJHm7`
  - Use the starter code available at the GitHub link; you need only edit the `maxSubArray` method in the MaxSubArray.java file on your own computer. I recommend you also implement two different `maximum` methods (one for two parameters, and one for three parameters). 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 implement an algorithm that is $O(n)$ time. Briefly explain why your method is $O(n)$ time in the comments.

- 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: 15 points total (there will be 5 total test cases, each for +3 points. Three of the test cases are already available to you).
   (b) $O(n)$ explanation: 3 points total (+3 for correctly explaining why your solution is $O(n)$ complexity)
   (c) style: 7 points total(+3 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)