Department of EECS
York University

Term: Winter 2020
Instructor: Andy Mirzaian

EECS2011ON: Fundamentals of Data Structures

Assignment 1

Due: 9 pm, Friday, January 24, 2020

- **This is an individual assignment; you may not share code with other students.**
- **Read the course FAQ on how to submit assignments and the marking scheme.**
- **Print your name, EECS account and student ID number on top of EVERY file you submit.**

Further Instructions:

- This assignment consists of 3 problems and is designed to let you practice on arrays, (nested) loops, and some elementary constructs of the Java programming language.
- You can also access the following code templates:
  ArraySqeeze.java , ArrayLongestPlateau.java , TestHelper.java .
- You will design more classes/fields/methods and test cases of your own as required. Use informative Javadoc-style comments as well as inserted comments to further explain your code.
- In addition to the required .java files, you may also submit a file **a1sol.pdf** in which you provide further explanations and illustrations (particularly pertaining subtle parts of your codes) to help the reader better understand the underlying ideas behind your codes. It may also include (by cut-and-paste) output results of your programs for each of the 3 problems.

Problem 1 (25 points): Squeeze an array:
Implement the *squeeze()* method in the provided *ArraySqueeze* class so that it performs as indicated in the comment. The *main()* method in this class runs some test cases on *squeeze()*. You should also add a few nontrivial and interesting test cases of your own at the end of *main().*

Problem 2 (35 points): Longest plateau:
Consider a non-empty int array *ints*. A contiguous subarray *ints[start .. start + len -1]* (with starting index *start* and length *len*) is called a *flat* if all elements of that subarray are equal. Furthermore, such a subarray is called a *plateau* if it is flat and each of the elements *ints[start -1]* and *ints[start + len]* that immediately proceed/succeed the subarray are either nonexistent (i.e., out of array's index range) or are strictly smaller than the elements of the subarray. Your task includes the design of a public static method *longestPlateau(int[] ints)* that returns (a compact description of) the longest plateau of the input array *ints.* You may break ties arbitrarily if *ints* has more than one longest plateau. The return type should be a 3-element array representing *{ value , start , len }* : The first indicates common element value *ints[start]* of the plateau, the second its starting index, the third its length.

Implement the *longestPlateau()* method in the provided *ArrayLongestPlateau* class so that it performs as indicated. The *main()* method in this class runs some test cases on *longestPlateau()*. You should also add a few nontrivial and interesting test cases of your own at the end of *main().*

Problem 3 (40 points): Overlap/enclosure counts of windows:
Given an array of windows in the plane, we want to count how many overlapping and how many enclosing pairs of windows there are (without double counting).

A window is the region of the plane enclosed by a rectangle with sides parallel to the x and y axes. A window can be abstracted as an object with the 4 fields of doubles: *(left, right, bottom, top).* These fields should satisfy the *invariant: left < right* and *bottom < top*. The window is the Cartesian product of the x-interval [*left, right*] and the y-interval [*bottom, top*], i.e., the set of points (*x,y*) in the plane such that *left* $\leq x \leq$ *right* and *bottom* $\leq y \leq$ *top* .

Consider a pair of windows w_0 and w_1. We say these two windows *overlap* if they have a common interior point (just touching boundaries is not enough). We say w_0 *encloses* w_1 if no part of w_1 is outside w_0. (Note that the first relationship is symmetric but the second is anti-symmetric.)

Design a class called *Window* with the following fields and methods:
(Note: you should not confuse this with the Java API class java.awt.Window.)

- Protected fields *left, right, bottom, top*.
- A *constructor* that throws an exception named *InvalidWindowException* if the given 4 fields do not satisfy the above invariant.
- Public *getters* and *setters* for these fields. The setters also throw *InvalidWindowException* if invalid field value is attempted to be set.
- Boolean instance method *encloses(Window w)* returns true if and only if the instance window encloses the argument window w.
- Boolean instance method *overlaps(Window w)* returns true if and only if the instance window overlaps the argument window w.
- Static method *overlapCount(Window[] windows)* returns the number of (unordered) overlapping pairs of windows in the input array windows.
- Static method *enclosureCount(Window[] windows)* returns the number of (ordered) enclosing pairs of windows in the input array windows.
- main() method runs some interesting test cases of the above methods.