EE 422C
Assignment 1
Arrays and Sorting

For this project, you will write a few functions that create and manipulate sorted arrays of integers. This is an individual assignment.

You will not be provided with a "test program", only a written specification. Part of this assignment is to reasonably interpret/understand the specification, as well as designing and *validating* a solution.

For this particular assignment, you may post ideas about testing to Piazza, in as much detail as you wish. Do not post testcase code, and do not post solution code (except a line, perhaps, to ask about syntax, for example). Your testcase ideas should not hint at the solution algorithm.

Instructions:
Design, implement and validate a Java class called *SortTools* that has public static methods for each of the following functions. You may not use the static methods in java.util.Arrays or Java Collections in your solution, except, if you want, Arrays.copyOf and Arrays.copyOfRange. (Both these copy methods are not essential for the solution.)

- **boolean** isSorted(int[] x, int n) – returns **true** if the first *n* elements of *x* are sorted in non-decreasing order, returns **false** otherwise. The contents of *x* are not modified by this function.
    - The worst case time complexity of this function should be O(n)

- **int** find(int[] x, int n, int v) – if *v* is contained within the first *n* elements of *x*, return an index of *v* (i.e. return *k* where x[k] == v and *k* < n – if there are multiple such values for k, then your function must return one of those values, but may return any k < n where x[k] == v). If *v* is not contained within the first *n* elements of *x* return -1. The contents of *x* are not modified by this function.
    - **Precondition**: *x* must be sorted in non-decreasing order. If this precondition is not satisfied, then the result of calling *find* is undefined and may produce catastrophic behavior.
    - The worst case time complexity of *find* should be O(log n)

- **int[]** insertGeneral(int[] x, int n, int v) – return a newly created array of integers with the following properties.
    - The contents of the new array include the first *n* elements of x and the value v.
    - The contents of the new array are sorted in non-decreasing order.
    - If the first n elements of *x* contains at least one copy of thevalue v, then the new array will contain *n* values (i.e. do not add another copy of *v* if it is already in *x*).

- If *x* does not contain *v* then the new array will contain *n+1* values (i.e., the original contents plus *v*).
- **Precondition**: *x* must be sorted in non-decreasing order. If this precondition is not satisfied, then the result of calling this function is undefined and may produce catastrophic behavior.
- The worst case time complexity should be O(n)

- **int** insertInPlace(int[] x, int n, int v) – modify *x* so that it satisfies the following properties:
  - If *x* contained at least one copy of the value v within the first n elements (array elements x[0..n-1]) prior to executing the function, then x is not modified, and *n* is returned by the function.
  - Otherwise the modified array will contain the first *n* values from the original array and the value v. These n+1 values sorted in non-decreasing order and stored in array elements x[0..n]. The function returns n+1. The remaining elements of x (i.e. the elements after index n) are "don't care".
  - **Precondition**: x.length is at least n + 1. If this precondition is not satisfied, then the result of calling this function is undefined and may produce catastrophic behavior.
  - **Precondition**: *x* must be sorted in non-decreasing order. If this precondition is not satisfied, then the result of calling this function is undefined and may produce catastrophic behavior.
  - The worst case time complexity should be O(n).

- **void** insertSort(int[] x, int n) – sort the first *n* elements of *x* in non-decreasing order. You must obtain the following time complexity bounds:
  - In the general case, your function must have $O(n^2)$ time complexity (i.e. scale no worse than quadratically in n).
  - In the special case that *x* is nearly sorted, your function must have O(n) time complexity. The formal definition of nearly sorted is: $x[k] \leq x[k+1]$ for all $0 \leq k < n$, except for at most C values of k (where C is a constant).
  - Informally, your function must have linear time complexity if all the elements in x are sorted with just one value out of place.

More Instructions:

- You must use good style, including indentation, variable and method names, spacing, and comments.
- You must ensure that your program compiles and runs with Java 8.
- You must not delete or modify the package statement from the template .java file.
- When done, a good idea is to check out all files to a clean directory, and compile and run it on the Linux command line.
- When you are done, commit your SortTools.java file to Canvas.