EE 422C
Assignment 1

Arrays and Sorting

Due Tuesday 1/30/2018 at 10:00pm
75 points

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

You will be provided with a "test program" with a couple of simple test cases. 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 test case code, and do not post solution code (except a line, perhaps, to ask about syntax, for example). Your test case 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.) In all the cases below, there will be no illegal inputs (you can assume all the parameters that are passed to the methods are valid). For example, n will be >= 0 and n <= nums.length.

- `boolean isSorted(int[] nums, int n)`

  Returns **true** if the first *n* elements of *nums* are sorted in non-decreasing order, returns **false** otherwise. The contents of *nums* are not modified by this function.
  - The worst case time complexity of this function should be O(n).
  - **Precondition**: n <= nums.length, n >= 0.
  - If nums.length = 0 or n = 0, return false.
  - **Precondition**: nums will not be null.

- `int find(int[] nums, int n, int v)`

  If *v* is contained within the first *n* elements of *nums*, return an index of *v* (i.e. return *k* where nums[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 nums[k] == v). If *v* is not contained within the first *n* elements of *nums* return -1. The contents of *nums* are not

modified by this function.

- **Precondition**: *nums* 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.
- **Precondition**: nums will not be null.
- **Precondition**: n <= nums.length, n > 0.
- The worst case time complexity of *find* should be O(log n)

- `int[] insertGeneral(int[] nums, 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 nums and the value v.
- The contents of the new array are sorted in non-decreasing order.
- If the first n elements of *nums* contain at least one copy of the value v, then the new array will contain *n* values (i.e. do not add another copy of *v* if it is already in *nums*).
- If the first n elements of *nums* do not contain *v* then the new array will contain *n+1* values (i.e. the original contents plus *v*).
- **Precondition**: *nums* 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.
- **Precondition**: n <= nums.length, and n >= 0.
- **Precondition**: nums != null.
- **Precondition**:nums.length > 0.
- The worst case time complexity should be O(n).

- `int insertInPlace(int[] nums, int n, int v)`

Modify *nums* so that it satisfies the following properties:

- If *nums* contained at least one copy of the value v within the first n elements (array elements nums[0..n-1]) prior to executing the function, then nums 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 are sorted in non-decreasing order and stored in array elements nums[0..n]. The function returns n+1. The remaining elements of nums (i.e. the elements after index n) are “don’t care”.
- **Precondition**: nums.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**: *nums* 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.
- **Precondition**: n > 0. n < nums.length.**Precondition**: nums.length > 0.
- The worst case time complexity should be O(n).

- `void insertSort(int[] nums, int n)`

  Sort the first *n* elements of *nums* 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 *nums* is nearly sorted, your function must have O(n) time complexity. The formal definition of nearly sorted is: nums[k] ≤ nums[k+1] for all 0 ≤ 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 nums are sorted with just one value out of place.
  - You have choices of algorithm based on the above criteria alone, but we want you to implement insertion sort.
  - **Precondition:** nums.length > 0, n > 0, n <= nums.length.
  - **Precondition:** nums != null.

Testing:
You have been provided with a file containing 2 JUNIT test cases, and a testing script. You must ensure that your code runs with these test cases and script on the ECE Linux 64-bit machines, such as Kamek. Instructions for running the script are provided separately.

More Instructions:
- You must use good style, including indentation, variable and method names, spacing, and comments.
- You must complete the header.
- You must not delete or modify the package statement from the template
  - .java file.
- You must make sure that the test cases you are given pass with your code, using JUNIT testing.
- You must ensure that your program compiles and runs on kamek.ece.utexas.edu.
- Check out all files to a clean directory, and compile and run it on the Linux command line and/or in Eclipse/other IDE.
- When you are done, turn in only your SortTools.java file to Canvas (you don't need to turn in any other files).