Data Structures & Algorithms @ DE
Laboratory 4a: Add-ons to IntegerSet

Objectives

The objectives of this laboratory are:

(a) to make the following modifications to the original `IntegerSet`:

i. modify the `insert` method to handle overflowing the underlying array, as described below

ii. adding a public method, `isSorted`, that tests if the `IntegerSet` is sorted in ascending order

iii. adding a private method `shuffle`, that randomizes the order of the set, as described below

iv. adding a new sort, `quickSort`, along with its associated helper methods (`partition` and the overloading of `quickSort` itself) and modifying the `sort` method for its possible use

v. edit the client class `Tester` as needed for use of the modified `IntegerSet`

What to submit

Upon completion of your laboratory, please turn in the two files:

- `IntegerSet.java`
- `Tester.java`

Part 1: Preparation

Preparation: First, create a copy of your directory `Lab4Arrays` and change the copy's name to `Lab4ArraysA` within your course (Data Structures) directory.

You are to modify the class `IntegerSet` in that directory according to the specifications given below, in addition to modifying the class `Tester` as needed for the updated `IntegerSet` class.

NOTE: do NOT modify your original `IntegerSet` class that you handed in earlier!

Part 2: Modifying the `IntegerSet` Class

Program the following additions/modifications to the `IntegerSet` class:

(a) `A.insert(x)` - if `A` does NOT already contain x, this method now does the following:

- if the underlying array is not yet full, adds x to the set as before
- if the underlying array IS full, modifies it by increasing its capacity by 50 percent, then adds x to the set

(b) `quickSort()` - a public method to sort the elements of the set using the quickSort algorithm. The code for this method is as follows (you may copy it directly into your class):

```
public void quickSort(){
    // shuffle the array to mitigate possibility of worst case
    shuffle();

    // call the sort on the entire set
    quickSort(0, size-1);
}
```

(c) `quickSort(first, last)` - a private method that sorts elements from index `first` to index `last` inclusive using the quick sort algorithm. Pseudocode:

```
1. if last is greater than first
     1a. let p be equal to partition(first, last)
     1b. quick sort the array from first to (p-1)
     1c. quick sort the array from (p+1) to last
```

(d) `partition(first, last)` - a private method that does the heavy-lifting in the quick sort algorithm. Essentially modifies the elements array so that the items are rearranged into a half that consists of values LESS than some pivot value, and another half that are greater than or equal to the pivot value,then places the pivot value between the two halves, and returns the index where the pivot value was placed.Pseudocode:

```
1. store the value at index first in pivot
2. set left to (first+1) and right to last
3. while left is less than right
  3a. increase left until the value at left is greater than pivot
      OR left equals right
  3b. decrease right until the value at right is less than
      or equal to pivot OR right equals left
  3c.  swap the values at left and right
4. swap the values at right and pivot
5. return right
```

Part 3: Testing correctness

Modify `Tester` to test the correctness of your newly modified `IntegerSet` class.

Part 4: Documentation

Document all public methods using Javadoc. Modify the existing Javadoc and add inline comments to explain your code.