Data Structures & Algorithms @ DE
Laboratory 4: Arrays

Objectives

The objectives of this laboratory are:

(a) to (re)familiarize yourself with Java arrays

(b) to make use of an interface, and

(c) to implement a set of integers in a Java class that supports six basic operations, namely

   i. testing emptiness
   ii. testing membership
   iii. insertion
   iv. removal
   v. sorting
   vi. printing

What to submit

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

- `IntegerSet.java`
- `lab4.out`

Part 1: Preparation

Preparation: First, create a directory named `Lab4Arrays` within your course (Data Structures) directory. Download the file `lab4files.jar` from the DE course site and place it in the directory that you just created. This file is a **J**ava **AR**chive file (a *jar* file) and contains the necessary files for this lab. Navigate in your command line interpreter to that directory and then you can use the following command to extract the files (you can delete the jar file after you do this successfully):

```
jar xvf lab4files.jar
```

The files included in this download are:

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

You are to complete the class `IntegerSet` according to the specifications given below. The class `Tester` has been completed already and *should not be modified*.

Part 2: Interface `OrderedSet`

The `OrderedSet` interface defines a set of behaviors for an ordered set.

(a) Modify `IntegerSet.java` to explicitly state that it contains all behaviors defined in `OrderedSet.java`. Compile your code.

(b) Temporarily comment out one of the methods and recompile your code. Observe what happens.

(c) Uncomment the method, compile your code and proceed to the next part.

Part 3: `IntegerSet` Class

The class `IntegerSet` supports the following six operations for an integer set `A`, an integer `x`, and an integer `algNum`:

(a) `A.isEmpty()` - returns `true` if `A` is empty and `false` otherwise.

(b) `A.contains(x)` - returns `true` if `A` contains `x` and `false` otherwise.

(c) `A.insert(x)` - inserts `x` into `A`.

(d) `A.remove(x)` - removes `x` from `A`.

(e) `A.sort(algNum)` - sorts all the elements of `A` using the sorting algorithm defined by `algNum` as specified below.

(f) `A.print()` - prints all the elements of `A`.

Fields: This class has two fields (instance variables), `elements` and `size`, as shown below. The field `elements` refers to an `int` array that stores the integers of a set. The field `size` is an `int` variable to keep track of the size of a set (i.e. the number of integers in a set). The size of the array `elements` is always fixed by the constant `MAX_SIZE`.

```
private int elements[];
private int size;
```

For this implementation, you must store integers in *continguous* cells starting from the first cell (whose index is 0). This means there should be no gaps in the array after inserting or removing integers from the set. Also, the elements of a set must all be *distinct*; that is, there should not be more than one integer of the same value.

Methods: The constructor and the methods `isEmpty()` and `print()` are already completed. You are to implement the remaining four methods, in addition to the `private` methods `selectionSort` and `insertionSort`. Note also that, because the elements must always be distinct, the method `insert()` should check, before attempting to insert an integer `x` into a set `A`, whether or not `x` already exists in `A`; if so, there is no need to actually insert `x` into `A`. For the method `sort(algNum)`, use the following encoding to choose which algorithm is utilized:

0 denotes `selectionSort`

1 denotes `insertionSort`

See the pseudocode below for the insertion sort algorithm: you must implement it on your own (i.e. without relying on any external library), in addition to coming up with appropriate pseudocode and implemetation of the selection sort algorithm.

Insertion Sort: Please use the following pseudocode to implement the insertion sort:

```
Algorithm for InsertionSort(A):

Input: An array A of n comparable elements.
Output: The array A with elements rearranged in nondecreasing order.
     for k from 1 to n-1 do
       insert A[k] at its proper location within A[0], A[1], ..., A[k]
```

Part 4: Testing correctness

The class `Tester` is a test driver application. Use this to test the correctness of your `IntegerSet` class. Again, the class `Tester` is already completed and should not be modified. Save a snapshot of your test run into a text file named `lab4.out`.

Part 5: Documentation

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