CSC 364 Homework #3 Instructor: Jeff Ward
Assigned Tuesday, February 14, 2017 Due: 11:59pm, Friday, February 24
Covers: Sorting Worth 40 points

Comparing different sorting algorithms

You will obtain runtimes for three sorting algorithms:

- Insertion sort as implemented in the textbook
- Heap sort as implemented in the textbook using a Heap object
- Heap sort implemented in-place in the array that will be sorted, implemented by yourself

The textbook's code for Insertion sort and Heap sort is on Blackboard, as is a "test driver" program for running the tests and printing the results. Pseudocode for the in-place Heap sort is provided below, but you will have to write the Java code yourself. Implement this sort so that it works on an array of int.

The test driver will create four different arrays of int values, as follows:

- a sorted array of size 100,000
- a random array of size 100,000
- a sorted array of size 200,000
- a random array of size 200,000
- a sorted array of size 400,000
- a random array of size 400,000

It will then run each of the three sorting methods on copies of each of the four data sets, test the results for correctness, and display the runtimes. The test program writes its output both to the console and to a file named timings.txt.

Your in-place heapSort method should be static method and should take one parameter: an array of ints. The heapSort method should be in a class called InPlaceIntHeapSort.

Heap sort (in-place):
The textbook provides an implementation of heap sort, but it allocates a Heap object that has the same size as the original array. For this assignment, you are required to implement the heap sort so that it operates ***in-place***: It should use only a small number of local variables for its work space, and should not allocate any significantly large collections. (Do not allocate a Heap object!) Here is pseudocode for an in-place heap sort:

```
heapSort(array): // Sorts array in-place
Let n be the length of the array

// Part I:  Turn the array into a max-heap
for (i = 1 to n - 1)
   "Sift up" element at index i

// Part II:  Repeatedly extract the max element from the heap
for (i = n – 1 downto 1)
   swap array[0] with array[i]
   The last index into the heap is now i - 1
   "Sift down" element at index 0
```

Example output

Here is the output that I obtained when I ran the test program on my solutions.

```
Insertion sort runtime on ordered array of length 100000:  3
milliseconds
Insertion sort verified.

Insertion sort runtime on random array of length 100000:  3201
milliseconds
Insertion sort verified.

Object-based heap sort runtime on ordered array of length
100000:  46 milliseconds
Object-based heap sort verified.

Object-based heap sort runtime on random array of length 100000:
78 milliseconds
Object-based heap sort verified.

In place heap sort runtime on ordered array of length 100000:
20 milliseconds
In place heap sort verified.

In place heap sort runtime on random array of length 100000:  14
milliseconds
In place heap sort verified.

Insertion sort runtime on ordered array of length 200000:  0
milliseconds
Insertion sort verified.

Insertion sort runtime on random array of length 200000:  5000
milliseconds
Insertion sort verified.
```

```
Object-based heap sort runtime on ordered array of length
200000:  44 milliseconds
Object-based heap sort verified.

Object-based heap sort runtime on random array of length 200000:
66 milliseconds
Object-based heap sort verified.

In place heap sort runtime on ordered array of length 200000:
21 milliseconds
In place heap sort verified.

In place heap sort runtime on random array of length 200000:  26
milliseconds
In place heap sort verified.

Insertion sort runtime on ordered array of length 400000:  1
milliseconds
Insertion sort verified.

Insertion sort runtime on random array of length 400000:  20006
milliseconds
Insertion sort verified.

Object-based heap sort runtime on ordered array of length
400000:  92 milliseconds
Object-based heap sort verified.

Object-based heap sort runtime on random array of length 400000:
173 milliseconds
Object-based heap sort verified.

In place heap sort runtime on ordered array of length 400000:
40 milliseconds
In place heap sort verified.

In place heap sort runtime on random array of length 400000:  51
milliseconds
In place heap sort verified.
```

What to turn in:
Submit the following files on Blackboard:
InPlaceIntHeapSort.java, Timings.txt

Something to think about after the assignment is done:
How did the runtimes increase as n increased? Did the runtimes match the patterns that we would expect based on the big-O analyses that we studied? Did you expect your Heap sort to be faster than the one from the textbook? Was it faster?