**CSC 364 Assignment # 3**

**Total Points: 40**

**Comparing different sorting algorithms**

You will obtain runtimes for three sorting algorithms:

a) **Insertion Sort** (as implemented in the textbook)

b) **Heap Sort** (as implemented in the textbook using a Heap object)

c) **In-Place Heap Sort** (implemented in-place in the array that will be sorted, implemented by

yourself)

***InsertionSort,java*** and ***HeapSort,java*** is on Canvas. There is also a ***TestSorting.java*** file

included on Canvas, a “test driver” program for running the tests and printing the results. You

will write a new java class, ***InPlaceIntHeapSort.java***. The three algorithms will then be

compared against one another.

**Comparison of Algorithms**

The test driver *TestSorting.java* will create the following six different **int** arrays:

1. a sorted array of size 100,000

2. a random array of size 100,000

3. a sorted array of size 200,000

4. a random array of size 200,000

5. a sorted array of size 400,000

6. a random array of size 400,000

It will then run each of the three sorting methods on copies of each of the six 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*.

**In-Place Heap Sort**

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***.

Recall that, inplace implies that an algorithm 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!)

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.

You should name your class ***InPlaceIntHeapSort.java***. Your *InPlaceIntHeapSort* class should have a public static method **heapsort()**, and should take one parameter: an array of int.

**Pseudocode for 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)

Compare parent\_of\_i and "Sift up" item at index i

// Part II: Repeatedly extract the max element at 0-th position from heap

for (i = n – 1 down to 1)

// Move the largest item from 0-th to the end i-th index

swap array[0] with array[i]

// Note: The last index into the heap is now i – 1

Compare r/l-child and "Sift down" element at index 0

**Example output**

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

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.

Done!

**Submission**

Submit the following files on Canvas: 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?