Computer Science 282

October 28, 2019 **Programming Assignment #3** Due: November 13, 2019

You are to write static methods that implement insertion sort, various quicksort algorithms, and heapsort by completing the class and methods shown below.

The faster your sorts, the better. If your program runs too slowly, it probably means there is something wrong. I will eventually create a test program that will show the speeds of your sorts but I strongly recommend that you write your own test program so that you can start testing your sorts immediately.

A few reminders and rules (ask about these if you have questions!): All of your sorts should work on 0-element arrays. Shift rather than swap during insertion and heap sorts. Be sure your various quicksort methods are making recursive calls to the proper quicksort method. Since there are multiple quicksorts it is easy to accidentally make the wrong recursive calls. You need to write an insertion sort since you will need it in your quicksort driver methods.

The cutoff parameter in your quicksorts represents the smallest sized partition that will be recursively sorted. For example, if cutoff = 10 then a partition with 10 elements will be recursively sorted while a partition of size 9 or less will not. Be sure you get this exactly right. It is easy to be off by 1.

To be sure the stack does not overflow in your quicksorts write your recursive quicksort methods with a loop that: (1) selects the pivot, (2) partitions the array by calling a partition method, (3) makes the recursive call on the smaller partition(s), and (4) sets parameter values so that the loop simulates the second/third recursive call on the larger/largest partition. Don't forget that the recursive quicksort's will look a little different depending on which partition algorithm you use.

You are to write each of the following quicksort algorithms:

QuickSort1: outside-in partition, random pivot
QuickSort2: left-to-right-1-pivot partition, random pivot
QuickSort3: left-to-right-2-pivots, 2 random pivots
QuickSort4: outside-in partition, pivot = a[lf]
QuickSort5: left-to-right-1-pivot partition, pivot = a[lf]
AlmostQS1: Exact same driver as QuickSort1 except the call to InsertionSort is commented out
AlmostQS2: Exact same driver as QuickSort2 except the call to InsertionSort is commented out
AlmostQS3: Exact same driver as QuickSort3 except the call to InsertionSort is commented out

Also, write two heapsort methods. HeapSortTD builds the heap top down (the slow way) and HeapSortBU builds the heap bottom up (the fast way).

You should email me your classes in a single file named prog3.java in the usual way. The subject of your email should be "Roger Federer – prog 3 – 2:00" if your name is Roger Federer and you are in the 2:00 class. Adjust your name and class time accordingly. Also, turn in a hardcopy of your program along with a copy of the output it produces on my final test program. The usual warnings apply about following all instructions, keeping up with changes, etc.

Below are some of the driver method headings and some comments so that you'll be ready for the test program when it arrives. Note that the driver programs require a size parameter (named *n*) and the quicksort drivers also require a cutoff parameter as described above.

```
// To get random numbers you can either use Math.random() in which case you might
// have to import java.math.* or you can import java.util.Random to create a Random
// object

import java.math.*;  // maybe not needed?

// or

import java.util.Random;


// use this class to contain everything related to your sorts

class ArraySorts {

   // Some sample driver method headers

   public static void insertionSort(int a[], int n) { // Insertion Sort

   public static void QuickSort1(int a[], int n, int cutoff) {

   public static void HeapSortBU(int a[], int n) { // heapsort with linear buildheap

   ...

   public static String myName() {

}

// use this class to return two values in the outside-in and the 2-pivot
// partition methods

class pair {
       public int left, right;

       public pair(int left, int right) {
              this.left = left;
              this.right = right;
       }

       // some getters below
}
```