COMP 410 - Spring 2017
Programming Assignment 2

Due back by **9:05 am on February 20**

For this assignment, you are to

1) Implement the *priority queue* ADT (specified in interface form below) as a sorted array;
2) use your priority queue implementation to sort an array; and
3) compare the efficiency of this sorting algorithm with another one implemented using the `java.util.PriorityQueue` implementation of priority queue.

The **priority queue interface** that you are to implement is as follows:

```
public interface PQ<C extends Comparable<?super C>> {
  public boolean isFull();
  public boolean isEmpty();
  public void insert(C data); //Precondition: Not full
  public C min(); //Precondition: Not empty
  public C deleteMin(); //Precondition: Not empty
}
```

Below we provide a **skeleton for your implementation** of this interface; complete this skeleton by filling in the three methods `insert()`, `min()`, and `deleteMin()`.

```
public class PQasSortedArray<C extends Comparable<?super C>> implements PQ<C> {
  private C[] arr; // store the elements in the priority queue IN SORTED ORDER
  private int currentSize;
  public PQasSortedArray(int size){
    arr =  (C[]) new Comparable[size];
    currentSize = 0;
  }
  public boolean isFull(){return currentSize == arr.length;}
  public boolean isEmpty(){return currentSize == 0;}
  public void insert(C data){ //fill in the details here}
  public C min(){//fill in the details here}
  public C deleteMin(){//fill in the details here}
}
```

Note. You may ***not*** use Java's built-in `sort` methods within your method implementations — explicitly implement all sorting routines on your own.

Using, and evaluating, your priority queue. You are to use your priority queue implementation in a `main` class that

1) declares and fills in an array of a (specified) constant size `N` with random Doubles — read up on `java.util.Random` to learn how this is to be done.
2) **Sorts** this array using your priority queue implementation. Let `arr1[]` denote an array to be sorted; the following code sorts it:

```
for (int i=0; i < arr1.length; i++)pq1.insert(arr1[i]);
for (int i=arr1.length-1; i >=0 ; i--)arr1[i] = pq1.deleteMin();
```

   You should determine the amount of time your code takes to sort the array. You can time the performance of any piece of code in the following manner:

```
long startTime, endTime;
startTime = System.nanoTime();
//
//CODE TO BE TIMED GOES HERE
//
endTime = System.nanoTime();
System.out.println("Took " + ((endTime - startTime)/1000000) + " time units");
```

3) Repeat step 2 above (sort and time) for the same array, this time using the `java.util.PriorityQueue` implementation of priority queues. The sorting step would look like this:

```
for (int i=0; i < arr1.length; i++)pq2.offer(arr1[i]); // "offer()" is insert()
for (int i=arr1.length-1; i >=0 ; i--)arr1[i] = pq2.poll();  // "poll()" is deleteMin()
```

4) Compute the running time of both sorting algorithms for a range of values of `N`. Based on these computed values, estimate BigOh running times for both approaches. If they are different, in bigOh terms and/ or absolute ("real time") terms, can you provide an explanation as to why?

Submission instructions. You should upload the following three files in a `.zip`file to Sakai

1) A file titled `PQasSortedArray.java`, containing your implementation of the priority queue interface
2) A file titled `Tester.java`, containing your "main" class that generates random arrays, sorts this array (twice – once using your priority queue, once Java's), and times these sorts.
3) A pdf file listing, in tabular form, running times of both sorts for different values of `N`. Explain how these values were obtained (e.g., *"I ran the code 12 times for each value of* `N`*, discarded the largest and smallest running times, and took the average of the remaining 10"*). This pdf file should also contain your estimates of the BigOh running times of both sorts, and if different, your explanations as to why this is so.