QUEENS COLLEGE Department of Computer Science
CSCI 313 Practice problems on Priority Queues and Heaps

Instructor: Alex Ryba

These are some of the solutions to practice problems. Not all problems have solutions here.

Solutions to some older problems might not make use of generics. Generics are now required in this course.

Problem 1 A generic priority queue is implemented as a heap so that n entries of comparable type K occupy elements $1, 2, 3, \ldots (n+1)$ of an array *data* in the heap. Usual heap order and heap shape requirements are in force. (Note this uses slightly different array elements from the implementation described in class and in the textbook.) A skeleton for the class is as follows:

```
public class HeapPriorityQueue  // class title line to be completed as (a)
{  private K data[];  private int size = 0;  private int capacity = 100;
   // constructor  to be coded as (b)
   public void insert(K x) throws Exception {
      if (size >= capacity - 2) throw new Exception("Priority Queue Full");
      data[++size] = x;
      bubbleUp(size);
   }
   public K removeMin() throws Exception {  // omitted
   private void swapData(int n, int m) { // omitted, swaps entries n and m
   private void bubbleUp(int n) {   // omitted  to be coded as (c)
   private void bubbleDown(int n) { // omitted
}
```

(a) **Write the complete class title line, including a clause that makes it implement a *PriorityQueue*.**
Answer:

```
public class HeapPriorityQueue<K extends Comparable<K>> implements PriorityQueue<K>
```

(b) **Implement a constructor with no arguments.**
Answer:

```
public HeapPriorityQueue() {
   data = (K[]) new Comparable[capacity];
}
```

(c) **Implement the method bubbleUp.**
Answer:

```
private void bubbleUp(int n) {
   if (n <= 1) return;
   K node = data[n];
   K parent = data[n / 2];
   if (node.compareTo(parent) >= 0) return;
   swapData(n, n / 2);
   bubbleUp(n / 2);
}
```