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:

```java
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:

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

(b) Implement a constructor with no arguments.

Answer:

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

(c) Implement the method bubbleUp.

Answer:

```java
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);
}
```

Problem 2  The standard \( interface PriorityQueue \) and \( class HeapPriorityQueue \) include the following code.

```java
interface PriorityQueue<K extends Comparable<K>> {
    public void add(K x) throws Exception;
    public K removeMin() throws Exception;
}
```
Write complete code for a non-standard `HeapPriorityQueue` method `removeSecondMin` that efficiently removes and returns the second minimum element from the data structure. Your solution can make use of the methods `bubbleUp`, `bubbleDown` and `swapData` but must not apply the constructor or either of the methods `add` and `removeMin`.

For example, if the array contains the following elements:

```
1, 2, 3, 4, 5,
```

It should be changed by `removeSecondMin` to

```
1, 4, 3, 5,
```

Inefficient and excessively complicated solutions will lose points.

**Answer:**

```java
class HeapPriorityQueue<K extends Comparable<K>> implements PriorityQueue<K> {
    private K data[]; // the root is stored at index 0 in the array
    private int size;
    private int capacity;
    // constructors, add and removeMin method code omitted
    /* methods with titles
    void bubbleUp(int n)
    void bubbleDown(int n)
    void swapData(int n, int m)
    are available, but the code is not shown here */
}

public K removeSecondMin() throws Exception {
    if (size <= 1)
        throw new Exception("Priority Queue too short");
    int place = 1;
    if (size > 2 && data[2].compareTo(data[1]) < 0)
        place = 2;
    swapData(place, --size);
    bubbleDown(place);
    return data[size];
}
```