Problem 1  A generic priority queue is implemented as a heap so that $n$ entries of comparable type $K$ occupy elements 1, 2, 3, ..., $(n + 1)$ 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
        public K removeSecondMin() 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.
    (b) Implement a constructor with no arguments.
    (c) Implement the method bubbleUp.
```

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;
}

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 */
}
```

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:

Problem 3  (a) What two properties of a binary tree make it a heap?
(b) Give a Java implementation method for the method:

```java
public static <K extends Comparable<K>> boolean hasHeapOrder(BNode<K> r)
```

Here \( r \) is a node in a Binary Tree, so that \( r \) has instance variables \( data, parent, left \) and \( right \). The method should return true if the subtree rooted at \( r \) satisfies the heap ordering requirement.

Problem 4  (a) What two properties of a binary tree make it a heap?
(b) Give either a pseudocode outline (or for extra credit, a Java method) for an algorithm:

```java
public static <K> boolean hasHeapShape(BNode<K> r)
```

That returns true, if the subtree rooted at \( r \) satisfies the heap shape requirement.