Problem Set 10 - Solution

Heap

---

1. Given the following sequence of integers:

```
12, 19, 10, 4, 23, 7, 45, 8, 15
```

   1. Build a heap by inserting the above set, one integer at a time, in the given sequence. Show the heap after every insertion. How many comparisons in all did it take to build the heap?
   2. Perform successive *delete* operations on the heap constructed in the previous step, until the heap is empty. Show the heap after every deletion. How many comparisons in all did it take to perform these deletions?

   SOLUTION

   1. The heaps built one at a time are shown in the following table. The first column is the item that is inserted, the second column is the heap after this item is inserted (indicated by its level order traversal), and the third column is the number of item-to-item comparisons made by the sift up process.

```
Insert       Heap after insertion     Number of comparisons
-------------------------------------------------------------------
12           12                       0
19           19 12                    1
10           19 12 10                 1
4            19 12 10 4               1
23           23 19 10 4 12            2
7            23 19 10 4 12 7          1
45           45 19 23 4 12 7 10       2
8            45 19 23 8 12 7 10 4     2
15           45 19 23 15 12 7 10 4 8  2
```

      Total # of comparisons = 12

   2. The left column shows the heap (level order traversal representation) **after** deleting the top of the heap, and the right column shows the number of item-to-item comparisons made by the sift-down process.

```
Heap after deletion        Number of comparisons
-----------------------------------------------
23 19 10 15 12 7 8 4       4
19 15 10 4 12 7 8          4
15 12 10 4 8 7             4
12 8 10 4 7                4
10 8 7 4                   2
8 4 7                      2
7 4                        1
4                          0
empty                      0
```

      Total number of comparisons = 21

---

2. Suppose we have a (**max**) heap that stores integers. (By contrast, in a "min" heap the key at any node is *less than or equal to* the key at its children, so the *smallest* valued key is at the top of the heap.) Then, given an integer *k*, we would like to print all the values in this heap that are *greater than k*. Implement the following method to do this.

```
public void printGreater(int[] H, int n, int k) {
   /* your code here */
}
```

   H is the array storage for the max heap, and *n* is the number of entries in the heap.

   Note: The challenge is to do this efficiently. Use the heap order to reduce the number of entries of the heap to be examined.

   SOLUTION

```
public void printGreater(int[] H, int n, int k) {
   recursivePrintGreater(H, n, k, 0);
}

private void recursivePrintGreater(int[] H, int n, int k, int  rootIndex) {
   if (rootIndex >= n) return; // out of bounds
   if (H[rootIndex] <= k) return; // all values <= k in this sub-heap
   System.out.println(H[rootIndex]); // print root value
   recursivePrintGreater(H, n, k, 2*rootIndex+1); // left sub-heap
   recursivePrintGreater(H, n, k, 2*rootIndex+2); // right sub-heap
}
```

---

3. Consider a max heap that only supports the operations **insert**, **deleteMax**, **size**, and **isEmpty**. A client of the heap wants to update the priority of an entry in the heap. Since there is no search operation, the only way to accomplish the update is this:

   - Perform successive **deleteMax** operations until the entry is extracted
   - Update the entry's priority
   - **Insert** the entry, as well as all the other deleted entries back into the heap

   What would be the worst case running time (big *O*) of this update process on a heap with *n* entries?

   SOLUTION

   In the worst case, *n* deletes and *n* inserts would need to be done. The time for the deletes is as follows (approximated for big O):

```
log n + log (n-1) + log (n-2) + .... + 1 = log (n!) = O(n*log n)
```

   The total time for the inserts is also O(n*logn) by a similar argument. The update process is therefore O(n*log n)

---

4. * Suppose you are given two heaps, stored in arrays. Write a method to merge them into a single heap, and return this heap. The original heaps are not modified:

```
public static <T extends Comparable<T>> T[] merge(T[] heap1, T[] heap2) {
  // COMPLETE THIS METHOD
}
```

   SOLUTION

```
public static <T extends Comparable<T>> T[] merge(T[] heap1, T[] heap2) {
    T[] res = (T[])new Object[heap1.length + heap2.length];
    for (int i=0; i < heap1.length; i++) {
        res[i] = heap1[i];
    }
    for (int i=0; i < heap2.length; i++) {
        res[i+heap1.length] = heap2[i];
    }

    // in res, sift up starting with entries copied from the second heap
    for (int s=heap1.length; s < res.length; s++) {
        int k=s;
        // sift up res[k]
        while (k > 0) {
           int p = (k-1)/2;
           if (res[k].compareTo(res[p]) > 0) { // switch
              T temp = res[k]; res[k] = res[p]; res[p] = temp;
              k = p;
           } else {
              break;
           }
        }

    }
```

```
    return res;
}
```