1 ▸ As an implementation of a MIN-PRIORITY-QUEUE, consider the *Young Tableau* introduced in ***Problem 6-3*** on pages 167→168 of our text. Solve ***Problem 6-3***.

SOLUTION: ***a*** One of many Young Tableaux for these numbers is

2	3	8	15
4	8	∞	∞
5	14	∞	∞
12	∞	∞	∞

b Denote the Young Tableau $Y[1..m,1..n]$. Since the elements in the top row are sorted, $\infty = Y[1,i] \le Y[1,i+1], 1 \le i < n$, it must be the case that $Y[1,i] = \infty, 1 \le i \le n$. Since the elements in each column are sorted, $Y[i,j] \le Y[1,j] = \infty, 1 \le i \le m, 1 \le j \le n$, so Y only contains ∞s, and must be empty. Since the elements in the bottom row are sorted, $Y[m,i] \le Y[m,i+1] < \infty, 1 \le i \le i+1$, it must be the case that $Y[m,i] < \infty, 1 \le i \le n$. Since the elements in each column are sorted, $Y[i,j] \le Y[m,j] < \infty, 1 \le i \le m, 1 \le j \le n$, so Y doesn't contain any ∞s, and must contain mn finite elements.

c To simplify the algorithm, we assume that Y has a column n+1 and a row m+1, all of whose entries are ∞.

```
Extract-Min(Y)
    return Extract-Min-Aux(1,1)
Extract-Min-Aux(i_lo,j_lo)
    Min ← Y[i_lo, j_lo]
    if i_lo = m ∧ j_lo = n then Y[i_lo, j_lo] ← ∞
    else if Y[i_lo + 1, j_lo] ≤ Y[i_lo, j_lo + 1]
      then Y[i_lo, j_lo] ← EXTRACT-MIN-AUX(i_lo+1,j_lo)
      else Y[i_lo, j_lo] ← EXTRACT-MIN-AUX(i_lo,j_lo+1)
    return Min
```

For every recursive call of EXTRACT-MIN-AUX, either i_{lo} increases by 1 or j_{lo} increases by 1. Since every invocation of EXTRACT-MIN-AUX takes time in $O(1)$, and ultimately $i_{lo} + j_{lo} \le m + n$, it follows that the execution time of EXTRACT-MIN is in $O(m+n)$.

d To simplify the algorithm, we assume that Y has a column n+1 and a row m+1, all of whose entries are ∞.

```
INSERT(x,Y)
    Insert-Aux(x,1,1)
INSERT-AUX(x,i_lo,j_lo)
    if x > Y[i_lo, j_lo + 1] ≥ Y[i_lo + 1, j_lo]
      then INSERT-AUX(x,i_lo+1,j_lo)
    else if x > Y[i_lo + 1, j_lo] ≥ Y[i_lo, j_lo + 1]
      then INSERT-AUX(x,i_lo,j_lo+1)
```

e Since EXTRACT-MIN and INSERT each take time in $O(m+n)$, we start with an empty Young Tableau and INSERT each of the n^2 numbers, in time in $O(n^3)$, and then EXTRACT-MIN each of the n^2 numbers, also in time in $O(n^3)$.

f The function MEMBER($x,i_{lo},i_{hi},j_{lo},j_{hi}$), which tests whether or not x belongs to $Y[i_{lo}..i_{hi}, j_{lo}..j_{hi}]$, is invoked by MEMBER(1,m,1,n), and it eliminates one column (the left one) or one row (the left one) with each recursive call. It does this by comparing x to the bottom left element of $Y[i_{lo}..i_{hi}, j_{lo}..j_{hi}]$.s

```
MEMBER(x,i_lo,i_hi,j_lo,j_hi)
    if i_lo = i_hi ∧ j_lo = j_hi
      then return x==Y[i_lo,j_lo]
    else if x=Y[i_hi,j_lo]
      then return true
    else if x>Y[i_hi,j_lo]
      then return MEMBER(i_lo,i_hi,j_lo+1,j_hi)   ▸ remove left column
      else return MEMBER(i_lo,i_hi-1,j_lo,j_hi)   ▸ remove bottom row
```

****

2 ▸ Show that for any n and any k such that $n \ge k \ge 1$ and for any array $A[1..n]$ of integers, the k^{th} largest element of A can be found in worst-case time in $O(n + k \lg n)$.

Solution:

```
Build-Max-Heap(A)                          O(n)
for i ← 1 to k-1 do EXTRACT-MAX(A)         O(klgn)
    return A[1]                            O(1)
```

****

3 ▸ Assume you want to INSERT a new element into a heap on $n \ge 1$ elements. For the following, show exact answers (with floors $\lfloor\ \rfloor$ and ceilings $\lceil\ \rceil$ as needed).

a How many elements, $f(n)$, are on the path from where the new element in INSERTed to the root, counting the root but not counting the new element. For example, 1 comparison is needed for n=1.

b Since the $f(n)$ elements on the path from where the new element in INSERTed to the root are sorted, propose and analyze a faster technique that the one from class and the text to locate **where** to INSERT the new element.

c With your faster location technique from part ***b***, analyze the worst-case time to actually INSERT the new element.

SOLUTION:

a $\lfloor \lg(n+1) \rfloor$

b $\lceil \lg(\lfloor \lg(n+1) \rfloor)+1 \rceil = \lceil \lg \lceil \lg(n+2) \rceil \rceil = \lceil \lg\lg(n+2) \rceil$

c $\lceil \lg(n+2) \rceil$

****

4 ▸ **Problem *6-1*** on pp. 166-167

SOLUTION: Posted at http://mitpress.mit.edu/algorithms/

****

5 ▸ **Problem *6-2***, parts ***a***, ***b***, ***c*** and ***d*** on pg. 167

SOLUTION: ***a*** The element at position *i* has a parent at $\left\lfloor \frac{i-2}{d}+1 \right\rfloor$ and the j^{th} child, $1 \le j \le d$, is at $d(i-1)+j+1$.

b $\Theta(\lg_d n)$

c HEAP-EXTRACT-MAX of the text is the same, but MAX-HEAPIFY has to compare the element to as many of its *d* children as exist. The running time becomes $\Theta(d \log_d n)$.

d MAX-HEAP-INSERT from the text works. Running time is still in $\Theta(\log_d n)$.

****

6 ▸ The *right path* of a node in a leftist tree is the path encountered by following the RIGHTCHILDren until a node with a RIGHTCHILD = NULL is found.

a Prove that the right path from any node in a leftist tree is a shortest path to a node with a NULL child.

b Prove that a leftist tree τ must contain at least $2^{dist(root(\tau))+1}-1$ nodes.

SOLUTION: ***a*** The proof is by induction on the length of the right path. Let ν be the root of the tree. If the length of the right path is 0, then RIGHTCHILD(ν)=Null, and is trivial right path. So it is a shortest path from ν to a node with a NULL child, having 1 node and 0 edges.

If the length of the right path is >0, then by the way the tree was constructed, RIGHTCHILD(ν) and LEFTCHILD(ν) are leftist trees, and *dist*(*root*(RIGHTCHILD(ν))) ≤ *dist*(*root*(LEFTCHILD(ν))). By the induction hypothesis, the right path of RIGHTCHILD(ν) is a shortest path of RIGHTCHILD(ν), by the way the tree was constructed, the right path of RIGHTCHILD(ν) is no lomnger than any path of LEFTCHILD(ν).