Andrew Letz
CIS 313
Professor Proskurowski
Written Homework #2

Question 1

6.1-1: In a heap with height *h* where the root node is height 1,

the minimum # of elements is: 2^{h-1}

the maximum # of elements is: $2^{h} - 1$

6.1-4: The largest element in a min-heap would reside anywhere on the last row (height h) or row h - 1. This is because for the heap-property to remain upkept in a min-heap, the largest element can not have any children. To not have any children in a tree means that it is a terminal/leaf node, which in a heap is restricted to the final two rows.

6.1-5: If the array is sorted in "traditional" order (values go from minimum to maximum), then yes, it is a min-heap. This is because it would also maintain the heap-property in that every index A[i] <= A[i+1].

6.1-7: The consequence of this fact on the bottom-up BUILD-HEAP method is that you only have to run MAX-HEAPIFY on every element in the subarray A[A.length / 2], which greatly simplifies the problem. This is because it would be redundant to run MAX-HEAPIFY on a leaf node who has no children.

Question 2

A = <12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1>, A.length / 2 = 6

Step	Function call	Array A after calling function
1	MIN-HEAPIFY(A,6)	<12, 11, 10, 9, 8, 1, 6, 5, 4, 3, 2, 7>
2	MIN-HEAPIFY(A,5)	<12, 11, 10, 9, 2, 1, 6, 5, 4, 3, 8, 7>
3	MIN-HEAPIFY(A,4)	<12, 11, 10, 4, 2, 1, 6, 5, 9, 3, 8, 7>
4	MIN-HEAPIFY(A,3)	<12, 11, 1, 4, 2, 10, 6, 5, 9, 3, 8, 7> <12, 11, 1, 4, 2, 7, 6, 5, 9, 3, 8, 10>
5	MIN-HEAPIFY(A,2)	<12, 2, 1, 4, 11, 7, 6, 5, 9, 3, 8, 10> <12, 2, 1, 4, 3, 7, 6, 5, 9, 11, 8, 10>

6	MIN-HEAPIFY(A,1)	<1, 2, 12, 4, 3, 7, 6, 5, 9, 11, 8, 10> <1, 2, 6, 4, 3, 7, 12, 5, 9, 11, 8, 10> <- final

Question 3

A = [1 … n] an array of numbers passed into BUILD-MIN-HEAP to produce H

H = [1 … n] is a sorted binary heap

Q = [1 … n] is a FIFO queue

```
while notEmpty(H) {
      Enqueue(Q, DeleteBest(H));
}
```
(sorts H into Q)

Invariant: At the start of each iteration of the loop, H[1] (the root of H) is greater than all elements Q[1 … n].

Initialization: Prior to the first iteration, Q is empty, and therefore H[1] will be greater than all elements of Q.

Maintenance: Following the heap property, the element deleted and returned from the method DeleteBest(H) will be H[1] and consequently the smallest element of that heap. If every element [1 … n] in Q were previously deleted in the same manner, then each of those elements would be smaller than H[1], meaning the root of H is greater than any given element in Q. Assuming DeleteBest(H) shifts each element of the remaining heap into its correct place, the invariant is established for the next iteration of the loop.

Termination: At termination, each element will have already been deleted from the heap, meaning the body of the while loop will not be executed (this would invalidate the invariant if H were empty). Following the correctness of the maintenance step, every element in H (and consequently A) will have been deleted and then enqueued into Q in increasing order. This shows that Q contains all elements of A sorted in increasing order, from head-to-tail.

Question 4

Example of functionality:

A = [2, 4, 6, 8, 10, 12, 14]

PQDecreaseKey(A, 3, 3) => [-1, 4, 3, 8, 10, 12, 14]

```
def PQDecreaseKey(array A, int i, int v) {
      A[i] = A[i] - v         # Decrement the first el. by the offset

      # If there are no predecessors we don't need to loop
      if (i == 1) {
            return
      }

      int index = i
      while (True) {
            # Get the parent of A[index]
            parent = floor(index / 2)

            # If this value is less than one, the index must be the root,
            meaning there is no parent left to "improve"
            if (!(parent < 1)) {
                  A[parent] = A[parent] -  v
                  index = parent
                  continue
            }
            break
      }
      return
}
```

The loop only reaches the **break** statement once the result of parent (floor of index divided by 2) is less than one, a.k.a when we get to the root node. This means that the loop will iterate a maximum of the binary heap's height, which is $O(\log_2 n)$.

(I have working Python code for this method if wanted)