CS360 - Assignment 3

6.1-1 What are the minimum and maximum number of elements in a heap of height h?

Since a heap is a *nearly complete* binary tree, all the levels above level h, i.e. the last level, will be full. Thus for $h-1$ levels (assuming $h > 0$) we will have the number of elements as

$$n_{h-1} = \sum_{i=0}^{h-1} 2^i = 2^{(h-1)+1} - 1 = 2^h - 1$$

Thus the heap with the *minimum* number of elements will only have a *single* element at level $h \Rightarrow n_h = 1$. Hence the *minimum* total number of elements in a heap of height h is

$$n_{h-1} + n_h = 2^h - 1 + 1 = 2^h$$

The heap with the *maximum* number of elements will have the entire last level *full* $\Rightarrow n_h = 2^h$. Hence the *maximum* total number of elements in a heap of height h is

$$n_{h-1} + n_h = 2^h - 1 + 2^h = 2 \cdot 2^h - 1 = 2^{h+1} - 1$$

(which is exactly the number of elements in a complete binary tree of height h).

6.1-4 Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

The max-heap property states that *every* parent node is greater than or equal to *both* children. Thus by induction, the smallest node must be a *leaf* node since it will have to be smaller than *all* nodes in the path to the root. If it were not a leaf node, then it would have at least one child which would imply that it is greater than its child (since the elements are distinct) contradicting the fact that it is the *smallest* element.

6.4-3 What is the running time of heapsort on an array A of length n that is already sorted in increasing order? Decreasing order?

If the array is in *increasing* order, then it is the *worst case* for BUILD_MAX_HEAP (i.e. the largest values are at the bottom of the tree). Hence BUILD_MAX_HEAP will take roughly the upper bound running time of $\approx 2n = O(n)$. However the element extraction phase will still require $O(n \lg n)$ operations so the total running time will be $O(n) + O(n \lg n) = O(n \lg n)$.

If the array is in *decreasing* order, then it is the *best case* for BUILD_MAX_HEAP (i.e. the array is already a max-heap). Therefore it will only take $\lfloor \frac{n}{2} \rfloor$ checks

(with no swaps) to verify. Therefore it will take $O(\lfloor \frac{n}{2} \rfloor) = O(n)$. However the element extraction phase ***will still*** require $O(n \lg n)$ operations so that the total running time will still be $O(n) + O(n \lg n) = O(n \lg n)$.

Even though the best case (decreasing) will run slightly faster in practice due to less work to BUILD_MAX_HEAP, the extraction phase asymptotically dominates the algorithm giving roughly the same *asymptotic* running time.