Heapify and Heapsort

Heapify the list $L_1, L_2, \ldots, L_n$:

Step 1: Load elements into balanced tree structure of height $\lceil \log_2 n \rceil$

Step 2: Find the largest element in the tree, then bubble it into the root position by a sequence of swaps from its current position to the root position.

Step 3: Heapify the subtrees hanging from the root. Since the largest element is already in the root position, the entire tree is a heap.

Analysis: In the course of heapifying, every element in the original tree gets bubbled upwards some number of levels. Since there are h levels to begin with, each bubbling operation takes at most h swaps. So at most $nh = n\lceil log_2 n \rceil$ swaps are required to heapify a list of n elements.

Heapsort the list $L_1, L_2, \ldots, L_n$:

Step 1: Heafipy the list.

Step 2: Continue to do the following: extract the root element, then adjust the remaining elements to form a heap. Adjusting proceeds as follows: after extracting the root element, the root position is empty. Swap the larger child of the root into the root position, changing the empty position of the tree. Now swap the larger child of the empty position into this position. Keep on going until the empty position is a leaf position.

In the course of extracting elements from the heap, we are emptying the heap in such a way that the elements extracted decrease from largest to smallest. So in the process of extracting from the heap we are ordering the original list in reverse order.

Analysis: each extraction requires no more than h swaps to bubble the empty root position down to a leaf position. So extraction requires at most $nh = n\lceil log_2 n \rceil$ swaps to perform. Combined with Heapify, Heapsort requires at most $2n\lceil log_2 n \rceil$ swaps to order a list of length n.