

# Heapify

## *Asymptotic Run-Time Complexity Analysis*

Given a list of  $n$  items in an array, it is easy to convert the list into a heap in two different ways, top-down and bottom-up. On casual consideration, the two approaches may appear to cost the same. On closer analysis, the bottom-up approach is seen to be asymptotically more efficient.

For the following analysis, assume a full tree of height  $h$ , so  $n = 2^{h+1} - 1$  and  $h = \log_2(n+1) - 1$  where  $n$  is the number of nodes in the tree. (The general case of heapifying a complete tree has a time complexity that is squeezed between those of two full trees.)

### Top-Down Heapify

The heap is grown from left to right in the array, one entry at a time. Each new entry from the remaining portion of the array is added to the heap by shifting it upward into the heap as needed to achieve the heap property.

The  $k^{th}$  node added to the heap may need to be shifted up as many as  $\log_2(k)$  times, so an upper bound on number of shifts is

$$\sum_{k=1}^n \log_2(k) = \log_2\left(\prod_{k=1}^n k\right) = \log_2(n!)$$

and, as we showed earlier in the semester,  $\log(n!) = \Theta(n \log(n))$ .

### Bottom-Up Heapify

Bottom-up heapify proceeds as follows

1. first heapify all the little trees of height 1 at the bottom
2. then heapify the trees of height 2 up one from bottom
3. then heapify the trees of height 3 up one from the previous step
- $\vdots$  and so on
- $h$ . until heapifying the one final tree of height  $h$

Here's a list of the number of trees at each height:

height	num trees
1	$2^{h-1}$
2	$2^{h-2}$
$\vdots$	$\vdots$
$h-1$	2
$h$	1

For each of the  $2^{h-k}$  trees of height  $k$ , up to  $k$  swaps will be required to heapify it (since both of its subtrees are already heaps), so that's an upper bound of  $k2^{h-k}$  swaps for the trees of height  $k$ . Summing all those products for  $1 \leq k \leq h$  gives

$$\sum_{k=1}^h k2^{h-k} = 1 \cdot 2^{h-1} + 2 \cdot 2^{h-2} + \cdots + (h-1)2^1 + h2^0$$

To figure out this sum, multiply it by 2

$$2 \sum_{k=1}^h k2^{h-k} = 1 \cdot 2^h + 2 \cdot 2^{h-1} + 3 \cdot 2^{h-2} + \cdots + (h-1)2^2 + h2^1$$

and take the difference

$$2 \sum_{k=1}^h k2^{h-k} - \sum_{k=1}^h k2^{h-k} = 1 \cdot 2^h + 1 \cdot 2^{h-1} + 1 \cdot 2^{h-2} + \cdots + 1 \cdot 2^2 + 1 \cdot 2^1 - h2^0$$

then rewrite

$$\sum_{k=1}^h k2^{h-k} = 2^h + 2^{h-1} + 2^{h-2} + \cdots + 2^2 + 2^1 - h2^0 = \left( \sum_{k=1}^h 2^k \right) - h$$

and note that

$$\left( \sum_{k=1}^h 2^k \right) = 2^1 + 2^2 + \cdots + 2^h$$

and

$$2 \left( \sum_{k=1}^h 2^k \right) = 2^2 + 2^3 + \cdots + 2^{h+1}$$

and the difference gives (using the same technique as with the original sum)

$$2 \left( \sum_{k=1}^h 2^k \right) - \left( \sum_{k=1}^h 2^k \right) = \sum_{k=1}^h 2^k = 2^{h+1} - 2$$

so, finally,

$$\sum_{k=1}^h k2^{h-k} = \left( \sum_{k=1}^h 2^k \right) - h = 2^{h+1} - 2 - h = \Theta(2^h) = \Theta(n).$$

Note that Sedgewick gets

$$\sum_{k=1}^h (k-1)2^{h-k} = 2^h - 1 - h.$$

Either I'm missing something or his  $k - 1$  is a typo. It's not important in terms of the asymptotic analysis, though. The run-time complexity of bottom-up heapify is shown to be **linear** either way.

### Exercise

Using our understanding of  $\log_2(n!)$  in the top-down analysis is unnecessary. An analysis that is similar to what we did in the bottom-down case applies in the top-down case as well and leads to this

$$\sum_{k=1}^h k2^k$$

expression for the upper bound which is very similar to the one we found in the bottom-down case. Using the same technique as in the bottom down case, show that this sum is proportional to  $n \log(n)$ .