Notes on Binary Heaps

February 15, 2008

(Adapted partially from Cormen *et al*'s *Introduction to Algorithms*).

Suppose we're creating a (Max) heap with n elements to be placed in it. Here's an algorithm

makeheap(h) (h is an array containing n values to be placed in the heap, in arbitrary order)
for i in $\lfloor n/2 \rfloor$ **downto** 1 **do**
  heapify(h,i)
end for

heapify(A,i)
$l = \text{LEFT}[i]; r = \text{RIGHT}[i]$
if ($l \leq |A|$ AND $A[l] > A[i]$) **then**
  largest = l
else
  largest = i
end if
if ($r \leq |A|$ AND $A[r] > A[largest]$) **then**
  largest = r
end if
if (largest $\neq i$) **then**
  swap($A[i], A$[largest])
  heapify(A, largest)
end if

Algorithm 1: Algorithm for (max) heap creation

To prove correctness, we introduce the following *loop invariant:*
At the start of the loop in **makeheap**, each node $i+1, i+2, \ldots, n$ is the root of a max-heap.

At initialization, this is obvious, because every element in the array with index greater than i is a leaf node. To show maintenance of the invariant, we know that i's children l and r are the roots of max-heaps to start. **heapify** only swaps (recursively down) if one is greater than i, thereby maintaining the max-heap property. At termination, $i = 0$, so the node with index 1 is the root of a max-heap (which contains all the elements).

Naive run-time analysis: There are n calls to the $O(\log n)$ **heapify**, so it is $O(n \log n)$. But this $\log n$ is worst case behavior. We can improve on this analysis and get a tighter upper bound.

heapify takes time $O(h)$ when called on a node of height h (that is, when the node is the root of a heap of height h). How many nodes are there of height h? Convince yourself that the answer is $\lceil n/2^{h+1} \rceil$ (for $h = \log n$ this gives 1, for $h = 0$ this gives $\lceil n/2 \rceil$, etc.)

So now suppose we sum over all heights the product of two things: the number of nodes of that height, and the amount of time heapify takes on one call to a node of that height. This should give us the total running time. Working it out:

$$\sum_{h=0}^{\log n} \lceil n/2^{h+1} \rceil O(h) = O\left(n \sum_{h=0}^{\log n} \frac{h}{2^h} \right)$$

Now,

$$\sum_{h=0}^{\log n} \frac{h}{2^h} < \sum_{h=0}^{\infty} h \left(\frac{1}{2}\right)^h = 2$$

How do we get this last step? From $\sum_{h=0}^{\infty} hx^h = \frac{x}{(1-x)^2}$ for $x < 1$, which can be obtained by differentiating both sides of the sum of an infinite geometric progression.

Putting this into the equation above, we get a better bound for the running time: $O(n)$!