Notes on Heaps

1 heapsize(A)- -

There is a subtlety in decrementing the heapsize. We’ll take heapsize(A) = 70 as an illustrative example. After heapsize(A) — we have heapsize(A) = 69. The subtlety is in the definition of leftchild. Recall (rightchild is similar)

\[\text{leftchild}(i) = \begin{cases} 2i & \text{if } 2i \leq \text{heapsize}(A) \\ \text{NIL} & \text{if } 2i > \text{heapsize}(A) \end{cases}\]

Consider leftchild(35). When heapsize(A) = 70 the definition gives leftchild(35) = 70. But after decrementing to heapsize(A) = 69 the definition gives leftchild(35) = NIL. Effectively 70 has disappeared! Say, for example, we do MAX – HEAPIFY(35). Recall LARGEST was the largest value of A(35) and A of its children. But it now has no children! So LARGEST = 35 and MAX – HEAPIFY stops.

This shows the importance of having both length and heapsize. The value A(70) is still in the array and can be useful later. But it is no longer in the heap!

2 BUILD-ARRAY

There was an arithmetic error in class. Consider BUILD–ARRAY(A) with the length \(n = 2^k - 1\). The rows have lengths \(2^0, 2^1, 2^2, \ldots, 2^{k-2}, 2^{k-1}\). The number of flips is 1 for the \(2^{k-2}\) nodes in the penultimate row; 2 for the \(2^{k-3}\) in the row above that, continuing up to \(k - 1\) flips for the root. The total number of flips (call it \(TF\)) is therefore

\[TF = 2^{k-2}(1) + 2^{k-3}(2) + \ldots + 2^0(k - 1)\]

It is convenient to take out a common factor of \(2^{k-1}\) (In class this was \(2^{k-3}\) by error.) Then we have

\[TF = 2^{k-1} \left[ \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \ldots + \frac{k-1}{2^{k-1}} \right]\]

There is actually an exact expression for the bracketed term. But we use that the infinite sum

\[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \ldots = 2\]
Therefore the bracketed sum is less than 2. So

$$TF \leq 2^{k-1} \cdot 2 = 2^k = n + 1$$

So $TF = O(n)$. (This “big oh” will be formally defined in week three.) The key conclusion: BUILD - ARRAY is a linear time algorithm!