Lecture 12: Friday January 31, 2003

today and next class: heapsort analysis

- will show these run time results:
  - time in $O(n \log n)$ proof is easy
  - WC $\Theta(n \log n)$
  - BC
    - all keys equal: $\Theta(n)$
    - all keys distinct: $\Theta(n \log n)$

```
heap shaped tree: positions          depth

                 1                     0
              /     \
           2           3               1
         /   \        / \
        4     5      6   7             2
       / \   / \    /
      8   9 10 11  12                  3
```

some heap shaped binary tree facts

- def'n: *depth* of a node — number of links in path to root
- def'n: *depth of tree* — max'm depth of node, say d
- depth of node at position j? — $\lfloor \lg j \rfloor$

  why? — positions on path to root: $j \quad \lfloor j/2 \rfloor \quad \lfloor j/4 \rfloor \quad \ldots \quad 1$
- so …sum of depths of all nodes? — $\sum_{j=1}^{n} \lfloor \lg j \rfloor \in \Theta(n \lg n)$
- number of nodes at depth t?
  - if $t < d$ (so not bottom): 2^t
  - if $t = d$ (so bottom): $\leq 2^t$
- number nodes at bottom $\leq \lceil n/2 \rceil$

  proof: number nodes not at bottom $\quad 1 + 2 + \ldots + 2^{d-1}$

  number nodes at bottom $\quad \leq 2^d$

recall heapsort

```
proc. Heapsort(A)  * at start of line 3, A[1..j] is heap
1  Build-Max-Heap(A)
2  for j <- length[A] downto 2
3    do exchange A[1] <-> A[j]
4       heapsize[A] <- heapsize[A]-1
5       Max-Heapify(A,1)
```

heapsort runtime in $O(n \log n)$

- phase 1: buildheap
  - MH from positions $n/2$, $n/2 - 1$, ..., 2, 1
  - each takes $O(\lg n)$ time — why?
  - total time in $O(n \lg n)$ — why?
- phase 2: remove max, swap, and MH
  - for heapsize $n - 1$, $n - 2$, ..., 2, 1
  - each takes $O(\lg n)$ time — why?
  - total time in $O(n \lg n)$ — why?
- total heapsort runtime in $O(n \lg n)$ ☺

heapsort best case time, all keys distinct, in $\Omega(n \log n)$

- to simplify argument somewhat,
  suppose $n = 2^k - 1$ (so 2^{k-1} nodes on bottom)
- after 1st $\lceil n/2 \rceil = 2^{k-1}$ keys sorted
  - removed from initial heap: largest $\lceil 2^{k-1} \rceil$ keys
  - remaining heap: smallest $2^{k-1} - 1$ keys
- in original heap, for nodes with largest 2^{k-1} keys:
  - if node at bottom depth of heap colour node red
  - if node not at bottom depth colour node blue
- blue nodes form binary tree
- by above, $\leq 2^{k-2}$ red nodes, so $\geq 2^{k-2} > n/4$ blue nodes
- how did each blue node leave? moved up to root
- so ...
  - time for first $\lceil n/2 \rceil$ extractions ...
  - $\geq$ time to move all blue keys to root ...
  - in $\Omega($ sum of depths of blue nodes $)$
  - in $\Omega(\sum_{j=1}^{n/4} \lfloor \lg j \rfloor) = \Omega(n \log n)$ ☺

conclusions: heapsort run time

- WC in $O(n \log n)$
- BC, keys distinct, in $\Omega(n \log n)$
- so every case (keys distinct) in $\Omega(n \log n)$
- all keys equal: in $\Theta(n)$ exercise ☺