# Lecture 13: Friday October 5, 2001

## today

- heapsort (conclusion)
  - best case in $\Theta(n \lg n)$
  - improvement: modify Max-Heapify
  - in practice? quicksort is better
- priority queue
- quicksort [CLRS Ch. 7]

## announcements

- assignment 4: due Wednesday October 17

**binary tree**

each node has at most two children

**lemma**

in a binary tree with $n$ nodes,
at most $\lceil n/2 \rceil$ nodes are on the bottom

**example**

**proof (outline)**

- suppose $t$ nodes at bottom level
- so at least $\lceil \dfrac{t}{2} \rceil$ nodes at next level
- so at least $\lceil \dfrac{\lceil t/2 \rceil}{2} \rceil \;=\; \lceil \dfrac{t}{4} \rceil$ nodes at next level
- so at least $\lceil \dfrac{\lceil t/4 \rceil}{2} \rceil \;=\; \lceil \dfrac{t}{8} \rceil$ nodes at next level
- so ...
- so at least $\lceil \frac{t}{2} \rceil \;+\; \lceil \frac{t}{4} \rceil \;+\; \lceil \frac{t}{8} \rceil \;+\; \ldots \;+\; \ldots 2+1 \geq t-1$ other nodes

**another lemma**

in a binary tree with $n$ nodes,
sum of depths of nodes $\in \Omega(n \lg n)$

**example**

**proof (outline)**

- sum of depths minimimum $\Rightarrow$ all non-bottom levels full
- with each non-bottom level full
  - for $n = 2^k - 1$, this implies all levels full
    - $k = \lg(n+1) > \lg n$
    - $2^{k-1} > n/2$
    - sum $= (k-2)2^{k-1} + 2 > (\lg n - 2)\, n/2$
  - for $2^k - 1 < n < 2^{k+1} - 1$, bottom level may not be full
    - $k > \lg(n+1) - 1 > \lg n - 1$
    - $2^{k-1} > n/4$
    - sum $> (k-2)2^{k-1} + 2 > (\lg n - 3)\, n/4$
  - so for all $n$, sum $\in \Omega(n \lg n)$

## heapsort best case

- suppose $n = 2^k - 1$ (so $2^{k-1}$ nodes on bottom)
- after $2^{k-1} = \lceil n/2 \rceil$ extractions
  - largest $\lceil n/2 \rceil$ keys removed
  - remaining heap: $\lfloor n/2 \rfloor$ keys
- in original heap
  - colour largest $2^{k-1}$ keys: red
  - recolour all red keys **not** at bottom level: blue
- red keys form binary tree
- by lemma, at most $2^{k-2}$ red keys ...
- so at least $2^{k-2}$ blue keys
- how did blue keys leave? must have moved up to root!
- so ...
  - time for first $\lceil n/2 \rceil$ extractions
  - greater than time to move all blue keys to root
  - $\in \Omega($ sum of depths of blue nodes $)$
  - $\in \Omega(n \lg n)$
- conclusion: heapsort run time in $\Theta(n \lg n)$

## a heapsort improvement

```
modified Max-Heapify(A) [McDiarmid/Reed and others]
1 temp <- A[1]    (*save root key; consider A[1] empty
2 trickledown     (*empty key at root, so no swapping
3 x <- where empty key ends up (*leaf location
3 A[x] <- temp
4 bubbleup(A[x])
```

## example

## improvement? consider average case

- after Max-Heapify, original root ends up at/near bottom
- Max-Heapify
  - data moves: $\approx 3 \lg n$ (3 per interchange)
  - comparisons: $\approx 2 \lg n$ (larger child; parent)
- modified Max-Heapify
  - data moves: $\approx \lg n$ (no interchange necessary)
  - comparisons: $\approx \lg n$) (larger child)
- modified Heapsort: $\Theta(n \lg n)$ but better constant
- heapsort in practice? not usually ☺

**priority queue**

- abstract data type (defined by operations, not implementation)
- operations
  - initialize (insert all keys at once)
  - insert
  - extract max
  - remove max
  - increase key

**priority queue implementation?** **heap (of course!)**

- initialize? BuildHeap $\Theta(n)$
- insert? bubble up $\Theta(\lg n)$
- extract max? A[1] $\Theta(1)$
- remove max? Max-Heapify $\Theta(\lg n)$
- increase key? bubble up $\Theta(\lg n)$

## quicksort: sorting meets divide and conquer

- idea
  - pick one key
  - compare to others: partition into smaller/larger sublists
  - recursively sort two sublists
- details: next class

☺ ☺ ☺ **Thanksgiving weekend** ☺ ☺ ☺