Lecture 10: Wednesday Sept 26, 2001

today

- Chapter 4: recurrences (conclusion)
  - iterated substitution
  - recursion trees
  - master theorem
- Chapter 6: heapsort

announcements

- assignment 3 due next Monday in class
- midterm: next Wednesday in class

recall

$$T(n) = \begin{cases} aT(\frac{n}{b}) + n^c & \text{if } \ n \geq b \\ \text{bounded} & \text{if } \ n < b \end{cases}$$

- if $\lg_b a > c$ then $T(n) \ \in \ \Theta(\ n^{\lg_b a}\ )$
- if $\lg_b a = c$ then $T(n) \ \in \ \Theta(n^{\lg_b a} \lg n)$
- if $\lg_b a < c$ then $T(n) \ \in \ \Theta(n^c)$

master theorem

assume

$$T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & \text{if } \ n \geq b \\ \text{defined} & \text{if } \ 0 \leq n < b \end{cases}$$

where $\quad a \geq 1,\, b > 1 \quad$ and $\dfrac{n}{b}$ can be $\lfloor\dfrac{n}{b}\rfloor$ or $\lceil\dfrac{n}{b}\rceil$

conclude

let $\alpha = \lg_b a \quad \alpha^- < \alpha \quad \alpha^+ > \alpha$

- $f(n) \in O(\, n^{\alpha^-})$? then $T(n) \ \in \ \Theta(\, n^\alpha\,)$
- $f(n) \in \Theta(\, n^\alpha)$? then $T(n) \ \in \ \Theta(\, n^\alpha \lg n\,)$
- $f(n) \in \Omega(\, n^{\alpha^+})$? and
  for $c < 1$, $\forall n \geq n_0$, $af(n/b) \leq cf(n)$? then $T(n) \ \in \ \Theta(\, f(n)\,)$

some examples ...

$$T(n) = \begin{cases} 8T(n/2) + n^3 + n^2(\lg n)^5 & \text{if } \ n \geq 2 \\ T(1) & \text{if } \ n = 1 \end{cases}$$

$f(n) \in \Theta(n^3) = \Theta(n^{\lg_b a})$ $\qquad T(n) \in \Theta(n^3 \lg n)$

$$T(n) = \begin{cases} 4T(n/2) + n^3 & \text{if } \ n \geq 2 \\ T(1) & \text{if } \ n = 1 \end{cases}$$

$f(n) \in \Theta(n^3) \subseteq \Omega(n^{\lg_b a} + \varepsilon)$ and
$af(n/b) = 4(n/2)^3 = (1/2)n^3 \leq cn^3$ $\qquad T(n) \in \Theta(n^3)$

$$T(n) = \begin{cases} 10T(n/3) + 6n \lg^3 n + n^2 & \text{if } \ n \geq 2 \\ T(1) & \text{if } \ n = 1 \end{cases}$$

$f(n) \in \Theta(n^2) \subseteq O(n^{\lg_b a - \varepsilon})$ $\qquad T(n) \in \Theta(n^{\lg_3 10})$

$$T(n) = \begin{cases} 7T(n/2) + n^2 \lg n & \text{if } \ n \geq 2 \\ T(1) & \text{if } \ n = 1 \end{cases}$$

$f(n) \in \Theta(n^2 \lg n) \subseteq O(n^{\lg_b a - \varepsilon})$ but
$af(n/b) \not< cf(n)$ for $c < 1$ $\qquad$ master theorem does not apply

exercise: in the last example, show that $T(n) \in O(n^{\lg_2 7})$

upcoming sorting topics

heapsort [CLRS Ch. 6]

- heapsort: a data structure algorithm
  - proc. heapify (almost-heap to heap)
  - proc. buildheap (bottom-up, top-down)
  - correctness
  - time: $\Theta(n \lg n)$
- priority queue (abstract data type)
  - insert, remove max, increase-key

quicksort [CLRS Ch. 7]

- quicksort: a divide and conquer algorithm
  - proc. partition
  - correctness
  - time: WC $\Theta(n^2)$ AC $\Theta(n \lg n)$
  - improvements (randomized; median of 3; stack depth)

sorting lower bound [CLRS Ch. 8.1]

- comparison based: time $\Omega(n \lg n)$

(max-)heap data structure

- unless otherwise stated, heap means max-heap
- comparable ($<$, $=$, $>$) keys stored in array $A[1 \ldots n]$
- array considered an implicit binary tree:
  - $A[2j]$ is left child of $A[j]$
  - $A[2j+1]$ is right child of $A[j]$
  - $A[j/2]$ is parent of $A[j]$
- keys satisfy (max-)heap* property: key(parent) $\geq$ key(node)

example

```
  j       1  2  3  4  5  6  7  8  9 10
A[j]      4  1  3  2 16  9 10 14  8  7          heap?  no
A[j]     16 14 10  8  7  9  3  2  4  1          heap? yes
```

heapsort algorithm

```
length[A]:   number keys in A                    (never changes)
heapsize[A]: number keys in heap rooted at A[1]    (decreases)

proc. Heapsort(A)       invariant: start of 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)

proc. Max-Heapify(A,j)      (*turn almost-heap into heap*)
(* precond'n:  tree* rooted at A[j] is almost-heap
(* postcond'n: tree* rooted at A[j] is heap
(* *up to position heapsize[A]
(* 'trickle-down' idea: if A[j] < some child then
(*    interchange A[j] with larger child
(*    continue trickle-down from swapped child
 1 lc <- left(j)   rc <- right(j)
 3 if lc <= heapsize[A] and A[lc] > A[j]
     then largest <- lc else largest <- j
 6 if rc <= heapsize[A] and A[rc] > A[largest]
     then largest <- rc
 8 if largest <> j
 9   then exchange A[j] <-> A[largest]
10         Max-Heapify(A,largest)
```