Lecture 11: Friday Sept 28, 2001

today

- Ch. 6: heapsort
  - proc Max-Heapify
  - proc BuildHeap
  - proc Heapsort
  - time complexity

announcements

- assignment 3 Monday
  - collaboration reminder
  - problems in order
  - name, id, acknowledgement
- midterm: next Wednesday in class

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

(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
```

Max-Heapify: from almost-heap to heap

- heap: array (implicit binary tree), and
  keys satisfy (max-)heap property: parent $\geq$ child
- almost heap: heap, except root may not satisfy h.p.

```
Max-Heapify(A,j)
(* turn almost-heap (root j) into heap via ‘trickledown’
(* see previous lecture or text for more detail

node <- A[j]
while node < larger child
  do  interchange
```

heapsort

```
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)
```

Build-Heap (bottom-up; Floyd's alg'm)

- level -1 nodes (bottom) are heaps (size 1) so
- level -2 nodes are almost-heap roots: Max-Heapify now
- level -3 nodes are almost-heap roots: Max-Heapify ...

```
heapsize[A] <- length[A]
for j <- length[A]/2 downto 1
  do Max-Heapify(A,j)
```

Other-Build-Heap (top-down; Williams' alg'm)

- A[1] is heap (size 1) so
- 'bubble-up' A[2], now A[1..2] heap so
- 'bubble-up' A[3], now A[1..3] heap . . .

```
Other-Build-Heap(A)
for j <- 2 to length[A]
  do Bubble-Up-Insert(A,j)
```

analysis

- correctness: loop invariants
- space: in place
- running time
  - Max-Heapify $\in O(\lg n)$
  - Build-Heap $\in \Theta(n)$
  - Heapsort (BC,WC) $\in \Theta(n \lg n)$
- improvements
  - Max-Heapify
    * trickledown empty location
    * insert key into (bottom) location
    * bubble-up