

## Analysis of Algorithms CS 3343 Lecture Thirteen

Prof. William Winsborough  
November 6, 2008

## Business

- Reading: 6.5, ch8
- Homework 8 due Thursday 11/13
  - 6.5-1, 6.5-2, 6.5-5, 6.5-7
- Recall:
  - Homework 7 due Tuesday 11/11
    - 6.1-1, 6.1-2, 6.1-3 (hint: use induction on the height of the subtree), 6.1-7
    - 6.2-3, 6.2-4, 6.2-5
    - 6.4-2, 6.4-3
  - Midterm 2 will be Thursday November 20
    - Tuesday November 18 will be devoted to review

6 November 2008

Winsborough CS 3343 Lecture 13

2

## Recall: Max-Heapify

- Max-heap property:
  - For all  $i$ ,  $A[\text{parent}(i)] \geq A[i]$
- Max-Heapify
  - Inputs:  $A, i$
  - Assumes that binary trees rooted at  $\text{Left}(i)$  and  $\text{Right}(i)$  are max-heaps
  - But  $A[i]$  may be smaller than its children
  - The routine moves such values down in the tree, making the tree rooted at  $i$  a max-heap
- Refer to p.130 for routine
- $O(\lg n)$

6 November 2008

Winsborough CS 3343 Lecture 13

3

## Building a Heap

- Build the heap bottom up by using Max-Heapify (see p.133)
- Tight upper bound:  $O(n)$ 
  - $O(n \lg n)$  is clear
  - An  $n$ -element heap has height  $\lfloor \lg n \rfloor$  and at most  $\lceil n/2^{h+1} \rceil$  nodes of height  $h$ 
    - Recall: Height of a node is the number of edges in the longest downward path from the node to a leaf
- Refer to p.135
- Work 6.3-3: there are at most  $\lceil n/2^{h+1} \rceil$  nodes of height  $h$

6 November 2008

Winsborough CS 3343 Lecture 13

4

## Heapsort

Heapsort( $A$ )

1. Build-Max-Heap( $A$ )
  2. for  $i \leftarrow \text{length}[A]$  downto 2
  3. do exchange  $A[1] \leftrightarrow A[i]$
  4.  $\text{heap-size}[A] \leftarrow \text{heap-size}[A] - 1$
  5. Max-Heapify( $A, 1$ )
- $O(n \lg n)$ : Build-Max-Heap is  $O(n)$  and each of the  $n - 1$  calls to Max-Heapify is  $O(\lg n)$
  - Example: p137

6 November 2008

Winsborough CS 3343 Lecture 13

5

## Priority Queues

- Though quicksort usually beats heapsort, heaps are very useful data structures
- Max-priority queue operations:
  - $\text{Insert}(S, x)$ :  $S \leftarrow S \cup \{x\}$
  - $\text{Maximum}(S)$ : returns element with largest key
  - $\text{Extract-Max}(S)$ : removes and returns same
  - $\text{Increase-Key}(S, x, k)$ : increase key of element  $x$  to new, larger key value  $k$
- Useful for scheduling jobs on time-sharing hosts
- Min-priority queues are useful, e.g., in simulating events according to the time at which they occur

6 November 2008

Winsborough CS 3343 Lecture 13

6

## Practical Consideration

- Keys are associated with *handles*, which are references to application objects
- Application objects also contain handles to heap elements
  - These must be maintained correctly when heap elements are moved from one array index to another in the heap

6 November 2008

Winsborough CS 3343 Lecture 13

7

## Procedures to Implement Priority Queue Operations

- Maximum:  $\Theta(1)$   
Heap-Maximum(A)
  1. return A[1]
- Extract-Max :  $O(\lg n)$   
Heap-Extract-Max(A)
  1. If  $\text{heap-size}[A] < 1$
  2. then error "heap underflow"
  3.  $\text{max} \leftarrow A[1]$
  4.  $A[1] \leftarrow A[\text{heap-size}[A]]$
  5.  $\text{heap-size}[A] \leftarrow \text{heap-size}[A] - 1$
  6. Max-Heapify(A, 1)
  7. Return  $\text{max}$

6 November 2008

Winsborough CS 3343 Lecture 13

8

## Operation: Increase-Key

Heap-Increase-Key(A, i, key)

1. If  $\text{key} < A[i]$
  2. then error "new key is smaller than current key"
  3.  $A[i] \leftarrow \text{key}$
  4. While  $i > 1$  and  $A[\text{Parent}(i)] < A[i]$
  5. do exchange  $A[i] \leftrightarrow A[\text{Parent}(i)]$
  6.  $i \leftarrow \text{Parent}(i)$
- Runtime:  $O(\lg n)$
  - Example: p.141
  - Loop invariant: heap property holds except that  $A[i]$  maybe be bigger than  $A[\text{Parent}(i)]$

6 November 2008

Winsborough CS 3343 Lecture 13

9

## Operation: Insert

Max-Heap-Insert(A, key)

1.  $\text{heap-size}[A] \leftarrow \text{heap-size}[A] + 1$
  2.  $A[\text{heap-size}[A]] \leftarrow -\text{infinity}$
  3. Heap-Increase-Key(A,  $\text{heap-size}[A]$ , key)
- Runtime:  $O(\lg n)$ 
    - So any priority queue operation on a heap of size  $n$  can be implemented in  $O(\lg n)$  time

6 November 2008

Winsborough CS 3343 Lecture 13

10