

Binary Heaps

CSE 326
Data Structures
Lecture 11

Readings and References

- Reading
 - › Sections 6.1-6.4

2/14/05

Binary Heaps - Lecture 11

2

Queues

- The Queue ADT: First In First Out (FIFO)
 - › enqueue(data)
 - › dequeue() => data
- Queues are everywhere in CS:
 - › Process scheduling in OS
 - › Handling requests in servers
 - › Printer queues
 - › Event simulation

2/14/05

Binary Heaps - Lecture 11

3

Review: Queue Implementation

- Linked List:
- Circular Array:

2/14/05

Binary Heaps - Lecture 11

4

A New Problem...

- Application: Find the smallest (or highest priority) item quickly
 - › Operating system needs to schedule jobs according to priority
 - › Doctors in ER take patients according to severity of injuries
 - › Event simulation (bank customers arriving and departing, ordered according to when the event happened)

2/14/05

Binary Heaps - Lecture 11

5

Priority Queue ADT

- Priority Queue can efficiently do:
 - › FindMin (and DeleteMin)
 - › Insert
- What if we use...
 - › Lists: If sorted, what is the run time for Insert and FindMin? Unsorted?
 - › Binary Search Trees: What is the run time for Insert and FindMin?

2/14/05

Binary Heaps - Lecture 11

6

Less flexibility → More speed

- Lists
 - › If sorted: FindMin is $O(1)$ but Insert is $O(N)$
 - › If not sorted: Insert is $O(1)$ but FindMin is $O(N)$
- Balanced Binary Search Trees (BSTs)
 - › Insert is $O(\log N)$ and FindMin is $O(\log N)$
- BSTs look good but...
 - › BSTs are efficient for all Finds, not just FindMin
 - › We only need FindMin

2/14/05

Binary Heaps - Lecture 11

7

Better than a speeding BST

- We can do better than Balanced Binary Search Trees?
 - › Very limited requirements: Insert, FindMin, DeleteMin
 - › FindMin is $O(1)$
 - › Insert is $O(\log N)$
 - › DeleteMin is $O(\log N)$
- + uses arrays, given better _____ locality

2/14/05

Binary Heaps - Lecture 11

8

Binary Heaps

- A binary heap is a binary tree that is:
 - › Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
 - › Satisfies the heap order property
 - every node is less than or equal to its children
 - or every node is greater than or equal to its children
- The root node is always the smallest node
 - › or the largest, depending on the heap order

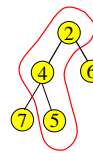
2/14/05

Binary Heaps - Lecture 11

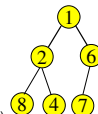
9

Heap order property

- A heap provides limited ordering information
- Each *path* is sorted, but the subtrees are not sorted relative to each other
 - › A binary heap is NOT a binary search tree



These are all valid binary heaps (minimum)



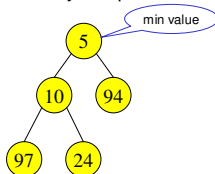
2/14/05

Binary Heaps - Lecture 11

10

Binary Heap vs Binary Search Tree

Binary Heap



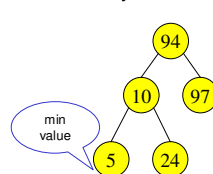
Parent is less than both left and right children

2/14/05

Binary Heaps - Lecture 11

11

Binary Search Tree



Parent is greater than left child, less than right child

Structure property

- A binary heap is a complete tree
 - › All nodes are in use except for possibly the right end of the bottom row

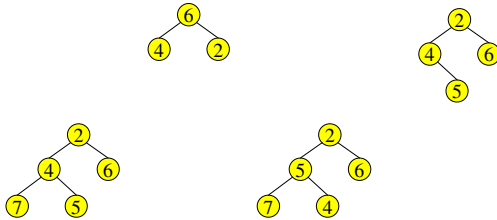


2/14/05

Binary Heaps - Lecture 11

12

Examples



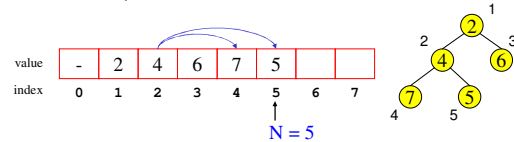
2/14/05

Binary Heaps - Lecture 11

13

Array Implementation of Heaps (Implicit Pointers)

- Root node = $A[1]$
- Children of $A[i] = A[2i]$, $A[2i + 1]$
- Keep track of current size N (number of nodes)



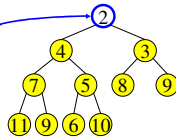
2/14/05

Binary Heaps - Lecture 11

14

FindMin and DeleteMin

- FindMin: Easy!
 - › Return root value $A[1]$
 - › Run time = ?
- DeleteMin:
 - › Delete (and return) value at root node



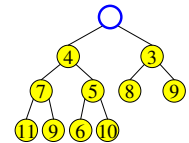
2/14/05

Binary Heaps - Lecture 11

15

DeleteMin

- Delete (and return) value at root node



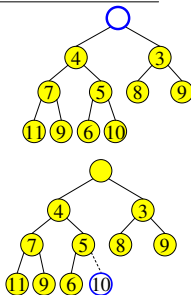
2/14/05

Binary Heaps - Lecture 11

16

Maintain the Structure Property

- We now have a "Hole" at the root
 - › Need to fill the hole with another value
- When we get done, the tree will have one less node and must still be complete



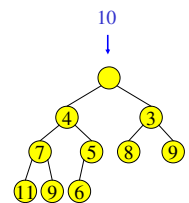
2/14/05

Binary Heaps - Lecture 11

17

Maintain the Heap Property

- The last value has lost its node
 - › we need to find a new place for it
- We can do a simple insertion sort operation to find the correct place for it in the tree

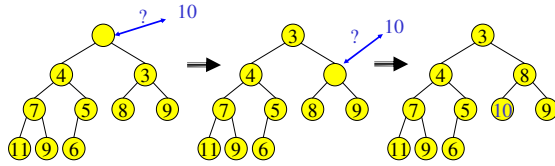


2/14/05

Binary Heaps - Lecture 11

18

DeleteMin: Percolate Down



- Keep comparing with children $A[2i]$ and $A[2i + 1]$
- Copy smaller child up and go down one level
- Done if both children are \geq item or reached a leaf node
- What is the run time?

2/14/05

Binary Heaps - Lecture 11

19

Percolate Down

```
PercDown(i:integer, x :integer): {
// N is the number of entries in queue//
j : integer;
Case{
  2i > N : A[i] := x; //at bottom//
  2i = N : if A[2i] < x then
    A[i] := A[2i]; A[2i] := x;
  else A[i] := x;
  2i < N : if A[2i] < A[2i+1] then j := 2i;
    else j := 2i+1;
    if A[j] < x then
      A[i] := A[j]; PercDown(j,x);
    else A[i] := x;
}}
```

2/14/05

Binary Heaps - Lecture 11

20

DeleteMin: Run Time Analysis

- Run time is $O(\text{depth of heap})$
- A heap is a complete binary tree
- Depth of a complete binary tree of N nodes?
 - › $\text{height} = \lceil \log_2(N) \rceil - 1$
- Run time of DeleteMin is $O(\log N)$

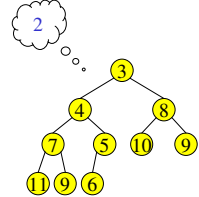
2/14/05

Binary Heaps - Lecture 11

21

Insert

- Add a value to the tree
- Structure and heap order properties must still be correct when we are done



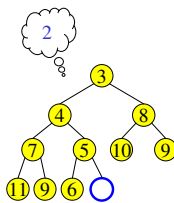
2/14/05

Binary Heaps - Lecture 11

22

Maintain the Structure Property

- The only valid place for a new node in a complete tree is at the end of the array
- We need to decide on the correct value for the new node, and adjust the heap accordingly



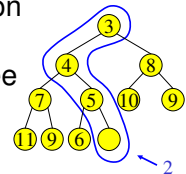
2/14/05

Binary Heaps - Lecture 11

23

Maintain the Heap Property

- The new value goes where?
- We can do a simple insertion sort operation to find the correct place for it in the tree

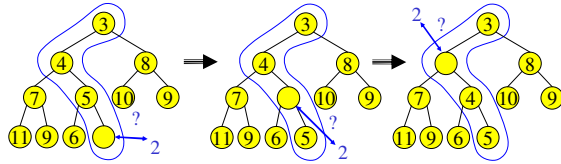


2/14/05

Binary Heaps - Lecture 11

24

Insert: Percolate Up



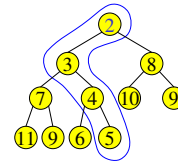
- Start at last node and keep comparing with parent $A[i/2]$
- If parent larger, copy parent down and go up one level
- Done if parent \leq item or reached top node $A[1]$
- Run time?

2/14/05

Binary Heaps - Lecture 11

25

Insert: Done



- Run time?

2/14/05

Binary Heaps - Lecture 11

26

PercUp

- Class participation
- Define PercUp which percolates new entry to correct spot.
- Note: the parent of i is $i/2$

```
PercUp(i : integer, x : integer): {
  ???
}
```

2/14/05

Binary Heaps - Lecture 11

27

Quiz

- Say we want to add decreaseKey(i, x). We do " $A[i] = A[i] - x$;" followed by?
- What about increaseKey(i, x)?

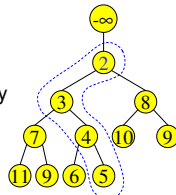
2/14/05

Binary Heaps - Lecture 11

28

Sentinel Values

- Every iteration of Insert needs to test:
 - › if it has reached the top node $A[1]$
 - › if parent \leq item
- Can avoid first test if $A[0]$ contains a very large negative value
 - › sentinel $-\infty < \text{item}$, for all items
- Second test alone always stops at top



value	$-\infty$	2	3	8	7	4	10	9	11	9	6	5		
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13

2/14/05

Binary Heaps - Lecture 11

29

Binary Heap Analysis

- Space needed for heap of N nodes: $O(\text{MaxN})$
 - › An array of size MaxN , plus a variable to store the size N , plus an array slot to hold the sentinel
- Time
 - › FindMin: $O(1)$
 - › DeleteMin and Insert: $O(\log N)$
 - › BuildHeap from N inputs : $O(N)$

2/14/05

Binary Heaps - Lecture 11

30

Average Case Analysis

- Question: At what height do we expect to find median element?
- Formally, this gives expected _____ moves for insert.
- Question: How far does last element need to be percolated down?
- This gives expected _____ moves for delete

2/14/05

Binary Heaps - Lecture 11

31

Build Heap

- Doing N inserts takes expected time $O(N)$, but worst-case time $O(N \log N)$.
- We can do it in guaranteed $O(N)$ time if we're clever.
- Idea is to treat input array as a broken heap and fix it with percolateDown

2/14/05

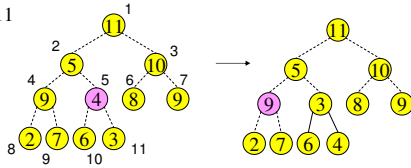
Binary Heaps - Lecture 11

32

Build Heap

```
BuildHeap {
  for i = N/2 to 1 by -1 PercDown(i, A[i])
}
```

N=11

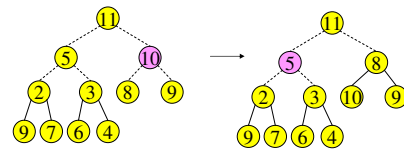


2/14/05

Binary Heaps - Lecture 11

33

Build Heap

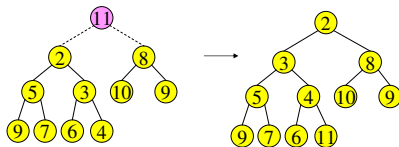


2/14/05

Binary Heaps - Lecture 11

34

Build Heap



2/14/05

Binary Heaps - Lecture 11

35

Analysis of Build Heap

- Assume $N = 2^k - 1$
 - › Level 1: $k - 1$ steps for 1 item
 - › Level 2: $k - 2$ steps for 2 items
 - › Level 3: $k - 3$ steps for 4 items
 - › Level i : $k - i$ steps for 2^{i-1} items

$$\begin{aligned} \text{Total Steps} &= \sum_{i=1}^{k-1} (k-i) 2^{i-1} = 2^k - k \\ &= O(N) \end{aligned}$$

2/14/05

Binary Heaps - Lecture 11

36

Other Heap Operations

- Find(X, H): Find the element X in heap H of N elements
 - › Heap is somewhat ordered, so maybe we can be smarter than a linear search?
 - › What is the running time?
- FindMax(H): Find the maximum element in H
 - › What is the running time?
 - › We sacrificed performance of these operations in order to get $O(1)$ performance for FindMin

2/14/05

Binary Heaps - Lecture 11

37

Other Heap Operations

- Delete(P,H): E.g. Delete a job waiting in queue that has been preemptively terminated by user
 - › Use DecreaseKey(P, ∞, H) followed by DeleteMin
 - › Running Time: $O(\log N)$

2/14/05

Binary Heaps - Lecture 11

38

Other Heap Operations

- DecreaseKey(P, Δ, H): Decrease the key value of node at position P by a positive amount Δ . eg, to increase priority
 - › First, subtract Δ from current value at P
 - › Heap order property may be violated
 - › so percolate up to fix
 - › Running Time: $O(\log N)$

2/14/05

Binary Heaps - Lecture 11

39

Other Heap Operations

- IncreaseKey(P, Δ, H): Increase the key value of node at position P by a positive amount Δ . eg, to decrease priority
 - › First, add Δ to current value at P
 - › Heap order property may be violated
 - › so percolate down to fix
 - › Running Time: $O(\log N)$

2/14/05

Binary Heaps - Lecture 11

40

Other Heap Operations

- Merge($H1, H2$): Merge two heaps H1 and H2 of size $O(N)$. H1 and H2 are stored in two arrays.
 - › Can do $O(N)$ Insert operations: $O(N \log N)$ time
 - › Better: Copy H2 at the end of H1 and use BuildHeap. Running Time: $O(N)$

2/14/05

Binary Heaps - Lecture 11

41

One last question

- Pointer based trees could grow indefinitely. Array based tree has to be copied ($O(N)$ runtime) to a larger array when space runs out.
- Does this affect our asymptotic runtime?

2/14/05

Binary Heaps - Lecture 11

42

PercUp Solution

```
PercUp(i : integer, x : integer): {  
  if i = 1 then A[1] := x  
  else if A[i/2] < x then  
    A[i] := x;  
  else  
    A[i] := A[i/2];  
    Percup(i/2, x);  
}
```