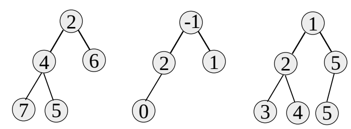
CSE 373 Lecture 10: Heaps and Priority Queues

- Today's Topics:
  - Binary Heaps
    - Array Implementation
    - FindMin/DeleteMin and Percolate Down
    - Insert and Percolate Up
    - BuildHeap, DecreaseKey, IncreaseKey
  - Introduction to Binomial Queues
- Covered in Chapter 6 in the text

Heaps

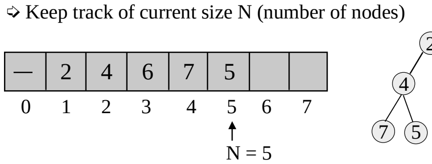
- A binary heap is a binary tree that is:
  1. Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  2. Satisfies the heap order property: every node is smaller than (or equal to) its children
- Therefore, the root node is always the smallest in a heap



Which of these is not a heap?

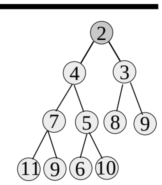
Array Implementation of Heaps

- Since heaps are complete binary trees, we can avoid pointers and use an array
- Array Implementation:
  - Root node = A[1]
  - Children of A[i] = A[2i], A[2i + 1]
  - Keep track of current size N (number of nodes)



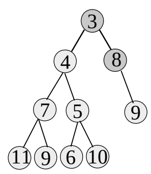
Heaps: FindMin and DeleteMin Operations

- FindMin: Easy! Return root value A[1]
  - Run time = ?
- DeleteMin:
  - Delete (and return) value at root node
  - We now have a "Hole" at the root
  - Need to fill the hole with another value
  - Replace with smallest child?
    - Try replacing 2 with smallest child and that node with its smallest child, and so on…what happens?



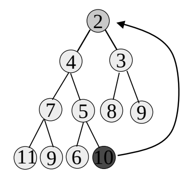
Heaps: DeleteMin Operation

- DeleteMin:
  - Delete (and return) value at root node
  - We now have a “Hole” at the root
  - Need to fill the hole with another value
  - Replace with smallest child?
    - Try replacing 2 with smallest child and so on…what happens?
    - Tree is no longer complete!
    - Let’s try another strategy…



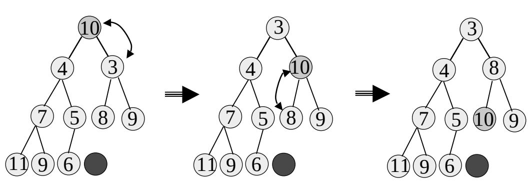
Heaps: DeleteMin Operation

- DeleteMin:
  - Delete (and return) value at root node
  - We now have a “Hole” at the root
  - Need to fill the hole with another value
- Since heap is one node smaller, we need to empty the last slot
- Steps:
  - Move last item to top; decrease size by 1
  - Percolate down the top item to its correct position in the heap



—	2	4	3	7	5	8	9	11	9	6	10	

DeleteMin: Percolate Down



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

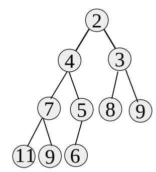
DeleteMin: Run Time Analysis

- Run time is O(depth of tree)
- What is the depth of a complete binary tree of N nodes?

DeleteMin: Run Time Analysis

- Run time is O(depth of heap)
- A heap is a complete binary tree
- What is the depth of a complete binary tree of N nodes?
  - At depth d, you can have: $N = 2^d$ (one leaf at depth d) to $2^{d+1}-1$ nodes (all leaves at depth d)
  - So, depth d for a heap is: $\log N \le d \le \log(N+1)-1$ or $\Theta(\log N)$
- Therefore, run time of DeleteMin is O(log N)

Heaps: Insert Operation

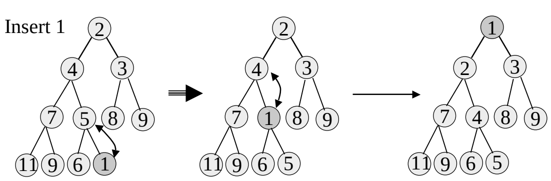


Insert 1

N = 10

—	2	4	3	7	5	8	9	11	9	6		

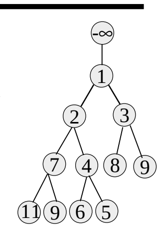
Insert: Percolate Up



- Insert at last node and keep comparing with parent A[i/2]
- If parent larger, replace with parent and go up one level
- Done if parent ≤ item or reached top node A[1]
- Run time?

Sentinel Values

- Every iteration of Insert needs to test:
  1. if it has reached the top node A[1]
  2. if parent ≤ item
- Can avoid first test if A[0] contains a very large negative value (denoted by $-\infty$)
- Then, test #2 always stops at top
  - $-\infty <$ item for all items
- Such a data value that serves as a marker is called a sentinel
  - Used to improve efficiency and simplify code



A	$-\infty$	1	2	3	7	4	8	9	11	9	6	5	

Summary of Heap ADT Analysis

- Consider a heap of N nodes
- Space needed: O(N)
  - Actually, O(MaxSize) where MaxSize is the size of the array
  - One more variable to store the size N
  - With sentinel, array-based implementation uses N+2 space
  - Pointer-based implementation: pointers for children and parent
    - Total space = 3N + 1 (3 pointers per node + 1 for size)
- FindMin: O(1) time; DeleteMin and Insert: O(log N) time
- BuildHeap from N inputs: What is the run time?
  - N Insert operations = O(N log N). Actually, can do better…
  - O(N): Treat input array as a heap and fix it using percolate down
    - See text for proof that this takes O(N) time.

Other Heap Operations

- Find(X, H): Find the element X in heap H of N elements
  - What is the running time?
- FindMax(H): Find the maximum element in H
  - What is the running time?

Other Heap Operations

- Find and FindMax: O(N)
- DecreaseKey(P,Δ,H): Decrease the key value of node at position P by a positive amount Δ. E.g. System administrators can increase priority of important jobs.
  - First, subtract Δ from current value at P
  - Heap order property may be violated
  - Percolate up or down?
  - Running time?

Other Heap Operations

- DecreaseKey(P,Δ,H): Subtract Δ from current key value at P and percolate up. Running Time: O(log N)
- IncreaseKey(P,Δ,H): Add Δ to current key value at P and percolate down. Running Time: O(log N)
  - E.g. Schedulers in OS often decrease priority of CPU-hogging jobs
- Delete(P,H): E.g. Delete a job waiting in queue that has been preemptively terminated by user
  - Use DecreaseKey followed by DeleteMin.
  - How? (Idea: Decrease key to bubble node at P to the top)
  - Running Time?

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(H).
  - Running Time: O(log N)
- Merge(H1,H2): Merge two heaps H1 and H2 of size O(N). H1 and H2 are stored in two arrays. E.g. Combine queues from two different sources to run on one CPU.
  1. Can do O(N) Insert operations: O(N log N) time
  2. Better: Copy H2 at the end of H1 and use BuildHeap Running Time: O(N)

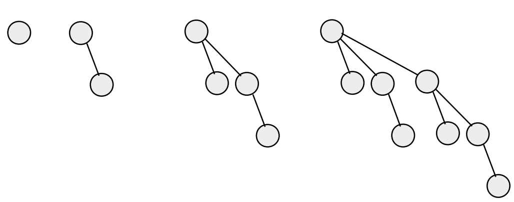
  Can we do even better? (i.e. Merge in O(log N) time?)

Binomial Queues

- Binomial queues support all three priority queue operations Merge, Insert and DeleteMin in O(log N) time
- Idea: Maintain a collection of heap-ordered trees
  - *Forest of binomial trees*
- Recursive Definition of Binomial Tree (based on height k):
  - Only one binomial tree for a given height
  - Binomial tree of height 0 = single root node
  - Binomial tree of height k = B_k = Attach B_{k-1} to root of another B_{k-1}
- Binomial tree of height k has exactly 2^k nodes (by induction)

The First Four Binomial Trees

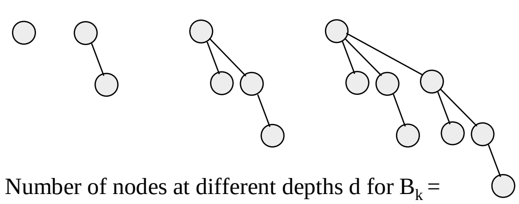
B_0 B_1 B_2 B_3



Why are these trees called binomial?
(hint: how many nodes at depth d?)

Why "Binomial"?

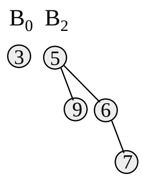
B_0 B_1 B_2 B_3 …………….. B_k



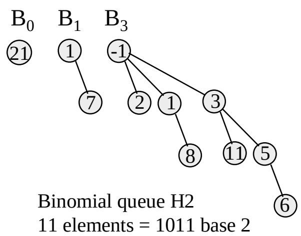
Number of nodes at different depths d for B_k = [1], [1 1], [1 2 1], [1 3 3 1], …, =
Binomial coefficients of $(a + b)^k = k!/((k-d)!d!)$

Binomial Queues

Binomial Queue = “forest” of heap-ordered binomial trees



Binomial queue H1
5 elements = 101 base 2
→ B_2 B_0



Binomial queue H2
11 elements = 1011 base 2
→ B_3 B_1 B_0

Next Class:

How do we merge H1 and H2?

More on Binomial Heaps and Priority Queues

To Do:

Read Chapter 6

Programming Assignment #1 will be on-line tomorrow (due April 27)