Priority Queues - Normally done as Heaps

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Making priority queues out of heaps
    - Queues from mole hills?

Announcements

- Midterm grades are due today, they’re in for this class
  - Grading system: only went down if you failed to turn things in
  - Did use exam and MA percentages
  - Basically, if you’re missing whole assignments you dropped down
- Next MA has to be out ASAP!
  - Grading, projects, etc. oh my!

Scientists Want to Build a Super-Fast, Self-Replicating Computer That "Grows as It Computes"

- Self-replicating computer that replaces silicon chips with processors made from DNA molecules
- Faster than any other form of computer ever proposed - even quantum computers
- Nondeterministic universal Turing machine (NUTM), it's predicted that the technology could execute all possible algorithms at once

http://www.sciencealert.com/scientists-want-to-build-a-super-fast-self-replicating-computer-that-grows-as-it-computes

Today’s footers brought to you by: EvilOverlord.com

The Top 100 Things I'd Do

If I Ever Became An Evil Overlord

http://eviloverlord.com/lists/overlord.html

Priority Queues (in general)

- Goal is to make finding the next prioritized element in the queue faster
- Simple implementations are easy to do, but are slow at scale
- VERY useful for many applications (I actually use them often)
  - Shortest Job First scheduling
  - Merging lists together
  - Print queues by print job size or priority of user
  - Greedy algorithms (graph traversal, AI, depth first search)
- A very elegant solution is possible - with no pointers!
  - If you’ve read 6.3 you should be going “huzzah!”

Priority Queue API in chapter 6

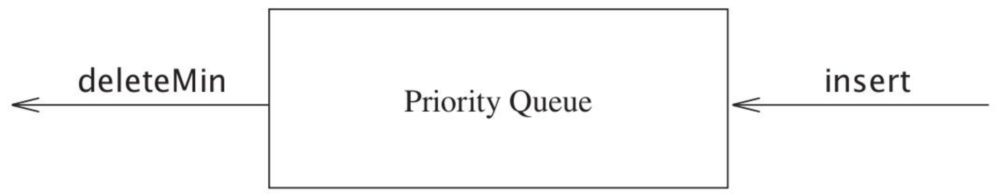
- void Insert(x)   // Add x to queue
- <type of x> DeleteMin()   // Remove & return smallest x in queue

  -- Analogous to enqueue and dequeue and could be called such

- C++11 -> std::priority_queue<T>
  top (accesses top element) | empty()
  size() | push(x) | pop(x)
  http://en.cppreference.com/w/cpp/container/priority_queue

Naïve Implementations of Priority Queues

- Linked list:
  - Insert head -> O(1) DelMin -> search for min in O(N)
- Sorted linked list:
  - Insertion sort style -> O(N/2) DelMin -> always at head for O(1)
- BST style:
  - Both insert and delete are O(log N) // But, left subtree will vanish over time
- AVL tree tyle:
  - See BST, but no imbalance over time // Still, more complex than needed



A better solution: The Binary Heap backed priority queue

- Binary Heap ~= Heap // As opposed to “The” Heap in memory
- Both insert and delete in O(log N)
- Building heap is O(N) time
  - Allows for O(N) time merging of unordered sets, which can be quite useful
  - I should have remembered this in one of my Google interviews!
- Can be done *without* pointers!

The Heap property

- Binary tree
- Parent key <= Children’s keys
- Height is log N
- Nodes stored in an array, not on The Heap with pointers
    - Still drawn as trees to make operations clear

Despite its proven stress-relieving effect, I will not indulge in maniacal laughter. When so occupied, it's too easy to miss unexpected developments that a more attentive individual could adjust to accordingly.

How to do the magic array?

- Root stored at array[1]
- Children of a node @ index i are: 2i and 2i+1
- Parent of a child @ index i is: i/2
    - (remember it's integer, so this rounds down)
- This will actually store any binary tree if you can allocate the array

If my doomsday device happens to come with a reverse switch, as soon as it has been employed it will be melted down and made into limited-edition commemorative coins.

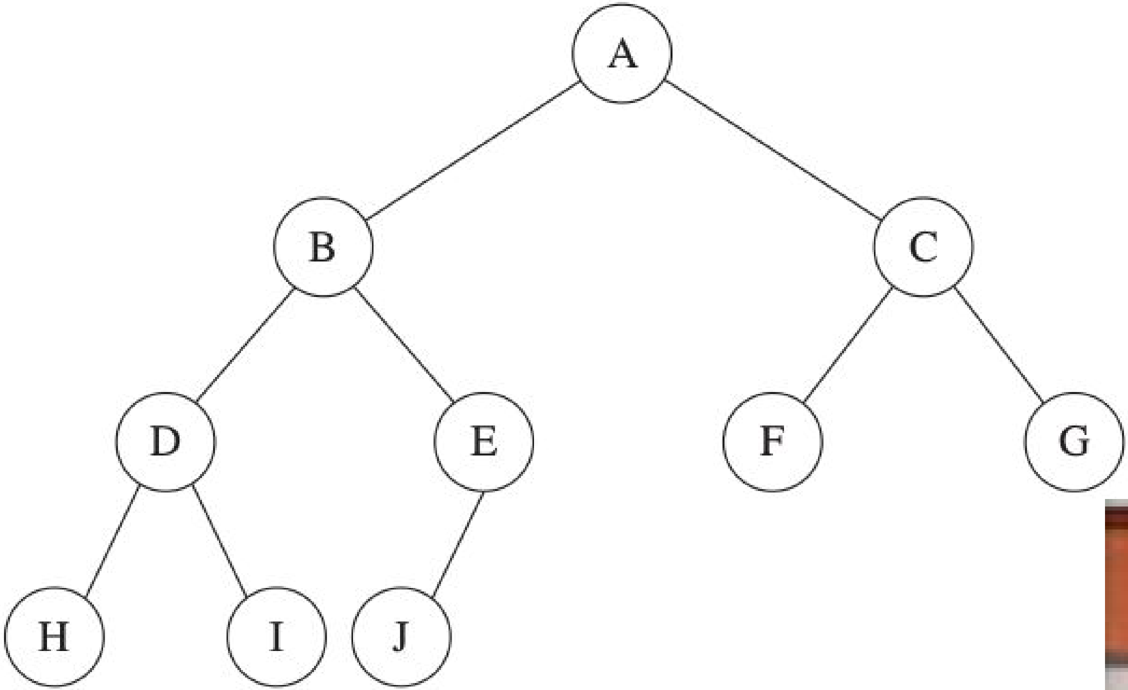


Figure 6.2 A complete binary tree

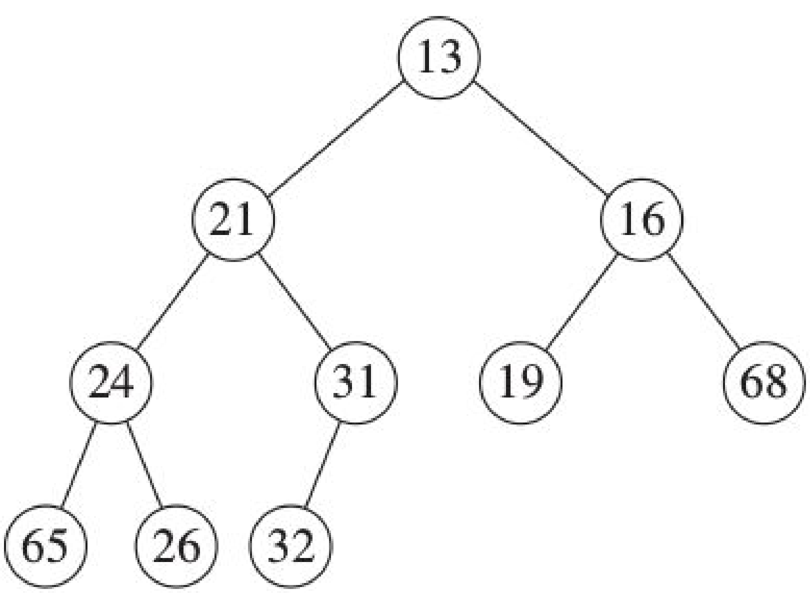
	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember! This is NOT a BST!

- Both children are larger than their parents in a Heap!
- Children are NOT sorted in any way
- Structure is NOT a BST

This is a valid Heap! ⇒

It's ***not*** a valid BST.



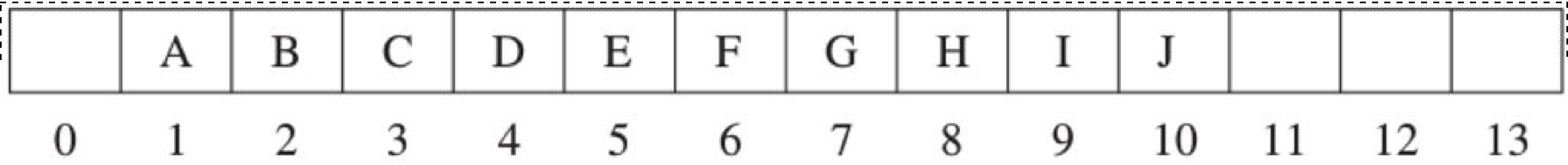
Heap-order property

- Parents <= Children
- Root will always be smallest value
- findMin will always be in constant time

I will be neither chivalrous nor sporting. If I have an unstoppable superweapon, I will use it as early and as often as possible instead of keeping it in reserve.

Insert operation
Or: “how I learned to percolate up”

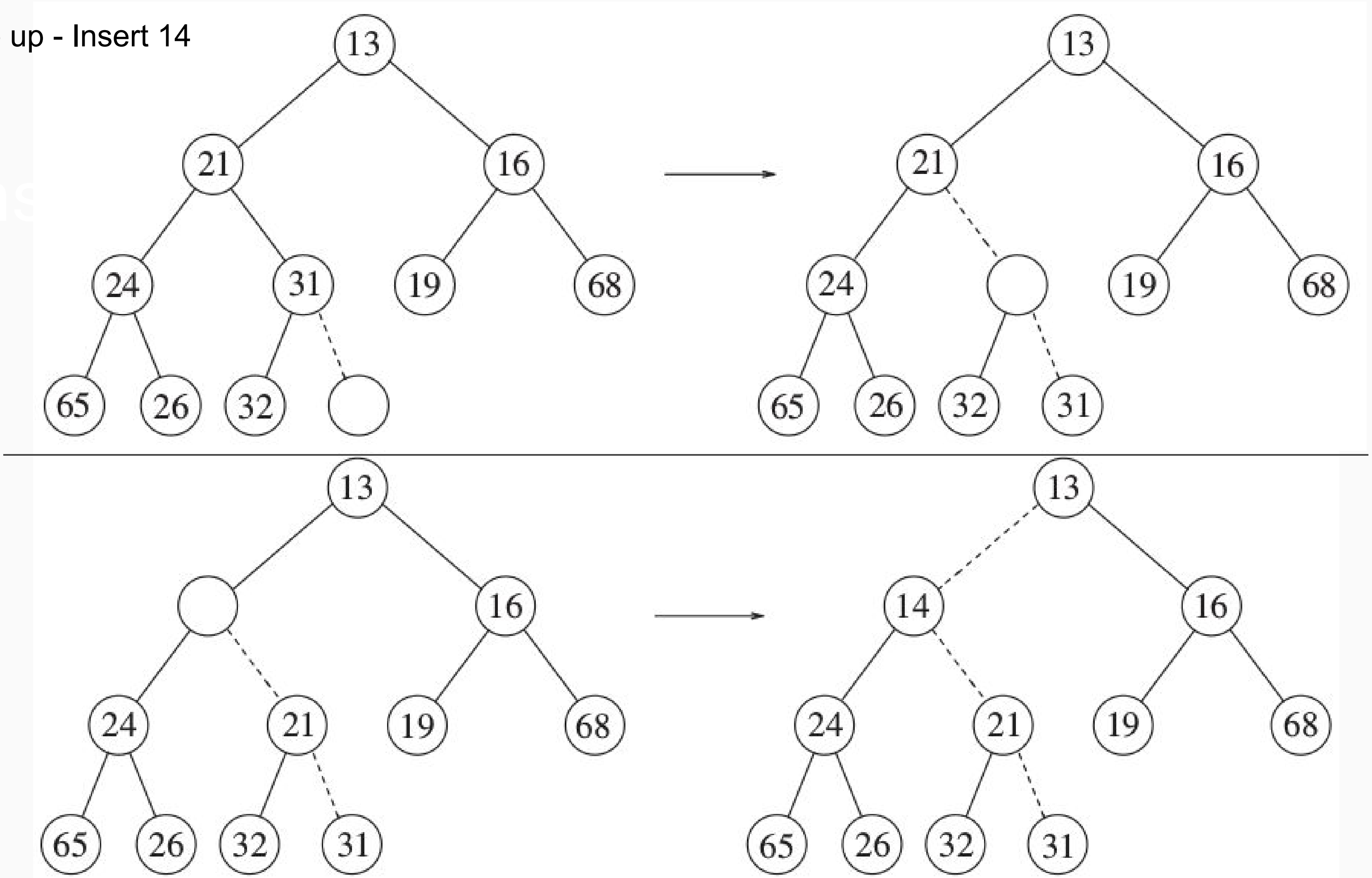
- Take insert(x) and put x at the next opening in the heap
- While x < parent(x) and x != root, swap
    - But, this is easily beaten by other code that’s clever

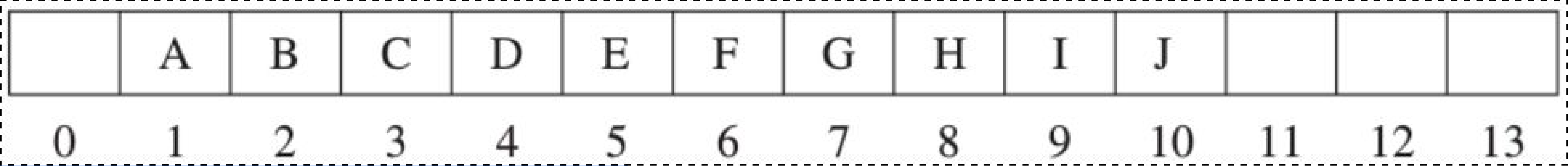


	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

If it becomes necessary to escape, I will never stop to pose dramatically and toss off a one-liner.

Percolate up - Insert 14

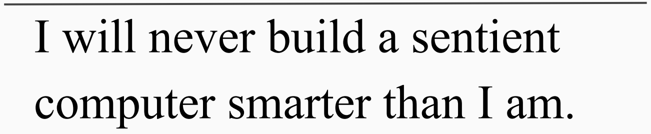




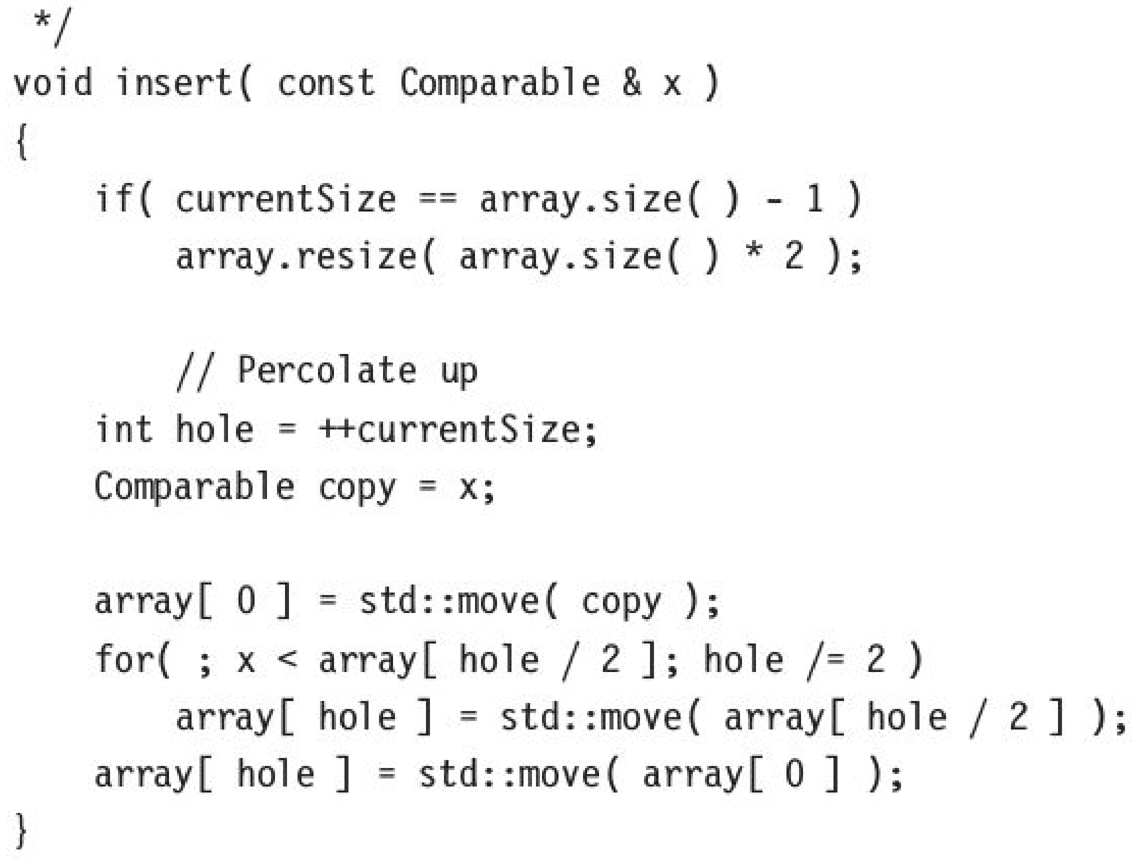
	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Wow, short code!

- Also, a bit magic here
  - Known as “optimized”
  - Harder to read!
  - Does d+1 assignments Instead of 3d assignments
- In practice, average is:
  - 2.607 compares
  - 1.607 levels



I will never build a sentient computer smarter than I am.



```
 */
void insert( const Comparable & x )
{
    if( currentSize == array.size( ) - 1 )
        array.resize( array.size( ) * 2 );

        // Percolate up
    int hole = ++currentSize;
    Comparable copy = x;

    array[ 0 ] = std::move( copy );
    for( ; x < array[ hole / 2 ]; hole /= 2 )
        array[ hole ] = std::move( array[ hole / 2 ] );
    array[ hole ] = std::move( array[ 0 ] );
}
```

PA3 - hashing dictionary

- This is a significant project
- You'll have just over two weeks, and the length of the project might surprise some of you once you dig into it
- You're making a dictionary where the keys are words which index your hash table.
- Word objects (word, definition) are held in separate chains (lists)
- There's notable string parsing and user interface dev here
    - Make sure to look at how I do the testing on this one. It'll inform you on how to parse!

I’ll see you Friday!

- More on classic heaps
- Friday will be the other kinds of heaps:
  - D-heaps
  - Leftist heaps (for the comrades among us!)

I will not design my Main Control Room so that every workstation is facing away from the door.