Priority Queues II

CSE 373
Data Structures & Algorithms
Ruth Anderson

Today's Outline

- **Announcements**
  - **Homework #3 due Wed, Feb 8th, 11pm.**
- **Today's Topics:**
  - **Priority Queues**
    - **Binary Min Heap - buildheap**
    - **D-Heaps**

Facts about Binary Min Heaps

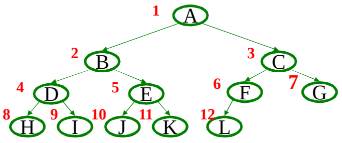
Observations:

- finding a child/parent index is a multiply/divide by two
- operations jump widely through the heap
- each percolate step looks at only two new nodes
- inserts are *at least* as common as deleteMins

Realities:

- division/multiplication by *powers* of two are equally fast
- looking at only **two** new pieces of data: bad for cache!
- with huge data sets, disk accesses dominate

Representing Complete Binary Trees in an Array

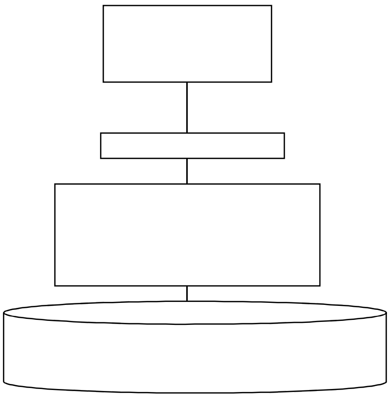


From node **i**:

left child:
right child:
parent:

implicit (array) implementation:

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



Cycles to access:

CPU

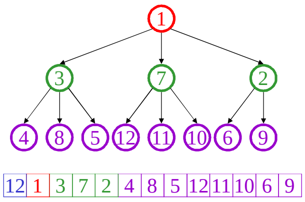
Cache

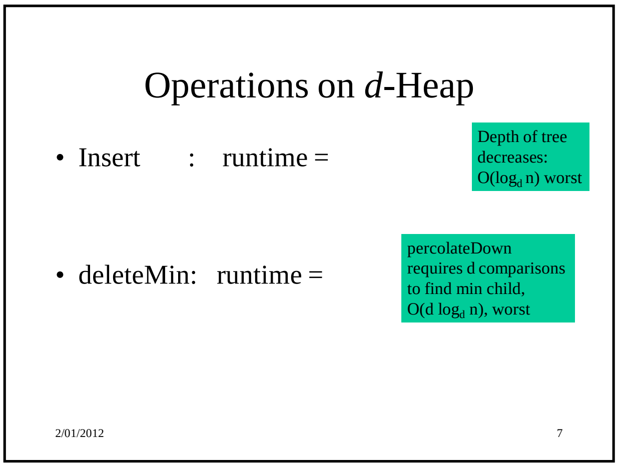
Memory

Disk

A Solution: *d*-Heaps

- Each node has *d* children
- Still representable by array
- Good choices for *d*:
  - (choose a power of two for efficiency)
  - fit one set of children in a cache line
  - fit one set of children on a memory page/disk block





Operations on *d*-Heap

- Insert : runtime =

Depth of tree decreases: $O(\log_d n)$ worst

- deleteMin: runtime =

percolateDown requires d comparisons to find min child, $O(d \log_d n)$, worst