CSE 326: Data Structures

Binary Heaps

James Fogarty

Autumn 2007

Lecture 5

Administrative

- This next section is here to show you why I am gone next week and to give a break to the person who comes late to class

- Be sure to tell them that they owe me perfect scores on my teaching evaluation

UIST 2007

- User Interface Software and Technology
- UW has 5 of 24 full papers
- If HCI is your thing, UW rocks it
- http://dub.washington.edu

- Assieme
  http://assieme.cs.washington.edu:8080

Administrative

- HW1 is due Today

Building a Heap

12	5	11	3	10	6	9	4	8	1	7	2

Building a Heap

- Adding the items one at a time is O(n log n) in the worst case
- I promised O(n) for today

Working on Heaps

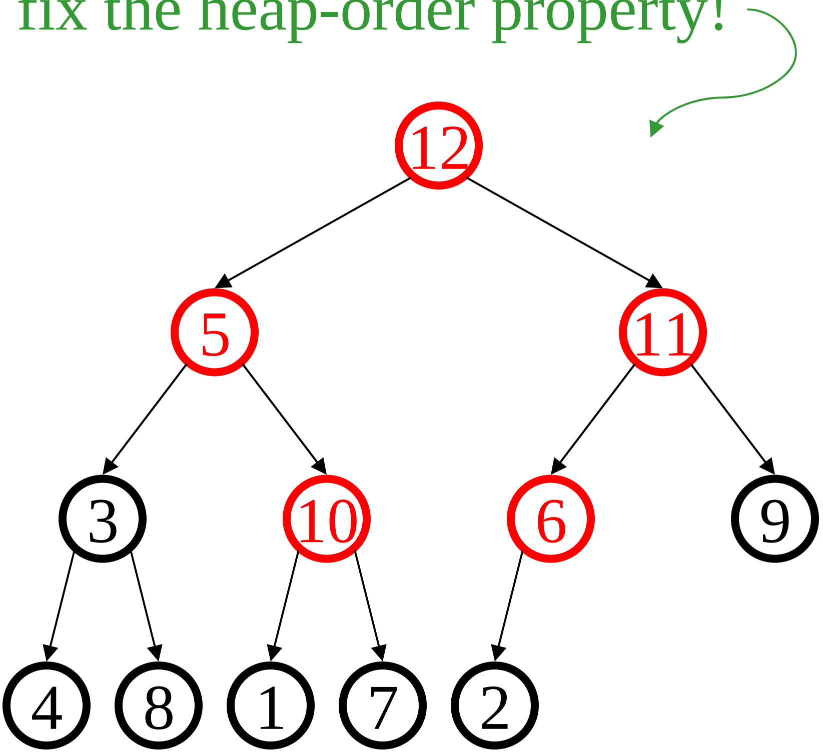
- What are the two properties of a heap?
  - Structure Property
  - Order Property

- How do we work on heaps?
  - Fix the structure
  - Fix the order

BuildHeap: Floyd's Method

12	5	11	3	10	6	9	4	8	1	7	2

Add elements arbitrarily to form a complete tree.
Pretend it's a heap and fix the heap-order property!

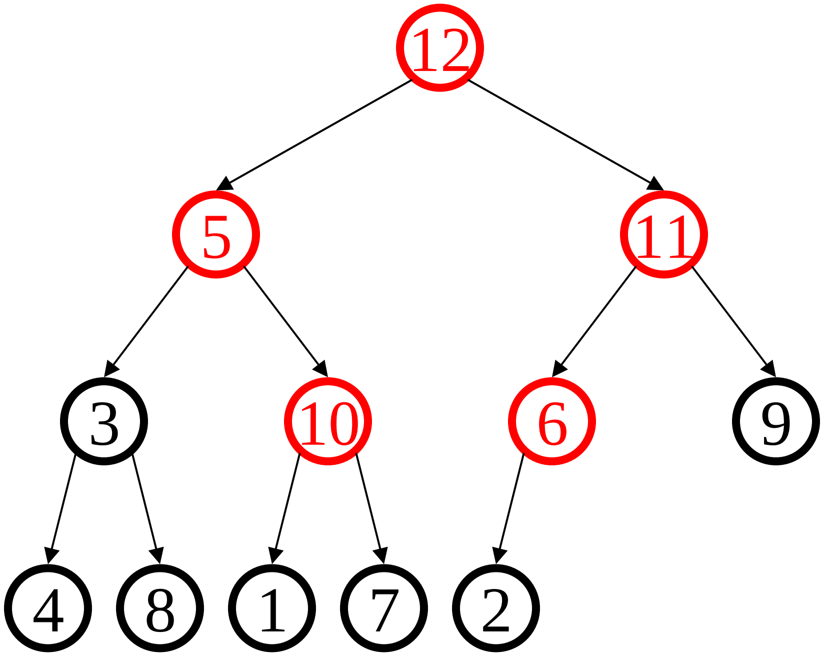


Buildheap pseudocode

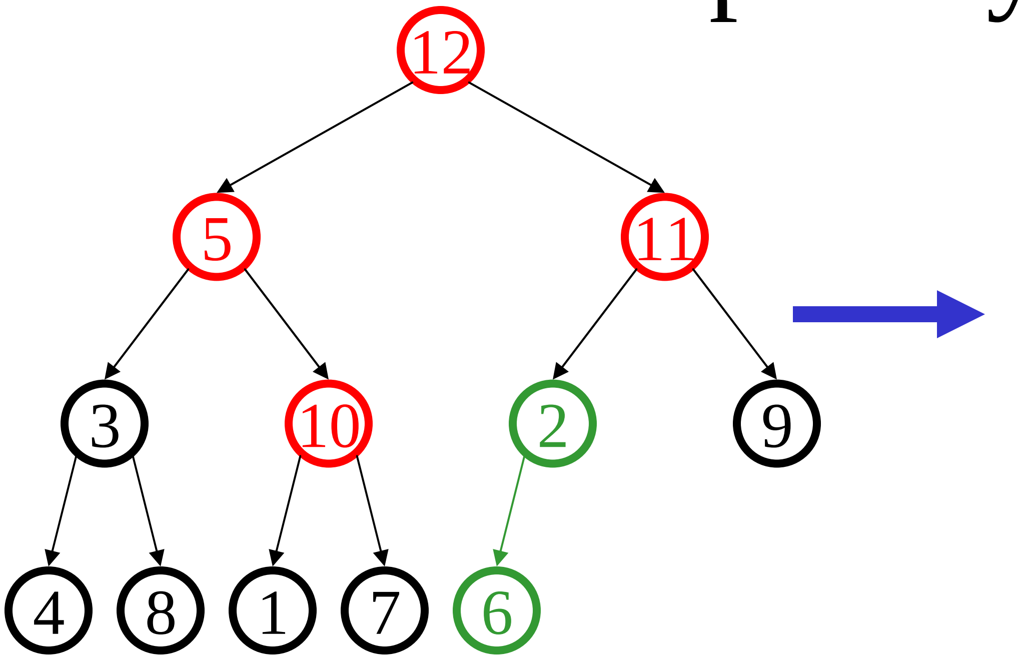
```
private void buildHeap() {
  for ( int i = currentSize/2; i > 0; i-- )
     percolateDown( i );
}
```

runtime:

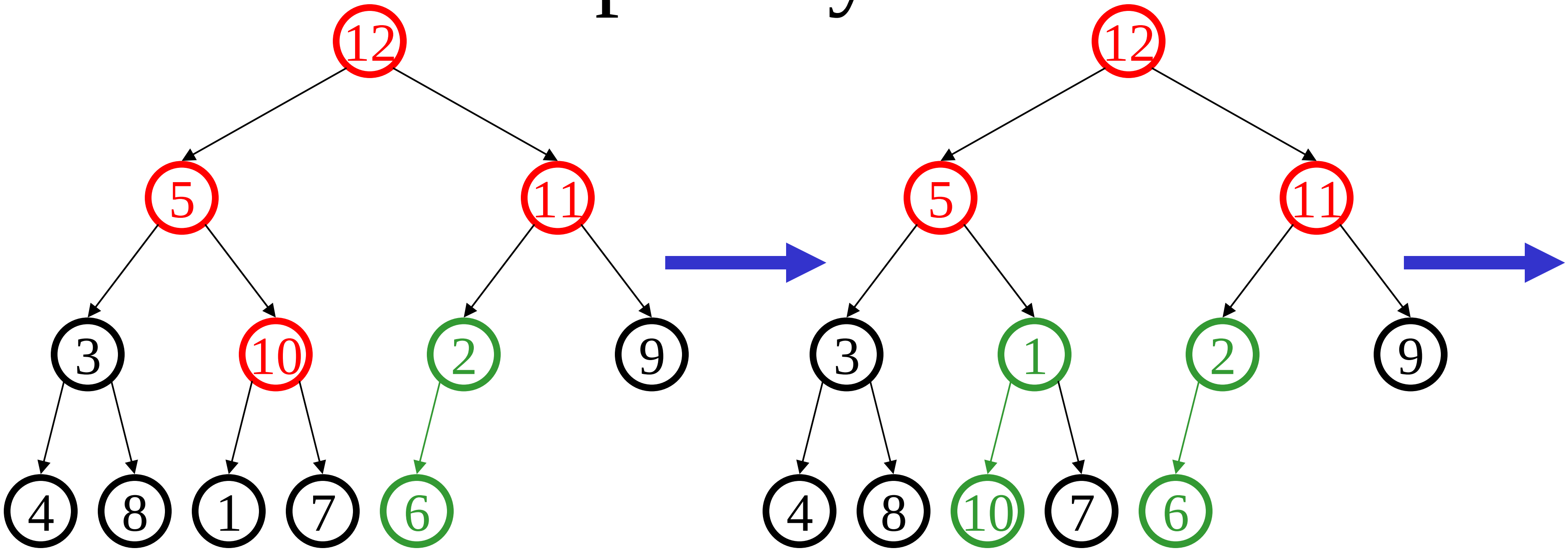
BuildHeap: Floyd's Method



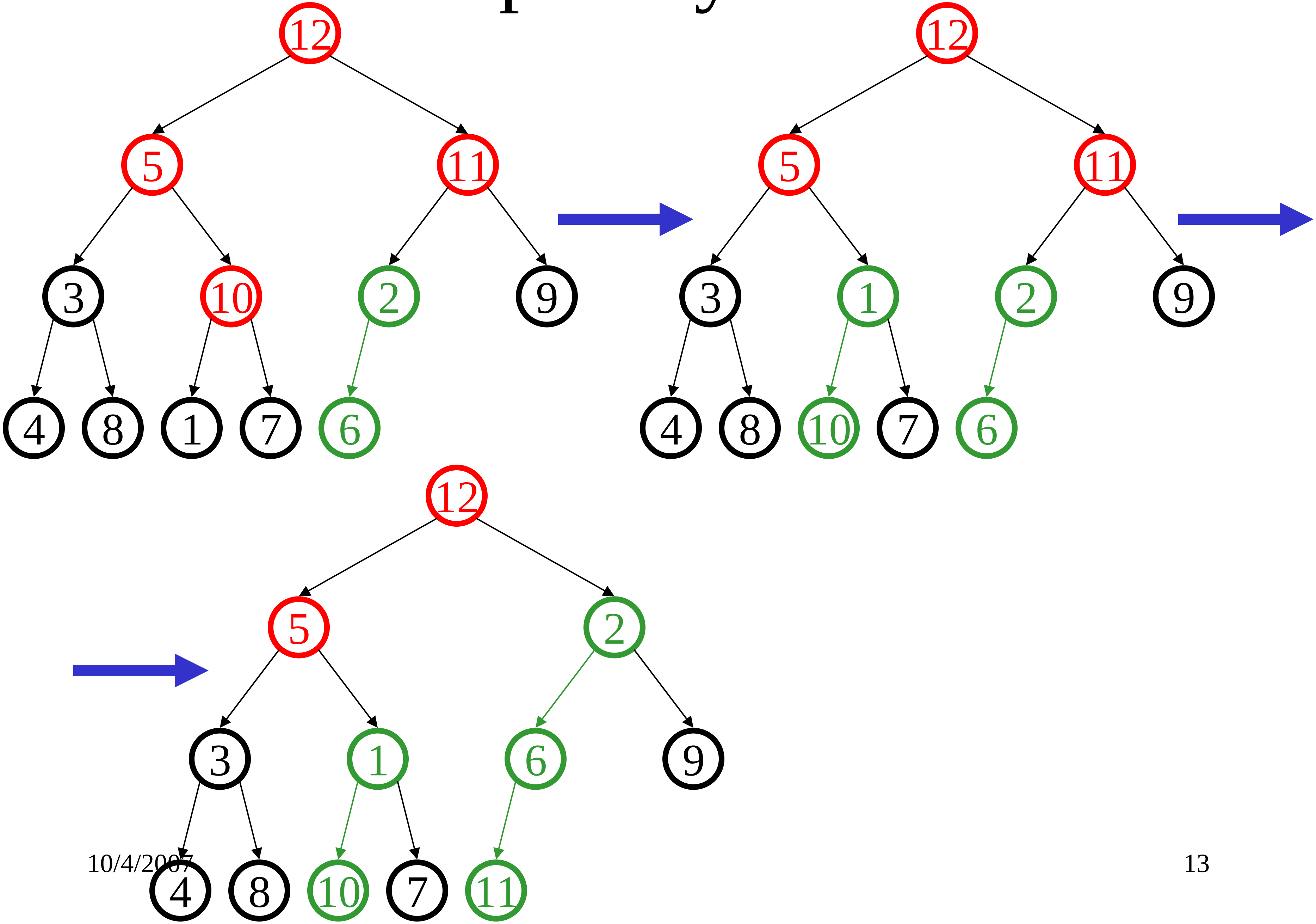
BuildHeap: Floyd’s Method



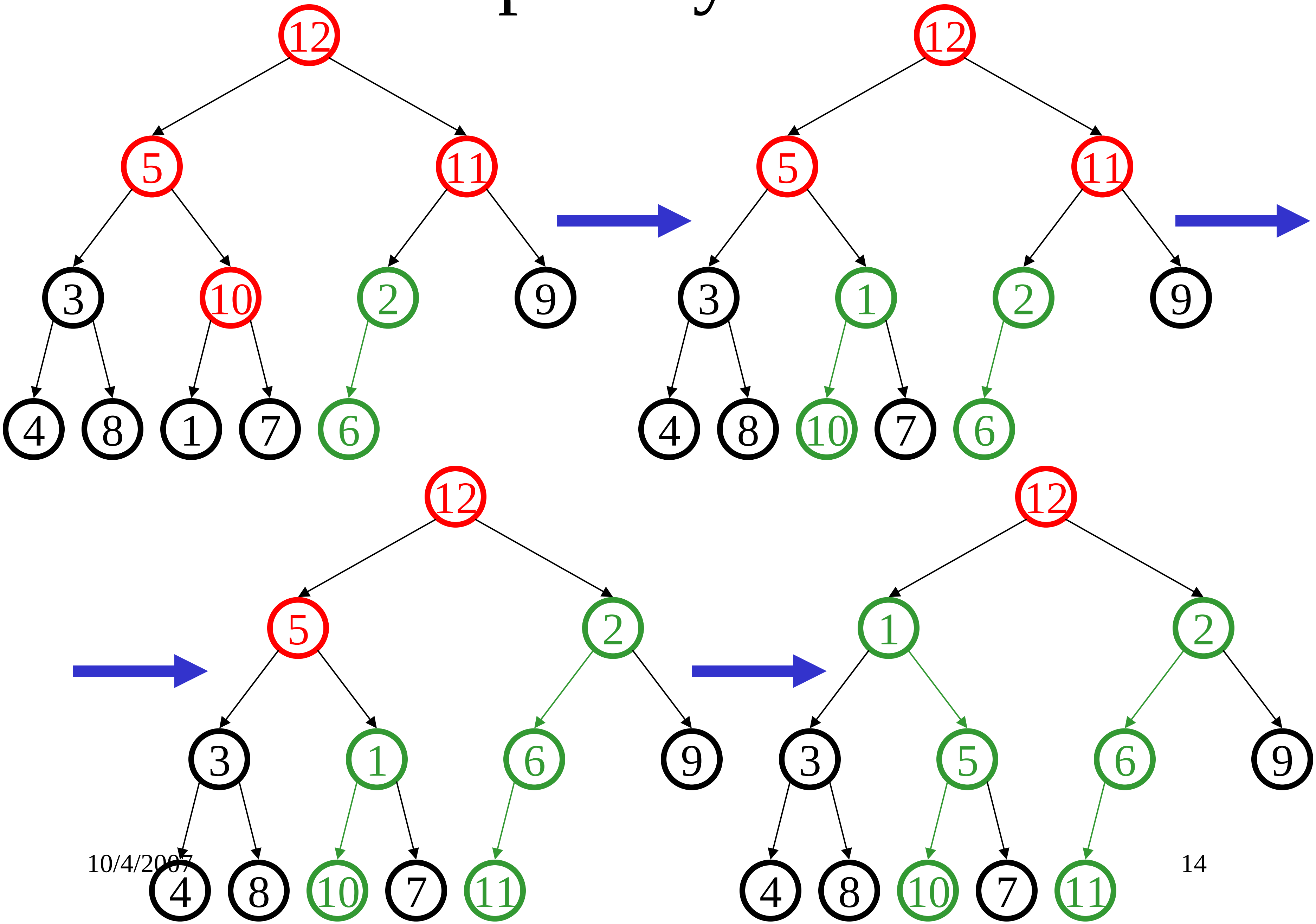
BuildHeap: Floyd’s Method



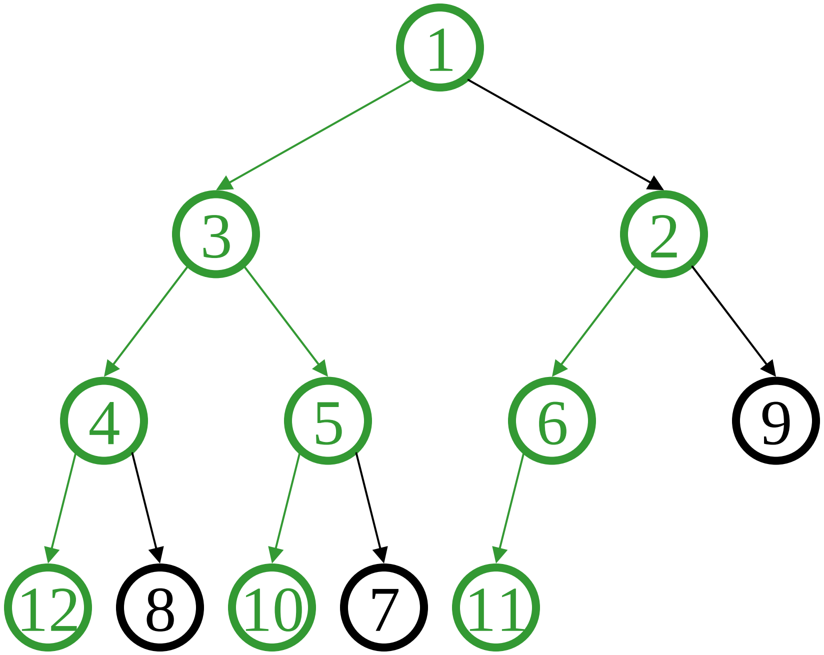
BuildHeap: Floyd’s Method



BuildHeap: Floyd's Method



Finally…



runtime:

More Priority Queue Operations

- **decreaseKey**
  - given a pointer to an object in the queue, reduce its priority value

  Solution: change priority and ________________________________

- **increaseKey**
  - given a pointer to an object in the queue, increase its priority value

  Solution: change priority and _________________________________

Why do we need a *pointer*? Why not simply data value?

More Priority Queue Operations

- **Remove(objPtr)**
  - given a pointer to an object in the queue, remove the object from the queue

Solution: set priority to negative infinity, percolate up to root and deleteMin

- **FindMax**

Facts about Heaps

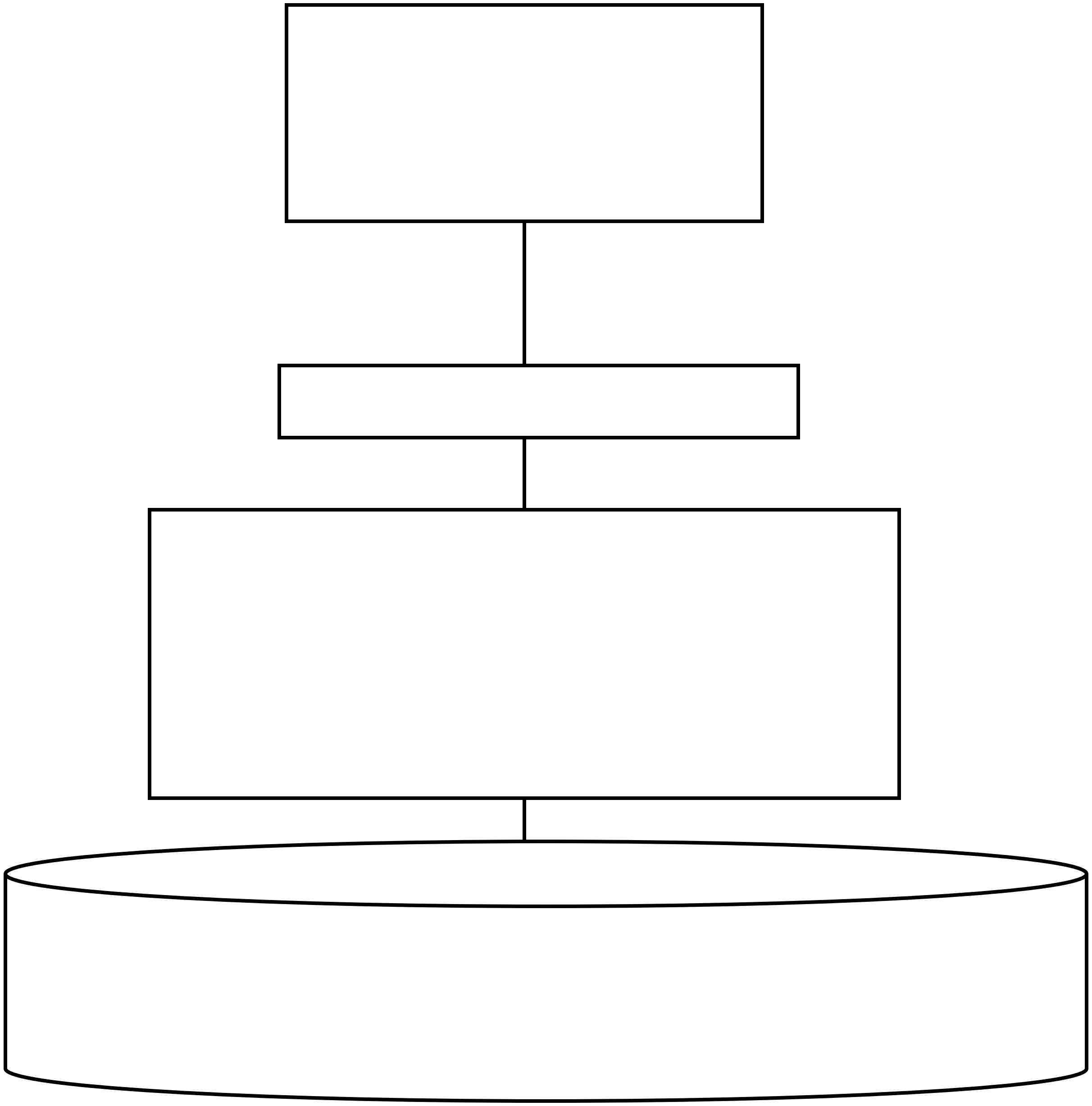
Observations:

- 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

Cycles to access:



CPU

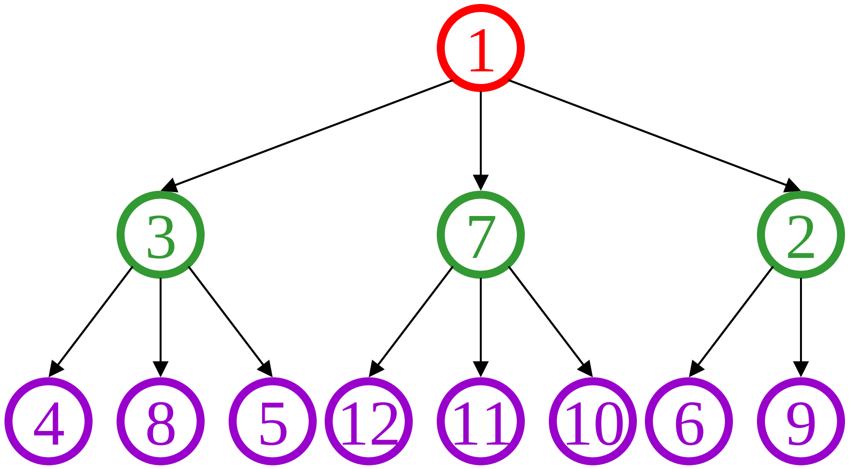
Cache

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



12	1	3	7	2	4	8	5	12	11	10	6	9

One More Operation

- Merge two heaps

- Add the items from one into another?
  - O(n log n)

- Start over and build it from scratch?
  - O(n)