CSE 332: Data Structures

Priority Queues – Binary Heaps Part II

Richard Anderson

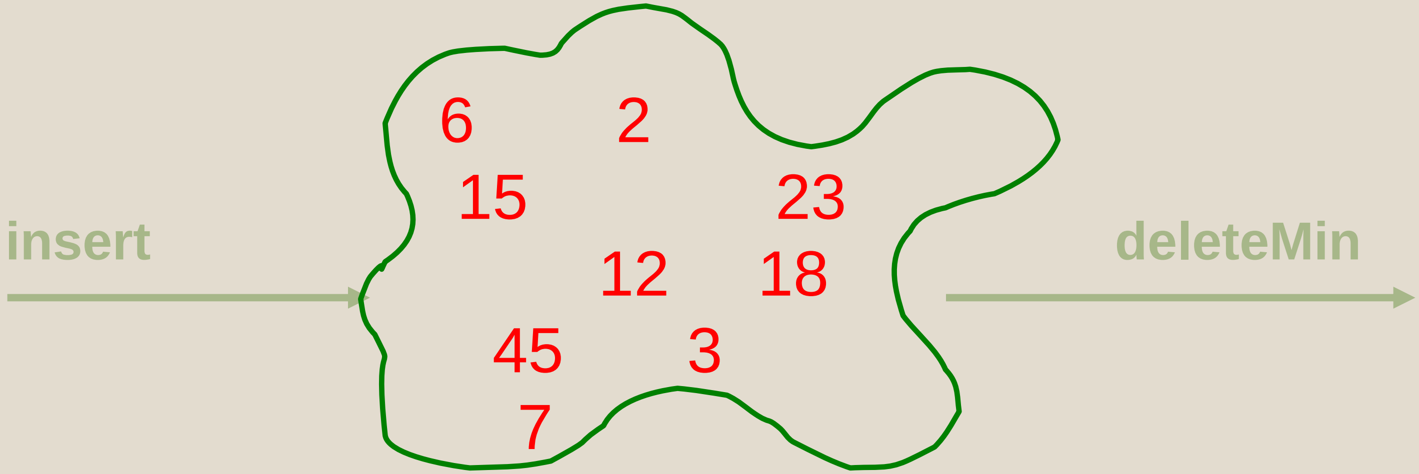
Spring 2016

Administrative

- Turn in HW1
- HW2 available
- P1 Due next Wednesday

Priority Queue ADT

- Insert(v)
- DeleteMin()

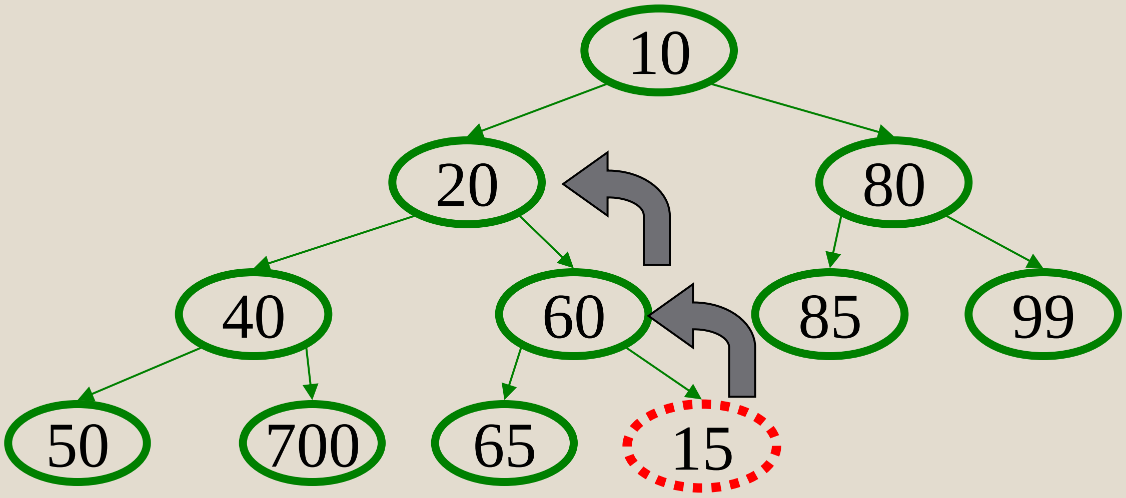


Binary Heap data structure

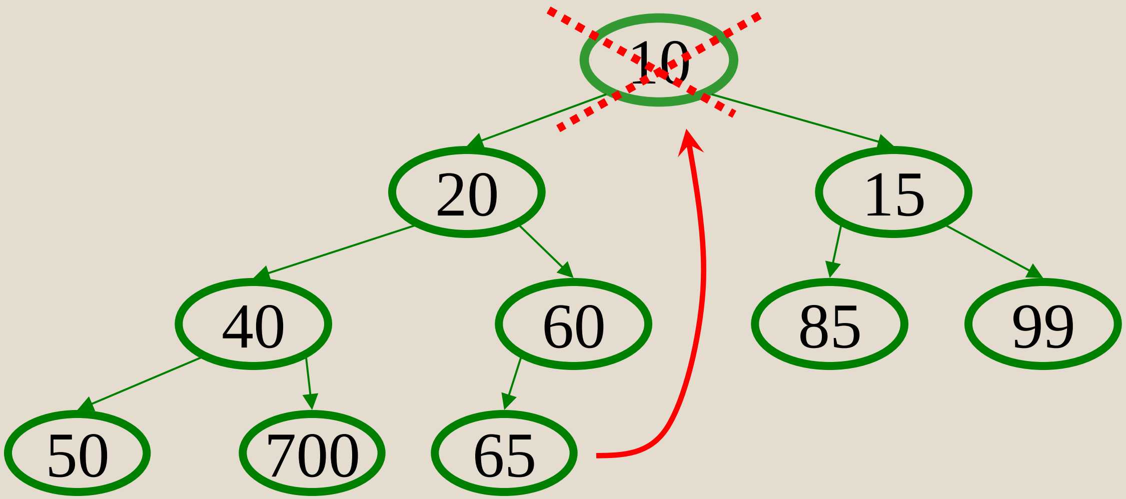
- binary heap for priority queues:
  - O(log n) worst case for both insert and deleteMin

- Heap properties
  - Complete binary tree
  - Value of a node less than or equal to the values of its children

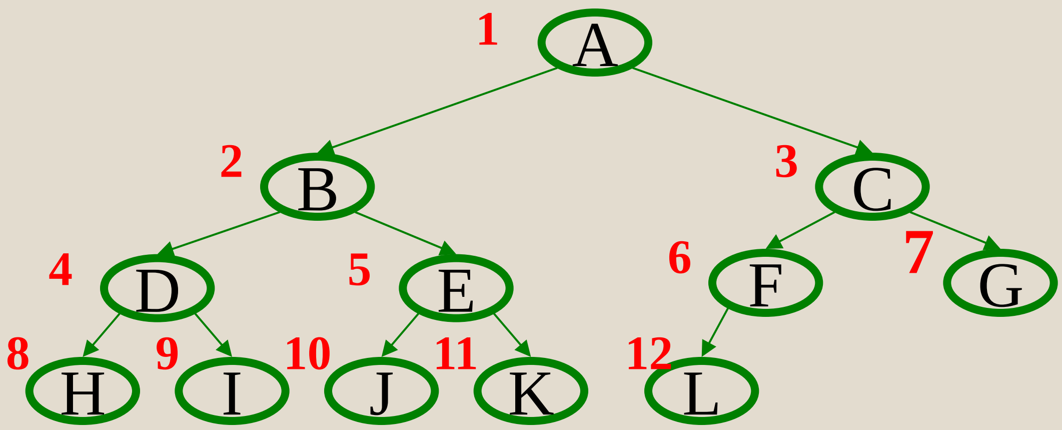
Insert: percolate up



DeleteMin: percolate down



Representing Complete Binary Trees in an Array



From node **i**:

left child:
right child:
parent:

	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

Why use an array?

DeleteMin Code

```
Object deleteMin() {
  assert(!isEmpty());
  returnVal = Heap[1];
  size--;
  newPos =
    percolateDown(1,
        Heap[size + 1]);
  Heap[newPos] =
    Heap[size + 1];
  return returnVal;
}
```

runtime:

(Java code in book)

```
int percolateDown(int hole,
                  Object val) {
while (2*hole <= size) {
    left = 2*hole;
    right = left + 1;
    if (right ≤ size &&
        Heap[right] < Heap[left])
      target = right;
    else
      target = left;

    if (Heap[target] < val) {
      Heap[hole] = Heap[target];
      hole = target;
    }
    else
      break;
  }
  return hole;
}
```

Insert Code

```
void insert(Object o) {
  assert(!isFull());
  size++;
  newPos =
    percolateUp(size,o);
  Heap[newPos] = o;
}
```

```
int percolateUp(int hole,
                Object val) {
  while (hole > 1 &&
         val < Heap[hole/2])
    Heap[hole] = Heap[hole/2];
    hole /= 2;
  }
  return hole;
}
```

runtime:

(Java code in book)

Insert: 16, 32, 4, 69, 105, 43, 2

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

More Priority Queue Operations

decreaseKey(nodePtr, amount):
given a pointer to a node in the queue, reduce its priority

Binary heap: change priority of node and ___________________

increaseKey(nodePtr, amount):
given a pointer to a node in the queue, increase its priority

Binary heap: change priority of node and ___________________

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

Worst case running times?

More Priority Queue Operations

remove(objPtr):

given a pointer to an object in the queue, remove it

Binary heap: ____

findMax( ):

Find the object with the highest value in the queue

Binary heap: ____

Worst case running times?

More Binary Heap Operations

expandHeap( ):

If heap has used up array, copy to new, larger array.

- Running time:

buildHeap(objList):

Given list of objects with priorities, fill the heap.

- Running time:

We do better with **buildHeap**...

Building a Heap: Take 1

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

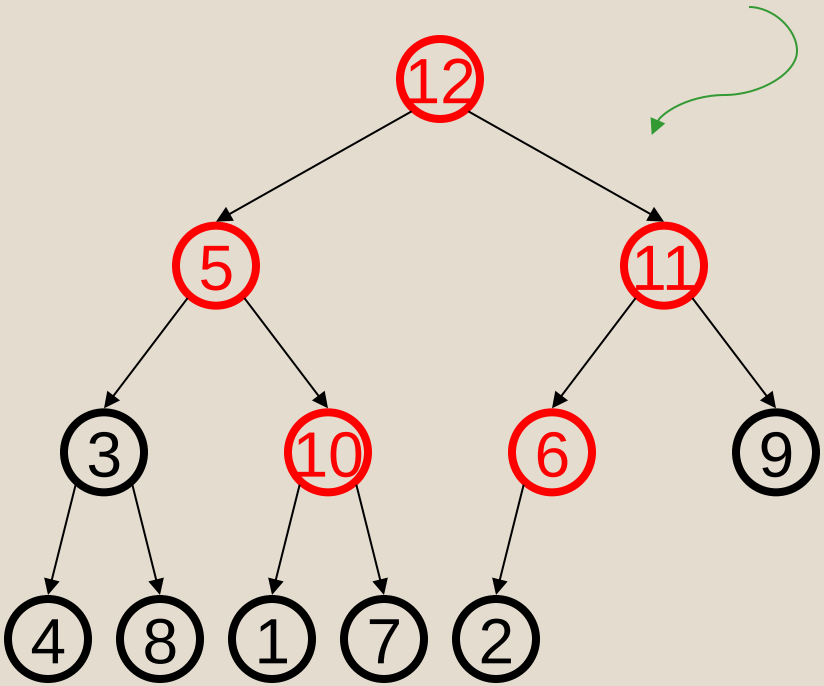
BuildHeap: Floyd's Method

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

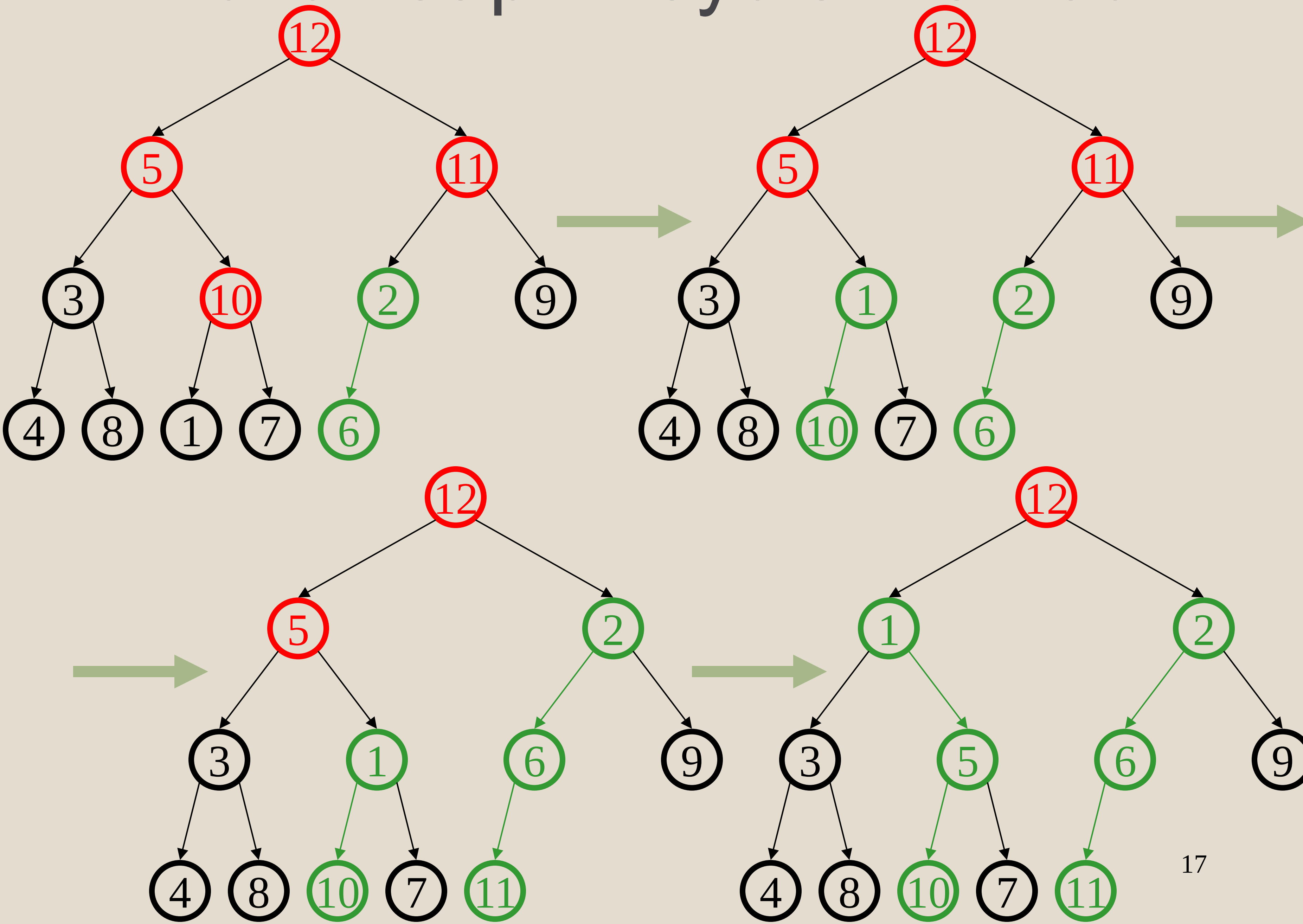
Add elements arbitrarily to form a complete tree.
Pretend it's a heap and fix the heap-order property!

Red nodes need to percolate down

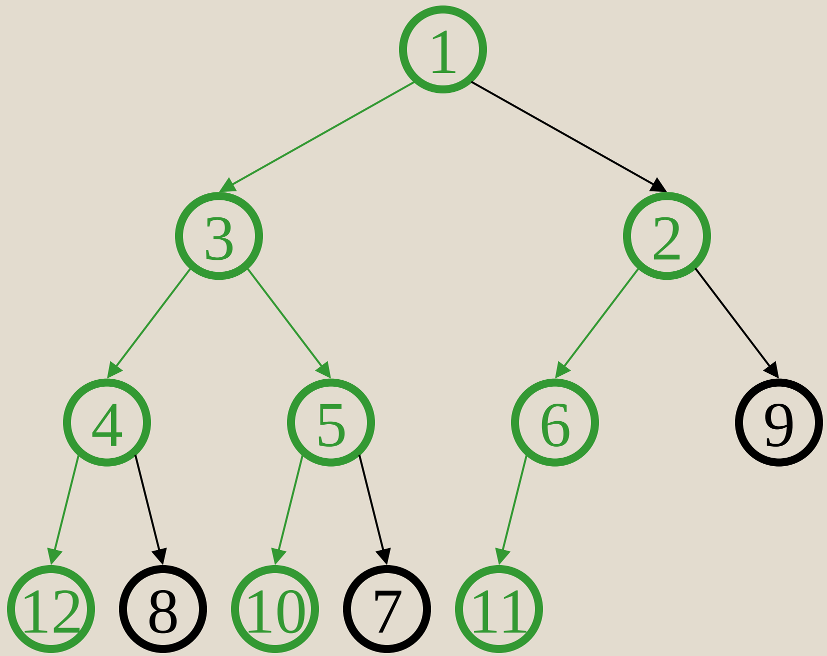
Key idea: fix red nodes from **bottom-up**



BuildHeap: Floyd's Method



Finally…



Buildheap pseudocode

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

runtime:

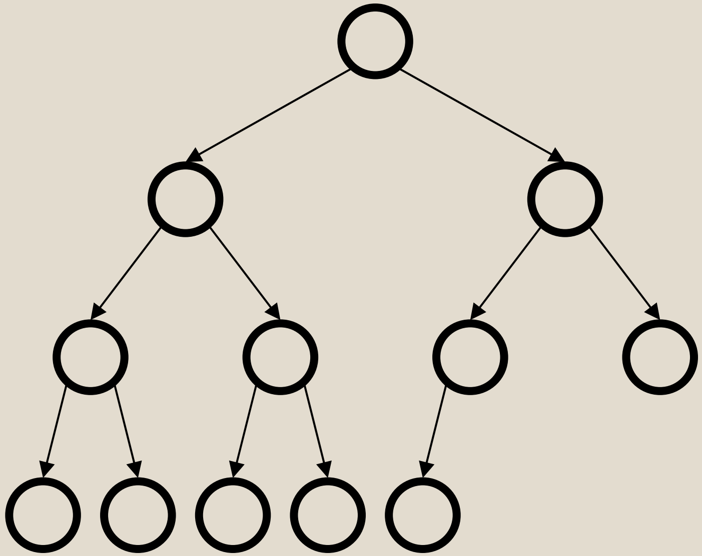
Buildheap Analysis

n/4 nodes percolate at most 1 level

n/8 percolate at most 2 levels

n/16 percolate at most 3 levels

...



runtime: