CS314H
DATA STRUCTURES

Lecture 19:
Priority Queue and Heap

Mikyung Han

Please, interrupt and ask questions AT ANY TIME !

Outline

1. Binary Tree Analysis

Complete Binary Tree

- All the levels of the tree are filled completely except the lowest level
  - At the lowest level, nodes are filled from as left as possible
- Which is NOT a complete binary tree?
  - Num nodes at level k?
  - Min num nodes at the lowest level?

```
     1
    / \
   2   3
  / \  /
 4   5 6
```

```
        10
       /  \
     20    30
    /  \  / \
   40  50 60 70
  / \  /
80 90 100
```

```
   1
  / \
 2   3
  \
   4
```

Complete Binary Tree

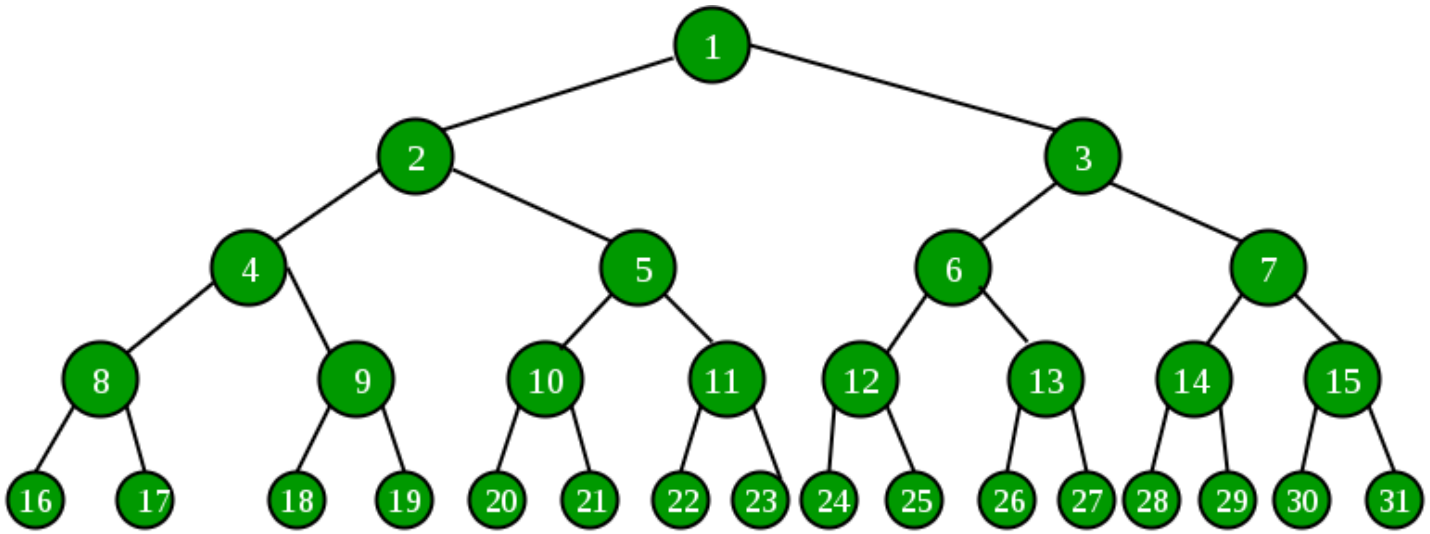
- All the levels of the tree are filled completely except the lowest level
  - At the lowest level, nodes are filled from as left as possible
- Num nodes at level k?
- Min/max num nodes at the lowest level?

Complete Binary Tree

- **All the levels of the tree are filled completely except the lowest level**
  - At the lowest level, nodes are filled from as left as possible
- **Num nodes at level k?**
  - Root is at level 0
  - 2^k
- **Min/max num nodes at the lowest level?**
  - Let L be the number of levels
  - Min: 1
  - Max: 2^{L-1}

Perfect Binary Tree

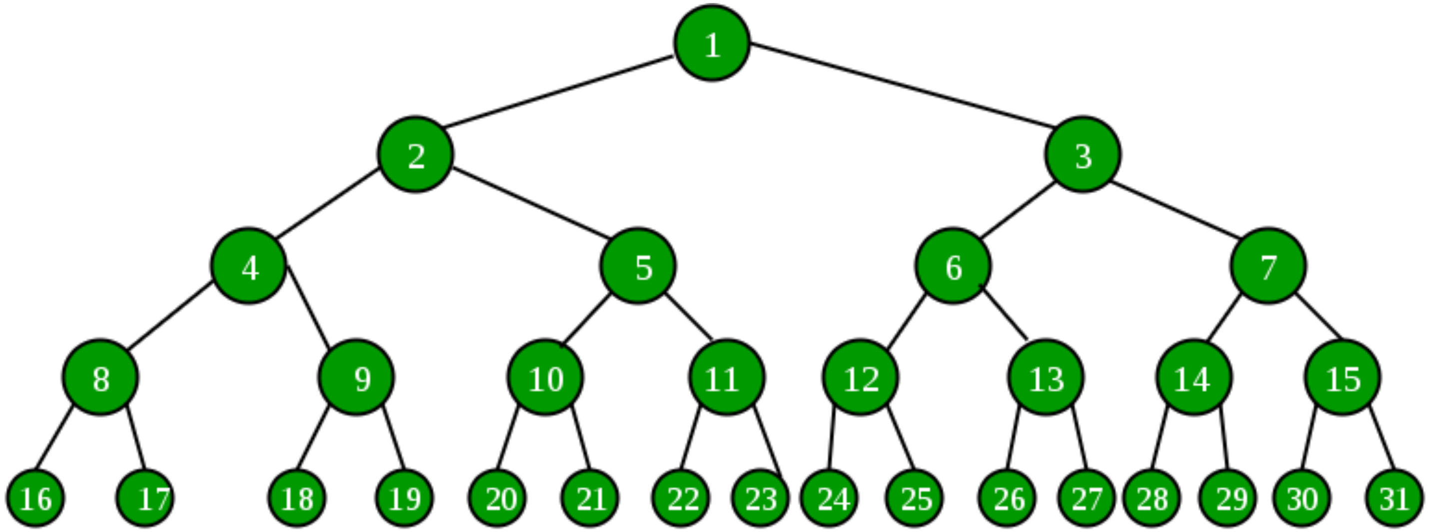
- All the levels of the tree are filled completely including the lowest level



- Total number of nodes given there are total L levels?
- How many leaf nodes?
- How many internal nodes?

Perfect Binary Tree

- All the levels of the tree are filled completely including the lowest level



Let L be the number of levels

- Total 2^L-1 nodes
- 2^{L-1} leaf nodes
- 2^{L-1}-1 internal nodes

Outline

1. Binary Tree Analysis
2. Priority Queue

FIFO Queue ADT

Queue (FIFOQueue) ADT

`enqueue(`**val**`)`	Adds **val** to the queue.
`dequeue()`	Returns the **least-recent** item not already returned by a `dequeue`. (Errors if empty.)
`peek()`	Returns the **least-recent** item not already returned by a `dequeue`. (Errors if empty.)
`isEmpty()`	Returns true if all inserted elements have been returned by a `dequeue`.

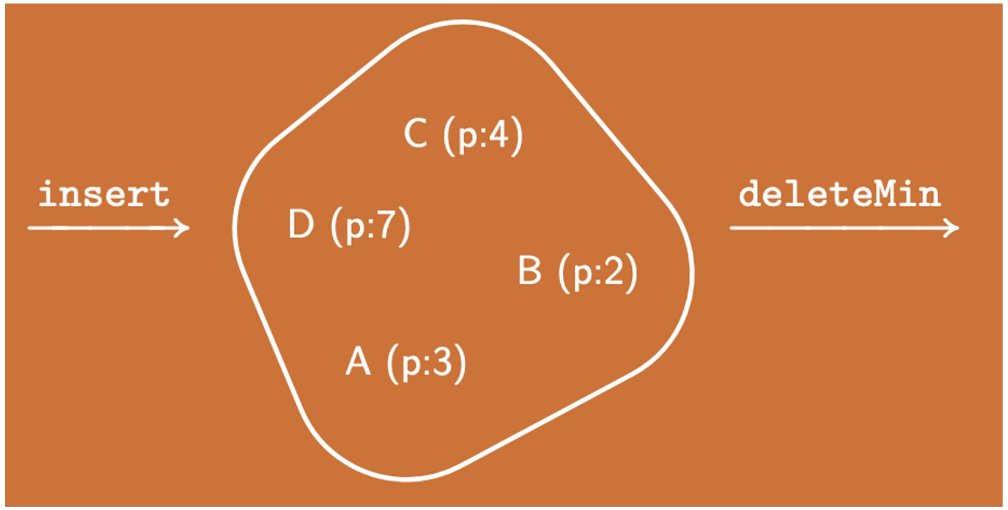
- **But sometimes we're interested in a PriorityQueue instead**
  - A hospital ER room
  - OS process scheduling
  - Network packet routing
  - Discrete event simulation
  - etc

Priority Queue ADT

PriorityQueue ADT	
`insert(val)`	Adds **val** to the queue.
`deleteMin()`	Returns the **highest priority** item not already returned by a `deleteMin`. (Errors if empty.)
`findMin()`	Returns the **highest priority** item not already returned by a `deleteMin`. (Errors if empty.)
`isEmpty()`	Returns true if all inserted elements have been returned by a `deleteMin`.

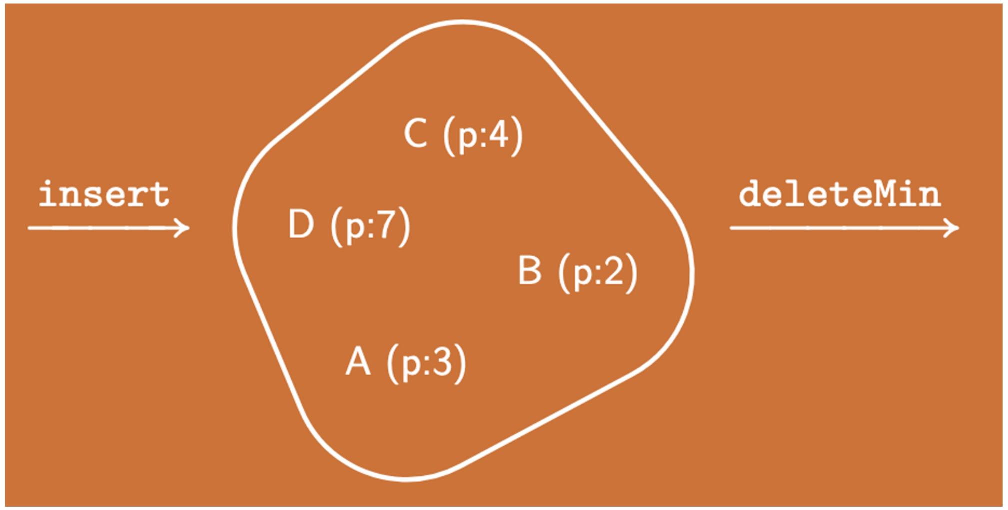
- Data in PriorityQueue must be comparable
- Highest priority == lowest priority value
- ADT does not specify tie case

Priority Queue Example




```
findMin
deleteMin
insert(E (p:1))
deleteMin
deleteMin
```

Priority Queue Example




```
findMin → B
deleteMin → B
insert(E (p:1))
deleteMin → E
deleteMin → A
```

Now let's talk about the implementation

Here are some options. What is the cost of insert and deleteMin?

- Unsorted Array/Unsorted Linked List
- Sorted Circular Array List
- Sorted Linked List
- Binary Search Tree
- AVL?

Here are some options.
What is the cost of insert and deleteMin?

- Unsorted Array
  - Insert at the end O(1) for Array, or at the front O(1) for LinkedList
  - deleteMin: linear search O(n) + shift O(n) for Array. No shifting for LinkedList.
- Sorted Circular Array List
  - Insert by binary search and shift O(n)
  - deleteMin: move front no shifting needed O(1)
- Sorted Linked List
  - Insert by linear search O(n)
  - deleteMin: remove front and update front to front->next O(1)
- Binary Search Tree
  - Insert walk down to the leaf O(n). Tree is not balanced
  - deleteMin: findMin and replace min with min's right child. O(n). Tree not balanced
- AVL
  - Insert: Go to the correct leaf node to insert O(log n) and rebalance O(1)
  - deleteMin: Find left most node O(log n) delete and rebalance. O(1)

Better data structure needed!!

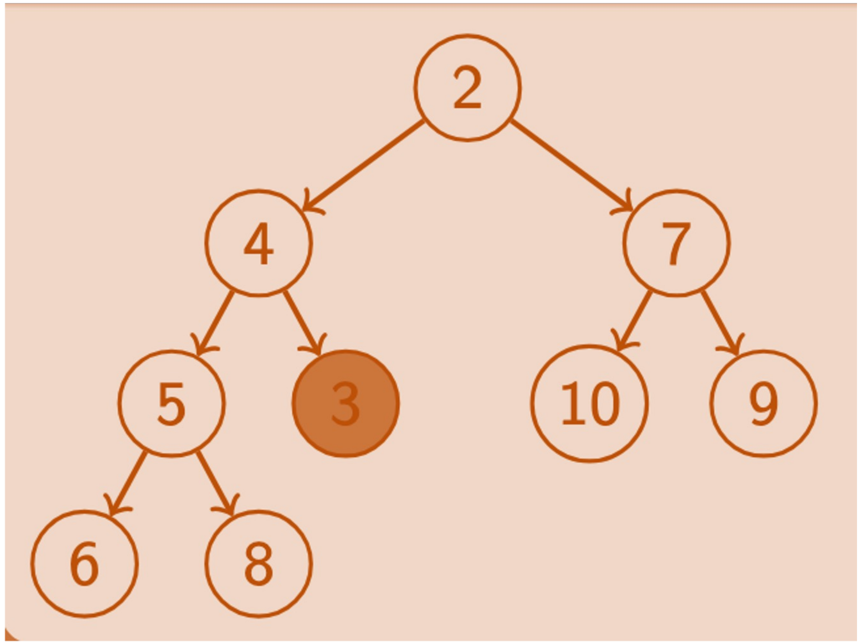
Outline

1. Binary Tree Analysis
2. Priority Queue
3. Heap

Heap has two properties

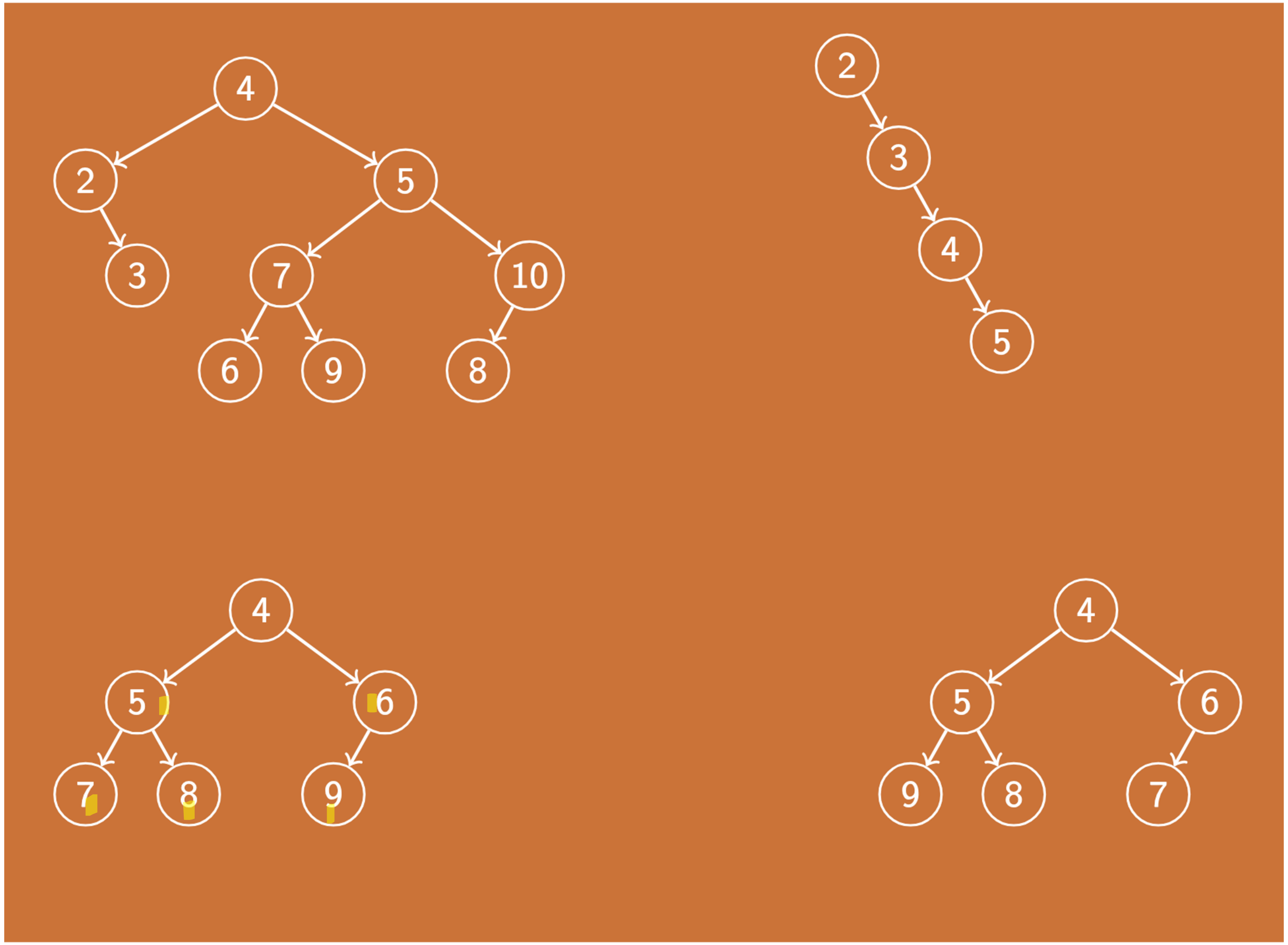
Heap Property:
All Children are larger

Structure Property:
Insist the tree has no "gaps"



- Where would be the min?
- Is this a heap?

Which ones are heap?



Heap properties

- Where is the minimum item in a heap?
  - At the root!

- What is the height h of a heap with n elements?
  - How many levels in this tree? $L = h+1$
  - Total number of nodes $n \approx 2^{(h+1)}-1$ (assuming perfect tree)
  - $h = O(\log_2 n)$

- FindMin is easy, how to insert, or deleteMin?

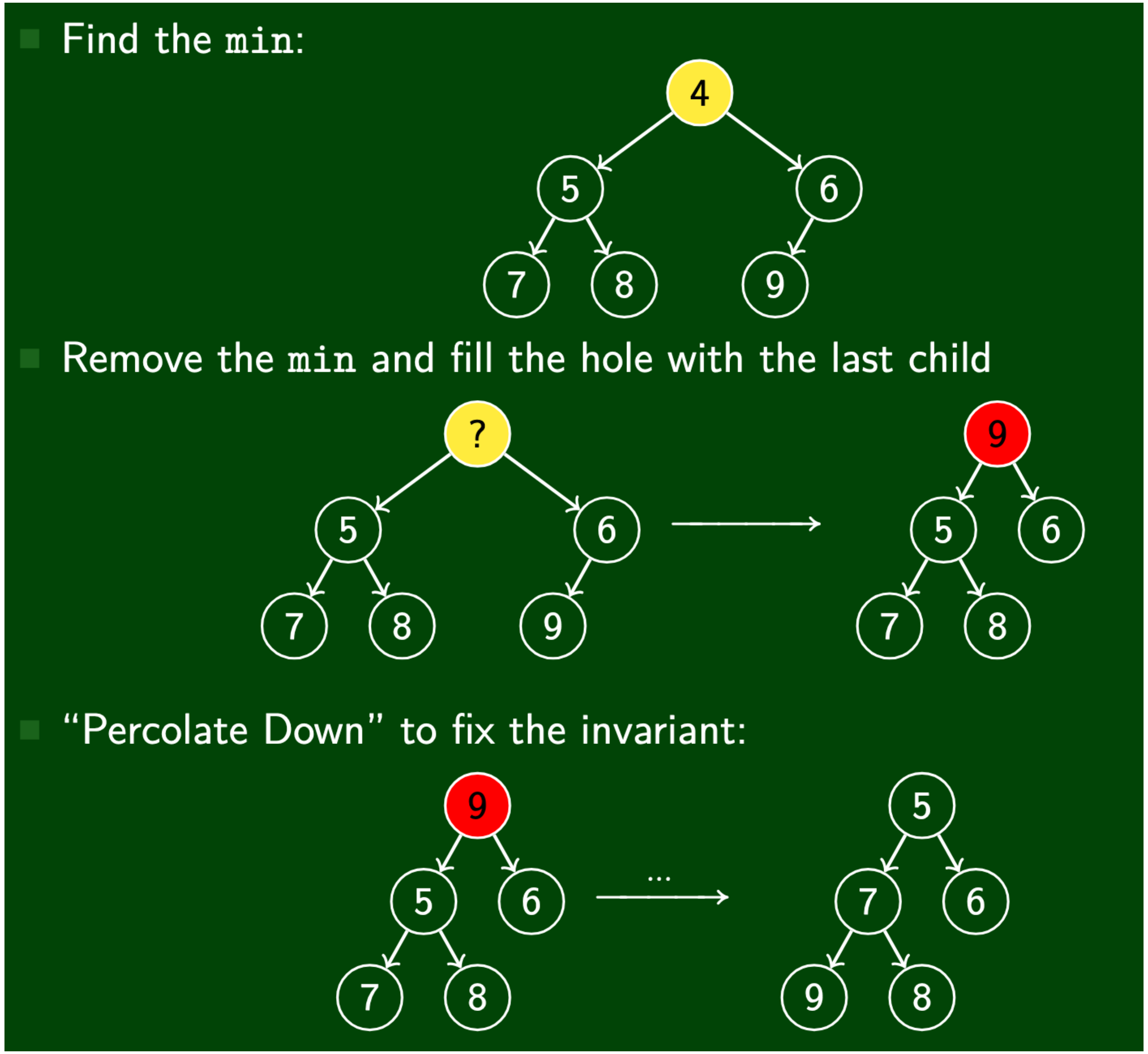
DeleteMin

- Step 1: find the min

- Step 2: To preserve structure property who should fill the root?
  - Hint: we do not want to shift O(n) times.
  - The last node is promoted as the root!

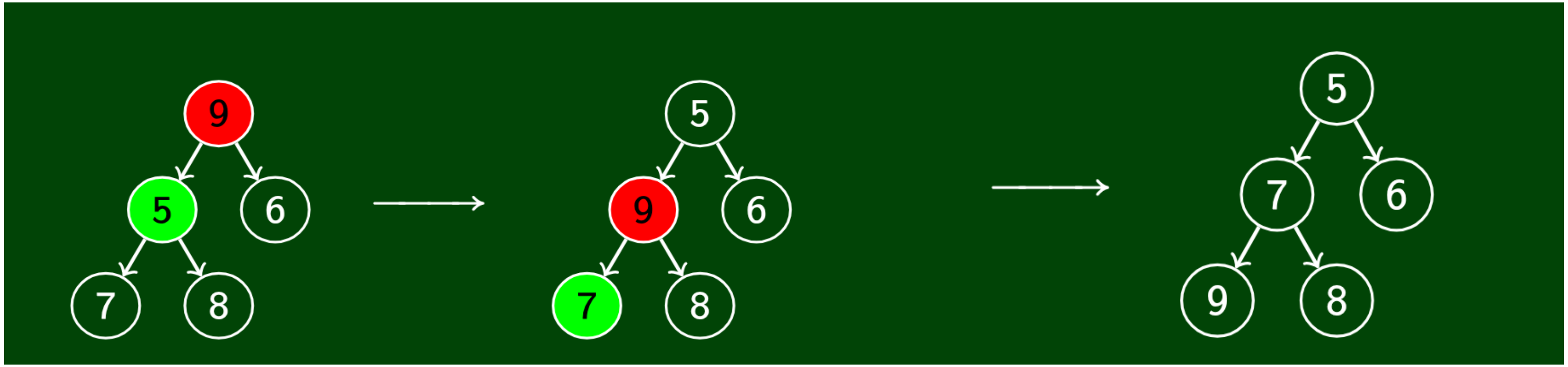
- Step 3: "Percolate down" to fix
  - To preserve heap property - all children is larger than itself
  - Pick one: Swap data with its [smaller or larger] child

DeleteMin Example

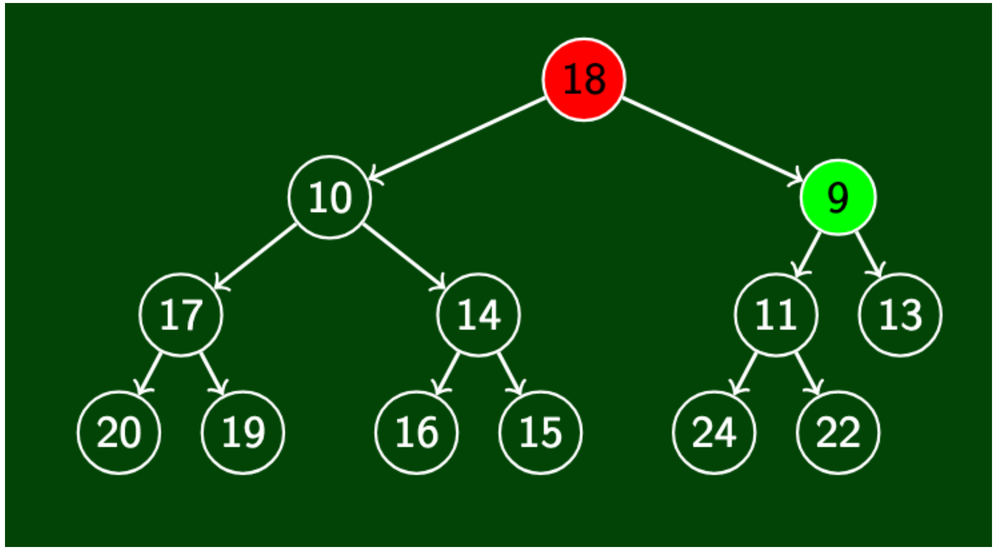


“Percolate down”

```
percolateDown(node) {
    while (node.data is greater than either child) {
        swap data with smaller child
    }
}
```



“Percolate down” another example



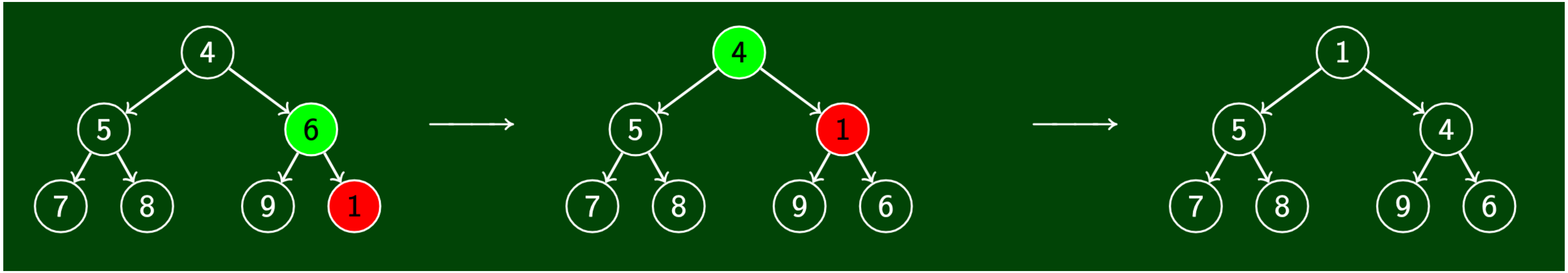
What is the time complexity of percolate down?

Insert

- **Step 1: To preserve structure property where to insert?**
  - Hint: we do not want to shift O(n) times.
  - The the very end!
- **Step 2: ”Percolate up” to fix**
  - If my data is smaller than the parent’s, swap data!

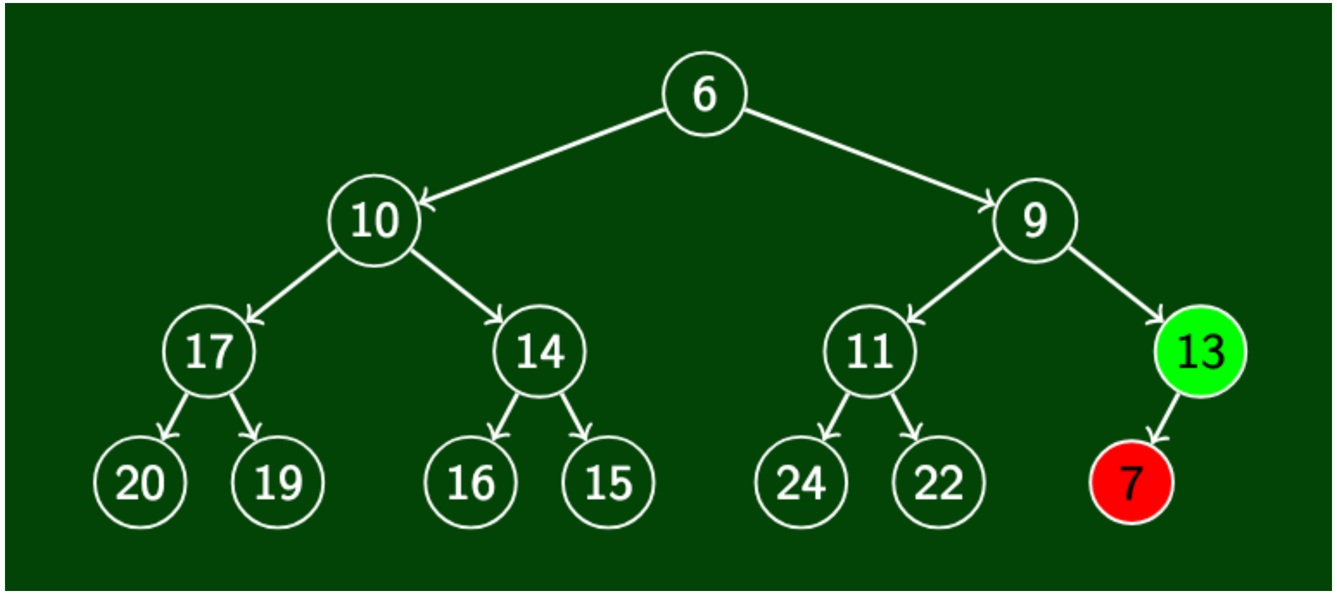
“Percolate Up”

```
percolateUp(node) {
    while (node.data is smaller than parent) {
        swap data with parent
    }
}
```



What is the time complexity of percolate up?

Percolate up another example



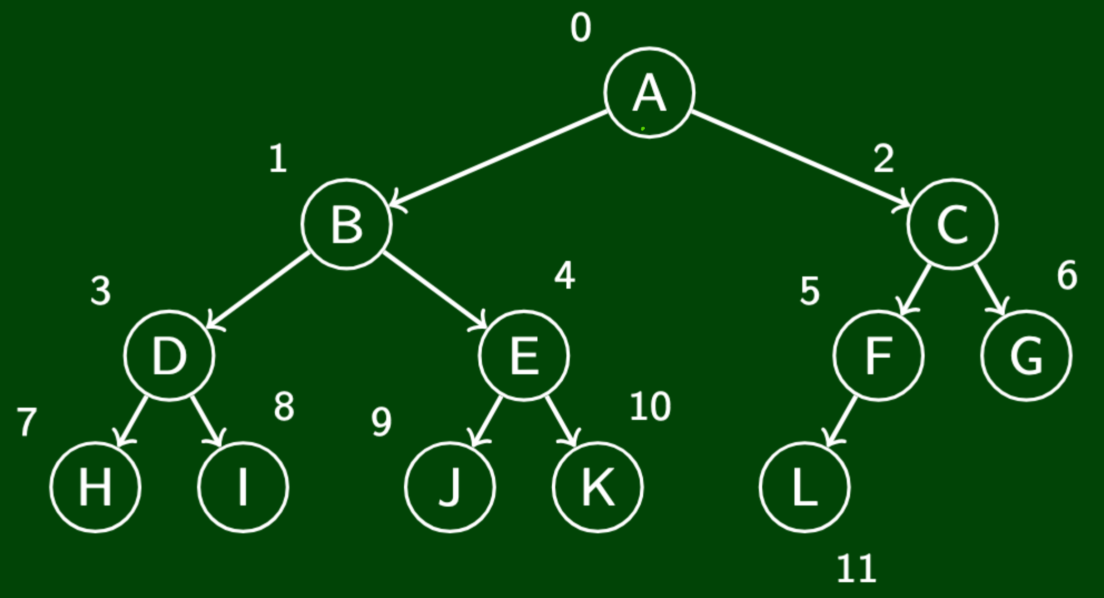
How to implement Heap?

Which option would you choose?

- Option 1: Use TreeNode with left/right references
- Option 2: Use array representation

Use array! Fill in array by level-order

Heap array



Fill in an array in **level-order** of the tree:

heap:	A	B	C	D	E	F	G	H	I	J	K	L	0	0	0
	h[0]	h[1]	h[2]	h[3]	h[4]	h[5]	h[6]	h[7]	h[8]	h[9]	h[10]	h[11]	h[12]	h[13]	h[14]

Outline

1. Binary Tree Analysis
2. Priority Queue
3. Heap analysis

Recap

- **Heap property**
  - Relationship between root and its children
  - Relationship between siblings
- **Structure property**
  - Complete binary tree

- **Insert**
  - Where do you insert to satisfy the structure property?
  - Percolate up or down to satisfy the heap property?

Average Case Analysis: Insert to a heap with n nodes

- How many nodes …
  - in the last level?
  - 2^{nd} to the last level?
  - 3^{rd} to the last level?
- What are the chances that the node just inserted eventually land …
  - On the last level?
  - 2^{nd} to the last level?
  - 3^{rd} to the last level?
  - What is the number of comparisons for each case?
- For average case, we pick a random value x
  - ½ probability that we compare once
  - ¼ probability that we compare twice

Average Case Analysis: Insert to a heap with n nodes

- For average case, we pick a random value x to insert
  - 1/2 probability that we compare once
  - 1/4 probability that we compare twice
  - 1/8 probability that we compare 3 times
  - Etc.
- Sum
  - $1/2 + 2/4 + 3/8 + 4/16 + 5/32 \ldots = \sum_{k=1}^{\infty} \frac{k}{2^k}$

Will this converge?

Average Case Analysis: Insert to a heap with n nodes

- Let Z = $\sum_{k=1}^{\infty} \frac{k}{2^k}$ = 1/2 + 2/4 + 3/8 + 4/16 + 5/32 …
- Div Z by 2: Z/2 = 1/4 + 2/8 + 3/16 + 4/32 + 5/64 …
- Z – Z/2= S
- S = 1/2 + 1/4 + 1/8 + 1/16 + 1/32 …
- S-S/2 = 1/2
- S = 1
- Z = 2

Constant time! What does this mean?

Outline

1. Administrative
2. Heap analysis
3. Other possible operations of Heap

More operations to support besides insert, findMin, deleteMin

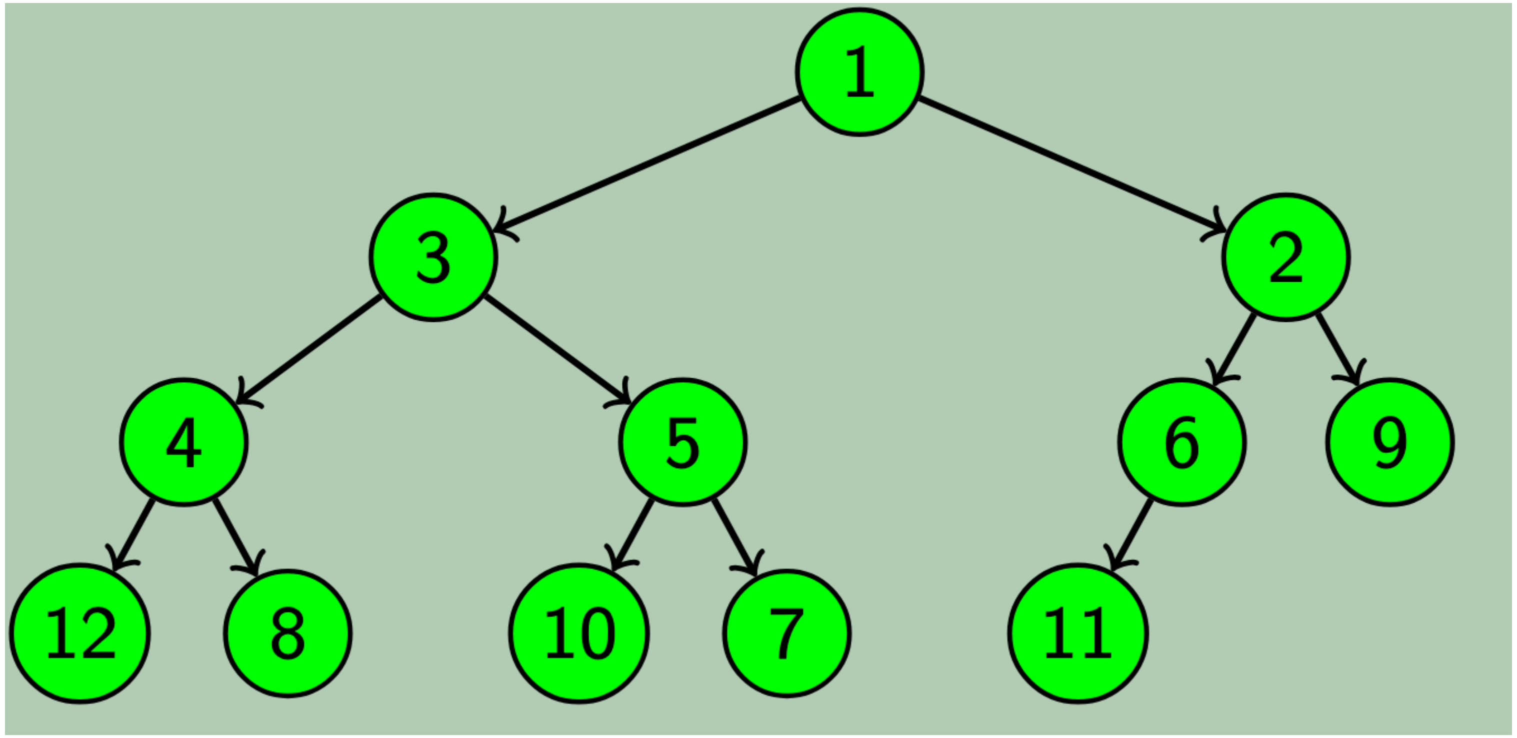
Given a particular index i of the array

- increasePriority(i, newPriority)
  - i has now higher priority
- decreasePriority(i, newPriority)
  - i has low lower priority
- buildHeap(array)
  - Where array is unsorted elements

How would you implement them? Time complexity?

Increase/Decrease Priority Example

- Increase priority: Change 6 to 0
- Decrease priority: Change 1 to 15



Outline

1. Administrative
2. Heap analysis
3. Other possible operations of Heap
4. Building heap

Building heap: Simply use insert

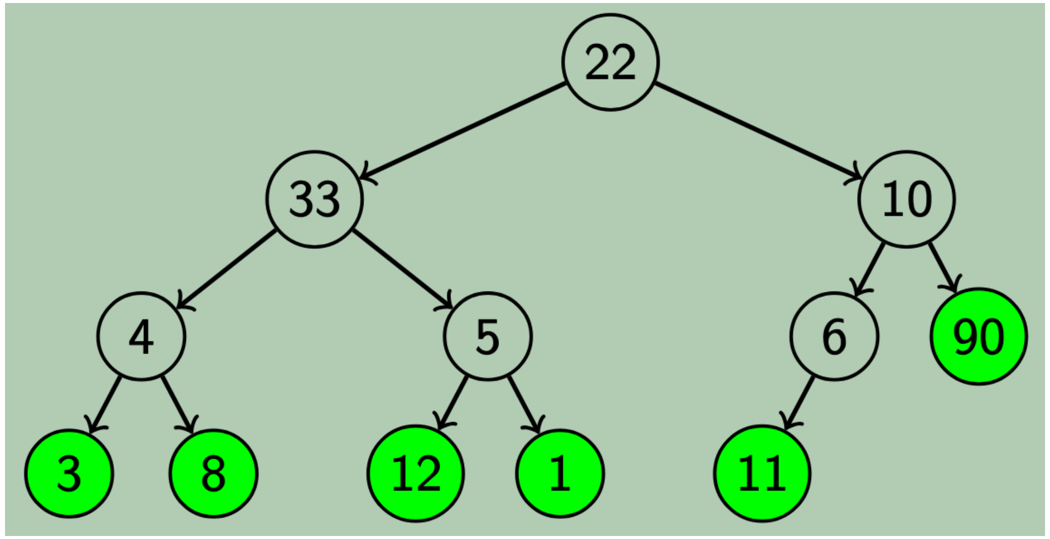
The Easy Way. . .

```
void buildHeap(int[] input) {
    for (int i = 0; i < input.length; i++) {
        insert(input[i]);
    }
}
```

What is the time complexity?

Floyd's algorithm

- Rather than insert and fix the heap one by one
- Insert all and fix all at once
  - input array itself is a heap to be fixed
- Example
  - input array = [22, 33, 10, 4, 5, 6, 90, 3, 8, 12, 1, 11]
  - Note leaf nodes are already valid heap!



How to fix?

Floyd's algorithm

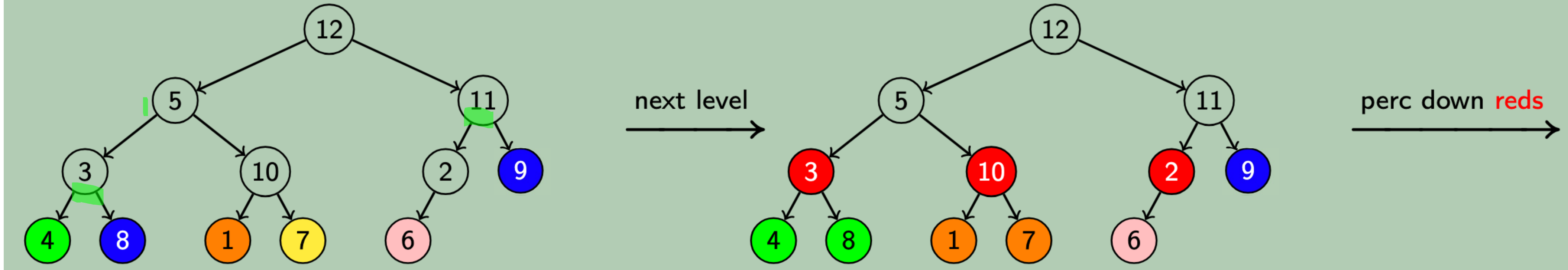
- **Insert all and fix all at once**
  - input array itself is a heap to be fixed
  - Starting with a largest index node with children, percolate down!

```
void buildHeap(int[] input) {
    for (i = size/2 - 1; i >= 0; i--) {
        percolateDown(i);
    }
}
```

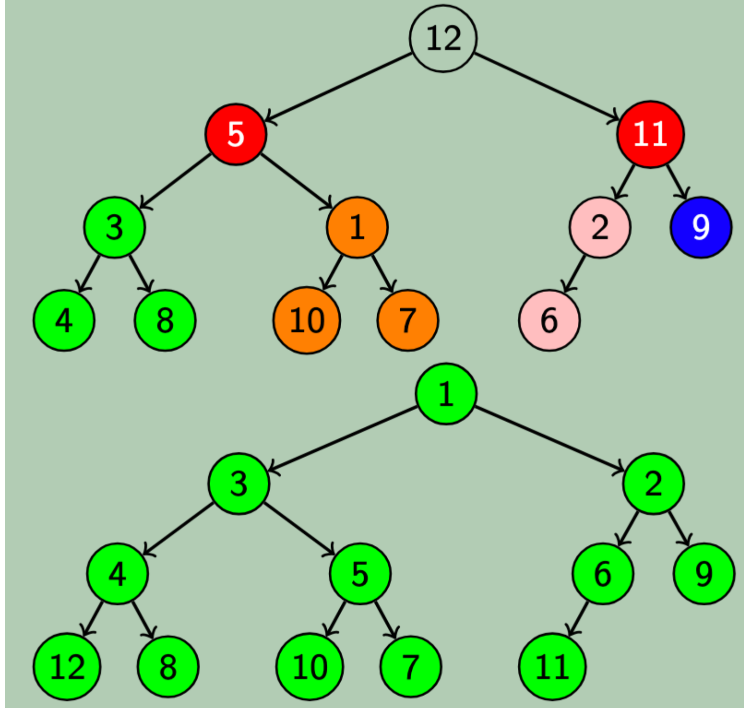
Floyd's `buildHeap`

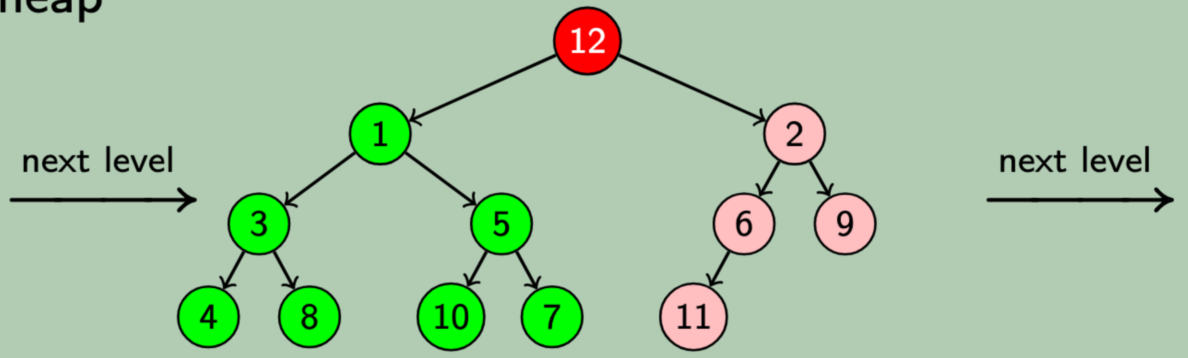
Each highlighted node is a valid heap!

Percolate down red nodes until at top



Each color represents a valid heap





Time Complexity of Floyd's algorithm

- Is Floyd's algorithm better?

```
void buildHeap(int[] input) {
    for (i = size/2 - 1; i >= 0; i--) {
        percolateDown(i);
    }
}
```

```
void buildHeap(int[] input) {
    for (int i = 0; i < input.length; i++) {
        insert(input[i]);
    }
}
```

A tighter worst-case analysis of Floyd's algorithm

- How many nodes on the lowest level?
  - n/2
- How many nodes on the 2nd lowest level?
  - n/4
- How many nodes in the 3rd lowest level?
  - n/8
- How many nodes in the 4th lowest level?
  - n/16
- Etc…

A tighter worst-case analysis of Floyd's algorithm

- Max num percolation for nodes on the lowest level?
  - 0
- Max num percolation for nodes on the 2^{nd} lowest level?
  - 1
- Max num percolation for nodes on the 3^{rd} lowest level?
  - 2
- Max num percolation for nodes on the 4^{th} lowest level?
  - 3
- Etc..

Putting it together..

A tighter worst-case analysis of Floyd's algorithm

- $1 \times n/4 + 2 \times n/8 + 3 \times n/16 + \ldots = nS$
- $S - S/2$
  - Do coloring or telescoping to solve S
- $S = 1$

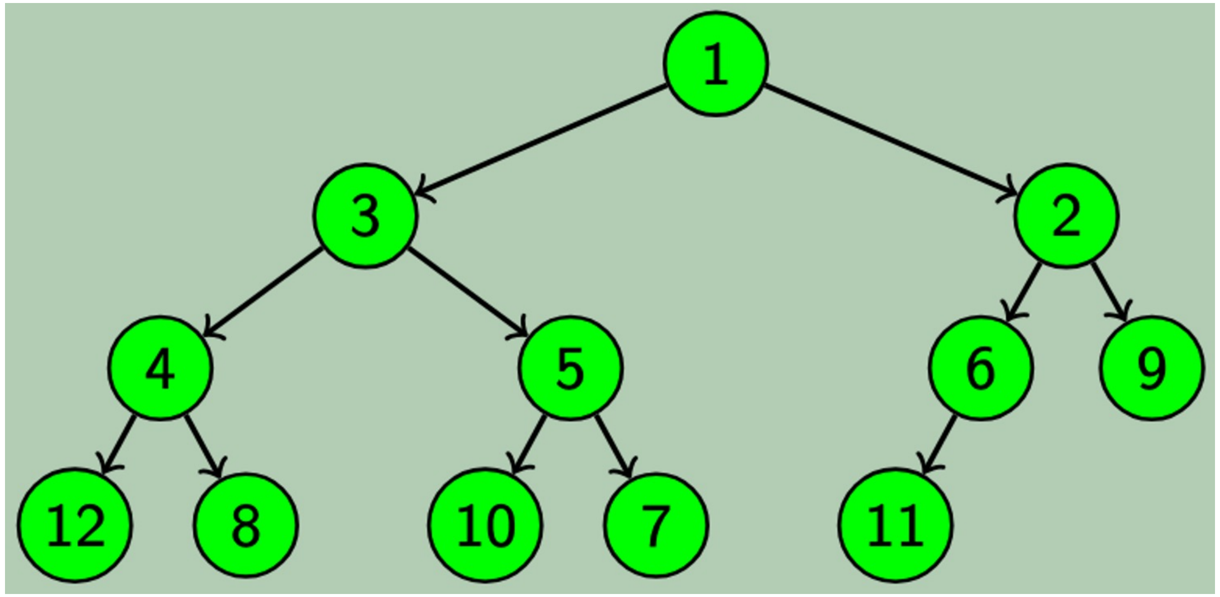
Floyd's Algorithm is O(n) in the WORST case,
which is better than inserting one by one n times

Outline

1. Administrative
2. Heap analysis
3. Other possible operations of Heap
4. Building heap
5. Heapsort

How to use heap to do sorting?

- Treat input array as a complete binary tree
- Heapify using Floyd's algorithm
- Call DeleteMin n times!
    - Instead of actually removing, keep the item on the back of the unsorted array



- Use Max heap sort ascending order