CSE 373 SP 18 - KASEY CHAMPION

Lecture 15: Intro to Heaps

CSE 373 Data Structures and Algorithms

Administrivia

Midterm grades out later this week

HW 4 due Wednesday night

HW 5 out Wednesday (partner project)

Heaps

Priority Queue ADT

Imagine you have a collection of data from which you will always ask for the extreme value

If a Queue is “First-In-First-Out” (FIFO) Priority Queues are “Most-Important-Out-First”
Example: Triage, patient in most danger is treated first

Items in Queue must be comparable, Queue manages internal sorting

Min Priority Queue ADT

state

Set of comparable values
- Ordered based on “priority”

behavior

removeMin() – returns the element with the smallest priority, removes it from the collection

peekMin() – find, but do not remove the element with the smallest priority

insert(value) – add a new element to the collection

Max Priority Queue ADT

state

Set of comparable values
- Ordered based on “priority”

behavior

removeMax() – returns the element with the largest priority, removes it from the collection

peekMax() – find, but do not remove the element with the largest priority

insert(value) – add a new element to the collection

Let’s start with an AVL tree

AVLPriorityQueue<E>

state

`overallRoot`

behavior

`removeMin()` - traverse through tree all the way to the left, remove node, rebalance if necessary

`peekMin()` - traverse through tree all the way to the left

`insert()` - traverse through tree, insert node in open space, rebalance as necessary

What is the worst case for peekMin()? **O(logn)**

What is the best case for peekMin()? **O(logn)**

Can we do something to guarantee best case for removeMin() and peekMin()?

Binary Heap

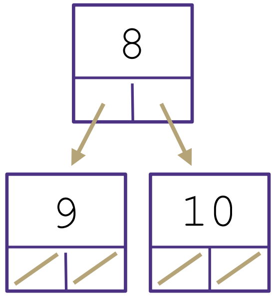
A type of tree with new set of invariants

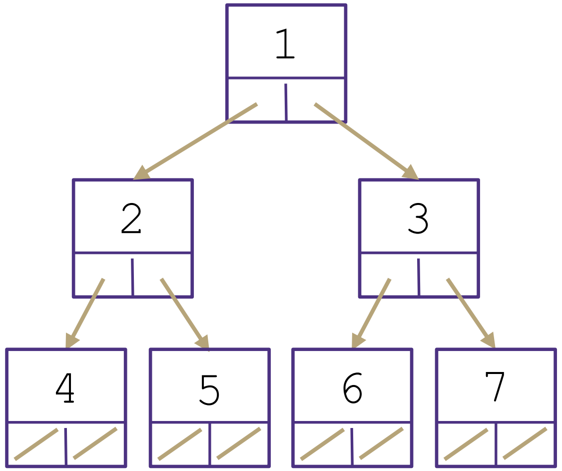
1. Binary Tree: every node has at most 2 children

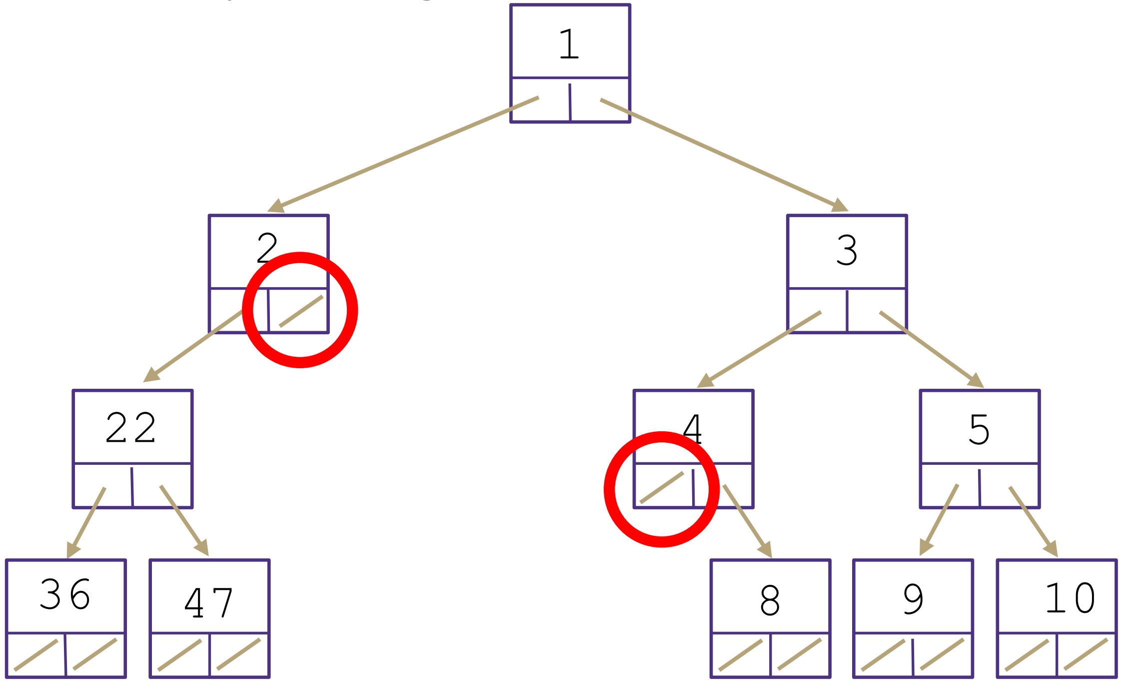
2. Heap: every node is smaller than its child

3. Structure: Each level is “complete” meaning it has no “gaps”

- Heaps are filled up left to right



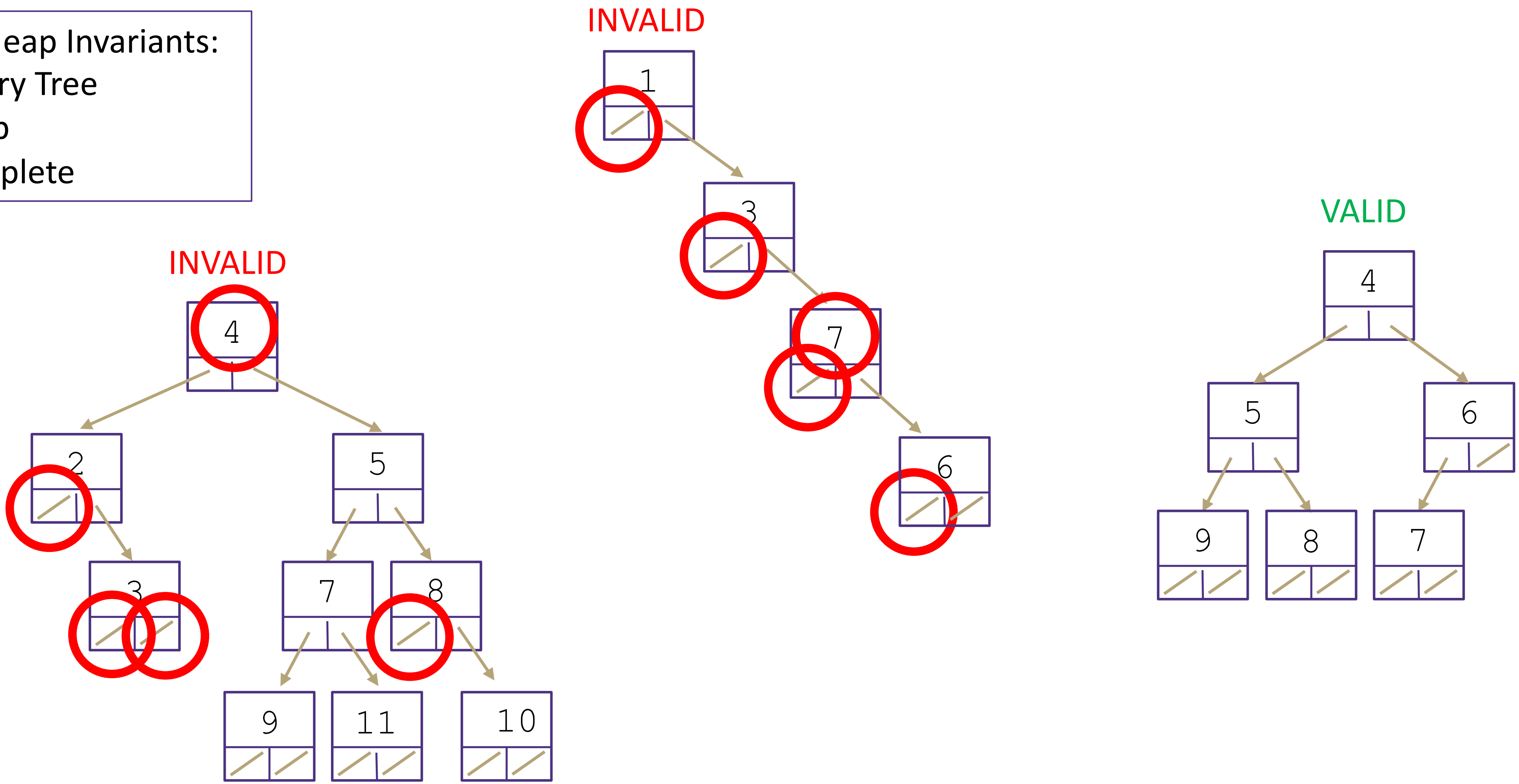




Self Check - Are these valid heaps?

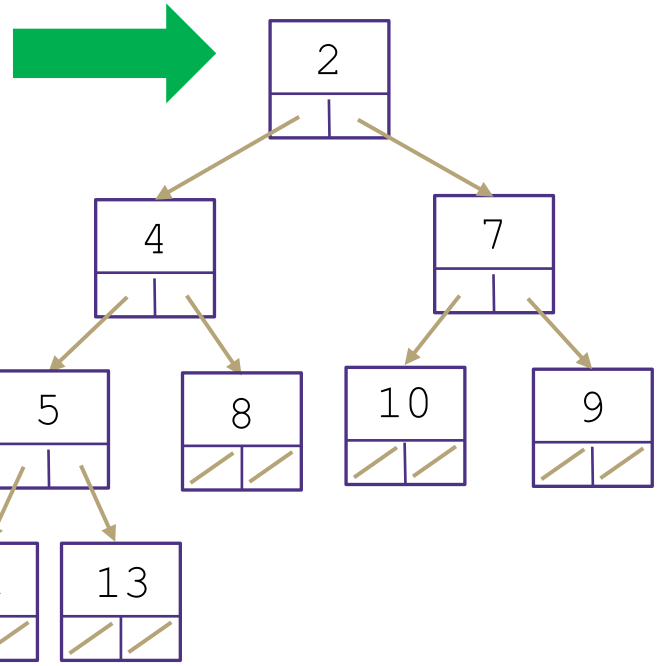
3 Minutes

Binary Heap Invariants:
1. Binary Tree
2. Heap
3. Complete

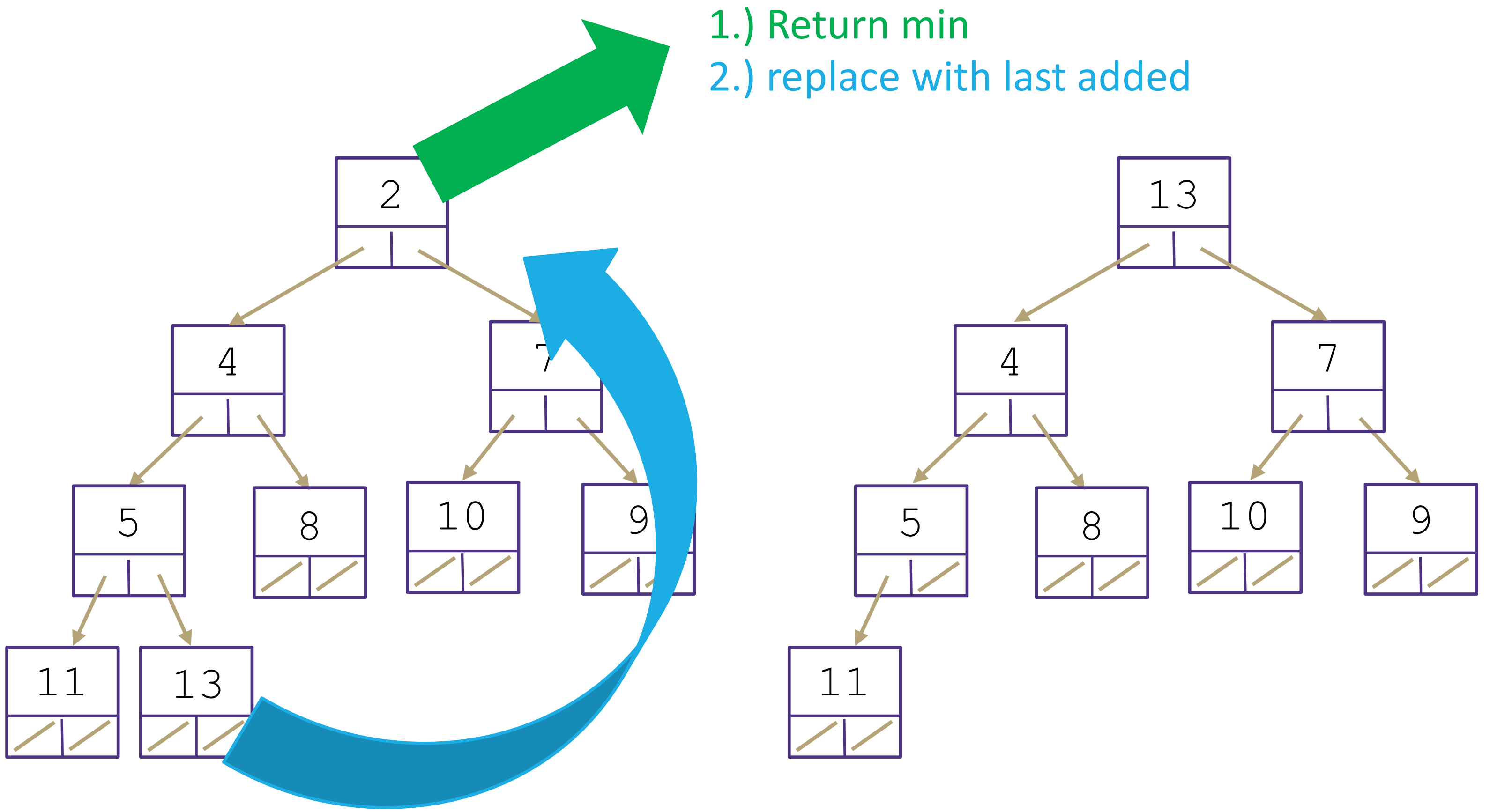


Implementing peekMin()

Runtime: O(1)



Implementing removeMin()

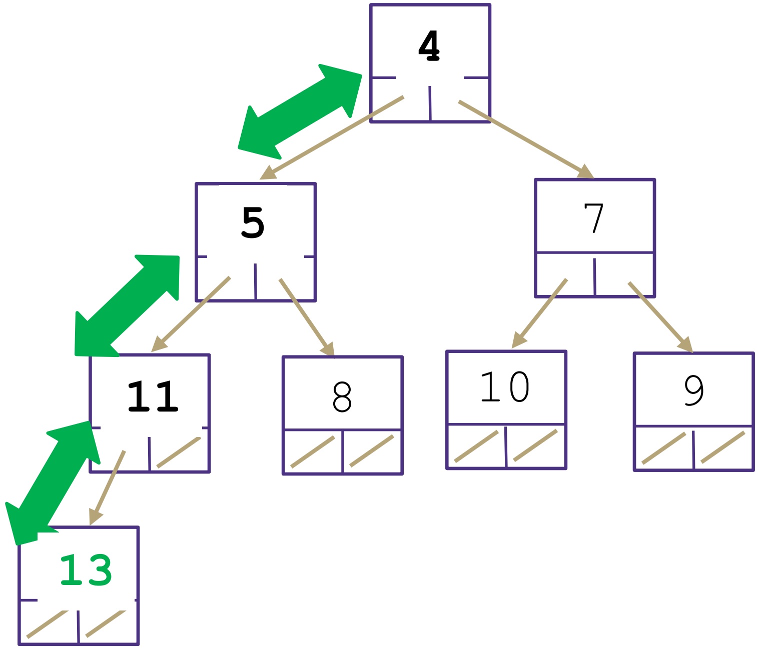


Structure maintained, heap broken

Implementing removeMin() - percolateDown

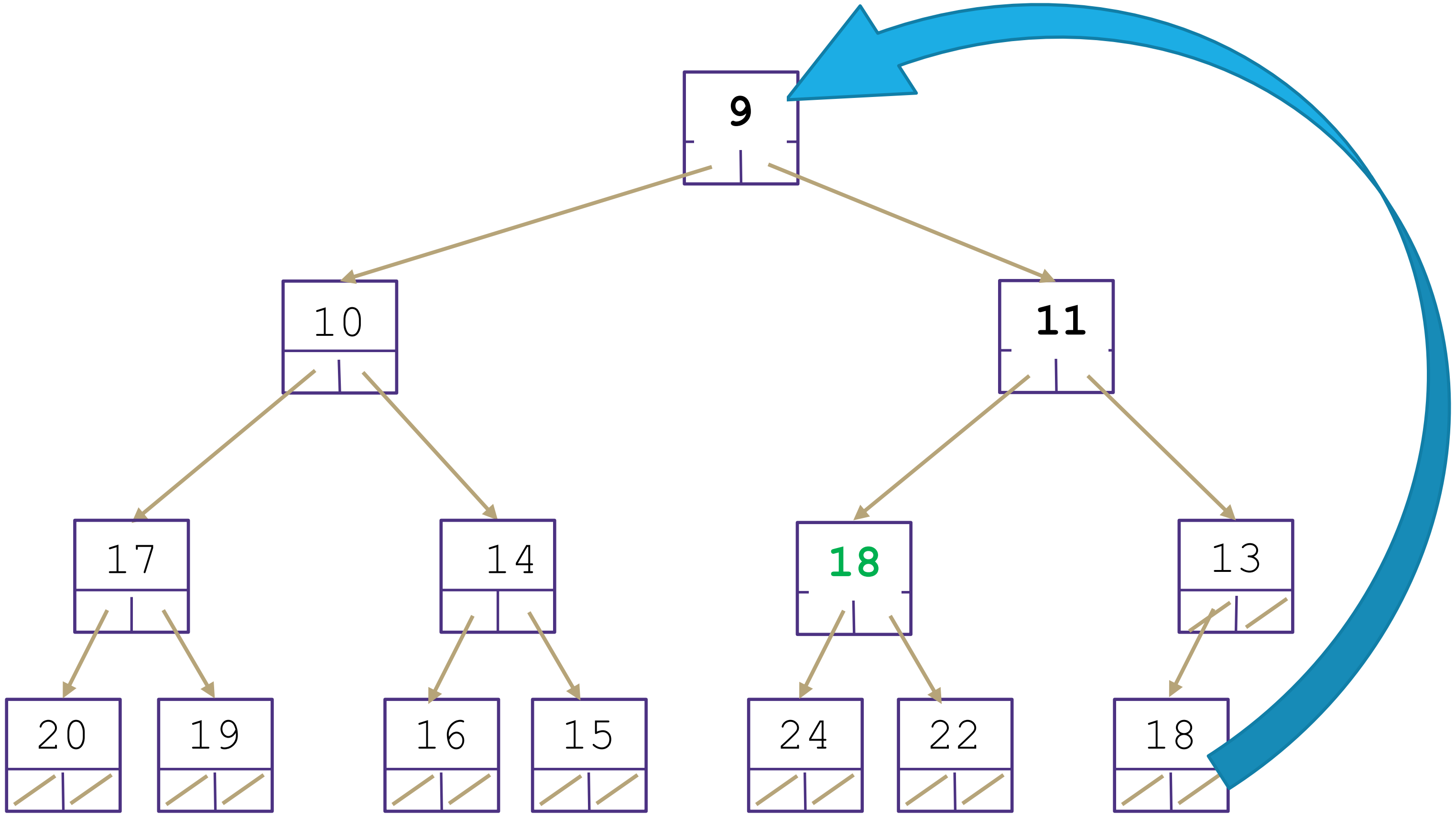
3.) percolateDown()

Recursively swap parent with smallest child



3 Minutes

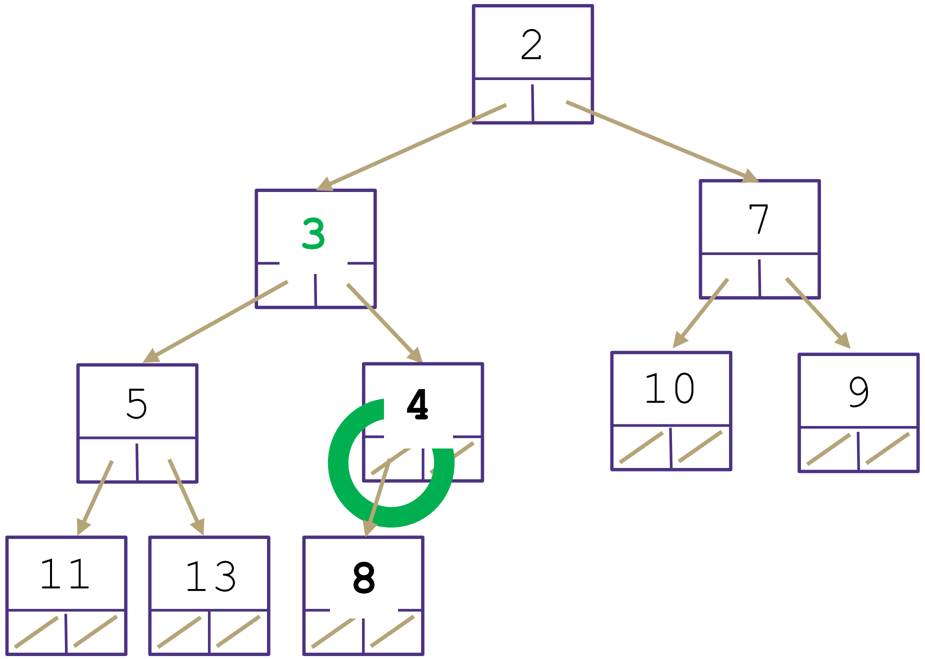
Practice: removeMin()



Implementing insert()

Algorithm:

- Insert a node to ensure no gaps
- Fix heap invariant
- percolate **UP**



3 Minutes

Practice: Building a minHeap

Construct a Min Binary Heap by inserting the following values in this order:

5, 10, 15, 20, 7, 2

Min Priority Queue ADT

state

Set of comparable values
- Ordered based on "priority"

behavior

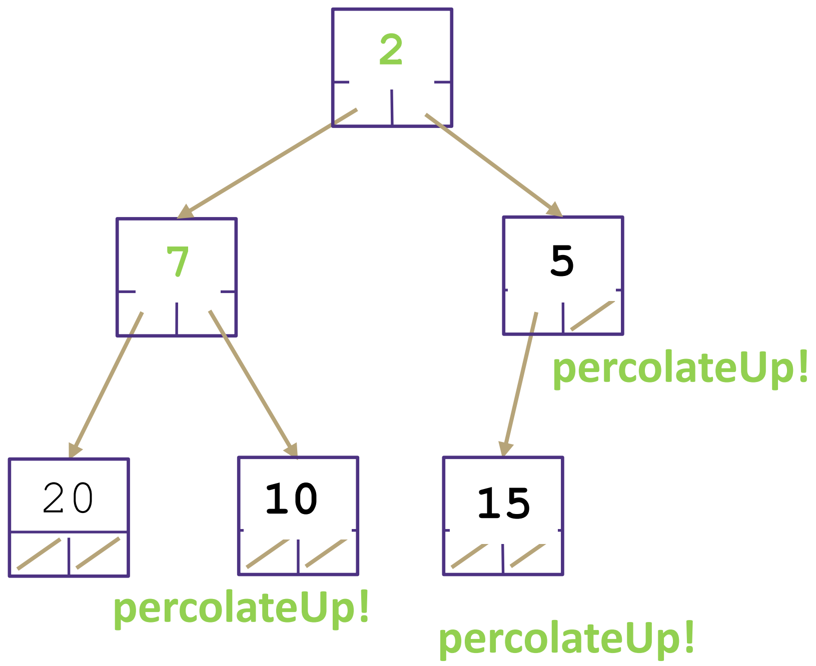
removeMin() – returns the element with the smallest priority, removes it from the collection

peekMin() – find, but do not remove the element with the smallest priority

insert(value) – add a new element to the collection

Min Binary Heap Invariants

1. **Binary Tree** – each node has at most 2 children
2. **Min Heap** – each node's children are larger than itself
3. **Level Complete** - new nodes are added from left to right completely filling each level before creating a new one



minHeap runtimes

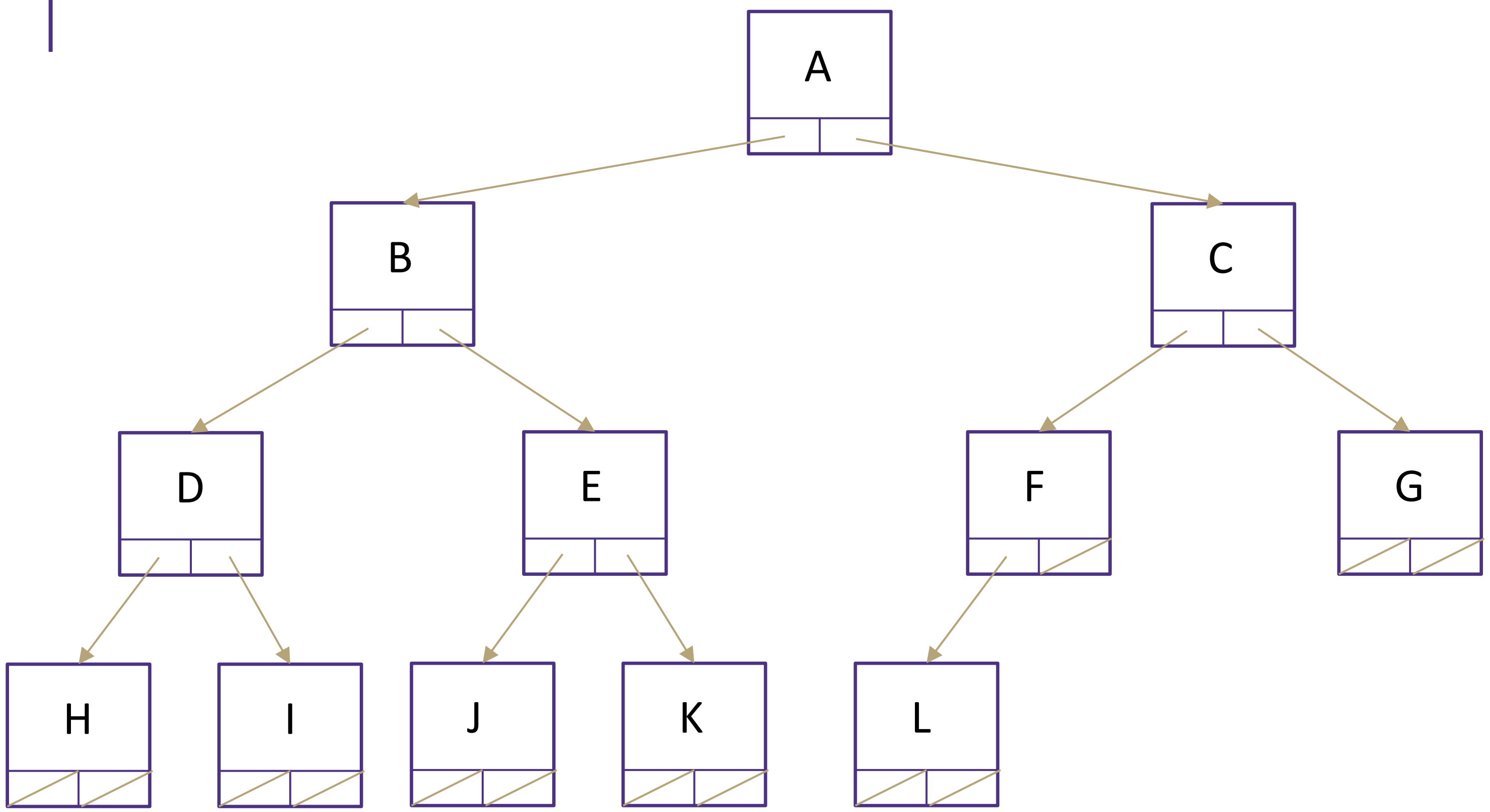
removeMin():

- Find and remove minimum node
- Find last node in tree and swap to top level
- Percolate down to fix heap invariant

insert():

- Insert new node into next available spot
- Percolate up to fix heap invariant

Implementing Heaps



Fill array in **level-order** from left to right

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

How do we find the minimum node?

$$peekMin() = arr[0]$$

How do we find the last node?

$$lastNode() = arr[size \; - 1]$$

How do we find the next open space?

$$openSpace() = arr[size]$$

How do we find a node's left child?

$$leftChild(i) = 2i + 1$$

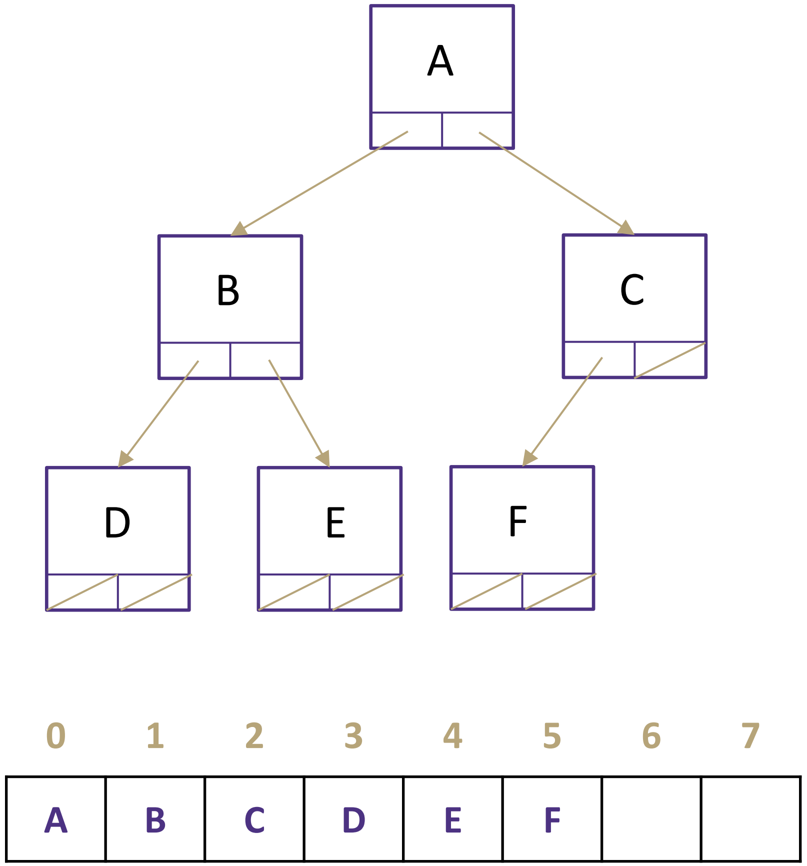
How do we find a node's right child?

$$rightChild(i) = 2i + 2$$

How do we find a node's parent?

$$parent(i) = \frac{(i - 1)}{2}$$

Heap Implementation Runtimes



`char peekMin()`

timeToFindMin

Tree $\Theta(1)$

Array $\Theta(1)$

`char removeMin()`

findLastNodeTime + removeRootTime + numSwaps * swapTime

Tree n + 1 + log(n) * 1 $\Theta(n)$

Array 1 + 1 + log(n) * 1 $\Theta(\log(n))$

`void insert(char)`

findNextSpace + addValue + numSwaps * swapTime

Tree n + 1 + log(n) * 1 $\Theta(n)$

Array 1 + 1 + log(n) * 1 $\Theta(\log(n))$

Building a Heap

Insert has a runtime of $\Theta(\log(n))$

If we want to insert a n items…

Building a tree takes $O(n\log(n))$

- Add a node, fix the heap, add a node, fix the heap

Can we do better?

- Add all nodes, fix heap all at once!

Cleaver building a heap – Floyd's Method

Facts of binary trees

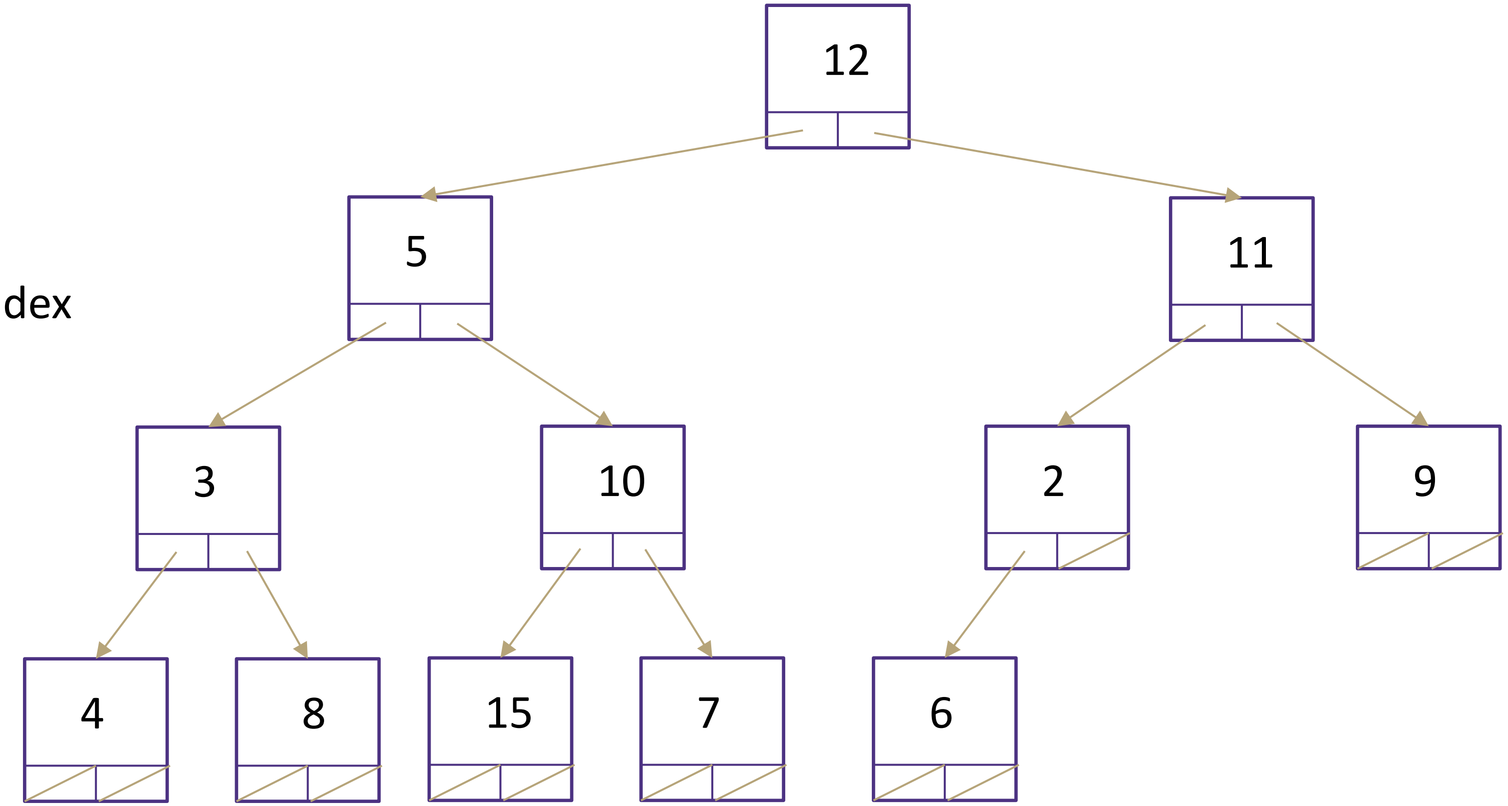
- Increasing the height by one level doubles the number of possible nodes
- A complete binary tree has half of its nodes in the leaves
- A new piece of data is much more likely to have to percolate down to the bottom than be the smallest element in heap

1. Dump all the new values into the bottom of the tree
   - Back of the array
2. Traverse the tree from bottom to top
   - Reverse order in the array
3. Percolate Down each level moving towards overall root

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index

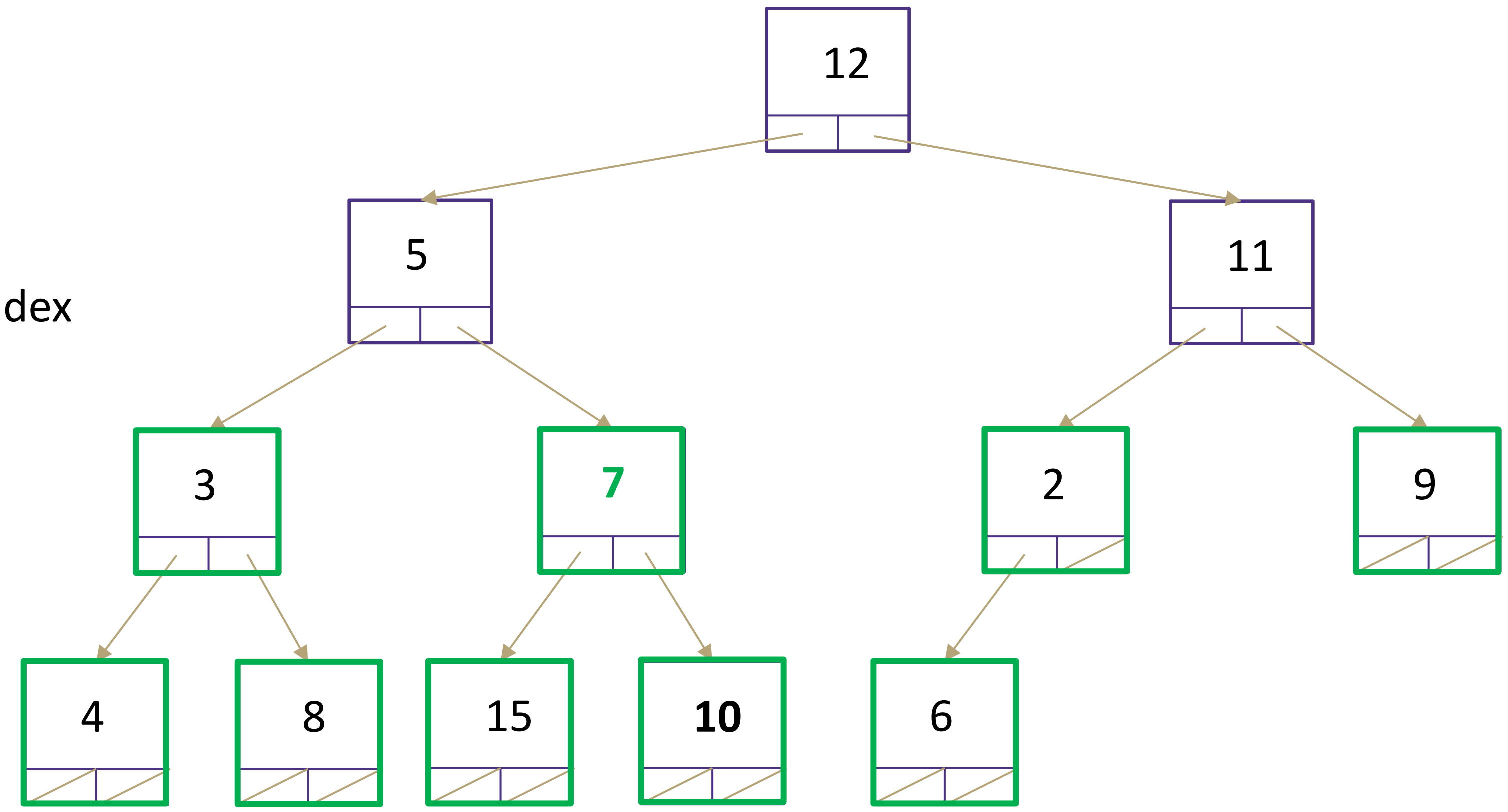


0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	10	2	9	4	8	15	7	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3



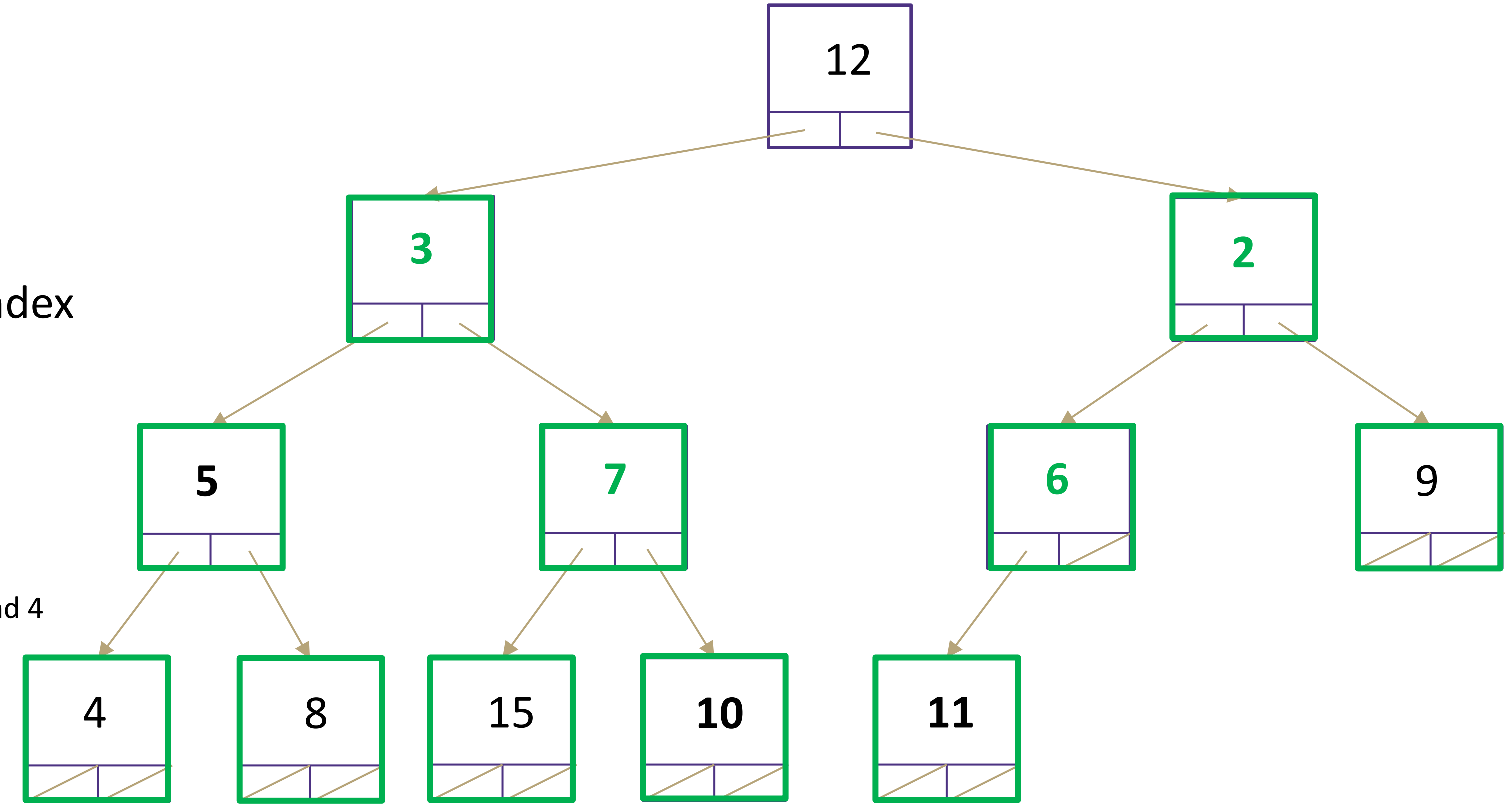
0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	10	2	9	4	8	15	7	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2

keep percolating down
like normal here and swap 5 and 4

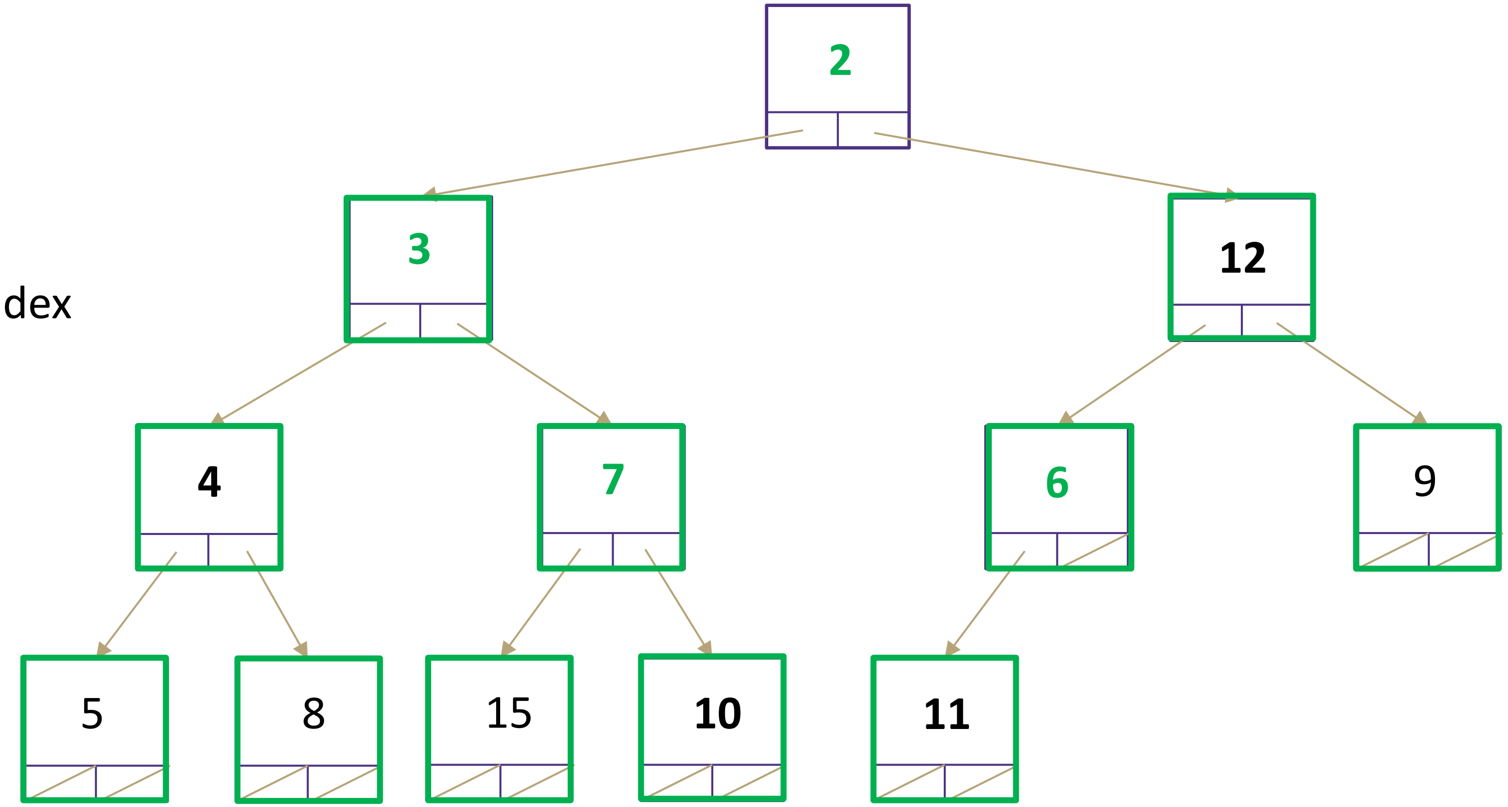


0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	7	2	9	4	8	15	10	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2
    4. percolateDown level 1

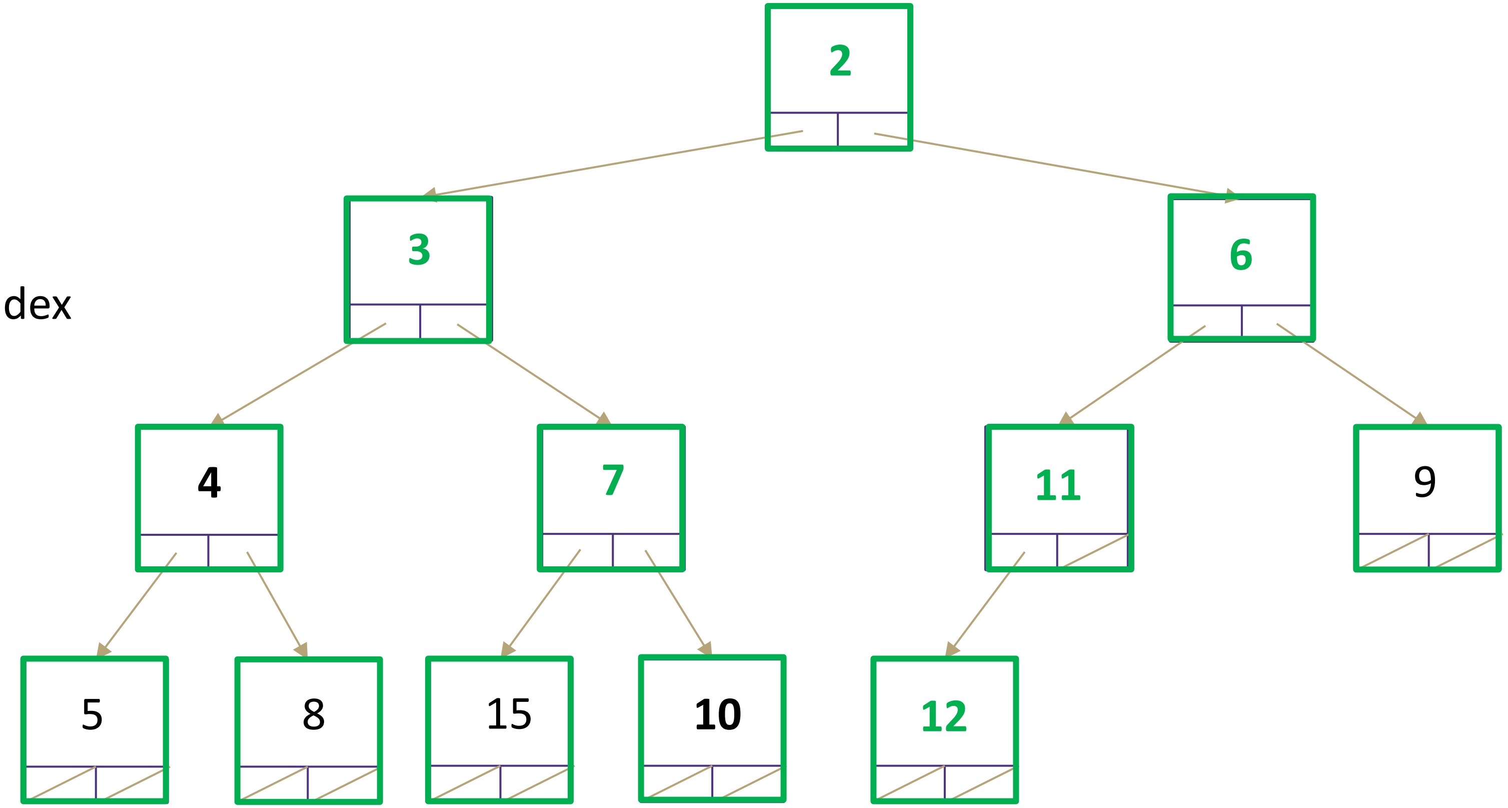


0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	3	2	4	7	6	9	5	8	15	10	11		

Floyd’s buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2
    4. percolateDown level 1



0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	3	6	4	7	11	9	5	8	15	10	12		

Floyd's Heap Runtime

We step through each node – n

We call percolateDown() on each n – log n

thus it's O(nlogn)

… let's look closer…

Are we sure percolateDown() runs log n each time?

- Half the nodes of the tree are leaves
  - Leaves run percolate down in constant time
- ¼ the nodes have at most 1 level to travel
- 1/8 the nodes have at most 2 levels to travel
- etc…

work(n) ≈ n/2 * 1 + n/4 * 2 + n/8 * 3 + …

Closed form Floyd's buildHeap

work(n) ≈ $\frac{n}{2} * 1 + \frac{n}{4} * 2 + \frac{n}{8} * 3 + \ldots$

factor out n

work(n) ≈ $n(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \ldots)$ find a pattern -> powers of 2 work(n) ≈ $n(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \ldots)$ Summation!

$$work(n) \approx n \sum_{i=1}^{?} \frac{i}{2^i}$$

? = how many levels = height of tree = log(n)

Infinite geometric series

$$work(n) \approx n \sum_{i=1}^{\log n} \frac{i}{2^i}$$

$$if -1 < x < 1\ then \sum_{i=0}^{\infty} x^i = \frac{1}{1-x} = x$$

$$work(n) \approx n \sum_{i=1}^{\log n} \frac{i}{2^i} \leq n \sum_{i=0}^{\infty} \frac{i}{2^i} = n * 2$$

Floyd's buildHeap runs in O(n) time!