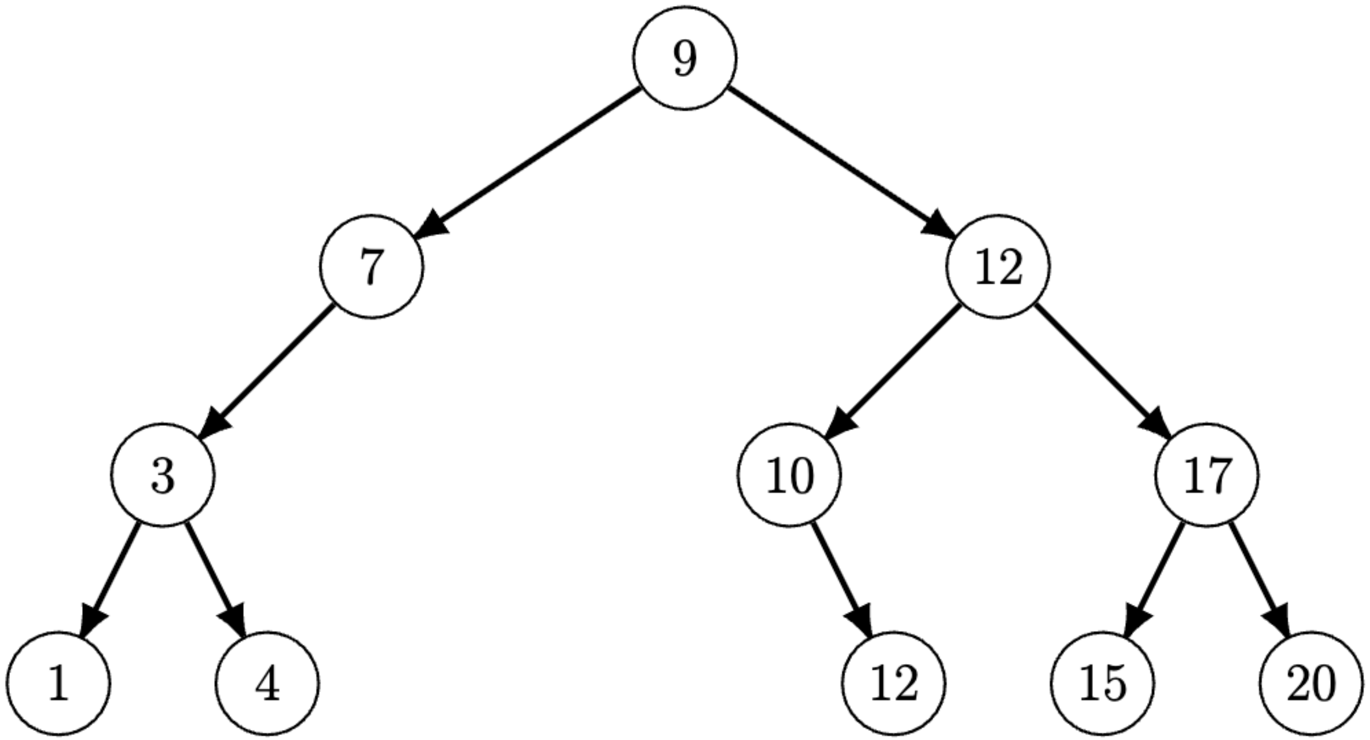
Complexity of Binary Search Tree `find()`

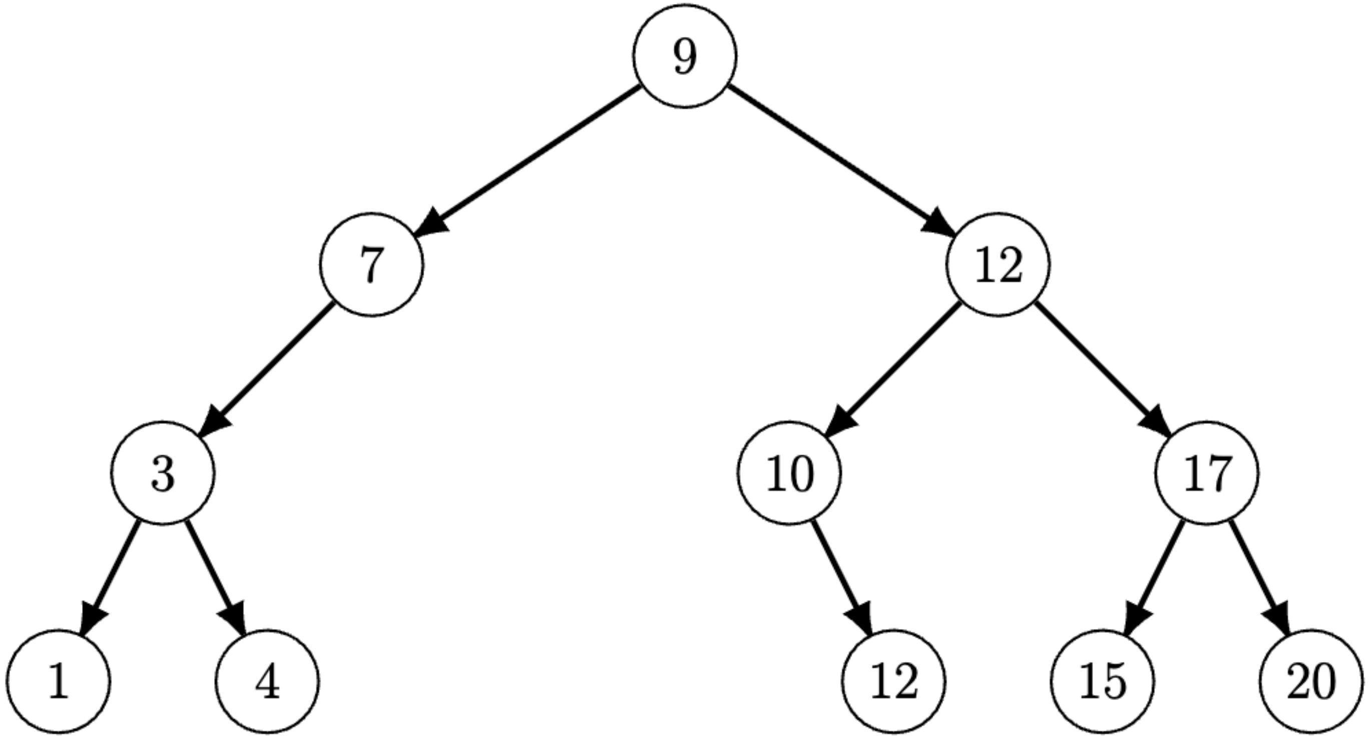
Recall the Binary Search Tree find method from last lecture.
What is the worst-case time complexity for `find()`?
Write your answer in terms of size, S, of the tree.



Complexity of Binary Search Tree `find()`

Recall the Binary Search Tree find method from last lecture.
What is the worst-case time complexity for `find()`?
Write your answer in terms of size, S, of the tree.

A) O(1)
B) O(log(S))
C) O(S)
D) $O(S^2)$

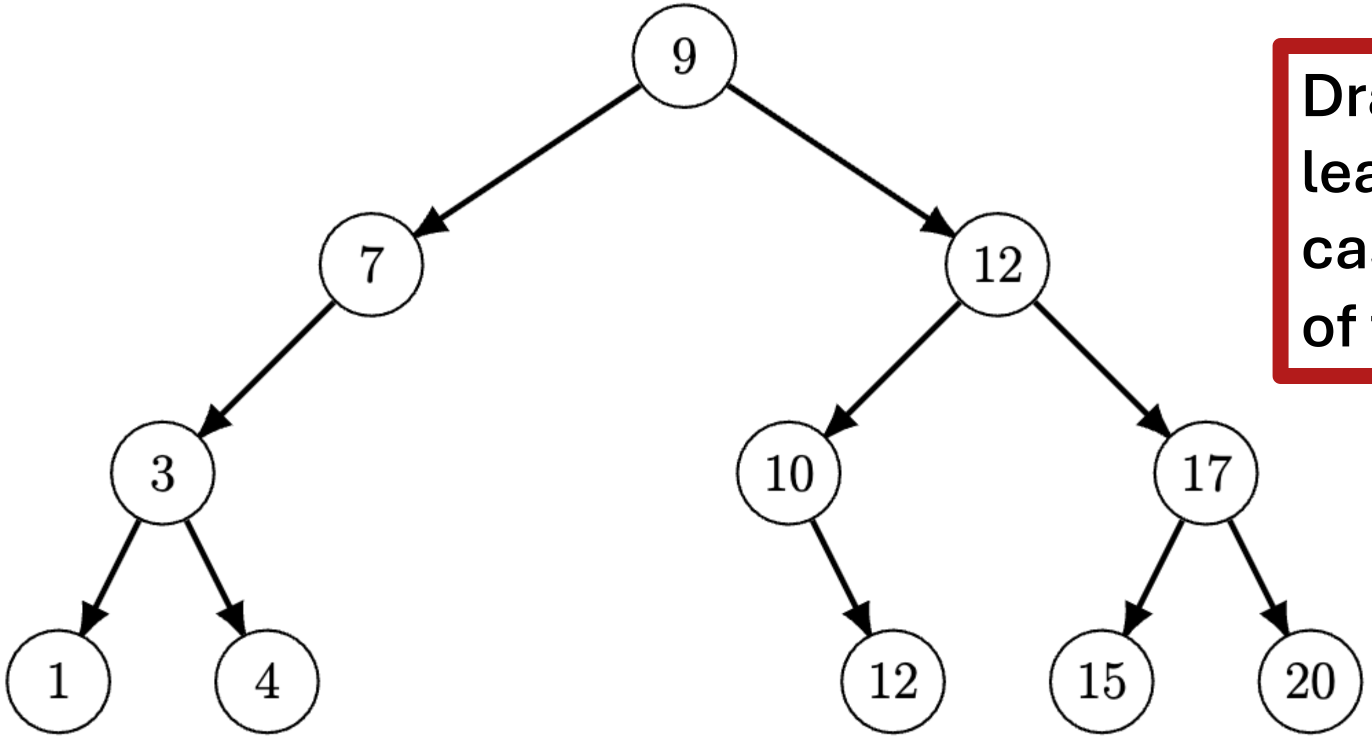


PollEv.com/leahp
text `leahp` to `22333`

Complexity of Binary Search Tree `find()`

Recall the Binary Search Tree find method from last lecture.
What is the worst-case time complexity for `find()`?
Write your answer in terms of size, S, of the tree.

A) O(1)
B) O(log(S))
C) O(S)
D) O(S^2)

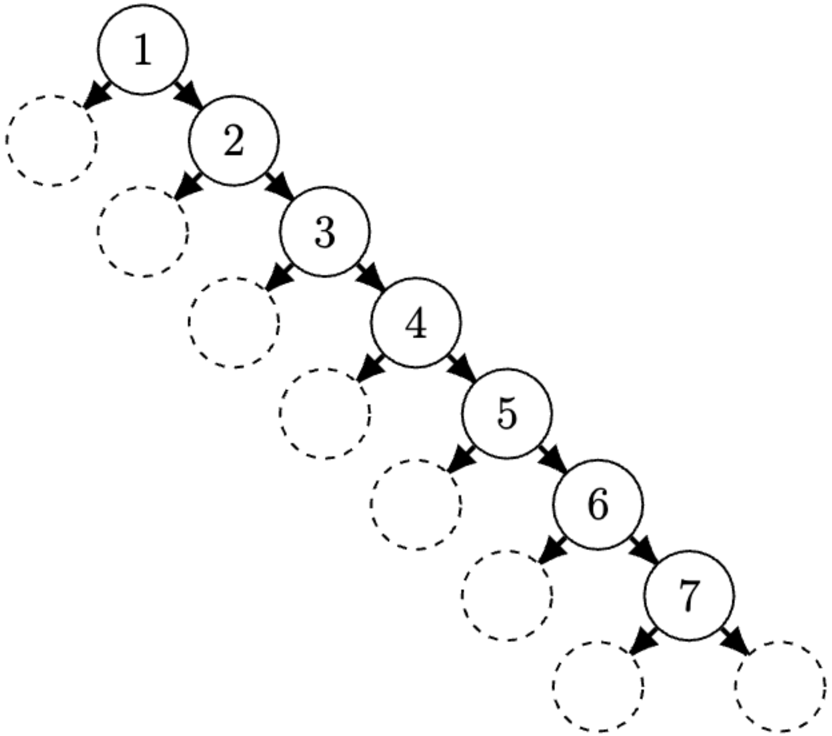


Draw a tree that leads to worst-case performance of find().

PollEv.com/leahp
text **leahp** to **22333**

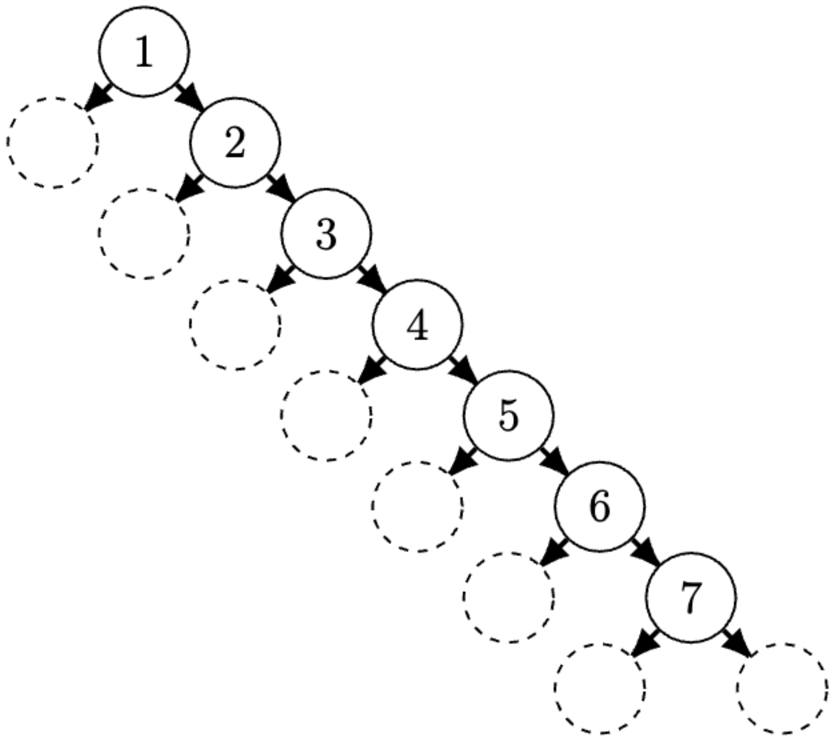
Complexity of Binary Search Tree `find()`

Answer: An unbalanced tree can lead to O(S) performance



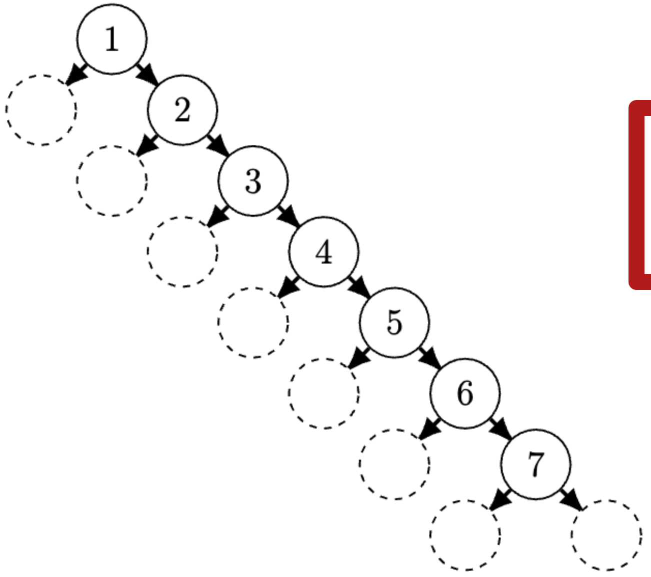
Complexity of Binary Search Tree `find()`

Answer: An unbalanced tree can lead to O(S) performance



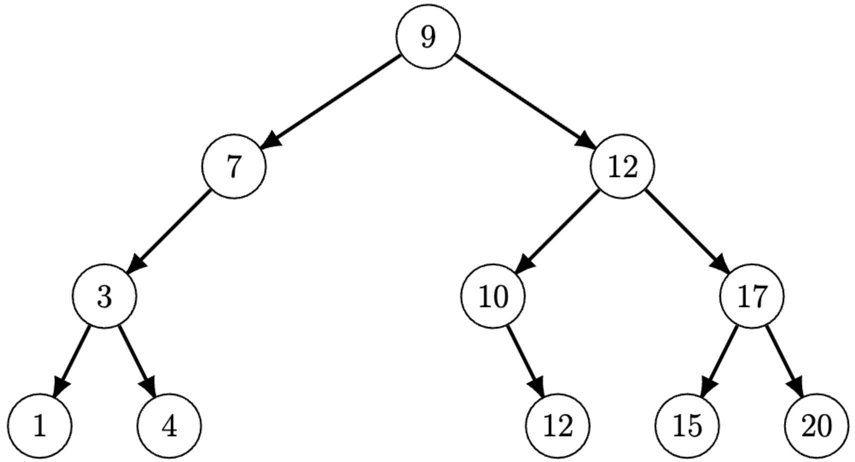
How can we keep BSTs balanced?

Balanced Binary Search Tree



How can we keep BSTs balanced?

- Perform rotations with every modification
- each rotation is O(1)
- number of rotations is up to height H (a rotation at every level)
- Balanced BST height is always O(log size)
- Balanced BST operations are at worst O(log size)



Lecture 18: Heaps and Priority Queues

CS 2110, Matt Eichhorn and Leah Perlmutter

October 28, 2025

Announcements

- Public Health Announcement
  - One of the best ways to care for each other is to keep each other healthy!
  - Stay home when you are very sick
  - If you must come to class when coughing or sneezing, wear a mask and practice social distancing
  - If someone sitting near you seems sick, you can get up and move farther away

Announcements

- Prelim 2 coming up Thursday 11/6
  - Conflict survey due Thurs 10/30 by 5pm --------→
  - Make sure you are all caught up in your learning
  - Ed post and practice exam coming Thursday

Today's Learning Outcomes

Heaps and Priority Queues

1. Write recursive methods on general and binary trees.
2. Describe the invariants of a heap and determine whether they are satisfied by a given binary tree.
3. Translate between the tree and array representations of a heap.
4. Implement operations on a heap and determine their time/space complexities.
5. Use a heap to implement a priority queue and analyze its performance.

Priority Queues

Common pattern: give me the “next” thing

Different choices for “next”:

- Queue (FIFO): who has been *waiting the longest*?
- Stack (LIFO): who was *added most recently*?
- **Priority queue: who is *most important*?**

Applications:

- Shortest paths
- Task deadlines (what is due soonest, regardless of when it was assigned)
- Emergency room triage

Priority Queue Operations

What are Queue Operations?

Does being a *priority* queue change anything?

- `peek()`: Return the most important element
- `remove()`: Remove and return the most important element
- `add()`: Add a new element

Want these operations to be *fast* (low time complexity)

- Ideally, `peek()` should be $O(1)$ (always know what the best value is)
- `remove()` and `add()` must preserve whatever invariant makes `peek()` fast without being slow themselves

Priority Queue: Possible Representations

Consider implementing a priority queue with the following data structures. What would the **worst-case time complexity** of each operation be? (let *N* denote the queue's size)

Data structure	peek()	remove()	add()
Unsorted linked list	O(N)	O(N)	O(1)
Sorted array	O(1)	O(1)	O(N)
Balanced BST	O(log N)	O(log N)	O(log N)

Do we need/want to keep all elements sorted?

Often, processing one element (remove) will cause many new elements to be added to the queue (add).

- E.g. exploring a cave: take the right fork, but at the end of that tunnel, three new tunnels open up

Keeping all these TODOs sorted is wasteful – we'll keep having to move things around when new tasks come in, and all we care about is which *one* is next

Strategy: relax invariant

✨ ✨ ✨ Max Heap ✨ ✨ ✨

Side note: Do not confuse the heap data structure with the memory heap

Side side note: Stack memory is organized as a stack, but heap memory is not organized as a heap.

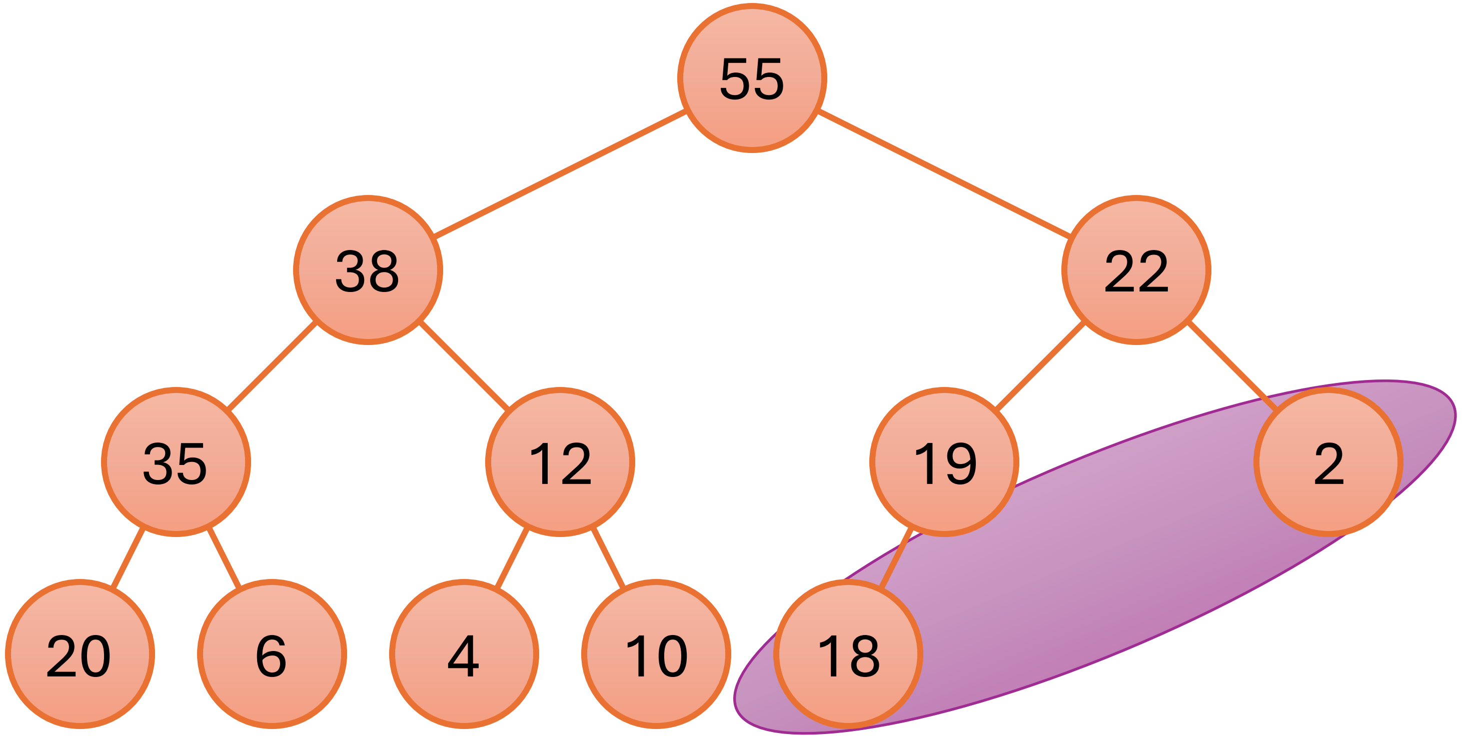
A Max Heap…

Is a **binary tree** (*not* BST) satisfying two additional properties:

1. **Heap-ordered** (*order invariant*). Every node is “more important” than its children
   - Min-heap: every node is <= its children (smallest on top)
     “earliest deadline,” “shortest distance”
   - Max-heap: every node is >= its children (biggest on top)
     “largest reward”

Heap-order (max-heap)

Every element is <= its parent



Note: Bigger elements
can be deeper in the tree!

A Heap…

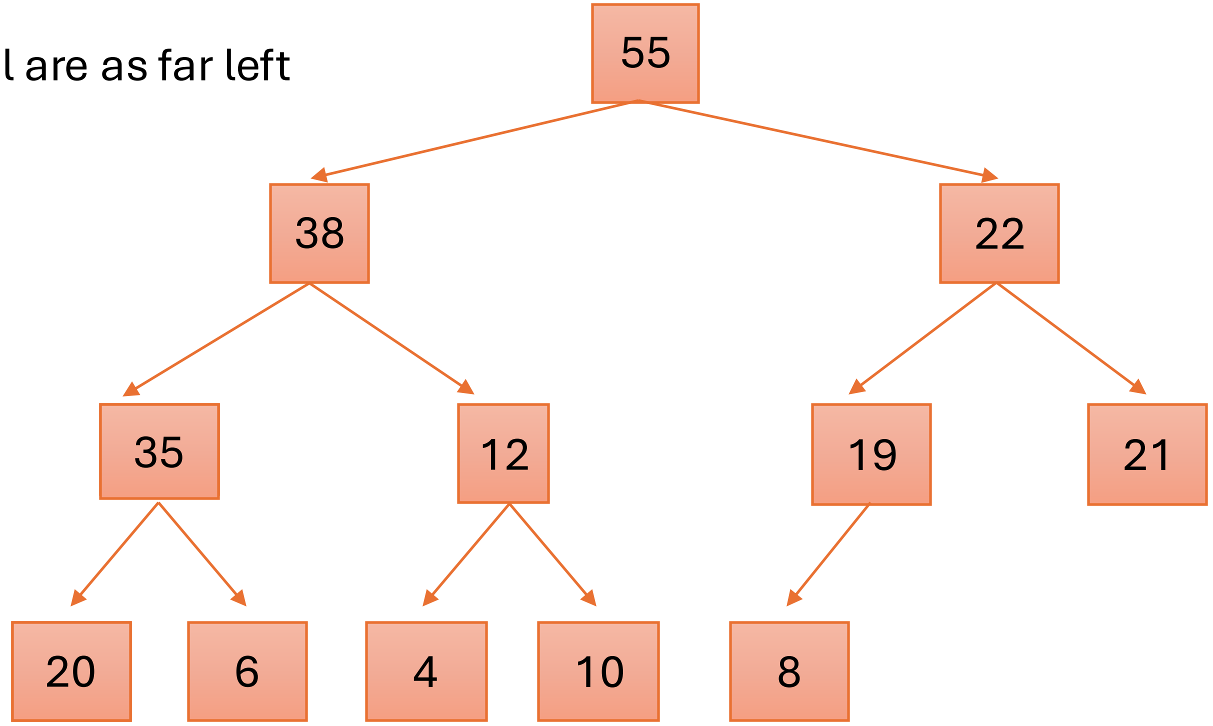
Is a **binary tree** (*not* BST) satisfying two additional properties:

1. **Heap-ordered** (*order invariant*). Every node is “more important” than its children
2. **Completeness** (*shape invariant*). Every level of the tree (except last) is completely filled, and on last level nodes are as far left as possible.
   - We call this shape a *nearly complete* binary tree

Completeness

Every level (except the last) is completely filled.

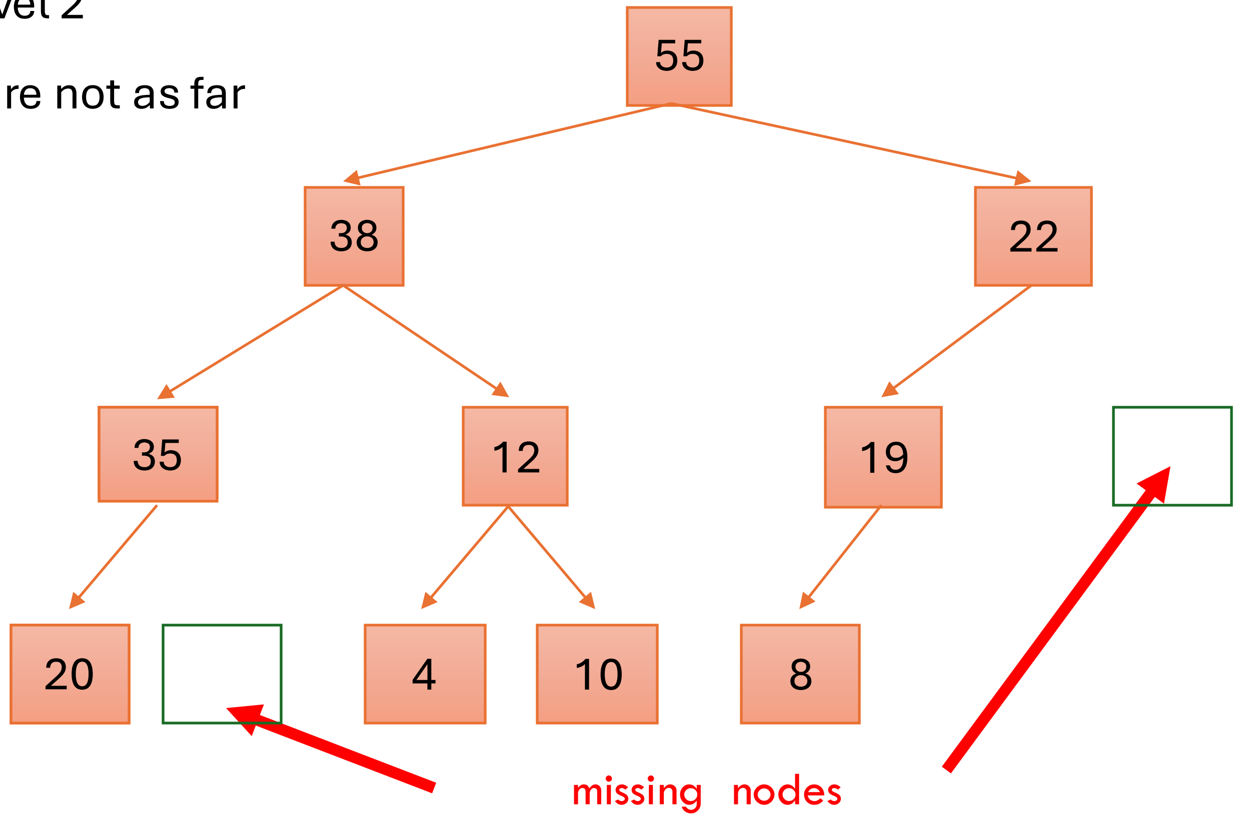
Nodes on bottom level are as far left as possible.



Completeness

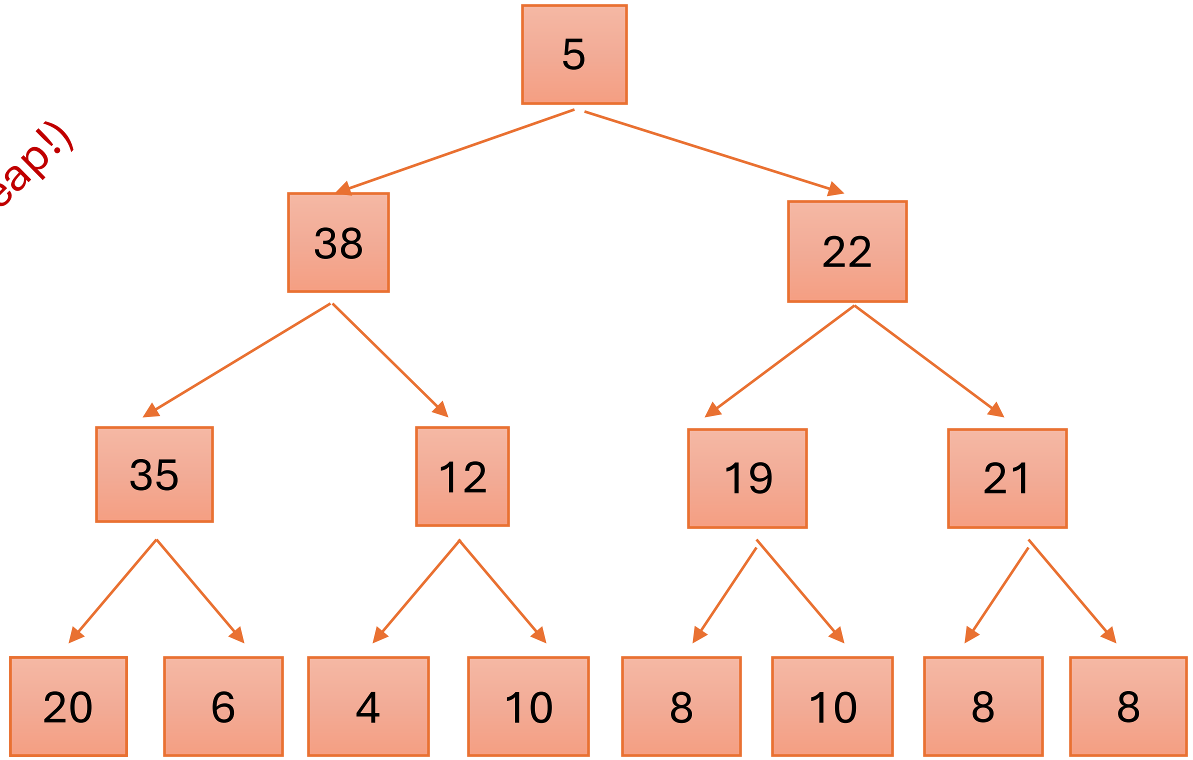
Not a heap because:

- missing a node on level 2
- bottom level nodes are not as far left as possible



Height of **complete** binary tree
Limiting case: a “perfect” tree

(Not a heap!)



Levels	total number of nodes
1	2^1 – 1 = 1
2	2^2 – 1 = 3
3	2^3 – 1 = 7
4	2^4 – 1 = 15

Perfect binary tree with 2^k-1 nodes has k levels

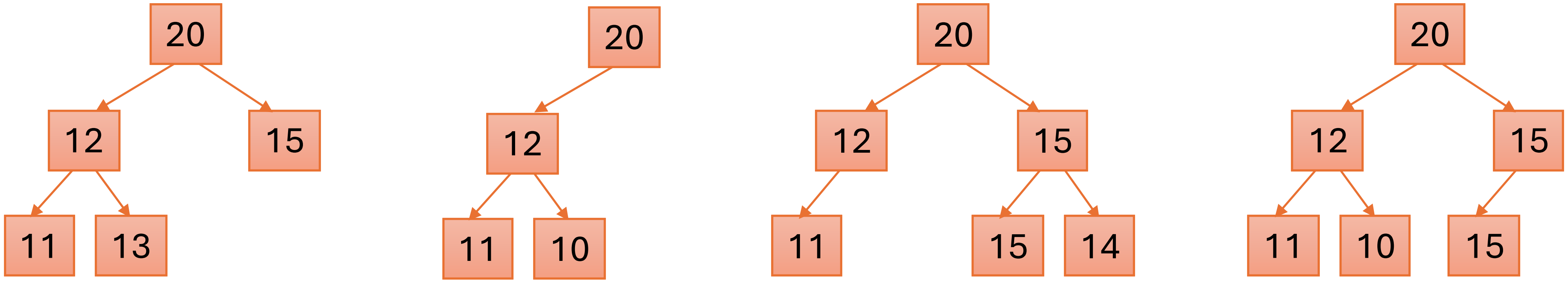
Add one more node: 2^k nodes has k+1 levels

Complete binary tree with n nodes has $\lceil \log(n+1) \rceil \subset$ O(log n) levels.
Takeaway: **height is O(log N)** (always balanced)

Poll 1

PollEv.com/leahp
text **leahp** to **22333**

Which of the following are valid max heaps?



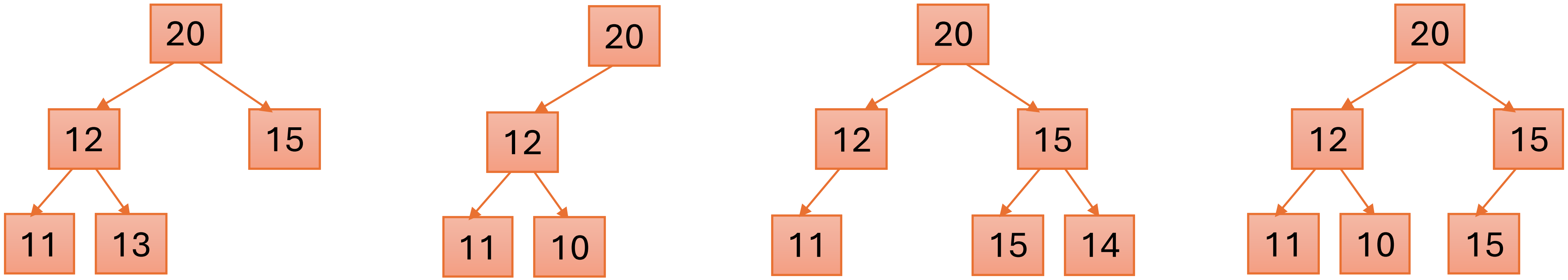
(a) (b) (c) (d)

(e) none of them

Poll 1

PollEv.com/leahp
text **leahp** to **22333**

Which of the following are valid max heaps?



(a) (b) (c) (d)

a – does not satisfy the order property because 13 is a child of 12
b – does not satisfy the completeness property – root is missing right child
c – does not satisfy the completeness property -- 12 is missing right child
d – satisfies both order and completeness properties [correct answer]

Back to priority queues

Max Heap can represent a Priority Queue

- Efficiency we will achieve (storing *N* elements):
  - add(): O(log *N*)
  - remove(): O(log *N*)
  - peek(): O(1)
- No linear-time operations: better than lists
- peek() is constant time: better (and simpler) than balanced trees

Max Heap Implementation

Naive Implementation: a tree with nodes

```
public class HeapNode<E> {
  private E value;
  private HeapNode<E> left;
  private HeapNode<E> right;
  ...
}
```

But since tree is complete, even more space-efficient implementation is possible…

Array implementation

```java
public class Heap<E> {
  /** represents a complete binary tree in `heap[0..size)` */
  private E[] heap;
  private int size;
  ...
}
```

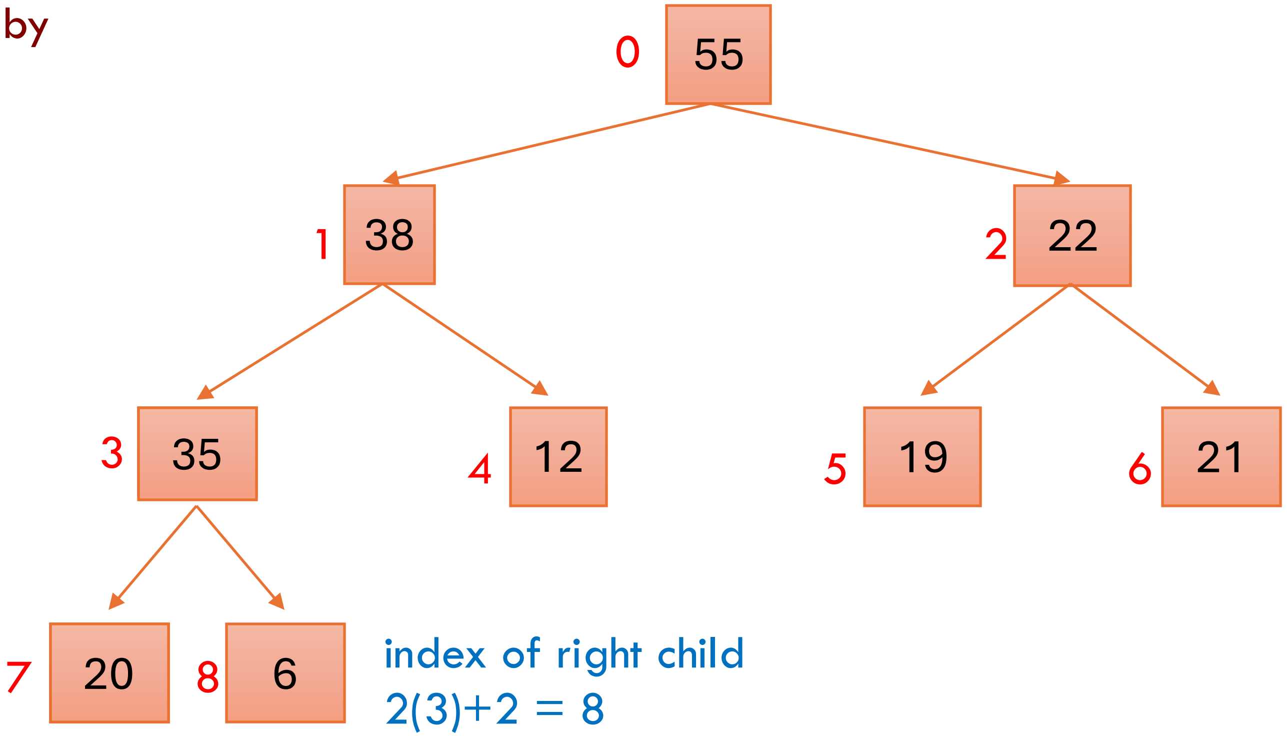
Indexing tree nodes

Number node starting at root row by row, left to right

Same order as
level-order traversal

index of parent
3 = (7-1)/2
3 = (8-1)/2 -.5

k=3



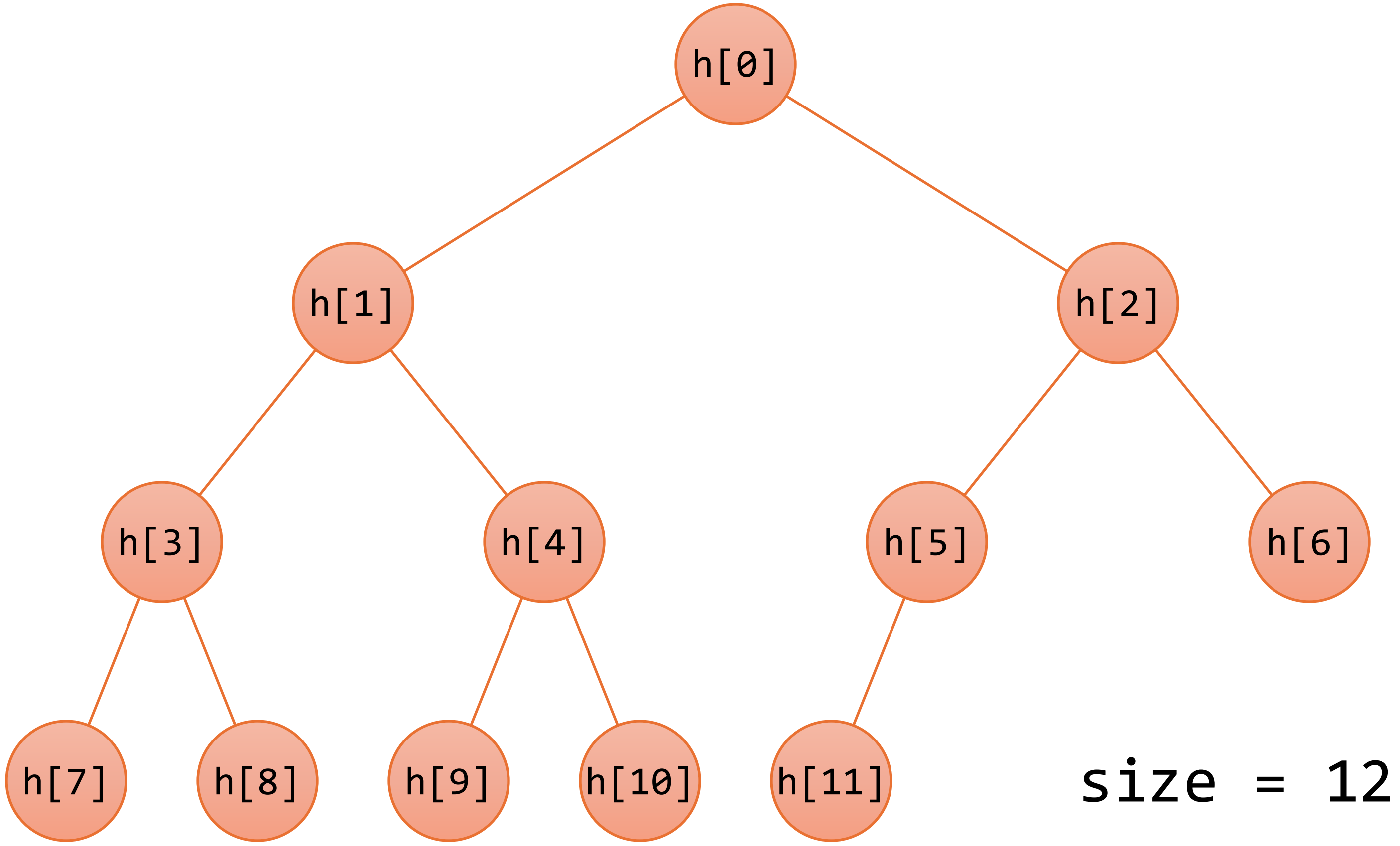
index of left child
2(3)+1 = 7

index of right child
2(3)+2 = 8

Children of node **k** are nodes **2k+1** and **2k+2**

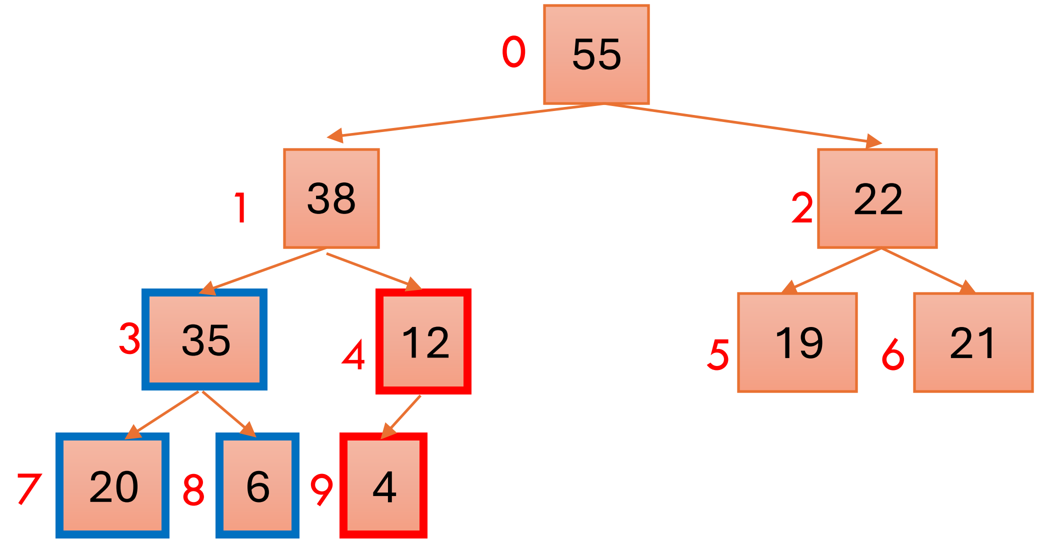
Parent of node **k** is node **(k-1)/2**
(integer division with flooring)

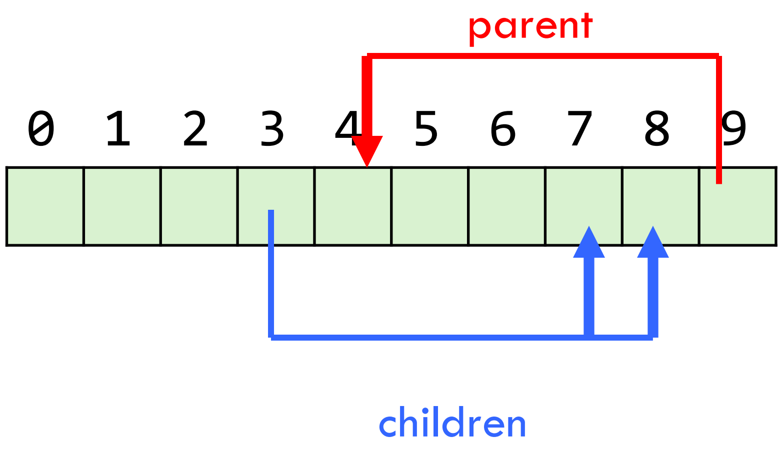
Tree nodes as array elements

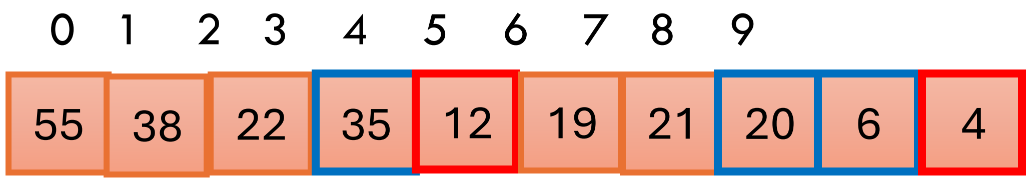


Represent tree with array

- Store node number *i* in index *i* of array b
- Children of b[*k*] are b[2*k* + 1] and b[2*k* + 2]
- Parent of b[*k*] is b[(*k*-1)/2]

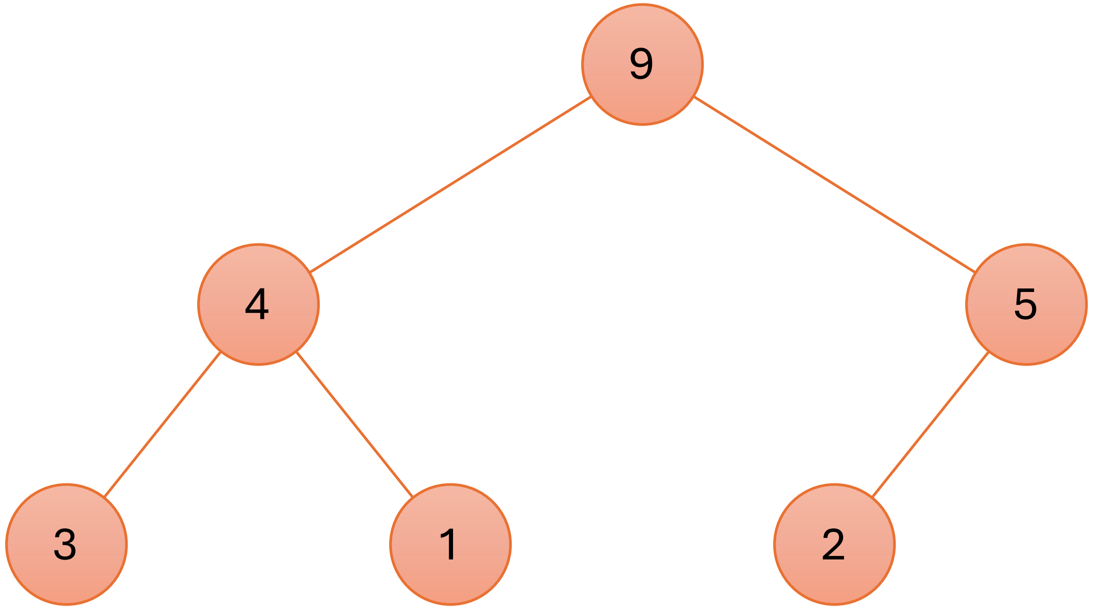






Exercise: map tree to array

What is the array representation of this tree?



A. {1, 2, 3, 4, 5, 9}
B. {3, 1, 2, 4, 5, 9}
C. {9, 4, 3, 1, 5, 2}
D. {9, 4, 5, 3, 1, 2}

PollEv.com/leahp
text **leahp** to **22333**

Demo code

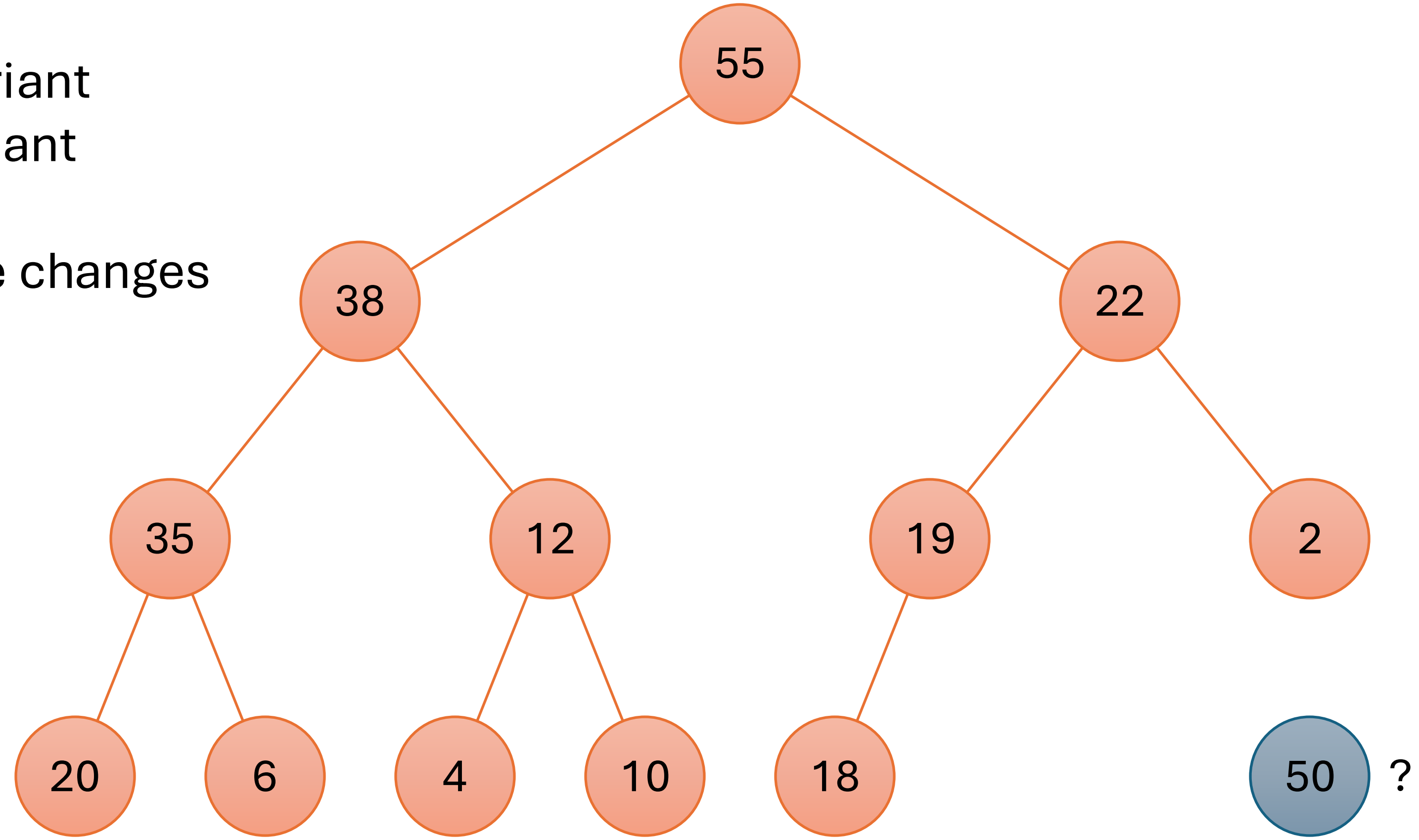
Max Heap Algorithms

Exercise: Adding an element

Must preserve:

1. Shape invariant
2. Order invariant

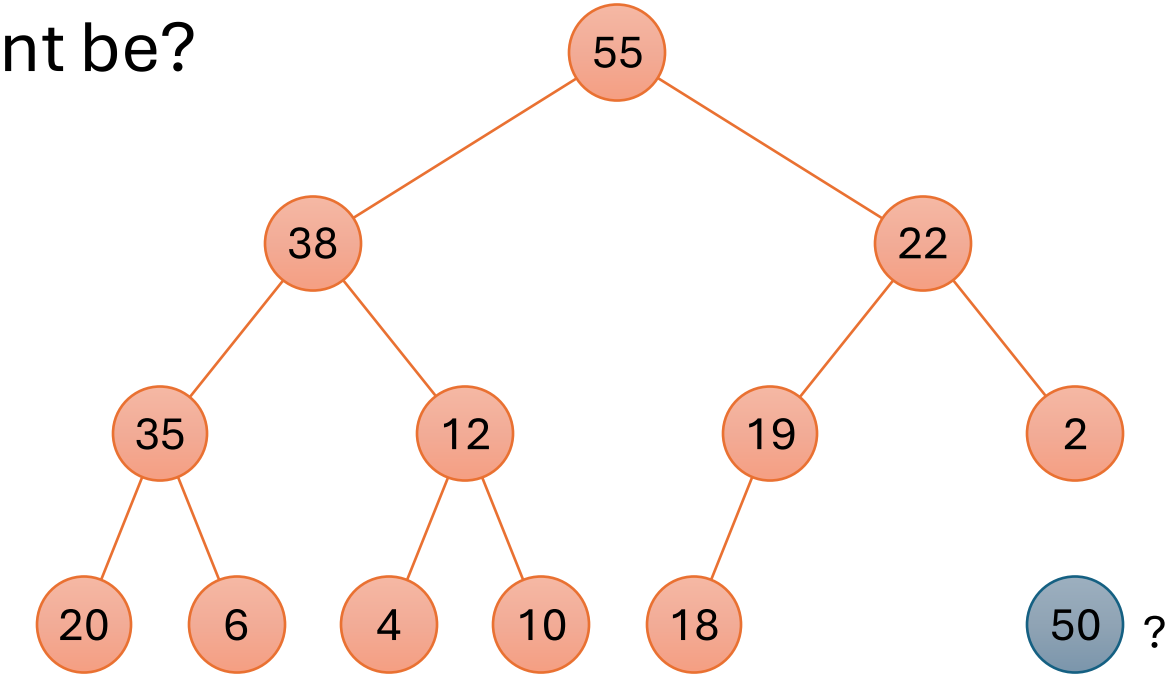
Goal: minimize changes



Step 1: Maintain shape invariant

What should 50's parent be?

A. 2
B. 19
C. 22
D. 38
E. Nothing

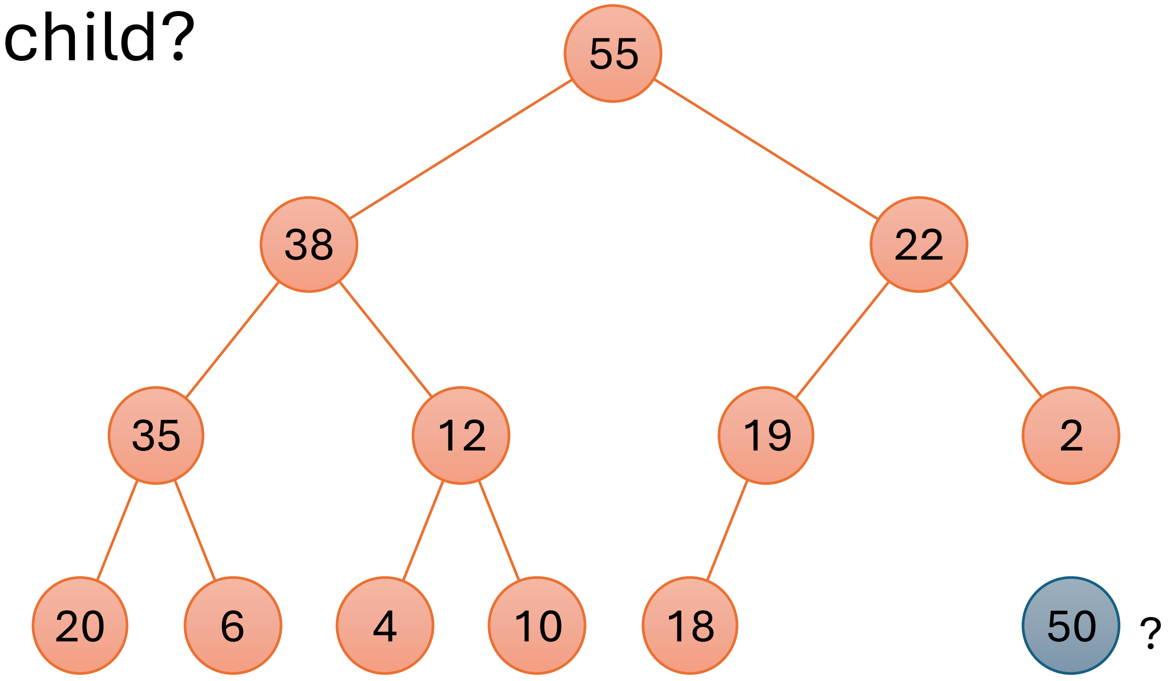


Step 2: Restore order invariant

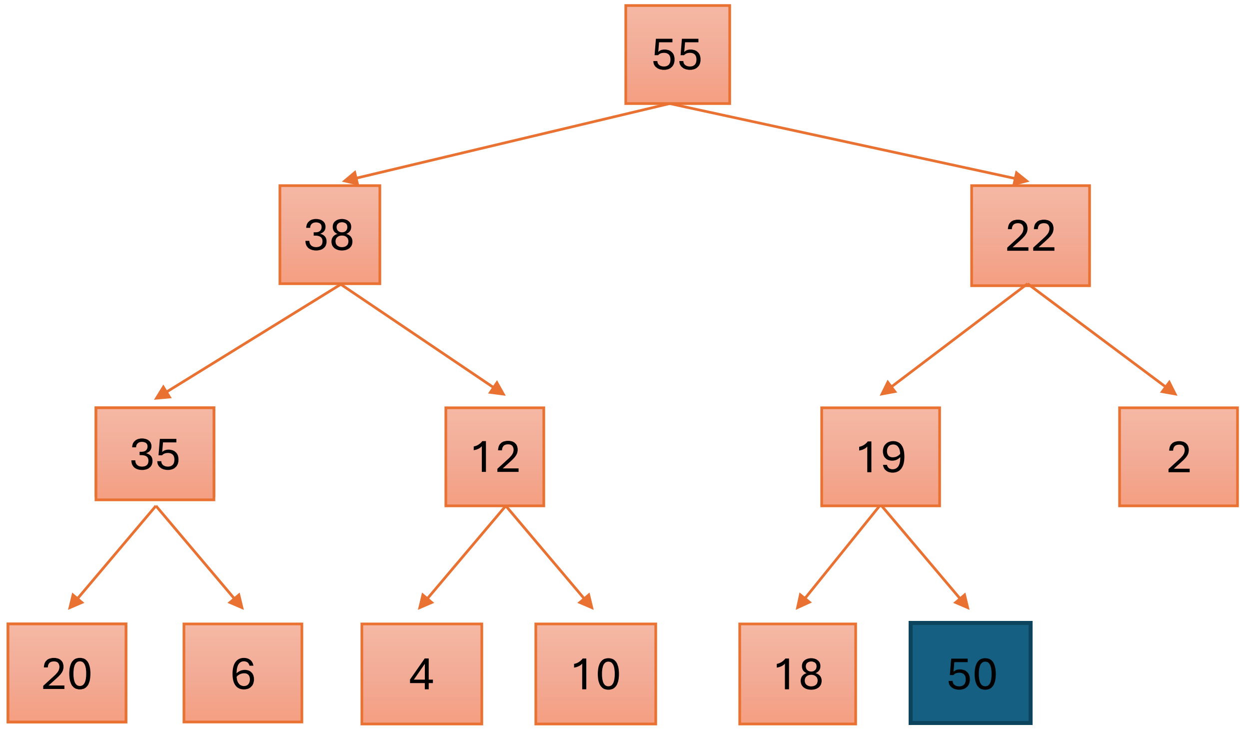
Swap with parent until satisfied.

When done, what is 50's left child?

A. 2
B. 19
C. 22
D. 38
E. Nothing



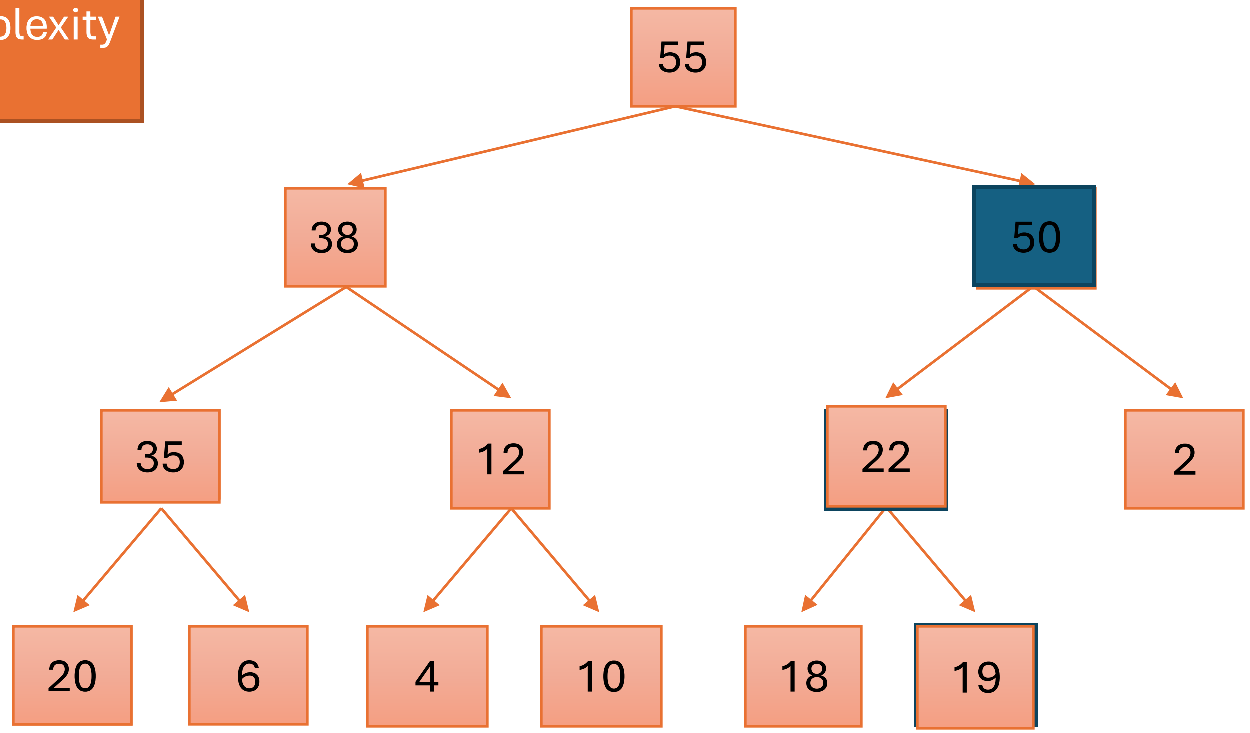
Heap: add(e)



1. Put in the new element in a new node (leftmost empty leaf)

Heap: add(e)

Time complexity
O(log size)

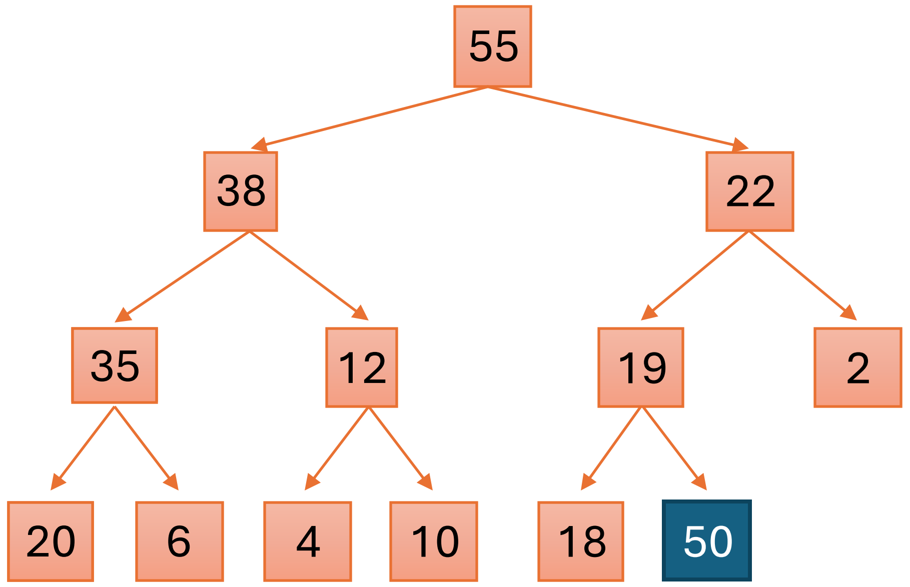


1. Put in the new element in a new node (leftmost empty leaf)
2. Bubble new element up while greater than parent

Does add() preserve the order invariant?

Let *x* denote the new node.

- Algorithm invariant
  - All nodes except *x* are <= their ancestors
- Algorithm body
  - Case 1: If x <= parent, then it is also <= all ancestors. Order invariant is satisfied everywhere – done!
  - Case 2: If x > parent, swap with parent

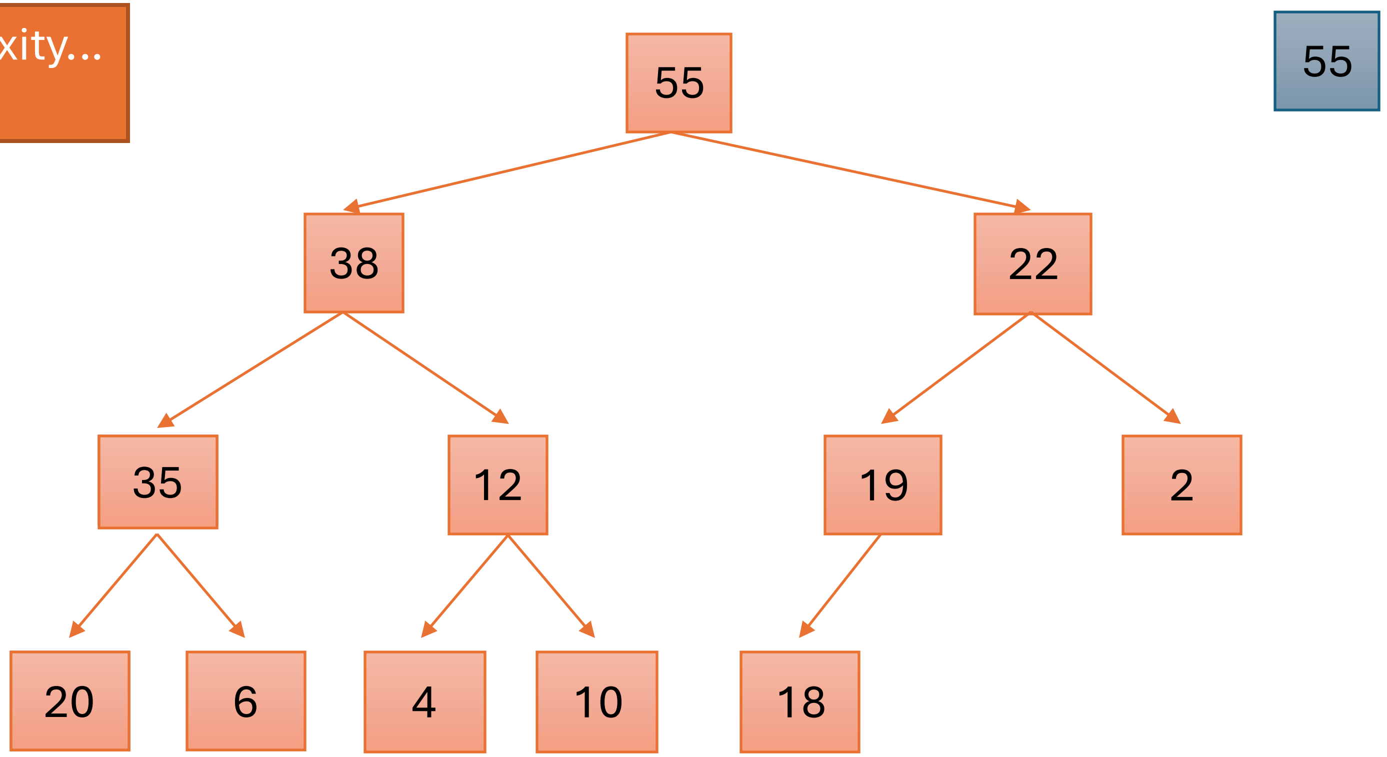


Does case 2 preserve the invariant?

- x's children must be <= all of their ancestors, so they are <= x's parent. Making x's parent their parent is ok
- x's sibling was <= x's parent, so making x its new parent is ok (since x > parent)

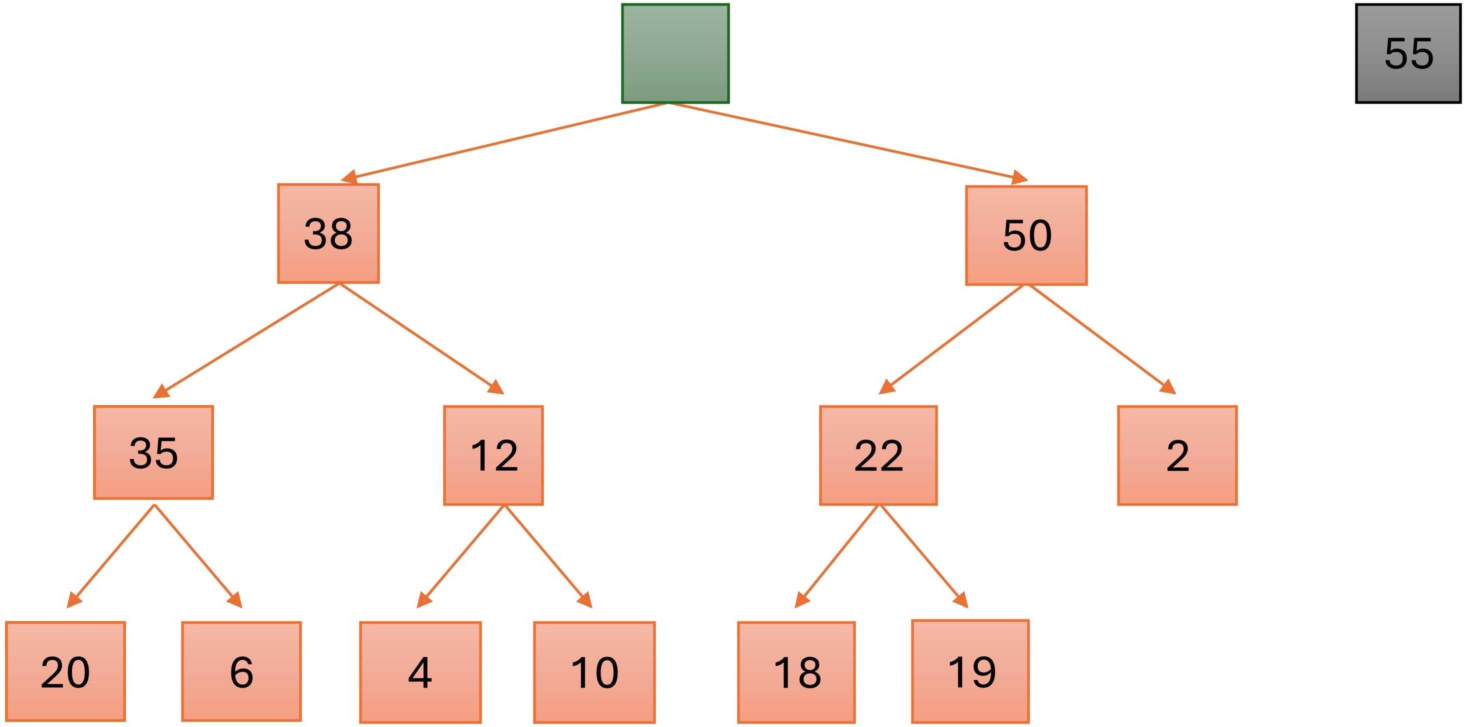
Heap: peek( )

Time complexity...
is O(1)



1. Return root value

Heap: `remove()`

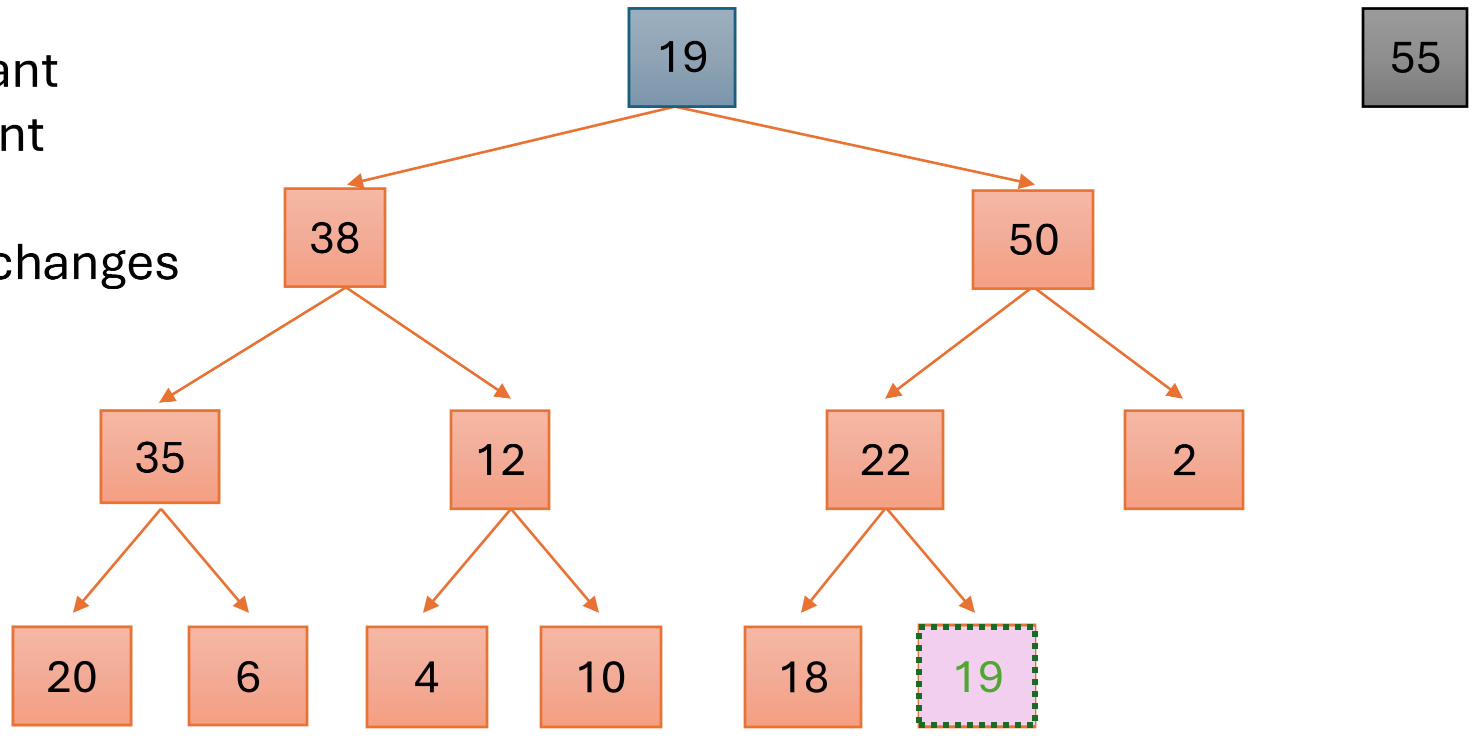


1. Save root element in a local variable

Heap: `remove()`

Must preserve:
1. Shape invariant
2. Order invariant

Goal: minimize changes



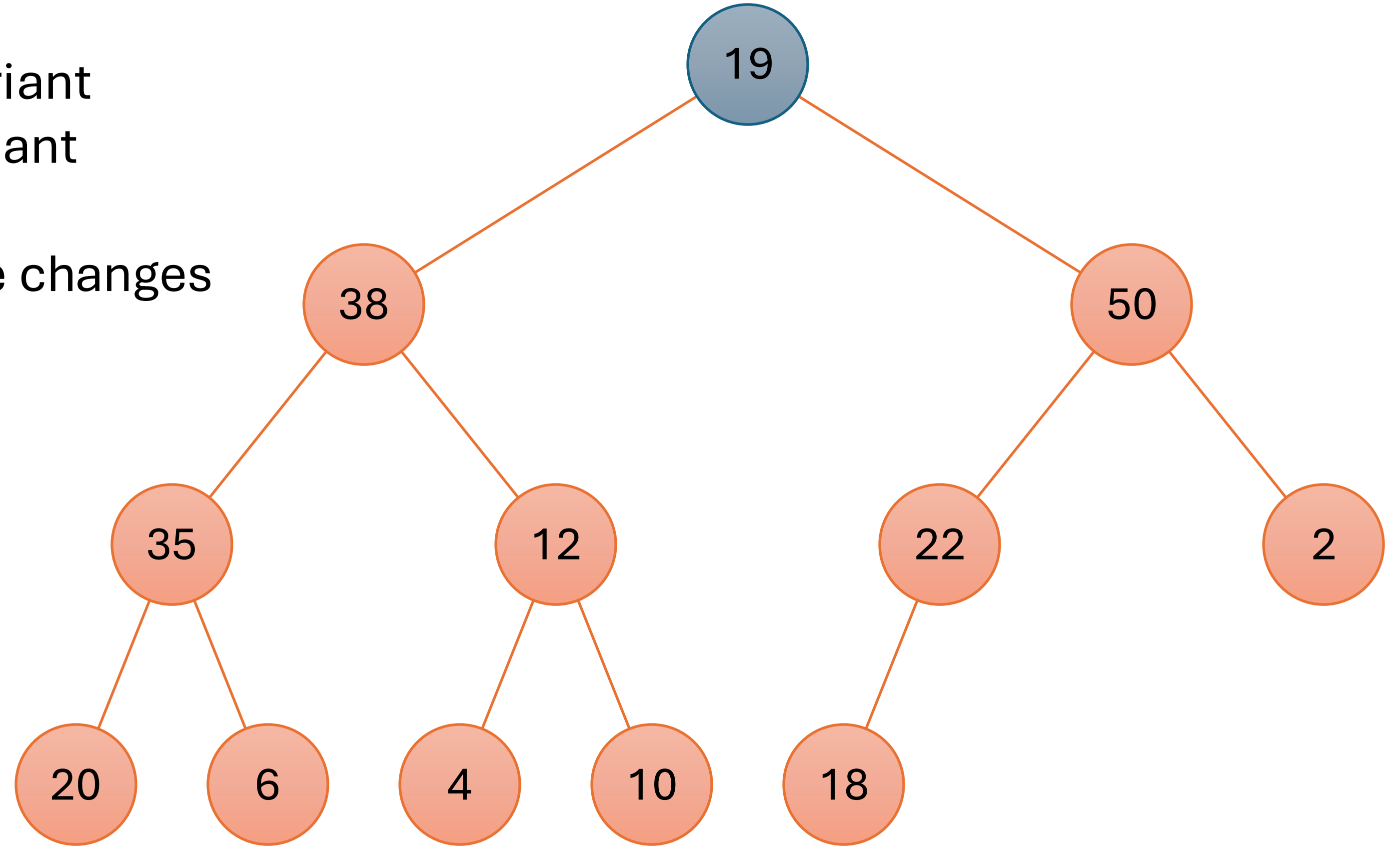
1. Save root element in a local variable
2. Assign last value to root, delete last node.

Exercise: restore the order invariant

Must preserve:

1. Shape invariant
2. Order invariant

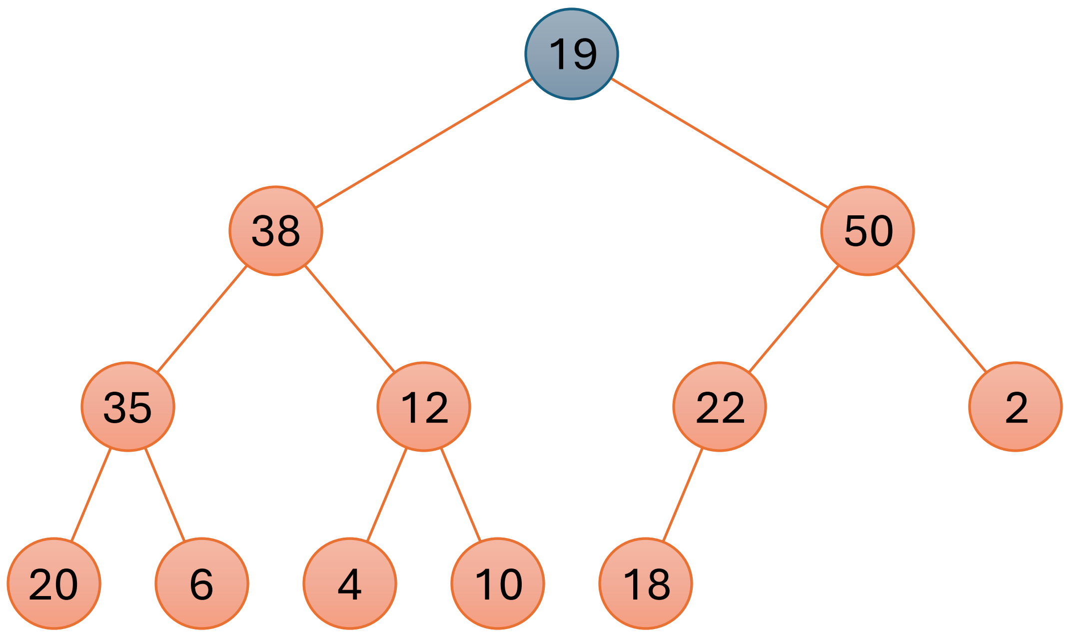
Goal: minimize changes



Checkpoint: Bubble down

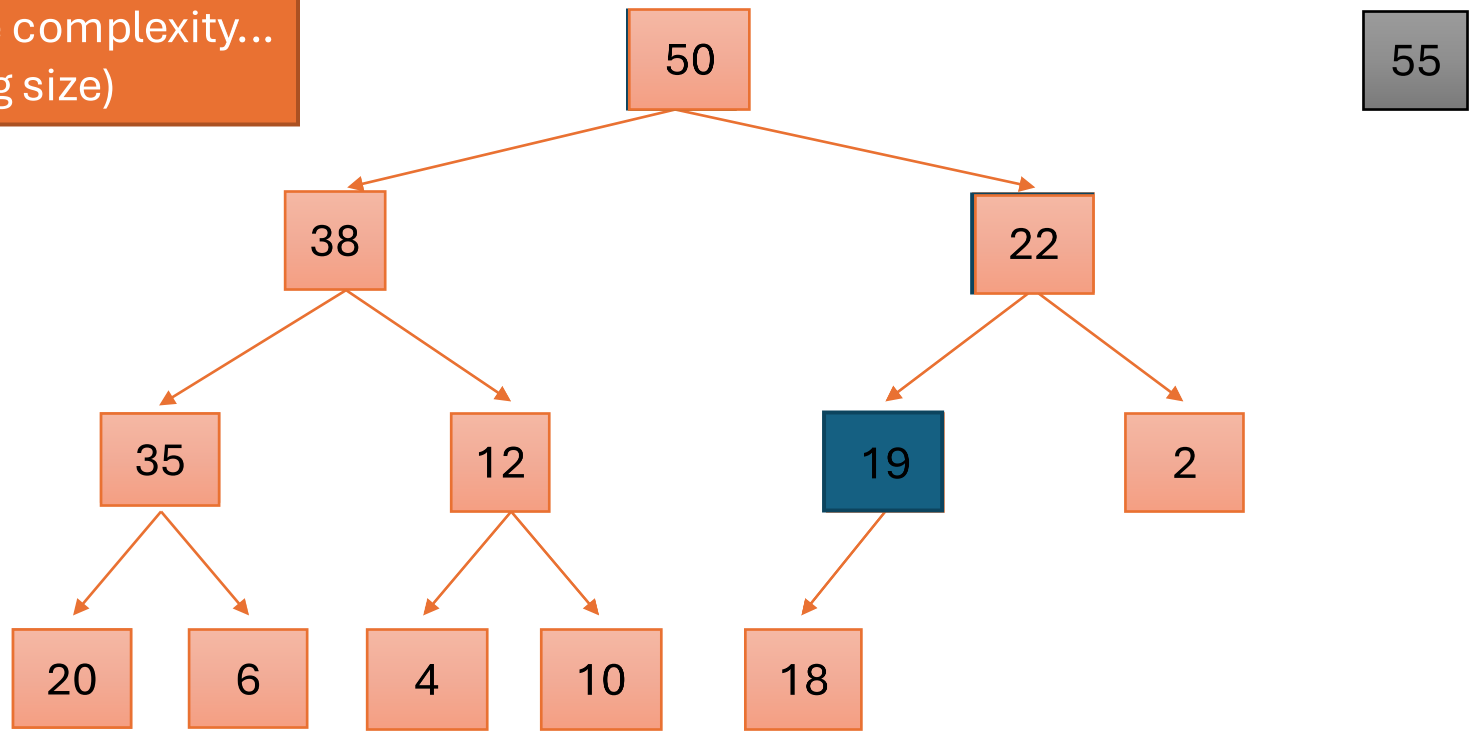
For a max-heap, when “bubbling down”, which child should you swap with?

A. Left
B. Right
C. Whichever is larger
D. Whichever is smaller
E. Doesn’t matter



Heap: `remove()`

Time complexity...
O(log size)



1. Save root element in a local variable
2. Assign last value to root, delete last node.
3. While less than a child, switch with bigger child (bubble down)

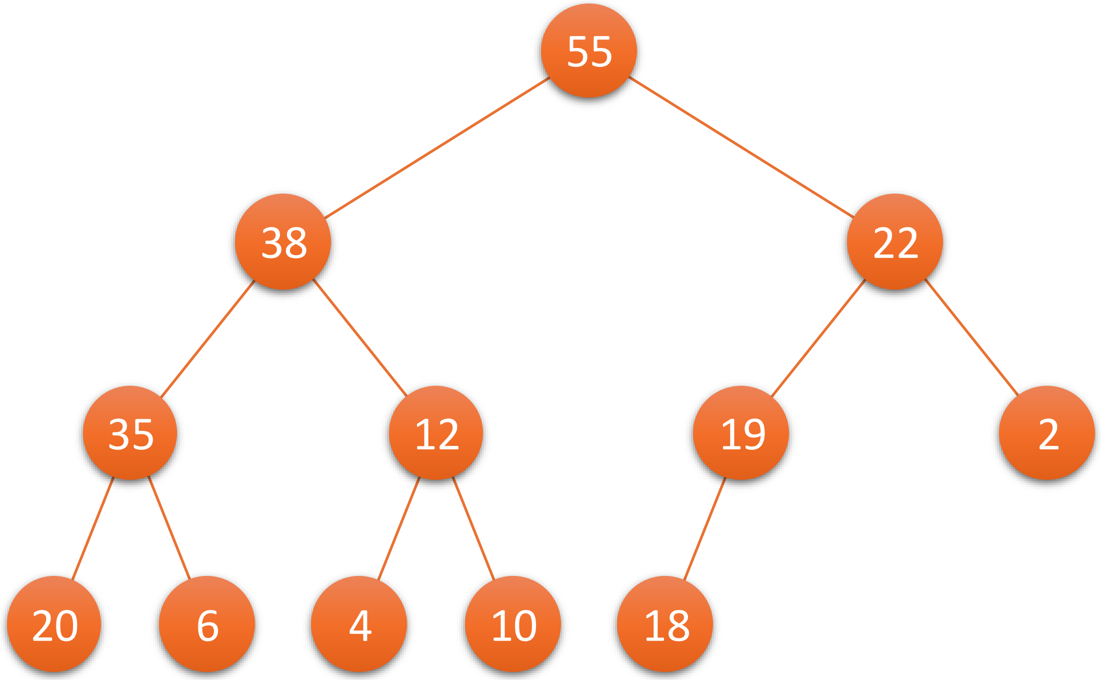
Specifying priorities

- Use element ordering: `Comparable` or `Comparator`
  - Example: Assignments ordered by their due date
  - Used by [`java.util.PriorityQueue<E>`] (min-heap)
- Separate priority values: heap stores (*element*, *priority*) pairs

Wrapup: Heap sort

Sorting with a heap

- A heap is *not* sorted
- But repeatedly removing values will yield them in order
  - **Max heap: descending order**
  - Min heap: ascending order
- Each removal takes $O(\log N)$ time (worst case)
  - N removals is $O(N \log N)$
- See animation in lecture notes



Metacognition

- Take 1 minute to write down a brief summary of what you have learned today

closing announcements to follow…

Announcements

- Prelim 2 coming up Thursday 11/6
  - Conflict survey due Thurs 10/30 by 5pm -------→
  - Make sure you are all caught up in your learning
  - Ed post and practice exam coming Thursday
- Keep each other healthy!