L21: (Balanced) Binary Search Trees

Alex Steiger

CompSci 201: Spring 2024

4/1/2024

Logistics, Coming up

- This Wednesday, April 3
  - APT 8 due
- Next Monday, April 8
  - Project P5: Huffman due
- Next Wednesday, April 10
  - APT Quiz 2 due
  - Covers linked list, sorting, trees
  - No regular APTs this week, just the quiz

Reminder: What is an APT Quiz?

- Set of 3 APT problems, 2 hours to complete.
  - Will be available starting this Saturday afternoon (look for a Canvas/email announcement)
  - Must *complete* by 11:59 pm Wednesday 4/12 (so start before 10)

- Start the quiz from link in instructions doc (like Quiz 1), begins your timer
  - Will look/work just like the regular APT page, just with only 3 problems.

Reminder: What is allowed?

Yes, allowed

- Zybook
- Course notes
- API documentation
- VS Code
- Jshell
- Searching internet for API usage, common patterns

No, not allowed

- Collaboration or sharing any code, including with ChatGPT/Copilot/other generative AI.
- Communication about the problems ***at all*** during the window.
- Searching internet for solutions.

Reminder: Don’t do these things

1. Do not collaborate. Note that we log all code submissions and will investigate for academic integrity.

2. Do not hard code the test cases (if(input == X) return Y, etc.).

   We show you the test cases to help you debug. But we search for submissions that do this and **you will get a 0 on the APT quiz if you hard code the test cases** instead of solving the problem.

Reminder: How is it graded?

Not curved, adjusted. 3 problems, 10 points each.

Raw score R out of 30.	Adjusted score A out of 30.	100 point grade scale
27 <= R <= 30	A = R	90 – 100
24 <= R <= 26	A = 26	~87
21 <= R <= 23	A = 25	~83
18 <= R <= 20	A = 24	80
15 <= R <= 17	A = 23	~77
12 <= R <= 14	A = 22	~73
9 <= R <= 11	A = 21	70
6 <= R <= 8	A = 20	~67
3 <= R <= 5	A = 19	~63
1 <= R <= 2	A = 18	60

Can still get in the B range even if you can't solve one; don't panic!

Only going to get a 0 if you collaborate or hard code test cases. Don't do it!

Binary Heap Wrapup

Reminder: You can see a simple DIY implementation of an array-based (actually ArrayList) binary heap at https://coursework.cs.duke.edu/cs-201-spring-24/live-coding/-/tree/main/DIYBinaryHeap

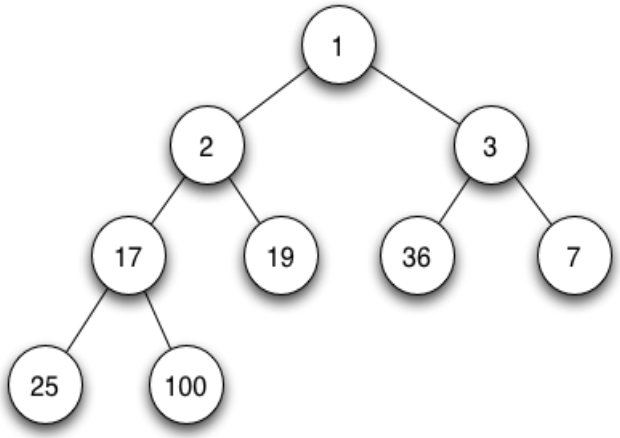
java.util.PriorityQueue API

Method	Behavior	Runtime Complexity
`add(element)`	Add an element to the priority queue	O(log(N)) comparisons
`remove()`	Remove and return the *minimal* element	O(log(N)) comparisons
`peek()`	Return (do *not* remove) the minimal element	O(1)
`size()`	Return number of elements	O(1)

Binary Heap at a high level

A **binary heap** is a binary tree satisfying the following structural invariants:

- ***heap property***: *every node is less than or equal to its successors*, and
- ***shape property***: *the tree is* ***complete*** (full except possibly last level, in which case it should be filled from left to right)



By Vikingstad at English Wikipedia - Transferred from en.wikipedia to Commons by LeaW., Public Domain, https://commons.wikimedia.org/w/index.php?curid=3504273

L20-WOTO2-BinaryHeap-Sp24

Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

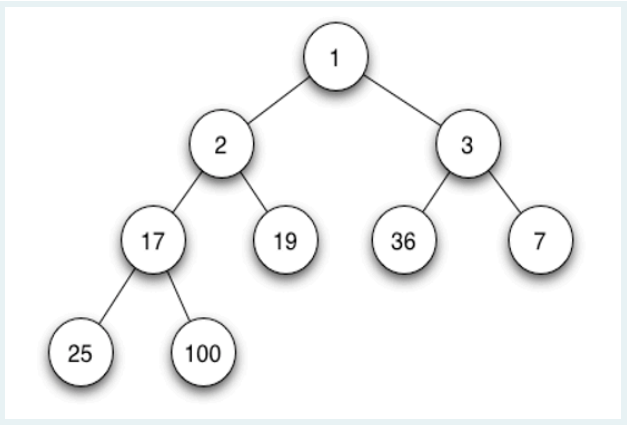
1

NetID *

solutions

2

Which of the following correctly shows the order in which elements of this heap would be stored in an array representation? *



- 1, 2, 17, 25, 100, 19, 3, 36, 7
- 1, 2, 3, 7, 17, 19, 25, 36, 100
- 1, 2, 3, 17, 19, 36, 7, 25, 100

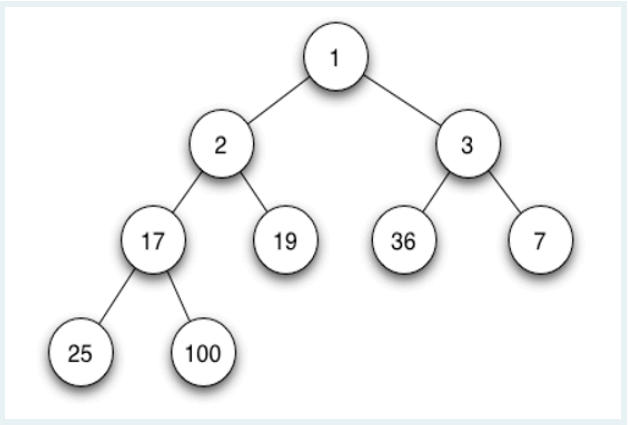
3

When 1-indexing, if a value in the heap is stored at **index 11** in the array representation, at which index will its **parent** be stored? *

- (●) 5
- ( ) 6
- ( ) 10
- ( ) 12
- ( ) 21
- ( ) 22

4

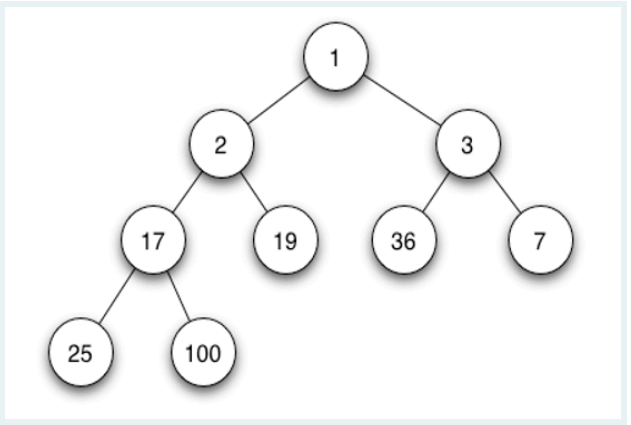
Suppose we add the value 20 to this heap. At what index in the array representation will it be added before any swapping? Assume we are 1-indexing the array (that is, the first element is stored at index 1). *



- ( ) 0
- ( ) 1
- ( ) 9
- (●) 10
- ( ) 20

5

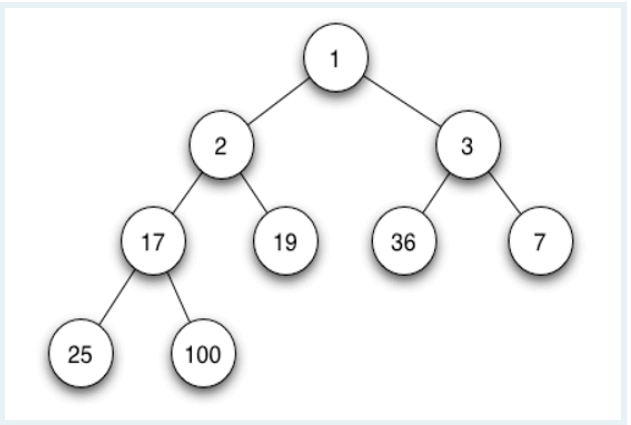
Suppose we add the value 20 to this heap. How many swaps will be needed to reestablish the heap property? *



- (●) 0
- ( ) 1
- ( ) 2
- ( ) 3

6

Suppose we remove the minimum value from this heap. AFTER replacing the root with the last node in the heap, how many swaps will be necessary to reestablish the heap property? *



- ( ) 0
- ( ) 1
- ( ) 2
- (●) 3

7

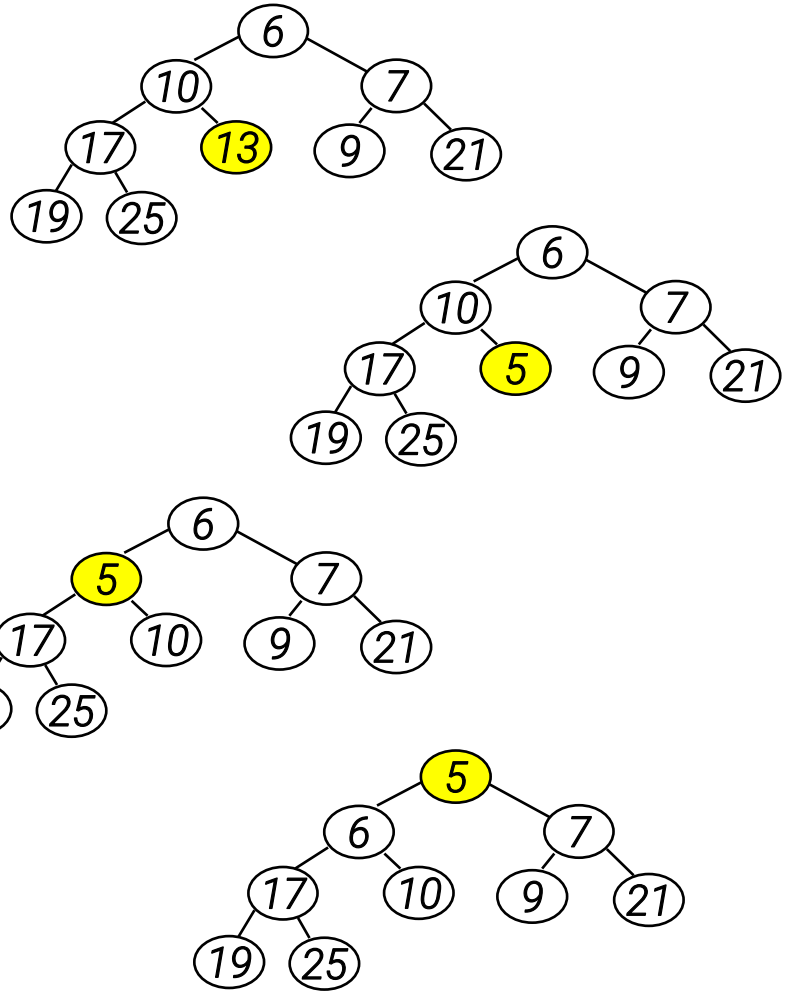
True or false: For a heap with N > 3 elements, it is possible that an add or remove operation might require N swaps to reestablish the heap property. *

- ○ True
- ◉ False

decreaseKey Operation?

- decreaseKey(val, newVal) changes val in heap to (smaller) newVal

- Suppose we decrease the 13 to 5
- Violates heap property
- Fix like in the add operation:
  - While violating heap property, swap with parent



decreaseKey NOT in java.util

- decreaseKey is important for some algorithms, but not supported in many standard libraries (including the java.util.PriorityQueue)
  - Used in later lectures!

- Why not supported?
  - Note that binary heap does not support efficient ***search***
  - In order to do decreaseKey in O(log(n)) time, need to store ***references/indices*** of all the "nodes."
  - Adds overhead, not done in java.util

Alternative Implementation: **Balanced** Binary Search Tree

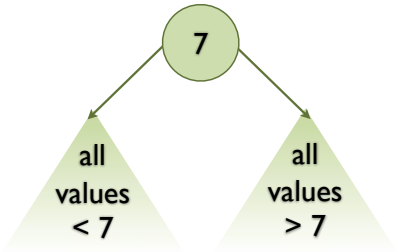
A binary tree is a binary ***search*** tree if *for every node*:

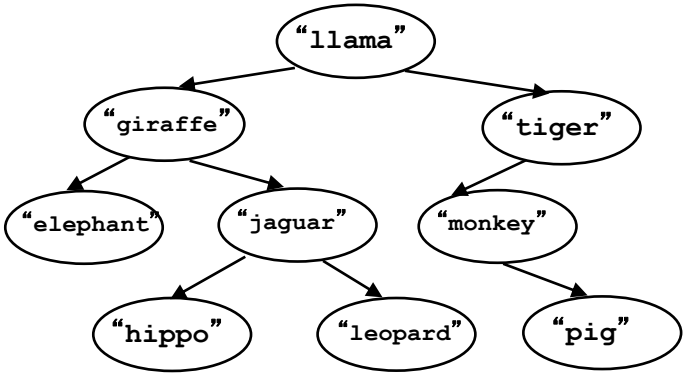
- Left subtree values are all ***less than*** the node's value

AND

- Right subtree values are all ***greater than*** the node's value

According to some ordering (natural ordering if Comparable or defined by Comparator)

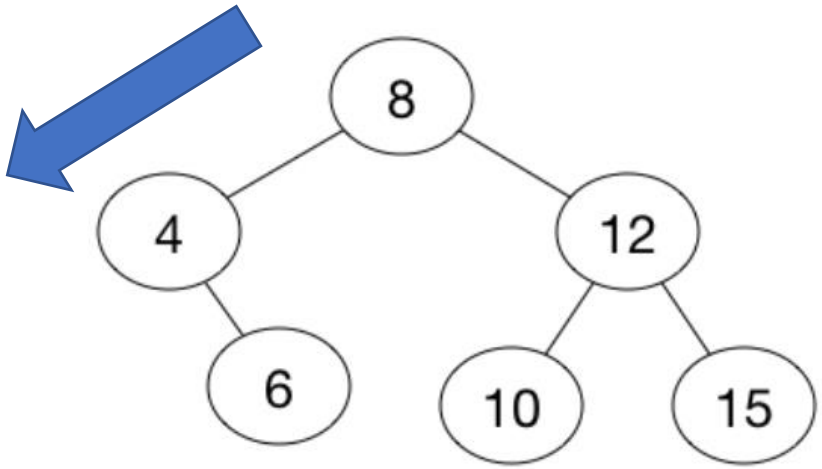




Enables efficient search, similar to binary search!

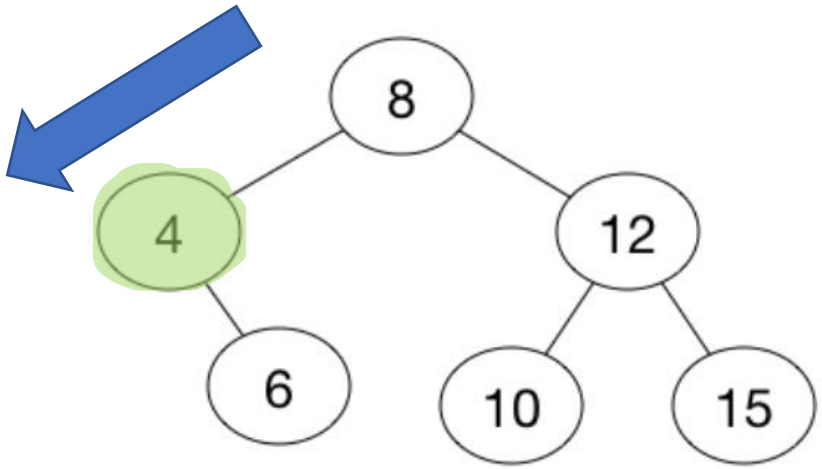
Alternative Implementation: **Balanced** Binary Search Tree

- If your keys happen to be unique…
- Can support O(log(n)) add & remove (including smallest) using a **balanced** binary search tree!
- Smallest is leftmost child



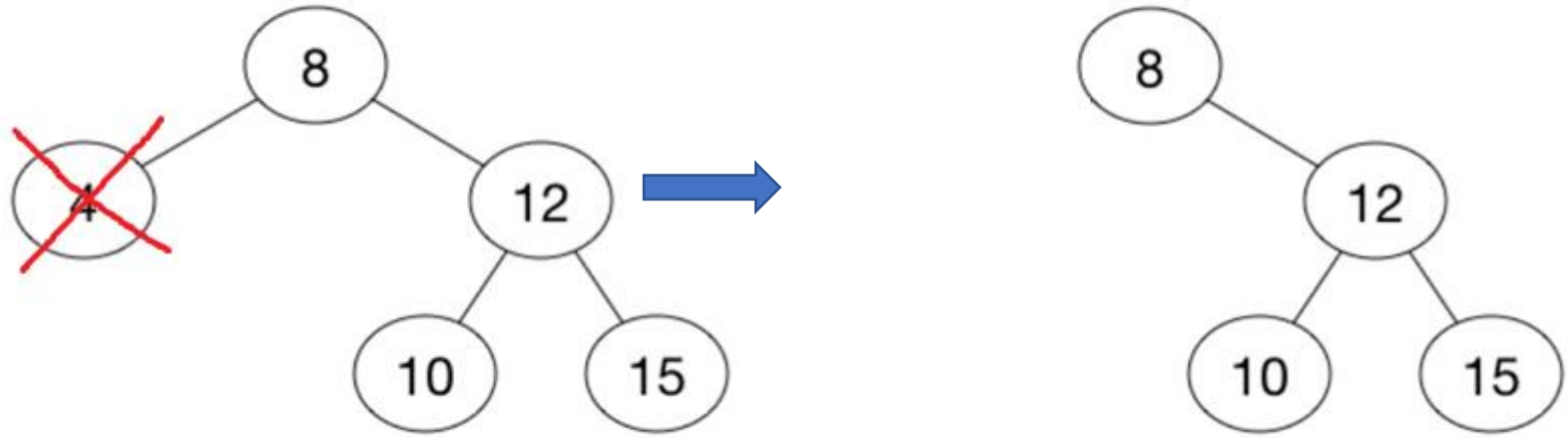
Removing the Minimum

- Smallest is leftmost child
- Iterate left until no left child
  - This is the node to remove and value to return
- How to repair BST after removal?



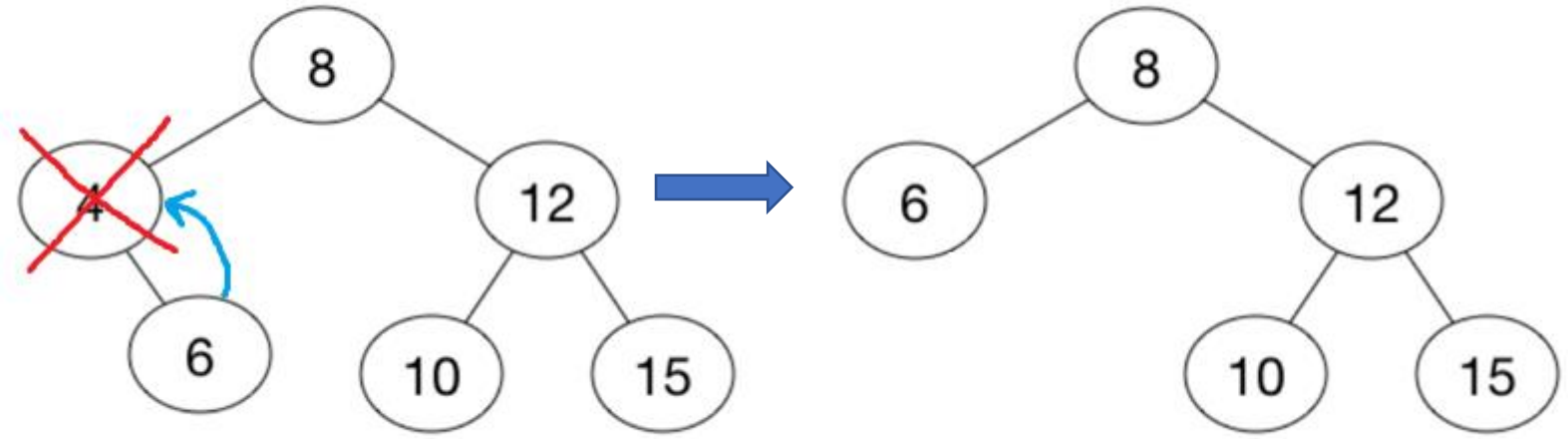
Removing the Minimum

- Smallest is leftmost child
- Iterate left until no left child
  - This is the node to remove and value to return
- How to repair BST after removal?
  - If leftmost is a leaf, set parent's left to null



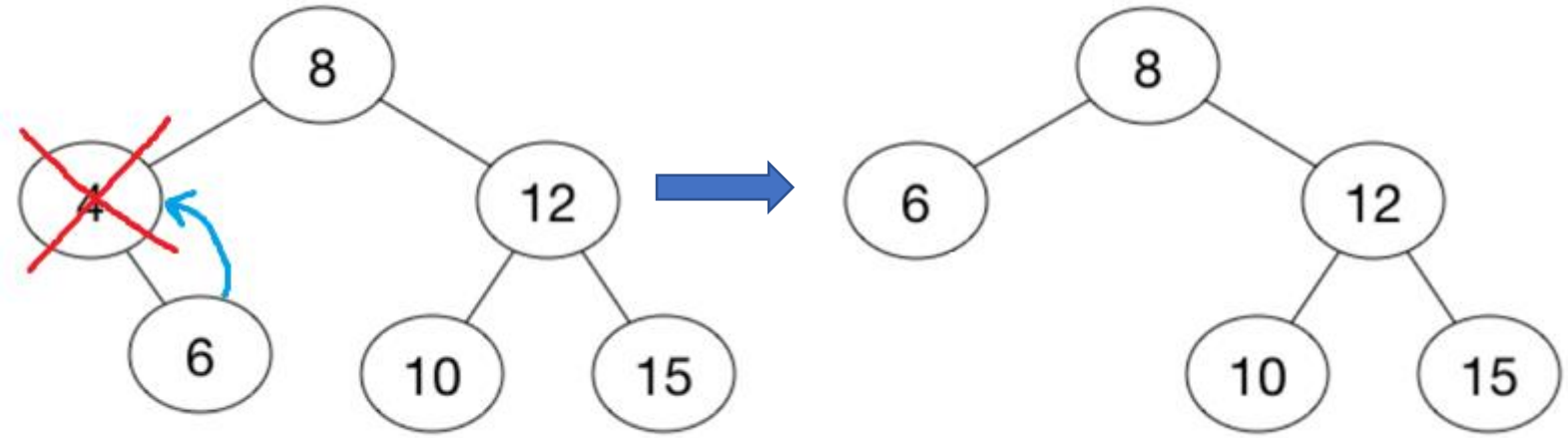
Removing the Minimum

- How to repair BST after removal?
  - If leftmost is a leaf, set parent's left to null
  - If leftmost has right child?
    - Leftmost's right child becomes parent's left child
    - BST property restored!



Removing the Minimum

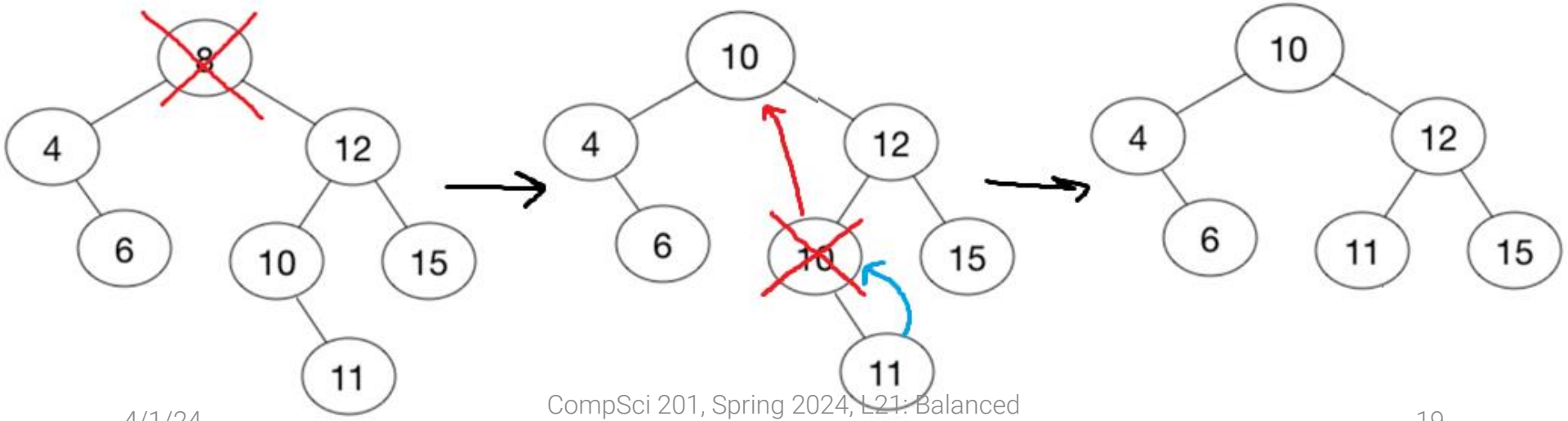
- How to repair BST after removal?
  - If leftmost is a leaf, set parent's left to null
  - If leftmost has right child?
    - Leftmost's right child becomes parent's left child
    - BST property restored!
  - If leftmost has left child? Can't happen (not the leftmost)



Aside: Removing any node?

Updated 4/4/24

- How to repair BST after removal?
  - Removal for node with 0-1 children generalizes from leftmost case
  - Node with 2 children?
    - Replace it with ***minimum*** in ***right*** subtree (leftmost!), then remove it using previous remove-min procedure
    - Restores BST property



Disadvantages to using a Binary Search Tree for your priority queue?

1. All elements must be unique

   (There exist variations of BSTs that handle duplicates, but more complicated / depends on usage)

2. Not array-based, uses more memory and has higher constant factors on runtime

3. Much harder to implement with guarantees that the tree will be balanced. ???

Live Coding

- BST with RemoveMin