CSE 373: Data Structures and Algorithms

Binary Search Trees

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

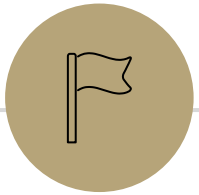
Administrivia

Due dates:

- HW2 Part 1 due today 11:59pm
- If you are submitting late, fill the late submission form (link available on the website)
- Double check that you are using the right checkstyle (CSE373-checkstyle)
  - See Step 2b in Eclipse setup (https://courses.cs.washington.edu/courses/cse373/18au/resources/eclipse-setup.html)
  - See HW2 Part 0b

Notes

- Any questions on HW2 Part 1 logistics?
- You can continue working on HW2 Part 2 (i.e., pushing commits to your repository). It won't affect your HW2 Part 1 submission.
- Section 2 solutions will be posted today 5pm (generally, we'll post Section solutions on Fridays around 5pm)



Modeling complex functions

Some summations identities

```
for (int i = 0; i <= n-1; i++) {
    System.out.println("A");
}
```

If system.out.println takes constant time '1', T(n) = $\sum_{i=0}^{n-1} 1 = n$

If system.out.println takes constant time 'c', T(n) = $\sum_{i=0}^{n-1} c = cn$

Some summations identities

$T(n) = 1 + 1 + 1 + 1 + \ldots$ // n times

$$T(n) = \sum_{i=0}^{n-1} 1 = n$$

$T(n) = c + c + c + c + \ldots$ // n times

$$T(n) = \sum_{i=0}^{n-1} c = c \sum_{i=0}^{n-1} 1 = cn$$

Summation of a constant

$T(n) = 0 + 1 + 2 + 3 + \ldots + (n-1)$

$$T(n) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

Gauss's identity

https://courses.cs.washington.edu/courses/cse373/18au/resources/math-review.html

Modeling complex loops: Simplifying summations

```
for (int i = 0; i <= n-1; i++) {
    for (int j = 0; j <= i-1; j++) {
        System.out.println("A");
    }
}
```

+c { 0 + 1 + 2 + 3 +…+ i-1 } n

(0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ repeat n times

T(n) = $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c$ = $\sum_{i=0}^{n-1} ci$ Summation of a constant

= $c \sum_{i=0}^{n-1} i$ Factoring out a constant

= $c \frac{n(n-1)}{2}$ Gauss's Identity

= $\frac{c}{2} n^2 - \frac{c}{2} n$ $O(n^2)$

Modeling recursion

```
public int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
 }
```

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 0, 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

Modeling recursion: Unfolding Method

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 0, 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

$$T(n) = C_2 + T(n-1)$$

$$T(n) = C_2 + C_2 + T(n-2)$$

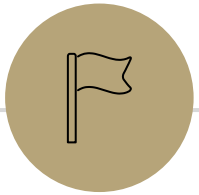
$$T(n) = C_2 + C_2 + C_2 + T(n-3)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + T(2)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + C_2 + T(1)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + C_2 + C_1$$

$$T(n) = (n-1)C_2 + C_1$$



Binary Search Trees

Storing Sorted Items in an Array

get() – O(logn)

put() – O(n)

remove() – O(n)

Can we do better with insertions and removals?

Trees!

A tree is a collection of nodes

- Each node has at most 1 parent and 0 or more children

Root node: the single node with no parent, "top" of the tree

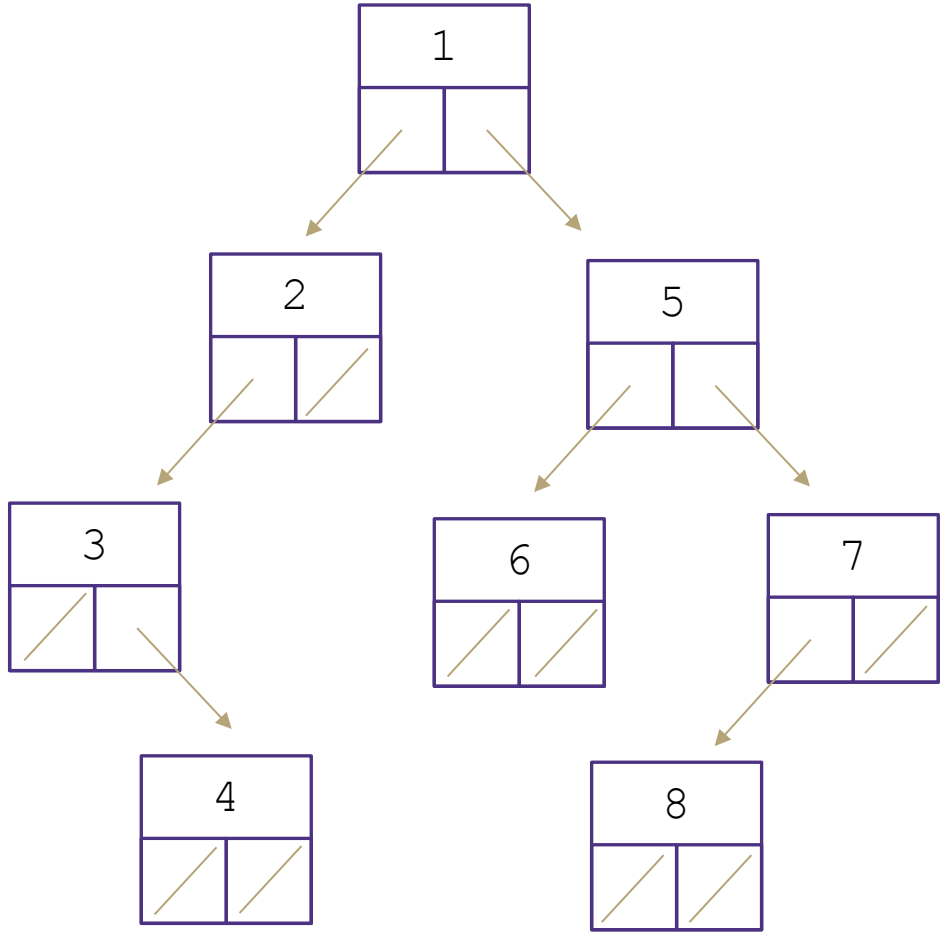
Branch node: a node with one or more children

Leaf node: a node with no children

Edge: a pointer from one node to another

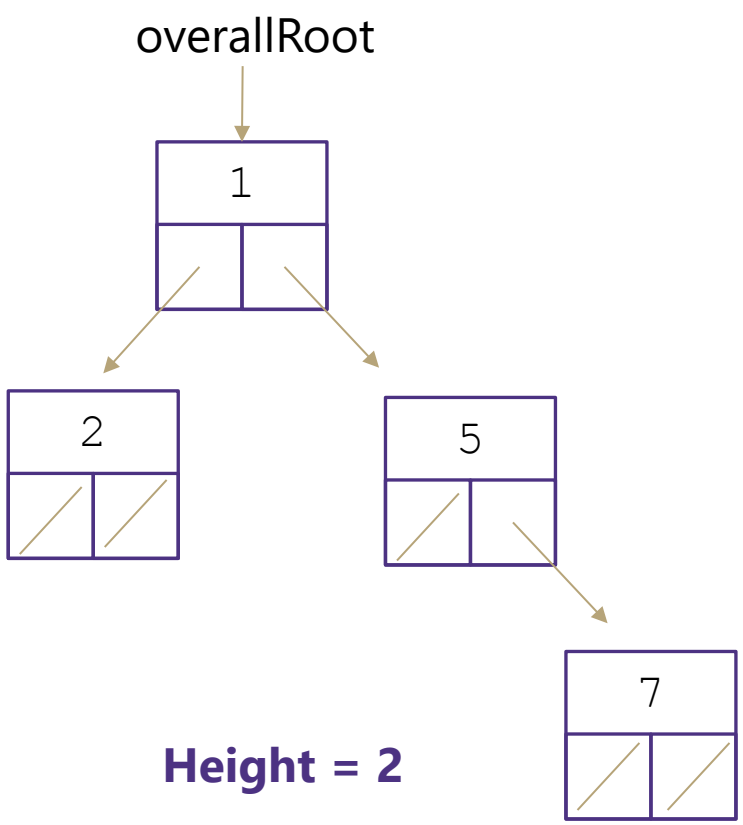
Subtree: a node and all it descendants

Height: the number of edges contained in the longest path from root node to some leaf node

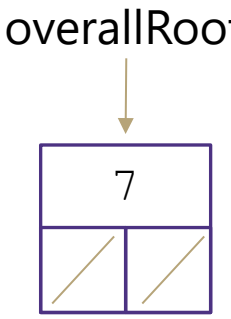


Tree Height

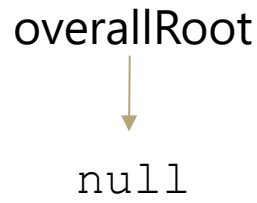
What is the height of the following trees?



Height = 2



Height = 0



Height = -1 or NA

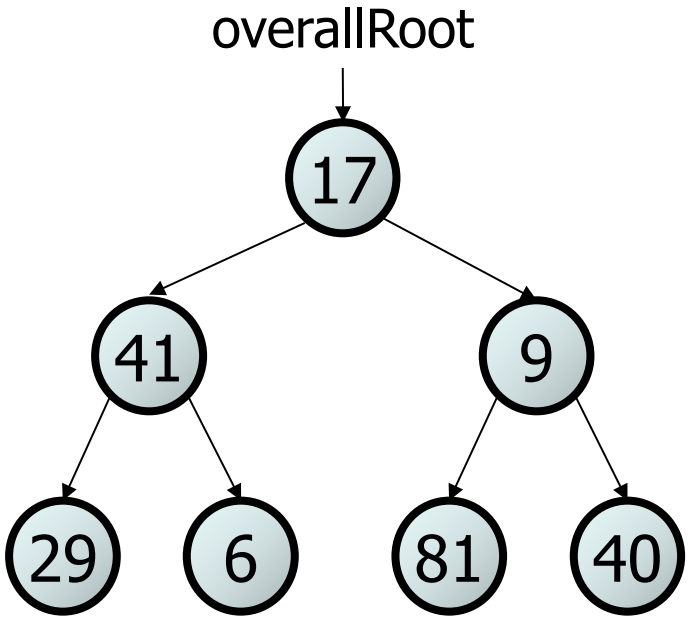
Traversals

traversal: An examination of the elements of a tree.

- A pattern used in many tree algorithms and methods

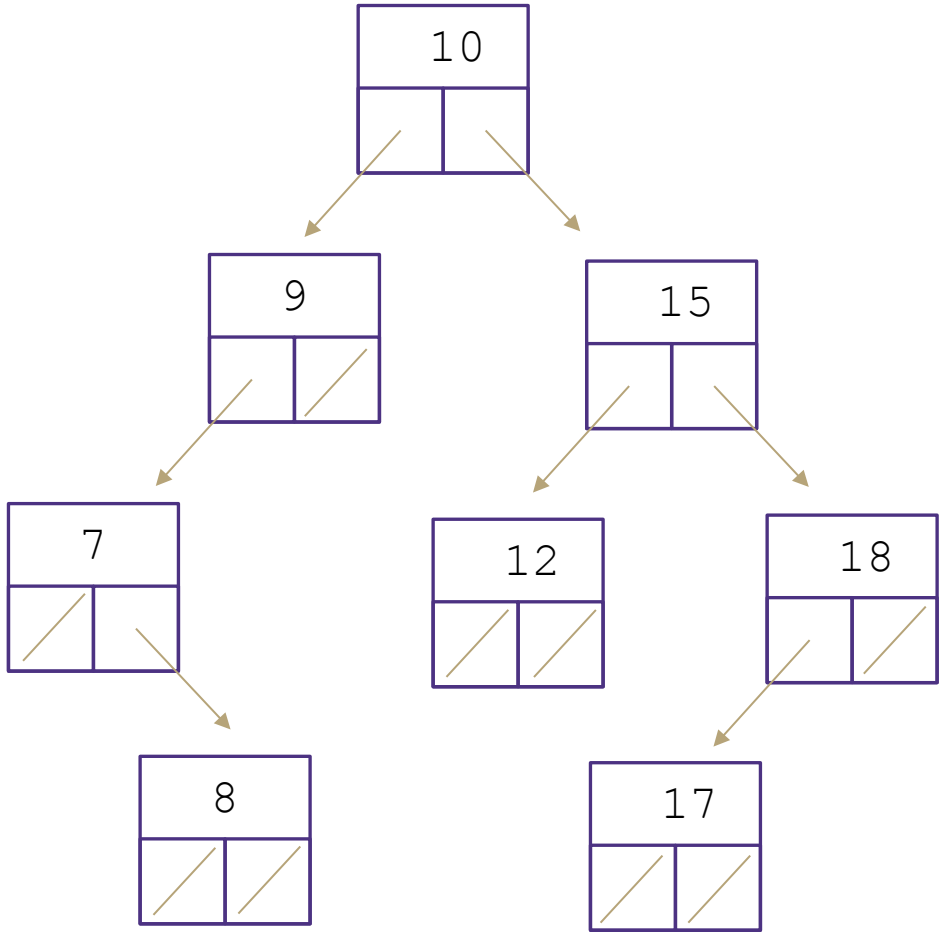
Common orderings for traversals:

- **pre-order**: process root node, then its left/right subtrees
  - `17 41 29 6 9 81 40`
- **in-order**: process left subtree, then root node, then right
  - `29 41 6 17 81 9 40`
- **post-order**: process left/right subtrees, then root node
  - `29 6 41 81 40 9 17`



Binary Search Trees

A binary search tree is a binary tree that contains comparable items such that for every node, all children to the left contain smaller data and all children to the right contain larger data.



Binary Search Trees

Binary Search Trees allow us to:

- quickly find what we're looking for
- add and remove values easily

Dictionary Operations:

Runtime in terms of height, "h"

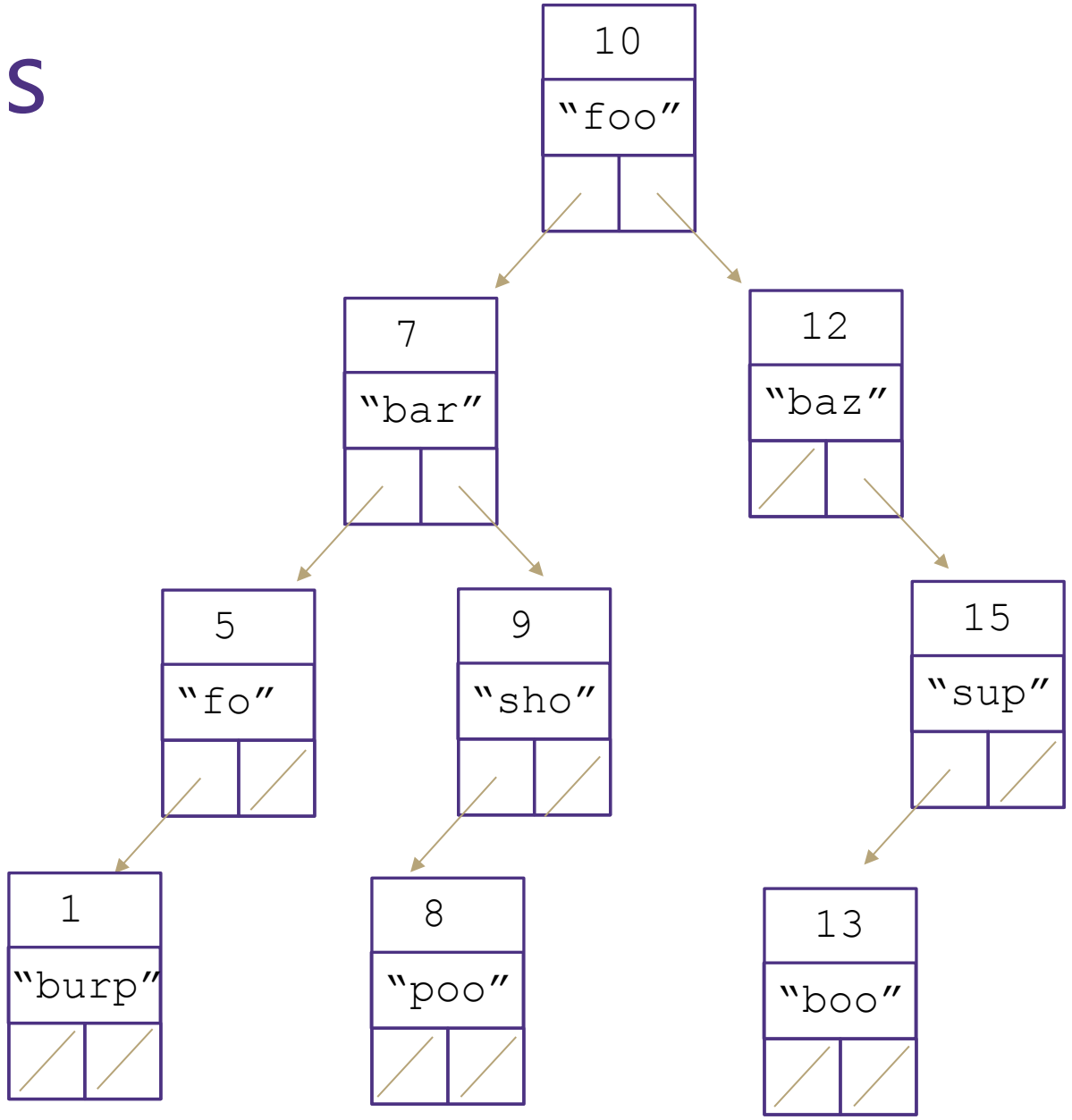
get() – O(h)

put() – O(h)

remove() – O(h)

What do you replace the node with?

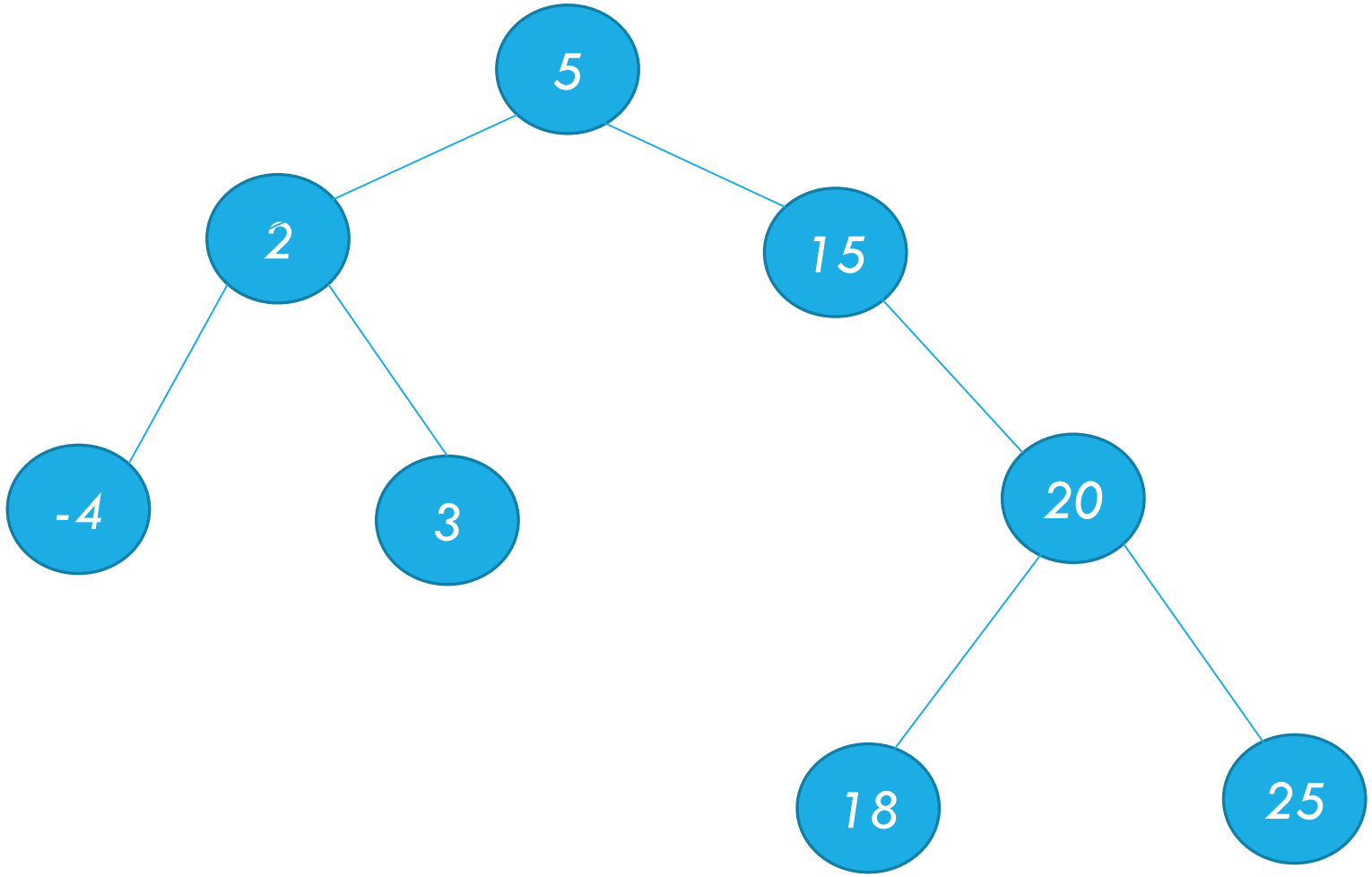
Largest in left sub tree or smallest in right sub tree



Binary Search Tree – deleting a node

remove(15)

// replace with 20



Binary Search Tree – deleting a node

remove(15)

// replace with 14 or 18