Binary Search Trees

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Compare binary search trees with sorted arrays
- Discuss the importance of a binary search tree's height
- Discuss common search tree algorithms

Exercise

- Search tree exercise

Extra Resources

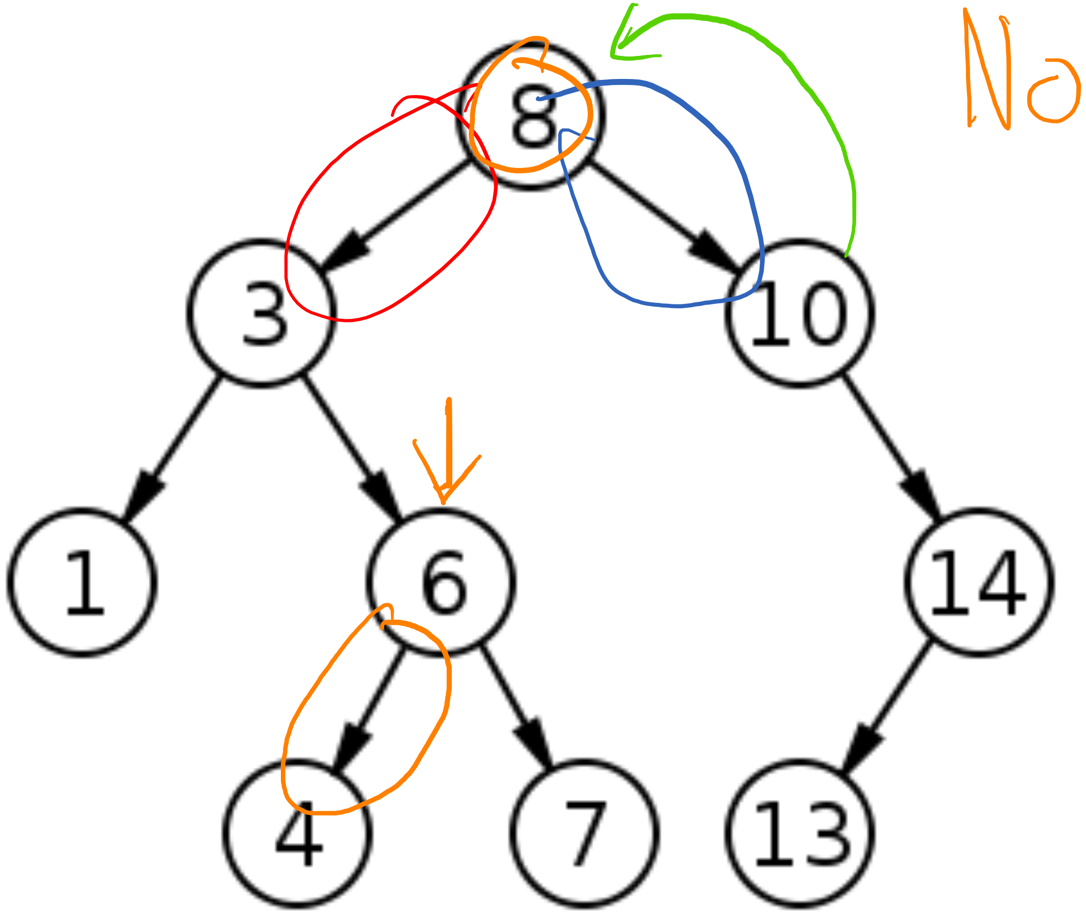
- Introduction to Algorithms, 3rd, chapter 12

Sorted Arrays

3	6	10	11	17	23	30	36

Operation	Running Time
Access	O(1)
Search	O(lg n)
Selection	O(1)
Predecessor	O(1)
Successor	O(1)
Output (print)	O(n)
Insert	O(n)
Delete	O(n)
Extract-Min	O(n)

Given a set of key values, is a BST unique?
(ignore ties)



No

Binary Search Tree

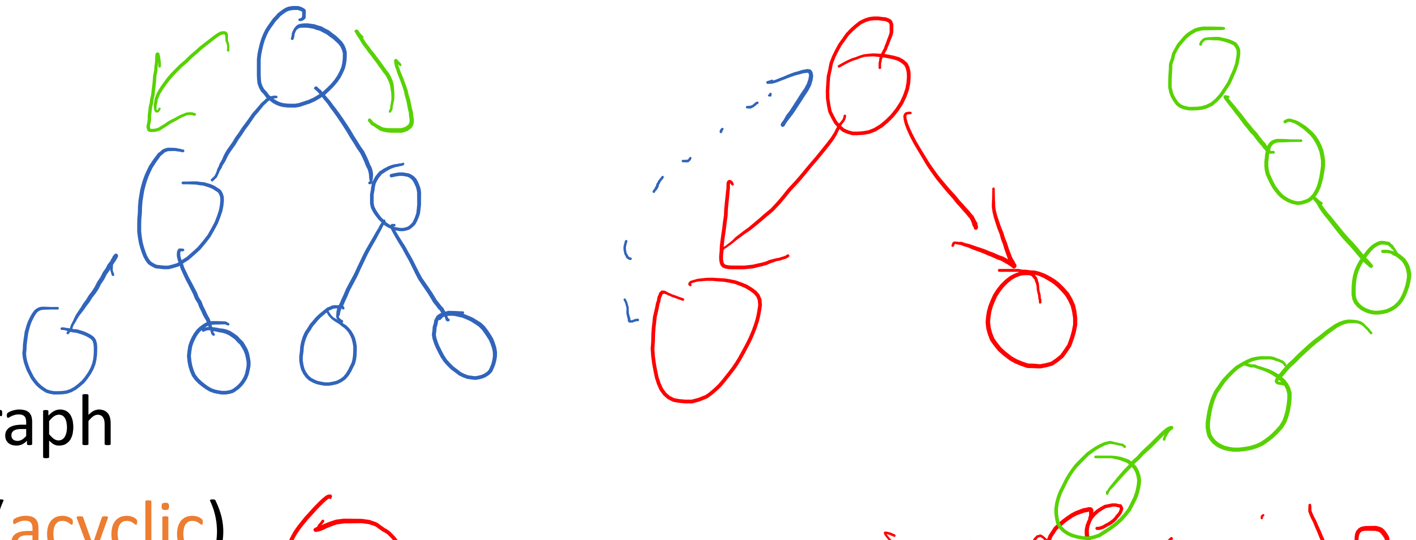
Each node has:

- A pointer to a left subtree
- A pointer to a right subtree
- A pointer to a parent node
- A piece of data (the key value)

Search tree property:

- All keys found in a left subtree must be less than the key of the current node
- All keys found in a right subtree must be greater than the key of the current node

Trees and Graphs



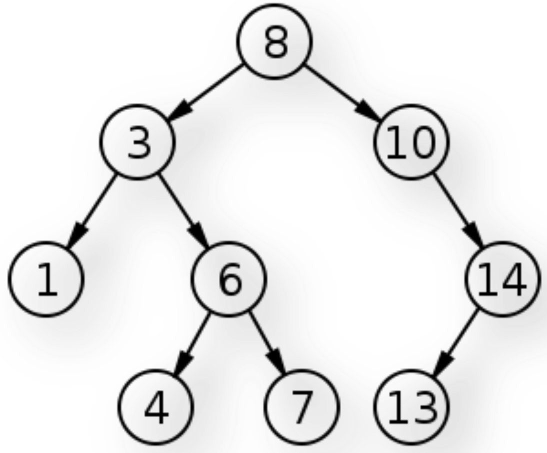
- Trees are a special type of graph
- Trees cannot contain cycles (acyclic)

if you ignore parent pointers

- Trees always have directed edges
- Trees have a single source (no incoming edges) vertex called root
- All tree vertices have one parent (except root, which has no parents)
- Trees always have n-1 edges
- BST compared to Heap?
  - Heap is always balanced, BSTs are not necessarily balanced
  - They have different properties (where are lesser values?)

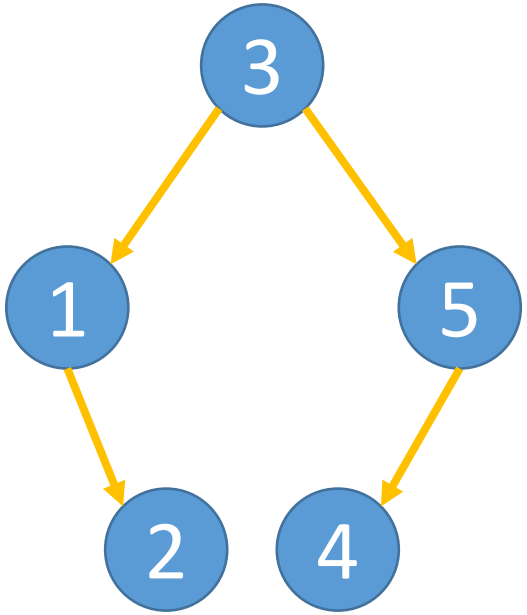
Balanced Binary Search Tree (vs Sorted Array)

Operation	Running Time
Access	O(1) → O(lg n)
Search	O(lg n)
Selection	O(1) → O(lg n)
Predecessor	O(1) → O(lg n)
Successor	O(1) → O(lg n)
Output (print)	O(n)
Insert	O(n) → O(lg n)
Delete	O(n) → O(lg n)
Extract Min	O(n) → O(lg n)

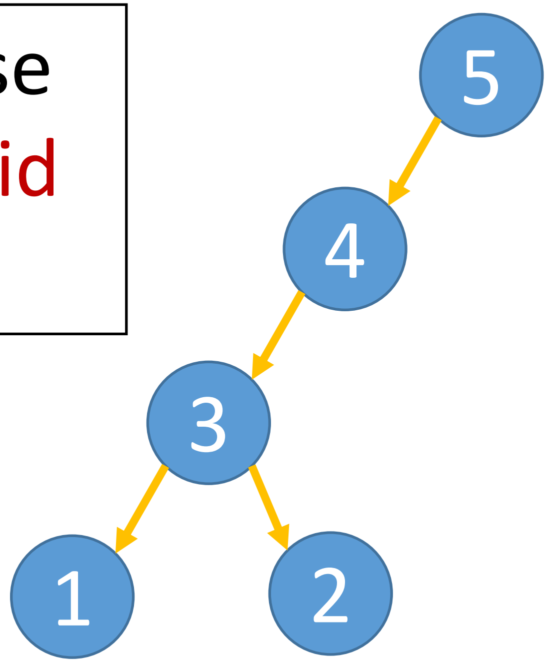


Height of a Binary Search Tree

- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)

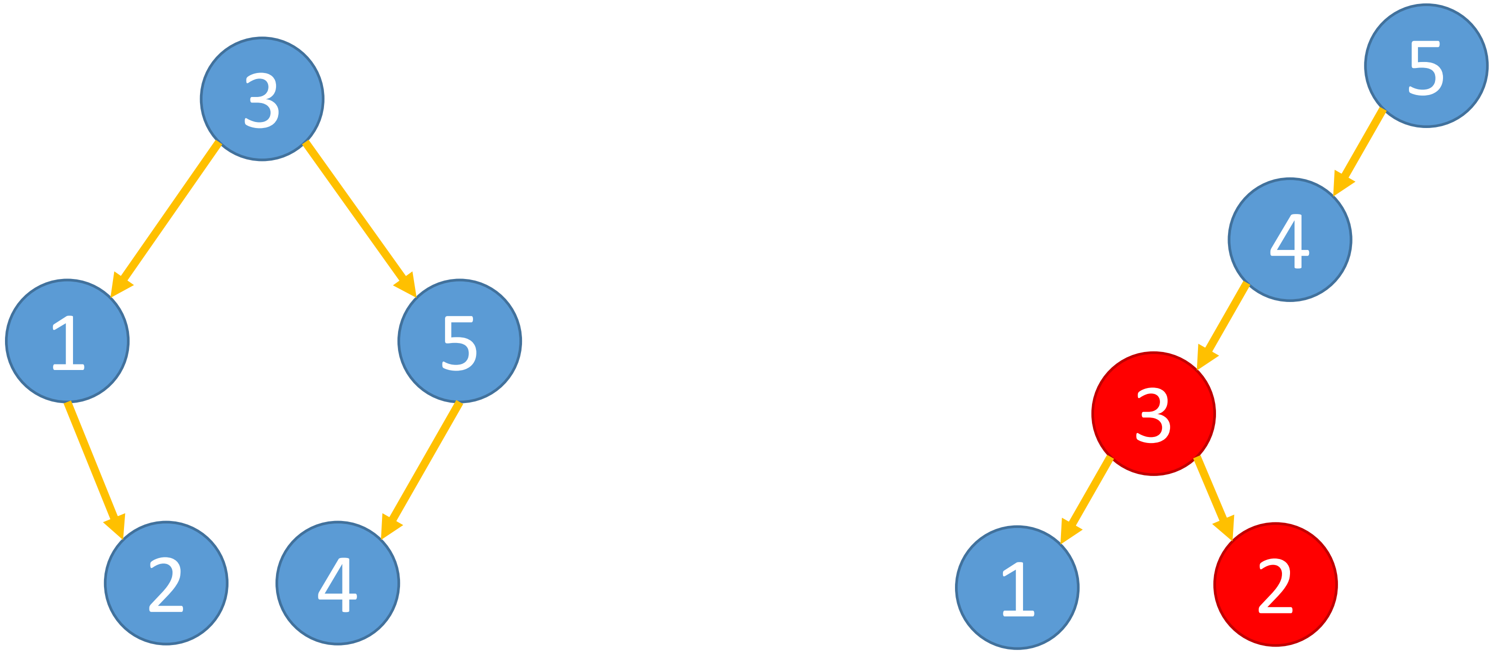


Are these both valid BSTs?



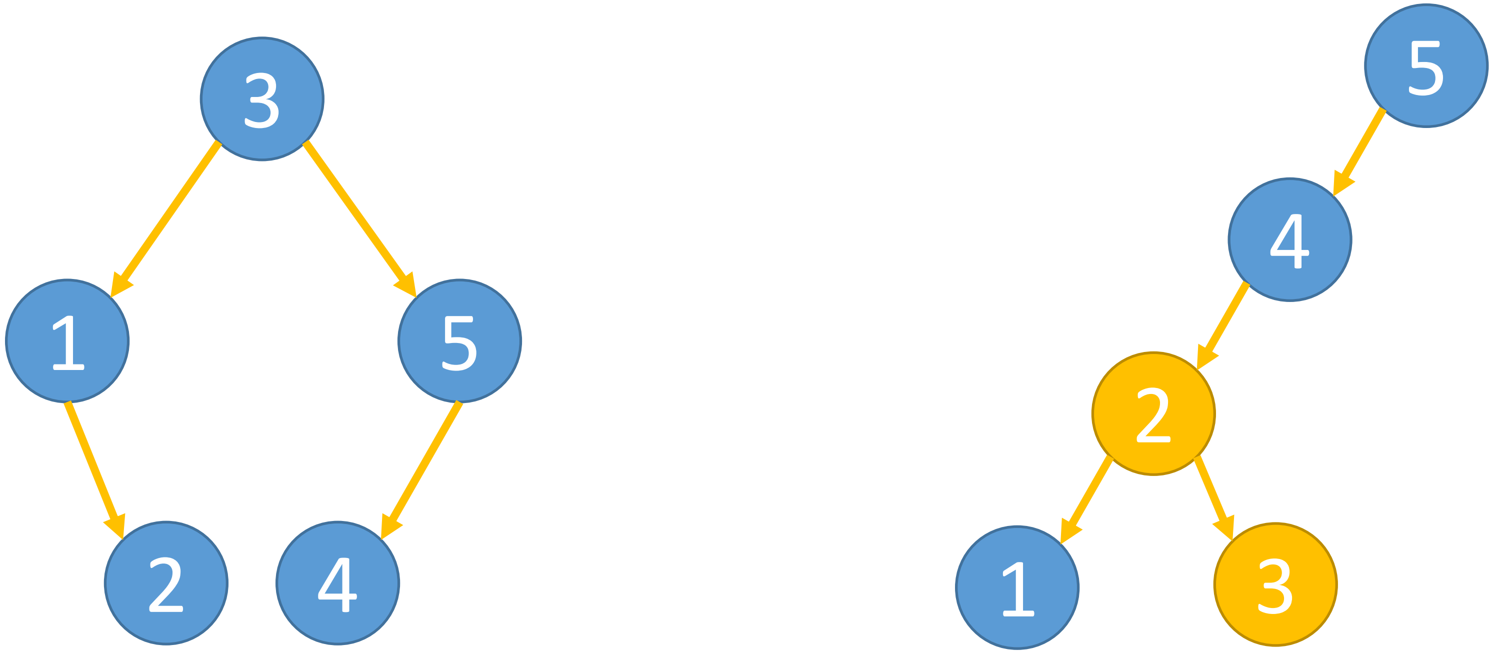
Height of a Binary Search Tree

- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)



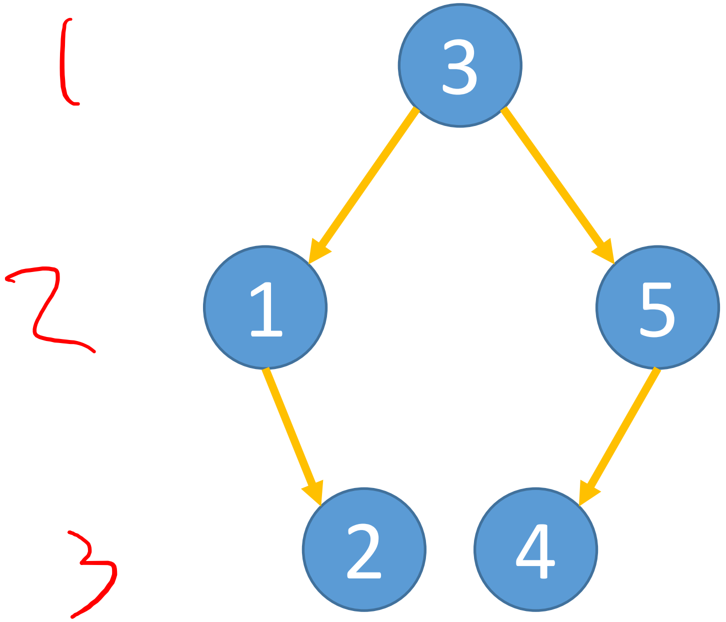
Height of a Binary Search Tree

- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)

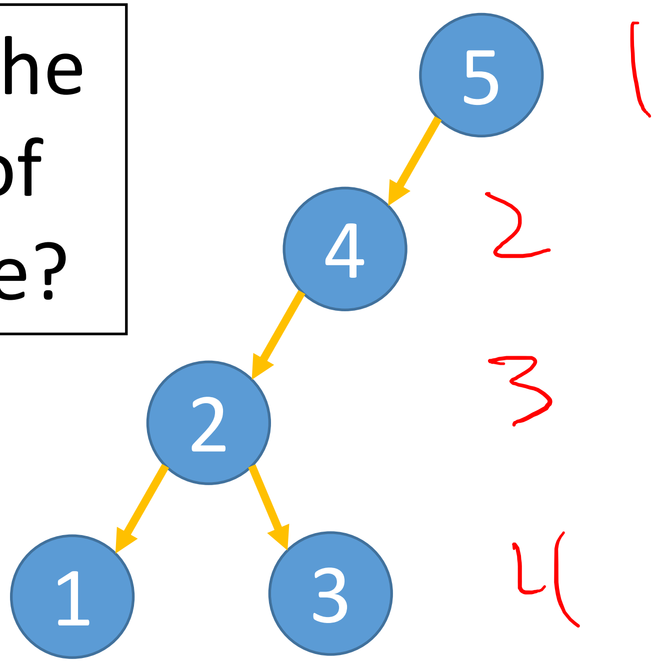


Height of a Binary Search Tree

- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)



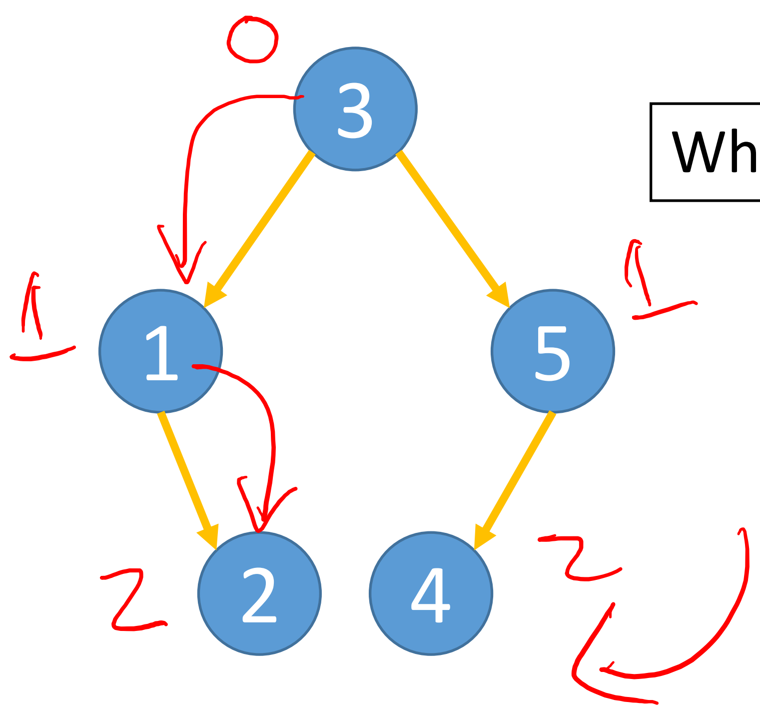
What is the height of each tree?



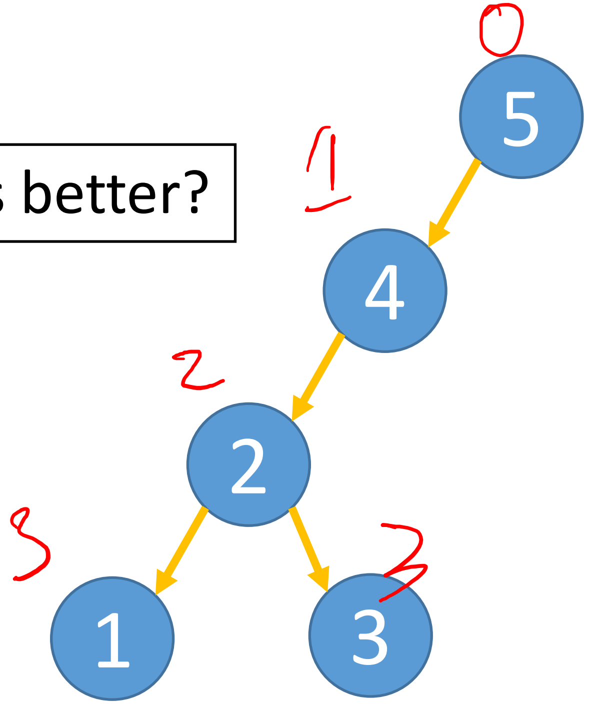
Height of a Binary Search Tree

- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)

$\frac{6}{5}$



Which one is better?

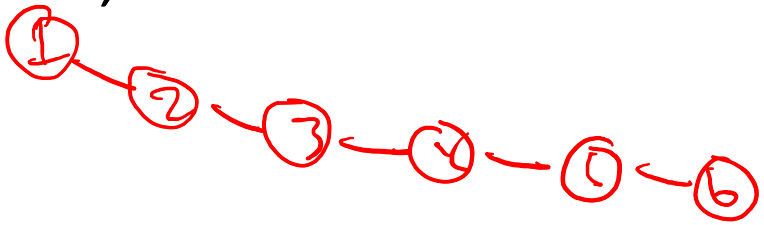


$\frac{9}{5}$

Height of a Binary Search Tree

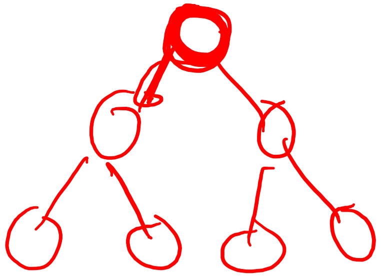
- Given a set of keys, we have many different choices for creating a binary search tree (we just have to satisfy the search tree properties)

- If we have n nodes, what is the maximum height of the tree?



n

- If we have n nodes, what is the minimum height of the tree?

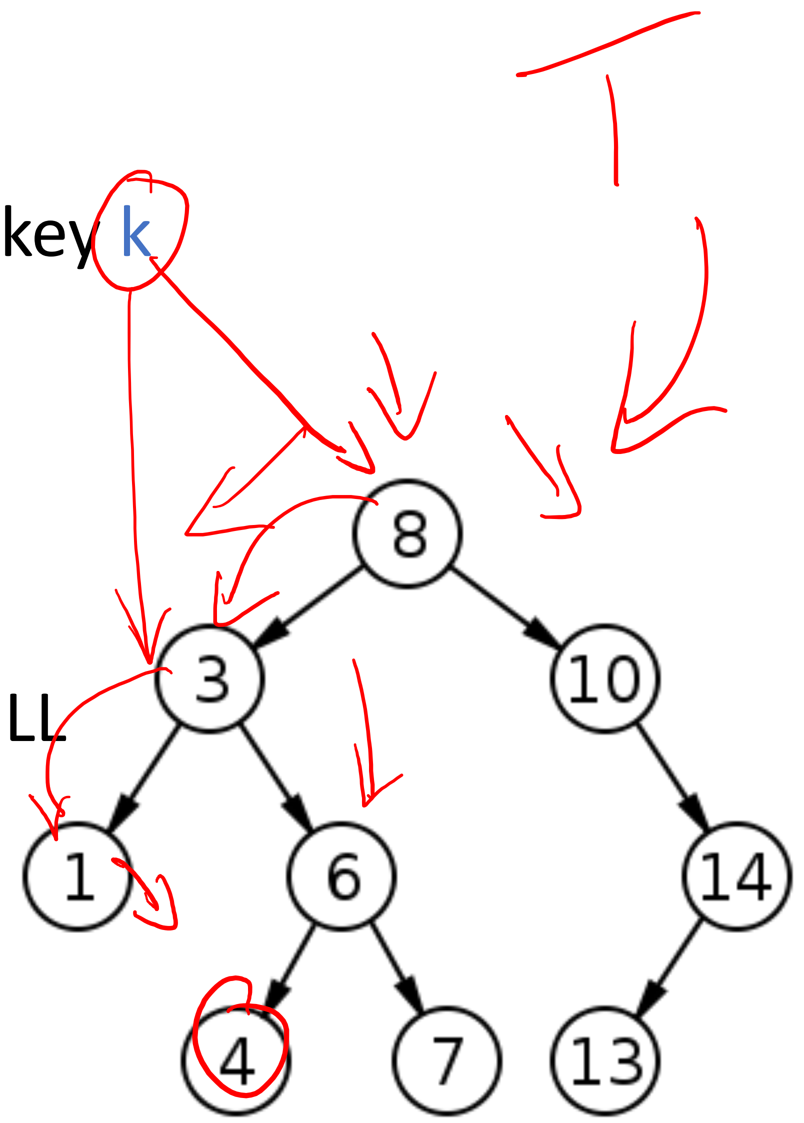


lg n

Searching a BST

Search the tree T for the key k

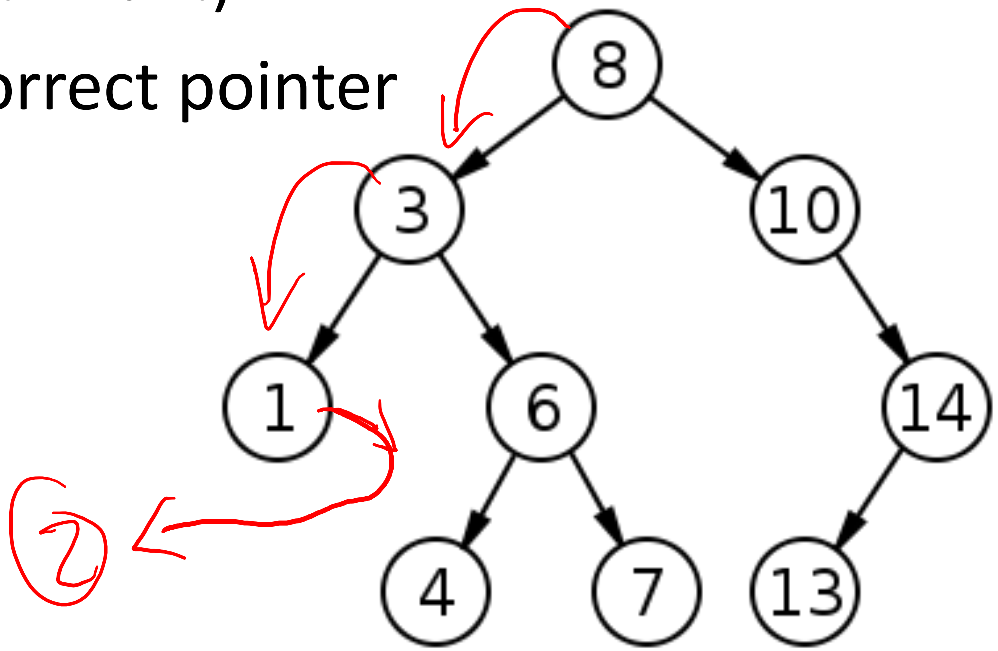
1. Start at the root node
2. Recursively:
    1. Traverse left if k < current key
    2. Traverse right if k > current key
3. Return the node when found or return NULL



Inserting into a BST

Insert the key k into the tree T

1. Start at the root node
2. Search for the key k (probably won't find it)
3. Create a new node and setup the correct pointer

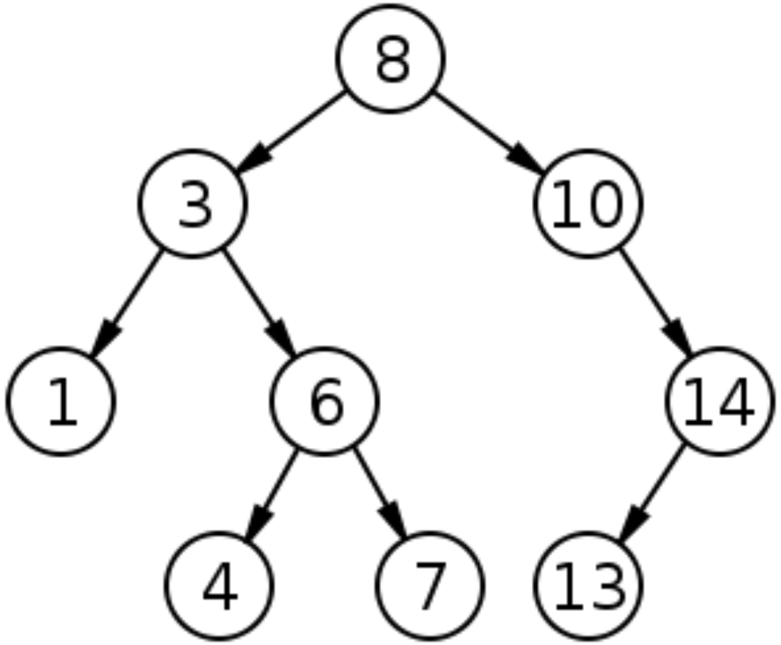


Question

Given a binary search tree that is not necessarily balanced or unbalanced, what is the maximum number of hops needed to search the tree or insert a new node?

Options:

a. 1
b. lg n
c. tree height
d. n



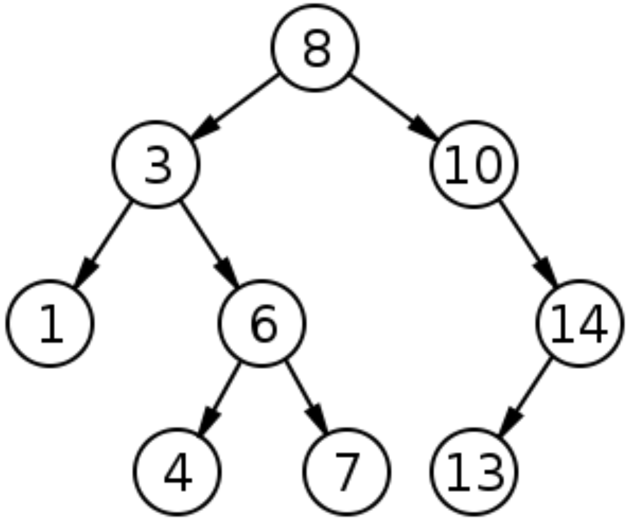
1	3	4	6	7	8	10	13	14

How do you find:

- Min
- Max
- Predecessor (k)
- Successor (k)

Exercise

- What is the running time?



1	3	4	6	7	8	10	13	14

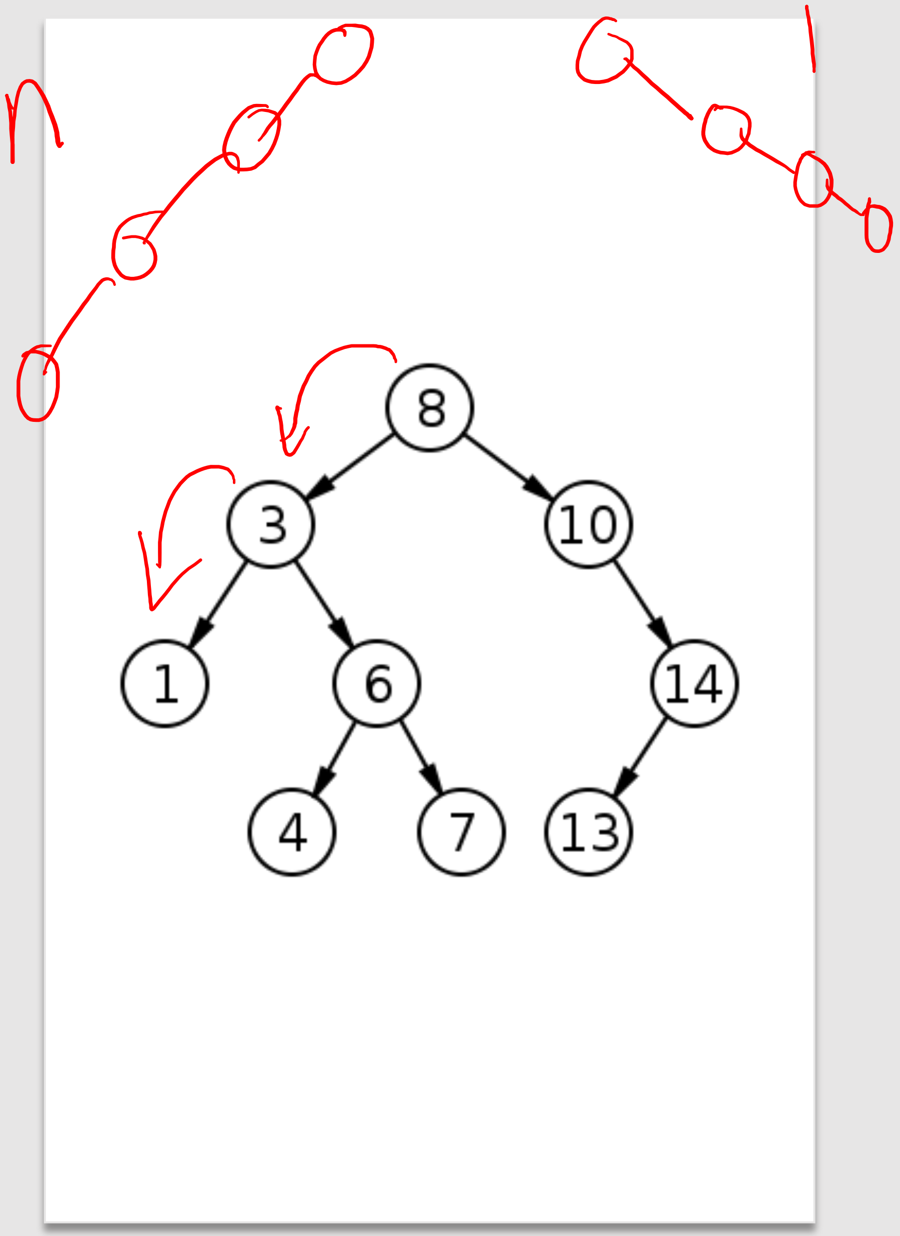
How do you find:

- Min
- Max
- Predecessor (k)
- Successor (k)

Go left

O(tree height)

- What is the running time?



1	3	4	6	7	8	10	13	14

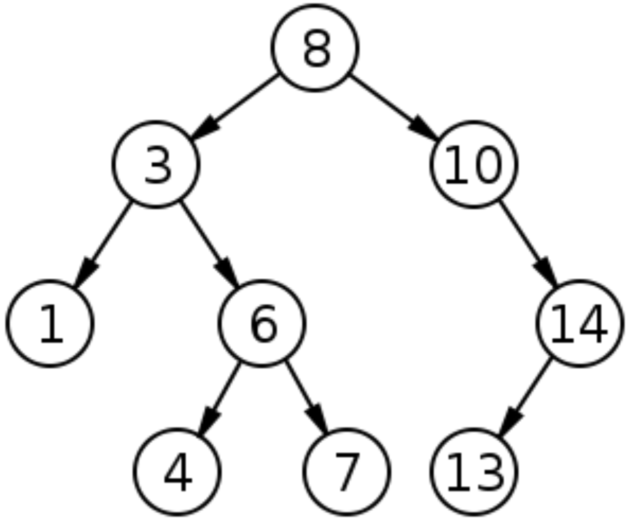
How do you find:

- Min
- Max
- Predecessor (k)
- Successor (k)

Go right

O(tree height)

- What is the running time?



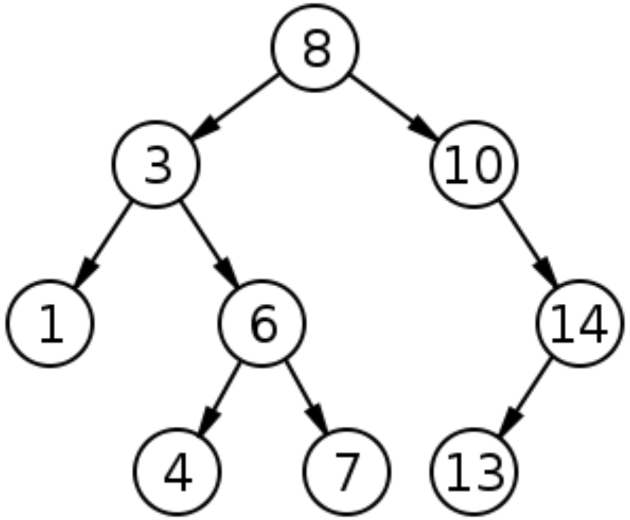
1	3	4	6	7	8	10	13	14

How do you find:

- Min
- Max
- Predecessor (k)
- Successor (k)

Max of left
OR First smaller Ancestor

- What is the running time?

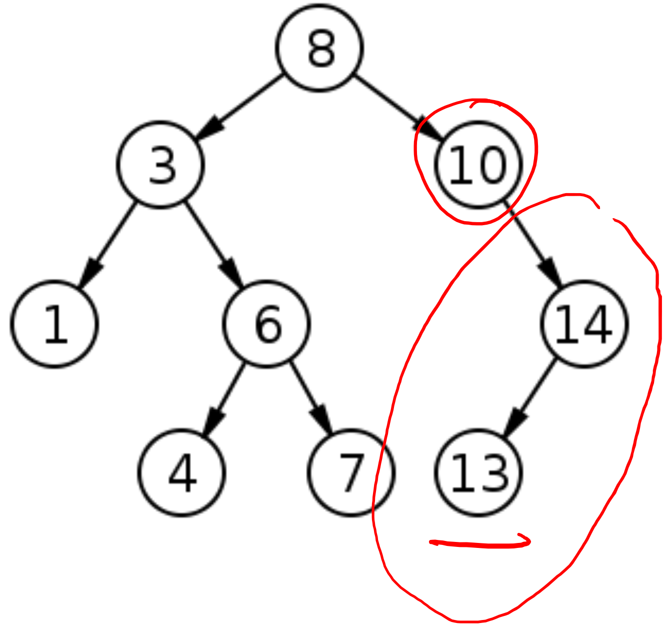


1	3	4	6	7	8	10	13	14

How do you find:

- Min
- Max
- Predecessor (k)
- Successor (k)

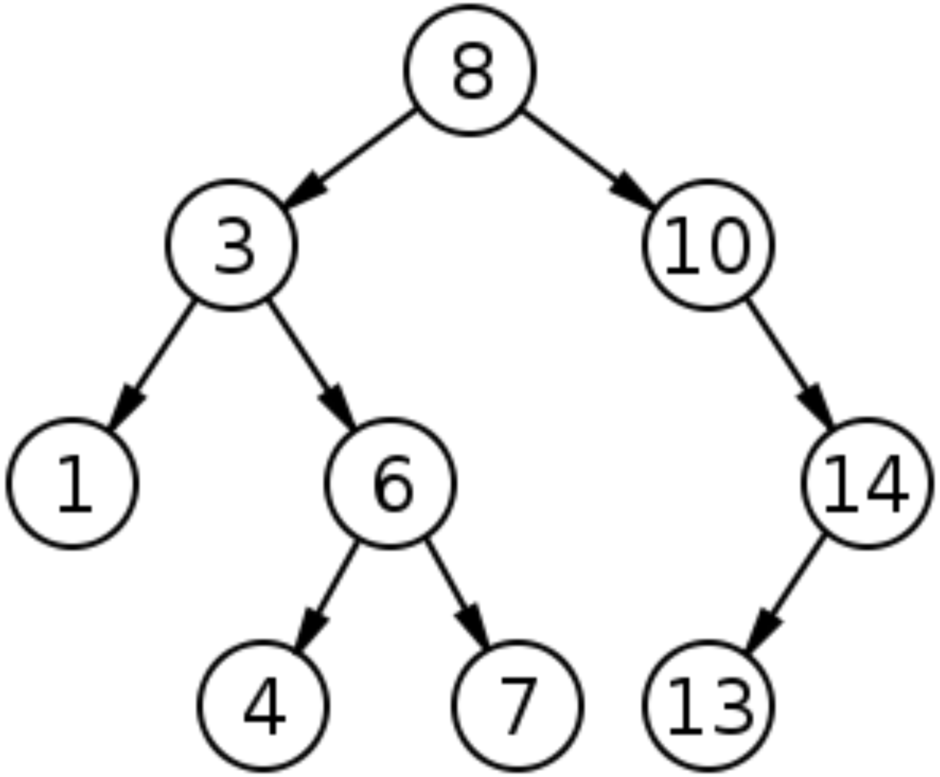
- What is the running time?



How would you print all nodes in order?

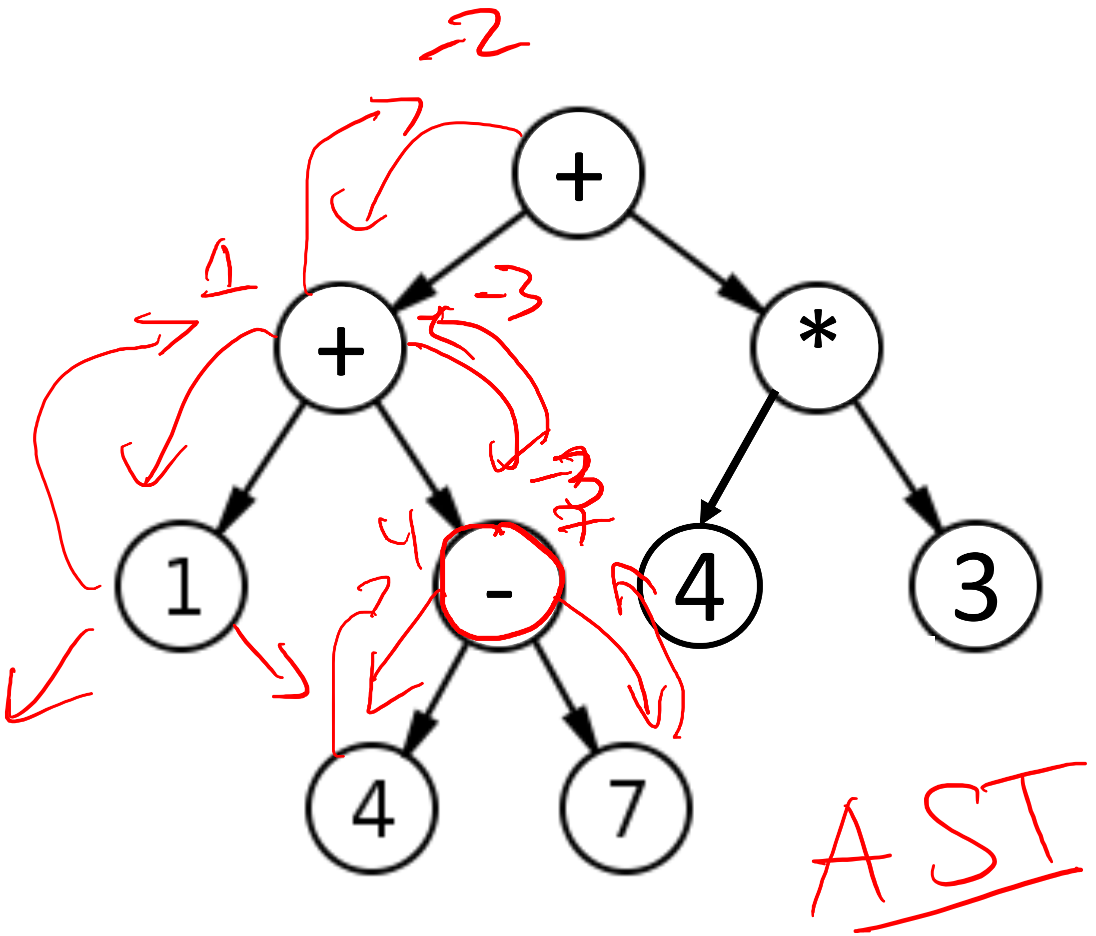
- In-order traversal:
  - Recursively visit nodes on the left
  - Print out the current node
  - Recursively visit nodes on the right

- What is the running time?



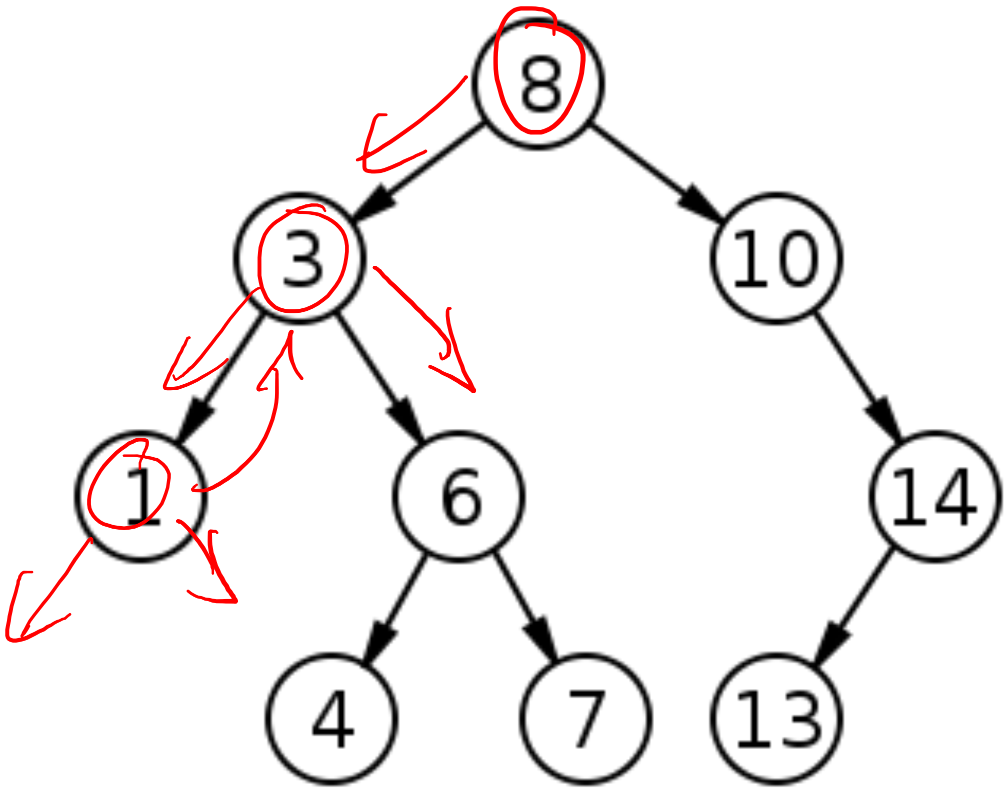
Post-Order Traversal

- Recursively visit nodes on the left
- Recursively visit nodes on the right
- “Visit” the current node



Pre-Order Traversal

- "Visit" the current node
- Recursively visit nodes on the left
- Recursively visit nodes on the right



1. What kind of traversal is this?
2. What is the output?

```html
<!DOCTYPE html>
<html>
<head>
  <title>DOM Walk Demo</title>
<body>
  <header>140</header>
  <main>
    <h1>Hello CSCI 140 PO</h1>
    <ul>
      <li>MergeSort</li>
      <li>Breadth First Search</
      <li>Dijkstra's Algorithm</
      <li>Binary Search Trees</l
      <li>Conquer The World</li>
    </ul>
  </main>
  <footer>Prof. Clark</footer>
```

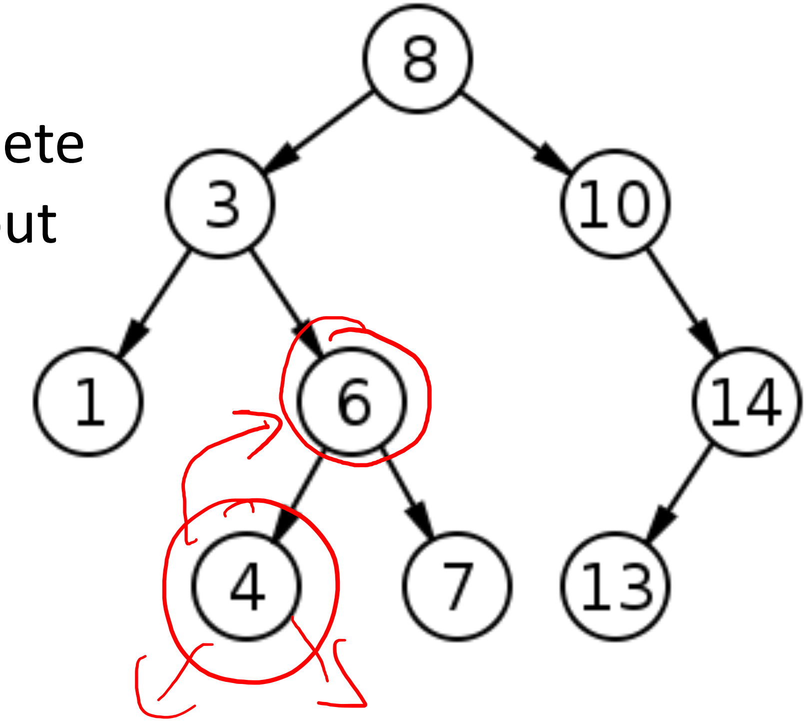
```javascript
var indentLevel = 0;
var walk_the_DOM = function walk(node, func) {
    func(node);
    indentLevel++;
    node = node.firstChild;
    while (node) {
        if (node.nodeName !== "#text") {
            walk(node, func);
        }
        node = node.nextSibling;
    }
    indentLevel--;
}

walk_the_DOM(document.body, function (node) {
    console.log("  ".repeat(indentLevel) + node.nodeName);
});
```

```
BODY
  HEADER
  MAIN
    H1
    UL
      LI
      LI
      LI
      LI
      LI
  FOOTER
```

Deleting a node from a BST

Deletion is often the most difficult task for tree-like structures

- Search for the key
  - Case 1: If the node has no children then just delete
  - Case 2: If the node has one child then splice it out
  - Case 3: if the node has both children
    - Find the node's predecessor
    - Swap the node with its predecessor
    - Delete the node



Selection and Rank with a BST

How would you compute the i^{th} order statistic using a BST?

Idea: store some metadata at each node

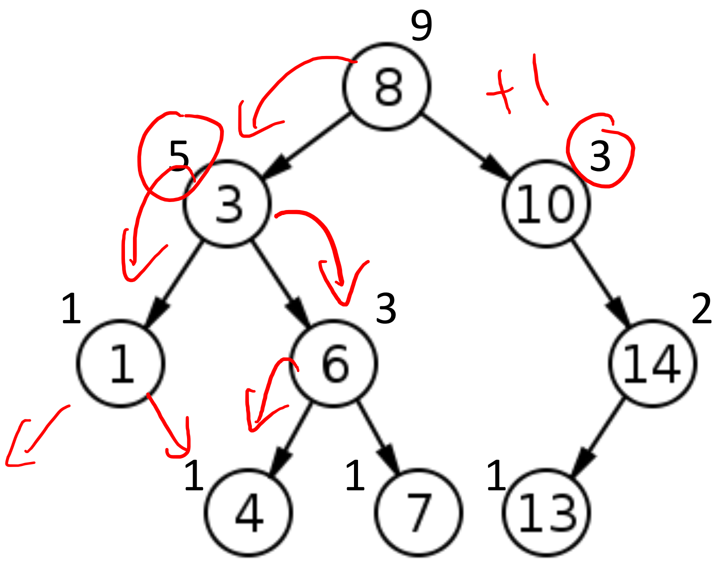
- Let `size(x)` = the number of nodes rooted at x (the number of nodes that can be reached via the left and right children pointers

How would you calculate `size(x)`?

- What kind of traversal would this use (in order, pre, or post)?
- `size(x) = size(left) + size(right) + 1`

null null

0 0



```
FUNCTION UpdateSizes(bst_node)
    IF bst_node != NONE
        UpdateSizes(bst_node.left)
        UpdateSizes(bst_node.right)

        bst_node.size = bst_node.left.size
                      + bst_node.right.size
                      + 1

    ELSE
        RETURN 0
```

Post Order

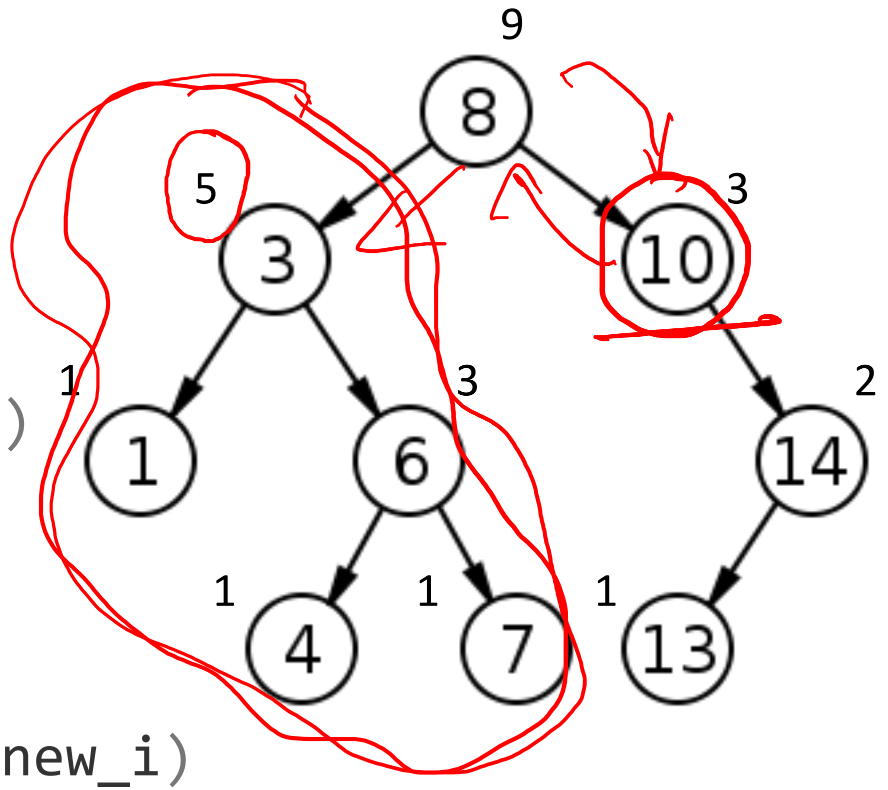
7th

Selection and Rank with a BST

```
FUNCTION GetIthOrderStatistic(bst_node, i)
    left_child_size = bst_node.left.size
```

5 7 - 1 = 6

```
    IF left_child_size == (i - 1)
        RETURN bst_node.value

    ELSE IF left_child_size ≥ i
        RETURN GetIthOrderStatistic(bst_node.left, i)

    ELSE
        new_i = i - left_child_size - 1
        RETURN GetIthOrderStatistic(bst_node.right, new_i)
```



Balanced Binary Search Trees

All ops depend on the height of the tree

- Why is balancing important?
- What is the worst-case height for a binary tree?
- <u>Balanced tree: the height of a balanced tree stays O(lg n) after insertions and deletions</u>
- Many different types of balanced search trees:
  - AVL Tree, Splay Tree, B Tree, Red-Black Tree