Binary Search Trees

Part One

Taking Stock: Where Are We?

- ☐ Stack
- ☐ Queue
- ☐ Vector
- ☐ string
- ☐ PriorityQueue
- ☐ Map
- ☐ Set
- ☐ Lexicon

- [x] Stack
- [ ] Queue
- [ ] Vector
- [ ] string
- [ ] PriorityQueue
- [ ] Map
- [ ] Set
- [ ] Lexicon

✓ Stack
✓ Queue
□ Vector
□ string
□ PriorityQueue
□ Map
□ Set
□ Lexicon

- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- ☐ PriorityQueue
- ☐ Map
- ☐ Set
- ☐ Lexicon

- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- ✓ PriorityQueue
- ☐ Map
- ☐ Set
- ☐ Lexicon

- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- ✓ PriorityQueue
- ☐ **Map**
- ☐ **Set**
- ☐ **Lexicon**

Implementing `Map` and `Set`

An Inefficient Implementation

- We could implement the `Set` as an unsorted list of all the values it contains.
- To add an element:
  - Check if the element already exists.
  - If not, append it.

  } **O(*n*)**
- To remove an element:
  - Find and remove it from the list.

  } **O(*n*)**
- To see if an element exists:
  - Search the list for the element.

  } **O(*n*)**

An Inefficient Implementation

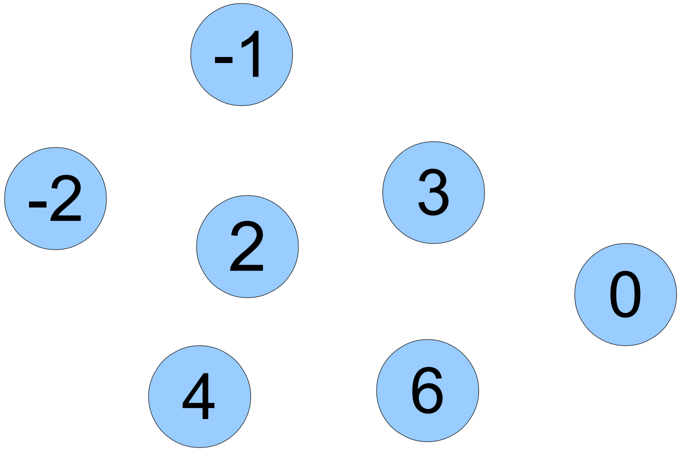
- We could implement the `Set` as a sorted list of all the values it contains.
- To add an element:
  - Check if the element already exists.
  - If not, insert it in the right spot.

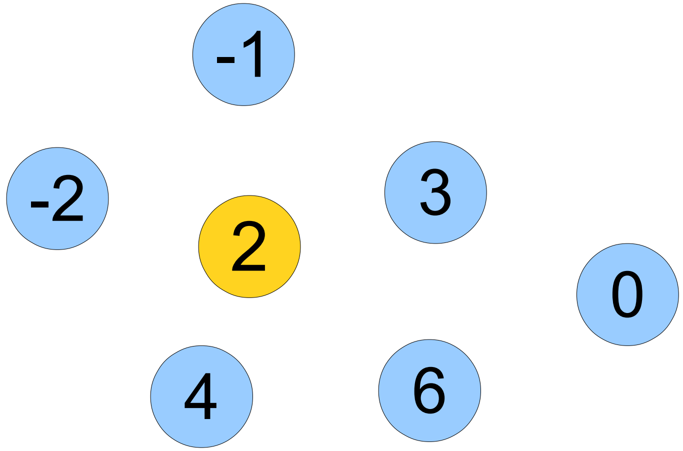
  O(*n*)
- To remove an element:
  - Find and remove it from the list.

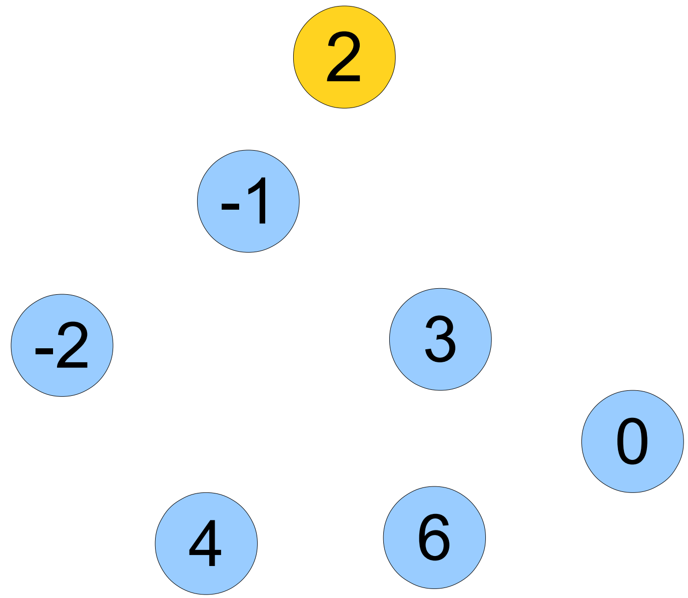
  O(*n*)
- To see if an element exists:
  - Search the list for the element.

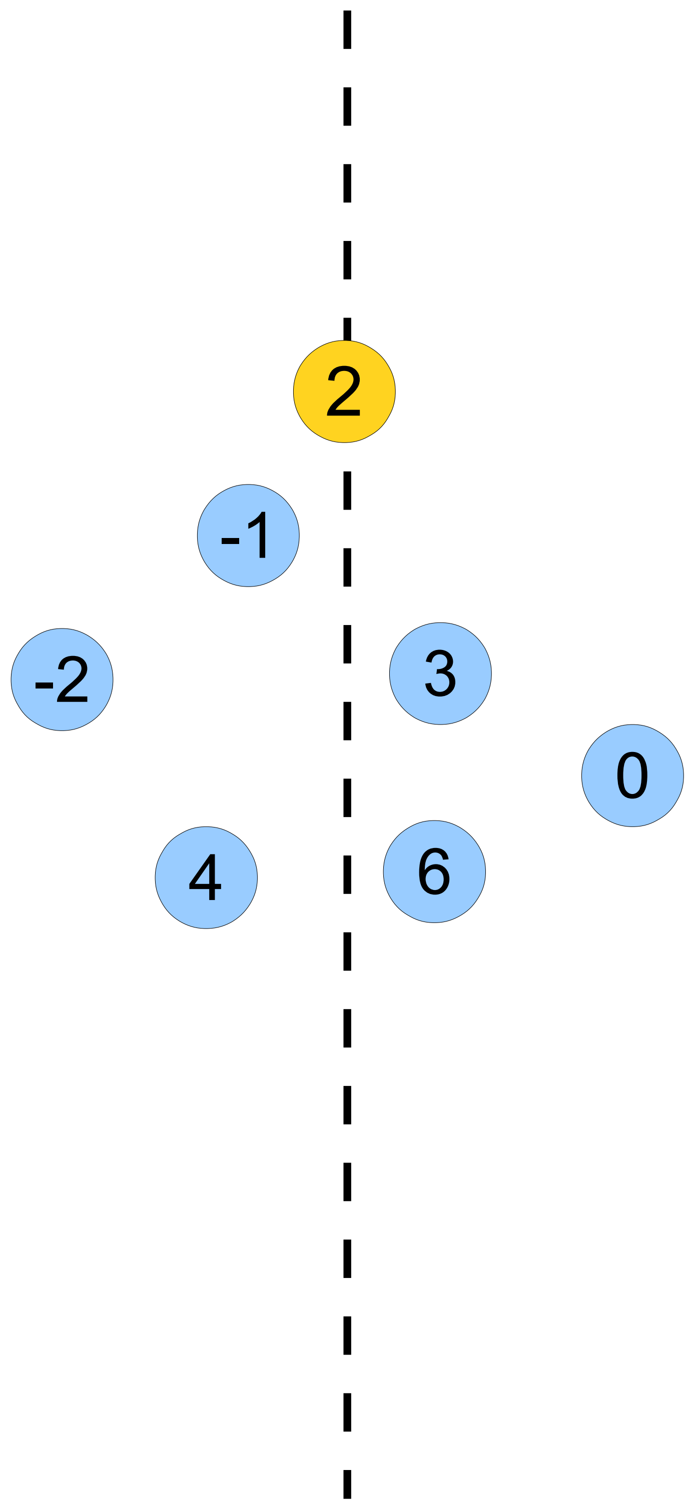
  O(log *n*)

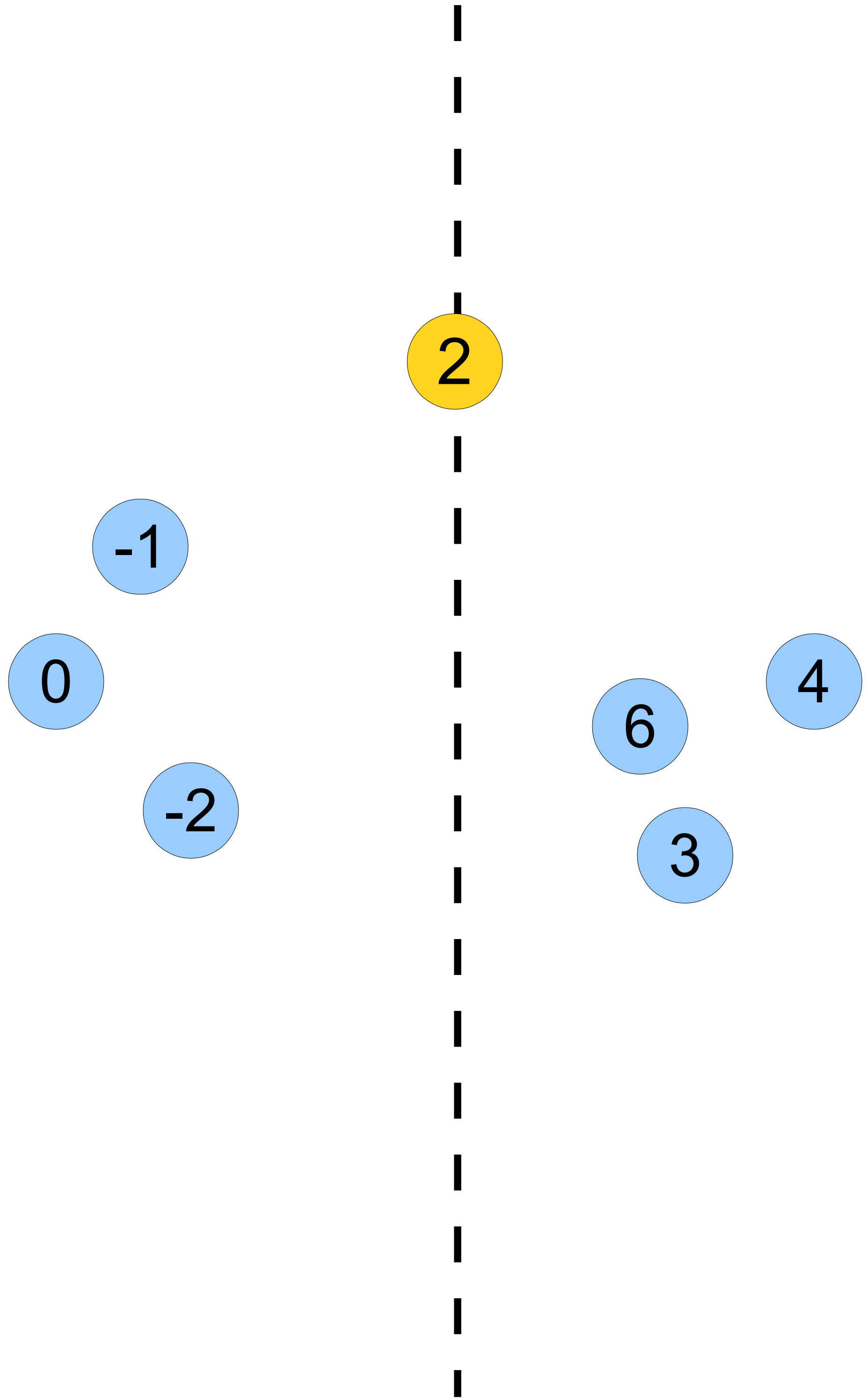
An Entirely Different Approach

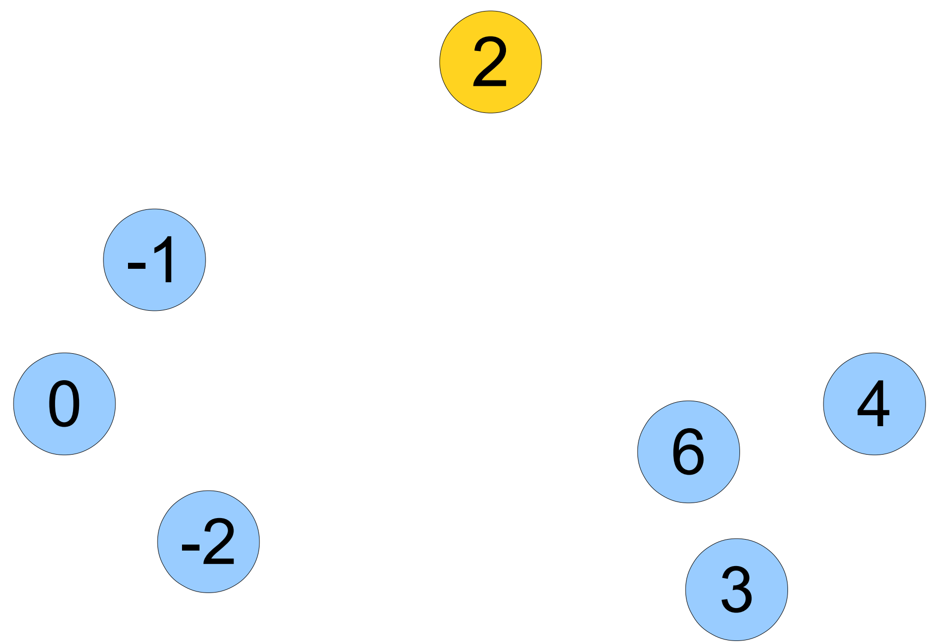

-1
-2
3
2
0
4
6

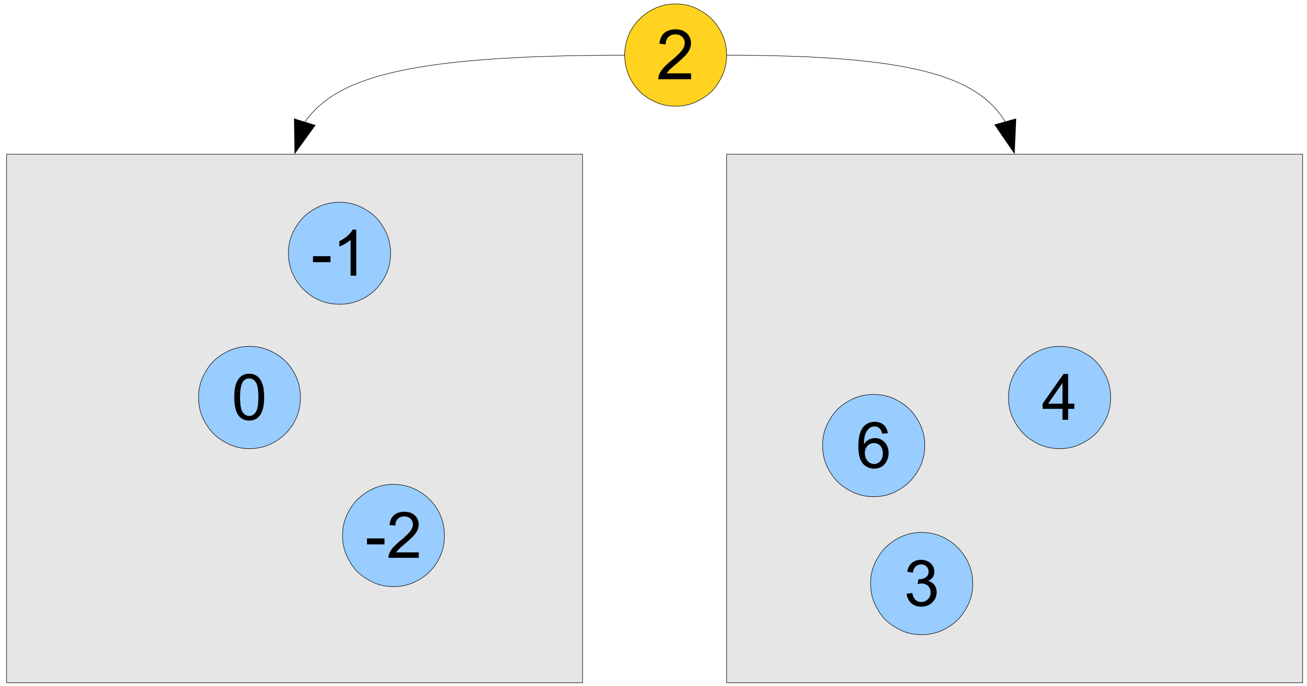

-1
-2
3
2
0
4
6


2
-1
-2
3
0
4
6

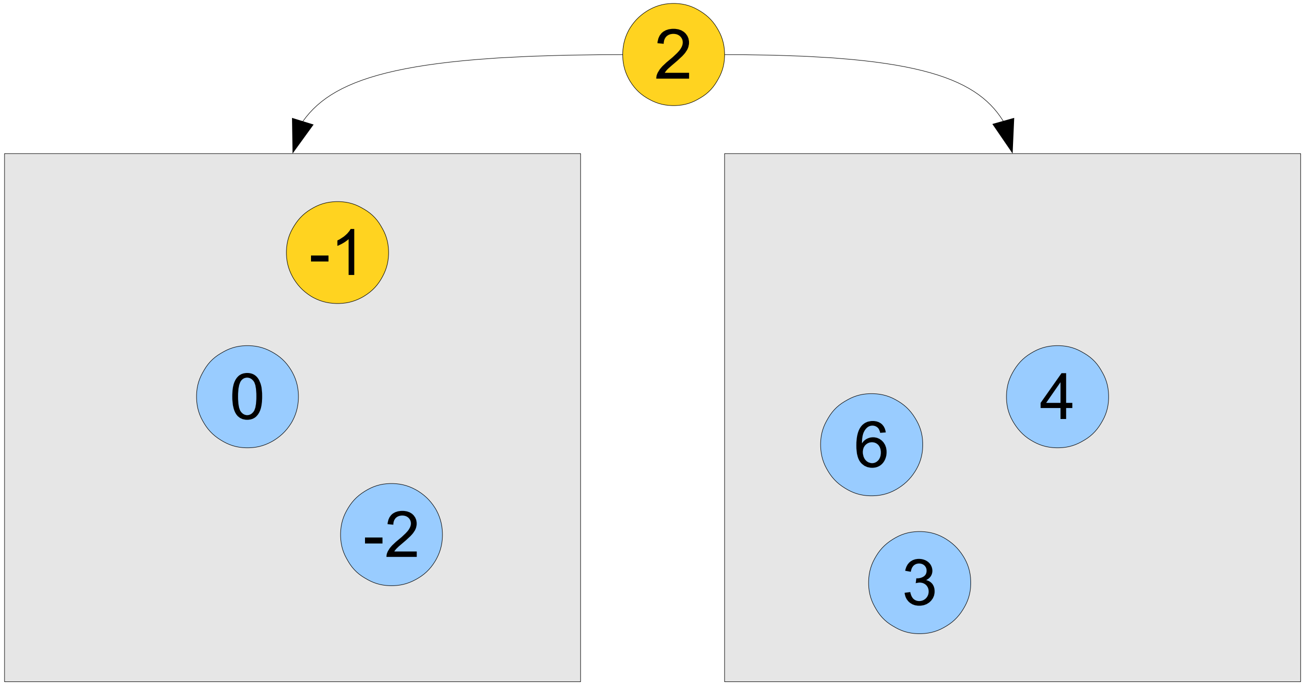

2
-1
-2
3
0
4
6

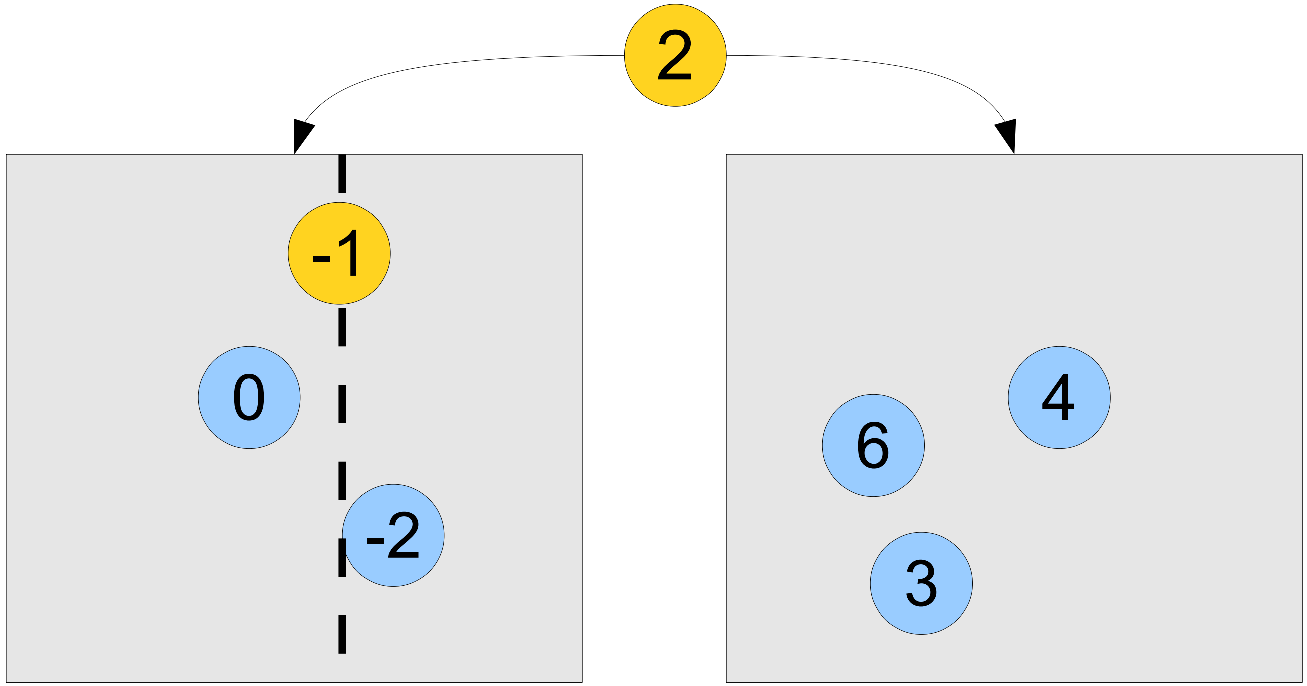

2
-1
0
-2
6
4
3

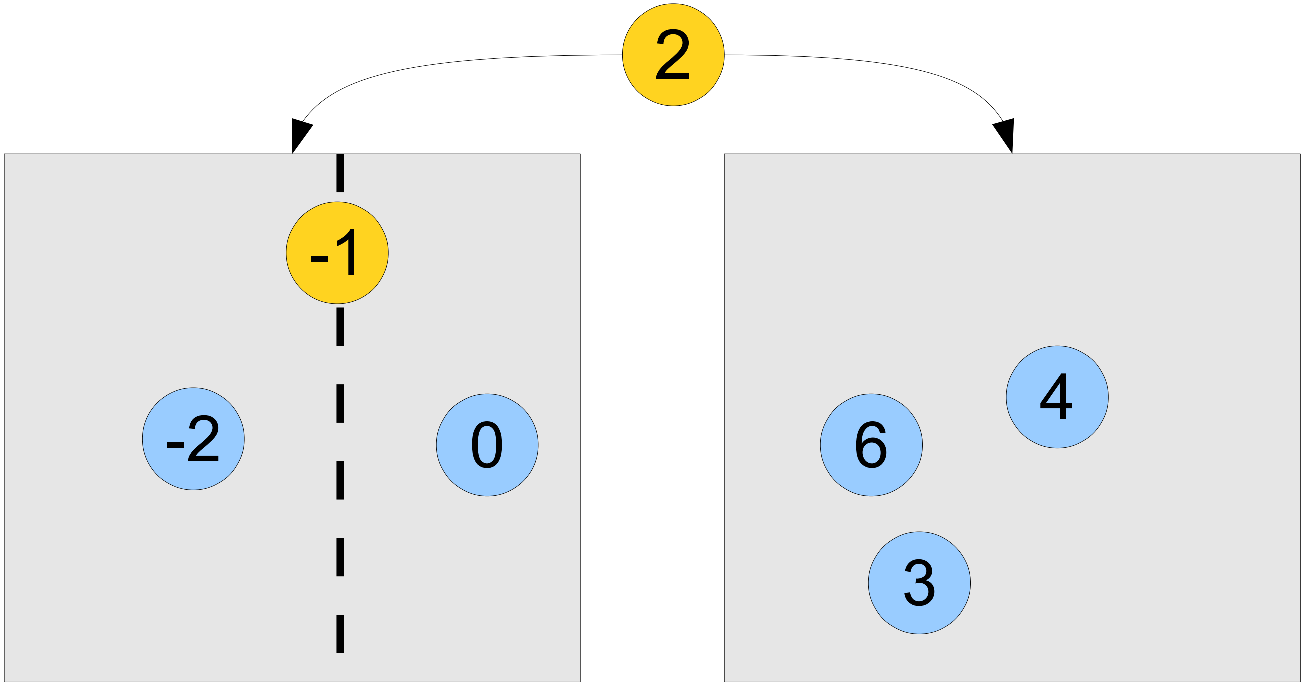

2
-1
0
6
4
-2
3

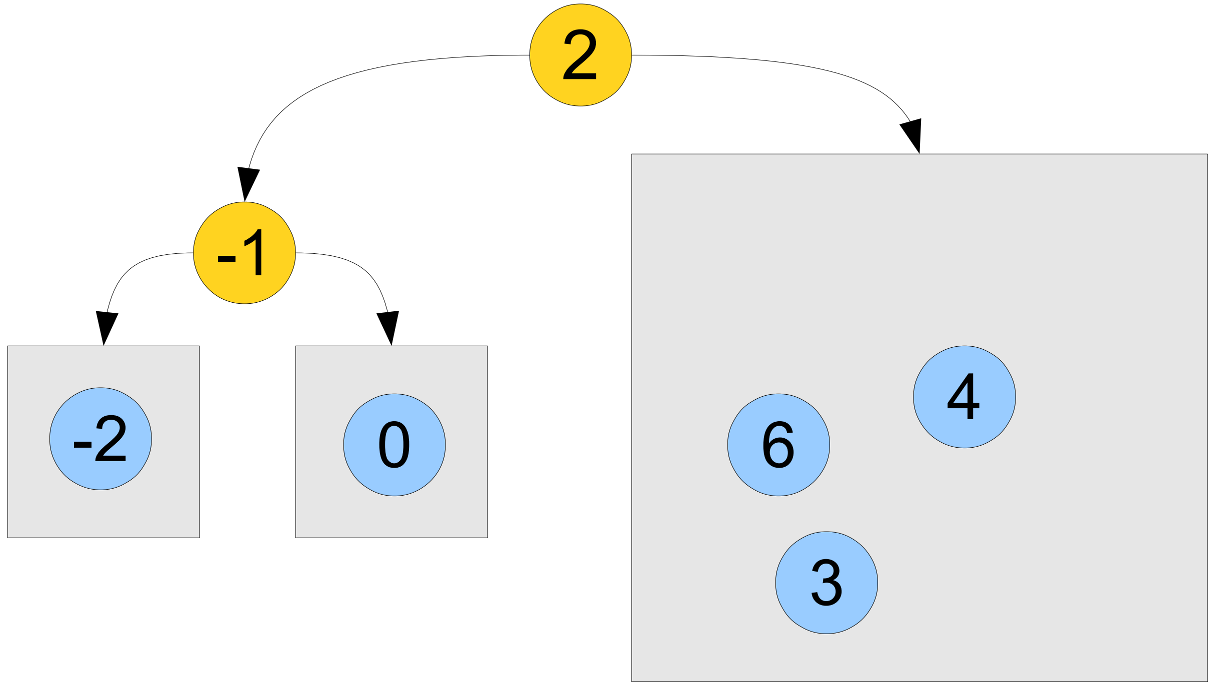


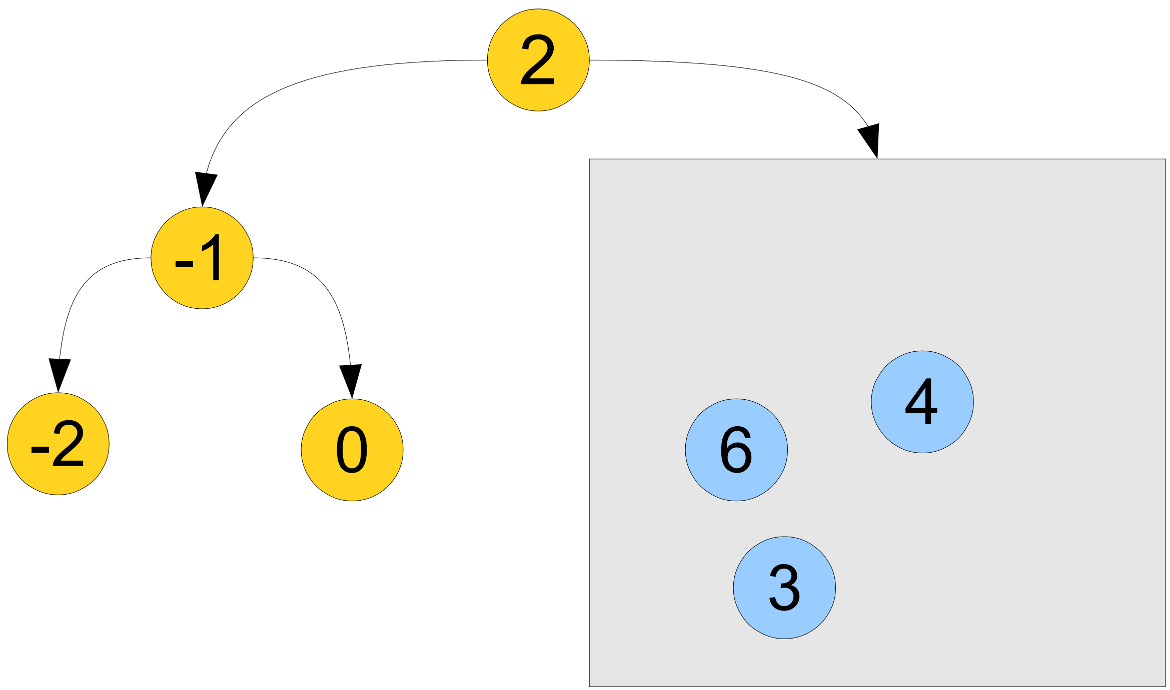
2
-1
0
-2
4
6
3

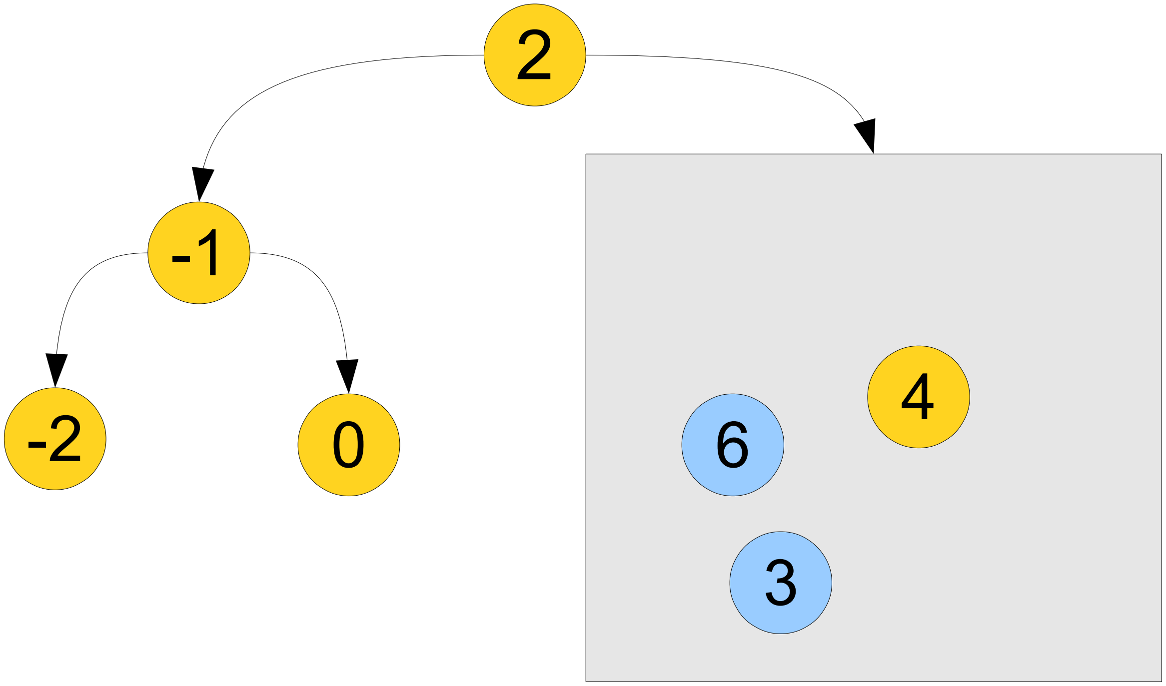

2
-1
0
-2
4
6
3

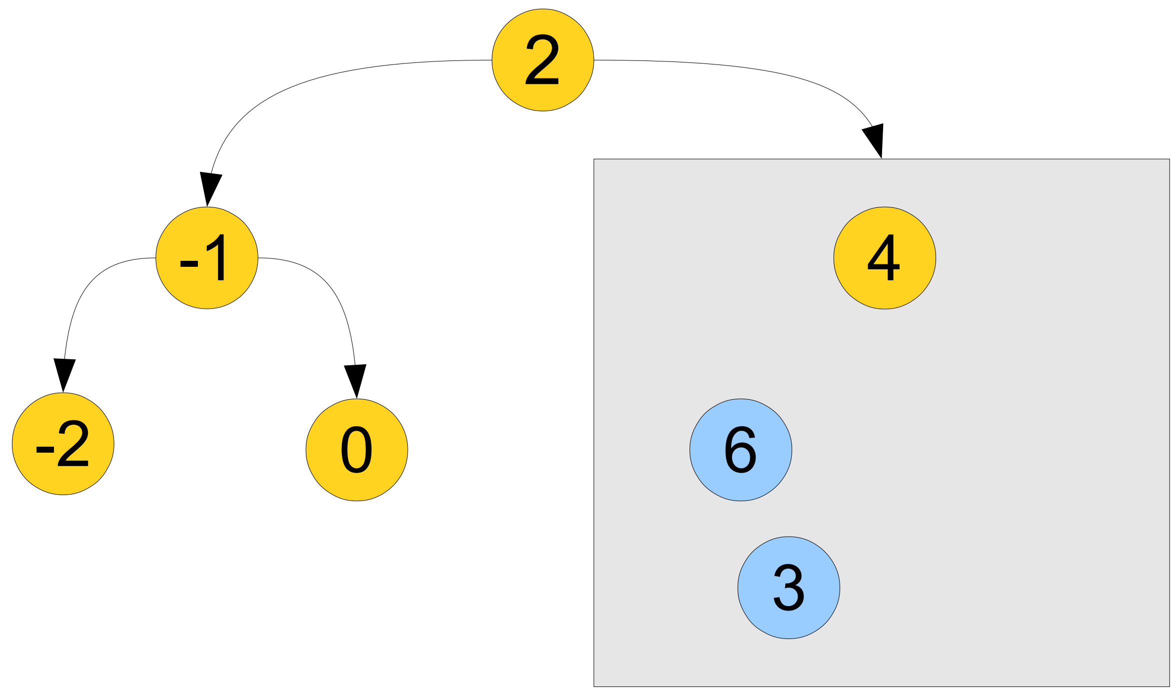

2
-1
0
-2
6
4
3


2
-1
-2
0
4
6
3

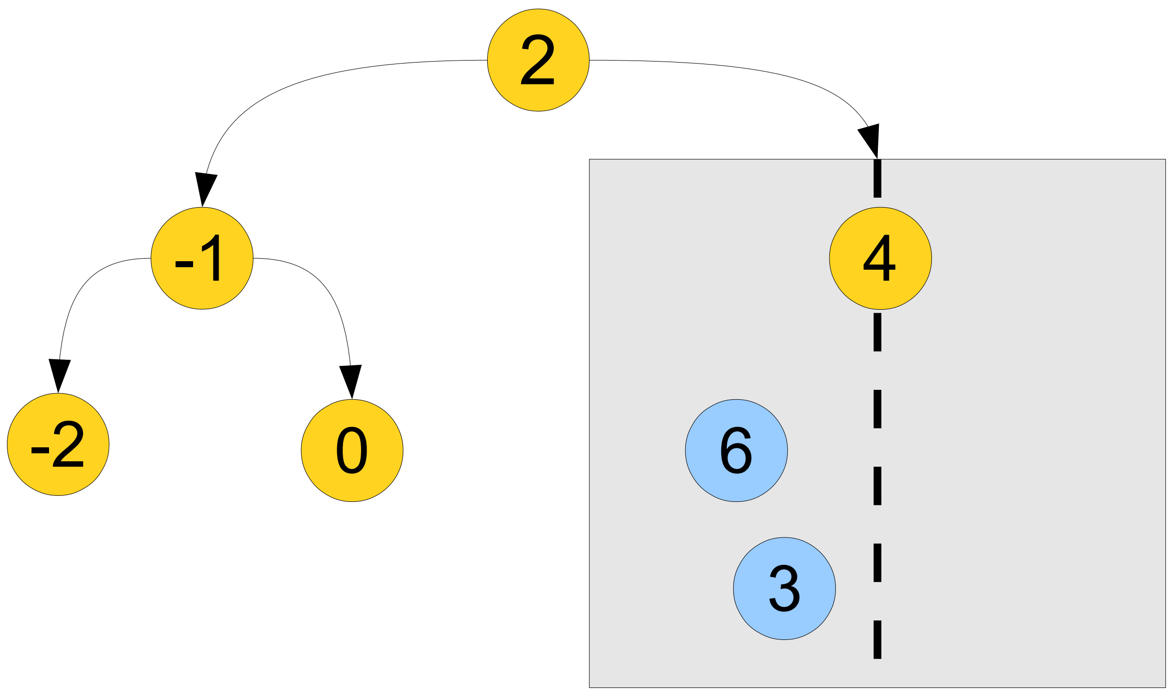

2
-1
-2
0
4
6
3

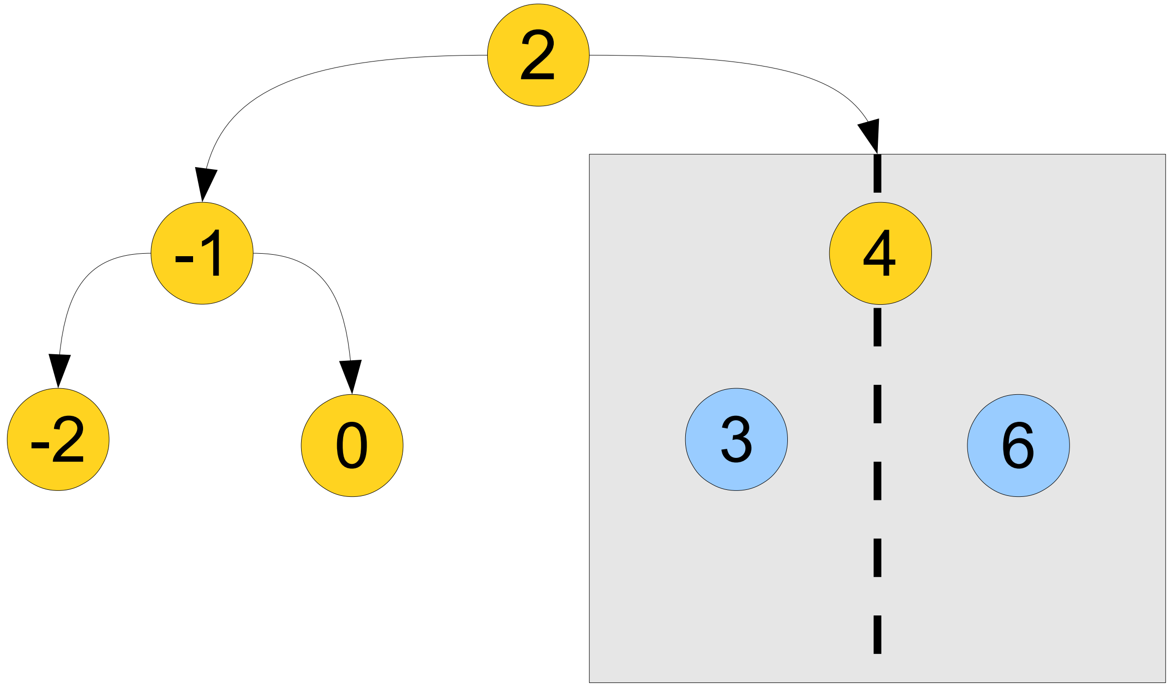

2
-1
-2
0
6
4
3

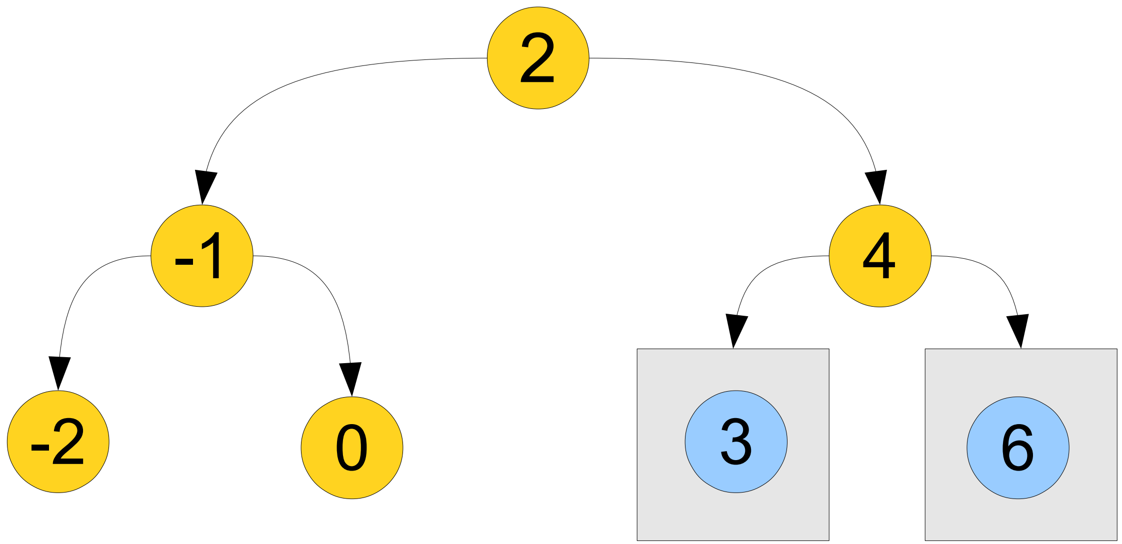

2
-1
-2
0
4
6
3

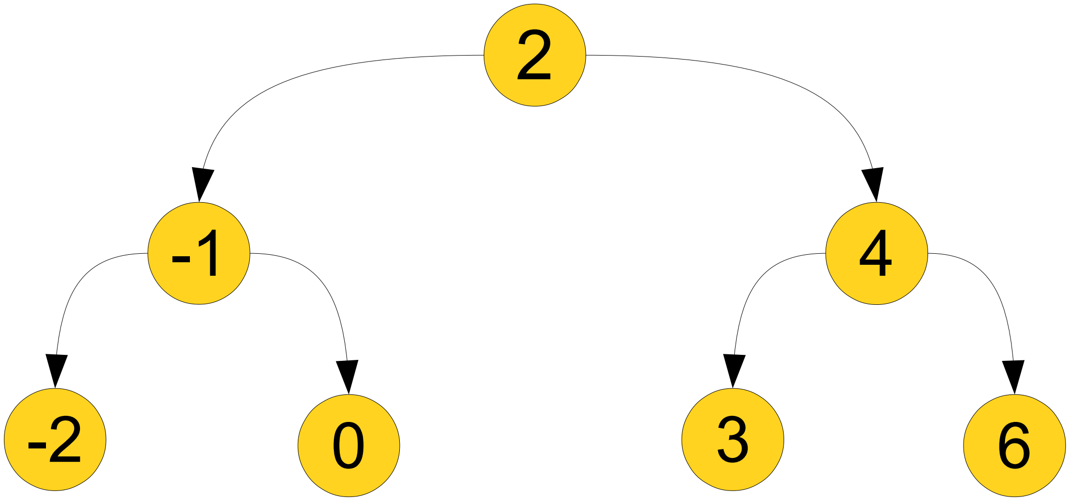


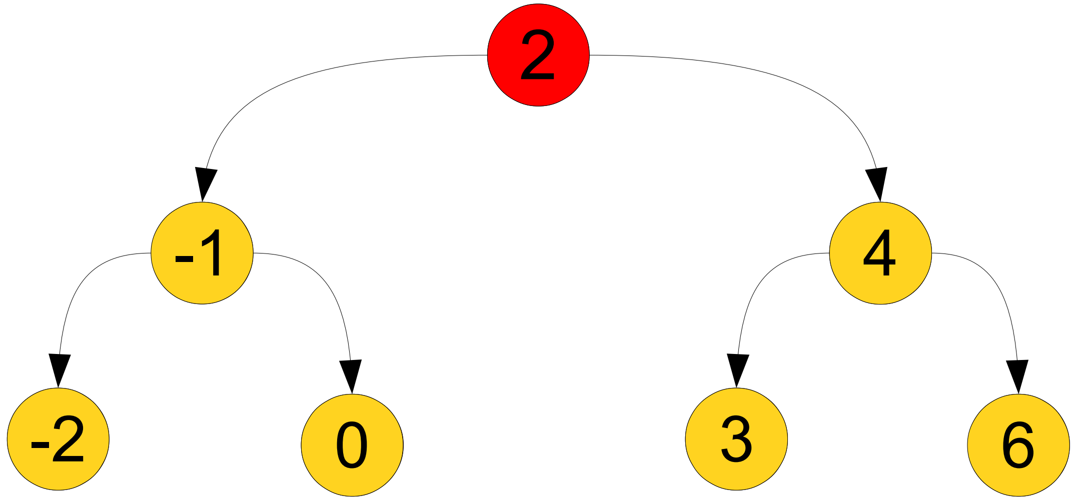
2
-1
4
-2
0
6
3

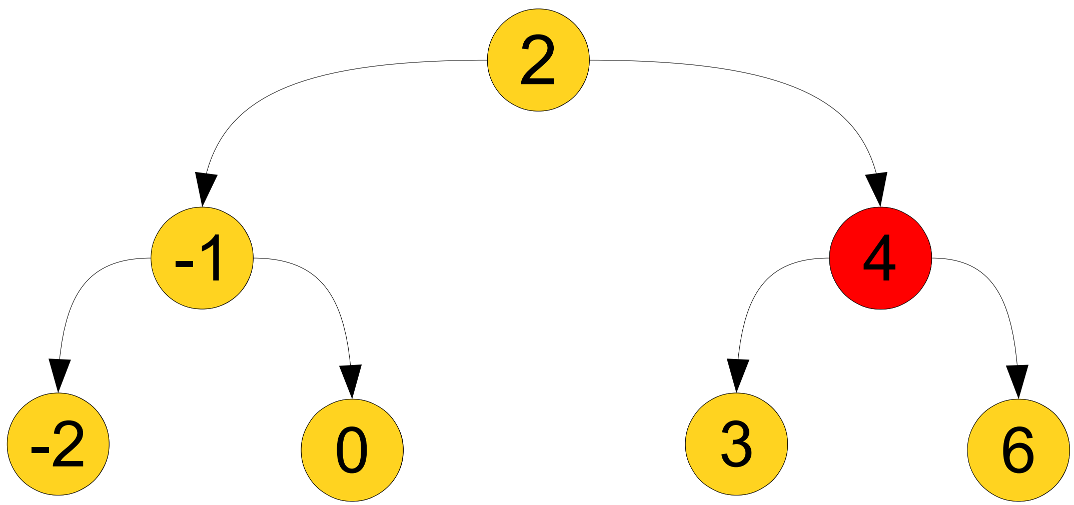

2
-1
4
-2
0
6
3

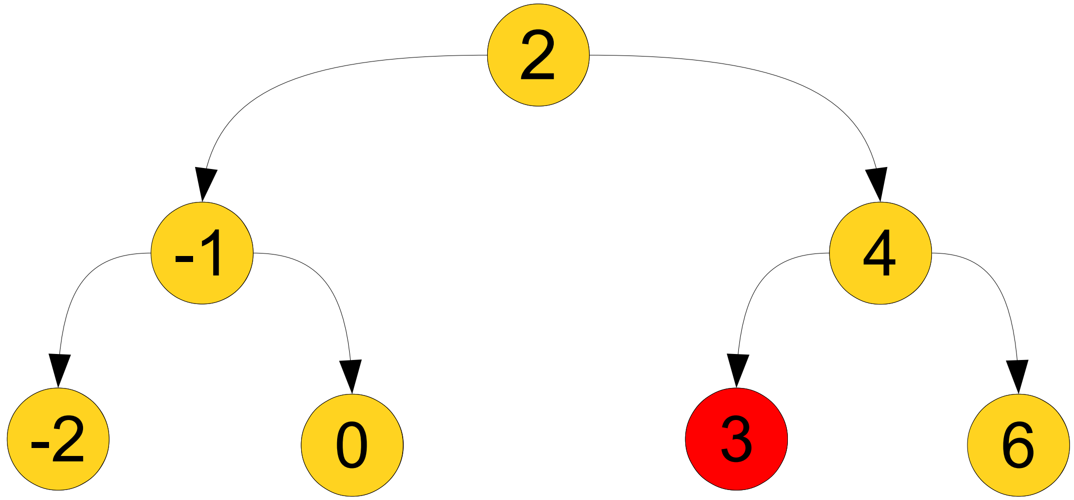

2
-1
4
-2
0
3
6

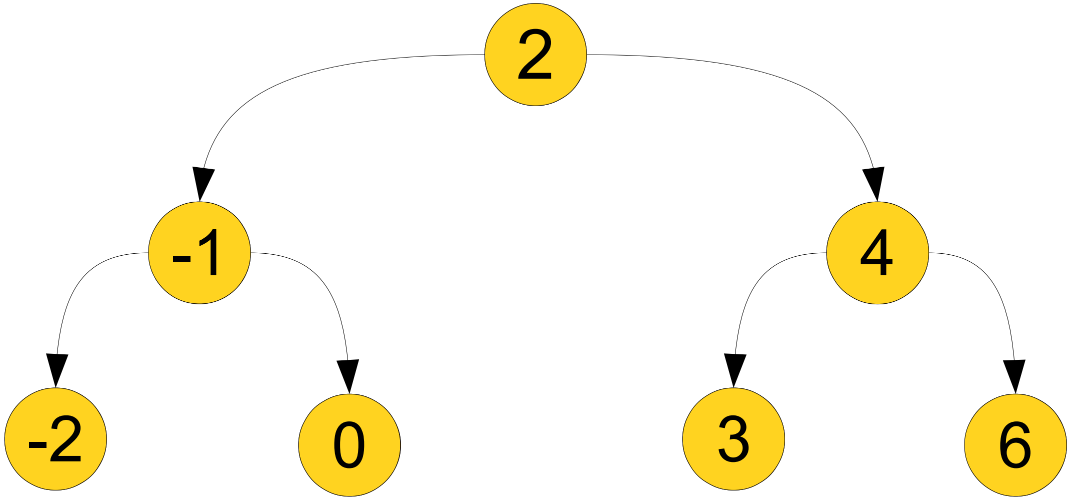

2
-1
4
-2
0
3
6

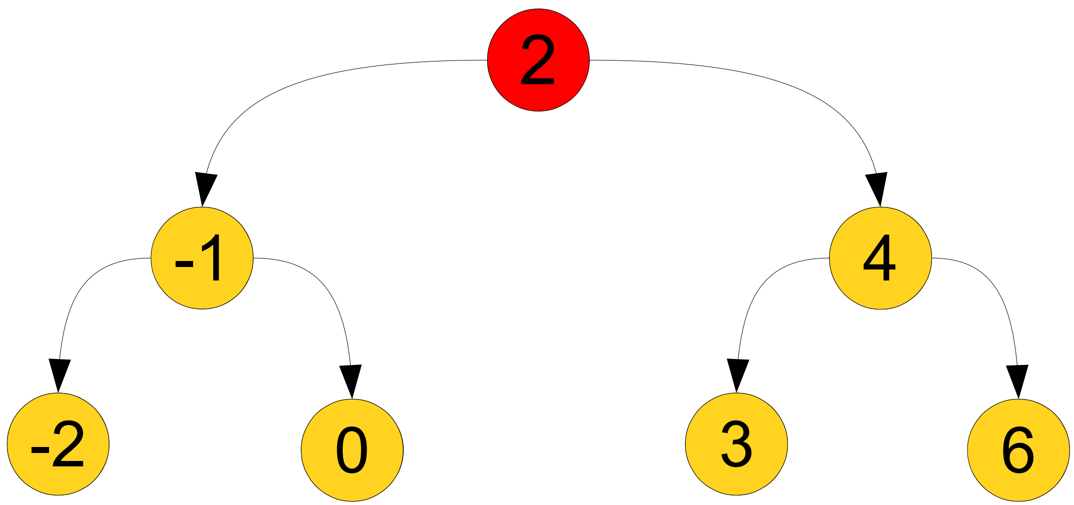

2
-1
4
-2
0
3
6

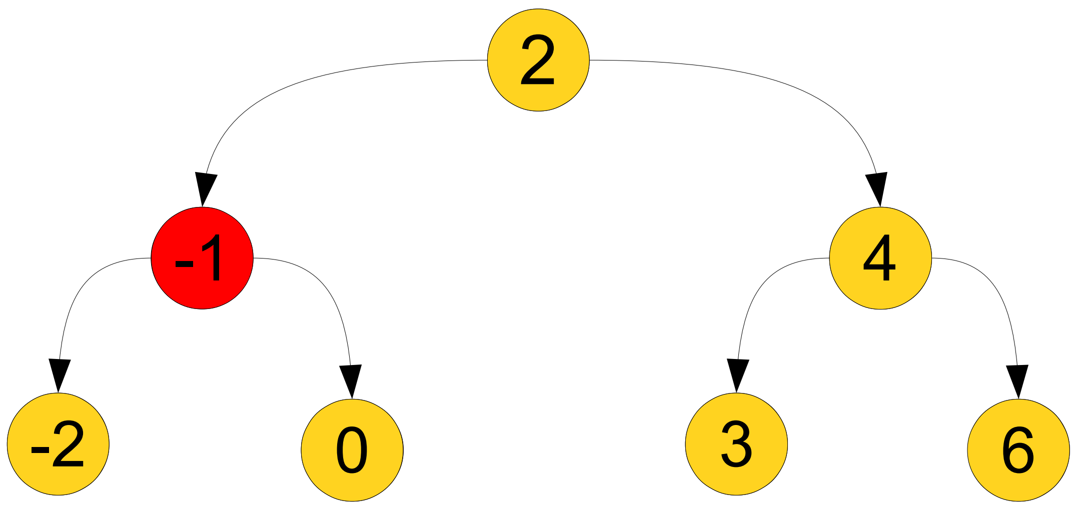

2
-1
4
-2
0
3
6

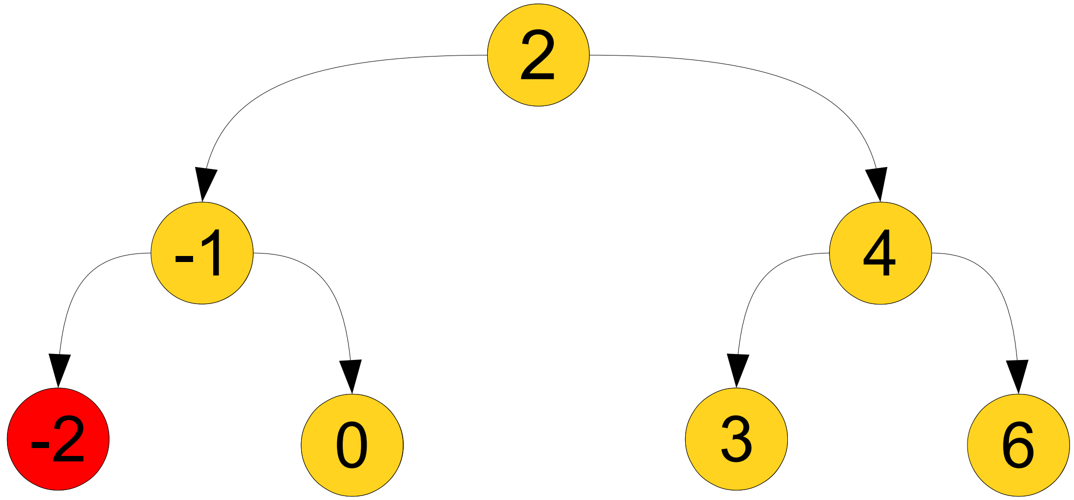

2
-1
4
-2
0
3
6

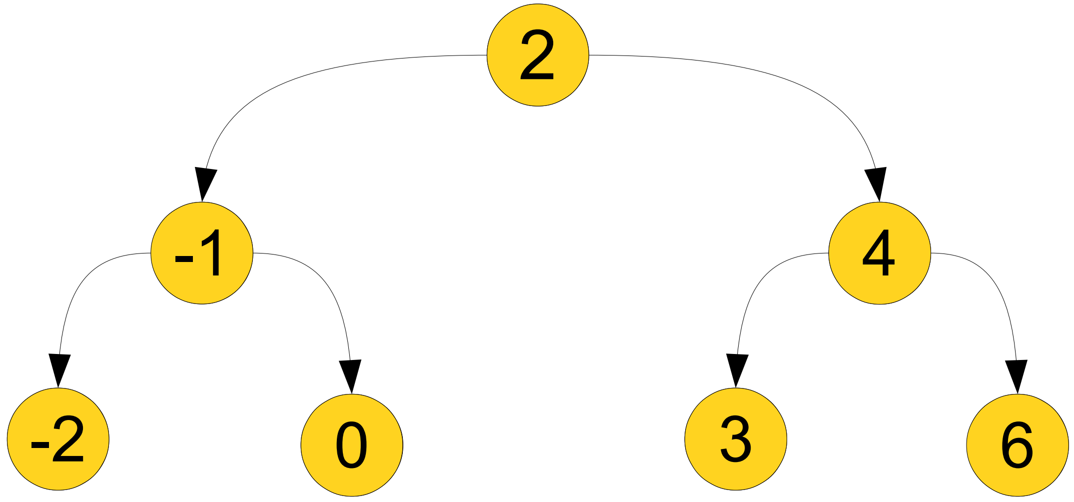

2
-1
4
-2
0
3
6


2
-1
4
-2
0
3
6


2
-1
4
-2
0
3
6

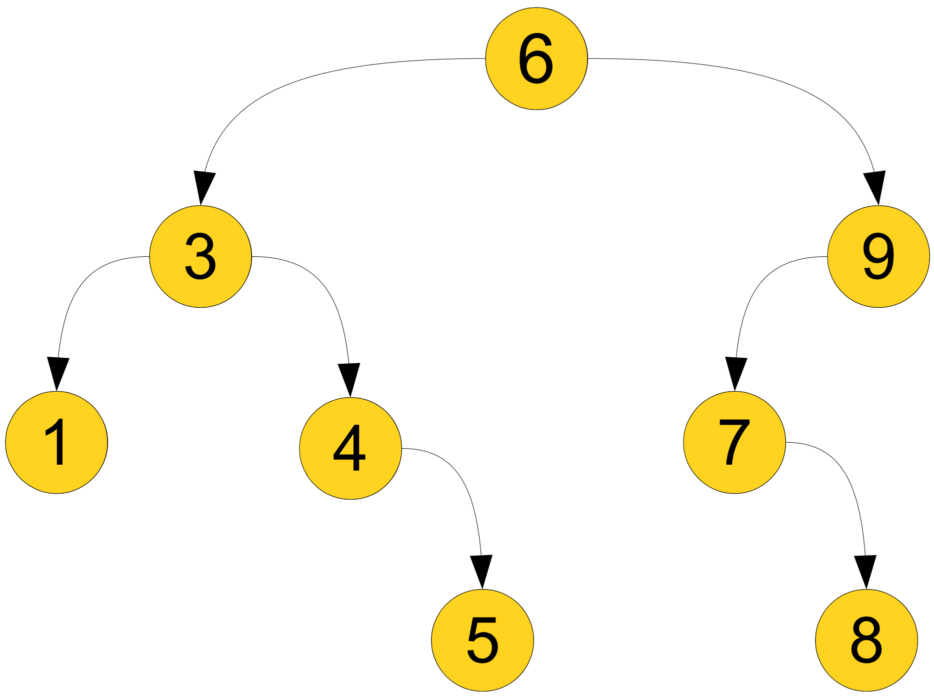

2
-1
4
-2
0
3
6


2
-1
4
-2
0
3
6


2
-1
4
-2
0
3
6

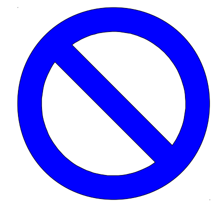
Binary Search Trees

- The data structure we have just seen is called a ***binary search tree*** (or ***BST***).
- The tree consists of a number of ***nodes***, each of which stores a value and has zero, one, or two ***children***.
- ***Key structural property:*** All values in a node's left subtree are ***smaller*** than the node's value, and all values in a node's right subtree are ***greater*** than the node's value.

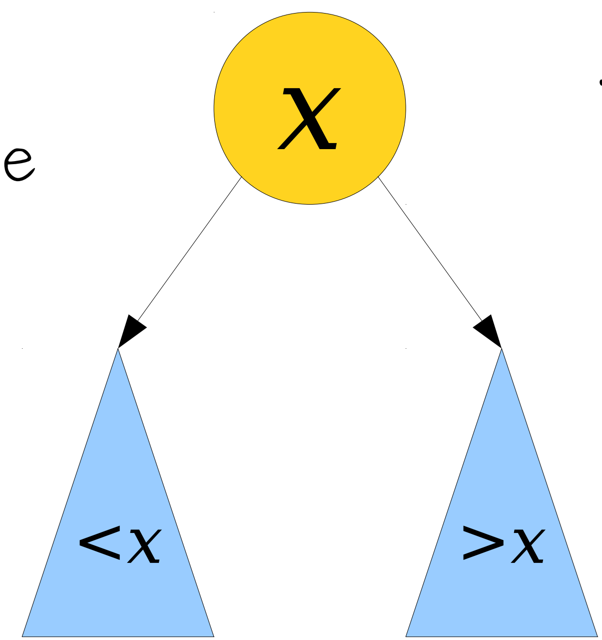


A Binary Search Tree Is Either…

an empty tree, represented by **nullptr**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.

Binary Search Tree Nodes

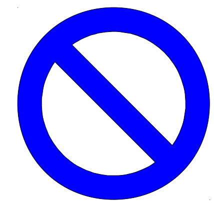
```
struct Node {
    Type value;
    Node* left;  // Smaller values
    Node* right; // Bigger values
};
```

Kinda like a linked list, but with two pointers instead of just one!

Operation 1: Searching a BST

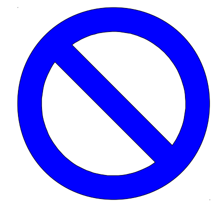
A Binary Search Tree Is Either…

an empty tree,
represented by
`nullptr`



A Binary Search Tree Is Either…

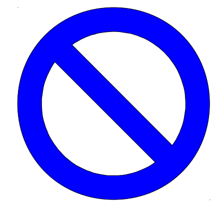
an empty tree, represented by `nullptr`



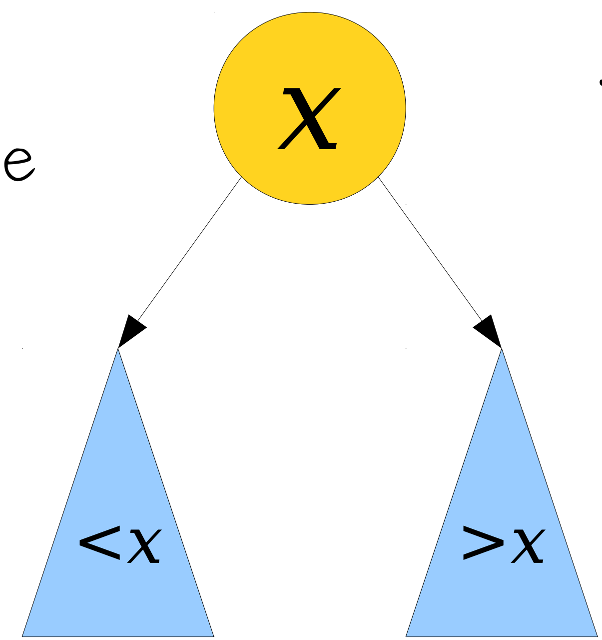
If you're looking for something in an empty BST, it's not there! Sorry.

A Binary Search Tree Is Either…

an empty tree, represented by `nullptr`, or…



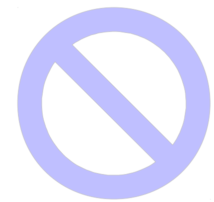
… a single node, whose left subtree is a BST of smaller values …



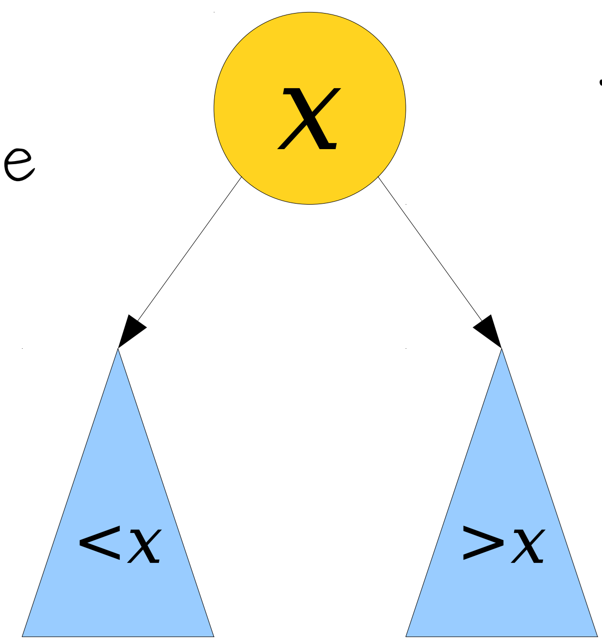
… and whose right subtree is a BST of larger values.

A Binary Search Tree Is Either…

an empty tree, represented by `nullptr`, or…



… a single node, whose left subtree is a BST of smaller values …



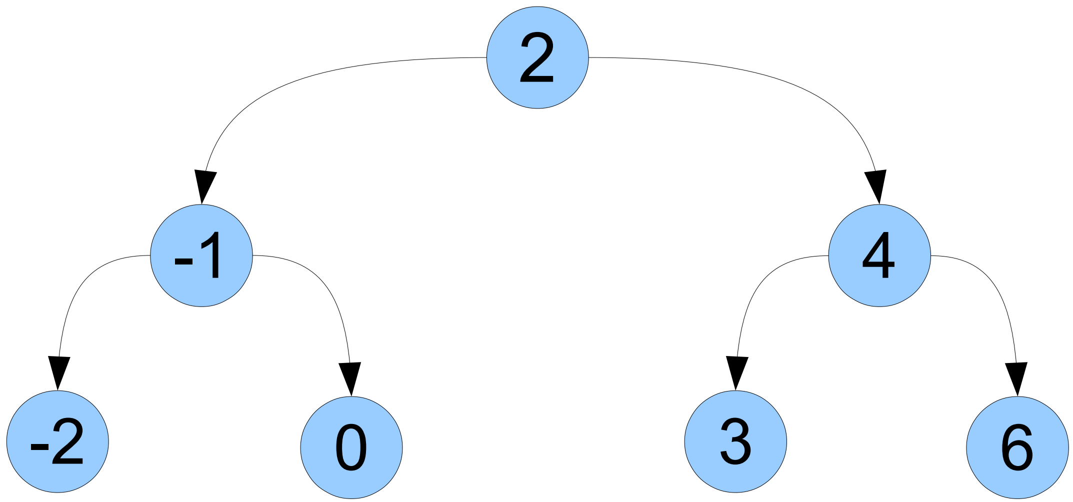
… and whose right subtree is a BST of larger values.

Good exercise:

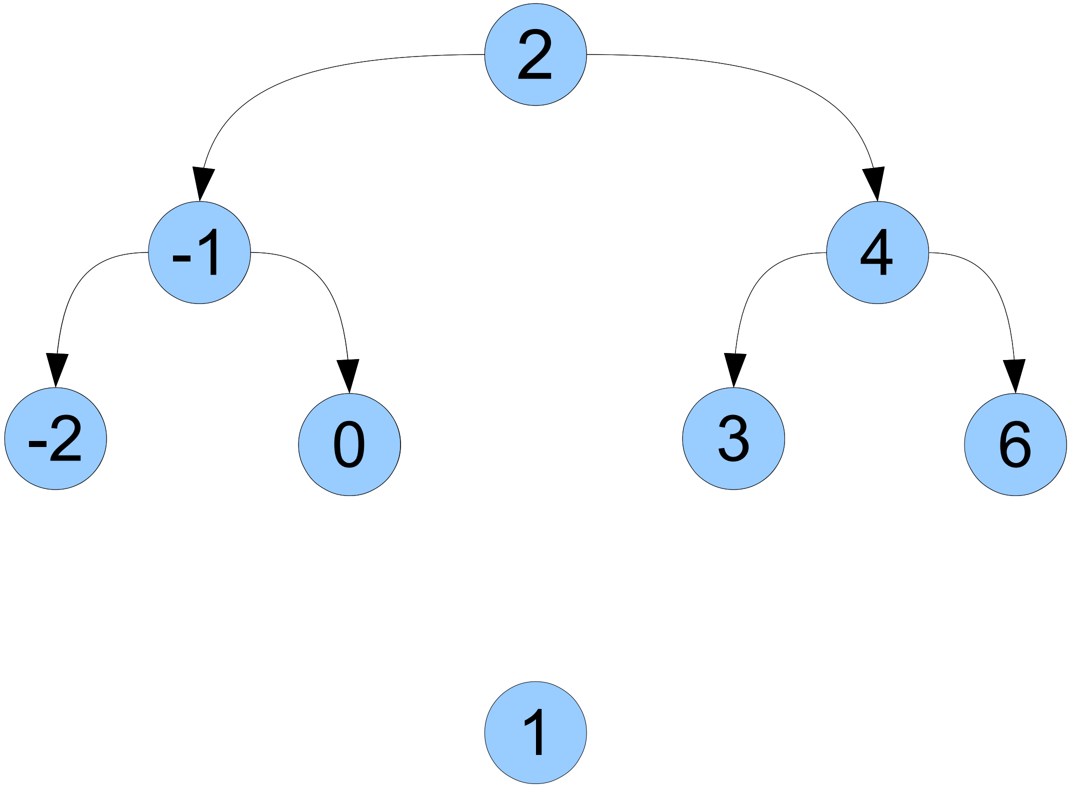
Rewrite this function iteratively!

Operation 2: Inserting into a BST

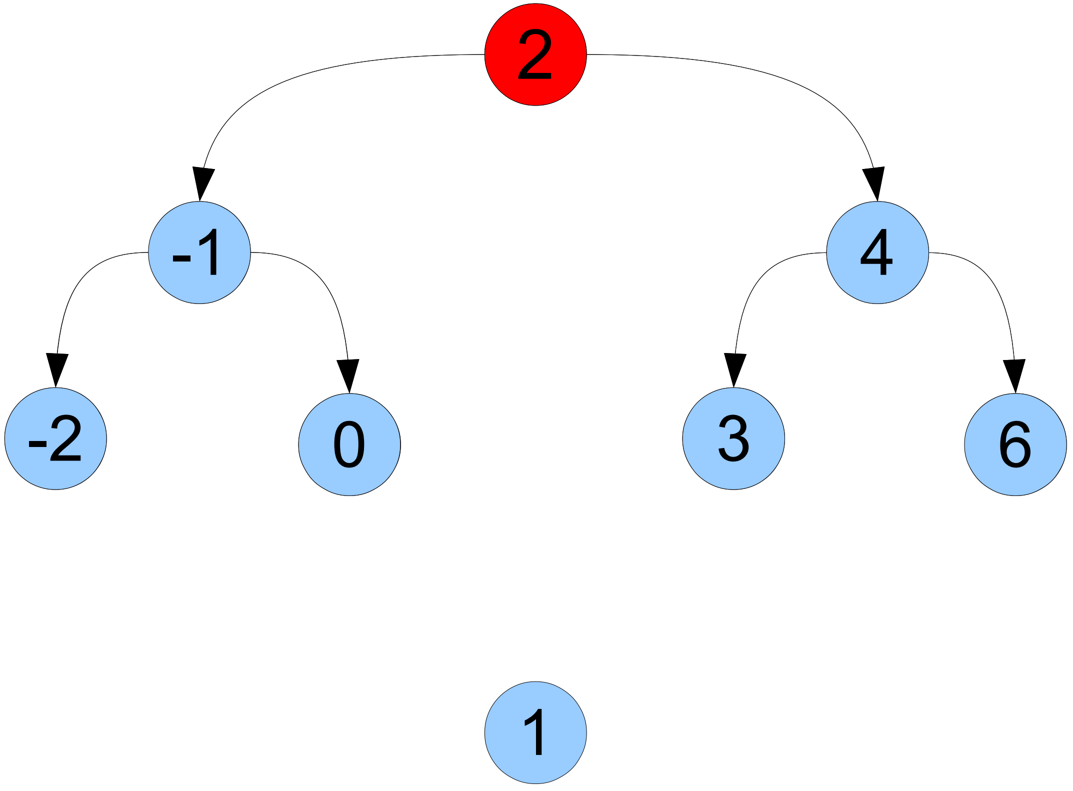
Inserting into a BST



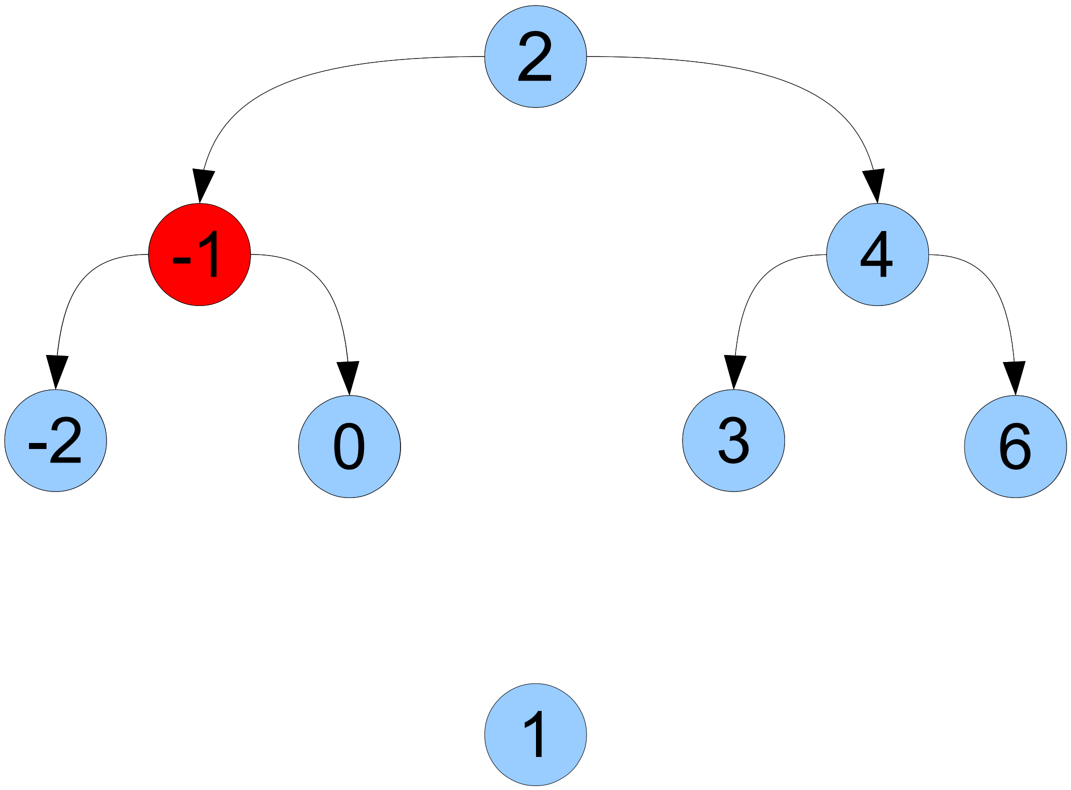
Inserting into a BST



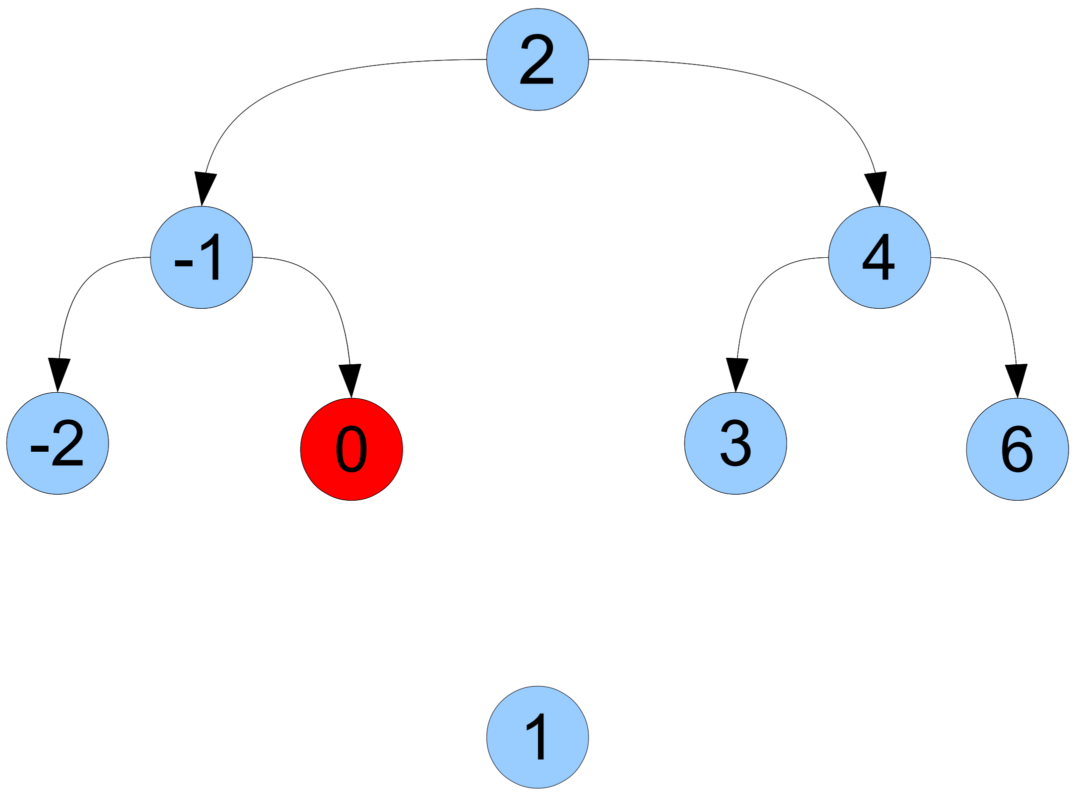
Inserting into a BST



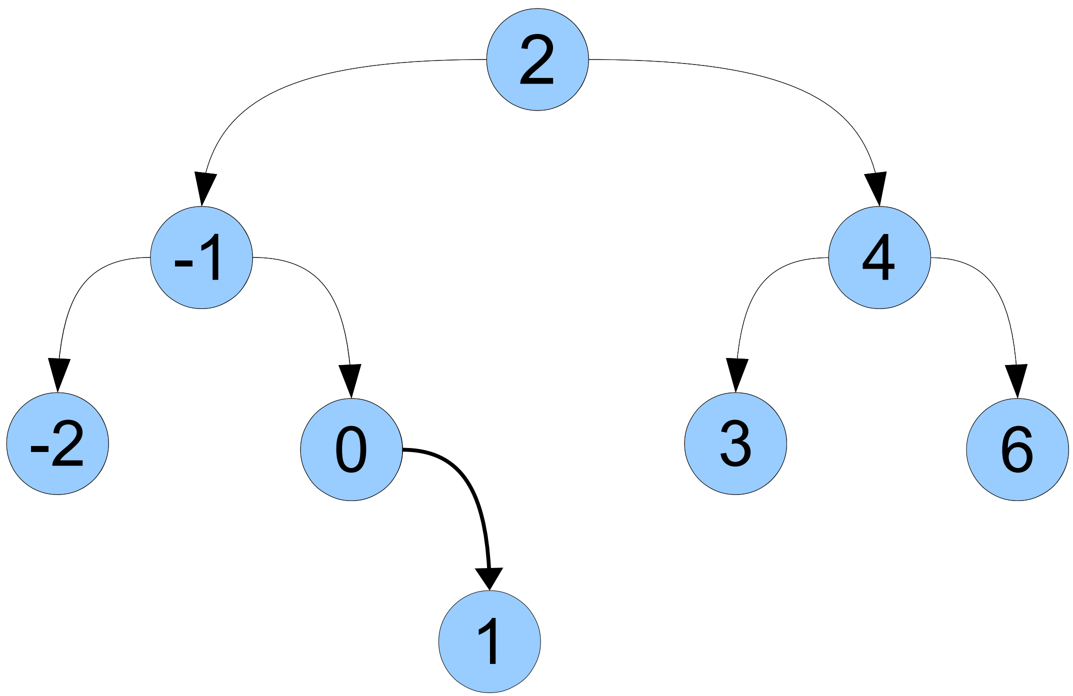
Inserting into a BST



Inserting into a BST



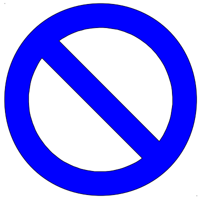
Inserting into a BST



Let's Code it Up!

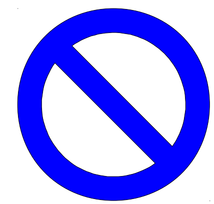
A Binary Search Tree Is Either…

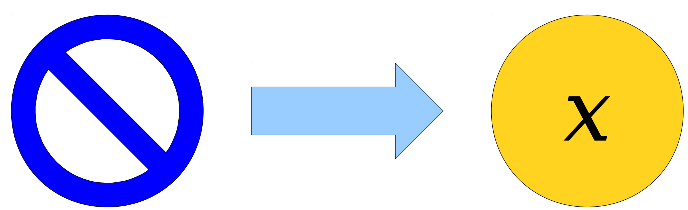
an empty tree,
represented by
`nullptr`



A Binary Search Tree Is Either…

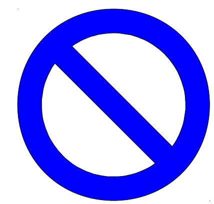
an empty tree,
represented by
`nullptr`



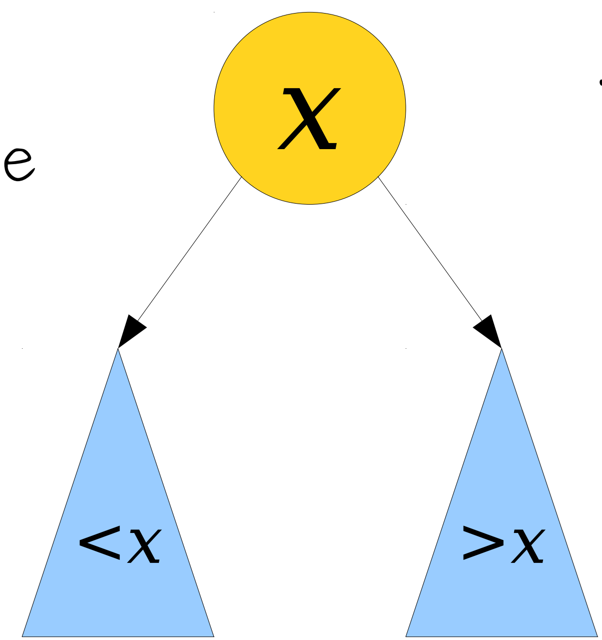


A Binary Search Tree Is Either…

an empty tree, represented by **`nullptr`**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.

Time-Out for Announcements!

Assignment 5

- Assignment 5 is due next Friday.
  - ***Recommendation:*** Complete the Vector implementation and the sorted, singly-linked list implementation by the end of this evening.
  - Try to complete the unsorted, doubly-linked list implementation by Monday.
- Questions? Concerns? Ad hominem attacks? Stop by the LaIR, our office hours, or ask on Piazza!

WiCS Casual CS Dinner

- WiCS will be holding their second biquarterly Casual CS Dinner this upcoming Monday from 6PM – 7PM in the WCC.
- Everyone is welcome – these are fantastic events!
- RSVP using **this link**.

Justice Sotomayor Visit

- Justice Sonia Sotomayor is coming to Stanford on March 10th.
- There's a lottery system for tickets. I would ***highly recommend*** putting your name in! She's really impressive!

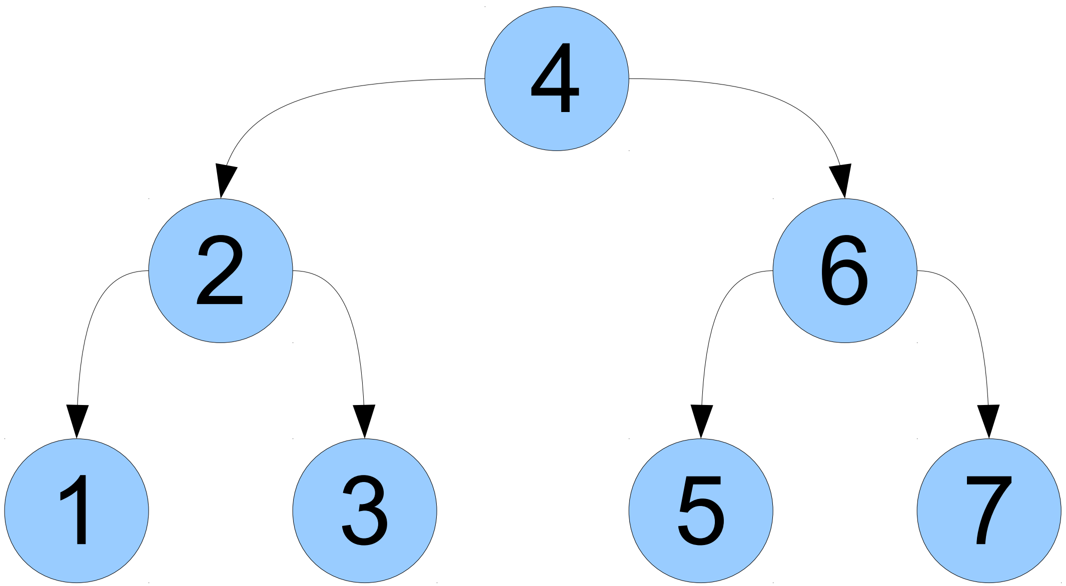
Back to our regularly scheduled programming…

So, how efficient is this?

Insertion Order Matters

- Suppose we create a BST of numbers in this order:

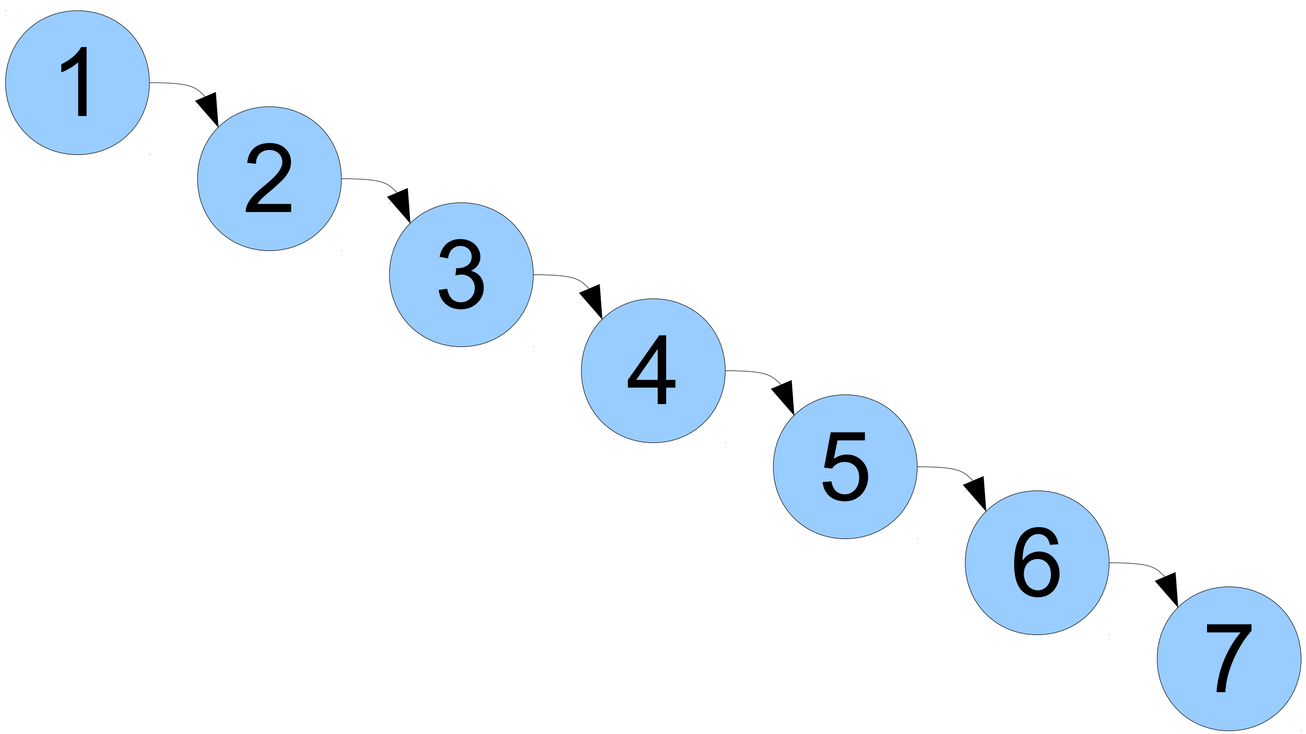
4, 2, 1, 3, 6, 5, 7



Insertion Order Matters

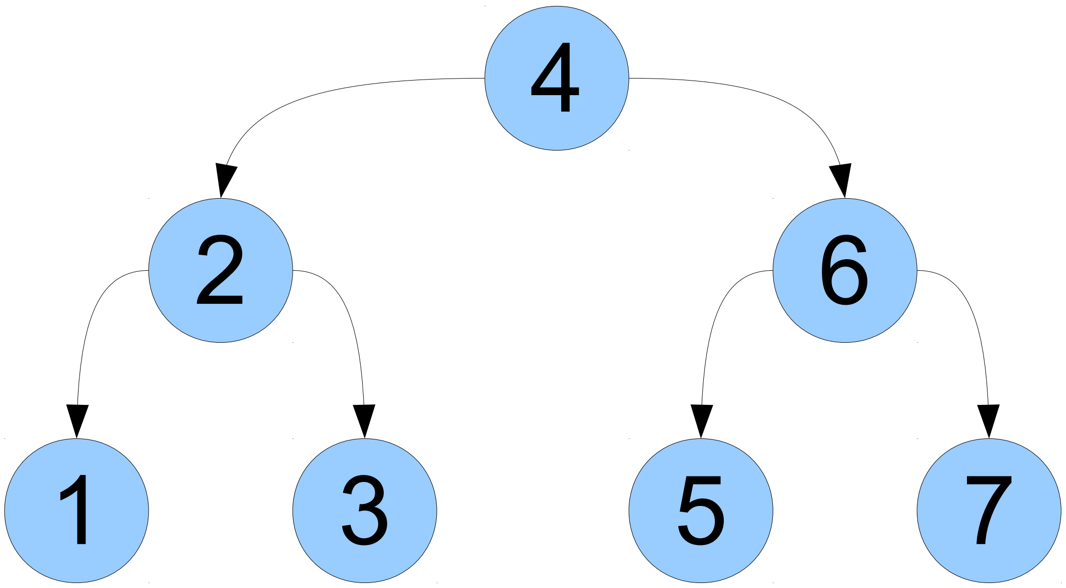
- Suppose we create a BST of numbers in this order:

1, 2, 3, 4, 5, 6, 7



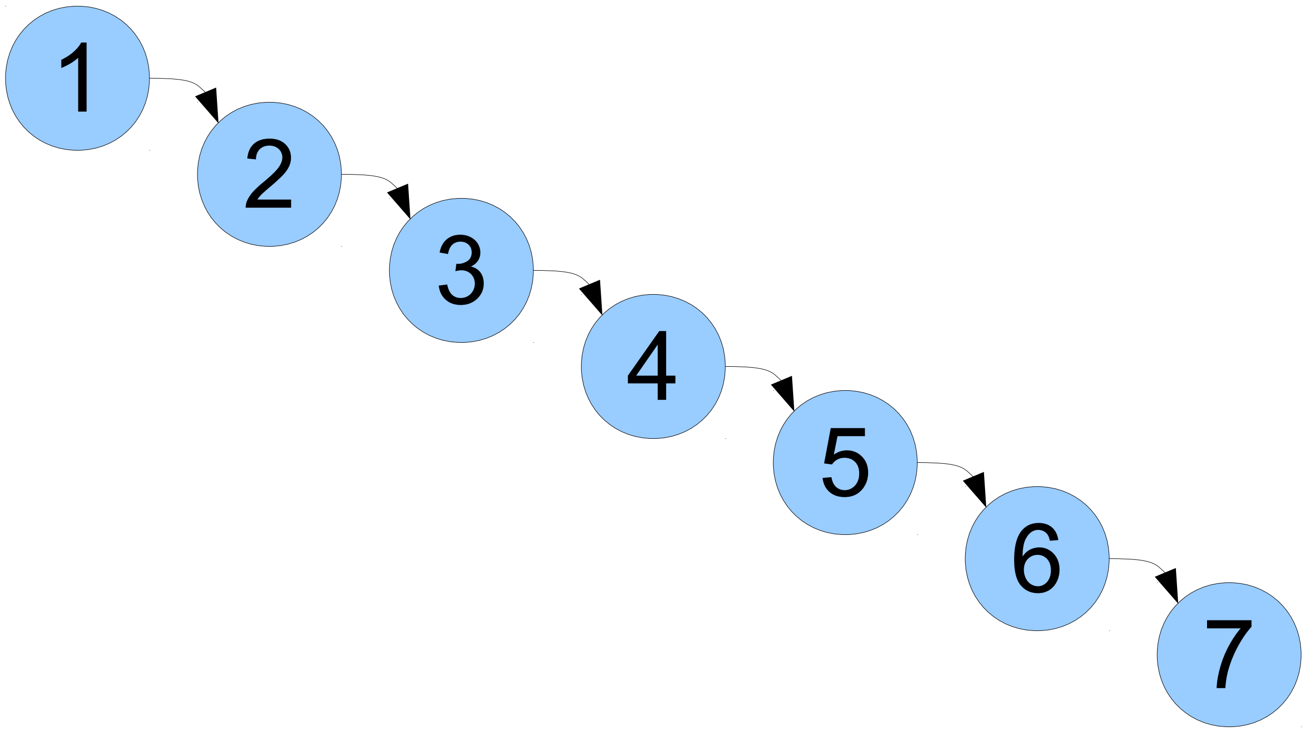
Tree Terminology

- The ***height*** of a tree is the number of nodes in the longest path from the root to a leaf.
- By convention, an empty tree has height -1.



Tree Terminology

- The ***height*** of a tree is the number of nodes in the longest path from the root to a leaf.
- By convention, an empty tree has height -1.



Efficiency Questions

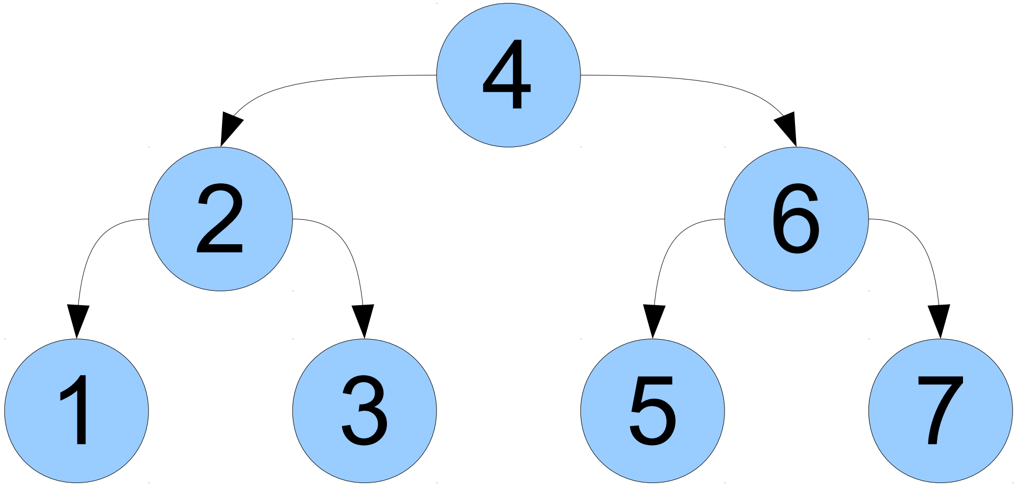
- What is the big-O complexity of adding a node into a BST, or searching a BST for a given value?
- ***Answer:*** It depends on the height of a tree!
- Each step in these processes does O(1) work and then drops us one level lower in the BST.
- The overall time spent is **O(*h*)**, where *h* is the height of the tree.

Tree Heights

- What are the maximum and minimum heights of a tree with n nodes?
- Maximum height: all nodes in a chain. Height is $O(n)$.

Tree Heights

- What are the maximum and minimum heights of a tree with n nodes?
- Maximum height: all nodes in a chain. Height is $O(n)$.
- Minimum height: Tree is as complete as possible. Height is $O(\log n)$.



Keeping the Height Low

- There are many modifications of the binary search tree designed to keep the height of the tree low (usually O(log *n*)).
- A ***self-balancing binary search tree*** is a binary search tree that automatically adjusts itself to keep the height low.
- The textbook talks about AVL trees, which are one way you can do this.
- You don't need to know these techniques for CS106B: honestly, they're complicated, require a ton of memorization, and rarely come up.
  - Take CS166 if you want to learn more!

Next Time

- ***More BST Fun***
  - Some other cool tricks and techniques!
- ***Custom Types in Sets***
  - Resolving a longstanding issue.