LEC 10

CSE 123

Comparable; Binary Search Trees

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Do you collect anything? What?

Respond on sli.do!

Instructor: Trien Vuong

TAs: Cora, Jonah, Maitreyi, Nhan, Nichole, Shiven

Music: [CSE 123 26su Lecture Tunes]

Lecture Outline: Announcements

- **Announcements**
- Comparable
- Binary Search Review
- Binary (Search) Trees Review
- More runtime!

Announcements

- Creative Project 2 due **Wednesday** (8/01) at 11:59pm
- Resubmission Cycle 3 due tonight at 11:59pm
  - ***C0***, P1 eligible

Lecture Outline: Comparable

- Announcements
- **Comparable**
- Binary Search Review
- Binary (Search) Trees Review
- More runtime!

Comparable

- `Comparable<E>` is an `interface` that allows implementers to define an ordering between two objects
  - Used by `TreeSet`, `TreeMap`, `Collections.sort`, etc.

- One required method:

```
public int compareTo(E other);
```

- Returned integer falls into 1 of 3 categories
  - `<` 0: `this` is “less than” `other`
  - `=` 0: `this` is “equal to” `other`
  - `>` 0: `this` is “greater than” `other`

```
a.compareTo(b);
```

`this` `other`

Lecture Outline: Binary Search Review

- Announcements
- Comparable
- **Binary Search Review**
- Binary (Search) Trees Review
- More runtime!

contains() on Collections

- contains() checks if an element is in a collection.
- How fast is contains() on an ArrayList? LinkedList? IntTree?
- Is it possible to be faster?

Looking through a dictionary

- How would you find your favorite word in the dictionary?
  - Look at every page?
  - That would take O(n) time where n is the size of the dictionary.

- Can you find a word more quickly?

Looking through a dictionary pseudocode

- Assuming a *sorted order* of elements to search through `list`
- Suppose you're looking for a specific element `target`
- Return the index of the given `target`, or -1 if it's not in the `list`

```
begin with the dictionary, from the first to last word,
        looking for target

search(dictionary, left, right, target):
    if there are no more words to look through
        give up
    else
        pick a midpoint between left and right
        pick the word at that midpoint
        if target is that word
            found it!
        else if target comes before that word
            search(dictionary, left, midpoint-1, target)
        else (target comes after that word)
            search(dictionary, midpoint+1, right, target)
```

Binary Search

- Assuming a *sorted order* of elements to search through `list`
- Suppose you're looking for a specific element `target`
- Return the index of the given `target`, or -1 if it's not in the `list`

```
begin with search(list, 0, list.size() - 1, target)

search(list, left, right, target):
    if (left > right):
        return -1
    else:
        mid = (left + right) / 2
```

How would we add an element to the array while retaining our ability to do binary search?

How fast is this add operation?

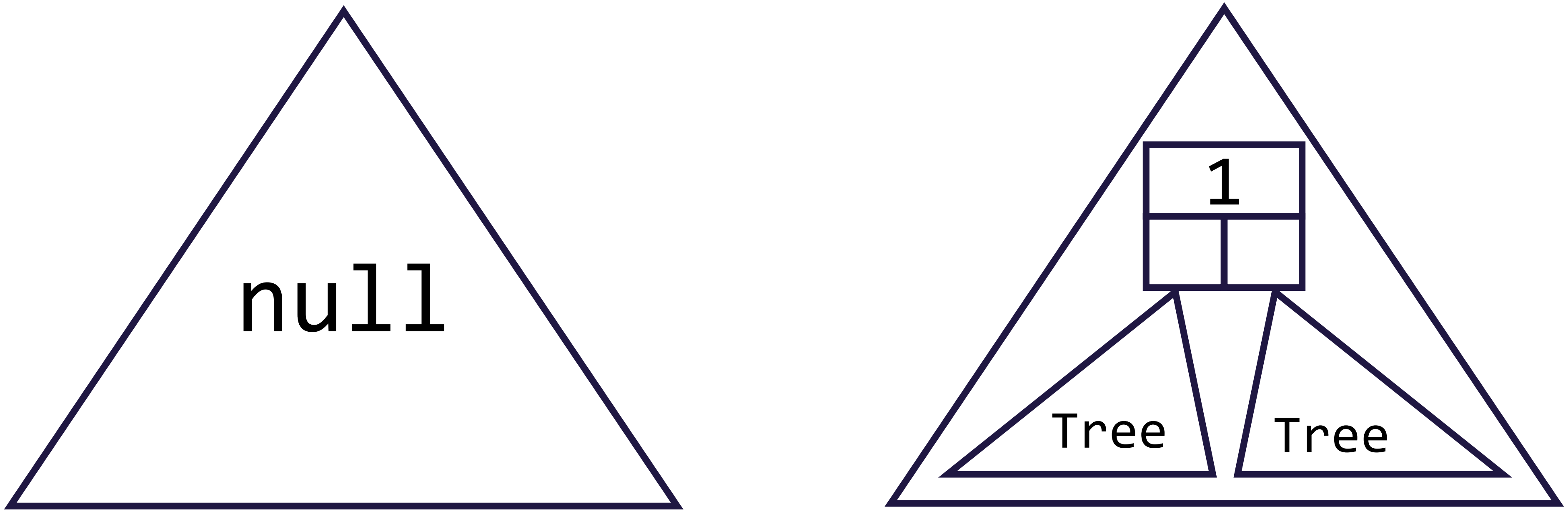
-5	4	18	23	30	49	55	108	184

Lecture Outline: Binary (Search) Trees Review

- Announcements
- Comparable
- Binary Search Review
- **Binary (Search) Trees Review**
- More runtime!

Binary Trees [Review]

- We'll say that any Binary Tree falls into one of the following categories:



Empty tree

`root == null`

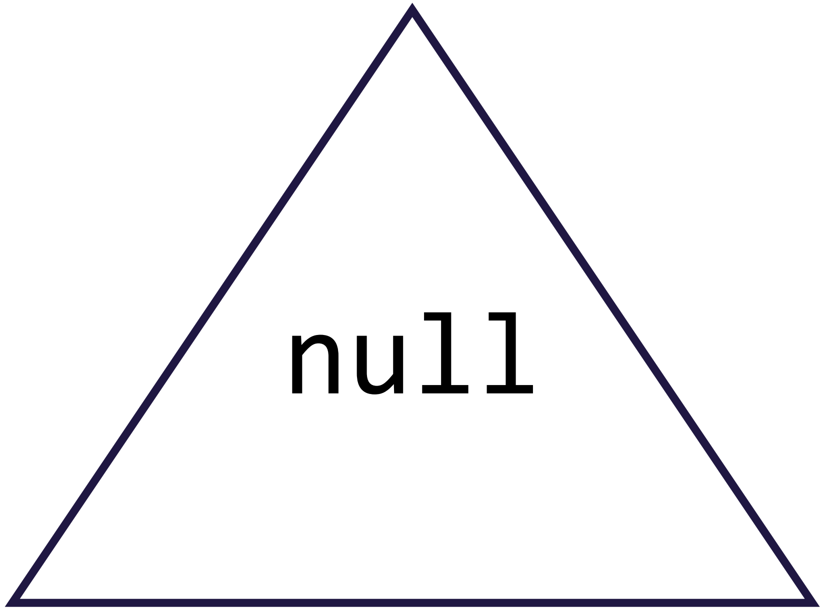
Node w/ two subtrees

`root != null`
`root.left / root.right = Tree`

This is a recursive definition! A tree is either empty or a node with two more trees!

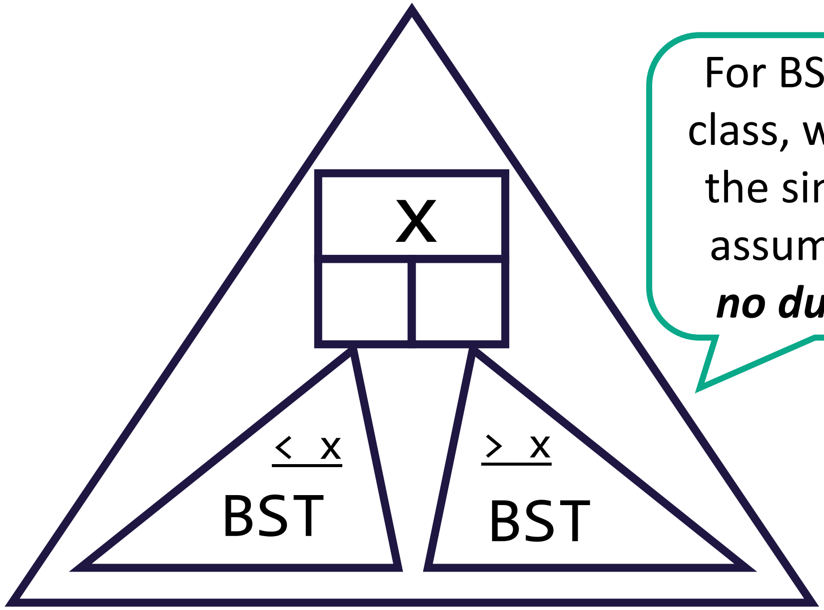
Binary Search Trees (BSTs)

- We'll say that any Binary Search Tree falls into the following categories:



Empty BST

```
root == null
```



For BSTs in this class, we'll make the simplifying assumption of ***no duplicates***

Node w/ two subBSTs

```
root != null
root.left / root.right = Tree
max(root.left) < x && min(root.right) > x
```

Note that not all Binary Trees are Binary Search Trees

Why BSTs?

- Our `IntTree` implementation to `contains(int value)`

```
private boolean contains(int value, IntTreeNode root) {
    if (root == null) {
        return false;
    } else {
        return root.data == value ||
               contains(value, root.left) ||
               contains(value, root.right);
    }
}
```

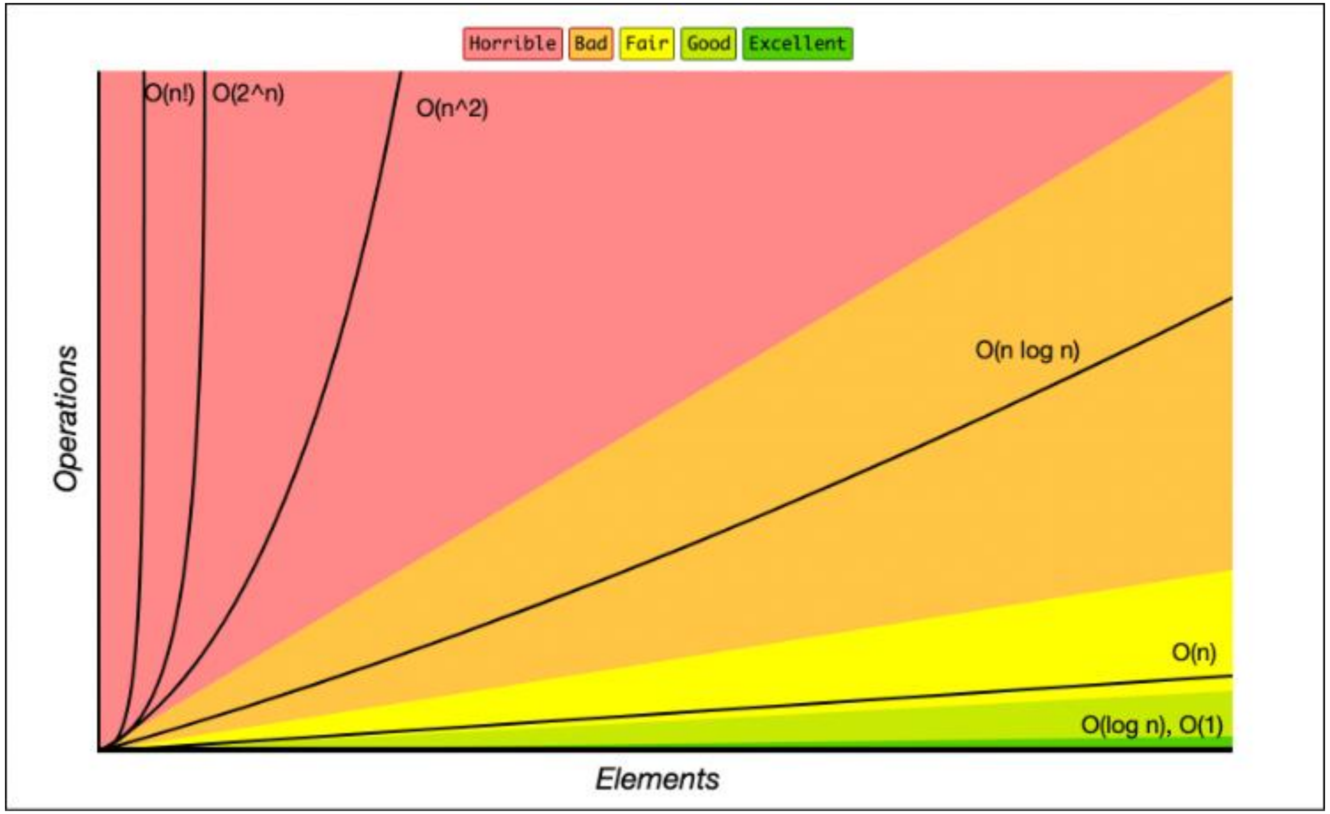
- Which direction(s) do we travel if `root.data != value`?
  - Both left and right
- In a Binary Search Tree, should we check both sides?
  - Remember, additional constraint: `max(root.left) < root.data &&`
    `min(root.right) > root.data`

Lecture Outline: More runtime!

- Announcements
- Binary Search Review
- Binary (Search) Trees Review
- **More runtime!**

BSTs & Runtime (1)

- Contains operation on a <u>balanced</u> BST runs in `O(log(n))`
  - Leverages removing half of the values at each step
  - *New runtime class unlocked!*



BSTs & Runtime (2)

- Contains operation on a <u>balanced</u> BST runs in `O(log(N))`
  - Leverages removing half of the values at each step
  - *New runtime class unlocked!*

- Comparison between data structures:

`Operation`	`ArrayIntList`	`LinkedIntList`	`IntSearchTree`
`contains(x)`	`O(N)`	`O(N)`	`O(log(N)) ?`

BSTs & Runtime (3)

- Contains operation on a balanced BST runs in `O(log(N))`
  - Leverages removing half of the values at each step
  - *New runtime class unlocked!*

- Comparison between data structures:

`Operation`	`ArrayIntList`	`LinkedIntList`	`IntSearchTree`
`contains(x)`	`O(N)`	`O(N)`	***`O(N)`***

`O(Log(N))` runtime is only guaranteed for **BALANCED** *BSTs. If your tree isn't balanced, we see `O(N)` runtime!*

BSTs In Java

- Self-balancing BST implementations (AVL / Red-black) exist
  - AVL better at contains, Red-black better at adding / removing
- Both the `TreeMap` / `TreeSet` implementations use self-balancing BSTs
  - Determines said ordering via the `Comparable` interface / `compareTo` method
  - Printing out shows natural ordering – preorder traversal
- Complete table comparing data structures:

Operation	ArrayList	LinkedList	TreeSet
contains(x)	O(N)	O(N)	O(log(N))
add(x)	O(1*)	O(1)	O(log(N)*)
remove(x)	O(N)	O(N)	O(log(N)*)

**It's slightly more complicated but we'll leave that for a higher level course*