CS 106B, Lecture 18
Binary Search Trees

Plan for Today

- Start with a discussion of how to implement a Set
  - The importance of choosing a good data structure
- Move into trees, a new kind of data structure
- We'll focus on "reading" trees today – modifying trees will be tomorrow's lecture

Designing a Set

- We've seen how to implement:
  - Stack (array or linked list)
  - Vector (array)
  - Queue (linked list)
- How would we implement Set?
  - Add
  - Contains
  - Remove

First Try

- Store all the elements in an **unsorted** array or linked list
  - What is the Big-Oh of contains?
  - What is the Big-Oh of adding an element?
  - What is the Big-Oh of removing an element?

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
3	8	9	7	5	12	4	8	1	6	75

Another attempt

- What if we **sorted** the array?
  - What is the Big-Oh of contains?
  - What is the Big-Oh of adding an element?
  - What is the Big-Oh of removing an element?

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2	5	6	8	11	13	17	22	23	29	31

Binary Search

- Fast way to search for elements in a **sorted array**
- Looping through elements one by one is slow [O(N)]
- Idea:

Jump to the middle element:

if the middle is what we're looking for, we're done. Hooray!

if the middle is too small (we didn't go far enough) – we rule out the entire **left side** of elements smaller than the middle element

if the middle is too big(we went too far) – we rule out the entire **right side** of elements bigger than the middle element

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2	5	6	8	11	13	17	22	23	29	31

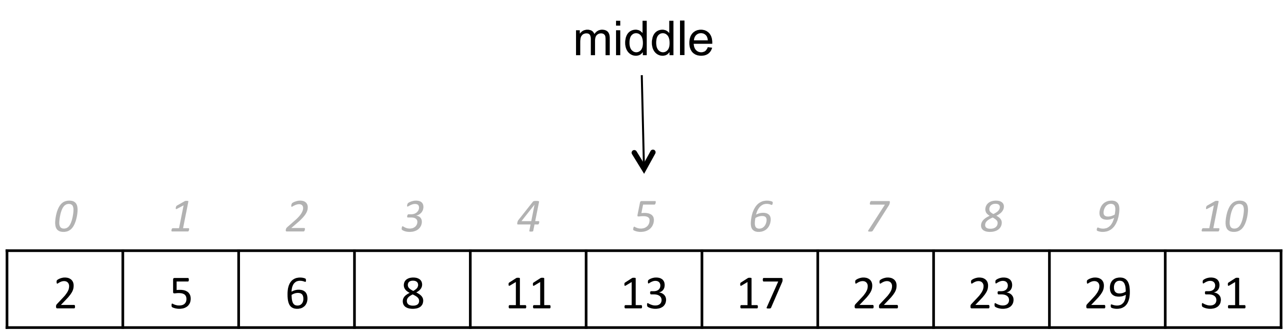
Binary Search in Action

- Search for 8:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2	5	6	8	11	13	17	22	23	29	31

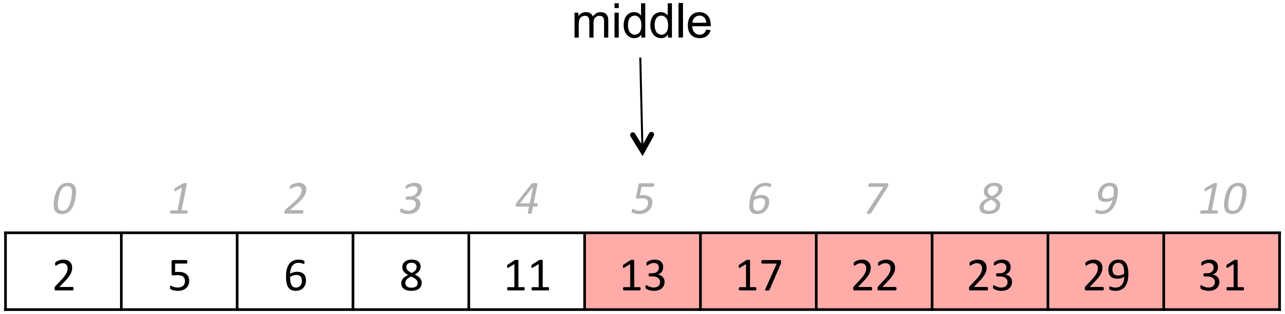
Binary Search in Action

- Search for 8:



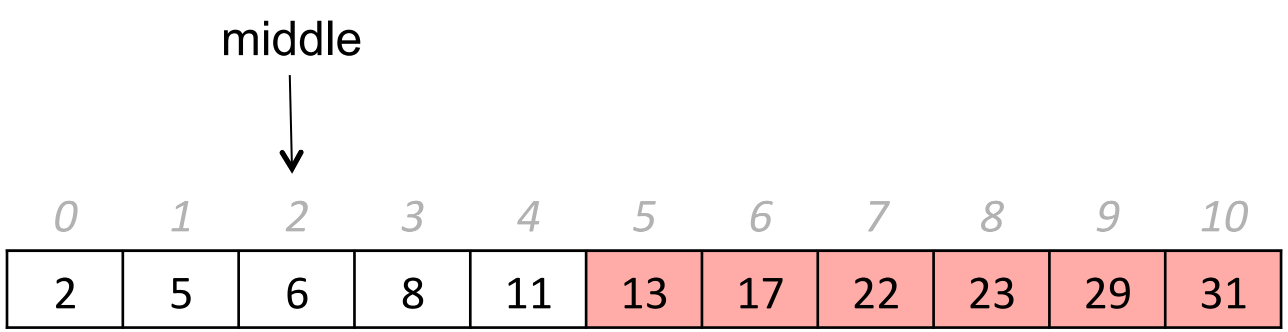
Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10



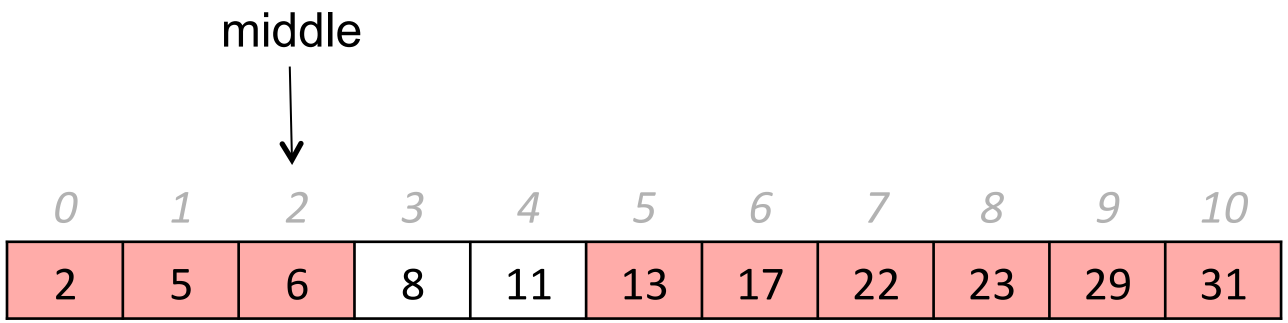
Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Pick the new middle of the remaining elements
- Look at 6:



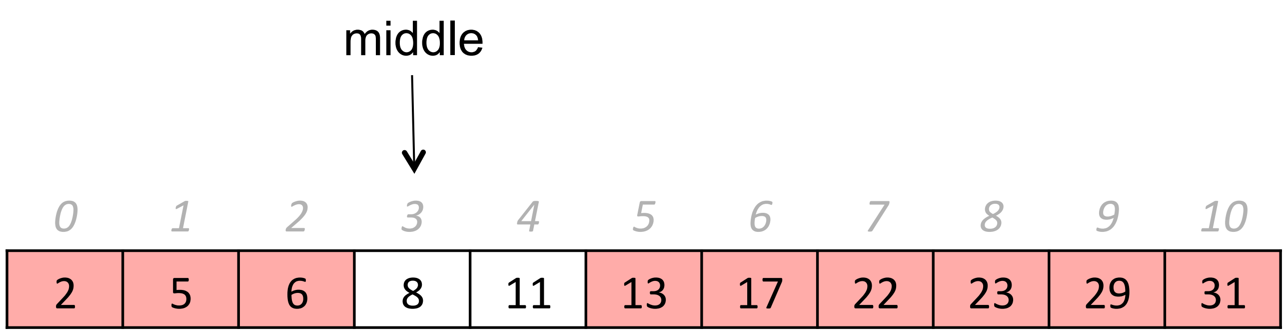
Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Pick the new middle of the remaining elements
- Look at 6:
  - it's too small, so we rule out indices 0-3



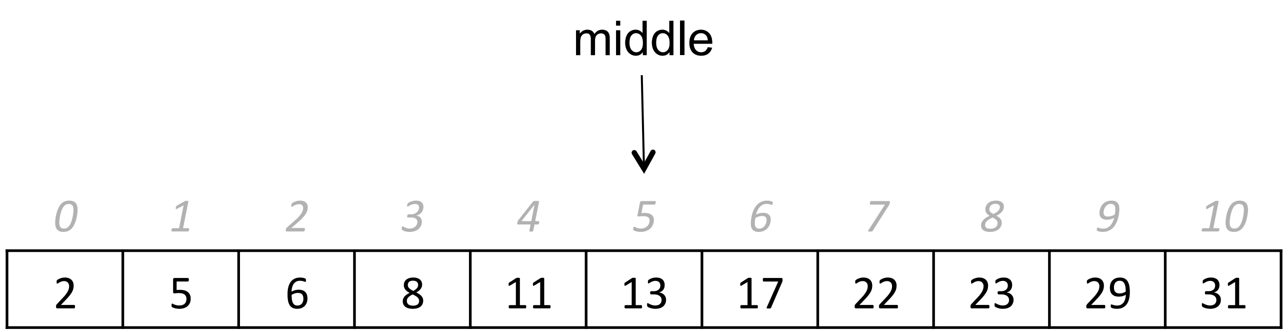
Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Pick the new middle of the remaining elements
- Look at 6:
  - it's too small, so we rule out indices 0-3
- Look at 8:
  - it's just right! We return true



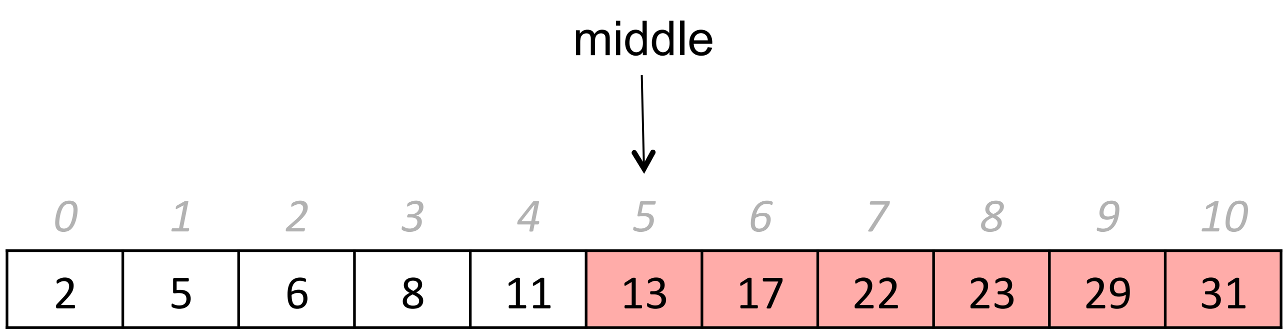
Binary Search in Action

- Search for 7:



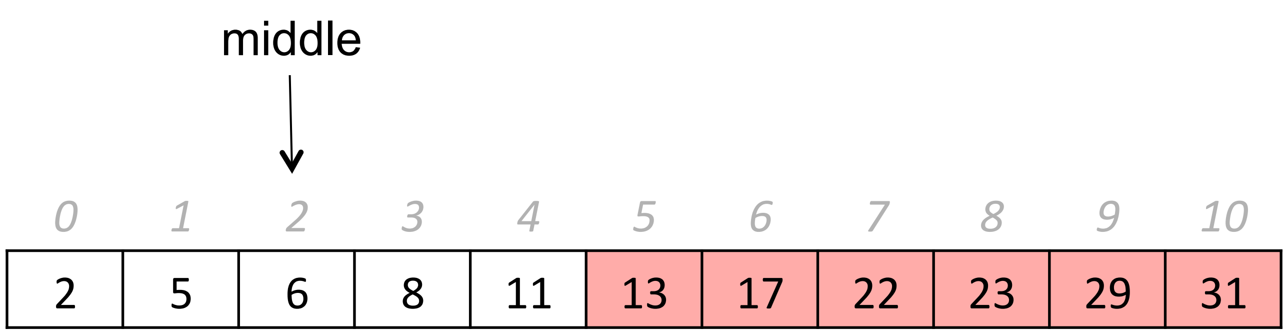
Binary Search in Action

- Search for 7:
- Look at 13
  - it's too big, so we rule out indices 5-10



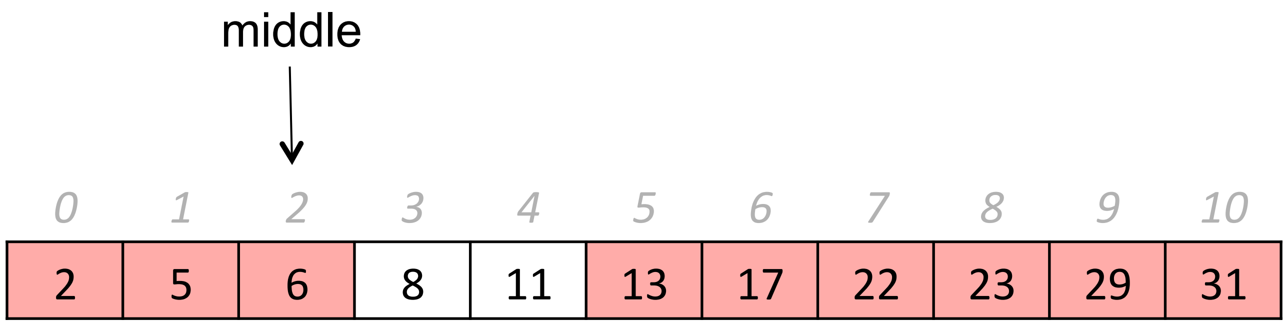
Binary Search in Action

- Search for 7:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Pick the new middle of the remaining elements
- Look at 6:



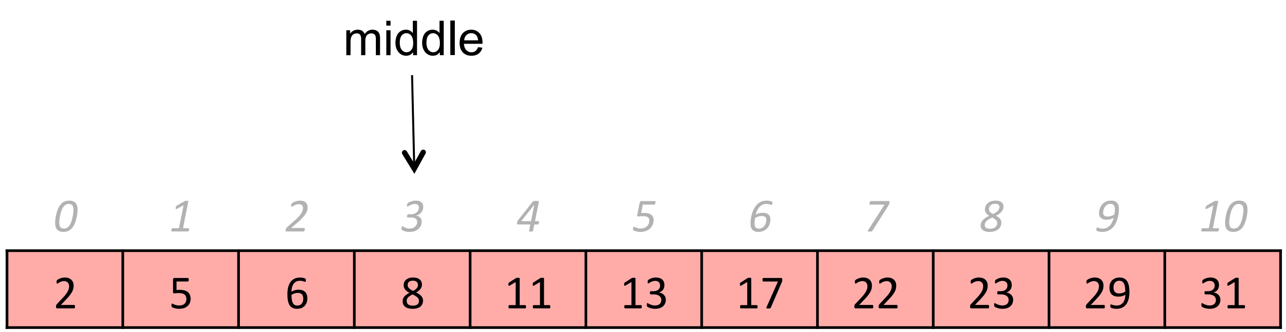
Binary Search in Action

- Search for 7:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Pick the new middle of the remaining elements
- Look at 6:
  - it's too small, so we rule out indices 0-3



Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Look at 6:
  - it's too small, so we rule out indices 0-3
- Look at 8:
  - it's too big! We rule out elements 3-4



Binary Search in Action

- Search for 8:
- Look at 13
  - it's too big, so we rule out indices 5-10
- Look at 6:
  - it's too small, so we rule out indices 0-3
- Look at 8:
  - it's too big! We rule out elements 3-4
- No elements left to search – we return false

middle ↓

0	1	2	3	4	5	6	7	8	9	10
2	5	6	8	11	13	17	22	23	29	31

Sorted Array

- What if we **sorted** the array?
  - What is the Big-Oh of contains?
    - O(log N)
  - What is the Big-Oh of adding an element?
    - O(N)
  - What is the Big-Oh of removing an element?
    - O(N)

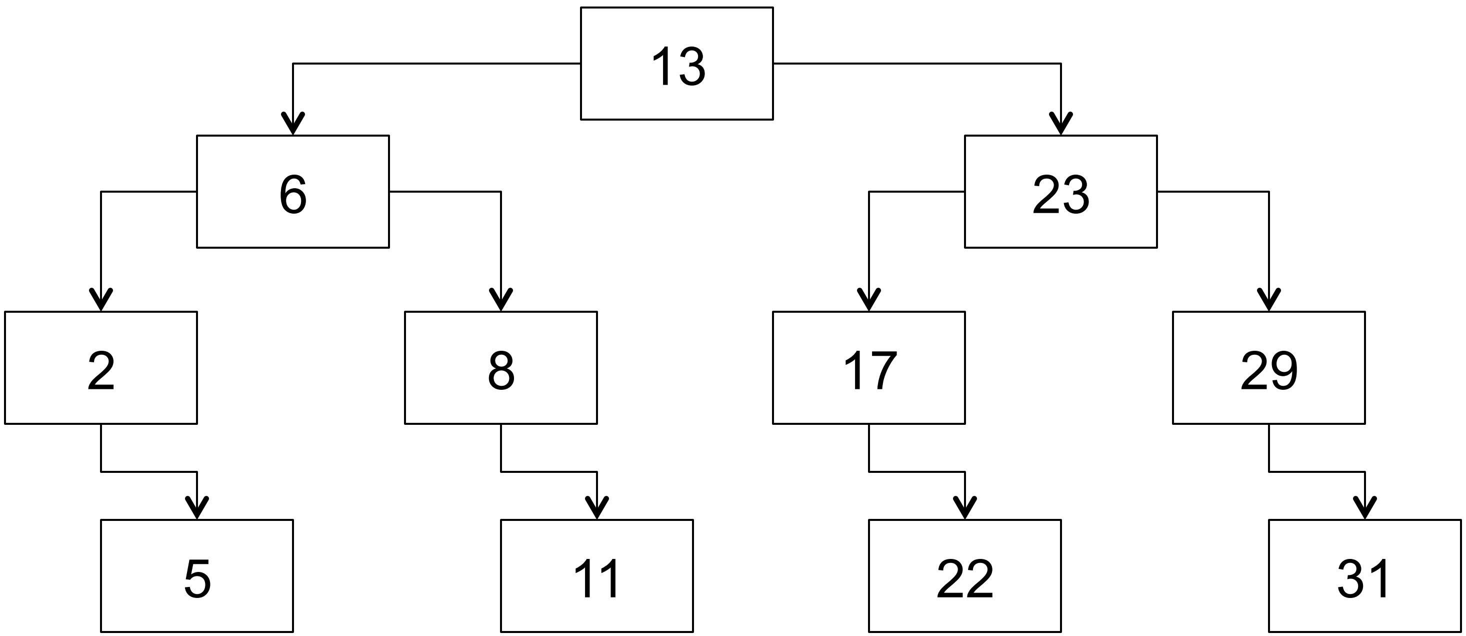
0	1	2	3	4	5	6	7	8	9	10
2	5	6	8	11	13	17	22	23	29	31

A Modification

- Problem: an array is slow to insert into or remove from
- Our solution was a **linked list** – have each element connected to one other element
  - Easy to add/remove elements
  - Can't skip elements – need to go in order
- Maybe we can find some way to implement the jumps necessary for binary search...

A Modification

- What are all the possible paths binary search could take on this array (ties are broken by choosing the smaller element)?



0	1	2	3	4	5	6	7	8	9	10
2	5	6	8	11	13	17	22	23	29	31

A Modification

- Key idea: we always jump to one of two elements in binary search (depending on if the element we're looking at is too big or too small)
- What if we had a Linked List where we stored two pointers, allowing us to make those jumps quickly?

Binary Search Tree

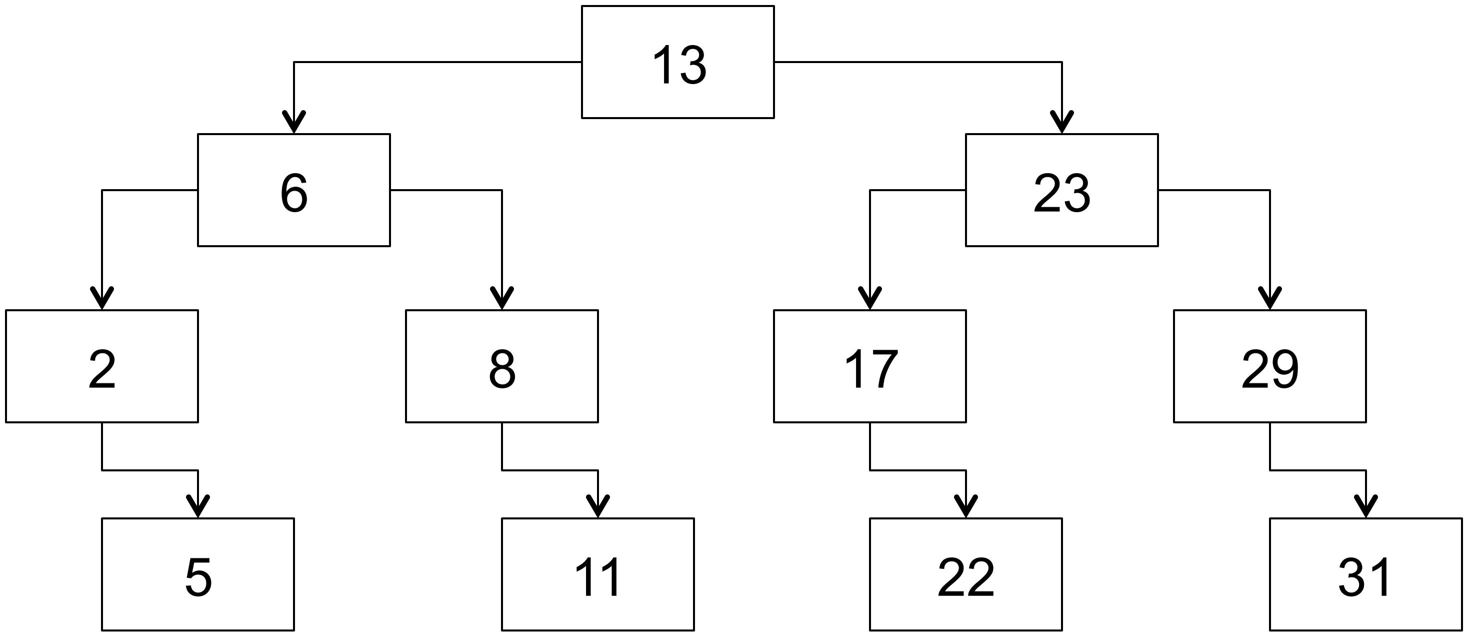
- A **tree** is a data structure where each element (**parent**) stores two or more pointers to other elements (its **children**)
  - A doubly-linked list doesn't count because, just like outside of computer science, a child can not be its own ancestor
- Each node in a **binary tree** has two pointers
  - Some of these pointers may be `nullptr` (just like in a linked list)
  - We'll see examples of non-binary trees in future lectures
- A **binary search tree** is a binary tree with special ordering properties that make it easy to do binary search
- Similar to a Linked List:
  - Each element in its own block of memory
  - Have to travel through pointers (can't skip "generations")

(Binary) TreeNode

```
struct TreeNode {
    int data; // assume that the tree stores ints
    TreeNode *left;
    TreeNode *right;
};
```

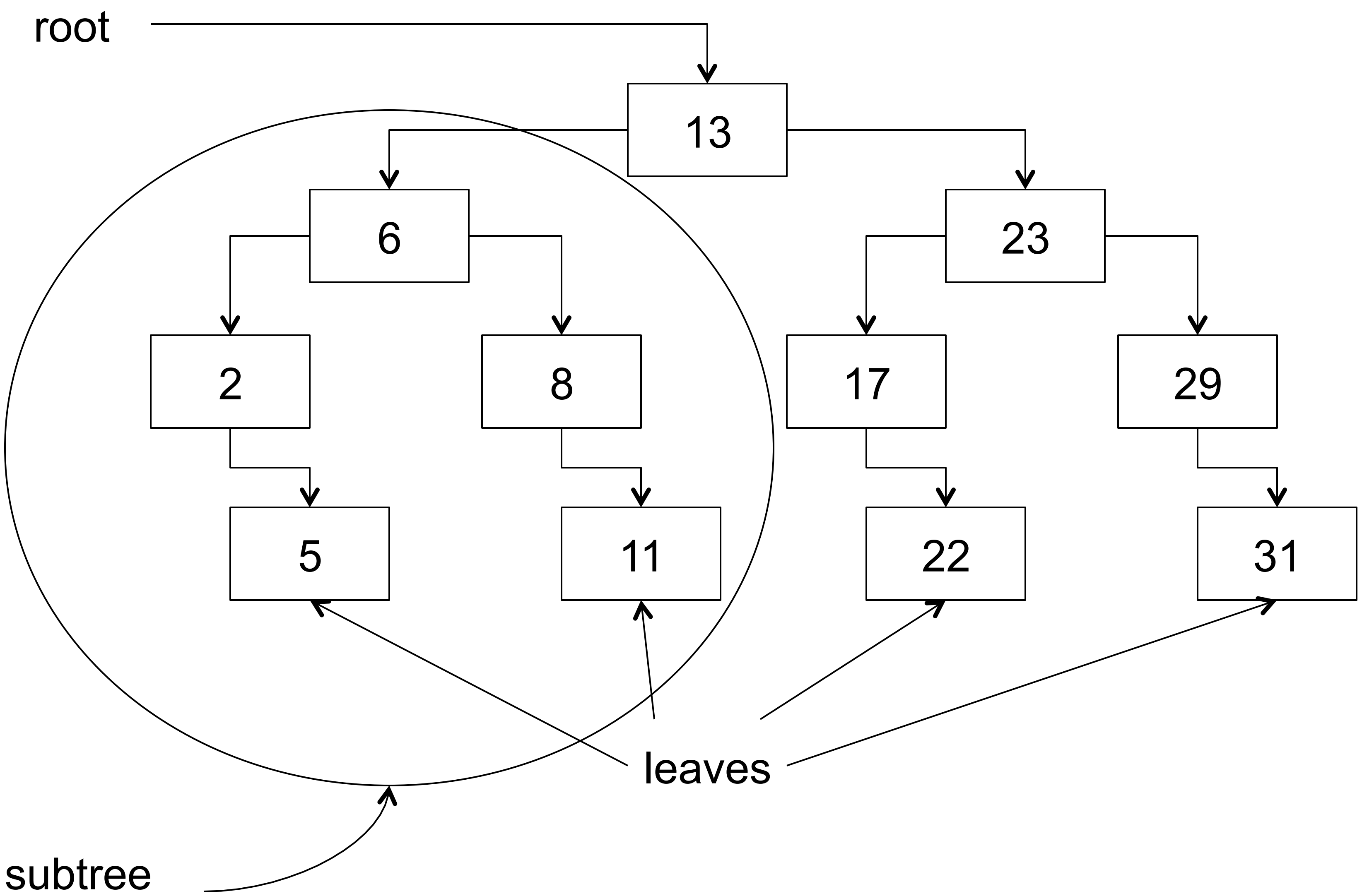
Binary Search Trees

- We'll say a binary search tree has the following property:
  - All elements to the left of an element are smaller than that element
  - All elements to the right of an element are bigger than that element
  - Just like our sorted array!



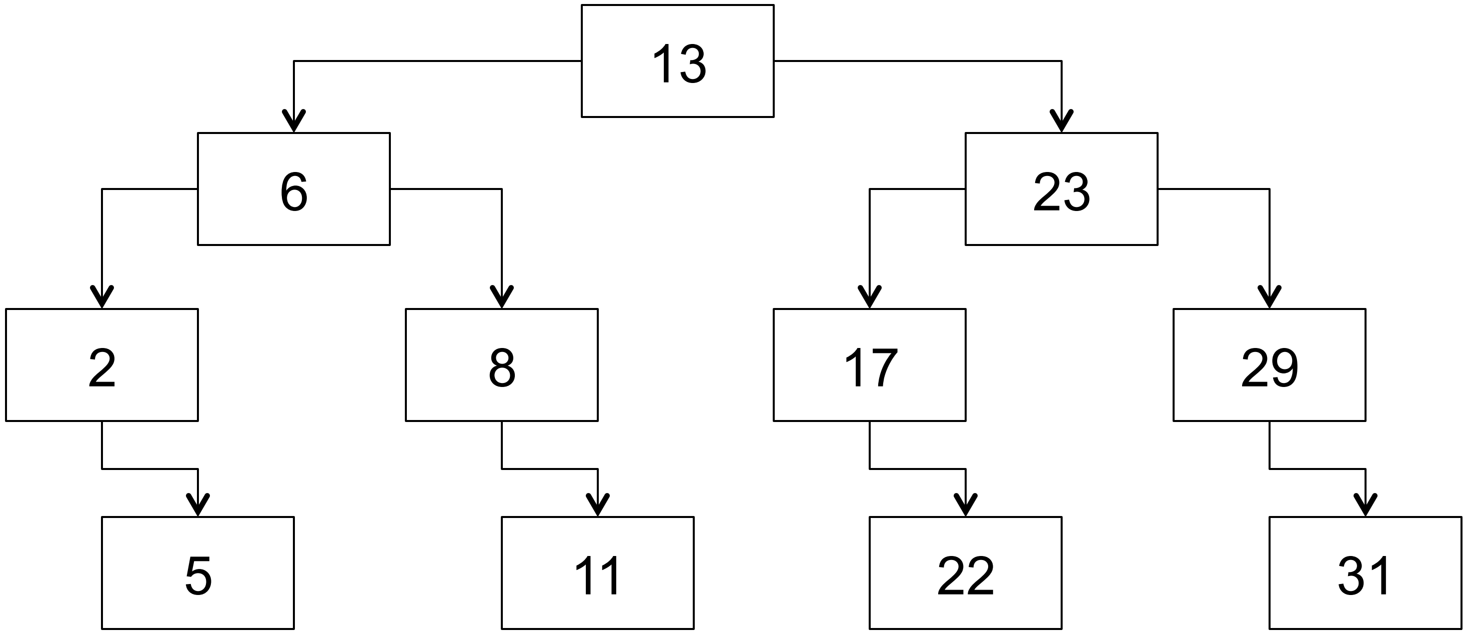
0	1	2	3	4	5	6	7	8	9	10
2	5	6	8	11	13	17	22	23	29	31

Tree anatomy



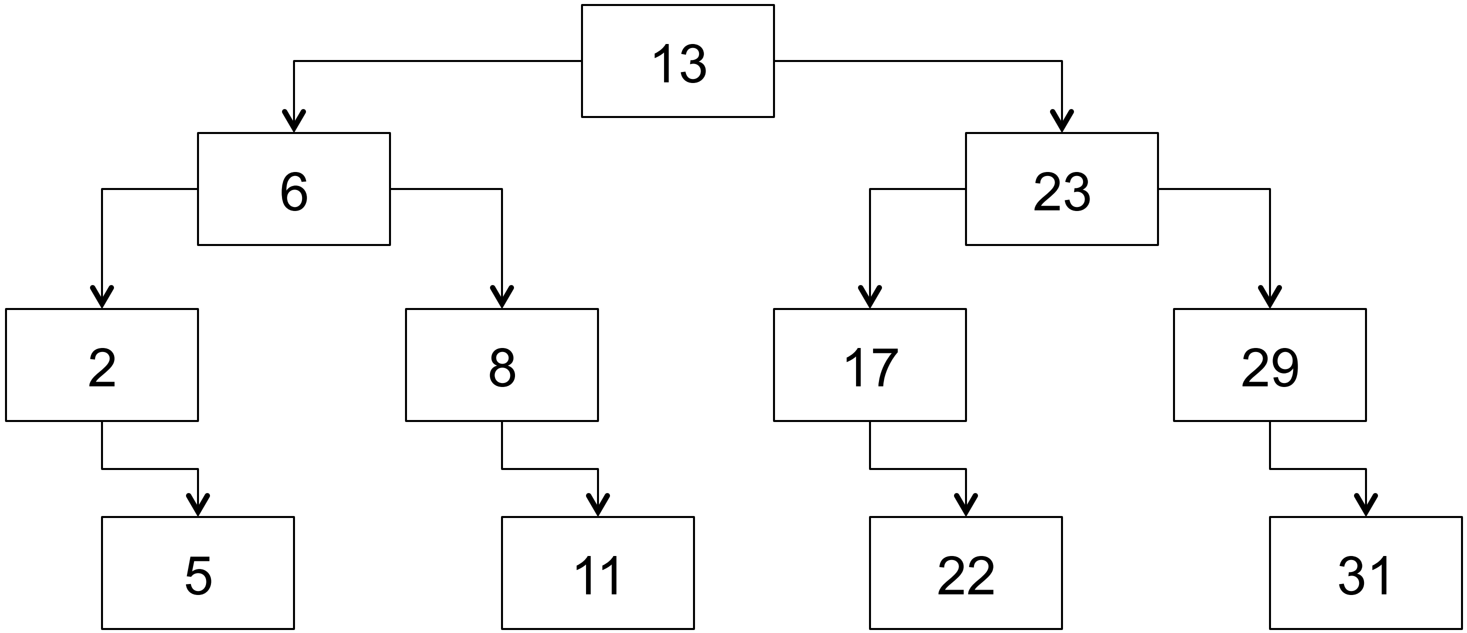
BST Contains

- How would you search a BST for an element?



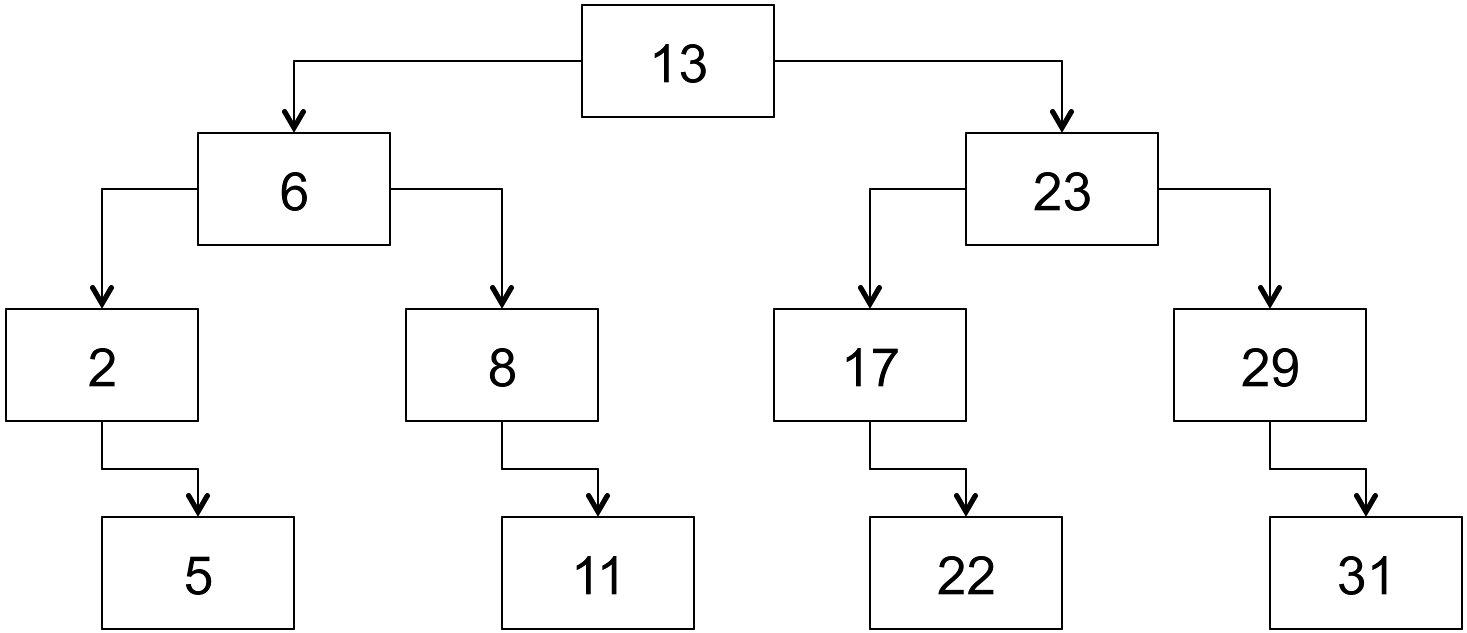
BST Contains

- How would you search a BST for an element?
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



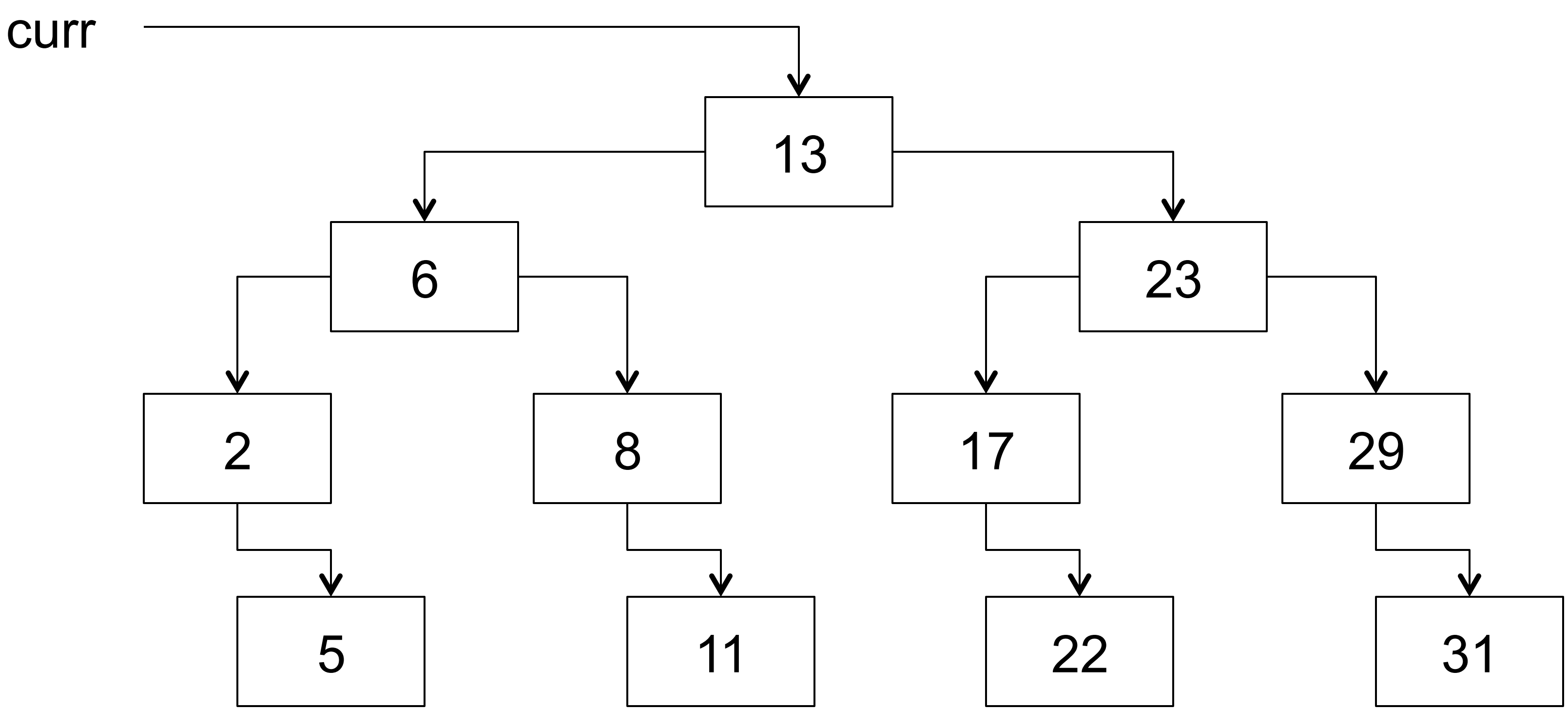
Trees and Recursion

- Trees are fundamentally **recursive (**subtrees are smaller trees)
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



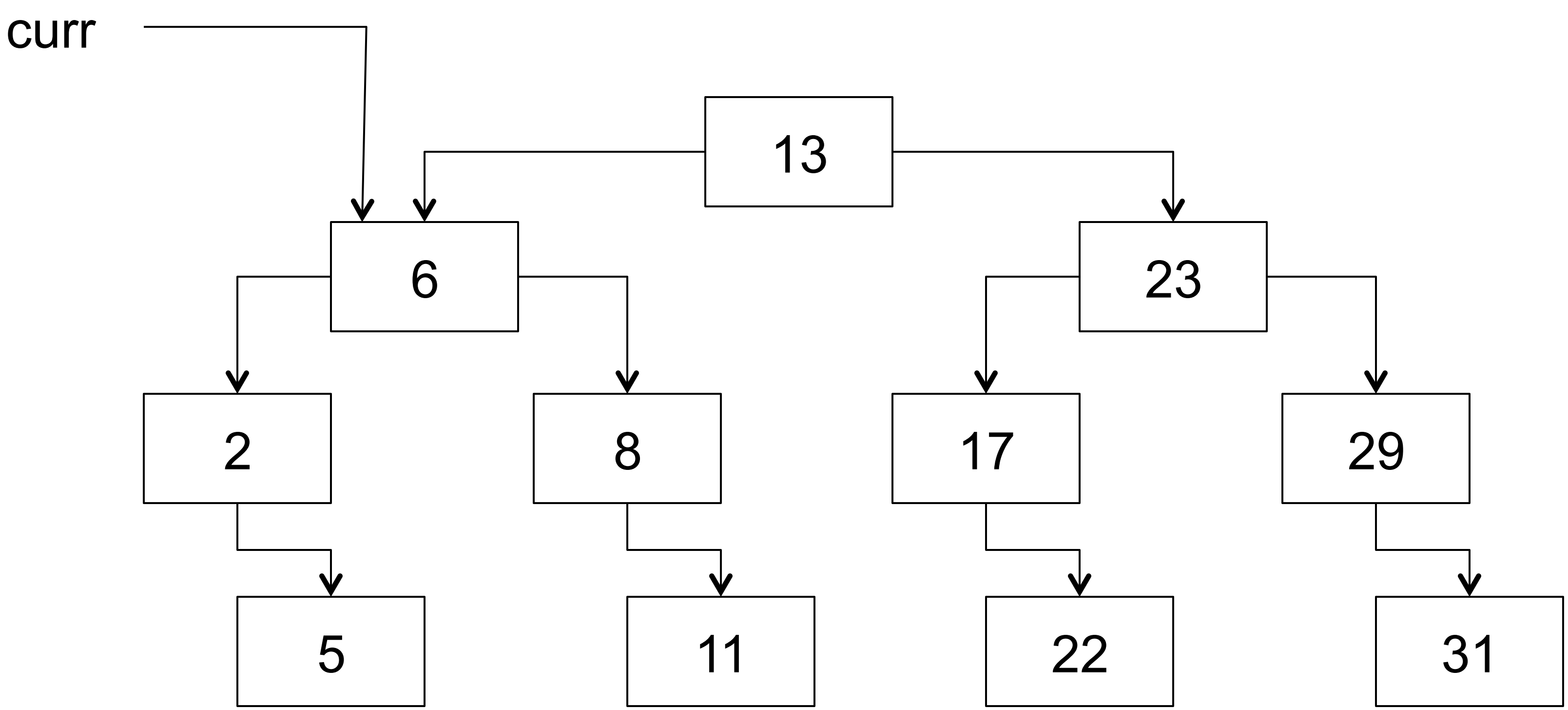
Trees and Contains

- Search for 5
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



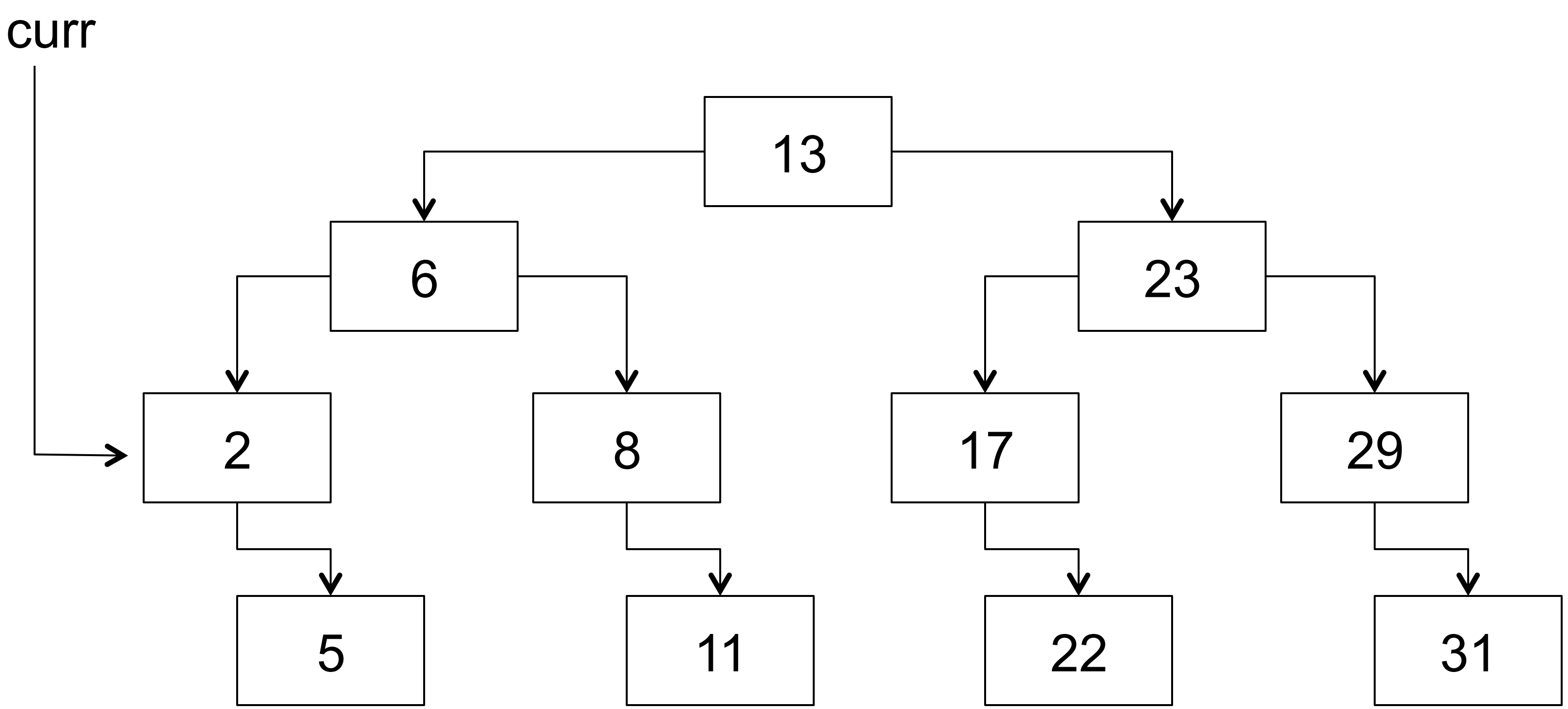
Trees and Contains

- Search for 5
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



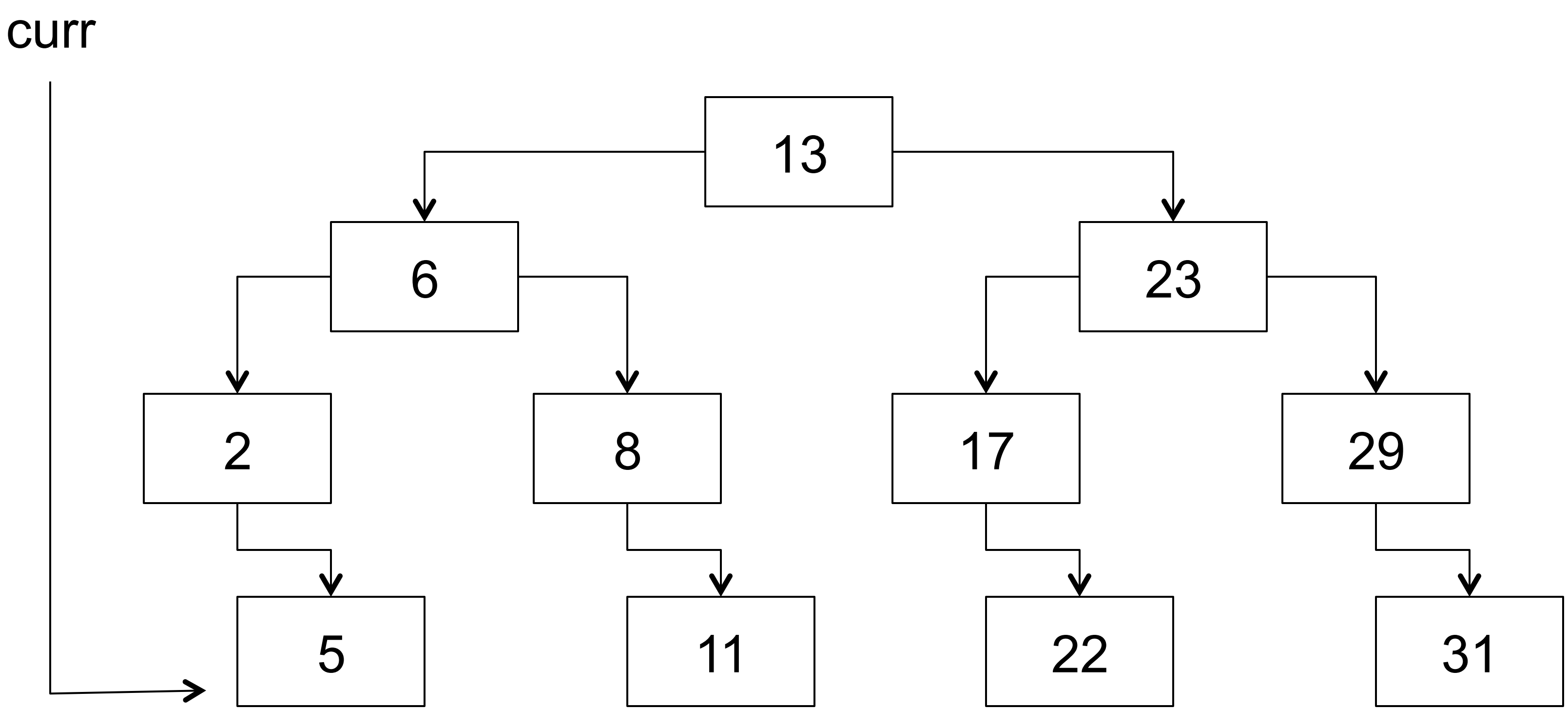
Trees and Contains

- Search for 5
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



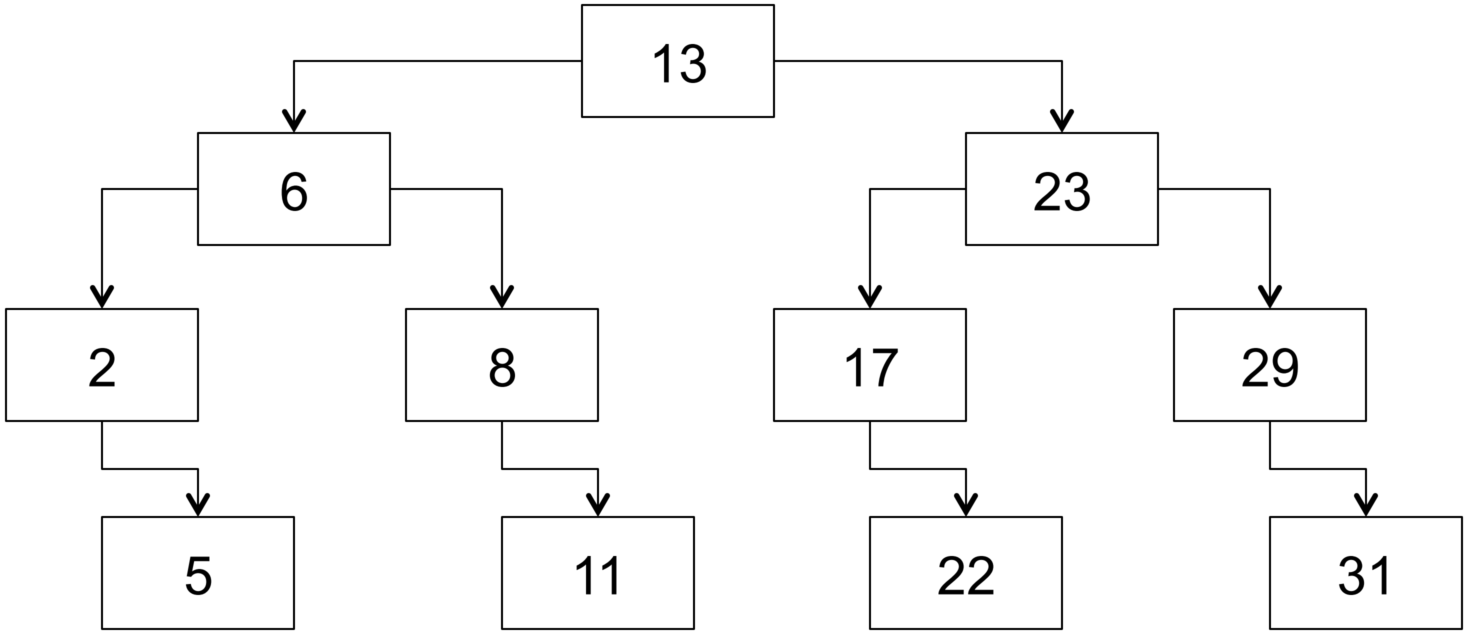
Trees and Contains

- Search for 5
- Start at root:
  - If root is too big, go left (entire right subtree is too big)
  - If root is too small, go right (entire left subtree is too small)



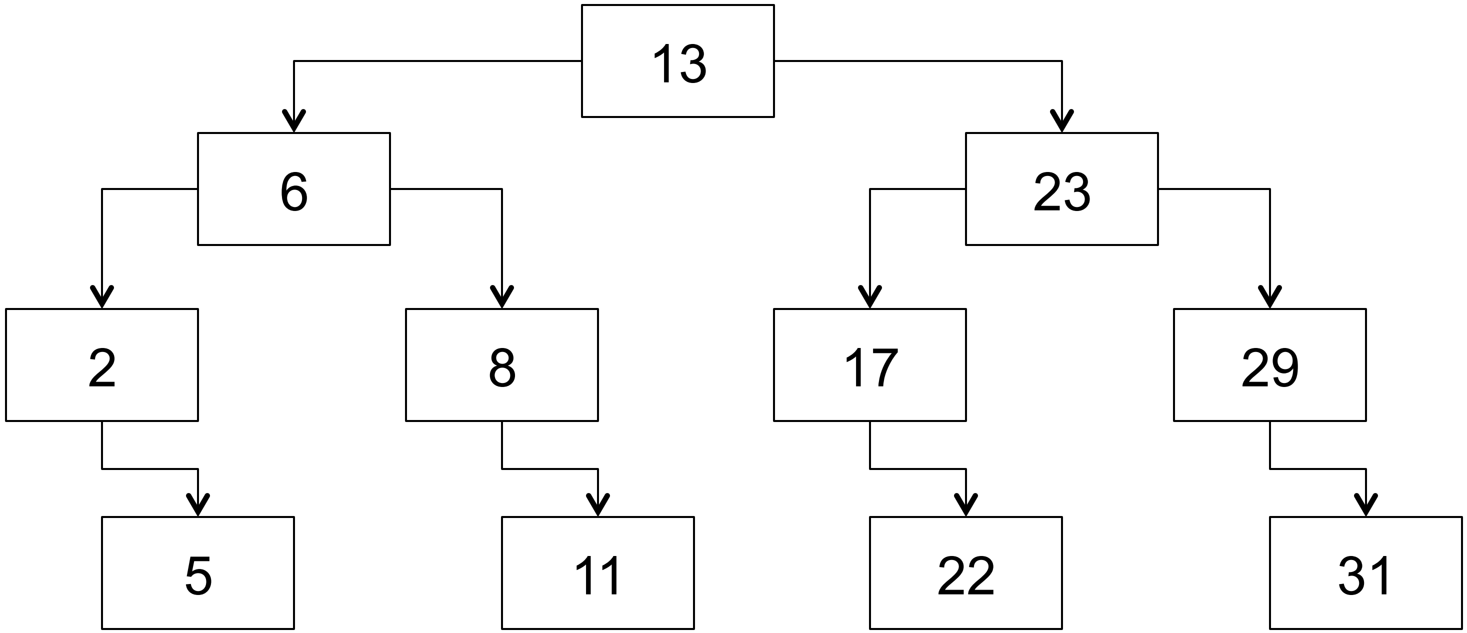
Printing Trees

- We need to be able to print our Set
- How would we print a tree?



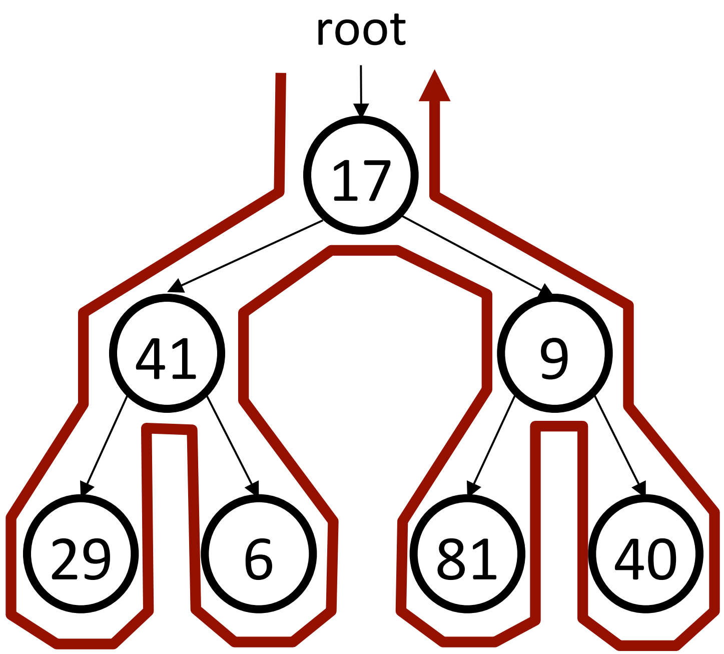
Printing Trees

- How would we print a tree?
  - Idea: need to recurse both left and right
  - **Traverse the tree!**
    - Most tree problems involve traversing the tree



Traversal trick

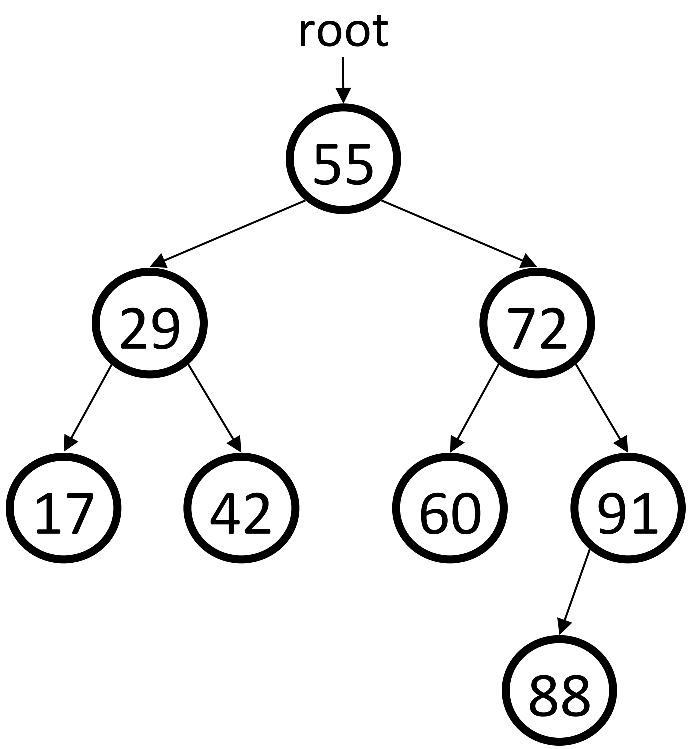
- To quickly generate a traversal:
  - Trace a path counterclockwise.
  - As you pass a node on the proper side, process it.
    - pre-order: left side
    - in-order: bottom
    - post-order: right side
- What kind of traversal does a for-each loop in a Set do?



- pre-order: `17 41 29 6 9 81 40`
- in-order: `29 41 6 17 81 9 40`
- post-order: `29 6 41 81 40 9 17`

getMin/getMax

- Sorted arrays can find the smallest or largest element in O(1) time (how?)
- How could we get the same values in a binary search tree?



Announcements

- Assignment 4 is due **tomorrow**
- Assignment 5 will be released tomorrow
  - More time to complete it, but this assignment will be significantly longer than the others you've seen this quarter
  - As a rough guide, part c took SLs about four times as long to solve as part a, so don't wait until the last minute
- You will get assignment 3 feedback on today
- Please give feedback (if you have the next 30 minutes free): cs198.stanford.edu
- Exam logistics
  - Midterm today, July 25, from 7:00-9:00PM in Hewlett 200

AMA

You've worked hard and have an exam today – you can leave early or stick around to ask me questions ☺