61A Lecture 19

Friday, October 18

Announcements

- Homework 6 is due Tuesday 10/22 @ 11:59pm
  - Includes a mid-semester survey about the course so far
- Project 3 is due Thursday 10/24 @ 11:59pm
- Midterm 2 is on Monday 10/28 7pm-9pm
- Guerrilla section 3 this weekend
  - Object-oriented programming, recursion, and recursive data structures
  - 2pm-5pm on Saturday and 10am-1pm on Sunday
  - Please let us know you are coming by filling out the Piazza poll

Comparing Orders of Growth

Comparing orders of growth (n is the problem size)

$\Theta(b^n)$	Exponential growth! Recursive fib takes $\Theta(\phi^n)$ steps, where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$
$\Theta(n^6)$	Incrementing the problem scales R(n) by a factor.
$\Theta(n^2)$	Quadratic growth. E.g., operations on all pairs. Incrementing n increases R(n) by the problem size n.
$\Theta(n)$	Linear growth. Resources scale with the problem.
$\Theta(\sqrt{n})$	
$\Theta(\log n)$	Logarithmic growth. These processes scale well. Doubling the problem only increments R(n).
$\Theta(1)$	Constant. The problem size doesn't matter.

Sets

Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Sets are unordered, just like dictionary entries

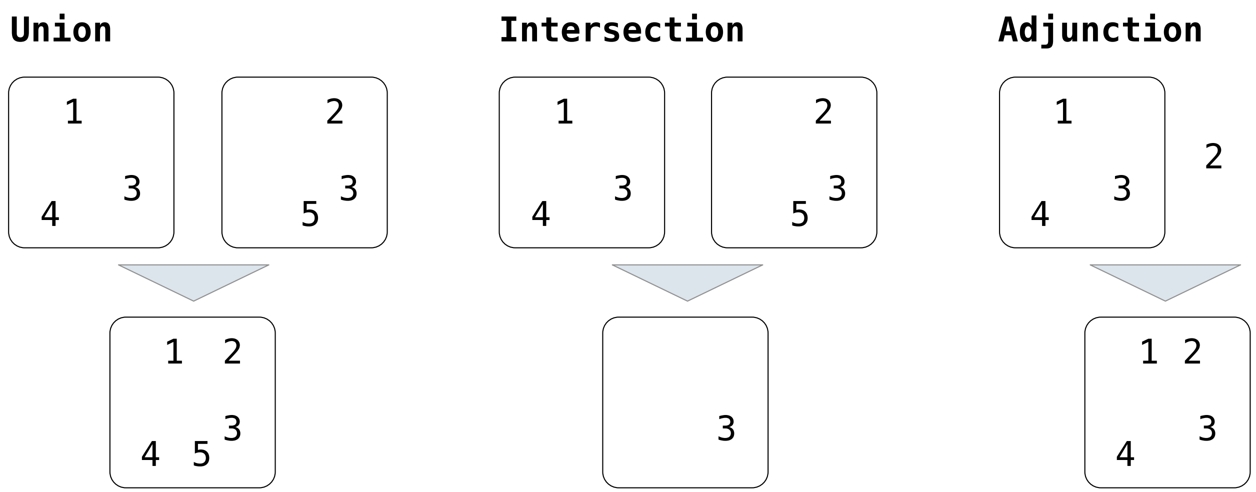
```
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}

>>> 3 in s
True
>>> len(s)
4
>>> s.union({1, 5})
{1, 2, 3, 4, 5}
>>> s.intersection({6, 5, 4, 3})
{3, 4}
```

Implementing Sets

Implementing Sets

What we should be able to do with a set:

- Membership testing: Is a value an element of a set?
- Union: Return a set with all elements in set1 **or** set2
- Intersection: Return a set with any elements in set1 **and** set2
- Adjunction: Return a set with all elements in s and a value v



Sets as Unordered Sequences

Sets as Unordered Sequences

Proposal 1: A set is represented by a recursive list that contains no duplicate items.

```
def empty(s):
    return s is Rlist.empty

def set_contains(s, v):
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return set_contains(s.rest, v)
```

(Demo)

Review: Order of Growth

For a set operation that takes "*linear*" time, we say that

n**:** size of the set

R**(*****n*****):** number of steps required to perform the operation

$$R(n) = \Theta(n)$$

An example f(n)

which means that there are positive constants k_1 and k_2 such that

$$k_1 \cdot n \leq R(n) \leq k_2 \cdot n$$

for sufficiently large values of ***n*****.**

Sets as Unordered Sequences

Time order of growth

```
def adjoin_set(s, v):
    if set_contains(s, v):
        return s
    else:
        return Rlist(v, s)
```

$\Theta(n)$

The size of the set

```
def intersect_set(set1, set2):
    in_set2 = lambda v: set_contains(set2, v)
    return filter_rlist(set1, in_set2)
```

$\Theta(n^2)$

Assume sets are the same size

```
def union_set(set1, set2):
    not_in_set2 = lambda v: not set_contains(set2, v)
    set1_not_set2 = filter_rlist(set1, not_in_set2)
    return extend_rlist(set1_not_set2, set2)
```

$\Theta(n^2)$

(Demo)

Sets as Ordered Sequences

Sets as Ordered Sequences

Proposal 2: A set is represented by a recursive list with unique elements ordered from least to greatest

```
def set_contains(s, v):
    if empty(s) or s.first > v:
        return False
    elif s.first == v:
        return True
    else:
        return set_contains(s.rest, v)
```

Order of growth? $\Theta(n)$

Set Intersection Using Ordered Sequences

This algorithm *assumes* that elements are in order.

```
def intersect_set(set1, set2):
    if empty(set1) or empty(set2):
        return Rlist.empty
    else:
        e1, e2 = set1.first, set2.first
        if e1 == e2:
            return Rlist(e1, intersect_set(set1.rest, set2.rest))
        elif e1 < e2:
            return intersect_set(set1.rest, set2)
        elif e2 < e1:
            return intersect_set(set1, set2.rest)
```

(Demo)

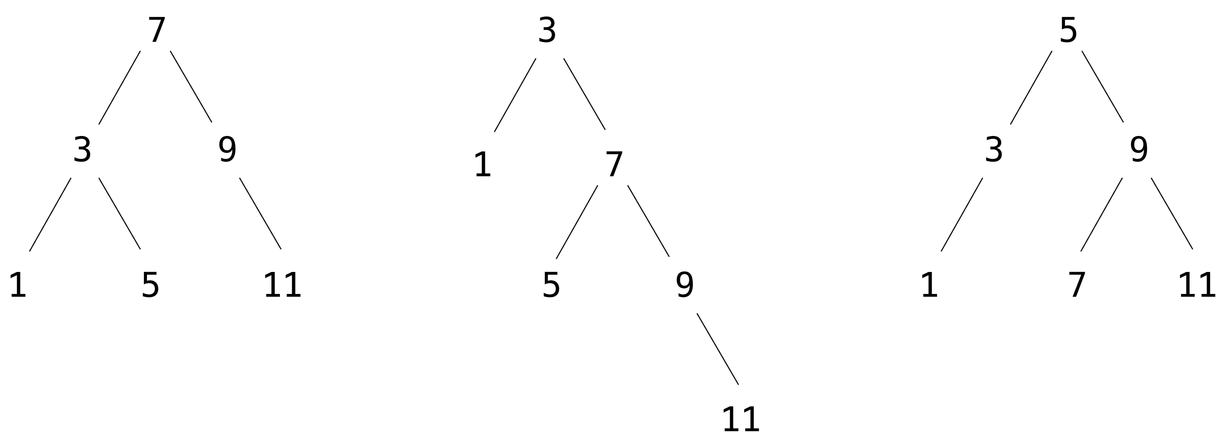
Order of growth? $\Theta(n)$

Sets as Binary Search Trees

Tree Sets

Proposal 3: A set is represented as a Tree. Each entry is:

- Larger than all entries in its left branch and
- Smaller than all entries in its right branch

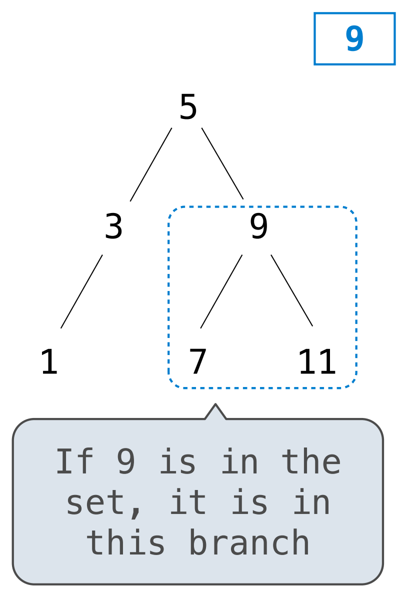


Membership in Tree Sets

Set membership traverses the tree

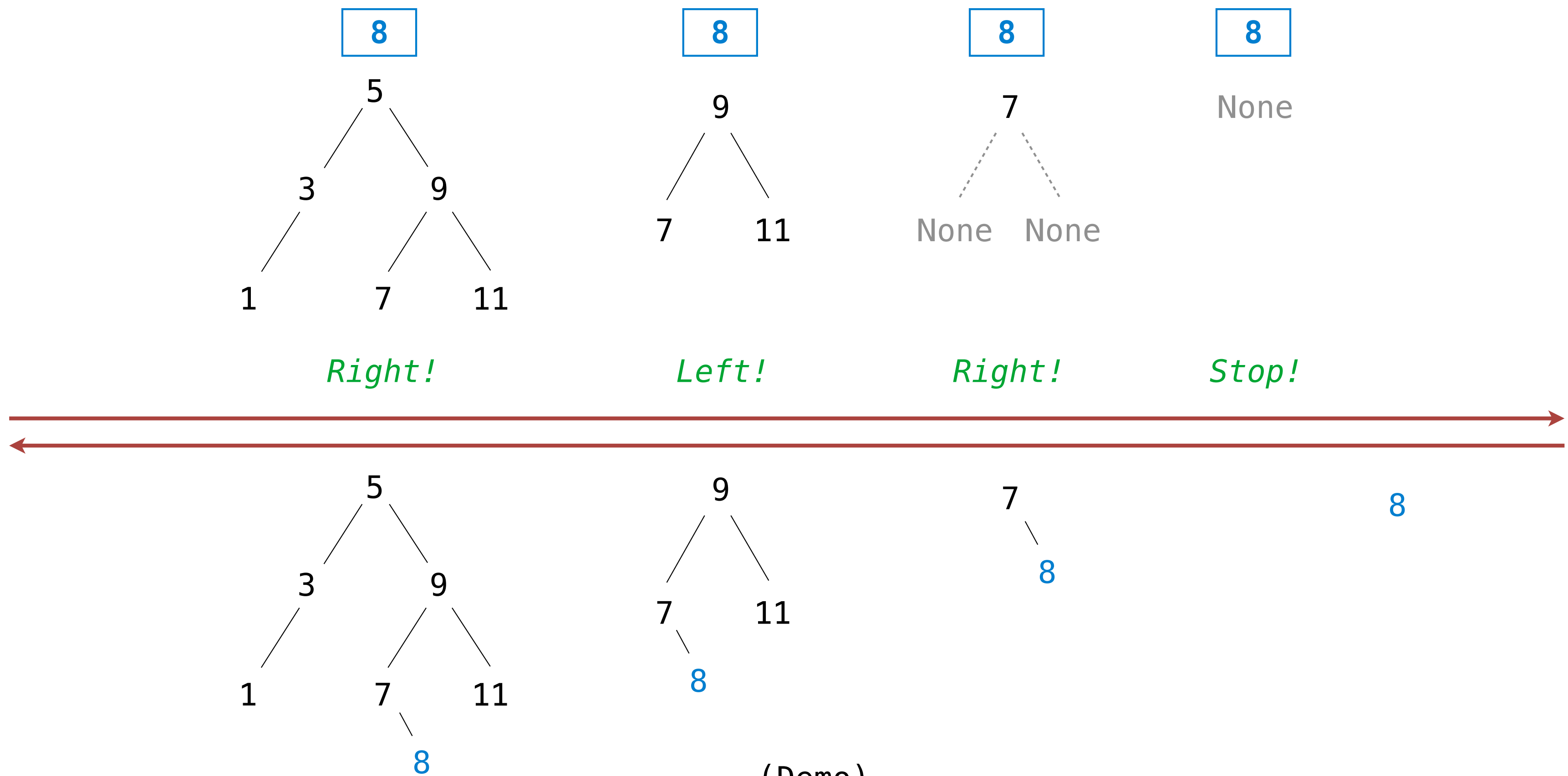
- The element is either in the left or right sub-branch
- By focusing on one branch, we reduce the set by about half

```
def set_contains(s, v):
    if s is None:
        return False
    elif s.entry == v:
        return True
    elif s.entry < v:
        return set_contains(s.right, v)
    elif s.entry > v:
        return set_contains(s.left, v)
```



Order of growth?

Adjoining to a Tree Set



(Demo)

More Set Operations

What Did I Leave Out?

Sets as ordered sequences:

- Adjoining an element to a set
- Union of two sets

Sets as binary trees:

- Intersection of two sets
- Union of two sets
- Balancing a tree

That's all on homework 7!