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)

- $\Theta(2^n)$: Exponential growth! Recursive fib takes $2^n$ steps, where $\phi = \frac{1 + \sqrt{5}}{2} \approx 1.61828$
- $\Theta(n^2)$: Quadratic growth. E.g., operations on all pairs.
- $\Theta(n^{1.65})$: Incrementing the problem scales R(n) by a factor.
- $\Theta(n^2)$: Linear growth. Resources scale with the problem.
- $\Theta(\log n)$: Logarithmic growth. These processes scale well.
- $\Theta(1)$: Constant. The problem size doesn't matter.

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

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

<table>
<thead>
<tr>
<th>Union</th>
<th>Intersection</th>
<th>Adjunction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 3 2 3</td>
<td>1 2 3</td>
</tr>
</tbody>
</table>

### Sets as Unordered Sequences

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

```python
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)
```

#### Review: Order of Growth

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

- \( m \): size of the set
- \( R(n) \): number of steps required to perform the operation

\[
R(n) = \Theta(n) \quad \text{An example f(n)}
\]

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 Ordered Sequences

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

```python
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.

```python
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)
```

Order of growth? $\Theta(n)$

Sets as Binary Search Trees

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

Tree Sets

```plaintext
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

```python
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?

If 9 is in the set, it is in this branch
Adjoining to a Tree Set

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!