Announcements
Sets
Sets

```python
>>> import re
>>> text = \n...   re.split(r'\s+',
...            open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

```python
>>> import re
>>> text = [      # list of words
...     re.split(r'\s+',
...             open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...    if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

- Set literals are enclosed in braces

```python
>>> import re
>>> text = \ # list of words
...   re.split(r'\s+',
...            open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction

```python
>>> import re
>>> text = \n...   re.split(r'\s+',
...            open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

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

```python
>>> import re
>>> text = 
    # list of words
...    re.split(r'\s+',
...             open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

• Set literals are enclosed in braces
• Duplicate elements are removed on construction
• Sets have arbitrary order, just like dictionary entries

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

```python
>>> import re
>>> text = 
...   re.split(r'\s+',
...            open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type
• Set literals are enclosed in braces
• Duplicate elements are removed on construction
• Sets have arbitrary order, just like dictionary entries

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
>>> import re
>>> text = \    # list of words
...    re.split(r'\s+',
...               open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

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

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
>>> 3 in s
True
```

```python
>>> import re
>>> text = re.split(r'\s+', open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type
• Set literals are enclosed in braces
• Duplicate elements are removed on construction
• Sets have arbitrary order, just like dictionary entries

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
>>> 3 in s
True
>>> len(s)
4
```

```python
>>> import re
>>> text = 
...   re.split(r'\s+',
...            open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

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

```python
>>> 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}
```

```python
>>> import re
>>> text = \ # list of words
...    re.split(r'\s+',
...              open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
...     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

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

```python
>>> 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}
```

```python
>>> import re
>>> text = ...
>>> re.split(r'\s+', open('shakespeare.txt').read())
>>> W = set(text)
>>> { w for w in W
>>>     if w == w[-1::-1] and len(w)==5 }
{'madam', 'refer', 'rever', 'minim', 'level'}
```
Sets

One more built-in Python container type

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

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
>>> 3 in s
True
```
Sets

One more built-in Python container type

• Set literals are enclosed in braces
• Duplicate elements are removed on construction
• Sets have arbitrary order, just like dictionary entries

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

(Demo)
Implementing Sets
Implementing Sets

What we should be able to do with a set:
Implementing Sets

What we should be able to do with a set:

- **Membership testing**: Is a value an element of a set?
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
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
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

Union

\[
\begin{array}{cc}
1 & 2 \\
3 & 3 \\
4 & 5 \\
\end{array}
\]

\[
\begin{array}{cc}
1 & 2 \\
3 & 3 \\
4 & 5 \\
\end{array}
\]

\[
\begin{array}{cc}
1 & 2 \\
3 & 3 \\
4 & 5 \\
\end{array}
\]
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
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
- **Adjoin**: Return a set with all elements in s and a value v
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
- **Adjoin**: Return a set with all elements in s and a value v
Sets as Linked Lists
Sets as Unordered Sequences

Proposal 1: A set is represented by a linked list that contains no duplicate items.
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty
```
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """"""Return whether set s contains value v.
    >>> s = Link(1, Link(3, Link(2)))
    >>> contains(s, 2)
    True
    """""""
```
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."

    >>> s = Link(1, Link(3, Link(2)))
    >>> contains(s, 2)
    True
    """

    (Demo)
```
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."""

    >>> s = Link(1, Link(3, Link(2)))
    >>> contains(s, 2)
    True
    """
```

(Demo)
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."
    >>> s = Link(1, Link(3, Link(2))
    >>> contains(s, 2)
    True
    """
```

Time order of growth

Θ(1)
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."

>>> s = Link(1, Link(3, Link(2)))
>>> contains(s, 2)
True

(Demo)
```

Time order of growth

$\Theta(1)$

Time depends on whether & where $v$ appears in $s$. 
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."

>>> s = Link(1, Link(3, Link(2)))
>>> contains(s, 2)
True
"""
```

Time order of growth

\[ \Theta(1) \]

Time depends on whether & where \( v \) appears in \( s \).

\[ \Theta(n) \]
Sets as Unordered Sequences

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

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."

>>> s = Link(1, Link(3, Link(2)))
>>> contains(s, 2)
True
    """
```

Time order of growth

\[ \Theta(1) \]

Time depends on whether & where \( v \) appears in \( s \).

\[ \Theta(n) \]

In the worst case: \( v \) does not appear in \( s \)
or

In the average case: appears in a uniformly distributed random location.
Sets as Unordered Sequences
Sets as Unordered Sequences

def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

Time order of worst-case growth

\( \Theta(n) \)
Sets as Unordered Sequences

def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)
```

Time order of worst-case growth

$\Theta(n)$

The size of the set
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
in_set2 = lambda v: contains(set2, v)
return filter_link(in_set2, set1)
```

Time order of worst-case growth

$\Theta(n)$

The size of the set

Return elements $x$ for which $\text{in\_set2}(x)$ returns a true value
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)
```

Time order of worst-case growth

- \( \Theta(n) \)
- \( \Theta(n^2) \)

The size of the set
Sets as Unordered Sequences

def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)

Return elements x for which in_set2(x) returns a true value

Time order of worst-case growth

\( \Theta(n) \)

The size of the set

\( \Theta(n^2) \)

If sets are the same size
Sets as Unordered Sequences

**Time order of worst-case growth**

\[ \Theta(n) \]

- \( \Theta(n) \) for the size of the set
- \( \Theta(n^2) \) if sets are the same size

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)

def union(set1, set2):
    not_in_set2 = lambda v: not contains(set2, v)
    set1_not_set2 = filter_link(not_in_set2, set1)
    return extend_link(set1_not_set2, set2)
```

Return elements \( x \) for which \( \text{in}_\text{set2}(x) \) returns a true value.
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
in_set2 = lambda v: contains(set2, v)
return filter_link(in_set2, set1)

def union(set1, set2):
    not_in_set2 = lambda v: not contains(set2, v)
    set1_not_set2 = filter_link(not_in_set2, set1)
return extend_link(set1_not_set2, set2)
```

Time order of worst-case growth

- $\Theta(n)$: The size of the set
- $\Theta(n^2)$: If sets are the same size

Return elements x for which in_set2(x) returns a true value

Return a linked list containing all elements in set1_not_set2 followed by all elements in set2
Sets as Unordered Sequences

**Time order of worst-case growth**

\( \Theta(n) \)

The size of the set

\( \Theta(n^2) \)

If sets are the same size

\( \Theta(n^2) \)

Return elements \( x \) for which \( \text{in\_set2}(x) \) returns a true value

Return a linked list containing all elements in \( \text{set1\_not\_set2} \) followed by all elements in \( \text{set2} \)

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)

def union(set1, set2):
    not_in_set2 = lambda v: not contains(set2, v)
    set1_not_set2 = filter_link(not_in_set2, set1)
    return extend_link(set1_not_set2, set2)
```
Sets as Ordered Linked Lists
Sets as Ordered Sequences

Proposal 2: A set is represented by a linked list with unique elements that is ordered from least to greatest
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td></td>
</tr>
</tbody>
</table>
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
</tbody>
</table>
## Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Sets as Ordered Sequences**

**Proposal 2:** A set is represented by a linked list with unique elements that is \textit{ordered from least to greatest}

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td>Ordered linked lists</td>
<td></td>
</tr>
</tbody>
</table>
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td>Ordered linked lists</td>
<td>first, rest, &lt;, &gt;, ==</td>
</tr>
</tbody>
</table>
**Sets as Ordered Sequences**

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td>Ordered linked lists</td>
<td>first, rest, &lt;, &gt;, ==</td>
</tr>
</tbody>
</table>
Sets as Ordered Sequences

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

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td>Ordered linked lists</td>
<td>first, rest, &lt;, &gt;, ==</td>
</tr>
</tbody>
</table>

Different parts of a program may make different assumptions about data.
Searching an Ordered List
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
```
Searching an Ordered List

>>> s = Link(1, Link(3, Link(5)))

Operation
Time order of growth

```
    Link instance
s:  
    first:  1
    rest:   

    Link instance
     first:  3
     rest:   

    Link instance
     first:  5
     rest:   
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
```

Operation: contains

Time order of growth:

```
s:   Link instance
     first: 1
     rest: 

     Link instance
     first: 3
     rest: 

     Link instance
     first: 5
     rest: 
```
Searching an Ordered List

>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>contains</td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
```

```python
Link instance

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

```python
Link instance

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

```python
Link instance

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

```python
Link instance

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td></td>
</tr>
</tbody>
</table>

```
>>> contains(s, 1)
True
>>> contains(s, 2)
```
Searching an Ordered List

>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td></td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td></td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

```
<table>
<thead>
<tr>
<th>Link instance</th>
<th>Link instance</th>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 1</td>
<td>first: 3</td>
<td>first: 5</td>
</tr>
<tr>
<td>rest:</td>
<td>rest:</td>
<td>rest:</td>
</tr>
</tbody>
</table>
```

```
```
Searching an Ordered List

>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)

Operation | Time order of growth
----------|-----------------------
contains   | $\Theta(n)$
adjoin     |
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td></td>
</tr>
</tbody>
</table>

```
s = Link(1, Link(3, Link(5)))
contains(s, 1)
contains(s, 2)
t = adjoin(s, 2)
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td></td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

```
Link instance
first: 1
rest: Link instance
first: 2
rest: Link instance
first: 3
rest: Link instance
first: 5
```

```
s: Link instance
first: 1
rest: Link instance
first: 2
```

```
t: Link instance
first: 1
rest: Link instance
```
Searching an Ordered List

>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False

>>> t = adjoin(s, 2)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>(\Theta(n))</td>
</tr>
<tr>
<td>adjoin</td>
<td>(\Theta(n))</td>
</tr>
</tbody>
</table>

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False

>>> t = adjoin(s, 2)
```

```
Link instance
first: 1
rest: 
```

```
Link instance
first: 3
rest: 
```

```
Link instance
first: 5
rest: 
```

```
Link instance
first: 1
rest: 
```

```
Link instance
first: 2
rest: 
```

```
Link instance
first: 1
rest: 
```

```
Link instance
first: 2
rest: 
```

```
Link instance
first: 3
rest: 
```

```
Link instance
first: 5
rest: 
```
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```
Set Operations
Intersecting Ordered Linked Lists

Proposal 2: A set is represented by a linked list with unique elements that is \textit{ordered from least to greatest}

Let $n$ be max of set1, set2 size
Intersecting Ordered Linked Lists

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

def intersect(set1, set2):
    Let $n$ be max of set1, set2 size
Intersecting Ordered Linked Lists

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

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
```
Intersecting Ordered Linked Lists

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

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        # Let n be max of set1, set2 size
```

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

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        e1, e2 = set1.first, set2.first
```

Let n be max of set1, set2 size
Intersecting Ordered Linked Lists

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

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        e1, e2 = set1.first, set2.first
        if e1 == e2:
            return Link(e1, intersect(set1.rest, set2.rest))
```

Let $n$ be max of set1, set2 size
Intersecting Ordered Linked Lists

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

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

Let $n$ be max of set1, set2 size
**Intersecting Ordered Linked Lists**

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

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

Let $n$ be max of set1, set2 size
Intersecting Ordered Linked Lists

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

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

Let $n$ be max of set1, set2 size

Order of growth?
**Intersecting Ordered Linked Lists**

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

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

Order of growth? \( \Theta(n) \)
```

Let \( n \) be max of set1, set2 size
Intersecting Ordered Linked Lists

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

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

Order of growth? $\Theta(n)$

Let $n$ be max of set1, set2 size (Demo)
Set Mutation
## Adding to an Ordered List

<table>
<thead>
<tr>
<th>Link instance</th>
<th>first:</th>
<th>rest:</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Link instance</th>
<th>first:</th>
<th>rest:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Link instance</th>
<th>first:</th>
<th>rest:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Adding to an Ordered List

\[
\text{s: } \text{Link instance}
\begin{array}{|c|}
\hline
\text{first:} & 1 \\
\hline
\text{rest:} & \text{ } \\
\hline
\end{array}
\]

\[
\text{Link instance}
\begin{array}{|c|}
\hline
\text{first:} & 3 \\
\hline
\text{rest:} & \text{ } \\
\hline
\end{array}
\]

\[
\text{Link instance}
\begin{array}{|c|}
\hline
\text{first:} & 5 \\
\hline
\text{rest:} & \text{ } \\
\hline
\end{array}
\]

\[
\text{add(s, } 0 \text{)} \quad \text{Try to return the same object as input}
\]
Adding to an Ordered List

```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 1</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>

s:

```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 0</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 3</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 5</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>
```
Adding to an Ordered List

\[
\text{add}(s, 3)
\]
Adding to an Ordered List

```
Link instance
first: 0
rest:

Link instance
first: 1
rest:

Link instance
first: 3
rest:

Link instance
first: 5
rest:

add(s, 3)
add(s, 4)
```
Adding to an Ordered List

- **s:**
  - **Link** instance
    - first: 0
    - rest: 
  - **Link** instance
    - first: 1
    - rest: 
  - **Link** instance
    - first: 3
    - rest: 

- **Link** instance
  - first: 4
  - rest: 
  - **Link** instance
    - first: 5
    - rest: 

- **Link** instance
Adding to an Ordered List

```
add(s, 6)
```
Adding to an Ordered List

s:

- **Link** instance
  - first: 0
  - rest: 

- **Link** instance
  - first: 1
  - rest: 

- **Link** instance
  - first: 3
  - rest: 

- **Link** instance
  - first: 4
  - rest: 

- **Link** instance
  - first: 5
  - rest: 

- **Link** instance
  - first: 6
  - rest: 

The diagram illustrates the process of adding elements to an ordered list.
Adding to a Set Represented as an Ordered List

S:

<table>
<thead>
<tr>
<th>Link instance</th>
<th>Link instance</th>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 0</td>
<td>first: 3</td>
<td>first: 4</td>
</tr>
<tr>
<td>rest:</td>
<td>rest:</td>
<td>rest:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Link instance</th>
<th>Link instance</th>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 1</td>
<td>first: 5</td>
<td>first: 6</td>
</tr>
<tr>
<td>rest:</td>
<td>rest:</td>
<td>rest:</td>
</tr>
</tbody>
</table>
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
```

![Diagram of adding elements to an ordered list](image_url)
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to a set s, returning modified s."""

Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""

    s = Link(1, Link(3, Link(5))))

>>> s = Link(1, Link(3, Link(5)))
```

...
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to a set s, returning modified s."
    ""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5)))))
def add(s, v):
    """Add v to a set s, returning modified s."""

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5)))))

Adding to a Set Represented as an Ordered List
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to a set s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""
    s = Link(1, Link(3, Link(5)))
    add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5))))))
    add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
```

```python
def add(s, v):
    """Add v to a set s, returning modified s."""
    s = Link(1, Link(3, Link(5)))
    s = Link(0, Link(1, Link(3, Link(5))))
    add(s, 0)
    s = Link(0, Link(1, Link(3, Link(5))))
    add(s, 3)
    s = Link(0, Link(1, Link(3, Link(5))))
    add(s, 4)
    s = Link(0, Link(1, Link(3, Link(4, Link(5))))))
    add(s, 6)
    s = Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
```
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    # Add v to a set s, returning modified s."

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

    if empty(s): return Link(v)
```
def add(s, v):
    """Add v to a set s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

    if s.first > v:
        s.first, s.rest = ___________________________ , ___________________________
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

    if s.first > v:
        s.first, s.rest = ____________________________ , ____________________________
    elif s.first < v and empty(s.rest):
        s.rest = ________________________________
```

```python
if s.first > v:
    s.first, s.rest = ____________________________ , ____________________________
elif s.first < v and empty(s.rest):
    s.rest = ________________________________
```
Adding to a Set Represented as an Ordered List

def add(s, v):
    '''Add v to a set s, returning modified s.'''

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = _____________________________________________________________
    elif s.first < v:
        _________________________________________________________________

    return s
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""
    if empty(s): return Link(v)
    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = __________________________________________________________________
    elif s.first < v:
        __________________________________________________________________________
    return s

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 4)
Link(0, Link(1, Link(3, Link(4, Link(5)))))
>>> add(s, 6)
Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))
```

if s.first > v:
    s.first, s.rest = __________________________ , __________________________
elif s.first < v and empty(s.rest):
    s.rest = __________________________________________________________________
elif s.first < v:
    __________________________________________________________________________
return s
def add(s, v):
    """Add v to a set s, returning modified s."""
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = ___________________________________________________________________
    elif s.first < v:
        _______________________________________________________________
    return s
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to a set s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

    if s.first > v:
        s.first, s.rest = __________________________ , ___________________________
    elif s.first < v and empty(s.rest):
        s.rest = ___________________________
    elif s.first < v:
        ___________________________
    return s
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""

    >>>
    s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))
    
    if s.first > v:
        s.first, s.rest = v, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = Link(v, s.rest)
    elif s.first < v:
        add(s.rest, v)
    return s
```
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set s, returning modified s."""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    
    if s.first > v:
        s.first, s.rest = v, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = Link(v, s.rest)
    elif s.first < v:
        add(s.rest, v)
    return s
```