Ordered Sets
Announcements
Sets
Sets
Sets

One more built-in Python container type
Sets

One more built-in Python container type

- Set literals are enclosed in braces
Sets

One more built-in Python container type

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

    >>> s = {'one', 'two', 'three', 'four', 'four'}
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

{'three', 'one', 'four', 'two'}
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

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
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 = {'one', 'two', 'three', 'four', 'four'}
s = {'three', 'one', 'four', 'two'}
'three' in s
len(s)
s.union({'one', 'five'})
```
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

{ 'three', 'four' }
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

```
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```
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

```
Union:

1  3
4  3

2  3
5  3

1  2
4  3
```
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

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

**Union**

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

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

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

**Intersection**

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

\[
\begin{array}{c}
3 \\
\end{array}
\]

\[
\begin{array}{c}
3 \\
\end{array}
\]
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
- **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
```

(Demo)
Sets as Unordered Sequences

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

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

- \( \Theta(1) \)

- \( \Theta(n) \)
  - In the worst case: \( v \) does not appear in \( s \)
  - In the average case: \( v \) appears in a uniformly distributed random location

(Demo)
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

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

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(s, t):
    if s is Link.empty:
        return Link.empty
    rest = __________________________
    if contains(t, s.first):
        return __________________________
    else:
        return rest
```

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(s, t):
    if s is Link.empty:
        return Link.empty
    rest = __________________________
    if contains(t, s.first):
        return ______________________
    else:
        return rest
```

Time order of worst-case growth

$\Theta(n)$

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(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest

Time order of worst-case growth

$\Theta(n)$

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(s, t):
    if s is Link.empty:
        return Link.empty
    rest = __________________________
    if contains(t, s.first):
        return ________________________
    else:
        return rest
```

Time order of worst-case growth

\[ \Theta(n) \]

The size of the set

\[ \Theta(n^2) \]
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

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

def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest
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>
</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 order\_ed 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 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,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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)))

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

```
<table>
<thead>
<tr>
<th>s:</th>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first:</td>
<td>1</td>
</tr>
<tr>
<td>rest:</td>
<td></td>
</tr>
</tbody>
</table>
```
```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first:</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>
```
```
<table>
<thead>
<tr>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first:</td>
</tr>
<tr>
<td>rest:</td>
</tr>
</tbody>
</table>
```
Searching an Ordered List

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td></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: 3</td>
<td>first: 5</td>
</tr>
<tr>
<td>rest:</td>
<td>rest:</td>
<td>rest:</td>
</tr>
</tbody>
</table>
```
Searching an Ordered List

```python
>>> 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></td>
</tr>
</tbody>
</table>

```
contains(s, 1)
```
Searching an Ordered List

>>> 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>contains</td>
</tr>
</tbody>
</table>

```
Link instance

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

```
Link instance

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

```
Link instance

<table>
<thead>
<tr>
<th>first:</th>
<th>rest:</th>
</tr>
</thead>
<tbody>
<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
>>> contains(s, 2)
False
```

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

![Link instance diagram](image)
Searching an Ordered List

>>> 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>

\[
\text{contains}(s, 1) = \text{True}
\]

\[
\text{contains}(s, 2) = \text{False}
\]
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>
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>Θ(n)</td>
</tr>
<tr>
<td>adjoin</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing the structure of ordered lists](image)

```python
s:

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

t:

<table>
<thead>
<tr>
<th>first:</th>
<th>rest:</th>
</tr>
</thead>
<tbody>
<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
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)

Operation | Time order of growth
----------|---------------------
contains   | \( \Theta(n) \)
adjoin     |
```

```
<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></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)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>Θ(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)
```
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>Θ(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)
```
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>

---

**Link instance**

- **First**: 1
- **Rest**:

---

**Link instance**

- **First**: 3
- **Rest**:

---

**Link instance**

- **First**: 5
- **Rest**:

---

**Link instance**

- **First**: 2
- **Rest**:

---

**Link instance**

- **First**: 1
- **Rest**:

---
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     | $\Theta(n)$

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

![Diagram of Linked List Operations]
Set Operations
Intersecting Ordered Linked Lists

Proposal 2: A set is represented by a linked list with unique elements that is ordered from least to greatest.
**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(s, t):
```
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(s, t):
    if empty(s) or empty(t):
        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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
```
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
```
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
```
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Order of growth? If s and t are sets of size n, then
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Order of growth? If s and t are sets of size n, then $\Theta(n)$
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(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Order of growth? If s and t are sets of size n, then $\Theta(n)$ (Demo)
Set Mutation
Adding to an Ordered List

s: Link instance
first: 1
rest: 

Link instance
first: 3
rest: 

Link instance
first: 5
rest: 

Link instance
first: 
rest: 

Link instance
first: 
rest: 

Link instance
first: 
rest: 

Adding to an Ordered List

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

1. Add a new list element to the ordered list.

   \[ \text{first: 1} \quad \rightarrow \quad \text{rest: } \]

2. The new element becomes the new first element of the list.

   \[ \text{first: 1} \quad \rightarrow \quad \text{rest: } \]

3. The next element in the list is added to the new first element.

   \[ \text{first: 3} \quad \rightarrow \quad \text{rest: } \]

4. This process continues for each element to be added.

   \[ \text{first: 5} \quad \rightarrow \quad \text{rest: } \]
Adding to an Ordered List

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

\[
\text{add}(s, 3) \\
\text{add}(s, 4)
\]
Adding to an Ordered List

s:

1. Link instance
   - first: 0
   - rest:

2. Link instance
   - first: 1
   - rest:

3. Link instance
   - first: 3
   - rest:

4. Link instance
   - first: 4
   - rest:

5. Link instance
   - first: 5
   - rest:
Adding to an Ordered List

s:

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

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

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

```
<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

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

add(s, 6)
Adding to an Ordered List

1. First: 0
   Rest:

2. First: 3
   Rest:

3. First: 4
   Rest:

4. First: 5
   Rest:

5. First: 6
   Rest:
Adding to a Set Represented as an Ordered List

\[ S := \begin{array}{c|c}
\text{Link instance} & \\
\hline
\text{first} & 0 \\
\text{rest} & \\
\end{array} \quad \begin{array}{c|c}
\text{Link instance} & \\
\hline
\text{first} & 3 \\
\text{rest} & \\
\end{array} \quad \begin{array}{c|c}
\text{Link instance} & \\
\hline
\text{first} & 4 \\
\text{rest} & \\
\end{array} \quad \begin{array}{c|c}
\text{Link instance} & \\
\hline
\text{first} & 5 \\
\text{rest} & \\
\end{array} \quad \begin{array}{c|c}
\text{Link instance} & \\
\hline
\text{first} & 6 \\
\text{rest} & \\
\end{array} \]
Adding to a Set Represented as an Ordered List

def add(s, v):

```python
s:

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

```
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)))
```

>> $s = \text{Link(1, Link(3, Link(5)))}$
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)))))
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)))))
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)))))
```

---

![Diagram showing the process of adding elements to a list](image.png)
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)))))

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)
    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)
```
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 = __________________________ , ____________________________
```

```python
>>> 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
if s.first > v:
    s.first, 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 = _____________________________

    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:  

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 = ___________________________ , ___________________________
    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."""

    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

```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 = ________________
    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
```

```
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
    elif s.first < v and empty(s.rest):
        s.rest = ___________________________________________________________________
        Link(v, s.rest)
    elif s.first < v:
        _______________________________________________________________________
        add(s.rest, v)
    return s

if empty(s): return Link(v)