61A Lecture 20
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

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

![Union Diagram]

1  
4 3

2  
5 3

1 2
4 5 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
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

\[
\begin{align*}
\text{Union} & : 1 & 2 \\
& 3 & 5 & 3 \\
\text{Intersection} & : 1 & 2 \\
& 3 & 5 & 3 \\
\end{align*}
\]
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

\[
\begin{array}{c|c}
\text{Union} & \text{Intersection} \\
\hline
1 & 1 \\
4 & 4 \\
3 & 3 \\
2 & 2 \\
5 & 5 \\
3 & 3 \\
\end{array}
\]
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
    """
```

Time order of growth

\[ \Theta(1) \]
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

- $\Theta(1)$
- $\Theta(n)$

Time depends on whether

- $\Theta(n)$
- Assuming $v$ either does not appear in $s$ or appears in a uniformly distributed random location
Sets as Unordered Sequences
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)
```
Sets as Unordered Sequences

\[ \text{Time order of growth} \quad \Theta(n) \]

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

\[
\text{def } \text{adjoin}(s, v):
    \text{if } \text{contains}(s, v):
        \text{return } s
    \text{else:}
        \text{return } \text{Link}(v, s)
\]
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)
Sets as Unordered Sequences

Time order of growth

\[ \Theta(n) \]

The size of the set

```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)

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

Return elements x for which 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)
```

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

Time order of growth

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

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 growth

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

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 growth

Θ(n)

The size of the set

Θ(n^2)

If sets are the same size
Sets as Unordered Sequences

\[
\Theta(n)
\]

Time order of growth

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

```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)
```

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 \(\text{set2}\).
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 growth

- \( \Theta(n) \)
  - The size of the set

- \( \Theta(n^2) \)
  - If sets are the same size

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 \( \text{set2} \)
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.
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>
</tbody>
</table>

Implement set operations
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>
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

```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>first:</td>
<td>1</td>
</tr>
<tr>
<td>rest:</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>

```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>
<tbody>
<tr>
<td>contains</td>
<td>s = Link(1, Link(3, Link(5)))</td>
</tr>
</tbody>
</table>

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

```
Link instance

s:

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

Link instance

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

Link instance

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

Diagram:
- s: Link instance
  - 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))
>>> 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>

```
s:
Link instance

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

```
Link instance

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

```
Link instance

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

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

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

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

```
Link instance
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)))
>>> 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>
Searching an Ordered List

```python
>>> 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 instance
  - `first`: 1
  - `rest`: (Link instance)
  - `first`: 3
  - `rest`: (Link instance)
  - `first`: 5
  - `rest`: (Link instance)
### 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>

**Link instance**

- **s**:
  - `first`: 1
  - `rest`: 

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

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

\[s = \text{Link}(1, \text{Link}(3, \text{Link}(5)))\]
\[\text{contains}(s, 1)\]
\[\text{contains}(s, 2)\]
\[\text{contains}(s, 1)\]
Searching an Ordered List

```python
given 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>
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></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>

```
>>> 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: 3
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></td>
</tr>
</tbody>
</table>

---

```
>>> t = adjoin(s, 2)
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> contains(t, 1)
True
>>> contains(t, 2)
True
>>> contains(t, 3)
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>$\Theta(n)$</td>
</tr>
</tbody>
</table>

\[
\text{s: first: 1} \quad \text{t: first: 2} \\
\text{rest: Link(3, Link(5))} \quad \text{rest: Link(3, Link(5))}
\]

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first: 1} \\
\text{rest:} \\
\hline
\end{array} \\
\begin{array}{|c|}
\hline
\text{first: 2} \\
\text{rest:} \\
\hline
\end{array}
\]
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>

\[
\text{Demo}
\]

```
s:
  \textbf{Link} instance
  \begin{tabular}{|l|}
  \hline
  first: & 1 \\
  \hline
  rest: & \\
  \hline
  \end{tabular}

rest: \\
```

```
t:
  \textbf{Link} instance
  \begin{tabular}{|l|}
  \hline
  first: & 1 \\
  \hline
  rest: & \\
  \hline
  \end{tabular}

rest: \\
```

```

\[
\text{Table demo}
\]

```

```
```

```
```

```
```
Set Mutation
Adding to an Ordered List

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

Diagram:

- First instance: first: 1, rest: Link instance
- Second instance: first: 3, rest: Link instance
- Third instance: first: 5, rest: (null)
Adding to an Ordered List

$s$: Link instance

first: 1
rest: 

add($s$, 0)

Link instance

first: 3
rest: 

Link instance

first: 5
rest: 

Link instance

first: 
rest: 
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>
```

```
<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>
```

s:
Adding to an Ordered List

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

```
Link instance
first: 0
rest: 

1

3

Link instance
first: 3
rest: 

5

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

<table>
<thead>
<tr>
<th>Link instance</th>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\times 0)</td>
<td></td>
</tr>
<tr>
<td>s:</td>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\times 4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[5]</td>
<td></td>
</tr>
</tbody>
</table>
Adding to an Ordered List

`s:`

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

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

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

add(s, 6)
Adding to an Ordered List

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first: } 0 \\
\text{rest: } \quad \text{Link instance} \\
\end{array}
\]

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

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first: } 4 \\
\text{rest: } \quad \text{Link instance} \\
\end{array}
\]

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

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first: } 6 \\
\text{rest: } \\
\end{array}
\]
Adding to an Ordered List
Adding to an Ordered List

def add(s, v):

```

```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and return s."""
```
Adding to an Ordered List

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

```
```
Adding to an Ordered List

def add(s, v):
    """Add v to a set s and return s."

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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))))
```

Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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)))))
```
Adding to an Ordered List

def add(s, v):
    """Add v to a set s and 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)))))
    >>>
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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))))))
    assert not empty(s), "Cannot add to an empty set."
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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 = ______________________________ , _____________________________
```

Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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 = __________________________
```

Adding to an Ordered List

def add(s, v):
    """Add v to a set s and 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:
        _________________________________
Adding to an Ordered List

def add(s, v):
    """Add v to a set s and 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
Adding to an Ordered List

def add(s, v):
    """Add v to a set s and 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
Adding to an Ordered List

def add(s, v):
    """Add v to a set s and return s."""
    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))))))
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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
```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and 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:
        _______ v
        s.first, s.rest = __________________________, __________________________
    elif s.first < v and empty(s.rest):
        _______ v
        s.rest = __________________________
    elif s.first < v:
        _______ v
        __________________________
    return s
```

---

**Diagram:**

- **s:**
  - 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:
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(set1, set2):
```

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

def intersect(set1, set2):
    if empty(set1) or empty(set2):
        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

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

Order of growth? $\Theta(n)$
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(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? $\Theta(n)$ (Demo)