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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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:

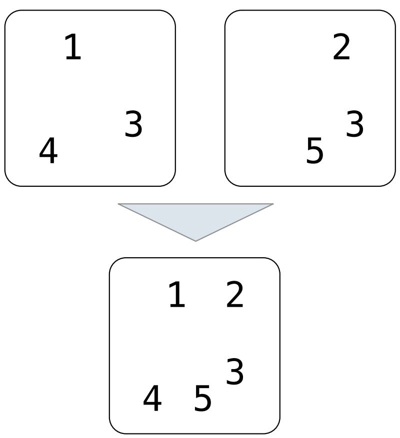
- **Membership testing:** Is a value an element of a set?
- **Union:** Return a set with all elements in set1 or set2

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

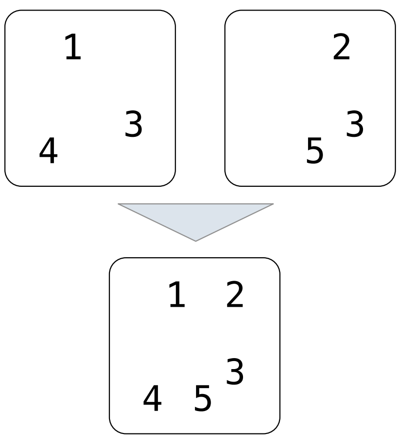


Implementing Sets

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

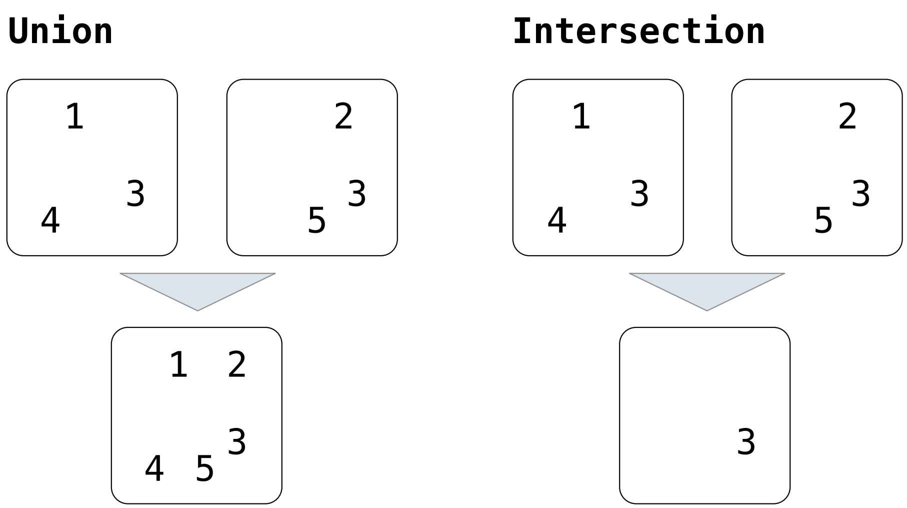
Union



Implementing Sets

What we should be able to do with a set:

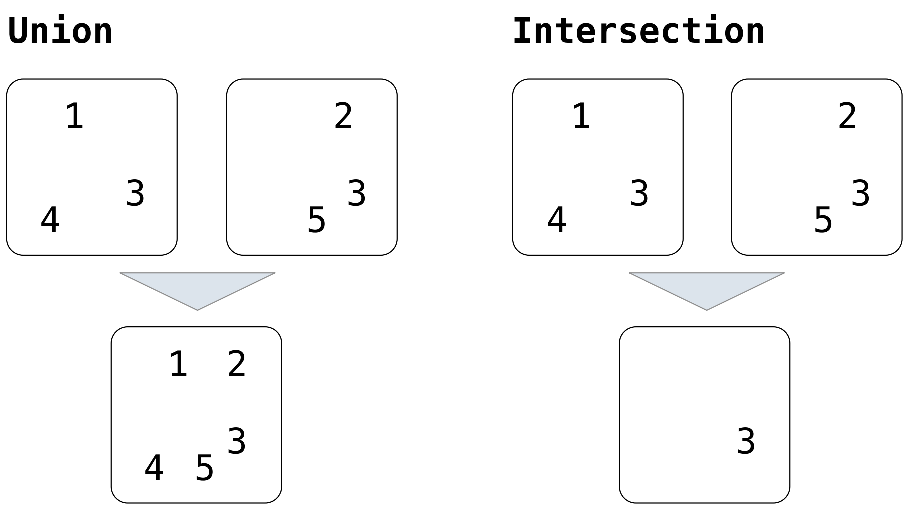
- **Membership testing:** Is a value an element of a set?
- **Union:** Return a set with all elements in set1 or set2
- **Intersection:** Return a set with any elements in set1 and set2



Implementing Sets

What we should be able to do with a set:

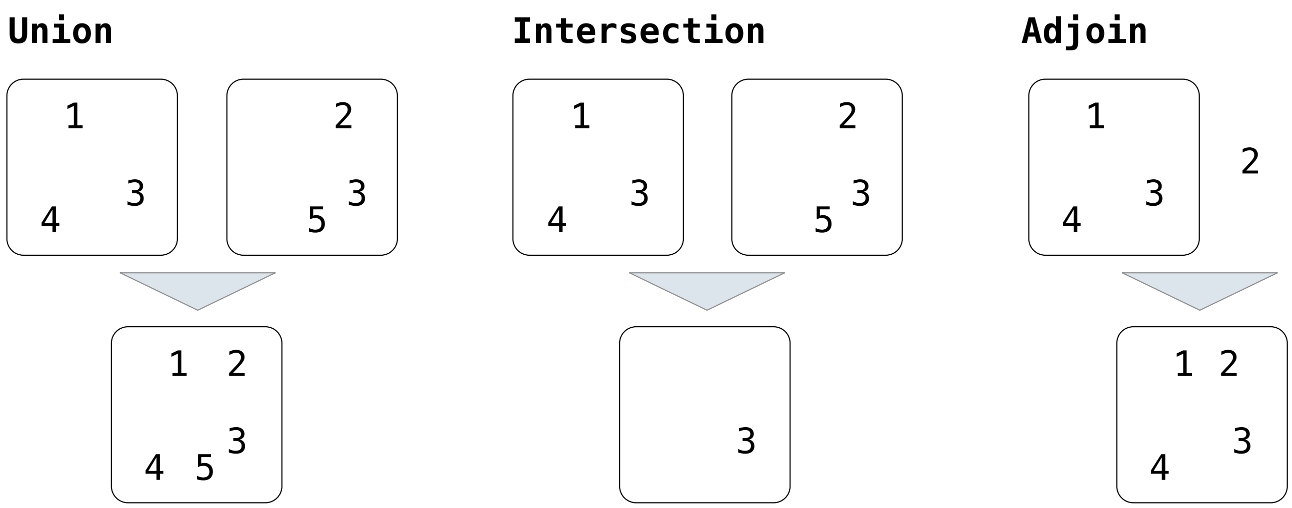
- **Membership testing:** Is a value an element of a set?
- **Union:** Return a set with all elements in set1 or set2
- **Intersection:** Return a set with any elements in set1 and set2
- **Adjoin:** Return a set with all elements in s and a value v



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.

```
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.

```
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.

```
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.

Time order of growth

```
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.

Time order of growth

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

$\Theta(1)$

```
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.

Time order of growth

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

$\Theta(1)$

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

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

Time depends on whether & where v appears in s

(Demo)

Sets as Unordered Sequences

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

Time order of growth

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

$\Theta(1)$

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

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

Time depends on whether & where v appears in s

$\Theta(n)$

(Demo)

Sets as Unordered Sequences

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

Time order of growth

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

$\Theta(1)$

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

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

Time depends on whether & where v appears in s

$\Theta(n)$

Assuming v either does not appear in s ***or*** *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

Time order of growth

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

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

Sets as Unordered Sequences

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

Time order of 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(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

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

$\Theta(n^2)$

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

$\Theta(n^2)$

If sets are the same size

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

$\Theta(n^2)$

If sets are the same size

```
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 Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

$\Theta(n^2)$

If sets are the same size

```
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 a linked list containing all elements in set1_not_set2 followed by all elements in set2

Sets as Unordered Sequences

Time order of growth

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

$\Theta(n)$

The size of the set

```
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

$\Theta(n^2)$

If sets are the same size

```
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)
```

$\Theta(n^2)$

Return a linked list containing all elements in set1_not_set2 followed by all elements in 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***

Parts of the program that...	Assume that sets are...	Using...

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values		

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
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***

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
Implement set operations	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***

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
Implement set operations	Ordered linked lists	first, rest, <, >, ==

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
Implement set operations	Ordered linked lists	first, rest, <, >, ==

Sets as Ordered Sequences

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

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
Implement set operations	Ordered linked lists	first, rest, <, >, ==

Different parts of a program may make different assumptions about data

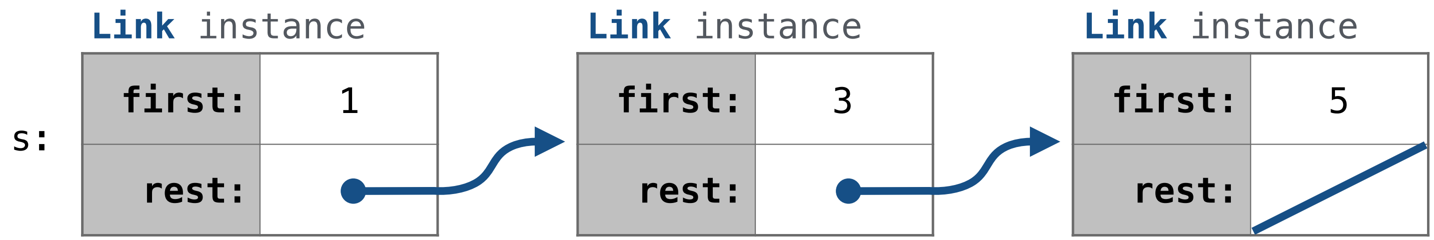
Searching an Ordered List

Searching an Ordered List

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

Searching an Ordered List

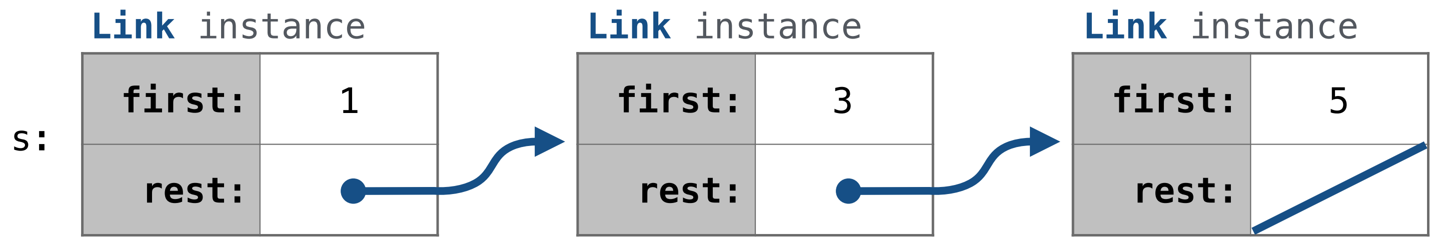
```
>>> s = Link(1, Link(3, Link(5)))
```



Searching an Ordered List

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

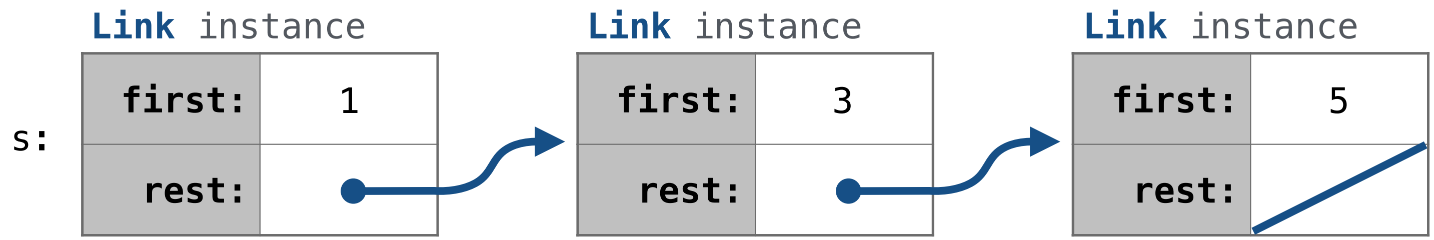
Operation **Time order of growth**



Searching an Ordered List

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

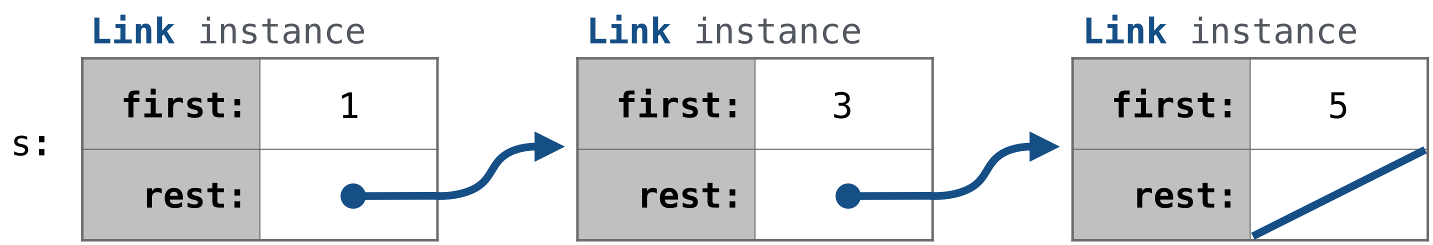
Operation	Time order of growth
contains	



Searching an Ordered List

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

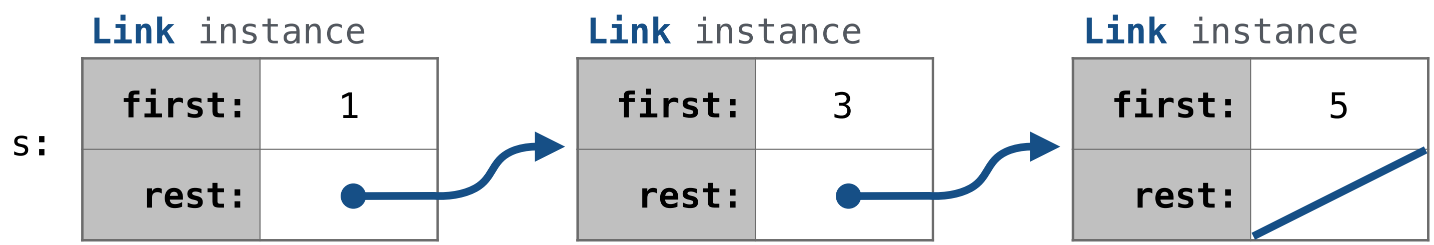
Operation	Time order of growth
contains	



Searching an Ordered List

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

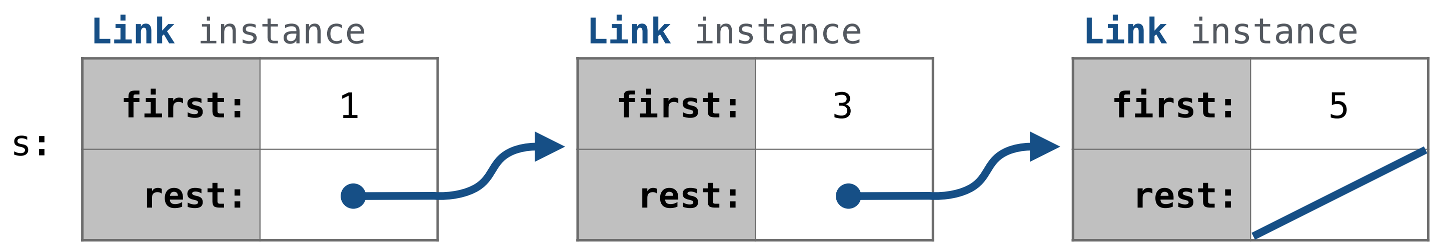
Operation	Time order of growth
contains	



Searching an Ordered List

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

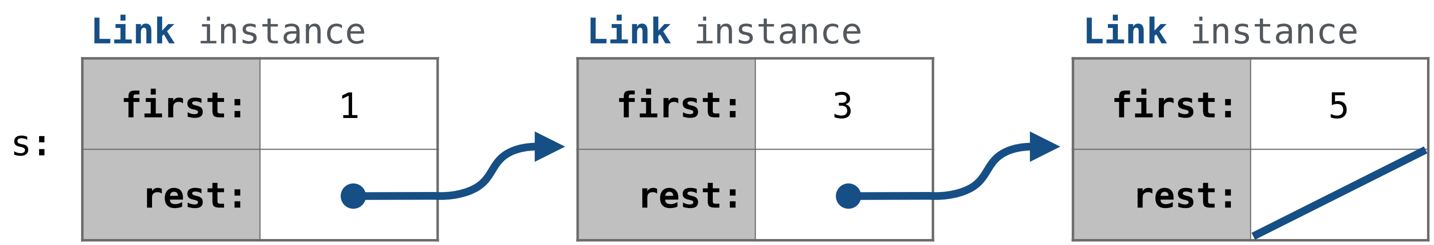
Operation	Time order of growth
contains	



Searching an Ordered List

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

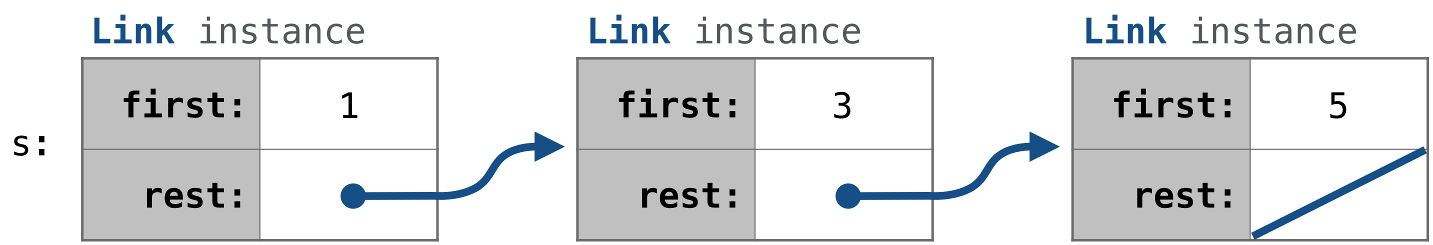
Operation	**Time order of growth**
contains	



Searching an Ordered List

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

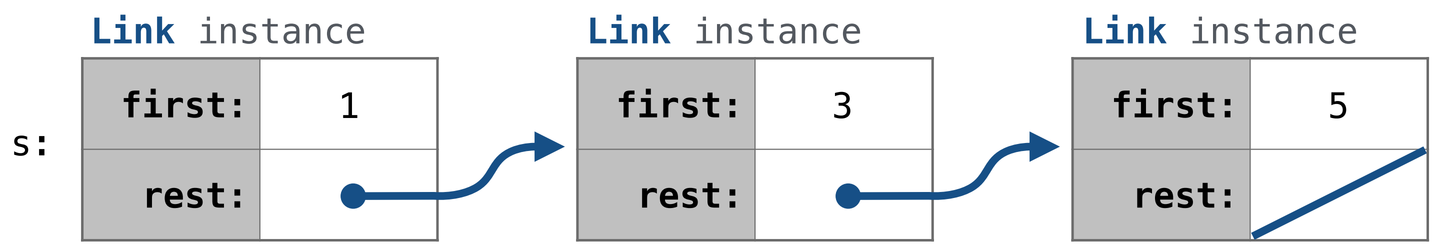
Operation	Time order of growth
contains	$\Theta(n)$



Searching an Ordered List

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

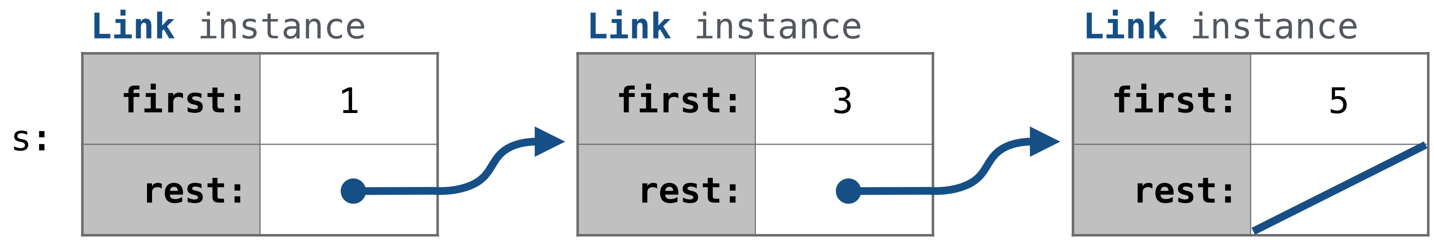
Operation	**Time order of growth**
contains	$\Theta(n)$
adjoin	



Searching an Ordered List

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

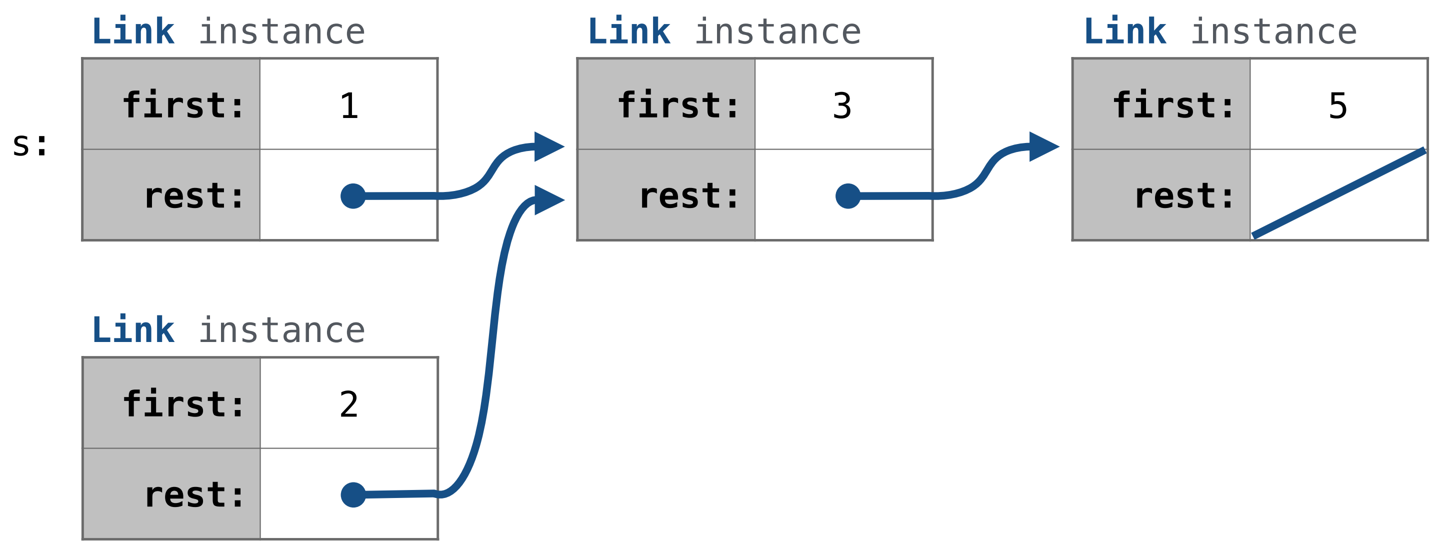
Operation	**Time order of growth**
contains	$\Theta(n)$
adjoin	



Searching an Ordered List

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

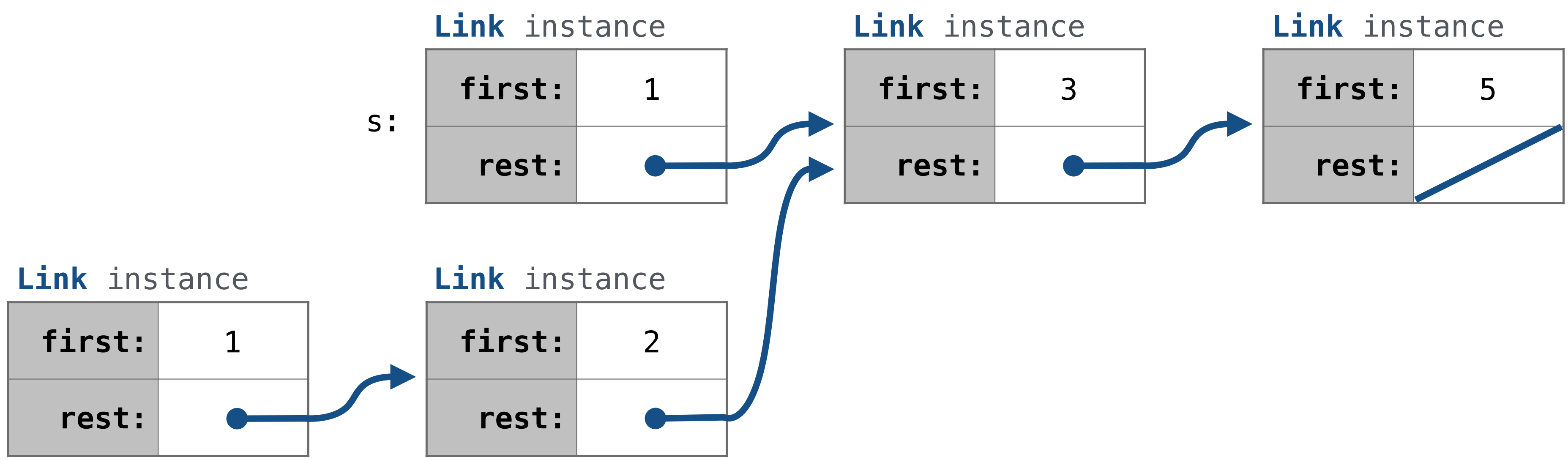
Operation	Time order of growth
contains	$\Theta(n)$
adjoin	



Searching an Ordered List

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

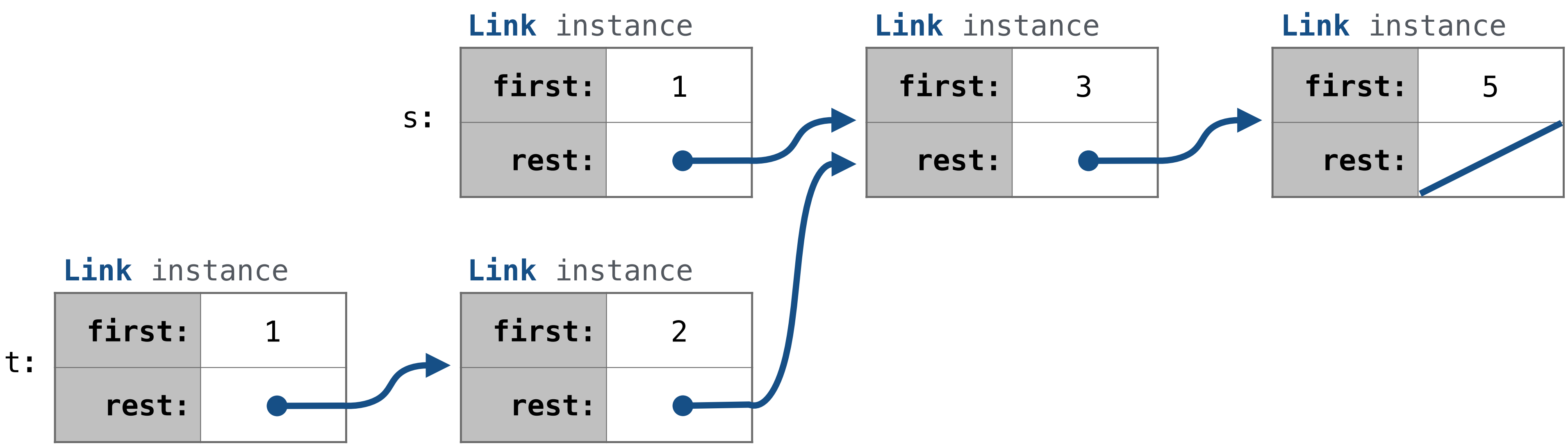
Operation	**Time order of growth**
contains	$\Theta(n)$
adjoin	



Searching an Ordered List

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

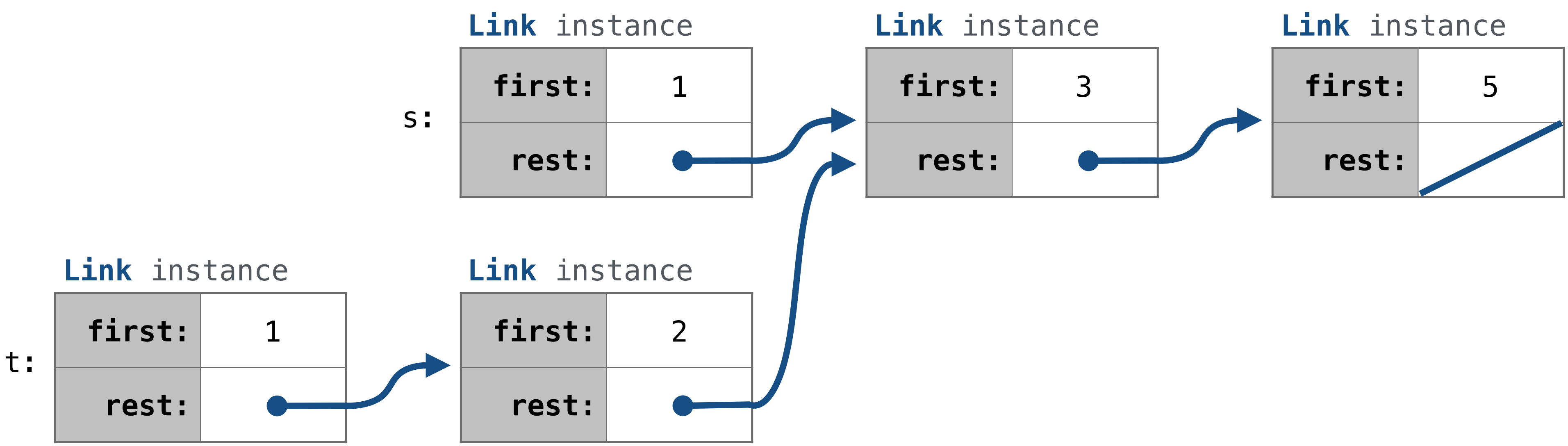
Operation	Time order of growth
contains	$\Theta(n)$
adjoin	



Searching an Ordered List

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

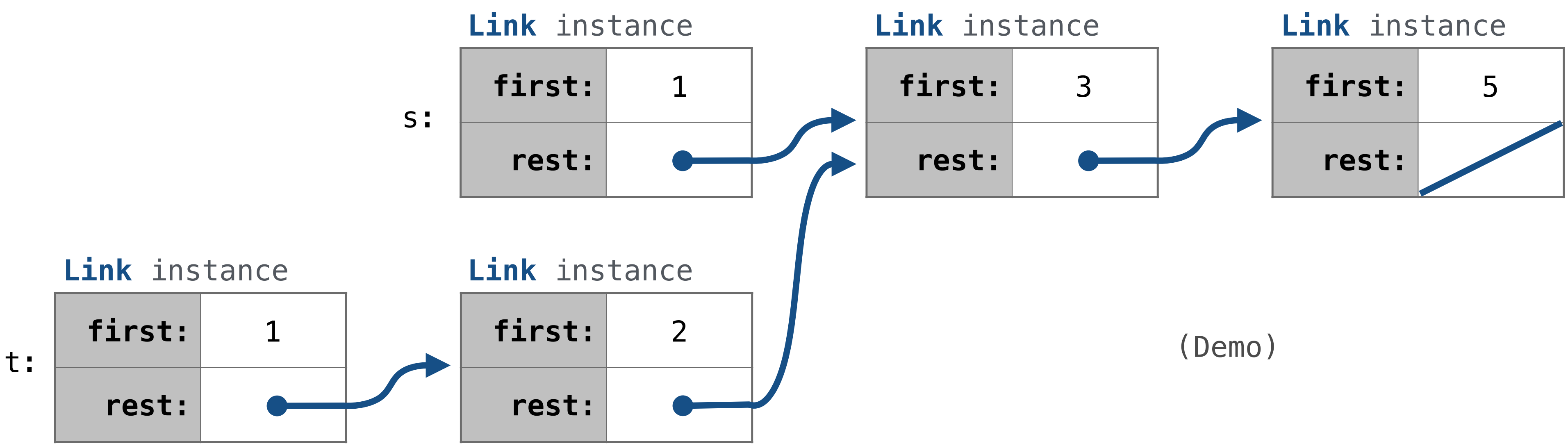
Operation	**Time order of growth**
contains	$\Theta(n)$
adjoin	$\Theta(n)$



Searching an Ordered List

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

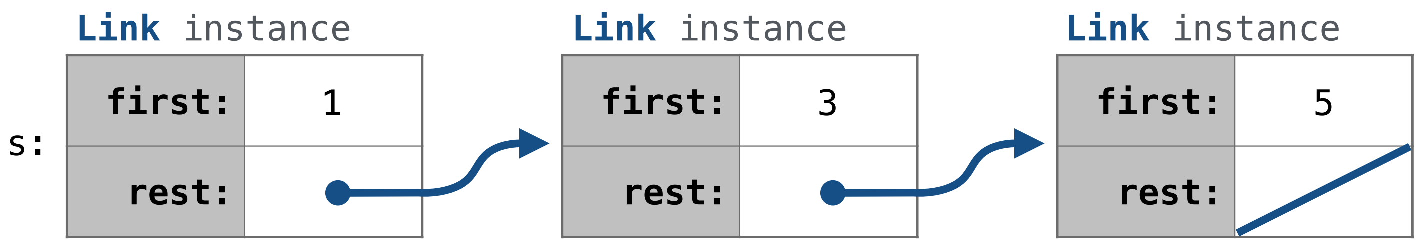
Operation	**Time order of growth**
contains	$\Theta(n)$
adjoin	$\Theta(n)$



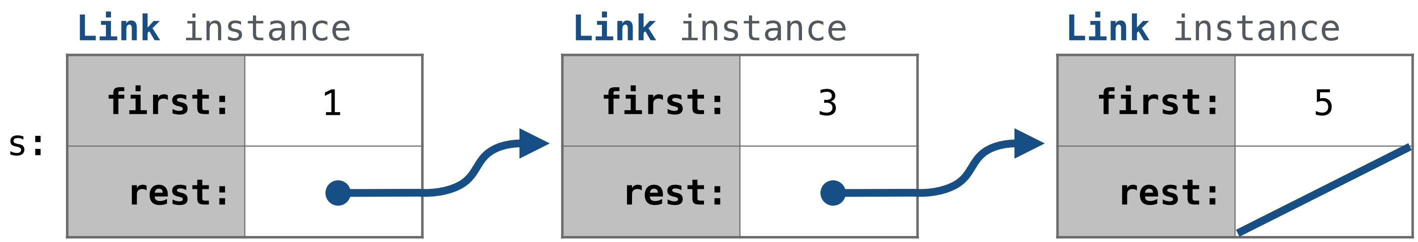
(Demo)

Set Mutation

Adding to an Ordered List

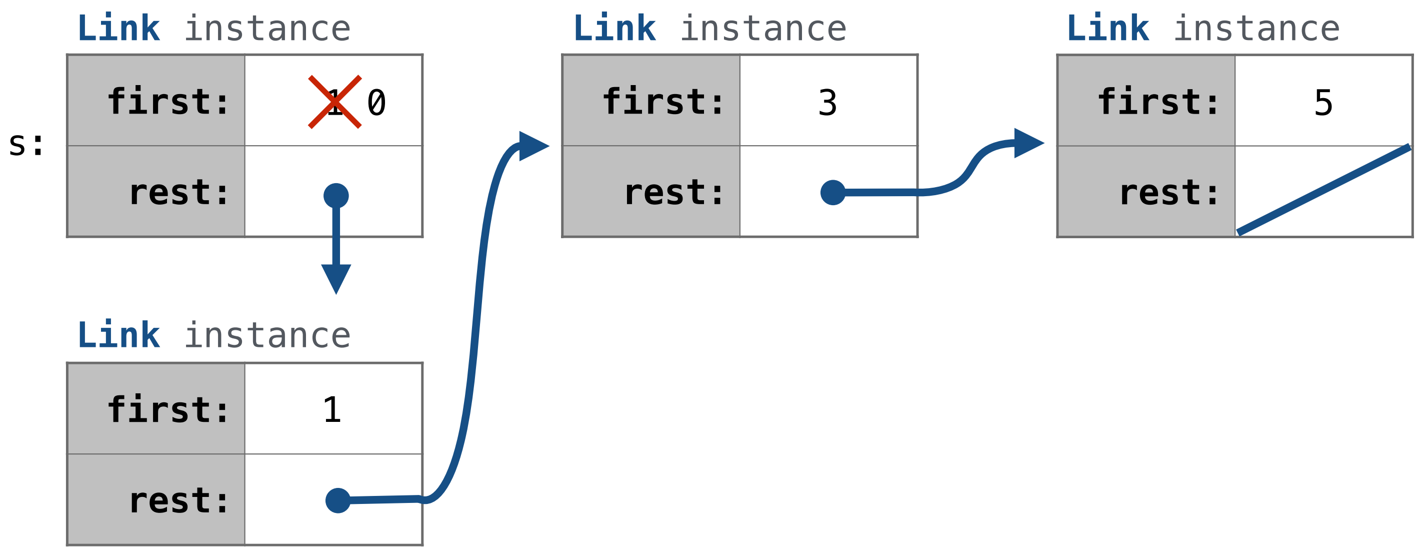


Adding to an Ordered List

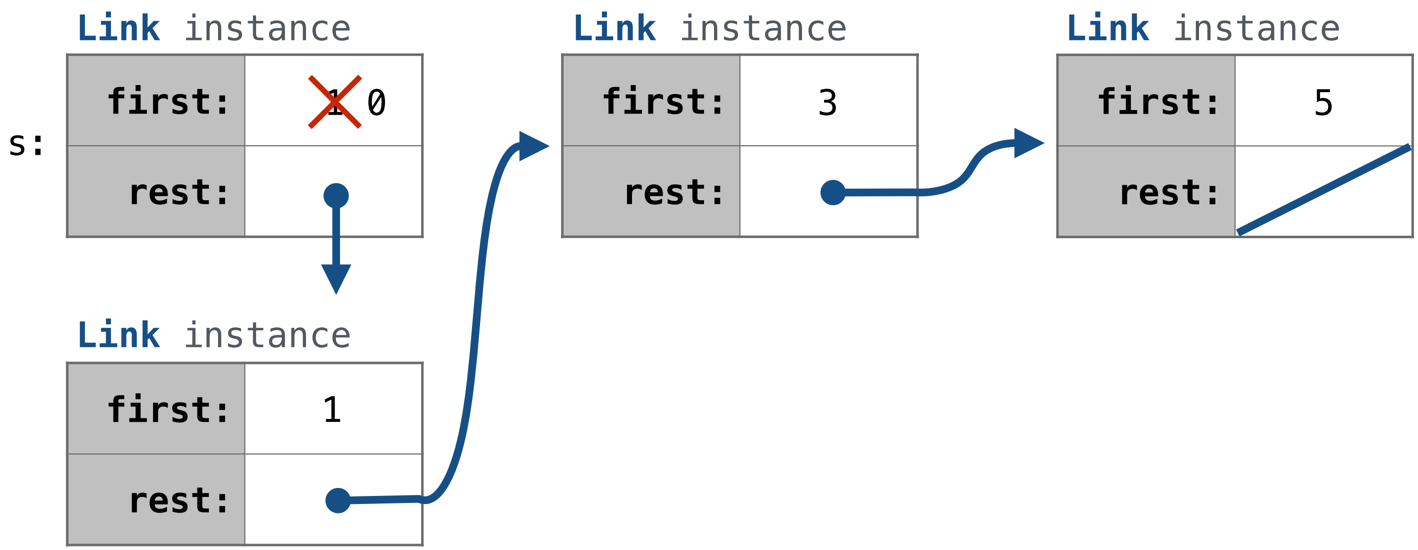



```
add(s, 0)
```

Adding to an Ordered List

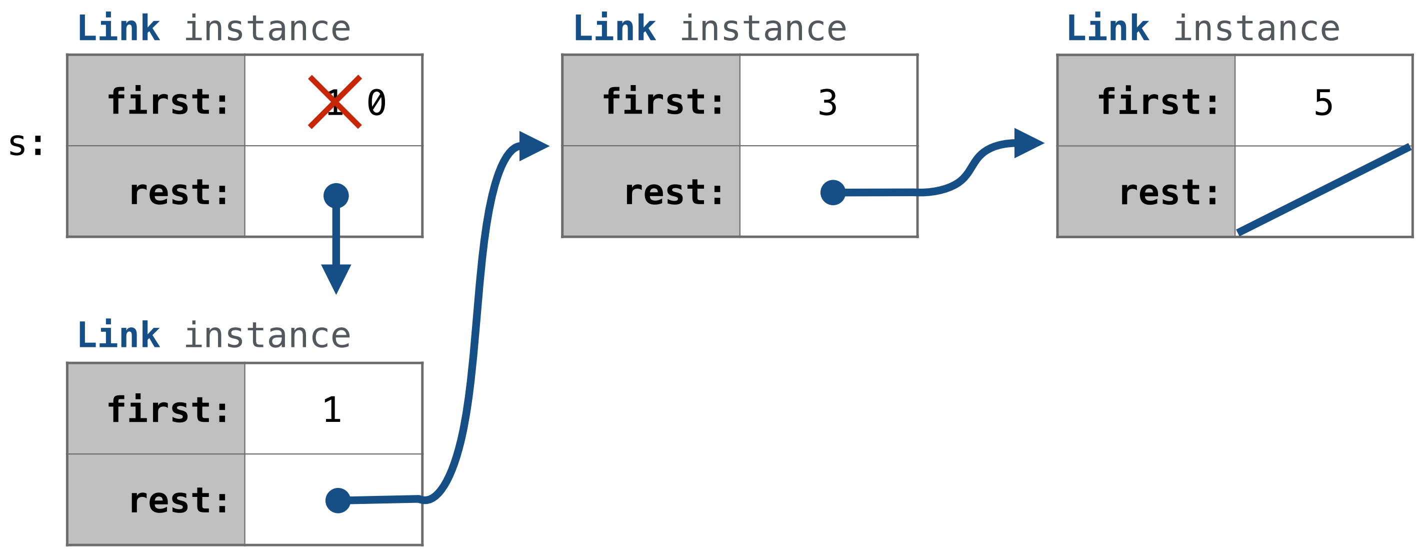


Adding to an Ordered List



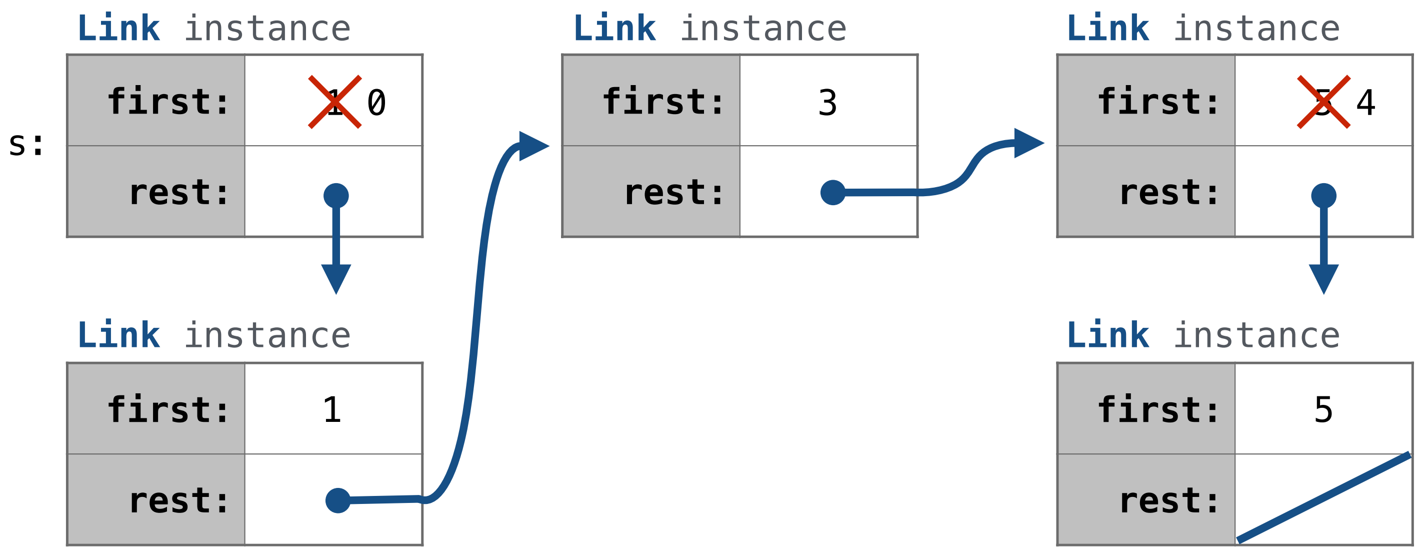
add(s, 3)

Adding to an Ordered List

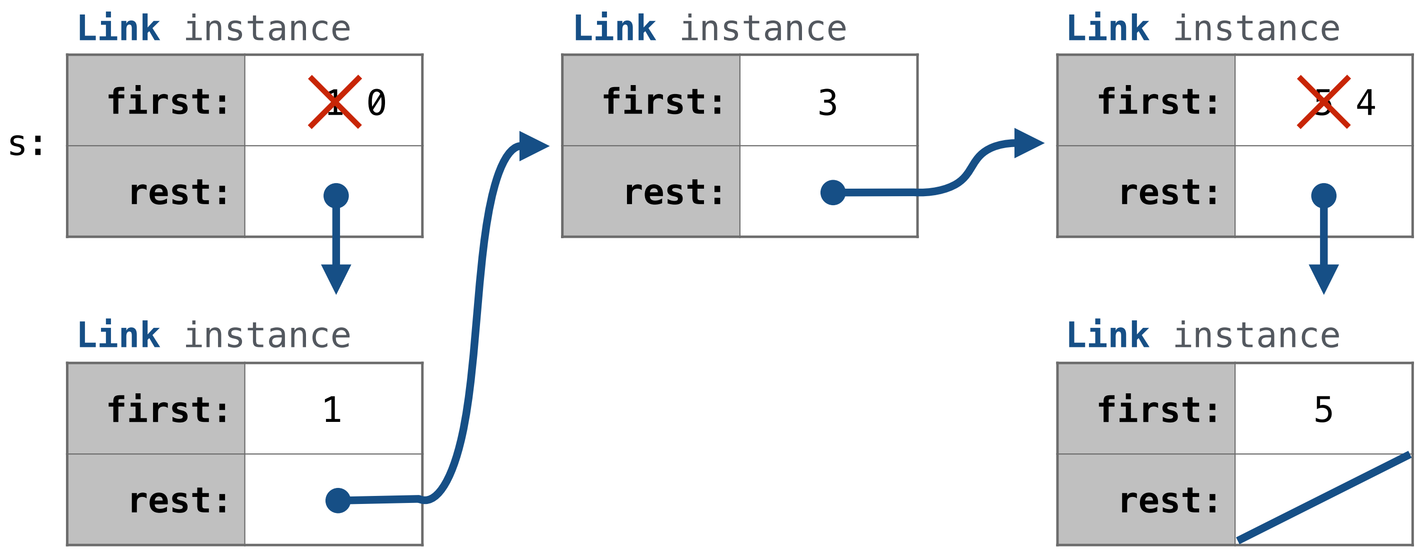



```
add(s, 3)

add(s, 4)
```

Adding to an Ordered List

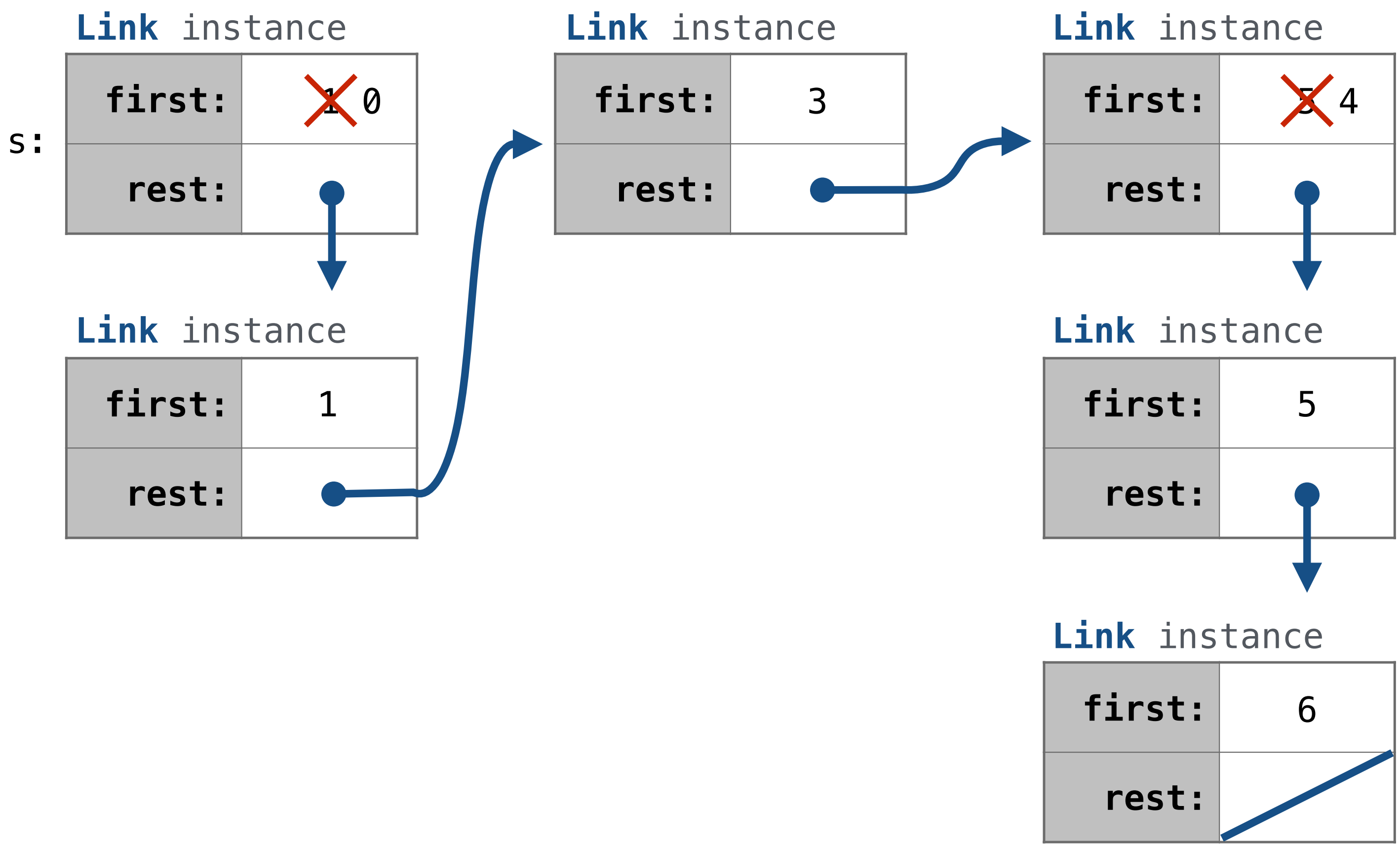


Adding to an Ordered List

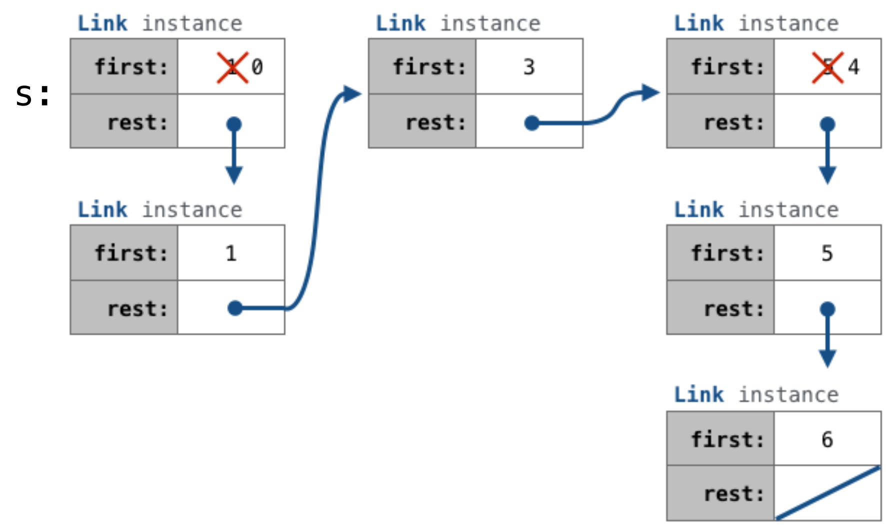


add(s, 6)

Adding to an Ordered List

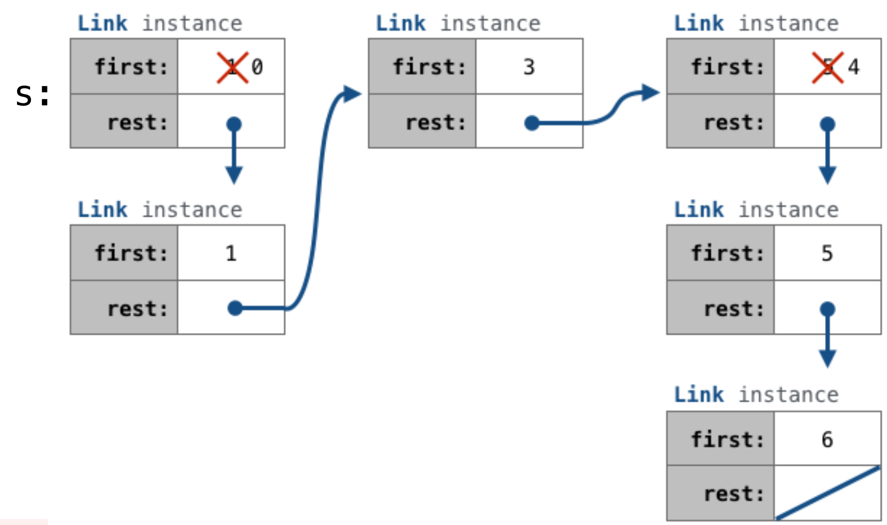


Adding to an Ordered List



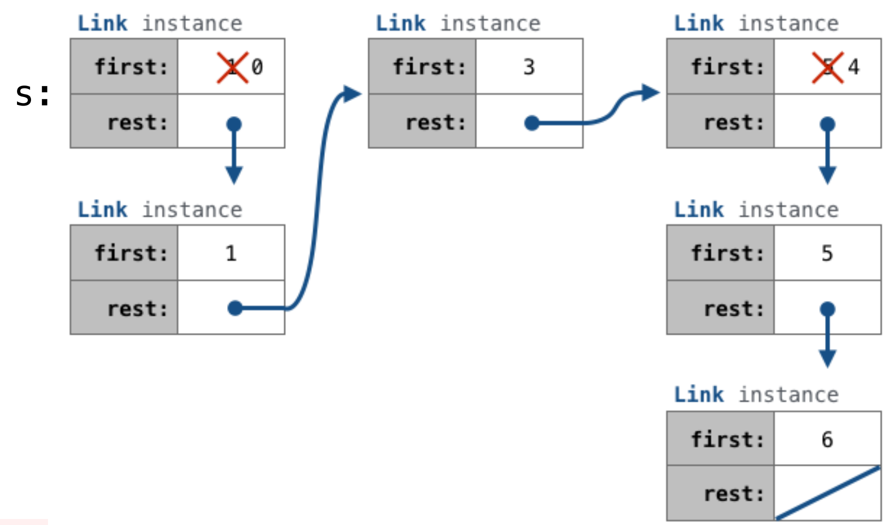
Adding to an Ordered List

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



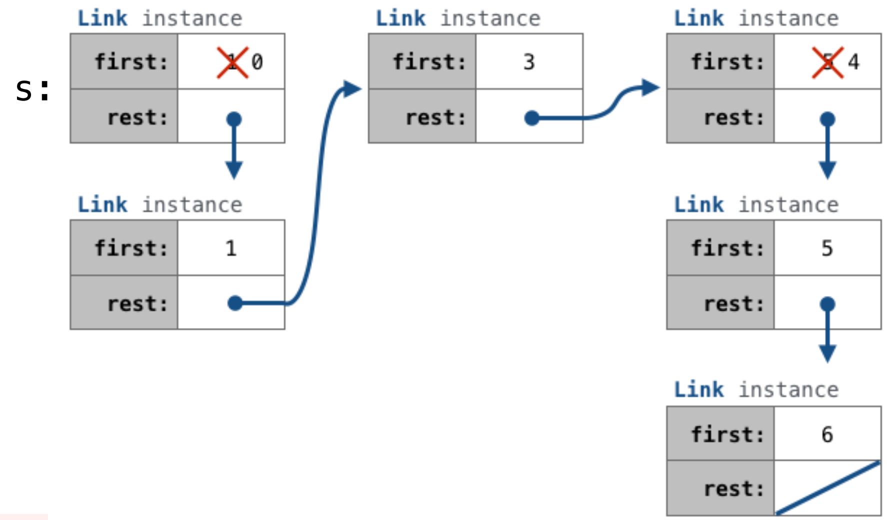
Adding to an Ordered List

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



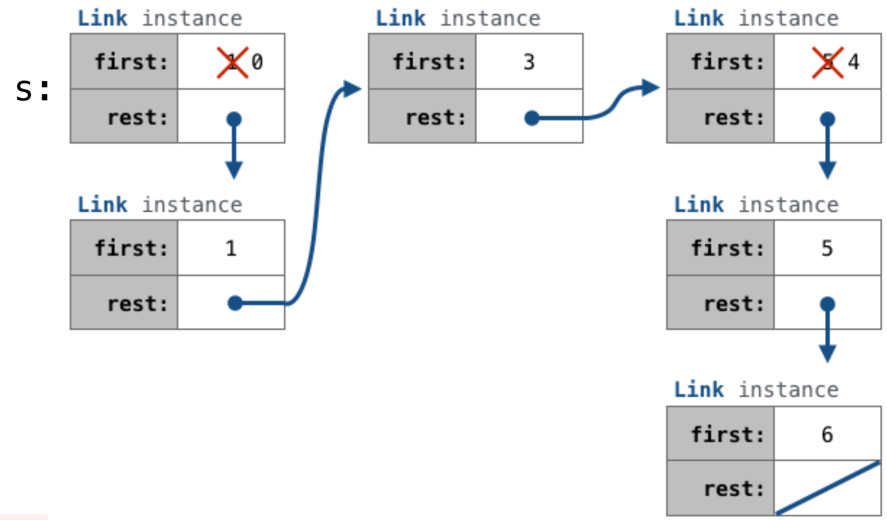
Adding to an Ordered List

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

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



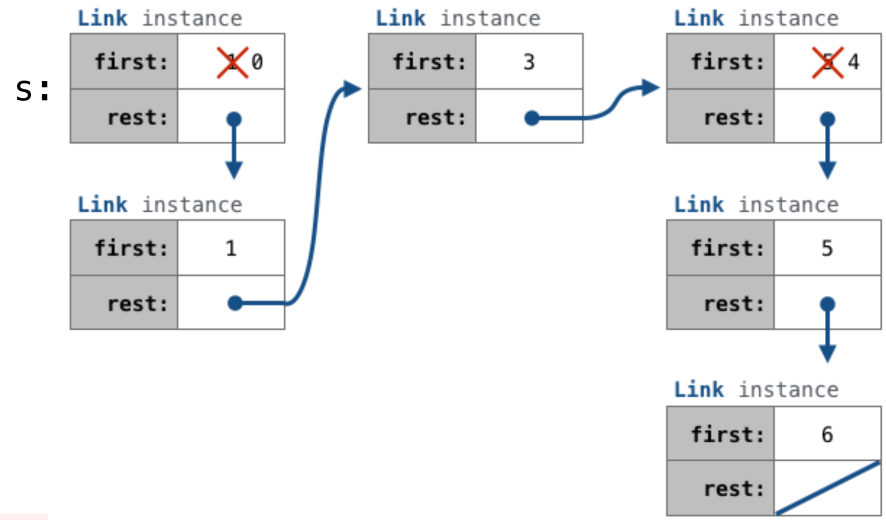
Adding to an Ordered List

```
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))))
```



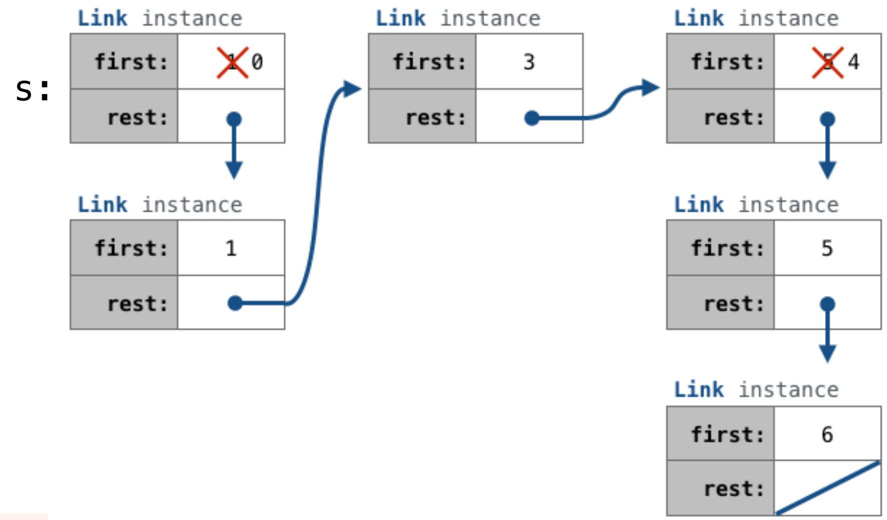
Adding to an Ordered List

```
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))))
```



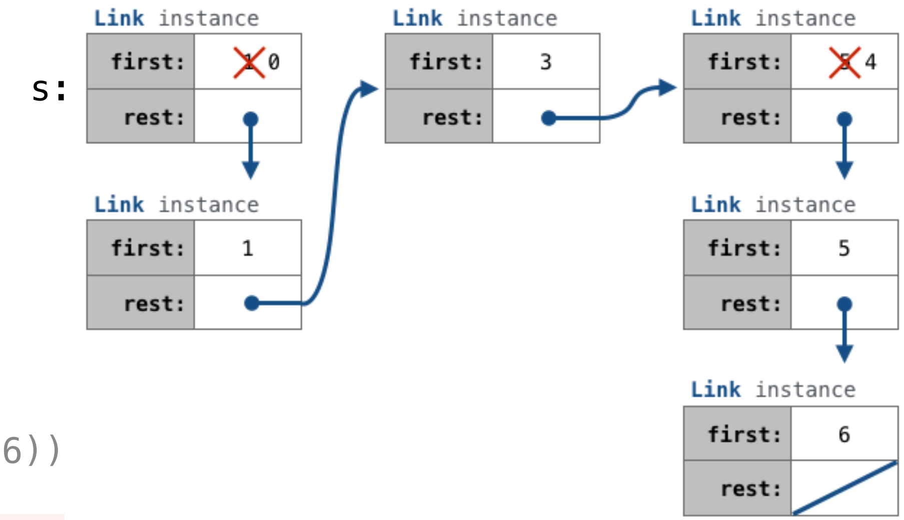
Adding to an Ordered List

```
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)))))
```



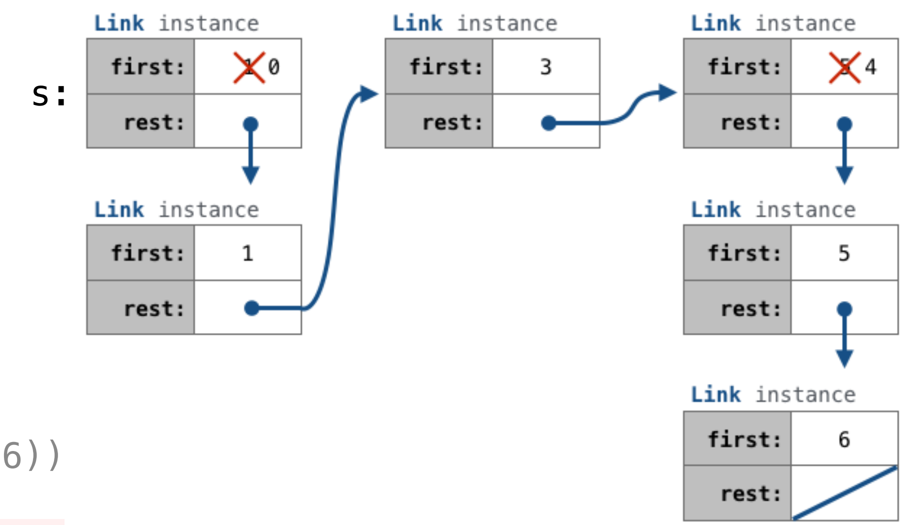
Adding to an Ordered List

```
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))
    """
```



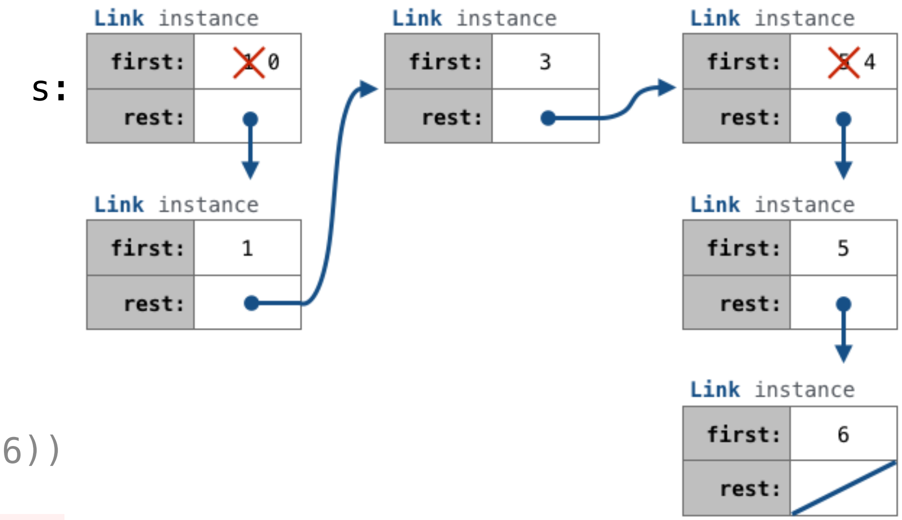
Adding to an Ordered List

```
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."
```



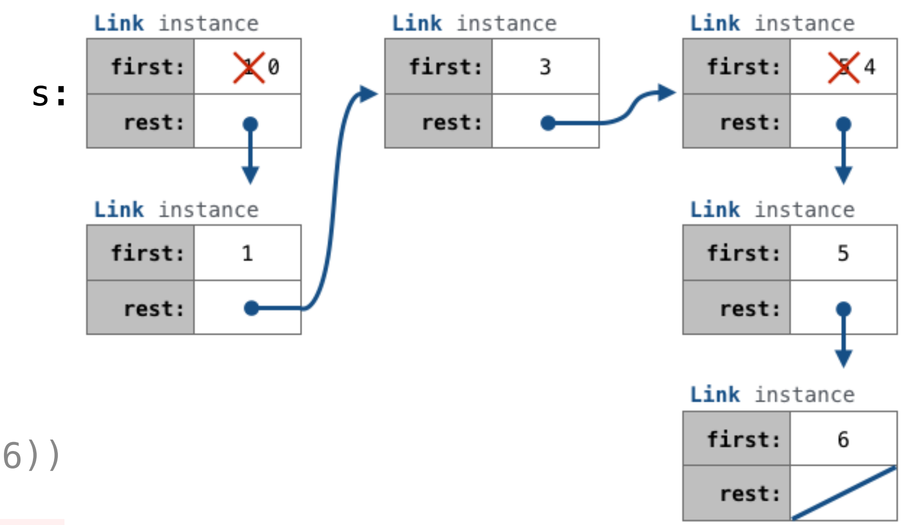
Adding to an Ordered List

```
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 = __________________________ , ______________________________
```



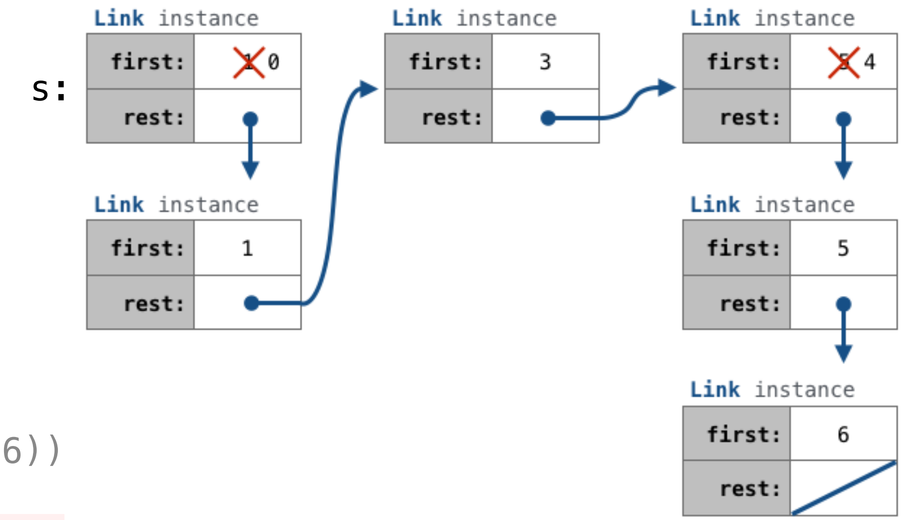
Adding to an Ordered List

```
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 = ___________________________________________________________________
```



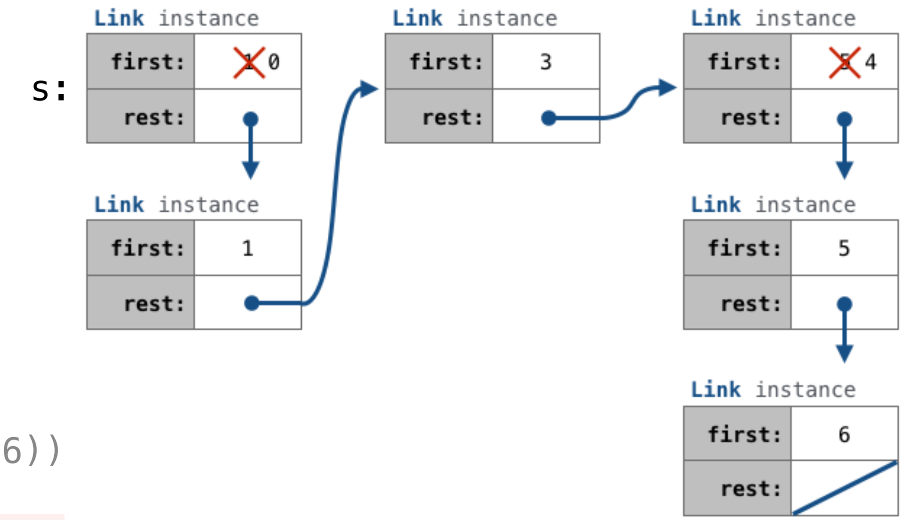
Adding to an Ordered List

```
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:
        __________________________________________________________
```



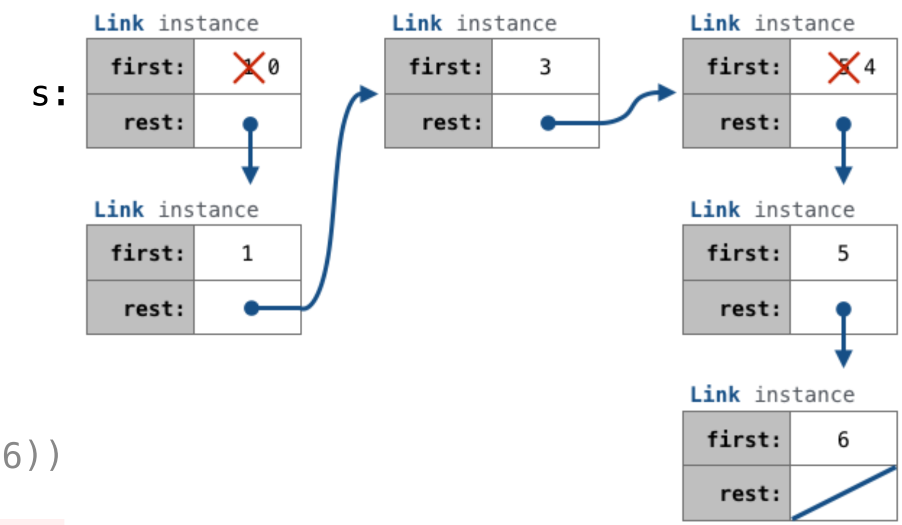
Adding to an Ordered List

```
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
```



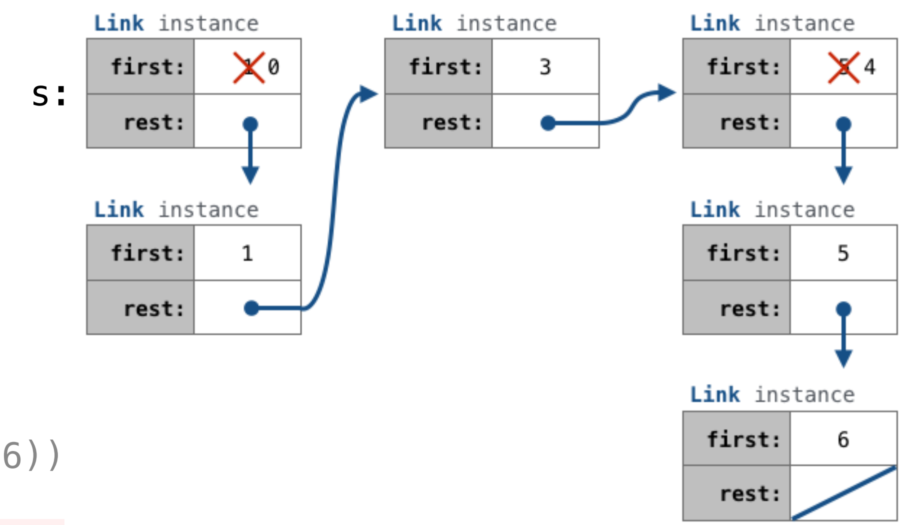
Adding to an Ordered List

```
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 = ______v______ , ____________
    elif s.first < v and empty(s.rest):
        s.rest = ____________________________
    elif s.first < v:
        ____________________________________
    return s
```



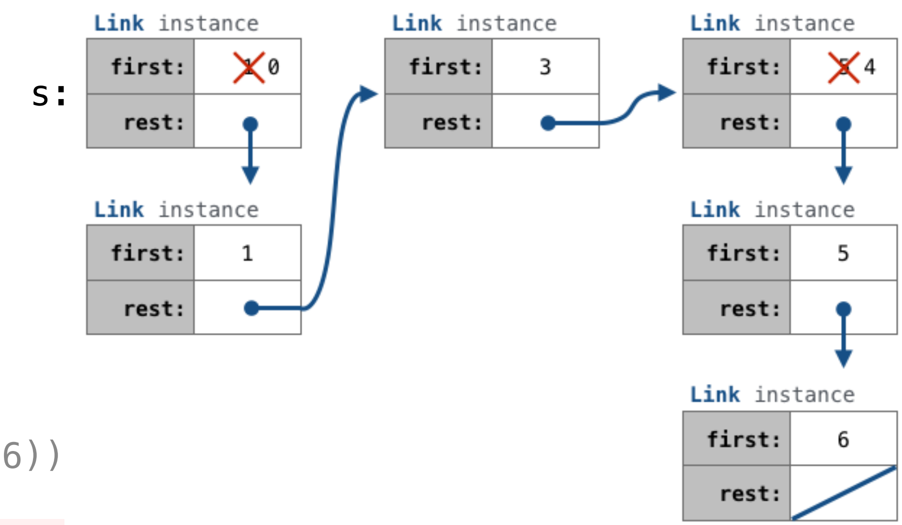
Adding to an Ordered List

```
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 = ___________v___________ , ______Link(s.first, s.rest)______
    elif s.first < v and empty(s.rest):
        s.rest = ____________________________________________
    elif s.first < v:
        ____________________________________________________
    return s
```



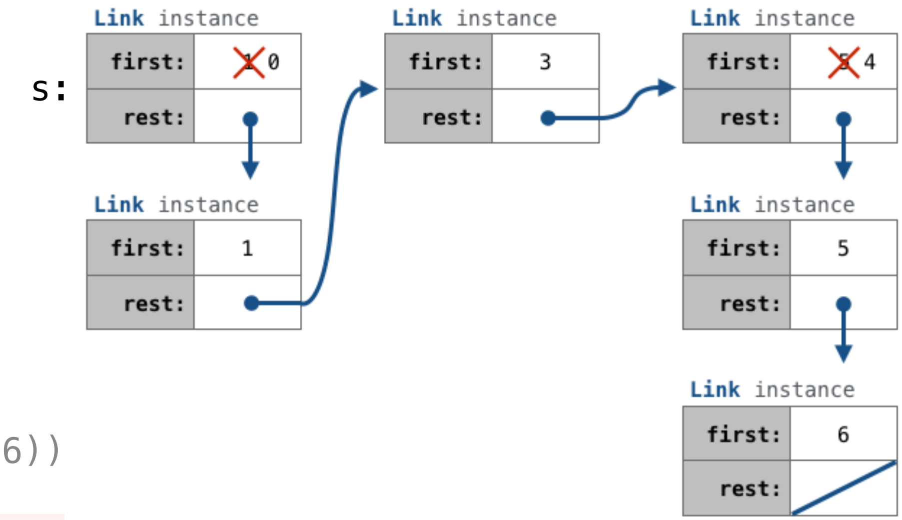
Adding to an Ordered List

```
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 = _____________v_____________ , _______Link(s.first, s.rest)_______
    elif s.first < v and empty(s.rest):
        s.rest = _______________________Link(v, 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 = ______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
```



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***

```
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***

```
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***

```
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***

```
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***

```
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***

```
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)
```

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:
        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***

```
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? $\Theta(n)$

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:
        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)