Decomposition (Order of Growth & Linked List Practice)

Announcements

Order of Growth Practice

Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time

```python
def search_sorted(s, v):
    """Return whether v is in the sorted list s.

    >>> evens = [2*x for x in range(50)]
    >>> search_sorted(evens, 22)
    True
    >>> search_sorted(evens, 23)
    False
    """
    if len(s) == 0:
        return False
    center = len(s) // 2
    if s[center] == v:
        return True
    if s[center] > v:
        rest = s[:center]
    else:
        rest = s[center + 1:]
    return search_sorted(rest, v)
```

Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

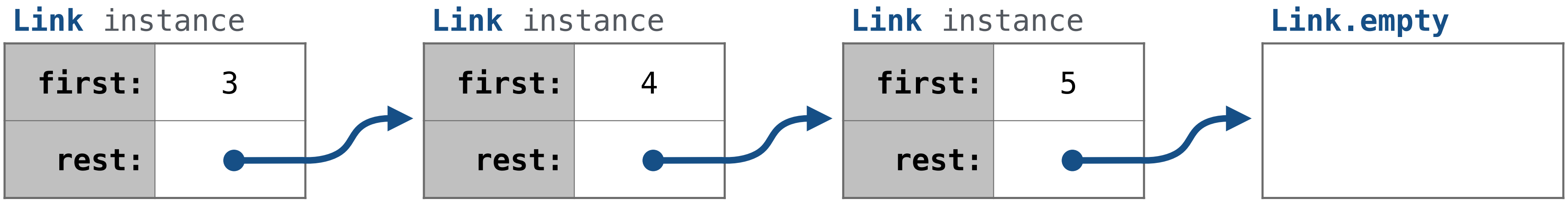
Constant growth. Increasing *n* doesn't affect time

```
def near_pairs(s):
    """Return the length of the longest contiguous
    sequence of repeated elements in s.

    >>> near_pairs([3, 5, 2, 2, 4, 4, 4, 2, 2])
    3
    """
    count, max_count, last = 0, 0, None
    for i in range(len(s)):
        if count == 0 or s[i] == last:
            count += 1
            max_count = max(count, max_count)
        else:
            count = 1
        last = s[i]
    return max_count

def max_sum(s):
    """Return the largest sum of a contiguous
    subsequence of s.

    >>> max_sum([3, 5, -12, 2, -4, 4, -1, 4, 2, 2])
    11
    """
    largest = 0
    for i in range(len(s)):
        total = 0
        for j in range(i, len(s)):
            total += s[j]
            largest = max(largest, total)
    return largest
```

Linked Lists Practice

Linked List Notation

```
s = Link(3, Link(4, Link(5)))
```



Link instance

first: 3

rest:

Link instance

first: 4

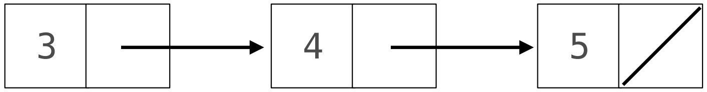
rest:

Link instance

first: 5

rest:

Link.empty



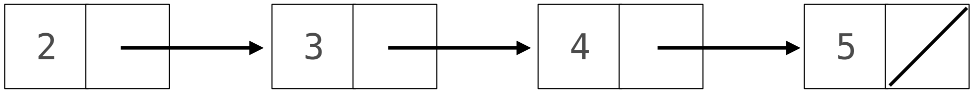
3

4

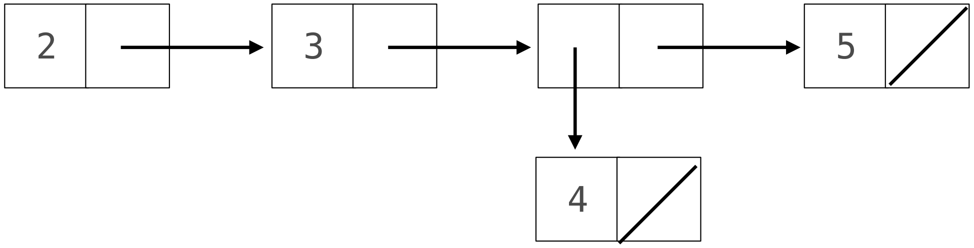
5

Nested Linked Lists

```
>>> s = Link(2, Link(3, Link(    4     , Link(5))))

>>> t = Link(2, Link(3, Link(  Link(4) , Link(5))))

>>> print(s)

<2 3 4 5>
```

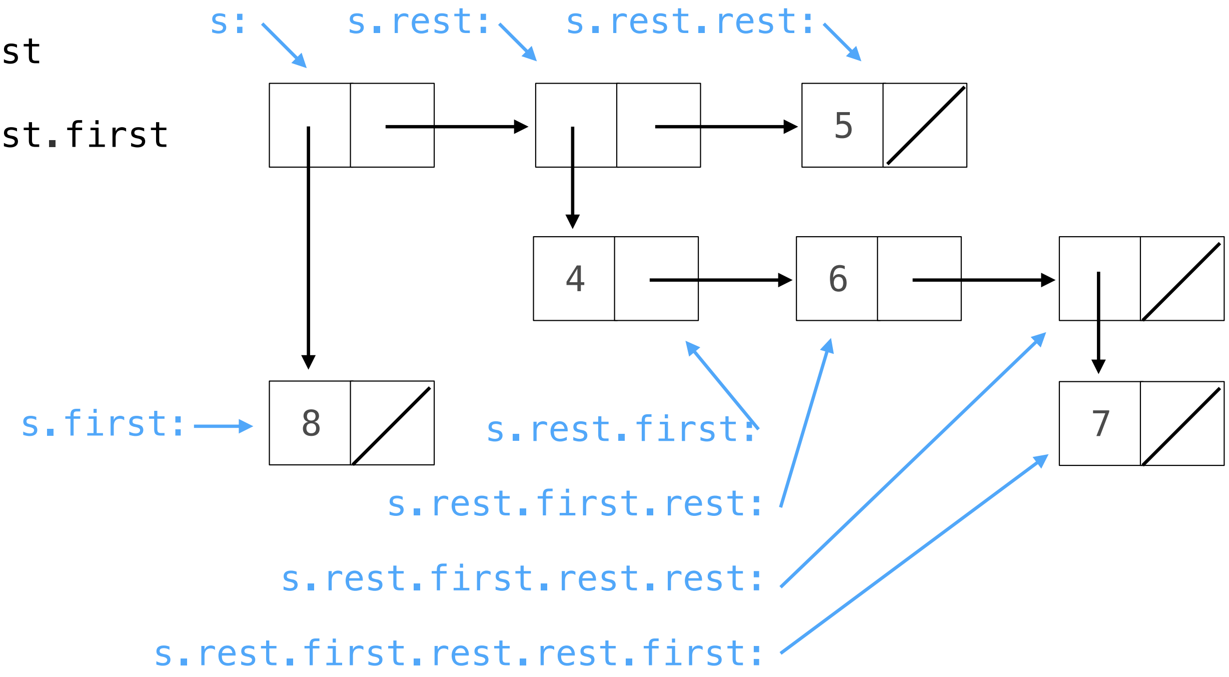


```
>>> print(t)

<2 3 <4> 5>
```



Nested Linked Lists

```
>>> s = Link(Link(8), Link(Link(4, Link(6, Link(Link(7)))), Link(5)))
>>> print(s)
<<8> <4 6 <7>> 5>
>>> s.first.first
8
>>> s.rest.first.rest.rest.first
Link(7)
>>> s.rest.first.rest.rest.first.first
7
```



Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    if s is Link.empty:
        return 0
    else:
        return 1 + length(s.rest)
```

Iterative approach:

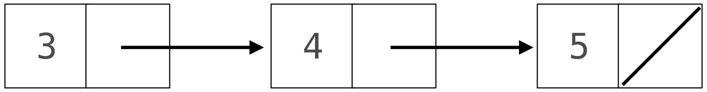
- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
- Implement the process using those names.

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    k = 0
    while s is not Link.empty:
        s, k = s.rest, k + 1
    return k
```

Constructing a Linked List

Build the rest of the linked list, then combine it with the first element.

```
s = Link.empty
s = Link(5, s)
s = Link(4, s)
s = Link(3, s)
```



```
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start up to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """

    if start >= end:

        return Link.empty

    else:

        return  Link(start, range_link(start + 1, end))
```

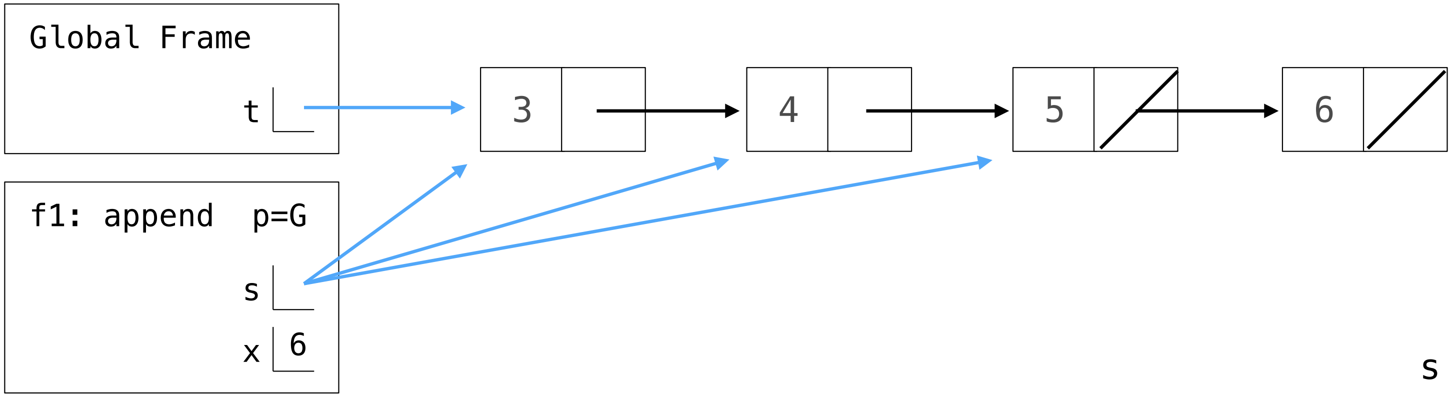
```
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """

    s = Link.empty

    k =  end - 1

    while  k >= start :

        s = Link(k, s)
         k -= 1

    return s
```

Linked List Mutation

To change the contents of a linked list, assign to first and rest attributes

Example: Append x to the end of non-empty s

```
>>> t = Link(3, Link(4, Link(5)))
>>> append(t, 6)
>>> t
Link(3, Link(4, Link(5, Link(6))))
```



s = s.rest

s.rest = Link(x)

Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    <3 4 5 6>
    """
    if  s.rest is not Link.empty :
        append(s.rest, x )
    else:
        s.rest = Link(x)
```

Iterative approach:

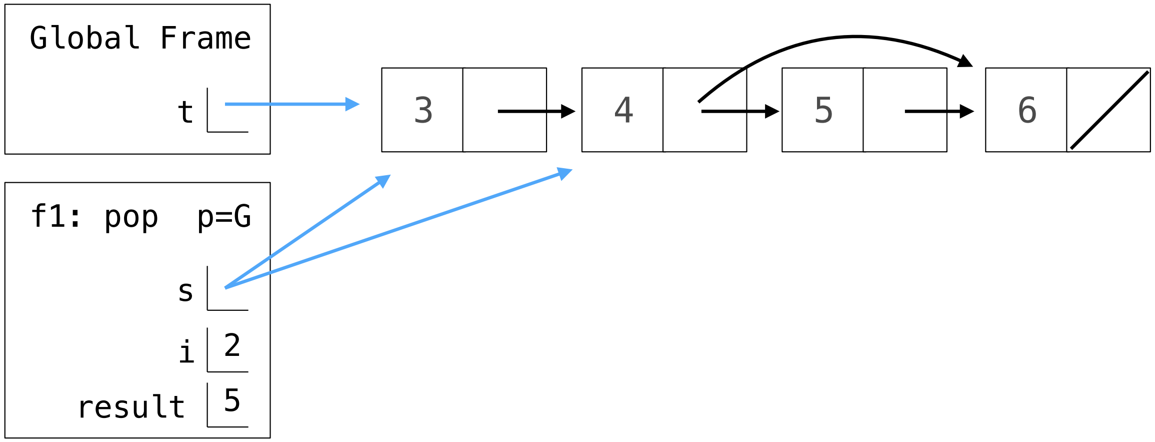
- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
- Implement the process using those names.

```
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    <3 4 5 6>
    """
    while  s.rest is not Link.empty :
        s = s.rest
    s.rest = Link(x)
```

Example: Pop

Implement pop, which takes a linked list s and positive integer i. It removes and returns the element at index i of s (assuming s.first has index 0).

```
def pop(s, i):
    """Remove and return element i from linked list s for positive i.
    >>> t = Link(3, Link(4, Link(5, Link(6))))
    >>> pop(t, 2)
    5
    >>> pop(t, 2)
    6
    >>> pop(t, 1)
    4
    >>> t
    Link(3)
    """
    assert i > 0 and i < length(s)

    for x in range( i - 1 ):

        s = s.rest
    result = s.rest.first

    s.rest = s.rest.rest

    return result
```


Global Frame
t
3
4
5
6
f1: pop p=G
s
i 2
result 5