The Art of Data Structures

Lists

Alan Beadle

CSC 162: The Art of Data Structures

UNIVERSITY of ROCHESTER

Agenda

- What is a List?
- Singly linked list DS
- The unordered list ADT
- The ordered list ADT
- Doubly linked list DS

Lists

Lists

Unordered Lists

- The list is a powerful, yet simple, collection mechanism
- Not all programming languages include a list collection
- A list is a collection of items where each item holds a relative position with respect to the others
- We will refer to this type of list as an unordered list
- A linked list will be the basis of this ADT

Lists

Ordered Lists

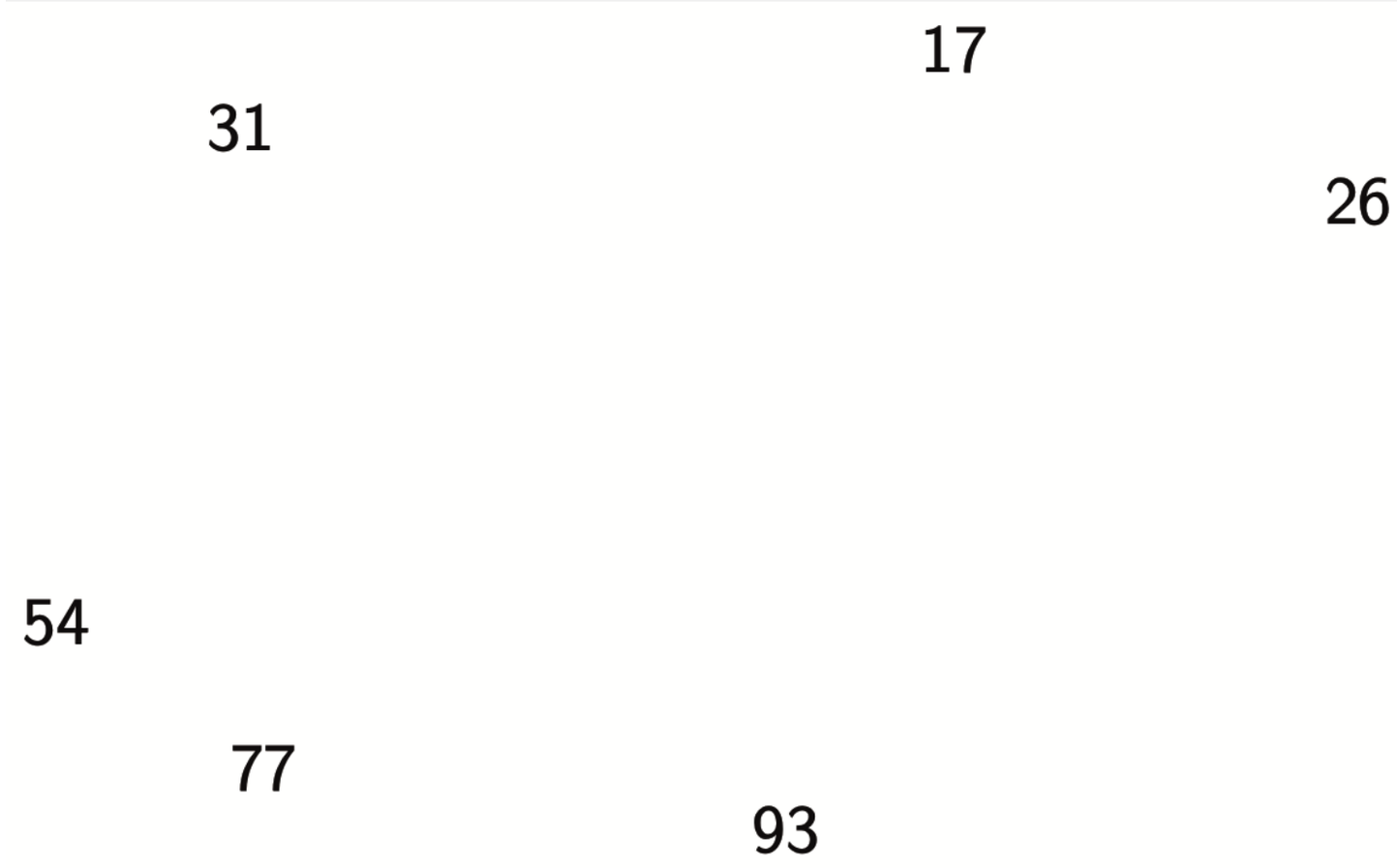
- An ordered list is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item
  - Ascending or Descending, typically
  - List items should have meaningful comparison operators should be defined
- A linked list is also the basis of this ADT

Implementation

Singly Linked List: Node

Implementation

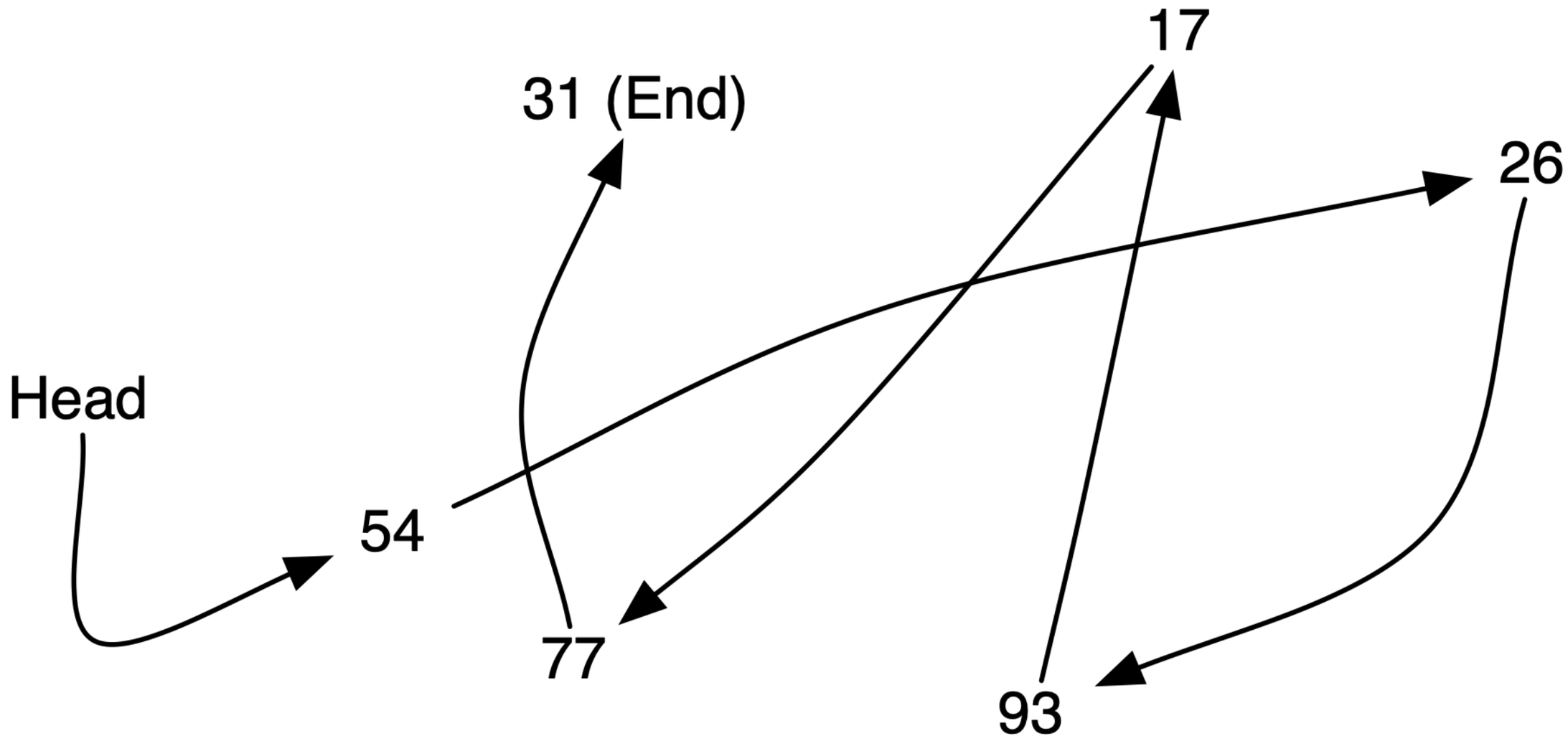
Visualization



Items Not Constrained in Their Physical Placement in Memory

Implementation

Visualization



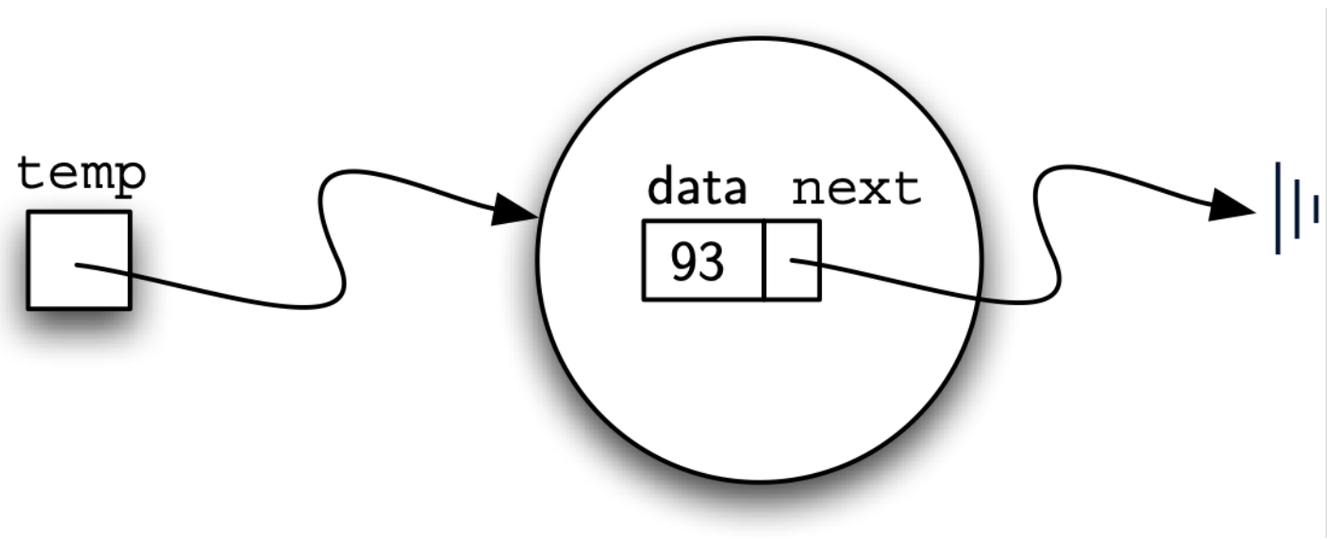
Relative Position of Items Maintained by Explicit Links

Implementation
Linked-List: Node

- A node is the basic building block for the linked list implementation
- Each node object must hold at least two pieces of information:
  - The list item (value, data field)
  - Reference to next node
- `None` is a useful object here

Implementation

Empty Linked-List



A Node Object Contains the Item and a Reference to the Next Node

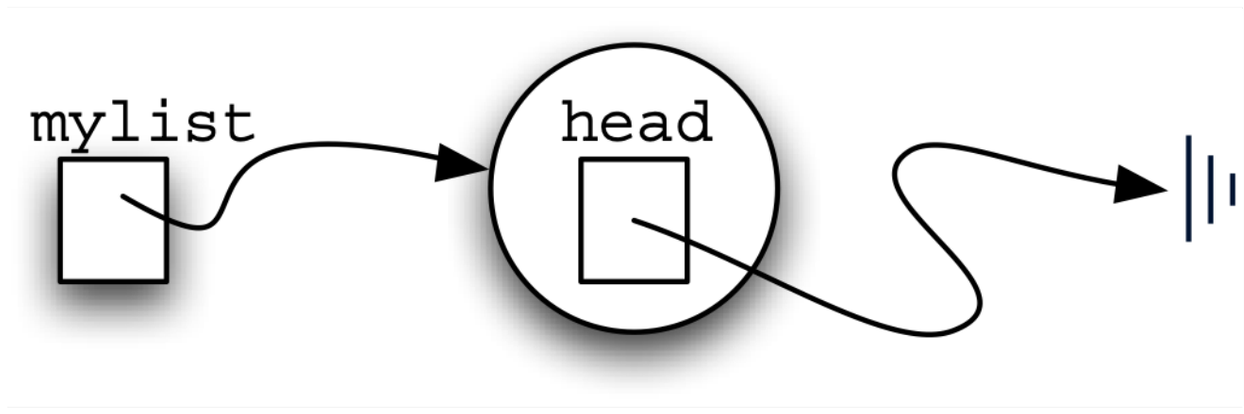
Implementation

Empty Linked-List

A Typical Representation for a Node

Implementation

Empty Linked-List



An Empty List

Implementation

Node Implementation in Python

```
>>> temp = Node(93)
>>> temp.get_data()
93
```

Implementation

Node Implementation in Python

```
class Node:
  def __init__(self, init_data):
    self.data = init_data
    self.next = None

  def get_data(self):
    return self.data

  def get_next(self):
    return self.next

  def set_data(self, new_data):
    self.data = new_data

  def set_next(self, new_next):
    self.next = new_next
```

Implementation

Unordered List

Implementation

Unordered List

- `UnorderedList()` creates a new list that is empty. It needs no parameters and returns an empty list

- `add(item)` adds a new item to the list. It needs the item and returns nothing; assume the item is not already in the list

- `remove(item)` removes the item from the list; It needs the item and modifies the list; assume the item is present in the list

Implementation
Unordered List

- `search(item)` searches for the item in the list; it needs the item and returns a boolean value

- `is_empty()` tests to see whether the list is empty; it needs no parameters and returns a boolean value

- `length()` returns the number of items in the list; it needs no parameters and returns an integer

Implementation
Unordered List

- `append(item)` adds a new item to the end of the list making it the last item in the collection; it needs the item and returns nothing; assume the item is not already in the list
- `index(item)` returns the position of item in the list. It needs the item and returns the index; assume the item is in the list

Implementation
Unordered List

- `insert(pos, item)` adds a new item to the list at position pos; it needs the item and returns nothing; assume the item is not already in the list and there are enough existing items to have position pos
- `pop()` removes and returns the last item in the list.; it needs nothing and returns an item; assume the list has at least one item

Implementation
Unordered List

- `pop(pos)` removes and returns the item at position pos; it needs the position and returns the item; assume the item is in the list

Implementation

Unordered List

Implementation in Python

```
>>> mylist = UnorderedList()
```

Implementation

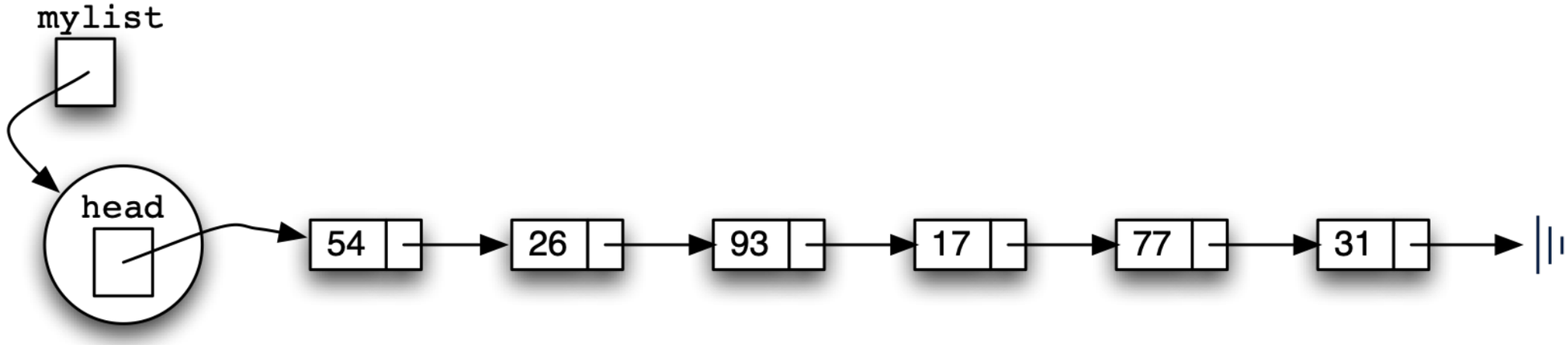
Unordered List

Implementation in Python

```
>>> mylist.add(31)
>>> mylist.add(77)
>>> mylist.add(17)
>>> mylist.add(93)
>>> mylist.add(26)
>>> mylist.add(54)
```

Implementation

Populated Linked-List



A Linked List of Integers

Implementation

Unordered List

Implementation in Python

```
class UnorderedList:
    def __init__(self):
        self.head = None
```

```
def add(self, item):
    temp = Node(item)
    temp.set_next(self.head)
    self.head = temp

def length(self):
    current = self.head
    count = 0
    while current != None:
        count = count + 1
        current = current.get_next()

    return count
```

Implementation

Unordered List

Implementation in Python

```
def search(self, item):
    current = self.head
    found = False
    while current != None and not found:
        if current.get_data() == item:
            found = True
        else:
            current = current.get_next()
    return found
```

Implementation

Unordered List Implementation in Python

```
def remove(self, item):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.get_data() == item:
            found = True
        else:
            previous = current
            current = current.get_next()

    if previous == None:
        self.head = current.get_next()
    else:
        previous.set_next(current.get_next())
```

Implementation

Ordered List

Implementation
Ordered List

- `OrderedList()` creates a new ordered list that is empty; it needs no parameters and returns an empty list
- `add(item)` adds a new item to the list making sure that the order is preserved; it needs the item and returns nothing; assume the item is not already in the list

Implementation
Ordered List

- `remove(item)` removes the item from the list; it needs the item and modifies the list; assume the item is present in the list
- `search(item)` searches for the item in the list; it needs the item and returns a boolean value
- `is_empty()` tests to see whether the list is empty; it needs no parameters and returns a boolean value

Implementation
Ordered List

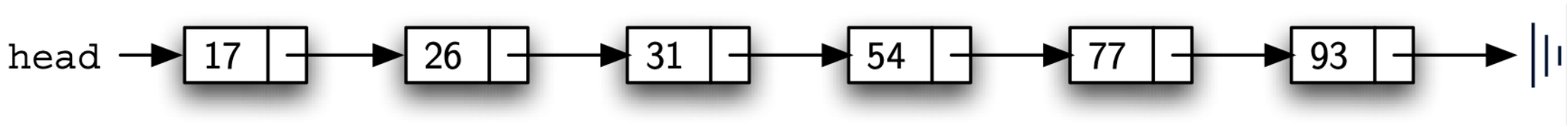
- `length()` returns the number of items in the list; it needs no parameters and returns an integer
- `index(item)` returns the position of item in the list; it needs the item and returns the index; assume the item is in the list
- `pop()` removes and returns the last item in the list; it needs nothing and returns an item; assume the list has at least one item

Implementation
Ordered List

- `pop(pos)` removes and returns the item at position pos; it needs the position and returns the item; assume the item is in the list

Implementation

Ordered List



An Ordered Linked List

Implementation

Ordered List

Implementation in Python

```
class OrderedList:
    def __init__(self):
        self.head = None

    def search(self, item):
        current = self.head
        found = False
        stop = False
        while current != None and not found and not stop:
            if current.get_data() == item:
                found = True
            else:
                if current.get_data() > item:
                    stop = True
                else:
                    current = current.get_next()

        return found
```

Implementation

Ordered List

Implementation in Python

```
def add(self, item):
    current = self.head
    previous = None
    stop = False
    while current != None and not stop:
        if current.get_data() > item:
            stop = True
        else:
            previous = current
            current = current.get_next()

    temp = Node(item)
    if previous == None:
        temp.set_next(self.head)
        self.head = temp
    else:
        temp.set_next(current)
        previous.set_next(temp)
```

Analysis of Singly LL

Analysis of Singly LL
Performance Complexity

- Initialization
  - `UnorderedList()`: $O(1)$
  - `OrderedList()`: $O(1)$
- `add(item)`
  - Unordered: $O(1)$
  - Ordered: $O(n)$

Analysis of Singly LL

Performance Complexity

- `remove(item)`: *O(n)*
- `search(item)`: *O(n)*
- `is_empty()`: *O(1)*
- `length()`: *O(n)*
- `append(item)`
  - Unordered list, only: *O(n)*

Analysis of Singly LL

Performance Complexity

- `index(item)`: O(n)
- `insert(pos, item)`:
  - Unordered list, only: O(n)
- `pop()`:
  - Unordered list: *O(1)*
  - Ordered list: *O(1)*
- `pop(pos)`: *O(n)*

Implementation

Doubly Linked List: `DNode`

Implementation
Doubly Linked List: `DNode`

- We extend the node concept for a doubly linked list implementation
- Now, each node object must hold at least *three* pieces of information:
  - The list item (value, data field)
  - Reference to next node
  - Reference to the previous node

Implementation

Doubly Linked List: `DNode`

- The time complexity is pretty similar to a singly linked list
- Benefits occur if a program holds a reference to any one node in a linked list
  - Given this reference, a node can be removed very quickly, *O(1)*.
  - If only values are known, a *O(n)* search is still required

Implementation
Doubly Linked List: `DNode`

- The code complexity is also less, as now you won't need to track `previous` and `current` as in the `remove` method
- A doubly linked list now has a head and tail, and can be used traverse the list forward or backwards

Implementation

DNode *Implementation in Python*

```
class DNode:
    def __init__(self, init_data):
        self.data = init_data
        self.next = None
        self.prev = None

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next

    def get_prev(self):
        return self.prev

    def set_data(self, new_data):
        self.data = new_data

    def set_next(self, new_next):
        self.next = new_next

    def set_prev(self, new_prev):
        self.prev = new_prev
```

Analysis of Doubly LL

Analysis of Doubly LL

Performance Complexity

- Initialization
  - `UnorderedList()`: $O(1)$
  - `OrderedList()`: $O(1)$
- `add(item)`
  - Unordered: $O(1)$
  - Ordered: $O(n)$

Analysis of Doubly LL

Performance Complexity

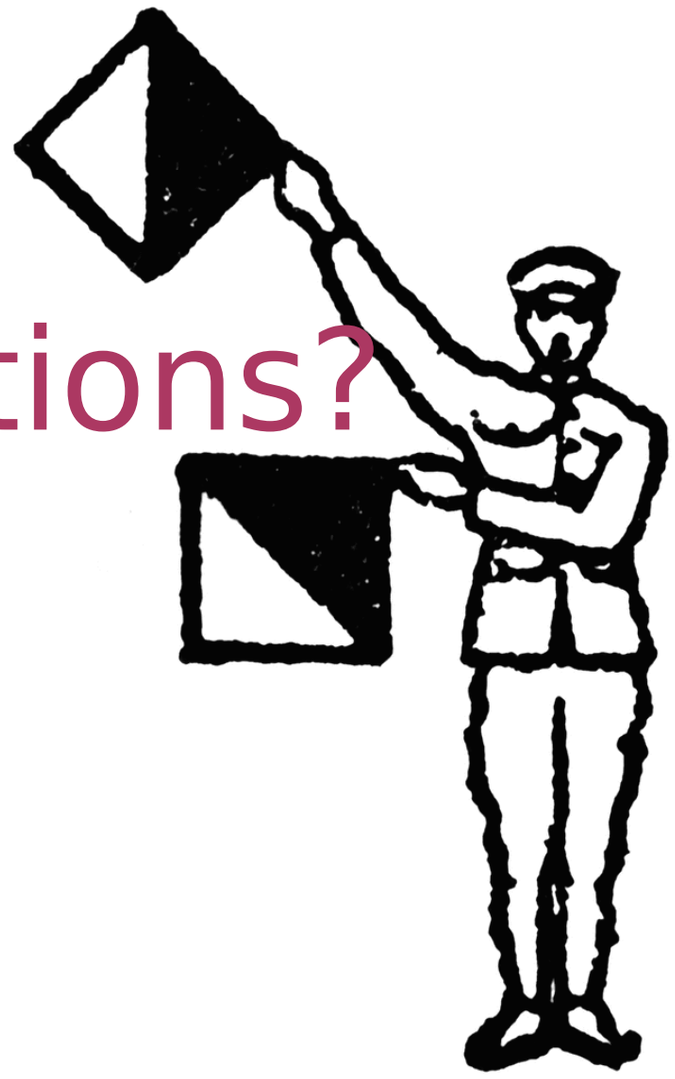
- `remove(item)`: *O(n) or O(1)*
- `search(item)`: *O(n)*
- `is_empty()`: *O(1)*
- `length()`: *O(n)*
- `append(item)`
  - Unordered list, only: *O(1)*

Analysis of Doubly LL

Performance Complexity

- `index(item)`: O(n)
- `insert(pos, item)`:
  - Unordered list, only: O(n)
- `pop()`:
  - Unordered list: *O(1)*
  - Ordered list: *O(1)*
- `pop(pos)`: *O(n)*

Questions?



- Workshop about lists, and two new labs
- Runtime lab (lab 2) due tonight
- Lab 4 is to re-implement a queue using linked nodes
- Lab 6 is to add some methods to the UnorderedList from the textbook
- Lab 6 is probably easier than lab 4 so consider doing 6 first to gain understanding of linked nodes
- There is no lab 5 (deques)
- Labs 4 & 6 are due in 1 week