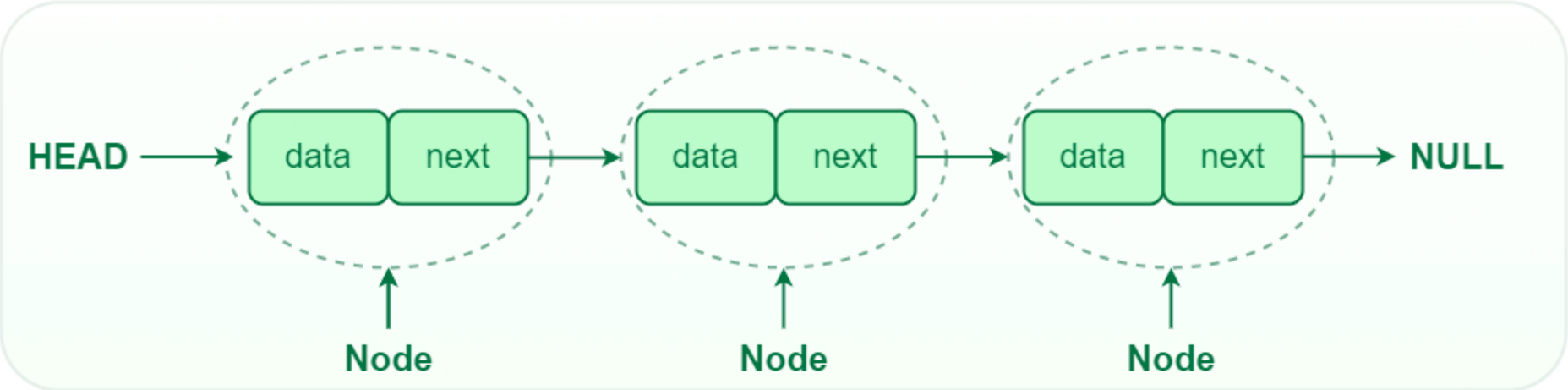
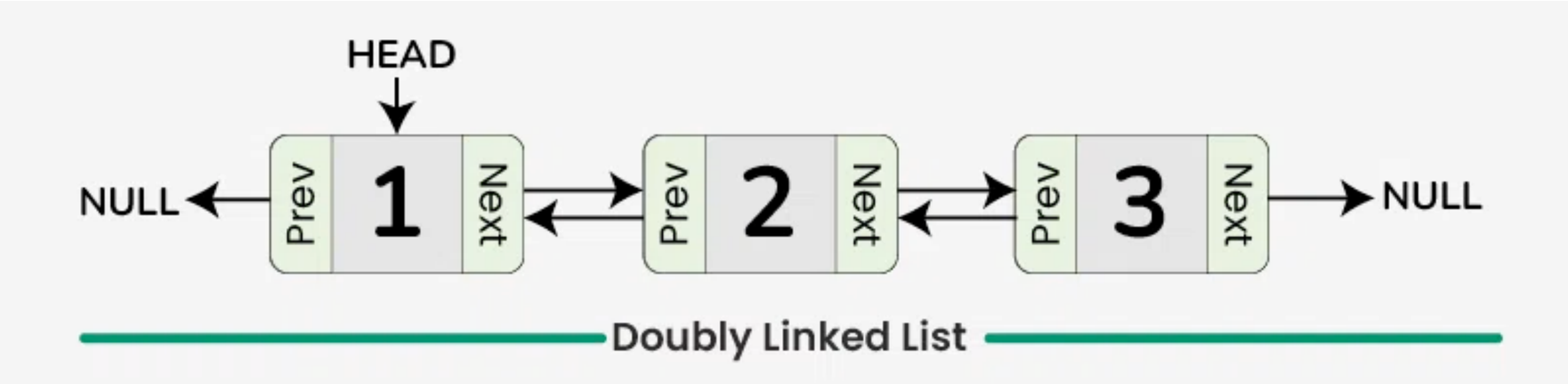
CS62 Class 9: Linked lists (again)

Basic Data Structures



https://www.geeksforgeeks.org/introduction-to-singly-linked-list/



https://www.geeksforgeeks.org/doubly-linked-list/

Review from last week

- For ArrayLists:
  - The amortized run time of add() is O(1). Mostly it runs in O(1), unless you have to call resize() which runs in O(n). However, because we only call resize every $2^{log_2(n-1)}$ times, it averages to O+(1). (See lecture 7 slide 36).
  - What are the run times of the default constructor, isEmpty(), size(), get()?
  - How about remove()? Remove() with an index as a parameter?

Review from last week

- What are the run times of the default constructor, isEmpty(), size(), get()?
  - all O(1)
- How about remove()? Remove() with an index as a parameter?
  - remove is O(1). remove(int index) is O(n). This is because of the shifting elements step, which takes O(n) worst and average case. (On average, you could expect n/2 elements to be shifted, which is still O(n)).

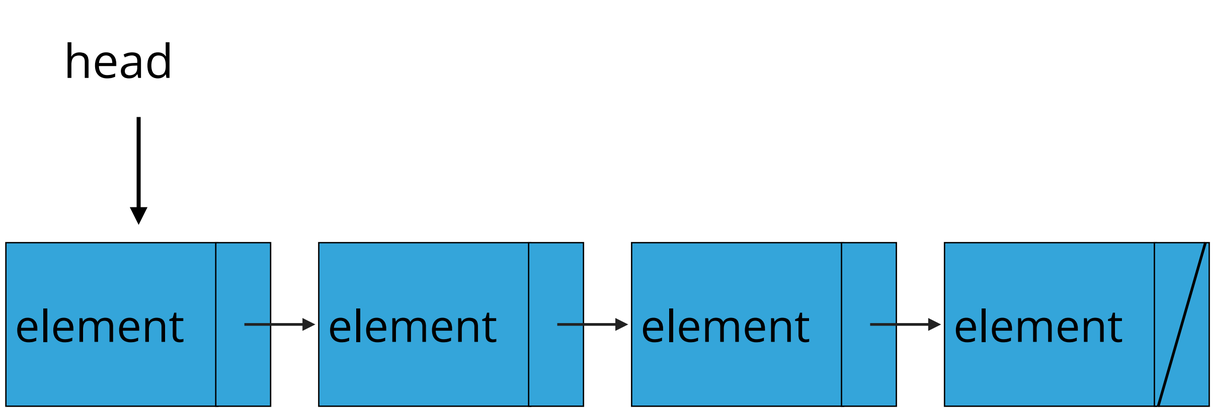
Agenda

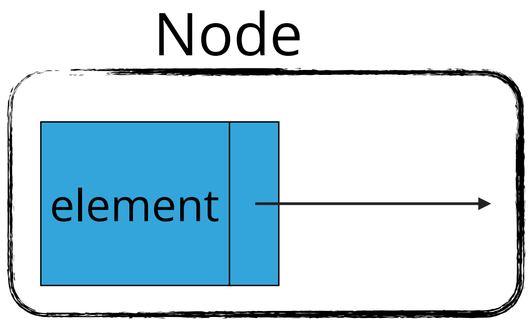
- More linked lists
- Runtime of linked lists operations (are singly and doubly the same?)
- Using built-in linked lists in the Java collections framework
- Brief intro to stacks

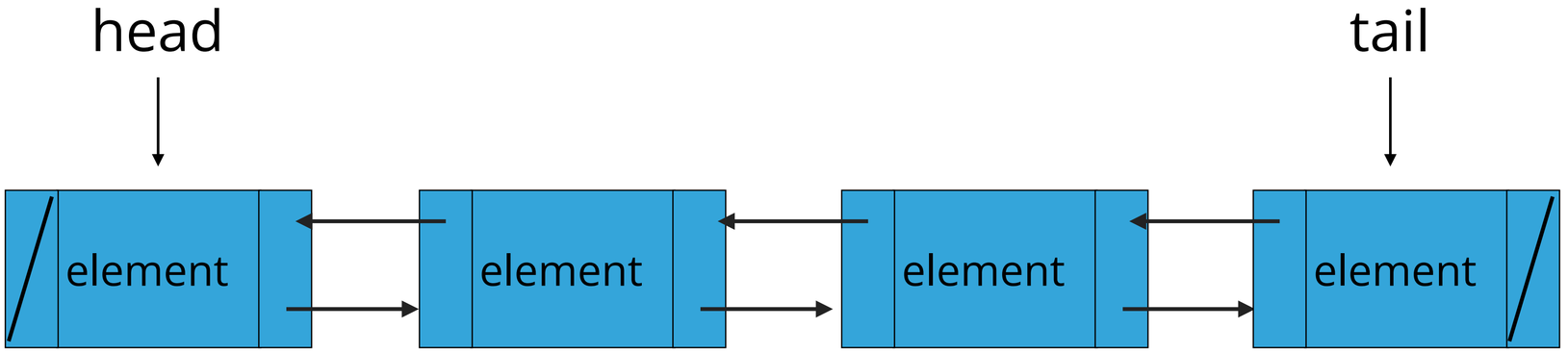
Reminder: Linked Lists

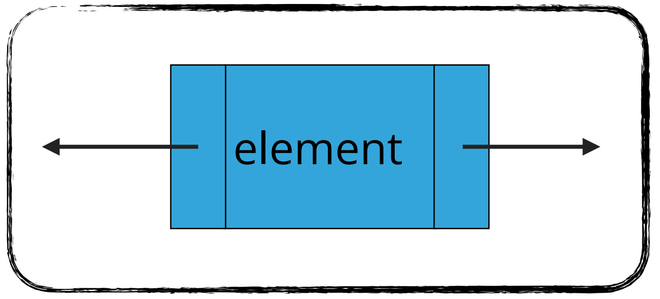
Singly vs doubly linked lists

- Singly linked lists only have a “next” pointer, and a head
  - add() and remove() happen at the head
- Double linked lists have both a “prev” and “next” pointer, and a head and a tail









Implementation: adding

Last time: Insert element at head of doubly linked list

```
/**
 * Inserts the specified element at the head of the doubly linked list.
 *
 * @param element the element to be inserted
 */
public void addFirst(E element) {
  // Create a pointer to head
  Node oldHead = head;

  // Make a new node and assign it to head. Fix pointers and update element
  head = new Node();
  head.element = element;
  head.next = oldHead;
  head.prev = null;

  // if first node to be added, adjust tail to it.
  if (tail == null){
    tail = head;
      }
  else{
     // else fix previous pointer to head
    oldHead.prev = head;
      }
  // increase number of nodes in doubly linked list.
      size++;
 }
```

set prev of new head to null

if it's just one node, also make the tail the node

set prev of old head to our new head

Worksheet time!

- Insert element at the tail for doubly linked lists.

```
/**
 * Inserts the specified element at the tail of the doubly linked list.
 *
 * @param element
 *            the element to be inserted
 */
public void addLast(E element) {
  // Create a pointer to tail

  // Make a new node and assign it to tail. Fix pointers and update element



  // if first node to be added, adjust head to it.


  // else fix next pointer to tail

  // increase number of nodes in doubly linked list.

}
```

Worksheet answers

```
public void addLast(E element) {
    // Save the old node
    Node oldTail = tail;

    // Make a new node and assign it to tail. Fix pointers.
    tail = new Node();
    tail.element = element;
    tail.next = null;
    tail.prev = oldTail;

    // if first node to be added, adjust head to it.
    if (head == null) {
        head = tail;
    } else {
        oldTail.next = tail;
    }
    size++;
}
```

Last time: Insert element at a specified index (SLL)

```
public void add(int index, E element) {
        // check that index is within range
      if (index > size || index < 0){
            throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
        }

        // if index is 0, then call one-argument add
   if (index == 0) {
       add(element);
        // else
   } else {
       // make two pointers, previous and finger. Point previous to null and finger to head
       Node previous = null;
       Node finger = head;
       // search for index-th position by pointing previous to finger and advancing finger
       for (int i=0; i<index; i++){
           previous = finger;
           finger = finger.next;
       }
       // create new node to insert in correct position. Set its pointers and contents
       Node current = new Node();
       current.next = finger;
       current.element = element;
       // make previous point to newly created node.
       previous.next = current;
       // increase number of nodes
       size++;
   }
}
```

When we are at the correct index to insert, finger points to the new element's next element, while previous pointers to the new element's previous element (we are inserting between previous and finger)

Insert element at a specified index (DLL)

- It's the same idea, we just need to move more prev pointers around

```
public void add(int index, E element) {
        // check whether index is valid
           if (index > size || index < 0){
            throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
         }
         // if index is 0, call addFirst
   if (index == 0) {
      addFirst(element);
         // if index is n, call addLast
   } else if (index == size()) {
      addLast(element);
         // else
   } else {
      // Make two new Node references, previous and finger. Set previous to null and finger to head
      Node previous = null;
      Node finger = head;
      // search for index-th position. Set previous to finger and move finger to next position
      for (int i=0; i< index; i++){
         previous = finger;
         finger = finger.next;
      }
      // create new Node, update its element, and fix its pointers taking into account where finger and previous are
      Node current = new Node();
      current.element = element;
      current.next = finger;
      current.prev = previous;
      previous.next = current;
      finger.prev = current;
      // increase number of nodes
      size++;
   }
}
```

First or last

same

make sure to change prev too!

Implementation: setting/ removing

Replace element at a specified index

```
  public E set(int index, E element) {
            // check that index is within range
           if (index >= size || index < 0){
                throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
             }

           Node finger = head;
           // search for index-th position by pointing previous to finger and advancing
finger
           for (int i=0; i< index; i++){
             finger = finger.next;
         }
         // reference old element
         E old = finger.element;
         // update element at finger
         finger.element = element;
         // return old element
         return old;
      }
  }
```

Q: Is this for singly linked lists or doubly linked lists?

A: It's the same for both!

Retrieve and remove head (SLL)

```java
 /**
  * Removes and returns the head of the singly linked list.
  *
  * @return the head of the singly linked list.
  */
public E remove() {
   // Make a temporary pointer to head
   Node temp = head;
   // Move head one to the right
   head = head.next;
   // Decrease number of nodes
   size--;
   // Return element held in the temporary pointer
   return temp.element;
}
```

Retrieve and remove head (DLL)

```
/**
 * Removes and returns the head of the doubly linked list.
 *
 * @return the head of the doubly linked list.
 */
public E removeFirst() {
  // Create a pointer to head
  Node temp = head;
  // Move head to next
  head = head.next;
  // if there was only one node in the doubly linked list
  if (head == null) {
    tail = null
  } else {
        head.prev = null;
  }
  // decrease number of nodes
  size--;
  // return old head's element
  return temp.element;
}
```

only this if statement block is different

Worksheet time!

- Remove the element at the tail for doubly linked lists. (Less guidance now!)

```
  /**
   * Removes and returns the tail of the doubly
linked list.
   *
   * @return the tail of the doubly linked list.
   */
  public E removeLast() {


  }
```

Worksheet answers

```
public E removeLast() {
  // Create a pointer to tail
  Node temp = tail;
  // Move tail to previous
  tail = tail.prev;
  // if removed the last node
  if (tail == null) {
      // set head to null
    head = null;
  // else
  } else {
      // set new tail's next to null
    tail.next = null;
  }
  // decrease number of nodes
  size--;
  // return old tail's element
  return temp.element;
}
```

Clear the linked list of all elements

```
/**
 * Clears the singly linked list of all elements.
 *
   */
public void clear(
      head = null;
      size = 0;

}
```

```
/**
 * Clears the doubly linked list of all elements.
 *
   */
public void clear(
    head = null;
    tail = null;
    size = 0;

}
```

Retrieve and remove element from a specific index (SLL)

```
public E remove(int index) {
        // check that index is within range
     if (index >= size || index < 0){
          throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
     }
        // if index is 0, then call remove
    if (index == 0) {
        return remove();
        // else
    } else {
        // make two pointers, previous and finger. Point previous to null and finger to head
        Node previous = null;
        Node finger = head;
        // search for index-th position by pointing previous to finger and advancing finger
        for (int i=0; i< index; i++){
            previous = finger;
            finger = finger.next;
        }
        // make previous point to finger's next
        previous.next = finger.next;
        // reduce number of elements
        size--;
        // return finger's element
        return finger.element;
    }

}
```

Same idea as Q1 on last lecture's worksheet

Worksheet time!

- Write remove at a specific index but for DLL.

```
/**
 * Removes and returns the element at the specified index.
 *
 * @param index
 *            the index of the element to be removed
 * @return the element previously at the specified index
 * @pre 0<=index<size
 */
public E remove(int index) {
        // check whether index is valid

       // if index is 0

              // return removeFirst

       // else if index is size-1

              // return removeLast

       // else
              // Make two new Node references, previous and finger. Set previous to null and finger to head

      // search for index-th position. Set previous to finger and move finger to next position



      // update pointers for previous and finger


      // decrease number of nodes

      // return the element that finger points to


}
```

Worksheet answers

```
public E remove(int index) {
   // check whether index is valid
    if (index >= size || index < 0) {
         throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
    }
  // if statements
    if (index == 0) {
       return removeFirst();
    } else if (index == size - 1) {
        return removeLast();
    } else {
     // Make two new Node references, previous and finger. Set previous to null and finger to head
     Node previous = null;
     Node finger = head;
     // search for index-th position. Set previous to finger and move finger to next position
     for (int i=0; i< index; i++){
       previous = finger;
       finger = finger.next;
     }
     // update pointers for previous and finger
     previous.next = finger.next;
     finger.next.prev = previous;
     // decrease number of nodes
     size--;
     // return the element that finger points to
     return finger.element;
   }
}
```

Those are all the methods!

- `SinglyLinkedList()`: Constructs an empty singly linked list.
- `DoublyLinkedList()`: Constructs an empty doubly linked list.
- `isEmpty()`: Returns true if the linked list does not contain any element.
- `size()`: Returns the number of elements in the linked list.
- `E get(int index)`: Returns the element at the specified index.
- `add(E element)`: Inserts the specified element at the head of the linked list.
  - `addFirst(E element)`: Inserts the specified element at the head of the **doubly** linked list.
  - `addLast(E element)`: Inserts the specified element at the tail of the **doubly** linked list.
- `add(int index, E element)`: Inserts the specified element at the specified index.
- `E set(int index, E element)`: Replaces the specified element at the specified index and returns the old element
- `E remove()`: Removes and returns the head of the linked list.
  - `E removeLast()`: Removes and returns the tail of the **doubly** linked list.
- `E remove(int index)`: Removes and returns the element at the specified index.
- `clear()`: Removes all elements.

Run time

add() in singly linked lists is $O(1)$ for worst case

```java
public void add(E element) {
    // Save the old node
    Node oldfirst = head;

    // Make a new node and assign it to head. Fix pointers.
    head = new Node();
    head.element = element;
    head.next = oldfirst;

    size++; // increase number of nodes in singly linked list.
}
```

`addFirst()` in doubly linked lists is $O(1)$ for worst case

```
public void addFirst(E element) {
       // Save the old node
       Node oldHead = head;

       // Make a new node and assign it to head. Fix pointers.
       head = new Node();
       head.element = element;
       head.next = oldHead;
       head.prev = null;

       // if first node to be added, adjust tail to it.
       if (tail == null)
          tail = head;
       else
          oldHead.prev = head;

       size++; // increase number of nodes in doubly linked list.
   }
```

addLast() in doubly linked lists is $O(1)$ for worst case

```
public void addLast(E element) {
       // Save the old node
       Node oldTail = tail;

       // Make a new node and assign it to tail. Fix pointers.
       tail = new Node();
       tail.element = element;
       tail.next = null;
       tail.prev = oldTail;

       // if first node to be added, adjust head to it.
       if (head == null)
          head = tail;
       else
          oldTail.next = tail;

       size++;
   }
```

get() in singly linked lists is $O(n)$ for worst case

```
public E get(int index) {
    if (index >= size || index < 0){
        throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
    }

    Node finger = head;
    // search for index-th element or end of list
    for (int i=0; i< index; i++){
        finger = finger.next;
    }
    return finger.element;
}
```

Same for DLL?

Yes!

`add(int index, E element)` in singly linked lists is $O(n)$ for worst case

```
public void add(int index, E element) {
         if (index > size || index < 0){
                throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
         }
        if (index == 0) {
            add(element);
        } else {

            Node previous = null;
            Node finger = head;
            // search for index-th position
            for (int i=0; i< index; i++){
                previous = finger;
                finger = finger.next;
            }
            // create new value to insert in correct position.
            Node current = new Node();
            current.next = finger;
            current.element = element;
            // make previous value point to new value.
            previous.next = current;

            size++;
        }
    }
```

Same for DLL?

Yes!

`set(int index, E element)` in linked lists is $O(n)$ for worst case

```
/**
 * Inserts the specified element at the specified index.
 *
 * @param index
 *            the index of the element to replace
 * @param element
 *            the element to be stored at the specific index
 * @return the old element that was replaced
 * @pre 0<=index<size
 */
public E set(int index, E element) {
        // check that index is within range
      if (index >= size || index < 0){
            throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
        }

        Node finger = head;
        // search for index-th position by pointing previous to finger and advancing finger
         for (int i=0; i< index; i++){
             finger = finger.next;
         }
         // reference old element
         E old = finger.element;
         // update element at finger
         finger.element = element;
         // return old element
         return old;
    }
}
```

`remove()` in singly linked lists is $O(1)$ for worst case

```
public E remove() {
    Node temp = head;
    // Fix pointers.
    head = first.next;

    size--;

    return temp.element;
}
```

Same (removeFirst, removeLast) for DLL?

Yes!

`remove(int index)` in singly linked lists is $O(n)$ for worst case

```
public E remove(int index) {
     if (index >= size || index < 0){
            throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
        }

    if (index == 0) {
        return remove();
    } else {
        Node previous = null;
        Node finger = head;
        // search for value indexed, keep track of previous
        for (int i=0; i< index; i++){
            previous = finger;
            finger = finger.next;
        }
        previous.next = finger.next;

        size--;
        // finger's value is old value, return it
        return finger.element;
    }

}
```

Same for DLL?

Yes!

`clear()` in singly linked lists is $O(1)$ for worst case

```
/**
 * Clears the singly linked list of all elements.
 *
  */
public void clear(
     head = null;
     size = 0;

}
```

Same for DLL?

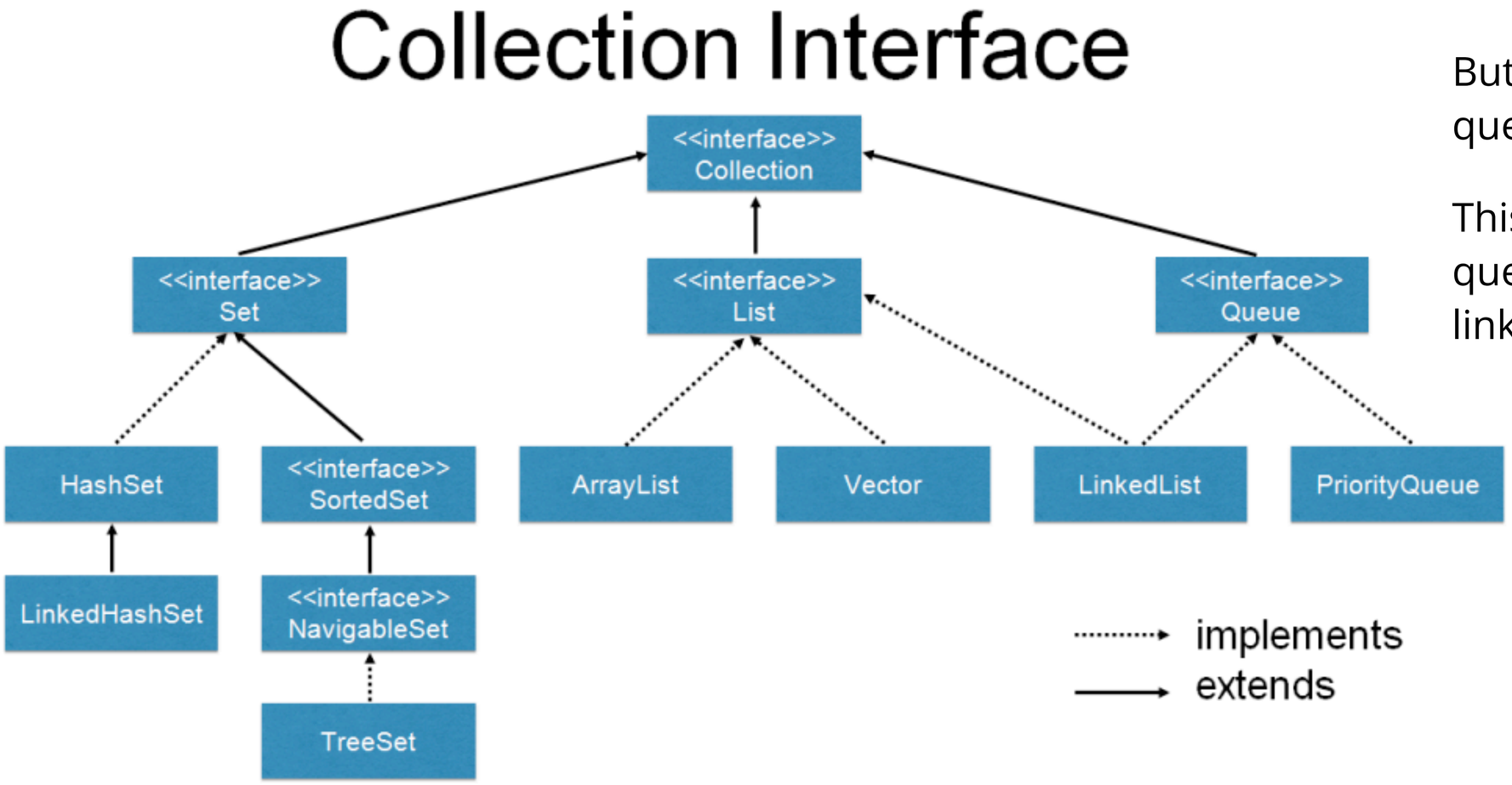
Yes!

Run time summary

- If we are iterating through the linked list to find a specific Node (“finger”), O(n) worst case
  - get(), set(), add() with index, remove() with index
- If we are just moving pointers around, O(1) worst case
  - add() from the head, remove() from the head, clear()

Using pre-built linked lists (Java Collections Framework)

LinkedLists also implement the List interface



But they also implement the queue interface!

This is because you can build queues (and stacks) with linked lists.

Using linked lists

- Doubly linked list implementation of the List and Deque (next lecture) interfaces.

```
import java.util.LinkedList;


public class LinkedList<E> extends
AbstractSequentialList<E> implements List<E>, Deque<E>
```

You'll be using this for HW4 in implementing a stack, so get used to its API!

https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Constructor Summary

Constructors

Constructor and Description
`LinkedList()` Constructs an empty list.
`LinkedList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`add(E e)` Appends the specified element to the end of this list.
`void`	`add(int index, E element)` Inserts the specified element at the specified position in this list.
`boolean`	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
`boolean`	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
`void`	`addFirst(E e)` Inserts the specified element at the beginning of this list.
`void`	`addLast(E e)` Appends the specified element to the end of this list.
`void`	`clear()` Removes all of the elements from this list.
`Object`	`clone()` Returns a shallow copy of this `LinkedList`.
`boolean`	`contains(Object o)` Returns `true` if this list contains the specified element.
`Iterator<E>`	`descendingIterator()` Returns an iterator over the elements in this deque in reverse sequential order.
`E`	`element()` Retrieves, but does not remove, the head (first element) of this list.
`E`	`get(int index)` Returns the element at the specified position in this list.

https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

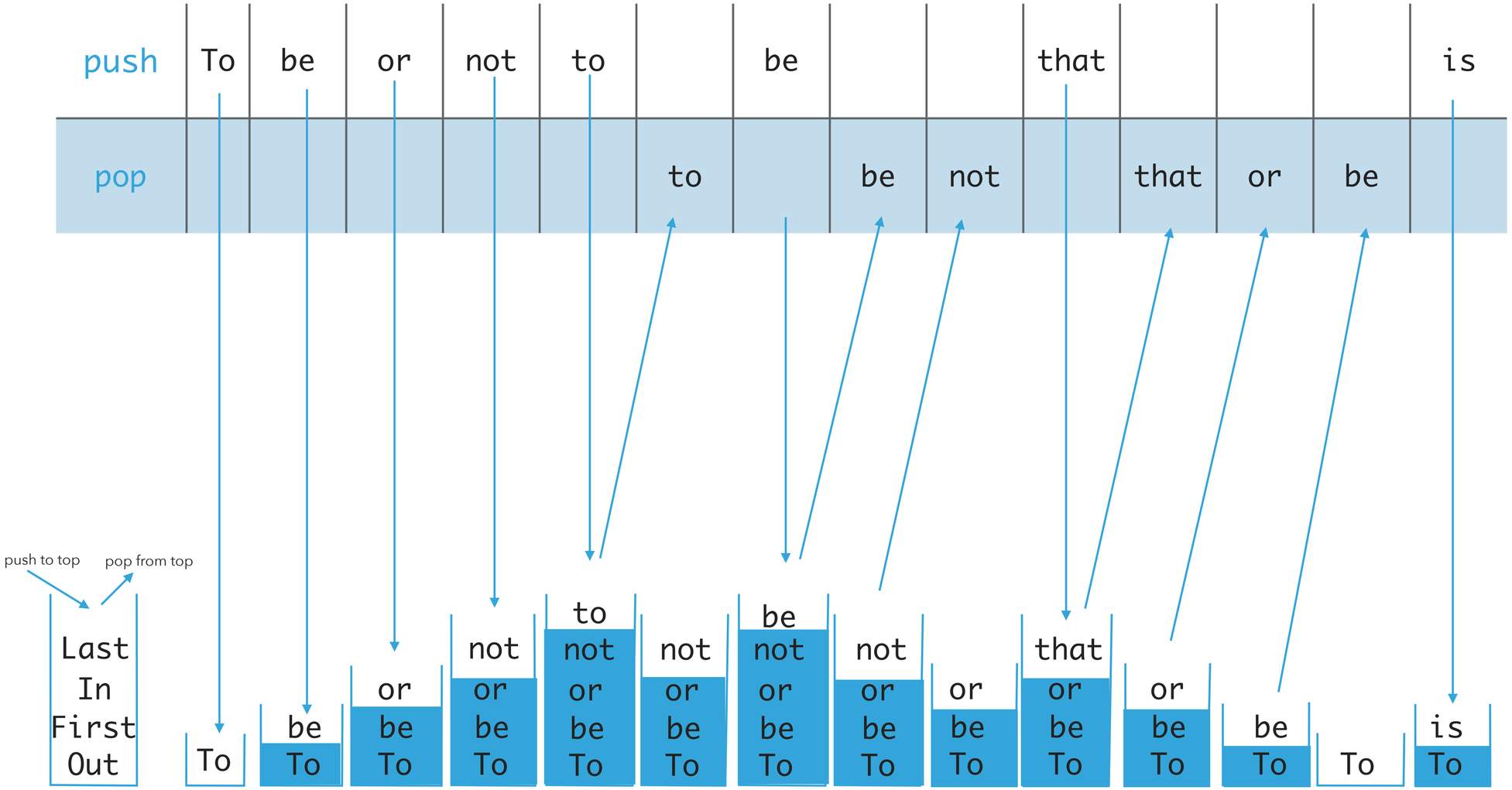
Brief intro to stacks

Stacks

- Dynamic linear data structures.
- Elements are inserted and removed following the LIFO paradigm.
- LIFO: Last In, First Out.
  - Remove the most recent element.
- Similar to lists, there is a sequential nature to the data.

- Metaphor of cafeteria plate dispenser.
  - Want a plate? Pop the top plate.
  - Add a plate? Push it to make it the new top.
  - Want to see the top plate? Peek.
  - We want to make push and pop as time efficient as possible.

Example of stack operations



Lecture 9 wrap-up

- Don’t forget to submit your own custom creature in Darwin Part II (due 11:59pm tonight). And don’t forget that it needs to loop.
  - Winners will be announced in Thursday’s lecture and presented with a copy of the board game “The Crew”
- Extra OH Weds 3-4pm by appointment (Slack me)
- We will be trying bucket grading for this week’s quiz (so you stress about your grades less :))
- Lab (debugger), HW4 (calculator, uses JCF’s linked lists to build a stack) released

Resources

- Linked lists from the textbook: https://algs4.cs.princeton.edu/13stacks/
- Oracle Linked Lists: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
- See slides following this for 2 more practice problems

Bonus practice problem

- Add a deleteLast method in the SinglyLinkedList class that removes the last node of a singly linked list. Think of edge cases.

Bonus answer

```
public void deleteLastNode() {
        if (!isEmpty()) {
            if (size == 1) {
                head = null;
            } else {
                Node current = head;
                for (int i = 0; i < size - 2; i++) {
                    current = current.next;
                }
                current.next = null;
            }
            size--;
        }
    else{//throw some appropriate exception}
    }
```

Bonus practice problem 2

- Add a method `removeAfter(Node node)` in the DoublyLinkedList class that removes the node following the given one.

Bonus answer 2

```
public void removeAfter(Node node) {
    if (isEmpty() || node == null) {
        return;
    }
    for (Node current = head; current != null; current = current.next) {
        if (current == node) {
            if (current.next != null) {
                current.next = current.next.next;
                if(current.next != null){
                    current.next.prev = current;
                }
                size--;
            }
            break;
        }
    }
}
```