administrivia...
- assignment 5 due tonight

- assignment 6 is out
  - due next Wednesday
  - is a SOLO assignment

- midterm next Monday in class
  - will cover through sorting algorithms
  - will NOT include Monday’s material
  - review in lab on Friday
last time...
linked structures
-linked structures are data storage in which individual items have *links* (references) to other items

-items don’t reside in a single contiguous block of memory

-items can be *dynamically* added or removed from the structure, simply by creating or destroying links
- Linked structures are data storage in which individual items have *links* (references) to other items.

- Items don’t reside in a single contiguous block of memory.

- Items can be *dynamically* added or removed from the structure, simply by creating or destroying links.

*How is this different than an array?*
-we’ve seen a list implemented with an array in ArrayList

-a linked list is another way to implement a list

-each node, or item in the list, has a link to the next item in the list

-a single node consists of some data and a reference to another node
inserting into an array:

inserting into a linked list:
inserting into an array:

\[
\begin{array}{c}
5 \\
9 \\
12 \\
17 \\
25 \\
8
\end{array}
\]

\[
\begin{array}{c}
5 \\
8 \\
9 \\
12 \\
17 \\
25
\end{array}
\]

inserting into a linked list:
inserting into an array:

```
5  9  12  17  25
```

8

```
5  8  9  12  17  25
```

what is the cost of insertion?
A) c
B) log N
C) N
D) N log N
E) N^2
F) N^3

inserting into a linked list:
inserting into an array:

```
| 5 | 9 | 12 | 17 | 25 |
```

what is the cost of insertion?
A) c
B) log N
C) N
D) N log N
E) N^2
F) N^3

inserting into a linked list:

```
5 → 9 → 12 → 17 → 25
```

8
inserting into an array:

| 5 | 9 | 12 | 17 | 25 |

inserting into a linked list:

what is the cost of insertion?

A) c  
B) log N  
C) N  
D) N \log N  
E) N^2  
F) N^3
deletion from a linked list:
deletion from a linked list:
deletion from a linked list:

5 → 9 → 12 → 17 → 25

9 is now stranded — garbage collector will clean it up
non-generic implementation (only stores floats):

```java
class LinkedList {
    private Node head;
    private int size;

    private class Node {
        private float data;
        private Node next;
        ...
    }
    ...
}
```
things to consider...

- what should `next` be for the last item in the list?

- don’t let a call to `new ListNode()` cause an infinite loop
  - ie. creating a new `ListNode`, which creates a new `ListNode`, and so on...

- constructor should set `next` to `null`
traversing a linked list:

```java
boolean contains(float item) {
    Node temp = head;

    while(temp != null) {
        if(temp.data == item)
            return true;
        temp = temp.next;
    }

    return false;
}
```
doubly-linked lists
-nodes have a link to next and previous node

-allows for traversal in either forward or reverse order

-maintains a tail node as well as a head node

-why?

-how can we use a doubly-linked list to optimize get(i)?
special cases (empty or single-item lists) are more tricky due to managing tail as well as head

what are the values of head and tail for any empty list?

what about for a single-item list?
doubly-linked list insertion:
doubly-linked list insertion:

```java
newNode = new Node<Character>();
newNode.data = 'n';
```
doubly-linked list insertion:

```java
newNode = new Node<Character>();
newNode.data = 'n';
```

really, we want this to be generic
doubly-linked list insertion:

```java
newNode = new Node<Character>();
newNode.data = 'n';
newNode.prev = current;
```

really, we want this to be generic
doubly-linked list insertion:

```java
doubly-linked list insertion:

newNode = new Node<Character>();
newNode.data = 'n';
newNode.prev = current;
newNode.next = current.next;
```

really, we want this to be generic
doubly-linked list insertion:

```java
newNode = new Node<Character>();
newNode.data = 'n';

newNode.prev = current;
newNode.next = current.next;
newNode.prev.next = newNode;
```

really, we want this to be generic
doubly-linked list insertion:

```java
newNode = new Node<Character>();
newNode.data = 'n';
newNode.prev = current;
newNode.next = current.next;
newNode.prev.next = newNode;
newNode.next.prev = newNode;
```

really, we want this to be generic
doubly-linked list insertion:

```java
doublylinkedlistinsertion:
newNode = new Node<Character>();
newNode.data = 'n';
newNode.prev = current;
newNode.next = current.next;
newNode.prev.next = newNode;
newNode.next.prev = newNode;
```

really, we want this to be generic

what is the cost of insertion?

A) c
B) log N
C) N
D) N log N
E) N^2
F) N^3

head
```
```
c current
```
```
```
tail
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
doubly-linked list deletion:

current.prev.next = current.next;
current.next.prev = current.prev;
doubly-linked list deletion:

current.prev.next = current.next;
current.next.prev = current.prev;

head

a

→
c

→

k

→

n

→

o

→
tail

y
doubly-linked list deletion:

current.prev.next = current.next;
current.next.prev = current.prev;

head

a

c

k

o

tail

y

current

n
doubly-linked list deletion:

```java
current.prev.next = current.next;
current.next.prev = current.prev;
```

![Diagram of a doubly-linked list with nodes labeled 'a', 'c', 'k', 'n', 'o', and 'y'. The current node is marked with an oval, and the head and tail markers are shown.]
doubly-linked list deletion:

current.prev.next = current.next;
current.next.prev = current.prev;

what is the cost of deletion?
A) c
B) log N
C) N
D) N log N
E) N^2
F) N^3
generic implementation:

class DoublyLinkedList<E> {
  private Node head;
  private Node tail;
  private int size;

  private class Node {
    private E data;
    private Node next;
    private Node prev;
    ...
  }

  ...
}

...
things to consider...

- adding to the front or end of a linked list is a little different than adding somewhere in the middle
  - why?

- removing from a list with 1 node
  - what happens to head/tail?

- adding to an empty list
  - what is the current value of head/tail?
LinkedList vs ArrayList
**LinkedList vs ArrayList**

- **insertion & deletion:**
  - (assuming position is known)
  - **LinkedList:** O(c)  
  - **ArrayList:** O(N)

- **accessing a random item:**
  - **LinkedList:** O(N)  
  - **ArrayList:** O(c)

- choose the structure based on the expected use
  - what is the common case?

- what if insertion / deletion is always from the front / end?
today...
midterm review
bubble sort
the (usually) most inefficient sorting algorithm
Compare each pair of adjacent items and swap them if necessary. Repeat.
bubble sort

1) for each item, compare it to its next neighbor and swap if necessary

2) repeat step 1 until sorted
selection sort
the simplest sorting algorithm
Find (ie. *select*) the smallest item in the unsorted portion of the array and move to the end of the sorted portion of the array.
selection sort

1) find the minimum item in the unsorted part of the array
2) swap it with the first item in the unsorted part of the array
3) repeat steps 1 and 2 to sort the remainder of the array
insertion sort

good for small $N$
Take the first item in the unsorted portion of the array and *insert* it into the sorted portion of the array.
insertion sort

1) the first array item in the unsorted array is the sorted portion of the array

2) take the second item and insert it in the sorted portion

3) repeat steps 1 and 2 to sort the remainder of the array
shellsort
the simplest subquadratic sorting algorithm
Divide the array (smartly) into subarrays. Do insertion sort on the subarrays. Repeat.

* Take the first item in the unsorted portion of the array and insert it into the sorted portion of the array.
1) set the **gap size** to $N/2$

2) consider the subarrays with elements at **gap size** from each other

3) do insertion sort on each of the subarrays

4) divide the **gap size** by 2

5) repeat steps 2 — 4 until the **gap size** is <1
mergesort
divide and conquer
Merge sorted subarrays together.
mergesort

1) divide the array in half
2) sort the left half
3) sort the right half
4) merge the two halves together
mergesort

1) divide the array in half
2) sort the left half
3) sort the right half
4) merge the two halves together

2) take the left half, and go back to step 1
3) take the right half, and go back to step 1
mergesort

1) divide the array in half
2) sort the left half
3) sort the right half
4) merge the two halves together

2) take the left half, and go back to step 1 until???
3) take the right half, and go back to step 1 until???
quicksort
another divide and conquer
Move all small items to the first subarray, move all large items to the second subarray. Sort each subarray.
quicksort

1) select an item in the array to be the *pivot*

2) *partition* the array so that all items less than the pivot are to the left of the pivot, and all the items greater than the pivot are to the right

3) sort the left half

4) sort the right half
quicksort

1) select an item in the array to be the *pivot*

2) *partition* the array so that all items less than the pivot are to the left of the pivot, and all the items greater than the pivot are to the right

3) take the left half, and go back to step 1

4) take the right half, and go back to step 1
quicksort

1) select an item in the array to be the pivot

2) partition the array so that all items less than the pivot are to the left of the pivot, and all the items greater than the pivot are to the right

3) take the left half, and go back to step 1 until???

4) take the right half, and go back to step 1 until???
recursion

[ri-kur-zhuh n]
noun

see recursion.
Towers of Hanoi
void towers(n, from, to, spare) {
    if ( n == 1)
        printOutMove(from, to)
    else
    {
        1. solve smaller problem
        2. solve basic problem
        3. solve smaller problem
    
    }
}

void towers(n, from, to, spare) {
    if (n == 1)
        printOutMove(from, to)
    else
    {
        towers(n-1, from, spare, to)
        towers(1, from, to, spare)
        towers(n-1, spare, to, from)
    }
}
topics

- Java basics
  - variables, types
  - control flow
  - reference types
  - classes, methods

-OOP
  - inheritance
  - polymorphism
  - interfaces
  - super
topics

- generics
  - wild cards
  - generic classes

- comparators

- Collections

- Iterators
topics
- algorithm analysis
  - growth rates
  - Big-O
  - determining complexity of loops and algorithms
  - recursion

- bubble sort
- selection sort
- insertion sort
- shellsort
- mergesort
- quicksort

be able to reason about performance and behavior of each!
test format

-problems may be of the following types:
  - short answer
  - determining output of code
  - writing code
  - filling in missing code
  - multiple choice
  - true / false
next time...
- **reading for next Wednesday**
  - chapter 16: stacks & queues
  - chapter 2: array-based lists
    - [http://opendatastructures.org/ods-java/](http://opendatastructures.org/ods-java/)

- **homework**
  - assignment 5 due tonight
  - assignment 6 is out