CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 14: Midterm #1 Review

Midterm Procedure

- Exam is during normal class time. Same time, same place.
- Seating is assigned randomly
  - Wait outside the room until instructed to enter
  - Immediately place all bags/electronics at the front of the room
- At your seat you should have:
  - Writing utensil
  - UB ID card
  - One 8.5x11 cheatsheet (front and back) if desired
  - Summation/Log rules will be provided

Content Overview

	Analysis Tools/Techniques	ADTs	Data Structures
Week 2/3	Asymptotic Analysis, (Unqualified) Runtime Bounds		
Week 3		Sequence	Array, LinkedList
Week 4	Amortized Runtime	List	ArrayList, LinkedList

Analysis Tools and Techniques

Recap of Runtime Complexity

Big-Θ — Tight Bound

- Growth functions are in the **same** complexity class
- If f(n) ∈ **Θ**(g(n)) then an algorithm taking f(n) steps is as "exactly" as fast as one that takes g(n) steps.

Big-O — Upper Bound

- Growth functions in the **same or smaller** complexity class.
- If f(n) ∈ O(g(n)), then an algorithm that takes f(n) steps is *at least as fast* as one taking g(n) (but it may be even faster).

Big-Ω — Lower Bound

- Growth functions in the **same or bigger** complexity class
- If f(n) ∈ **Ω**(g(n)), then an algorithm that takes f(n) steps is *at least as slow* as one that takes g(n) steps (but it may be even slower)

Common Runtimes (in order of complexity)

Constant Time:	$\Theta(1)$
Logarithmic Time:	$\Theta(\log(n))$
Linear Time:	$\Theta(n)$
Quadratic Time:	$\Theta(n^2)$
Polynomial Time:	$\Theta(n^k)$ for some $k > 0$
Exponential Time:	$\Theta(c^n)$ (for some $c \geq 1$)

Formal Definitions

$\mathbf{f(n)} \in O(\mathbf{g(n)})$ iff exists some constants $\mathbf{c}$, $\mathbf{n_0}$ s.t.

$\mathbf{f(n) \leq c * g(n)}$ for all $\mathbf{n > n_0}$

$\mathbf{f(n)} \in \Omega(\mathbf{g(n)})$ iff exists some constants $\mathbf{c}$, $\mathbf{n_0}$ s.t.

$\mathbf{f(n) \geq c * g(n)}$ for all $\mathbf{n > n_0}$

$\mathbf{f(n)} \in \Theta(\mathbf{g(n)})$ iff $\mathbf{f(n)} \in O(\mathbf{g(n)})$ and $\mathbf{f(n)} \in \Omega(\mathbf{g(n)})$

Amortized Runtime

If n calls to a function take $O(f(n))$...

We say the **<u>Amortized Runtime</u>** is $O(f(n) / n)$

The **<u>amortized runtime</u>** of `add` on an `ArrayList` is: $O(n/n) = O(1)$
The **<u>unqualified runtime</u>** of `add` on an `ArrayList` is: $O(n)$

What guarantees do you get?

If *f*(*n*) is a Tight Bound (Big Theta)

The algorithm **always** runs in ***cf*(*n*)** steps

If *f*(*n*) is a Worst-Case Bound (Big O)

The algorithm **always** runs in *at most* ***cf*(*n*)**

← Unqualified runtime

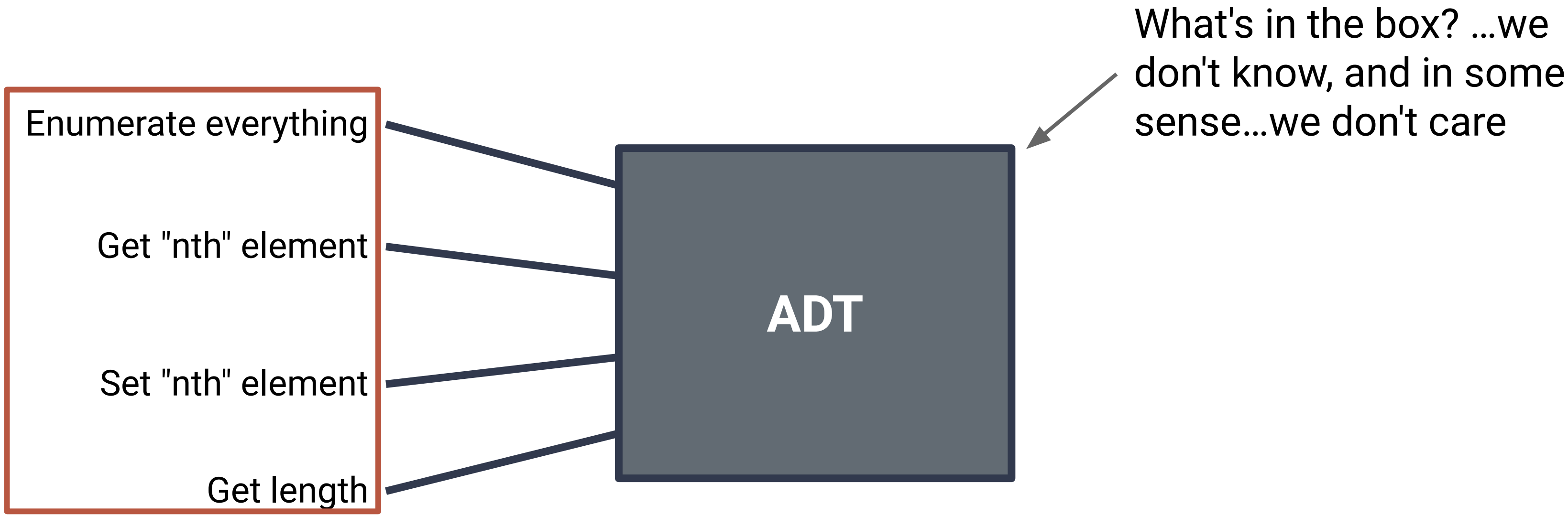
If *f*(*n*) is an Amortized Bound

n invocations of the algorithm **always** run in ***cnf*(*n*)** steps

ADTs and Data Structures

Abstract Data Types (ADTs)

The specification of **what** a data structure can do



Usage is governed by **what** we can do, not **how** it is done

Abstract Data Type vs Data Structure

ADT

The interface to a data structure

*Defines **what** the data structure can do*

Many data structures can implement the same ADT

Data Structure

The implementation of one (or more) ADTs

*Defines **how** the different tasks are carried out*

Different data structures will excel at different tasks

Abstract Data Type vs Data Structure

ADT

The interface to ...

*Defines **what** the...*

can...

Many data st...

implement th...

Data Structure

...tation of one (or

...) ADTs

...e different tasks

...ried out

...ructures will excel

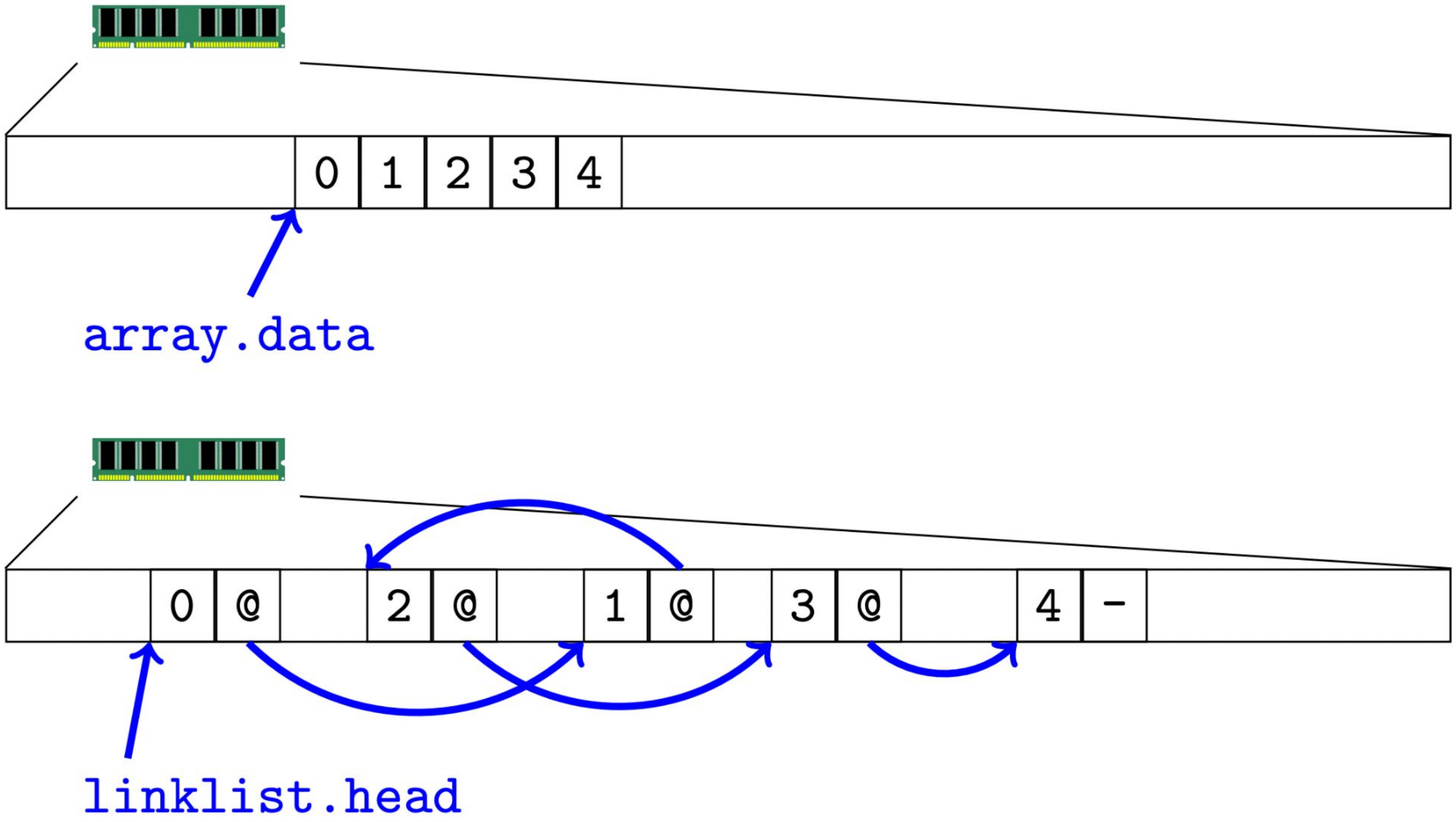
at different tasks

Think about the Linked List we are implementing for PA1.

The internal structure and the mental model of our sequence are very different.

The Sequence ADT

```
public interface Sequence<E> {
  public E get(idx: Int);
  public void set(idx: Int, E value);
  public int size();
  public Iterator<E> iterator();
}
```



Arrays and Linked Lists in Memory

The List ADT

```java
public interface List<E>
    extends Sequence<E> { // Everything a sequence has, and...
  /** Extend the sequence with a new element at the end */
  public void add(E value);

  /** Extend the sequence by inserting a new element */
  public void add(int idx, E value);

  /** Remove the element at a given index */
  public void remove(int idx);
}
```

Runtime Summary

	ArrayList	Linked List (by index)	Linked List (by reference)
`get(...)`	$\Theta(1)$	$\Theta(\text{idx})$ or $O(n)$	$\Theta(1)$
`set(...)`	$\Theta(1)$	$\Theta(\text{idx})$ or $O(n)$	$\Theta(1)$
`size()`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
`add(...)`	$O(n)$, Amortized $\Theta(1)$	$\Theta(\text{idx})$ or $O(n)$	$\Theta(1)$
`remove(...)`	$O(n)$	$\Theta(\text{idx})$ or $O(n)$	$\Theta(1)$