Lecture 2: CSE 373

Data Structures and Algorithms

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Whitaker Brand, Stuart Reges, Zora Fung, Justin Hsia, and many others for sample slides and materials ...

Warm Up – Discuss with your neighbors!

From last lecture:

- What is an ADT?
- What is a data structure?

From CSE 143:

- What is a “linked list” and what operations is it best at?
- What is a “stack” and what operations is it best at?

Review: Interfaces

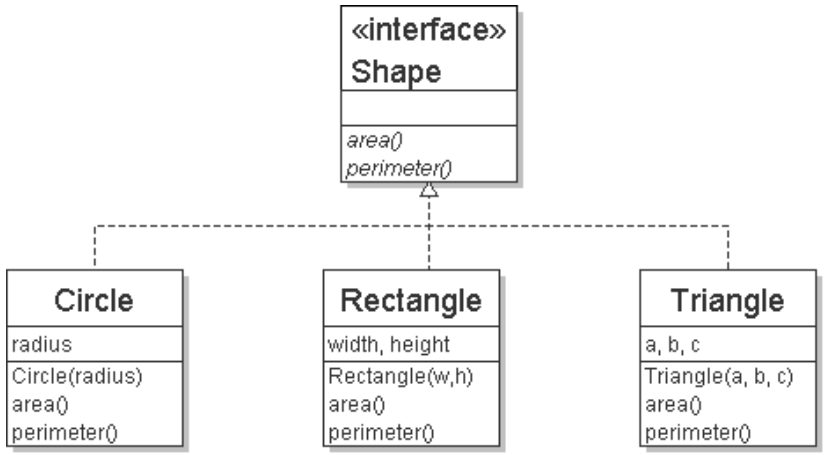
interface: A list of methods that a class promises to implement.

- Interfaces give you an is-a relationship *without* code sharing.
  - A `Rectangle` object can be treated as a `Shape` but inherits no code.
- Analogous to non-programming idea of roles or certifications:
  - "I'm certified as a CPA accountant.
    This assures you I know how to do taxes, audits, and consulting."
  - "I'm 'certified' as a Shape, because I implement the Shape interface.
    This assures you I know how to compute my area and perimeter."

```
public interface name {
    public type name(type name, ..., type name);
    public type name(type name, ..., type name);
    ...
    public type name(type name, ..., type name);
}
```

Example

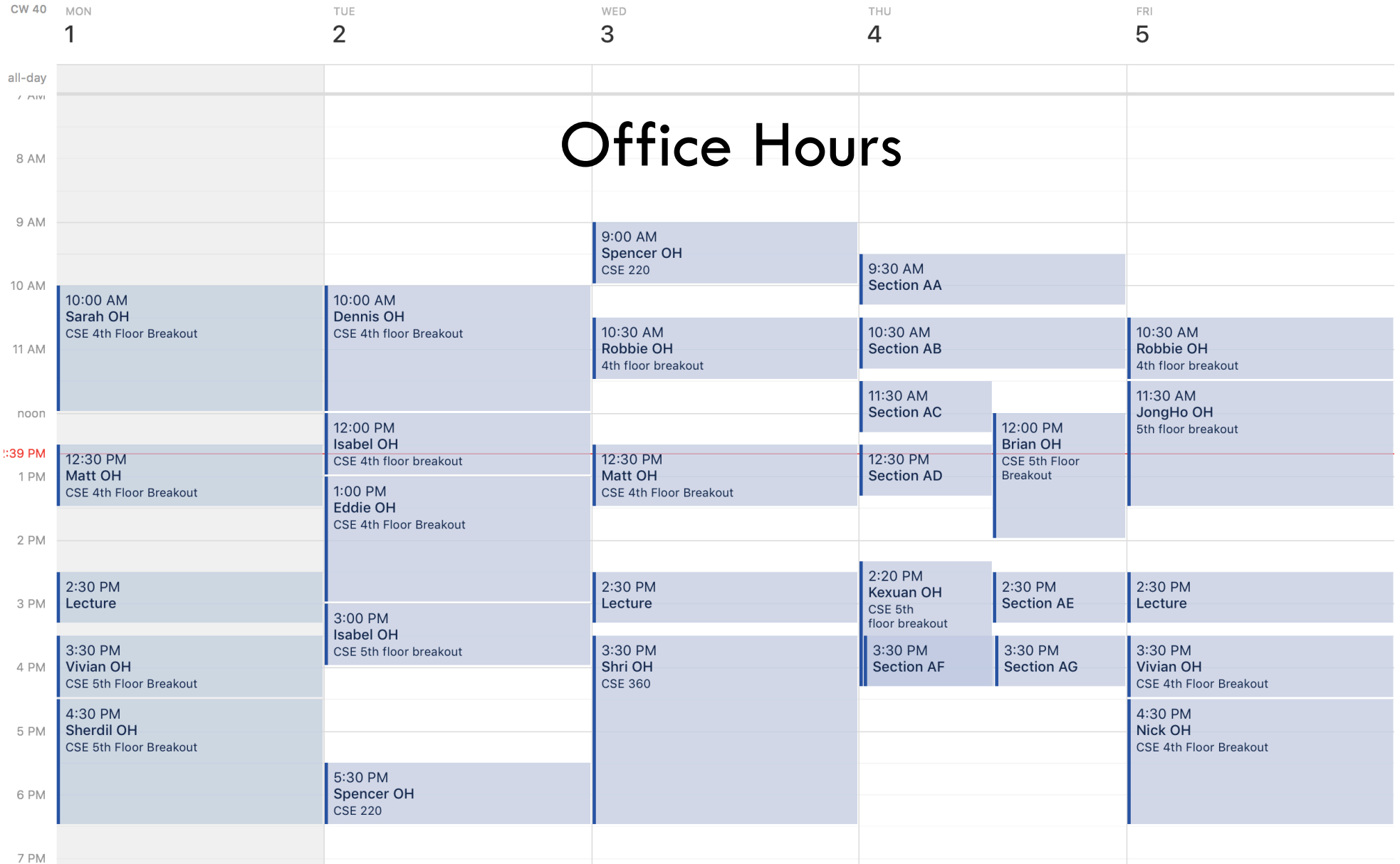
```
// Describes features common to all
// shapes.
public interface Shape {
    public double area();
    public double perimeter();
}
```



Announcements

Class webpage is live: https://courses.cs.washington.edu/courses/cse373/18au/

TA Introductions



Today’s Goals

- Framework to think and reason about data structure designs
- Revisit Big-Oh
- Analyze List implementation with Array and LinkedList
- Implementing Stack with Array and LinkedList

Design Decisions

For every ADT there are lots of different ways to implement them

Example: List can be implemented with an Array or a LinkedList

Based on your situation you should consider:

- Memory vs Speed
- Generic/Reusability vs Specific/Specialized
- One Function vs Another
- Robustness vs Performance

This class is all about implementing ADTs based on making the right design tradeoffs!

> A common topic in interview questions

Review: "Big Oh"

efficiency: measure of computing resources used by code.
- can be relative to speed (time), memory (space), etc.
- most commonly refers to run time

Assume the following:
- Any single Java statement takes same amount of time to run.
- A method call's runtime is measured by the total of the statements inside the method's body.
- A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.

We measure runtime in proportion to the input data size, N.
- **growth rate**: Change in runtime as N gets bigger. How does this algorithm perform with larger and larger sets of data?

Runs $2\mathbf{N^2 + N + 1}$ statements.
- We ignore constants like 2 because they are tiny next to N.
- The highest-order term (N^2) dominates the overall runtime.
- We say that this algorithm runs "on the order of" N^2.
- or $\mathbf{O(N^2)}$ for short ("**Big-Oh** of N cubed")

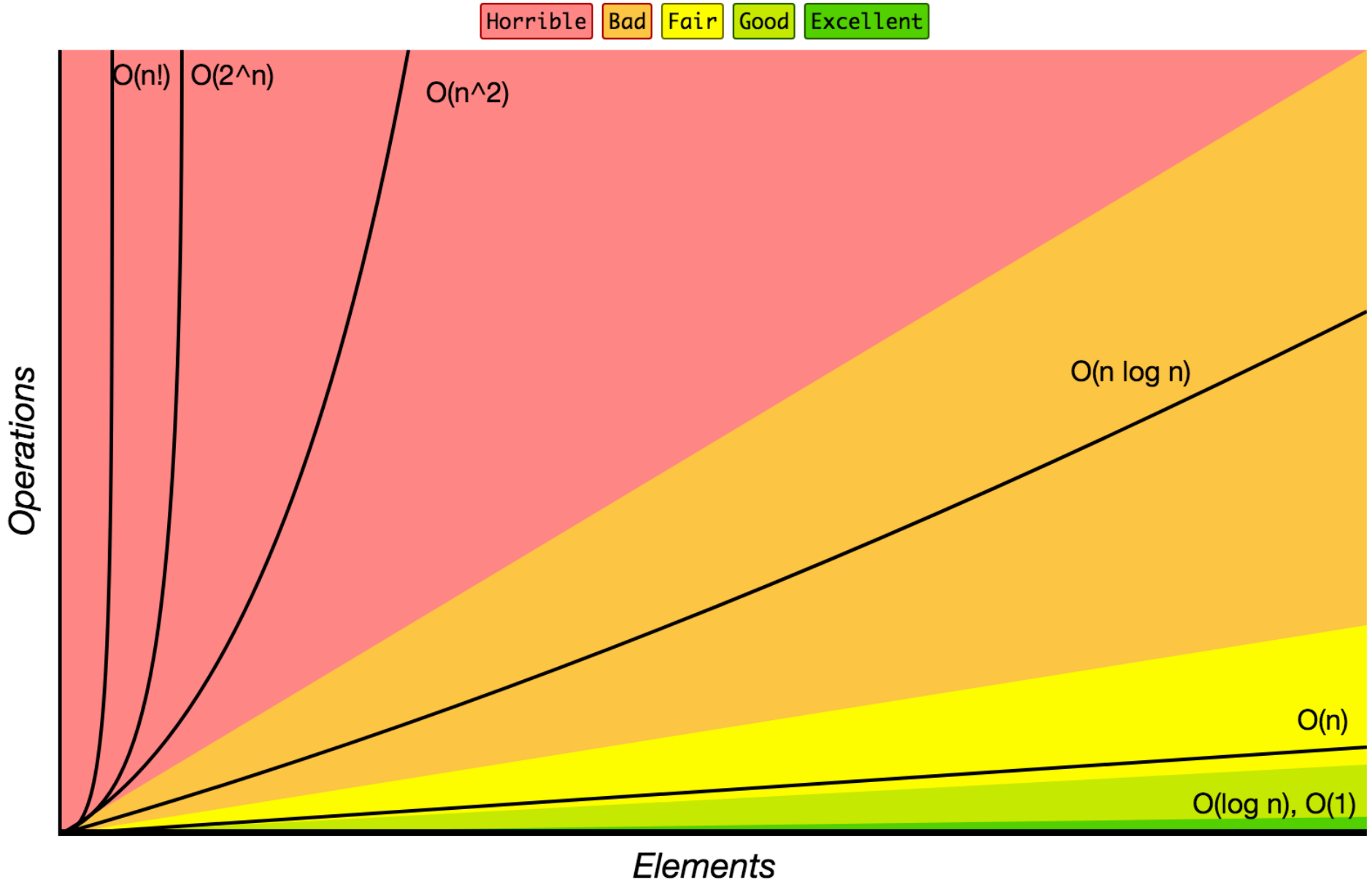
```
b = c + 10;

for (int i = 0; i < N; i++) {
    for (int j = 0; j < N ; j++) {
        array2[j][i] = array1[i][j];
        array1[i][j] = 0;
    }
}
for (int i = 0; i < N; i++) {
    array[i] = b;
}
```

Review: Complexity Class

complexity class: A category of algorithm efficiency based on the algorithm's relationship to the input size N.

Class	Big-Oh	If you double N, ...	Example
constant	O(1)	unchanged	Accessing an index of an array
logarithmic	$O(\log_2 N)$	increases slightly	Binary search
linear	O(N)	doubles	Looping over an array
log-linear	$O(N \log_2 N)$	slightly more than doubles	Merge sort algorithm
quadratic	$O(N^2)$	quadruples	Nested loops!
...	...	...	...
Exponential	$O(2^N)$	multiplies drastically	Fibonacci with recursion

Big-O Complexity Growth Chart



Review: Case Study: The List ADT

list: stores an ordered sequence of information.

- Each item is accessible by an index.
- Lists have a variable size as items can be added and removed

Supported Operations:

- **get(index):** returns the item at the given index
- **set(value, index):** sets the item at the given index to the given value
- **append(value):** adds the given item to the end of the list
- **insert(value, index):** insert the given item at the given index maintaining order
- **delete(index):** removes the item at the given index maintaining order
- **size():** returns the number of elements in the list

List ADT tradeoffs

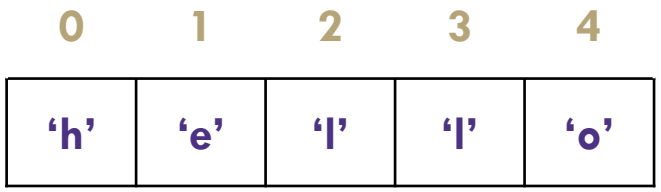
Time needed to access i-th element:
- Array: O(1) constant time
- LinkedList: O(n) linear time

Time needed to insert at i-th element
- Array: O(n) linear time
- LinkedList: O(n) linear time

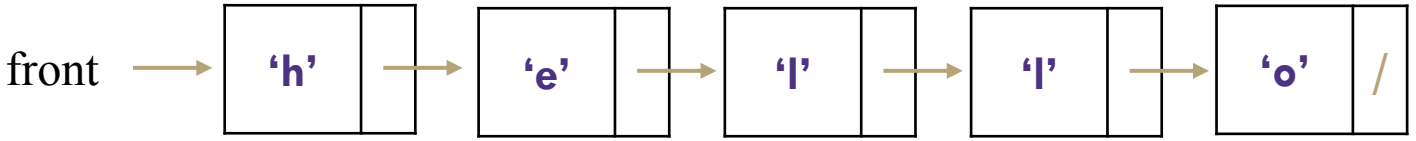
Amount of space used overall
- Array: sometimes wasted space
- LinkedList: compact

Amount of space used per element
- Array: minimal
- LinkedList: tiny extra

```
char[] myArr = new char[5]
```



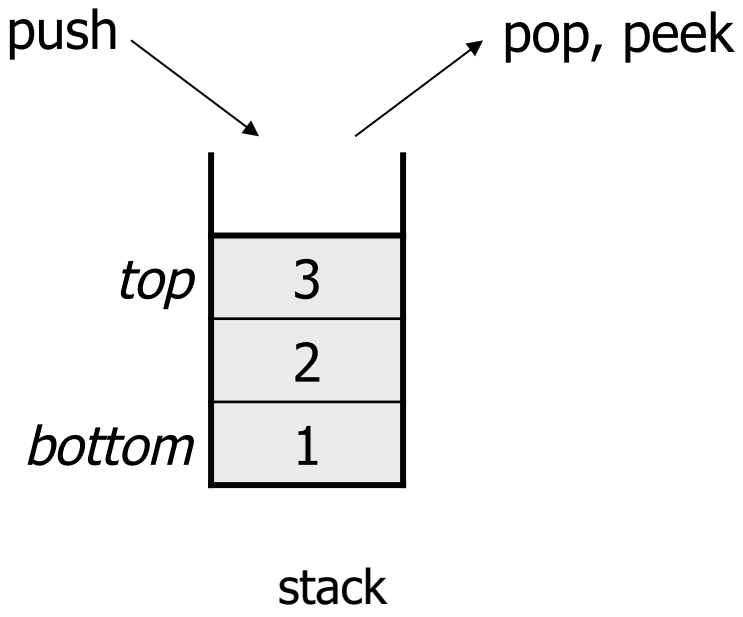

```
LinkedList<Character> myLl = new LinkedList<Character>();
```



Review: What is a Stack?

stack: A collection based on the principle of adding elements and retrieving them in the opposite order.

- Last-In, First-Out ("LIFO")
- Elements are stored in order of insertion.
  - We do not think of them as having indexes.
- Client can only add/remove/examine the last element added (the "top").



basic stack operations:

- **push(item)**: Add an element to the top of stack
- **pop()**: Remove the top element and returns it
- **peek()**: Examine the top element without removing it
- **size():** how many items are in the stack?
- **isEmpty():** true if there are 1 or more items in stack, false otherwise

Thought Experiment

Discuss with your neighbors: How would you implement the List ADT for each of the following situations? For each consider the most important functions to optimize.

Situation #1: Write a data structure that implements the List ADT that will be used to store a list of songs in a playlist.

LinkedList

Situation #2: Write a data structure that implements the List ADT that will be used to store the count of students who attend class each day of lecture.

ArrayList

Situation #3: Write a data structure that implements the List ADT that will be used to store the set of operations a user does on a document so another developer can implement the undo function.

Stack