Please fill out the Poll at- pollev.com/21sp373

Lecture 2: Abstract Data Types

CSE 373: Data Structures and Algorithms

Agenda

- Dust off data structure cobwebs
- Meet the ADT
- List Case Study

Announcements

Things are live!

- course website – one stop for all things 373
- Discord – connect with students + office hours
- Ed board – get your course content questions answered
- Gradescope

Proj 0 Assigned – Due Wednesday April 7th

- 143 review
- solo assignment

Find a partner in time for Proj 1 next Wednesday

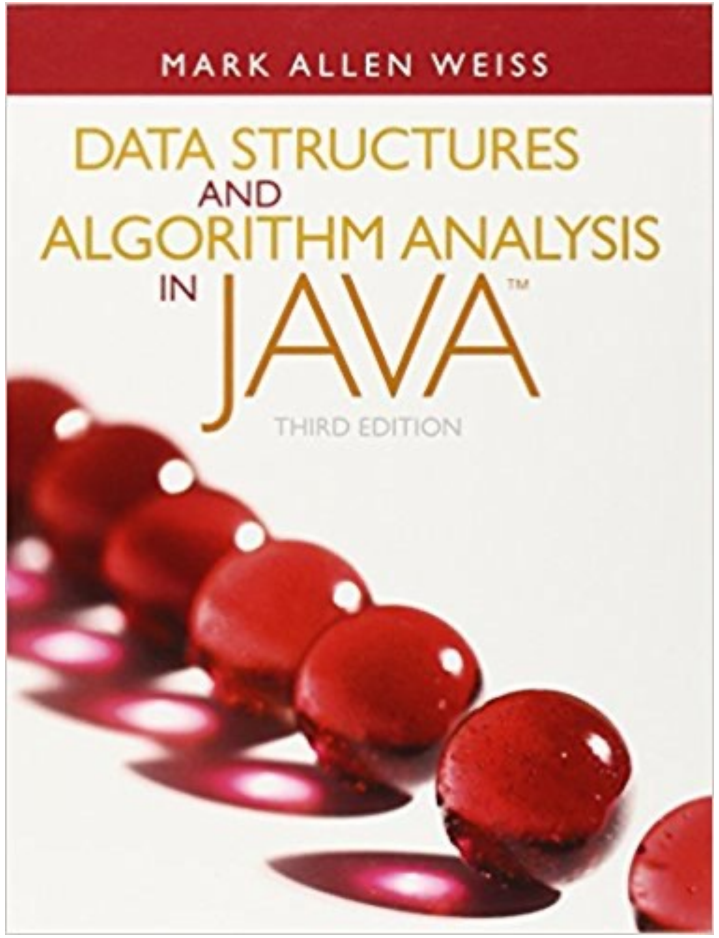
Textbook

Data Structures and Algorithm Analysis in Java
by Mark Allen Weiss

Completely **optional**

- Nothing assigned out of the textbook
- No readings

Advice: only purchase if you learn best with a textbook, otherwise not recommended



Questions?

Clarification on syllabus, General complaining/moaning

What is this class about?

CSE 143 – OBJECT ORIENTED PROGRAMMING

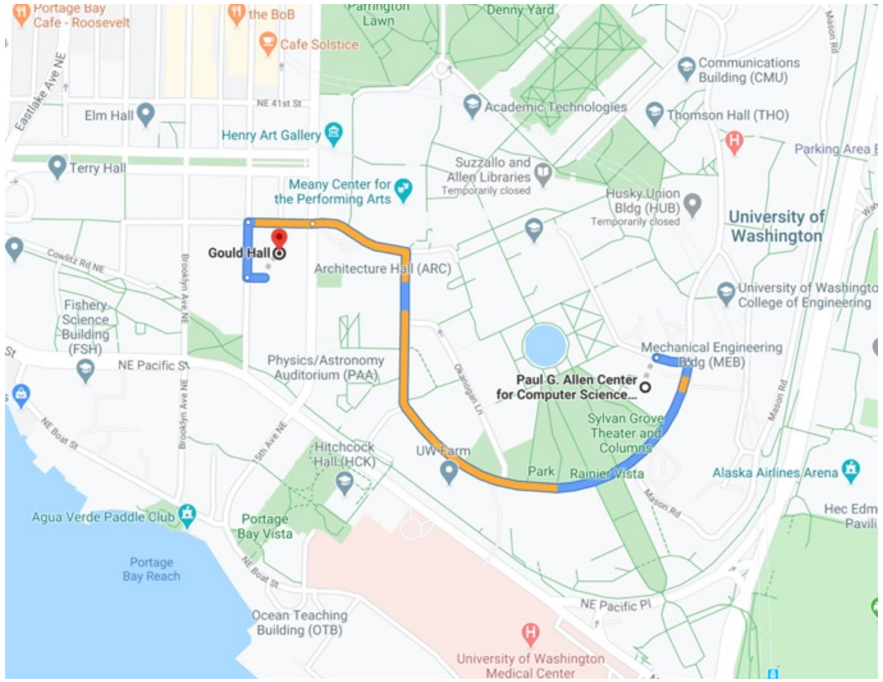
- Classes and Interfaces
- Methods, variables and conditionals
- Loops and recursion
- Linked lists and binary trees
- Sorting and Searching
- O(n) analysis
- Generics

CSE 373 – DATA STRUCTURES AND ALGORITHMS

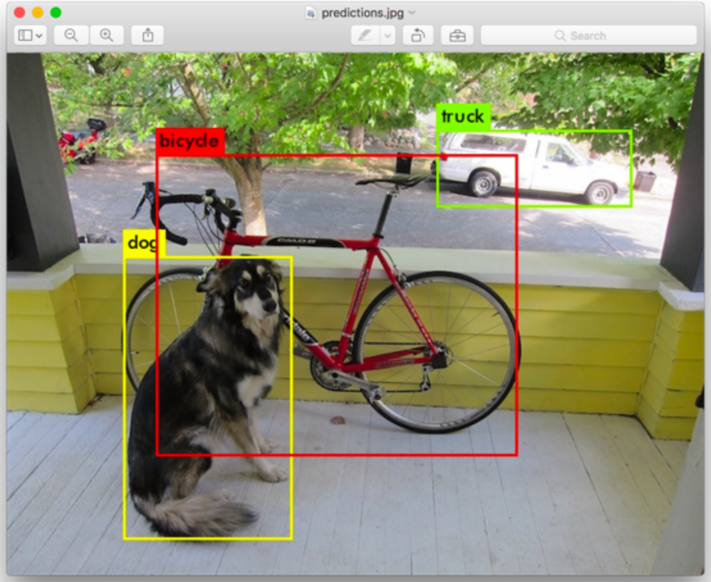
- Design decisions
- Design analysis
- Implementations of data structures
- Debugging and testing
- Abstract Data Types
- Code Modeling
- Complexity Analysis
- Software Engineering Practices

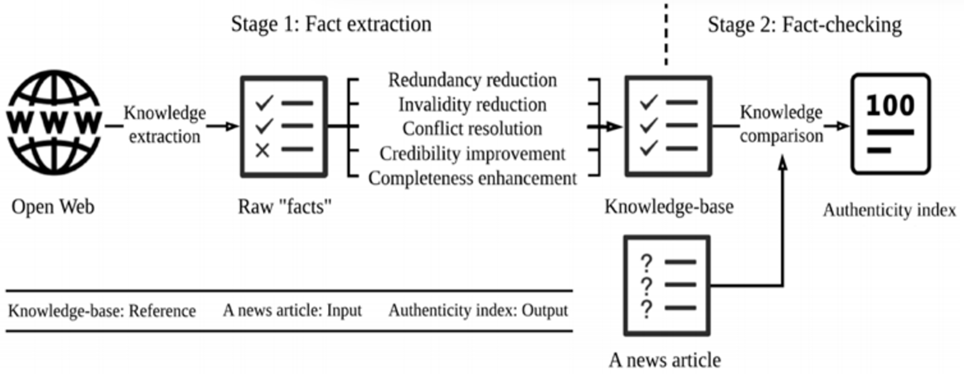
Why 373?

1. Build a strong foundation of data structures and algorithms that will let you tackle the biggest problems in computing



373 Data Structures & Algorithms





Fake News: A Survey of Research, Detection Methods, and Opportunities (Xinyi Zhou, Reza Zafarani/arXiv:1812.00315)

Why 373?

2. Pick up the vocabulary, skills, and practice needed to make **design decisions**. Learn to **evaluate** the tools in your CS toolbox

- Differences between technical implementations
- Evaluation can mean many different things!

Data Structures and Algorithms

What are they anyway?

Basic Definitions

Data Structure

- A way of organizing and storing data
- Examples from CSE 14X: arrays, linked lists, stacks, queues, trees

Algorithm

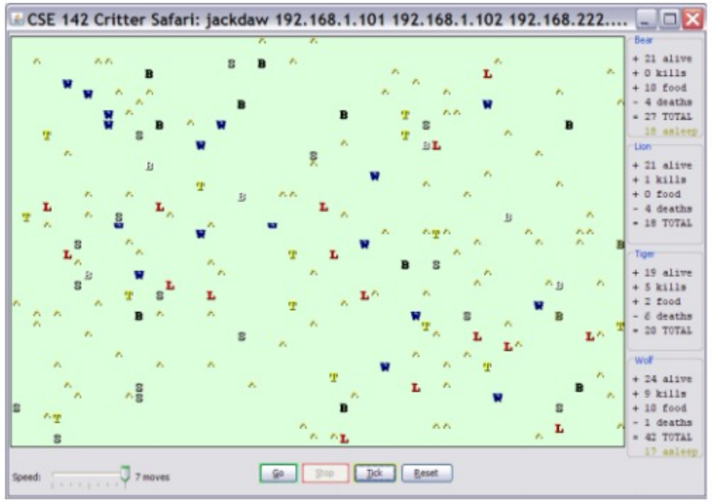
- A series of precise instructions to produce to a specific outcome
- Examples from CSE 14X: binary search, merge sort, recursive backtracking

Review: Clients vs Objects

CLIENT CLASSES

A class that is executable, in Java this means it contains a Main method

```
public static void main(String[] args)
```



OBJECT CLASSES

A coded structure that contains data and behavior

Start with the data you want to hold, organize the things you want to enable users to do with that data

1. Ant

constructor	`public Ant(boolean walkSouth)`
color	red
eating behavior	always returns `true`
fighting behavior	always scratch
movement	if the Ant was constructed with a `walkSouth` value of `true`, then alternates between south and east in a zigzag (S, E, S, E, ...); otherwise, if the Ant was constructed with a `walkSouth` value of `false`, then alternates between north and east in a zigzag (N, E, N, E, ...)
toString	`"%"` (percent)

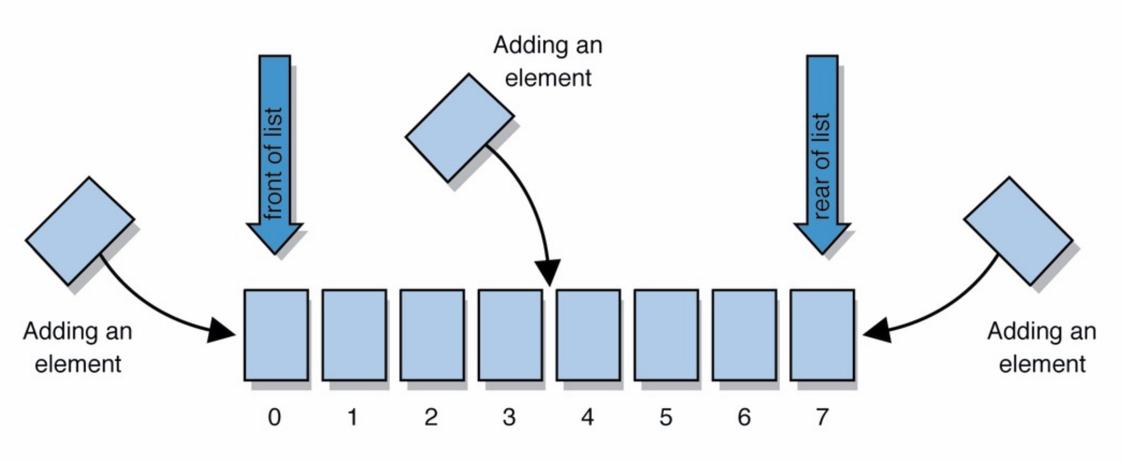
Abstract Data Types (ADT)

Abstract Data Types

- An abstract definition for expected operations and behavior
- Defines the input and outputs, not the implementations

Review: List - a collection storing an ordered sequence of elements

- each element is accessible by a 0-based index
- a list has a size (number of elements that have been added)
- elements can be added to the front, back, or elsewhere
- in Java, a list can be represented as an ArrayList object



Review: Interfaces

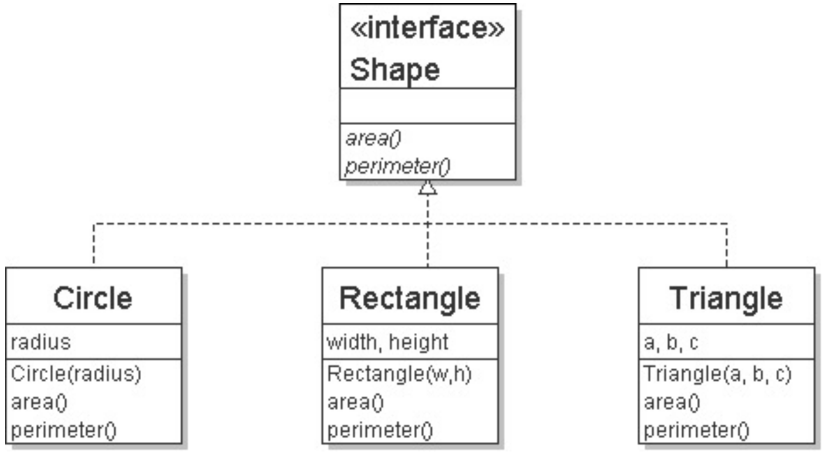
interface: A construct in Java that defines a set of methods that a class promises to implement

- 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();
}
```



Review: Java Collections

Java provides some implementations of ADTs for you!

ADTs	Data Structures
Lists	`List<Integer> a = new ArrayList<Integer>();`
Stacks	`Stack<Character> c = new Stack<Character>();`
Queues	`Queue<String> b = new LinkedList<String>();`
Maps	`Map<String, String> d = new TreeMap<String, String>();`

But some data structures you made from scratch… why?

Linked Lists - LinkedIntList was a collection of ListNode

Binary Search Trees – SearchTree was a collection of SearchTreeNodes

Full Definitions

Abstract Data Type (ADT)

- *A definition for expected operations and behavior*
- A mathematical description of a collection with a set of supported operations and how they should behave when called upon
- Describes what a collection does, not how it does it
- Can be expressed as an interface
- Examples: List, Map, Set

Data Structure

- *A way of organizing and storing related data points*
- An object that implements the functionality of a specified ADT
- Describes exactly how the collection will perform the required operations
- Examples: LinkedIntList, ArrayIntList

ADTs we'll discuss this quarter

- List
- Set
- Map
- Stack
- Queue
- Priority Queue
- Graph
- Disjoint Set

Learning to Bake in a CSE Class

Think of what you'll learn this quarter as a cookbook

- ADTs are the chapters/category: Soups, Salads, Cookies, Cakes, etc
  - High-level descriptions of a category of functionality
  - You don't serve a soup when guests expect a cookie!
- Data structures are the recipes: chocolate chip cookies, snickerdoodles, etc
  - Step-by-step, concrete descriptions of an item with specific characteristics
  - Understand your tradeoffs before replacing carrot cake with a wedding cake

When you go out into the world …

- Figure out which category is required
- Choose the specific recipe that best fit the situation

Case Study: The List ADT

list: a collection storing an ordered sequence of elements.

- Each item is accessible by an index.
- A list has a size defined as the number of elements in the list

```
List<String> names = new ArrayList<>();
names.add("Anish");
names.add("Amanda");
names.add(0, "Brian");
```

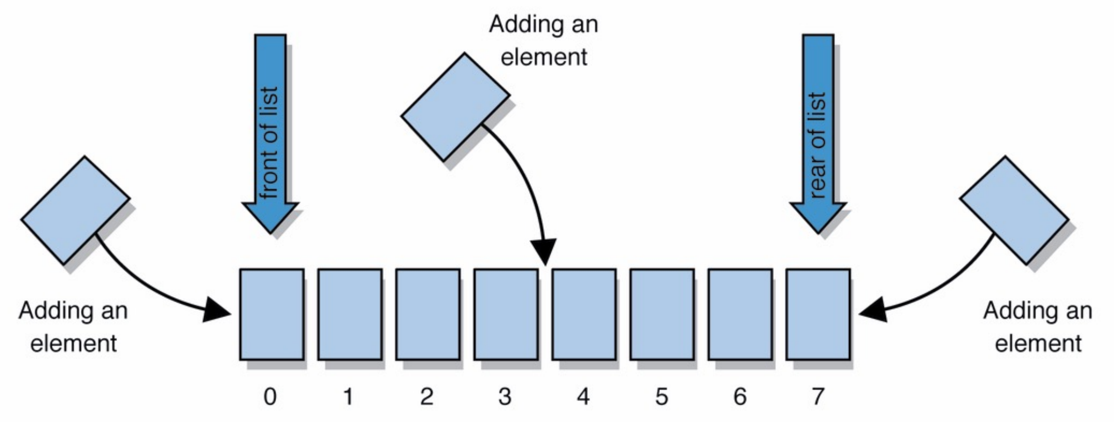
Case Study: The List ADT

list: a collection storing an ordered sequence of elements.

- Each item is accessible by an index.
- A list has a size defined as the number of elements in the list

Expected Behavior:

- **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

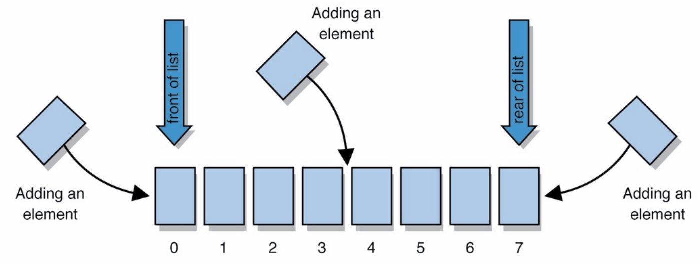


Case Study: List Implementations

List ADT

state
Set of ordered items
Count of items

behavior
get(index) return item at index
set(item, index) replace item at index
append(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items



ArrayList
uses an Array as underlying storage

ArrayList<E>

state
```
data[]
size
```
behavior
```
get return data[index]
set data[index] = value
append data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size
```

0	1	2	3	4
88.6	26.1	94.4	0	0
list			free space	

LinkedList
uses nodes as underlying storage

LinkedList<E>

state
```
Node front
size
```
behavior
```
get loop until index, return node's value
set loop until index, update node's value
append create new node, update next of last node
insert create new node, loop until index, update next fields
delete loop until index, skip node
size return size
```

Implementing ArrayList

ArrayList<E>

state
data[]
size

behavior
get return data[index]
set data[index] = value
append data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size

`insert(element, index)` with shifting

`insert(10, 0)`

0	1	2	3
10	4	5	

`numberOfItems =` 4

`delete(index)` with shifting

`delete(0)`

0	1	2	3
10	3	4	5

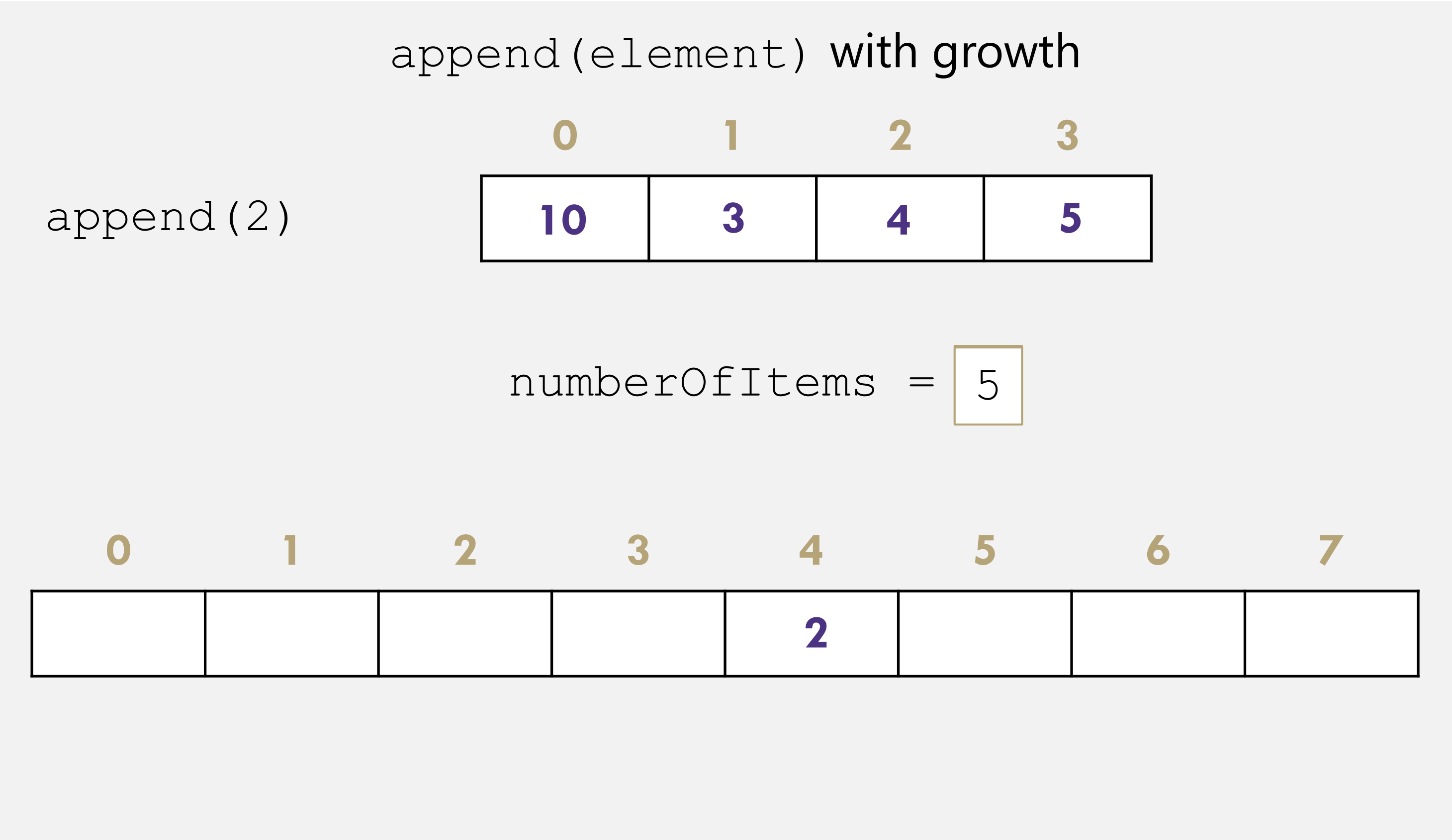
`numberOfItems =` 3

Implementing ArrayList

ArrayList<E>

state
data[]
size

behavior
get return data[index]
set data[index] = value
append data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size



Design Decisions

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

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