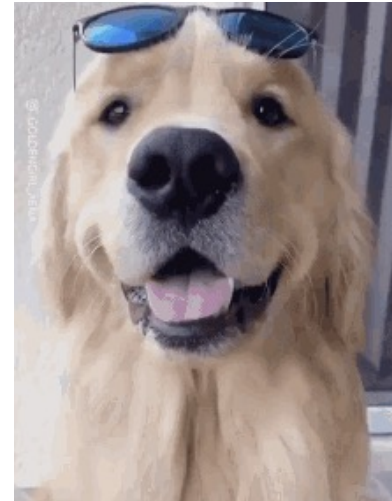


Grab a handout  
from the back!

# CSE 332: Data Structures & Parallelism

## Lecture 1: Intro, Stacks & Queues

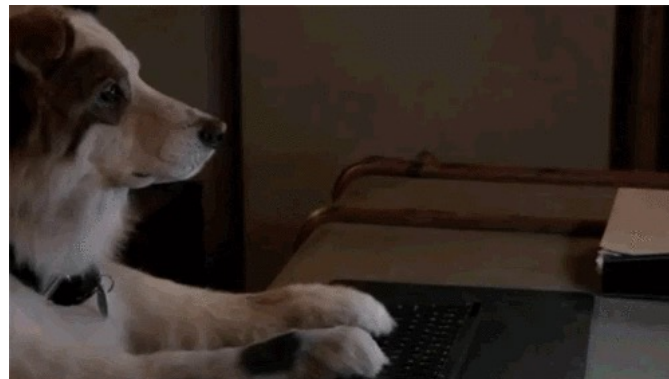
Arthur Liu  
Summer 2022



# Welcome!

We have 9 weeks to learn *fundamental data structures and algorithms for organizing and processing information*

- “Classic” data structures / algorithms and how to analyze rigorously their efficiency and when to use them
- Queues, dictionaries, graphs, sorting, etc.
- Parallelism and concurrency (!)



# Today

- **Introductions**
- Administrative Info
- What is this course about?
- Review: Queues and stacks

# CSE 332 Course Staff

## Instructor:

Arthur Liu

## Teaching Assistants:

- Nathan Akkaraphab
- Hans Easton
- Winston Jodjana
- Neel Jog
- Dara Stotland
- Thien Kim Tran

# Me

- BS/MS graduate
- 9+ quarters TA
- Previously: Amazon, Microsoft, Startup
- Meta
- Soccer, Triathlon



# Today

- Introductions
- **Administrative Info**
- What is this course about?
- Review: Queues and stacks

# Course Information

- **Instructor:** Arthur Liu, CSE210
  - Office Hours: see course web page, and by appointment
  - [artliu@cs.washington.edu](mailto:artliu@cs.washington.edu)
- **Course Web Page:**
  - [cs.uw.edu/332](http://cs.uw.edu/332)
- **Text (optional):**  
*Data Structures & Algorithm Analysis in Java*, (Mark Allen Weiss), 3rd edition, 2012  
(2<sup>nd</sup> edition also o.k.)

# Communication

- Course email lists:  
`cse332a_su22@uw`
  - You are already subscribed
- Ed Discussion board
  - Your first stop for questions about course content & assignments
  - Ed Announcement Emails - You must get and read announcements sent there
- Anonymous feedback link
  - For good and bad: if you don't tell us, we won't know!



# Course Meetings

- Lecture
  - Take notes, materials posted (sometimes afterwards)
  - Ask questions, focus on key ideas (rarely coding details)
  - Attend synchronously as much as possible and interact with peers!
- Section
  - Practice problems!
  - Answer Java/project/homework questions, etc.
  - Occasionally may introduce new material
  - An important part of the course (not optional)
- Office hours
  - Use them: *please visit us!*

Tries

# Course Materials

- Lecture and section materials will be posted
  - They are visual aids, so not always a complete description!
  - If you have to miss, find out what you missed
- Textbook: Weiss 3<sup>rd</sup> Edition in Java
  - Good read, but only responsible for lecture/section/hw topics
  - 3<sup>rd</sup> edition improves on 2<sup>nd</sup>, but 2<sup>nd</sup> is fine
- Parallelism / concurrency topics in separate free resource designed for 332

# Course Work

- ~15 weekly individual homework exercises (25%)
- Programming Projects (37%)
  - Use Java and IntelliJ, Gitlab
  - Done in partners\*, o.k. if partner is in other quiz section

\*Can do individually, but projects designed for partners

- Midterm (15%) (Week 5 Monday 7/18)
- Final (20%) (last Thursday section 8/18 and Friday of quarter 8/19)
- Course-Wide Participation (3%)
  - Many ways to earn credit here, relatively lenient on this

# Late Policy and Student Conduct

- Late Policy
  - Exercises: No late submissions allowed
  - Projects: 4 late day tokens for the entire quarter, max 2 per project
- Academic Conduct (see syllabus)
  - In short: don't attempt to gain credit for something you didn't do and don't help others to do so either
  - This does *not* mean suffer in silence!
    - Learn from course staff and peers, talk, share ideas; *but* don't share or copy work that is supposed to be yours
- Extenuating Circumstances

## Who is your favorite superhero?

- Make sure you are logged in with your uw netid account
- If you might have an issue with in-lecture polling, reach out to the instructor as soon as possible
- Only need to vote on 60% of questions to get full credit!
  - 1.05 points for correct answer, 1 point otherwise
- If you are sick, stay home!

# Homework for Today!!

1. **Project #1:** Fill out partner request survey by **Thursday 6pm**
  - Partner Mixer Thursday 12-1 @ CSE2 271
2. Preliminary Survey: fill out by **Friday night**
3. **Exercise #1: Due SUNDAY at 11:59pm**
4. **Make sure you are on Ed**
5. **Review Java & install IntelliJ**
6. **Reading (optional) in Weiss (see website)**

# Reading

- Reading in Weiss
- For this week:
  - Weiss 3.1-3.7 – Lists, Stacks & Queues (Topic for Project #1)
  - (Friday) Weiss 2.1-2.4 – Algorithm Analysis
  - (Useful) Weiss 1.1-1.6 – Mathematics and Java (NOT covered in lecture, will use some of these baseline facts)

# Today

- Introductions
- Administrative Info
- **What is this course about?**
- Review: Queues and stacks



# Data Structures & Parallelism

- About 70% of the course is a “classic data-structures course”
  - Timeless, essential stuff
  - Core data structures and algorithms that underlie most software
  - How to analyze algorithms

→ • **Implement them**

- Plus a serious first treatment of programming with *multiple threads*
  - For *parallelism*: Use multiple processors to finish sooner
  - For *concurrency*: Correct access to shared resources
  - Will make many connections to the classic material

# What 332 is about

- Deeply understand the basic structures used in software
  - Understand the data structures and their trade-offs
  - Rigorously analyze the algorithms that use them
  - Learn how to pick “the right thing for the job”
- Experience the purposes and headaches of multithreading
- Practice design, analysis, and implementation
  - The elegant interplay of “theory” and “engineering” at the core of computer science

# Goals

- You will understand:
  - What the tools are for storing and processing common data types
  - Which tools are appropriate for which need (read: tradeoffs)
- So that you will be able to:
  - Make good design choices
  - Justify and communicate your design decisions

# One view on this course

- This is the class where you being to think like a computer scientist
  - You stop thinking in Java code
  - You start thinking that this is a hashtable problem, a stack problem, etc.
  - Feel more comfortable not having “right” answers

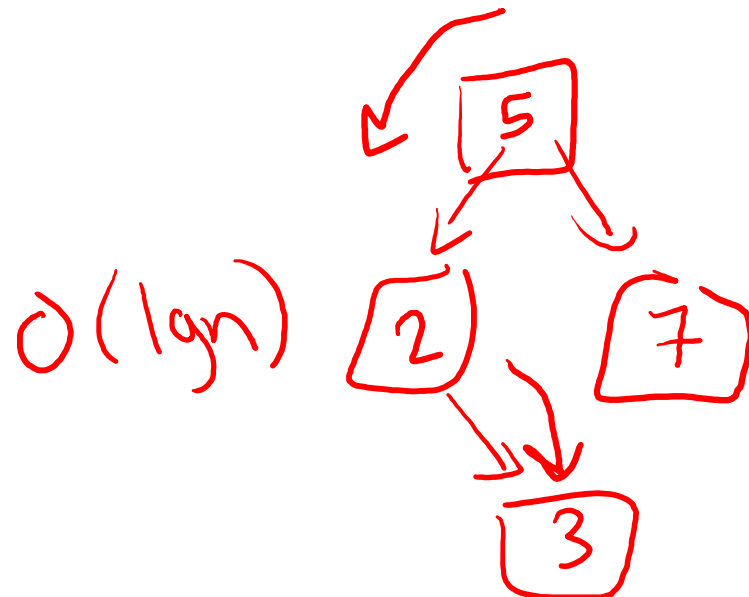
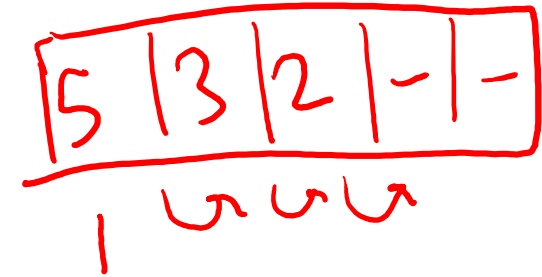
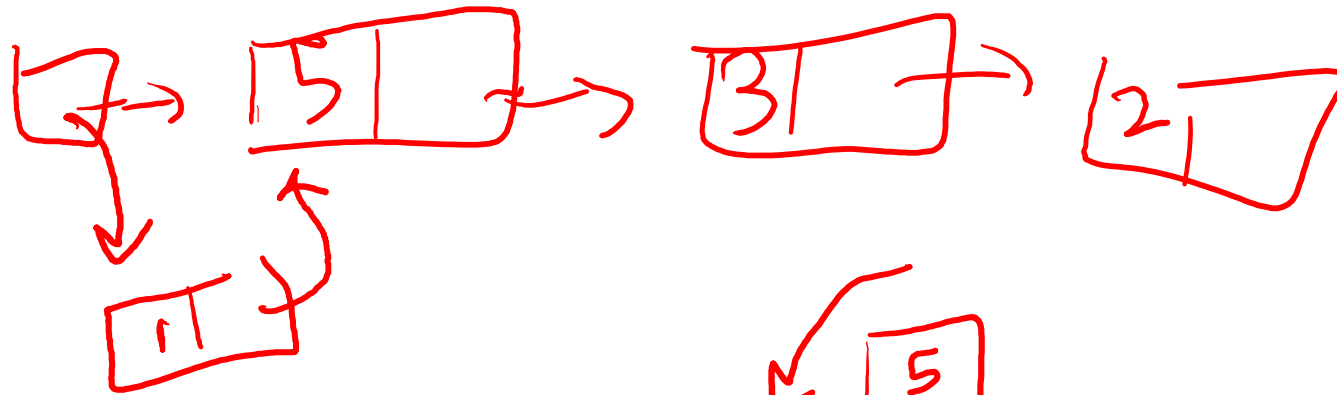


# Data Structures?

- “Clever” ways to organize information in order to enable *efficient* computation over that information

# Example Datastructures and Their Trade-Offs

- LinkedList, ArrayList



# Trade-Offs

- A data structure strives to provide many useful, efficient operations
- But there are unavoidable trade-offs:
  - Time vs. Space
  - One operation more efficient if another less efficient
  - Generality vs. Simplicity vs. Performance
- That is why there are many data structures; educated CSEers internalize their main trade-offs and techniques
  - Recognize and reason about logarithmic < linear < quadratic < exponential

More abstract

# Getting Serious: Terminology

- Abstract Data Type (ADT)

List

- Mathematical description of a "thing" with set of operations on that "thing"

- Data Structures

ArrayList, LL

- A specific organization of data and family of algorithms for implementing an ADT

resize

- Algorithm

- A high level, language-independent description of a step-by-step process

- Implementation of a data structure

- The actual code implementation in a specific language

more concrete

More

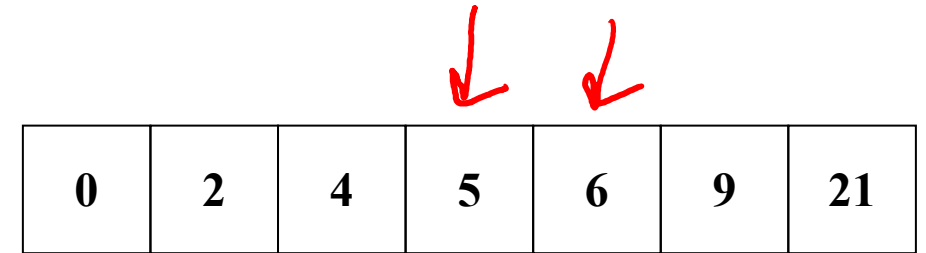


# Getting Serious: Terminology

- **Abstract Data Type (ADT)** **List**
  - Mathematical description of a "thing" with set of operations on that "thing"
- **Data Structures** **ArrayList, LinkedList**
  - A specific organization of data and family of algorithms for implementing an ADT
- **Implementation** of a data structure
  - The actual code implementation in a specific language **Your CSE143 code, Java Util Library**
- **Algorithm** **How to remove node, How to resize**
  - A high level, language-independent description of a step-by-step process

# Getting Serious: Terminology

- **Abstract Data Type (ADT)**
  - Mathematical description of a "thing" with set of operations on that "thing"
- **Data Structures**
  - A specific organization of data and family of algorithms for implementing an ADT
- **Implementation** of a data structure
  - The actual code implementation in a specific language



- **Algorithm**
  - A high level, language-independent description of a step-by-step process

# Today

- Introductions
- Administrative Info
- What is this course about?
- **Review: Queues and stacks**

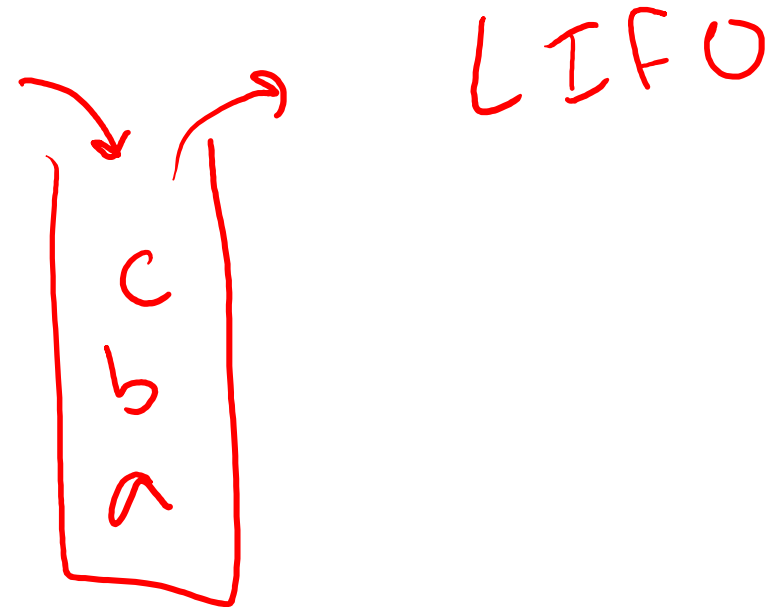
# Terminology Example: Stack and Queue ADT

Stack ADT
<p>State:</p> <ul style="list-style-type: none"><li>• Set of elements</li></ul> <p>Operations:</p> <ul style="list-style-type: none"><li>• <b>push(element)</b></li><li>• <b>pop()</b> – returns the most recent element that was added to the stack</li></ul>

Queue ADT
<p>State:</p> <ul style="list-style-type: none"><li>• Set of elements</li></ul> <p>Operations:</p> <ul style="list-style-type: none"><li>• <b>enqueue(element)</b></li><li>• <b>dequeue()</b> – deletes and returns the element that has been in the queue the longest</li></ul>

# Terminology Example: Stack and Queue ADT

Stack ADT
<p>State:</p> <ul style="list-style-type: none"><li>• Set of elements</li></ul> <p>Operations:</p> <ul style="list-style-type: none"><li>• <b>push(element)</b></li><li>• <b>pop()</b> – returns the most recent element that was added to the stack</li></ul>



# Why useful

The **Stack ADT** is a useful abstraction because:

- It arises all the time in programming (see Weiss for more)
  - Recursive function calls
  - Balancing symbols (parentheses) ( ) [ ]
  - Evaluating postfix notation:  $3\ 4\ +\ 5\ *$
  - Clever: Infix  $((3 + 4) * 5)$  to postfix conversion (see Weiss)
- We can code up a **reusable library**
- We can **communicate** in high-level terms
  - “Use a stack and push numbers, popping for operators...”
  - Rather than, “create a linked list and add a node when...”

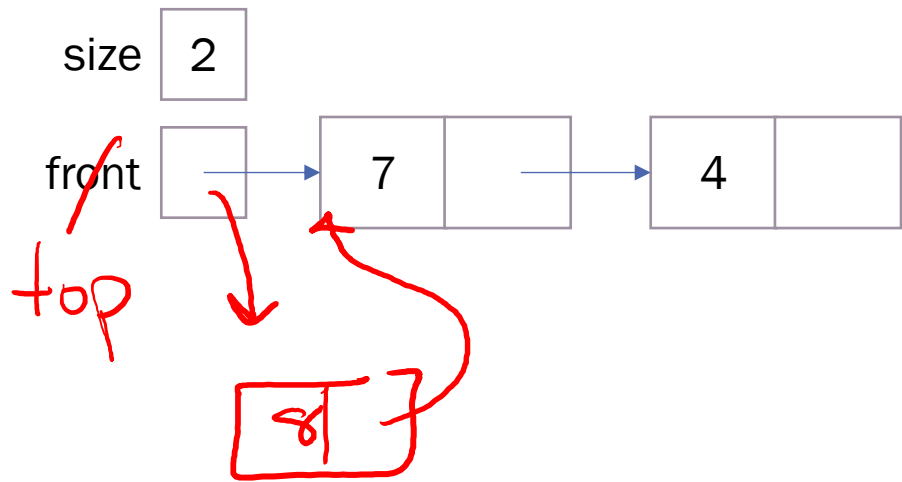
# Balancing Parenthesis

{([])}

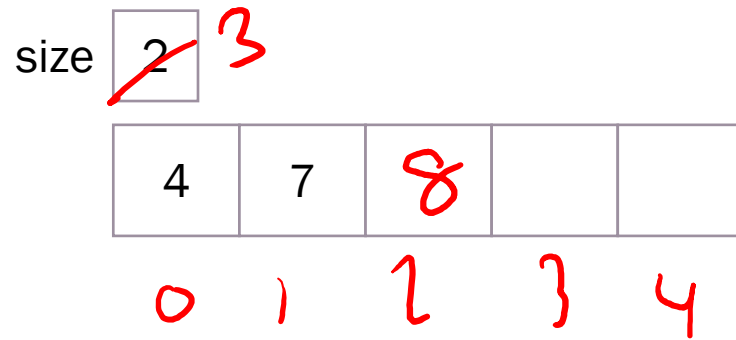
((()))

# Stack Datastructures

- Singly Linked List Implementation



- Array Implementation





# Queue Datastructures



← a b c →

FIFO

## Queue ADT

State:

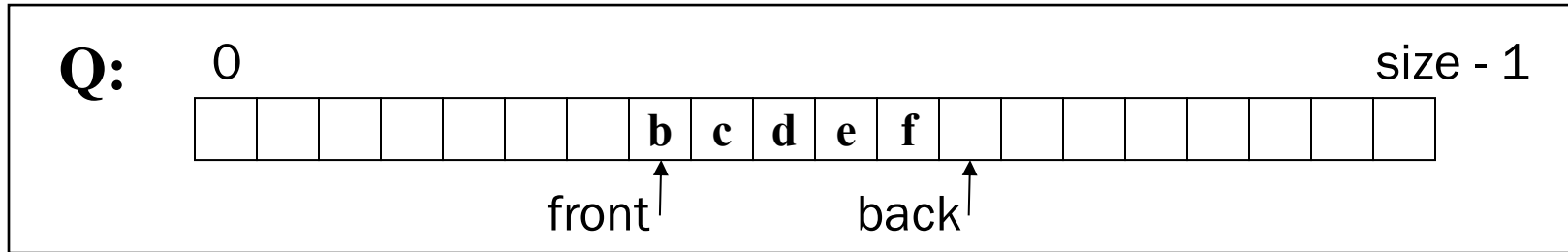
- Set of elements

Operations:

- **enqueue(element)**
- **dequeue()** – deletes and returns the element that has been in the queue the longest



# Circular Array Queue Data Structure



$O(1)$  [

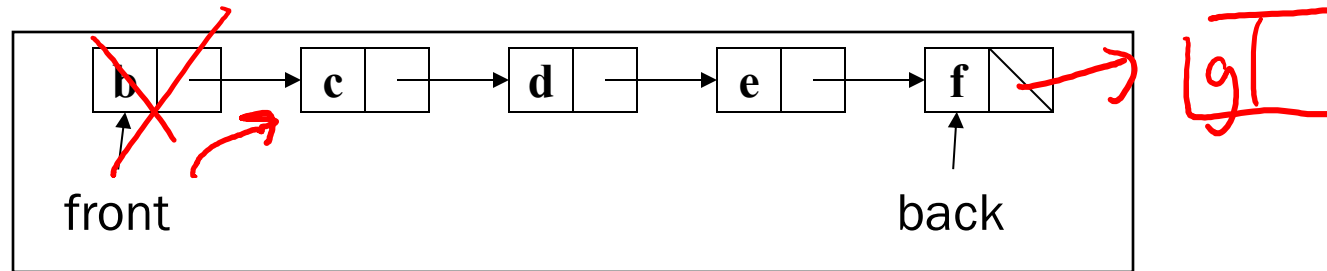
```
// Basic idea only!  
enqueue(x) {  
    Q[back] = x;  
    back = (back + 1) % size  
}
```

$O(1)$  [

```
// Basic idea only!  
dequeue() {  
    x = Q[front];  
    front = (front + 1) % size;  
    return x;  
}
```

- What if *queue* is empty?
  - Enqueue?
  - Dequeue?
- What if *array* is full? → doubling 2x
- How to test for empty? ←
- What is the *complexity* of the operations?

# Linked List Queue Data Structure



$O(1)$  [

```
// Basic idea only!  
enqueue(x) {  
    back.next = new Node(x);  
    back = back.next;  
}
```

$O(1)$  [

```
// Basic idea only!  
dequeue() {  
    x = front.item;  
    front = front.next;  
    return x;  
}
```

- What if *queue* is empty?
  - Enqueue?
  - Dequeue?
- Can *list* be full?
- How to *test* for empty?
- What is the *complexity* of the operations?

# What are advantages and disadvantages of using CircularArray vs. LinkedList datastructure to implement the Queue ADT?

Array:



- More efficient per element
- ease of implementing
- fast

Linked List:



- always just enough space
- fast

# Circular Array vs. Linked List

## Array:

- May waste unneeded space or run out of space
- Space per element is excellent
- Operations very simple / fast

## Operations not in Queue ADT, but also:

- Constant-time “access to kth element”
- For operation “insertAtPosition”, must shift all later elements

## Linked List:

- Always just enough space
- But more space per element
- Operations very simple / fast

## Operations not in Queue ADT, but also:

- No constant-time “access to kth element”
- For operation “insertAtPosition”, must traverse all earlier elements

# Homework for Today!!

1. **Project #1:** Fill out [partner request survey](#) by **Thursday 6pm**
  - Partner Mixer Thursday 12-1 @ CSE2 271
2. [Preliminary Survey](#): fill out by **Friday night**
3. **Exercise #1: Due SUNDAY at 11:59pm**
4. **Make sure you are on Ed**
5. **Review Java & install IntelliJ**
6. **Reading (optional) in Weiss (see website)**