Lecture 1: Intro, Stacks & Queues

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Welcome!

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

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

Today

- Introductions
- Administrative Info
- What is this course about?
- Lecture 01
  - ADTs, Data Structures (and Algorithm), and Implementation
  - Review: Queues and stacks

CSE 332 23su Course Staff

Instructor:

Yafqa Khan

Teaching Assistants:

- Aaron Honjaya
- Hana Smahi
- Jacklyn Cui
- Samarth Venkatesh

Me (Yafqa Khan)

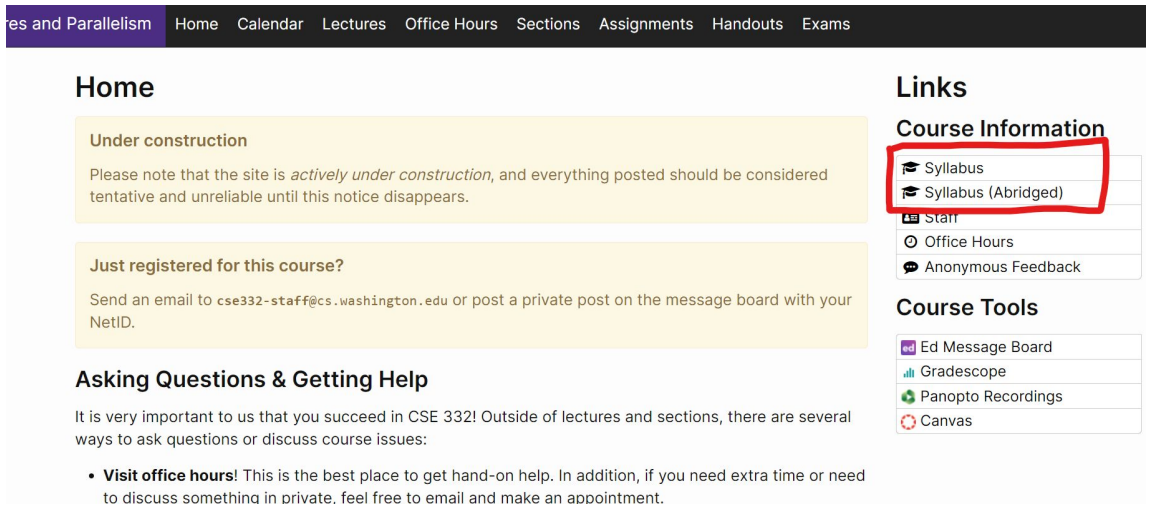
- BS/MS Graduate
- Too many quarters of TA’ing (14?)
- Previously: Amazon
- Hobbies: Reading, Anime/Manga, Finding Housing :_(

Today

- Introductions
- Administrative Info
- What is this course about?
- Lecture 01
  - ADTs, Data Structures (and Algorithm), and Implementation
  - Review: Queues and stacks

Course Information

- Instructor: Yafqa Khan, CSE206
  - Office Hours: see course web page and by appointment (email me)
  - yafqak@cs.washington.edu
- Course Web Page:
  - cs.uw.edu/332
- Syllabus:
  - On the website!



Communication (to you)

- Course email lists:
  - **cse332a_su25@uw.edu**
  - UW Email!
- Ed Discussion Board (to you)
  - To you: Ed Announcement Emails

Communication (to us)

- Course staff lists:
  - **cse332-staff@cs.washington.edu**
  - CS Email! Yes, very confusing, weird UW + Allen School infrastructure
- Ed Discussion Board (to us)
  - To us: Ask questions!
- Anonymous Feedback Form (feedback.cs.washington.edu)
  - Nobody sees your name (including me)
  - CANNOT REPLY
  - Good + Bad Feedback, if you don't say anything 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!

Course Material

- Lecture and section materials will be posted
  - They are visual aids, so not always a complete description!
  - If you must miss, find out what you missed
- Textbook: Weiss 3rd Edition in Java
  - Good read, but only responsible for lecture/section/hw topics
  - 3rd edition improves on 2nd, but 2nd is fine
- Parallelism / concurrency topics in separate free resource designed for 332
  - https://homes.cs.washington.edu/~djg/teachingMaterials/spac/sophomoricParallelismAndConcurrency.pdf

Course Work

- 13 individual homework exercises (60%)
  - 5% each, lowest scoring one dropped
  - 2 late days each EXCEPT exercise 12
  - 6 late days TOTAL
- 2 in-person exams
  - 20% each, both non-cumulative
  - Check website + syllabus for info
  - Email me ASAP if cannot make it

Homework :(

Sorry, the class is really condensed

1. EX 0 out today (spec on website)
   - due next monday
2. Check you have access to Ed
   - Email me if there are problems
   - https://courses.cs.washington.edu/courses/cse332/25su/calendar/lecturelist.html

Homework :(

4. Reading (optional)
   - Weiss textbook - free rent at Ode!
   - For this week:
     - (Today) Weiss 3.1-3.7 – Lists, Stacks & Queues (Topic for Project #1)
     - (Friday) Weiss 2.1-2.4 – Algorithm Analysis

       Weiss 1.1-1.6 – Mathematics and Java
       (NOT covered in lecture, will use some of these baseline facts)

Any Questions?

Today

- Introductions
- Administrative Info
- What is this course about?
- Lecture 01
  - ADTs, Data Structures (and Algorithm), and Implementation
  - Review: Queues and stacks

Data Structures & Parallelism

- About 70% of the course is the "data structures & algorithms course"
  - **What** and **how** of data structures & algorithms
  - Pick the correct data structures & algorithms (analyze & tradeoffs)
  - **Implement them**
- + a serious first treatment of programming with ***multiple threads***
  - For *parallelism*: Use multiple processors to finish sooner
  - For *concurrency*: Correct access to shared resources

Really should be called

Data Structures & Algorithms and Parallelism & Concurrency 🙃

One view on this course

- This is the class where you begin 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 one **best, correct** answer
    - Make **good** design choices
    - Justify and communicate your design choices

Today

- Introductions
- Administrative Info
- What is this course about?
- Lecture 01
  - ADTs, Data Structures (and Algorithm), and Implementation
  - Review: Queues and stacks

Data Structures?

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

Data Structures (Examples)

Trade-Offs

- A data structure strives to provide many useful, efficient operations
- But 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

Terminologies

- 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

Stacks and Queue ADT

Stack ADT		Queue ADT
State: • Set of elements		State: • Set of elements
Operations: • **push(element)** • **pop()** – returns the most recent element that was added to the stack • ... etc.		Operations: • **enqueue(element)** • **dequeue()** – deletes and returns the element that has been in the queue the longest • ... etc.

Terminology Example: Stack

- The ***Stack*** ADT supports operations:
  - **push**: adds an item
  - **pop**: raises an error if isEmpty, else returns *most-recently pushed item* not yet returned by a pop
  - **isEmpty**: initially true, later true if there have been same number of pops as pushes
  - etc.
- A Stack data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **java.util.Stack**

Why useful

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

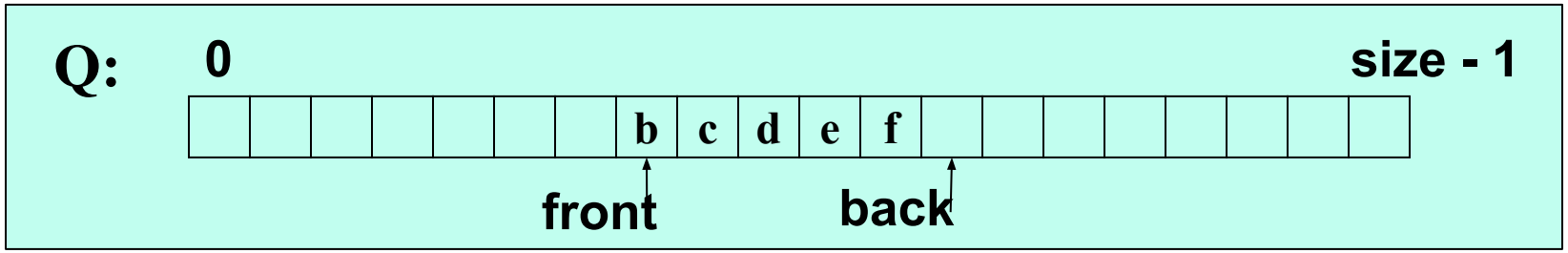
- It arises all the time in programming (see Weiss for more)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: 3 4 + 5 *
  - Clever: Infix ((3+4) * 5) to postfix conversion (see Weiss)
- We can code up a reusable library
- We can communicate in high-level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

Any Questions?

Terminology Example: Queue

- The ***Queue*** ADT supports operations:
  - **enqueue**: adds an item at the end
  - **dequeue**: raises an error if isEmpty, else returns item at the start
  - **isEmpty**: initially true, later true if there have been same number of enqueue as dequeues
  - etc.
- A Queue data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **java.util.Queue**

Circular Array Queue Data Structure

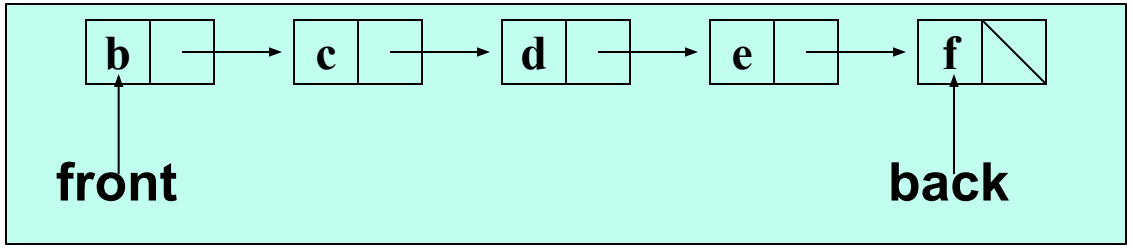



```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

```
// 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?
- How to *test* for empty?
- What is the *complexity* of the operations?

Linked List Queue Data Structure




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

```
// 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?

Any Questions?

Circular Array vs. Linked List

Array:	**List:**
• May waste unneeded space or run out of space • Space per element excellent • Operations very simple / fast	• Always just enough space • But more space per element • Operations very simple / fast

Operations not in Queue ADT, but also:	**Operations not in Queue ADT, but also:**
• Constant-time “access to k^{th} element” • For operation “insertAtPosition”, must shift all later elements	• No constant-time “access to k^{th} element” • For operation “insertAtPosition” must traverse all earlier elements

Homework Today :(

Sorry, the class is really condensed

1. EX 0 out today (spec on website)
   - due next monday
2. Check you have access to Ed
   - Email me if there are problems
   - https://courses.cs.washington.edu/courses/cse332/25su/calendar/lecturelist.html
3. Reading (optional)

Timeline

- ADTs, Data Structures (and Algorithm), and Implementation
  - Review: Queues and stacks
- What do we care about?
- Analyzing Code
  - Counting code constructs
  - Best Case vs. Worst Case
- Asymptotic Analysis
  - Big-Oh Definition