CSE 332 Summer 2026
Lecture 1: Intro to ADTs, Stacks, Queues

Michael Whitmeyer

http://www.cs.uw.edu/332

Michael Whitmeyer

- From NC, undergrad at Berkeley, PhD at UW!
  - I study complexity theory
- TA’ed many many times
  - Data structures, probability, discrete math, complexity, quantum, etc.
- First time instructor!
  - I hope to apply for teaching positions after this class concludes!

Topics

- **Data Structures**
  - Specific “classic” data structures
- Introduction to **Algorithms and Analysis**
- **Parallelism and Concurrency**
  - Parallelism: Use multiple processors to finish sooner
  - Concurrency: Correct access to shared resources
- Very brief intro to **complexity theory**

Warm Up!

Put up one hand (you can switch if it gets tired)!

Set your counter to 1

While you and at least 1 other person have their hand up:

1) Make a partnership with someone whose hand is still raised
2) Share your name with your partner
3) Add together your counter and your partner's counter
4) Identify which of you woke up earliest this morning
5) Release partnership
6) If you woke up later, then put your hand down and return to your seat

Course Info

- Text (optional):
  - Data Structures & Algorithm Analysis in Java, (Mark Allen Weiss), 3rd edition, 2012
    (2nd edition also o.k.)
- Course Page:
  - http://www.cs.uw.edu/332
- Topic Summaries
  - Posted on the website, a work in progress over the last couple quarters (mostly by Nathan, some by me)
  - Hopefully will eventually replace textbook

Communication

- Ed STEM Discussion board
  - Your first stop for questions about course content & assignments
- If emailing me, please include “332” or “CSE332” in subject

Course Meetings

- **Lecture**
  - Materials posted (slides before class, inked slides after)
  - Recorded using Panopto
  - Ask questions, focus on key ideas (rarely coding details)

- **Section**
  - Practice problems!
  - Answer Java/homework questions, etc.
  - Occasionally may introduce new material
  - An important part of the course

Office Hours

- Use them (come visit us!)
- See website for all OH
- I have 15 min 1-1 meetings available ☺

Grading

- Homework exercises (31% total, split evenly)
  - The lowest 2 will be dropped
- Midterm and final exam (65% total)
  - Midterm weighted 25%
  - Final weighted 40%
  - In-person
  - **Midterm in class on Wednesday 07/22**
  - **Final at 5-7pm on Thursday 08/20**
    - Makeup available if you have unavoidable conflict
- 17 Concept Checks (4% total)
  - The lowest 2 will be dropped
  - One per “concept” in the course
  - We instantly tell you whether you got each question correct
  - Unlimited submissions, so keep at it!

Collaboration

- **Try it yourself first**
  - Yourself ≠ AI
- Collaborate with classmates (no external interactive help on assignments permitted)
  - Collaboration is “whiteboard only”
  - Looking for a collaborator?
    - Post on the Ed Discussion board!
    - Go to the CSE study room (Allen Center 006, there’s a table specifically for 332!)
- Cite your sources!

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of a “thing” with set of operations on that “thing”
- Algorithm
  - A high level, language-independent description of a step-by-step process
- Data structure
  - An organization of data and family of algorithms for implementing an ADT
- Implementation of a data structure
  - The data organization and algorithms written in a programming language

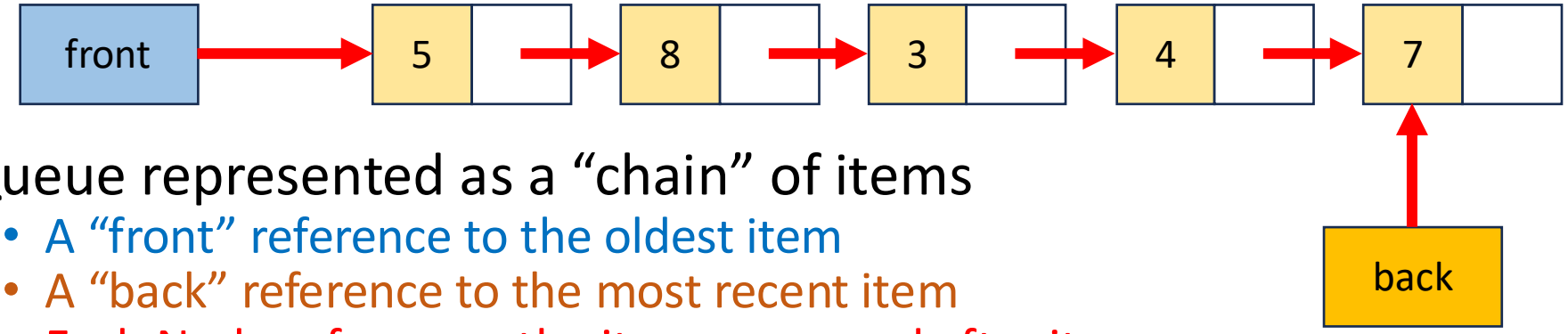
ADT: Queue (Activity)

- What is it?
- What operations do we need?
- Suggested data structures?

ADT: Queue

- What is it?
  - A collection of items that we interact with in a “First In First Out” (FIFO) way
- What operations do we need?
  - Enqueue
    - Add a new item to the queue
  - Dequeue
    - Remove the “oldest” item from the queue
  - IsEmpty
    - Indicate whether or not there are items still on the queue

Linked Queue Data Structure (Idea)

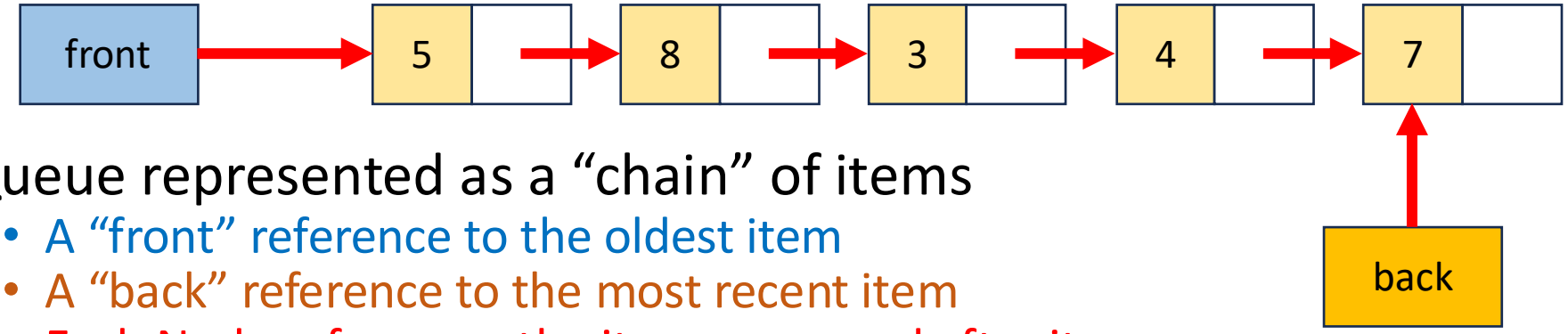


- Queue represented as a “chain” of items
  - A “front” reference to the oldest item
  - A “back” reference to the most recent item
  - Each Node references the item enqueued after it
- enqueue Procedure:

- dequeue Procedure:

- isEmpty Procedure:

Linked Queue Data Structure



- Queue represented as a "chain" of items
  - A "front" reference to the oldest item
  - A "back" reference to the most recent item
  - Each Node references the item enqueued after it
- enqueue Procedure:

```
enqueue(x):
    make new Node newBack containing x
    set back.next = newBack
    set back = newBack
```

- dequeue Procedure:

```
dequeue():
    x = front.value
    front = front.next
    return x
```

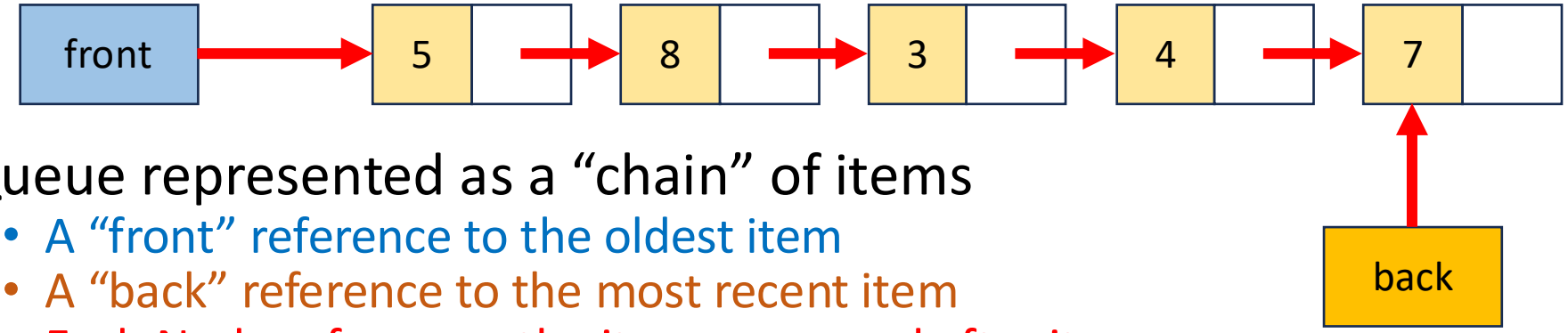
Any Issues?

What if back == front before dequeue?

- isEmpty Procedure:

```
isEmpty():
    return (front == null)
```

Linked Queue Data Structure



- Queue represented as a "chain" of items
  - A "front" reference to the oldest item
  - A "back" reference to the most recent item
  - Each Node references the item enqueued after it
- enqueue Procedure:

```
enqueue(x):
    make new Node newBack containing x
    set back.next = newBack
    set back = newBack
```

- dequeue Procedure:

```
dequeue():
    x = front.value
    front = front.next
    if (front == null): set back = null
    return x
```

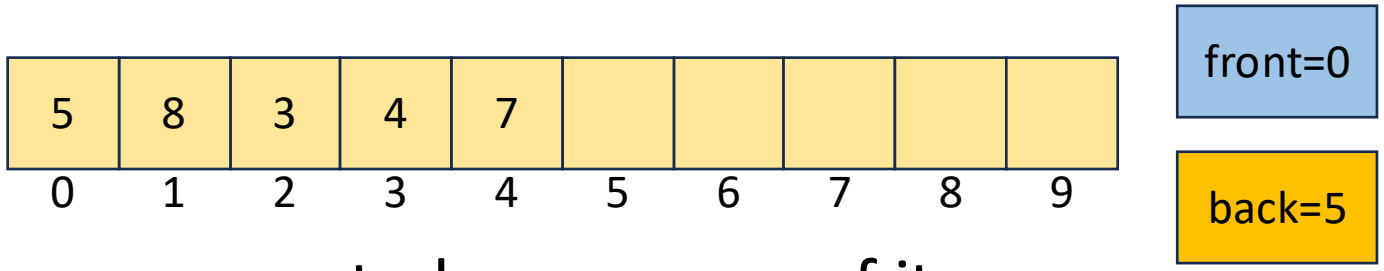
Any Issues?

What if back == front before dequeue?

- isEmpty Procedure:

```
isEmpty():
    return (front == null)
```

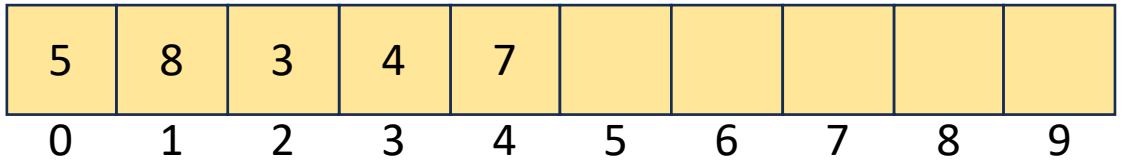
“Circular” Array Queue (Idea)



- Queue represented as an array of items
  - A “front” index to indicate the oldest item in the queue
  - A “back” index to indicate where the end of the queue is
    - Actually, the first “open” slot in the array
- enqueue Procedure:
- dequeue Procedure:
- isEmpty Procedure:

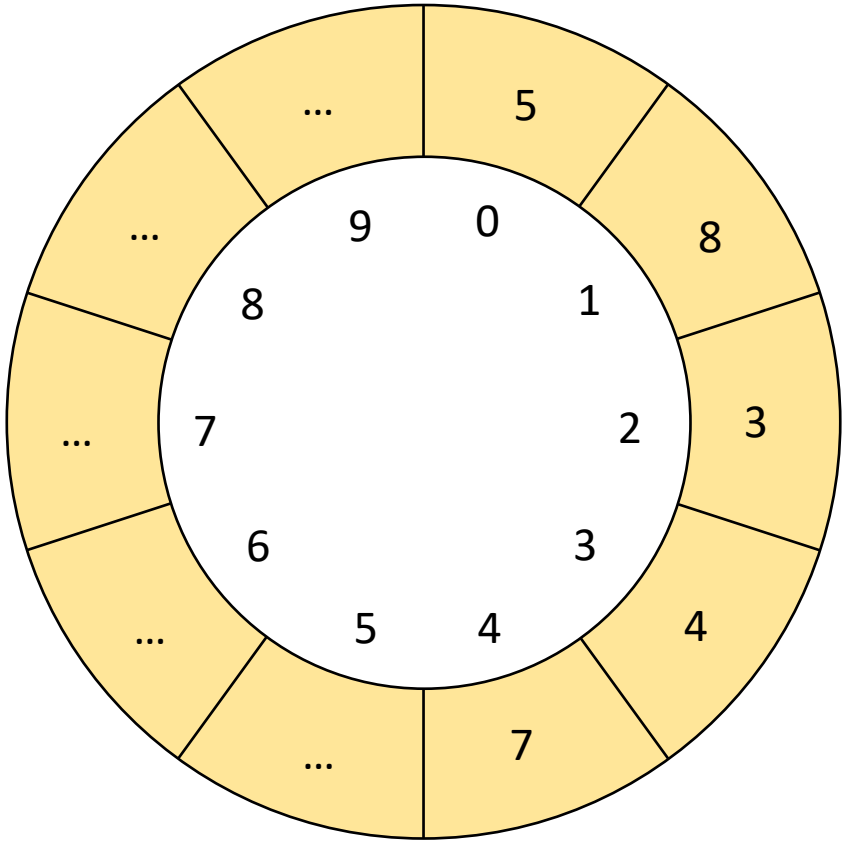
“Circular” Array

- Intuitively, An array of values arranged in a “circle” rather than a line
  - If you go beyond the last index, to wrap back around to 0

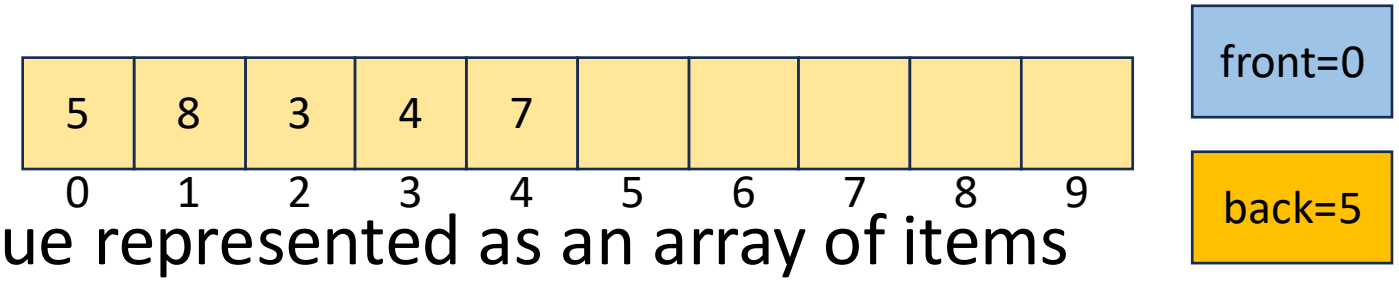


front=0

back=5



“Circular” Array Queue (Algorithms)



- Queue represented as an array of items
  - A “front” index to indicate the oldest item in the queue
  - A “back” index to indicate the most recent item in the queue
- enqueue Procedure:

```
enqueue(x):
   queue[back] = x
   back = (back + 1) % queue.length;
   size = size + 1
```

← Any Issues?

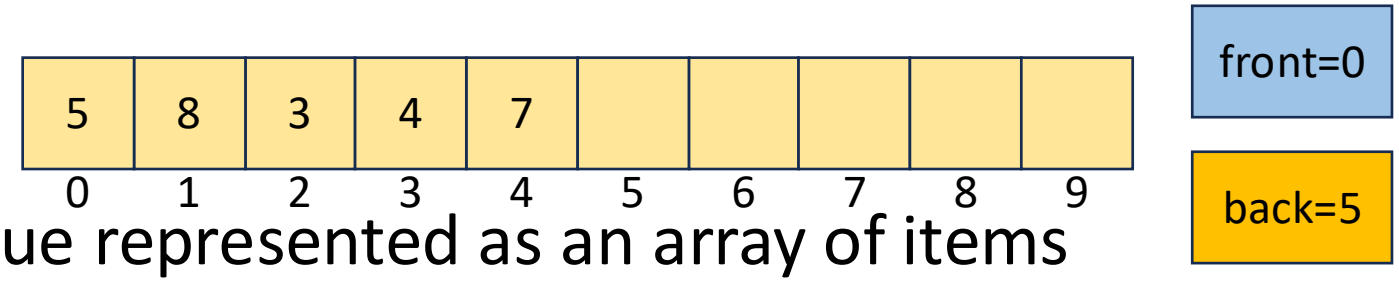
- dequeue Procedure:

```
dequeue():
   first = queue[front]
   front = (front + 1) % queue.length;
   size = size - 1
   return first
```

- isEmpty Procedure:

```
isEmpty():
   return (size == 0)
```

“Circular” Array Queue (Algorithms)



- Queue represented as an array of items
  - A “front” index to indicate the oldest item in the queue
  - A “back” index to indicate the most recent item in the queue
- enqueue Procedure:

```
enqueue(x):
   if (size == queue.length-1) then resize()
   queue[back] = x
   back = (back + 1) % queue.length;
   size = size + 1
```

- dequeue Procedure:

```
dequeue():
   first = queue[front]
   front = (front + 1) % queue.length;
   size = size - 1
   return first
```

- isEmpty Procedure:

```
isEmpty():
   return (size == 0)
```

How do you resize?

That’s for Exercise 0!

Linked List vs. Circular Array

- Let’s Summarize the benefits and drawbacks of each

ADT: Stack

- What is it?
  - A “Last In First Out” (LIFO) collection of items (sometimes called FILO)
- What operations do we need?
  - push
    - Add a new item onto the stack
  - peek
    - Return the value of the most recently pushed item
  - pop
    - Return the value of the most recently pushed item and remove it from the stack
  - isEmpty
    - Indicate whether or not there are items still on the stack

Runtime Analysis (motivating example)

- Suppose I am tied into my climbing rope, but the rest of it is a big mess
- I need to find the other end of the rope!
- How can I do this efficiently?

Algorithm Ideas

- Whatcha got?

A Possible Algorithm

Set aside the already-obtained beginning
Do the following until you find the end:
- Separate the pile of rope into 2 piles
- Label A to be the pile that the beginning enters
- Label B to be the other pile
- Count the number of strands crossing the piles
- If count is even, set the pile to be A
- Otherwise set the pile to be B.