

CSE 332 Winter 2026

Lecture 1: Intro to ADTs, Stacks, Queues

Nathan Brunelle

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

Nathan Brunelle

- Born: Virginia Beach, VA
- Ugrad: Math and CS at University of Virginia
- Grad: CS at University of Virginia
- Taught at UVA for 6 years
 - Intro to programming (e.g. 121)
 - Discrete Math (e.g. 311)
 - Algorithms (e.g. 421)
 - Theory of Computation (e.g. 431)
- Year 3 at UW
 - CSE123
 - CSE332
 - CSE373
 - CSE417
 - CSE421



About this course

Topics covered:

- 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

Warm Up!

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

Set your counter to 1

While (you and at least one other person have a hand up){

- Make a partnership with someone whose hand is still raised

- Share your name with your partner

- Add together your counter and your partner's counter

- Identify which of you woke up earliest this morning

- Release partnership

- If you woke up later, then put your hand down and return to your seat

}

Course Staff

- Instructor:
 - Nathan Brunelle
- TAs:

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>

Communication

- Ed STEM Discussion board
 - Your first stop for questions about course content & assignments

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 (not optional)
- Office hours
 - Use them: *please visit us!*

Grading

- 13 \geq Weekly homework exercises (4.5% each, 54% total)
 - The lowest 1 will be dropped
 - Except for EX12, which cannot be dropped
- Midterm and final exam (40% total)
 - Midterm weighted 15%
 - Final weighted 25%
 - In-person
 - Midterm in class on Friday 11/1
 - Final at 8:30am on Thursday 12/2
- Concept Checks (0.33% each, 6% total)
 - 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
- 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

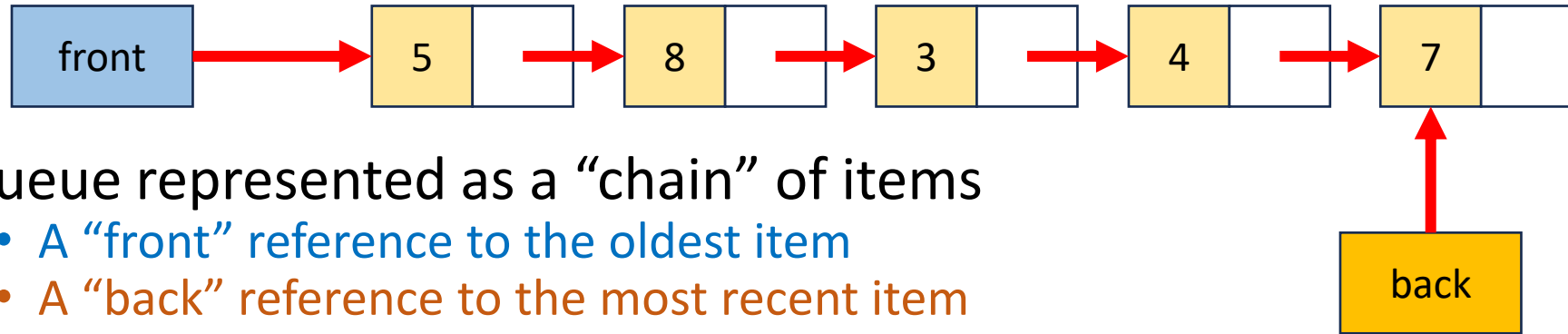
ADT: Queue (Activity)

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

ADT: Queue

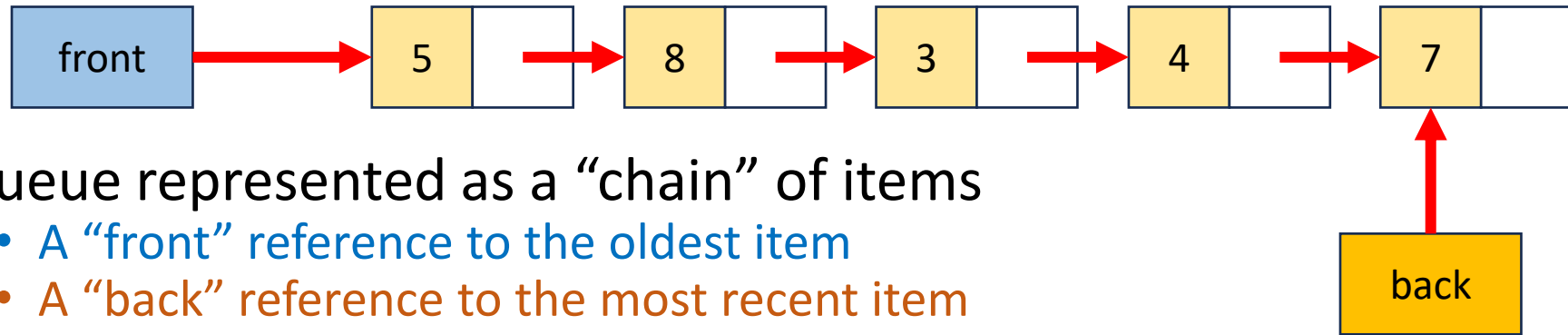
- What is it?
 - A collection of items that we interact with in a “First In First Out” (FIFO) way
- What operations do we need?
 - Enqueue
 - Add a new item to the queue
 - Dequeue
 - Remove the “oldest” item from the queue
 - IsEmpty
 - Indicate whether or not there are items still on the queue

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 (Algorithms)



- 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

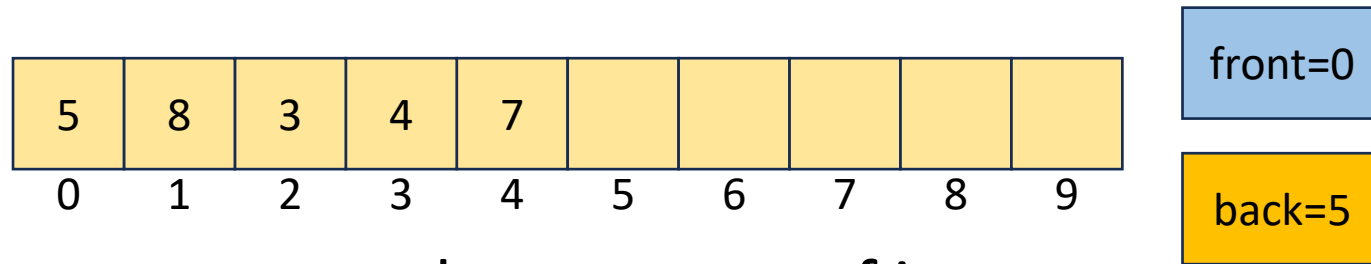
- enqueue Procedure:

```
enqueue(x){
    last = new ListNode(x);
    back.next = last;
    back = last;
}
```
- dequeue Procedure:

```
dequeue(){
    first = front.value;
    front = front.next;
    if (front == null) {back = null;}
    return first
}
```
- isEmpty Procedure:

```
isEmpty(){
    return front == null;
}
```

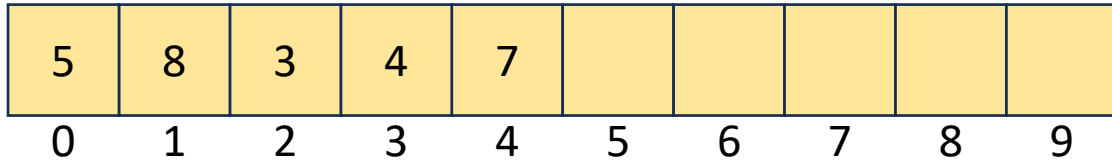
“Circular” Array Queue (Idea)



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

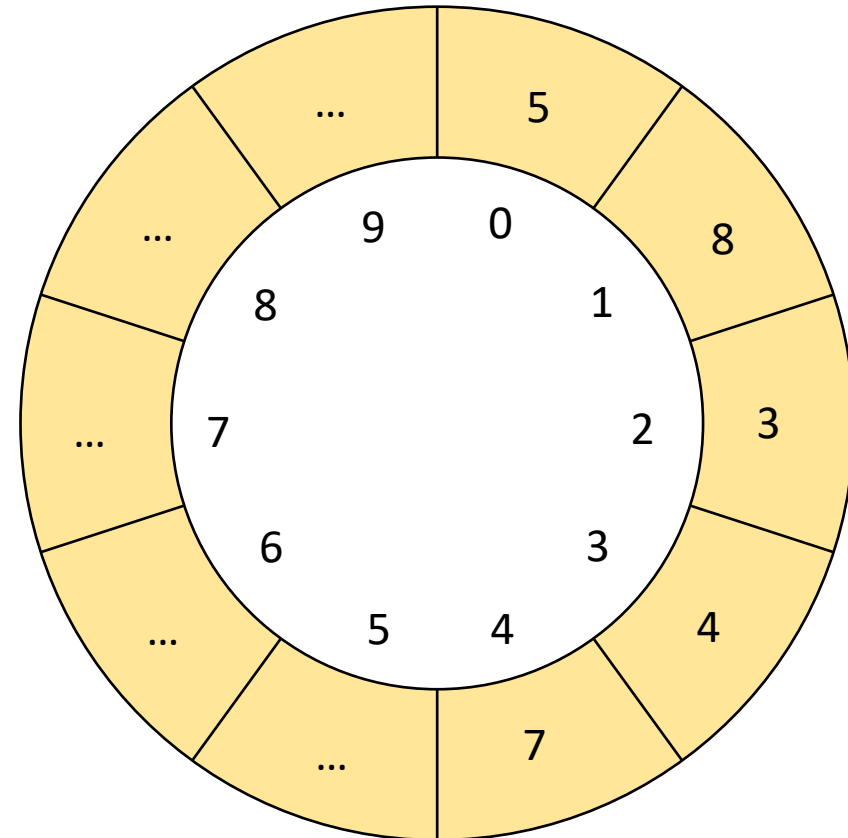
“Circular” Array

- 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

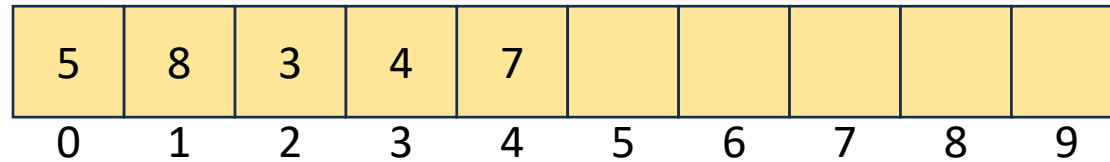


front=0

back=5



“Circular” Array Queue (Algorithms)



front=0

back=5

- 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++;  
}
```

- dequeue Procedure:

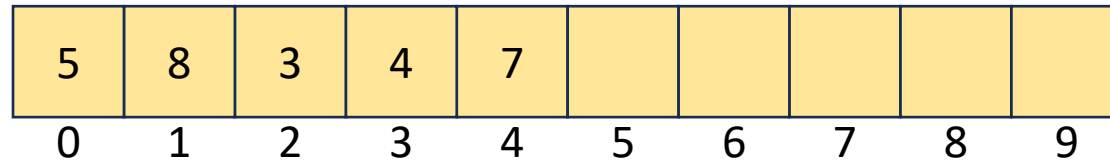
```
dequeue(){  
    first = queue[front];  
    front = (front + 1) % queue.length;  
    size--;  
    return first;  
}
```

- isEmpty Procedure:

```
isEmpty(){  
    return size == 0;  
}
```

What if we run out of space?!

Resizing “Circular” Array Queue



front=0

back=5

How do you resize?

That's for Exercise 0!

- 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) {resize();}  
    queue[back] = x;  
    back = (back + 1) % queue.length;  
    size++;  
}
```

- dequeue Procedure:

```
dequeue(){  
    // Assumes queue is not empty  
    first = queue[front];  
    front = (front + 1) % queue.length;  
    size--;  
    return first;  
}
```

- isEmpty Procedure:

```
isEmpty(){  
    return size== 0;  
}
```

Linked List vs. Circular Array

- Let's Summarize the benefits and drawbacks of each

ADT: Stack (Activity)

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

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