CSE 373: Data Structures and Algorithms

Lecture 1: Introduction, ADTs, Stacks & Queues

Instructor: Lilian de Greef
Quarter: Summer 2017

Welcome!

Today's Structure:

- Introductions and course mechanics
- Start material
  - Abstract Data Types (ADTs)
  - Stacks
  - Queues

Lilian de Greef

- CSE PhD Student
- Working with Shwetak Patel on health applications of CS
- Interests & Hobbies
  - Ultimate Frisbee
  - Piano
  - Hiking / backpacking
  - Some TV shows

ldegreef@cs.washington.edu

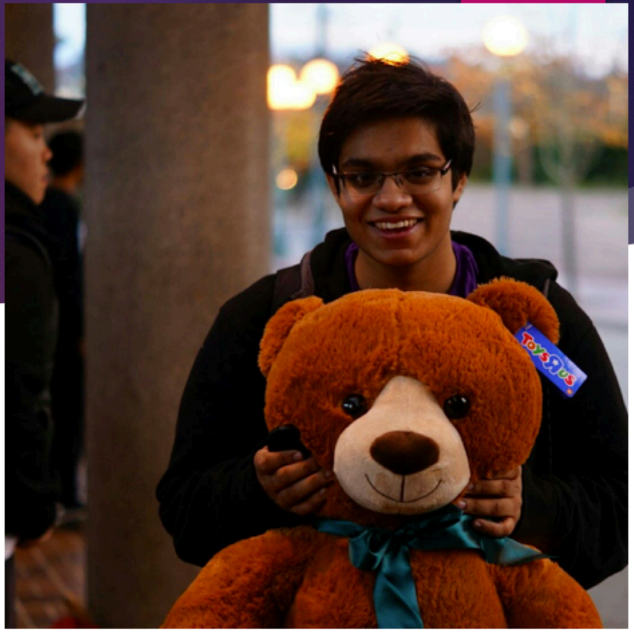
cse373-staff@cs.washington.edu

Kyle Thayer

Ben Jones

Vlad Shamalov

Anupam Gupta



- Junior - Majoring in CSE and HCDE.
- Hobbies: Watching Movies, Sleeping.
- Interests: AI, Programming Languages, Data Mining.
- Why TA? Because it's a lot of fun and also because I get to meet a lot of new, fun, people and talk to them about CS (which is awesome!!)
- See you all around!

Course Logistics

Classroom environment

- Laptop policy
- Lectures starting promptly at 10:50
- Will have discussions in class
  - With neighbors
  - With entire class
  - Hence, pack yourselves to the front and sit together
- Somewhere we can feel comfortable making mistakes
  - One of the best ways to learn!

General Logistics

- Website: http://cs.washington.edu/373
- Mailing list: cse373a_17su@uw.edu
- Piazza discussion board
- Textbook: Weiss 3rd Edition in Java
- Computers for homework assignments
  - College of Arts & Sciences Instructional Computing Lab: http://depts.washington.edu/aslab/
  - Or your own machine
- Java
  - Used for programming assignments
  - Recommended environment: Eclipse

Sections & Office Hours

- TBA by Tuesday, in class on Wednesday
- Lilian's office hours (*for just today)*:
  - 1:00 – 2:00pm
  - CSE 220

Contact

- Use Piazza!
  - https://piazza.com/washington/summer2017/cse373
  - Don't post code or solutions publicly
  - For questions with code, solutions, grades, etc., make private posts to instructors
  - Can post anonymously
- Email me
  - For "Lilian's eyes only" concerns
  - I'll reply within 24 hours
  - Put [CSE 373] at beginning of subject

Collaboration and Academic Integrity

DON'T CHEAT!

Seriously, read the policy online.

Using PollEverywhere

- How:
  - You anonymously vote on multiple choice questions in lecture
  - Via text messaging (SMS) or web browser (don’t need to buy a clicker)*
- Why:
  - A way for me to check in
  - A way for *you* to check in
  - Research shows using Peer Instruction with polling improves learning!

* If access to SMS or a web browser in class is a challenge for you, please come talk to me

Using PollEverywhere: for Peer Instruction

- Format
  1. I'll pose a question
  2. Vote individually, invisible to class
  3. Discuss!
  4. Group vote
- Discussion is key!
  - "Just getting the right answer" is not enough - need to be able to explain/argue for it!
  - Testing yourself helpful ("right answer"), but <u>learning</u> happens during discussion

Take part in class-wide discussion!

- I know, can be intimidating
- Your questions and explanations are critical for fellow students' learning
- If you have a question, it’s likely that others have the same one. You're not alone!

Let’s get started with Data Structures!

Today: Abstract Data Types (ADTs), Stacks, Queues

Expectations: Basic Understanding of

- Conditionals
- Loops
- Methods
- Fundamentals of defining classes and inheritance
- Basic algorithm analysis (e.g. O(n) vs O(n^2) etc.)
- Arrays
- Singly linked lists
- Simple binary trees
- Recursion
- A few sorting and searching algorithms

What is a Data Structure?

What is a Data Structure?

What should I put
my sandwich in?

The crux of this course

- Understanding your data structures and algorithms to choose the right one for the job.
- Fundamental CS skill
- After this course, I want you to be able to
  - Make good design choices
  - Justify and communicate design decisions

Terminology

- **Abstract Data Type (ADT)**
  - Mathematical description of “thing”
    - Meaning
    - Operations
  - No implementation details

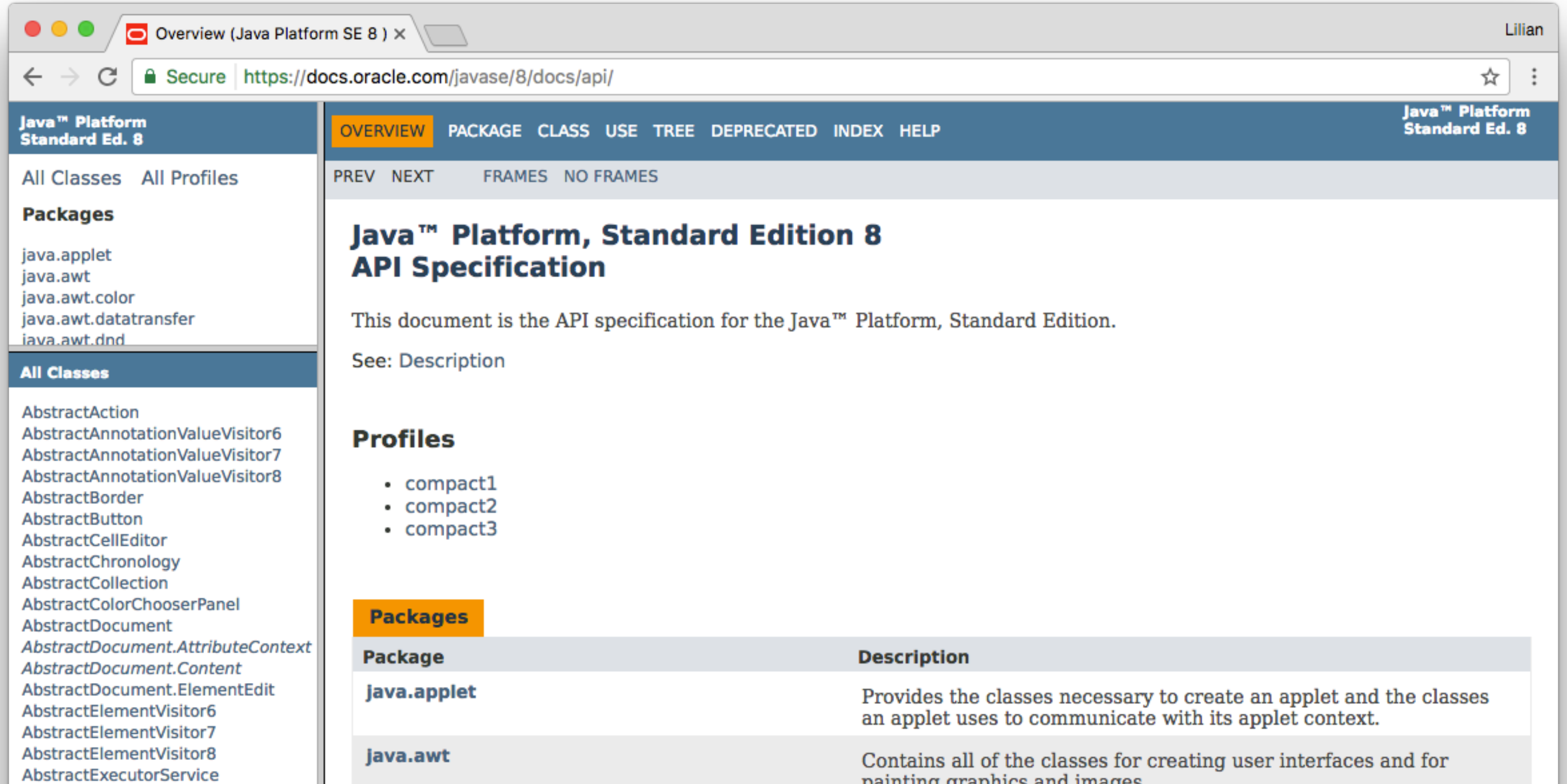
- **Data structure**
  - Specific way to implement ADT (organization of data & family of algorithms)

Terminology

- **Algorithm**
  - Language-independent description of step-by-step process
- **Implementation** of a data structure
  - Specific implementation in a specific language

Terminology

Interface to an ADT in particular language is said to be the **Application Programmer Interface (API)** for the ADT in that language

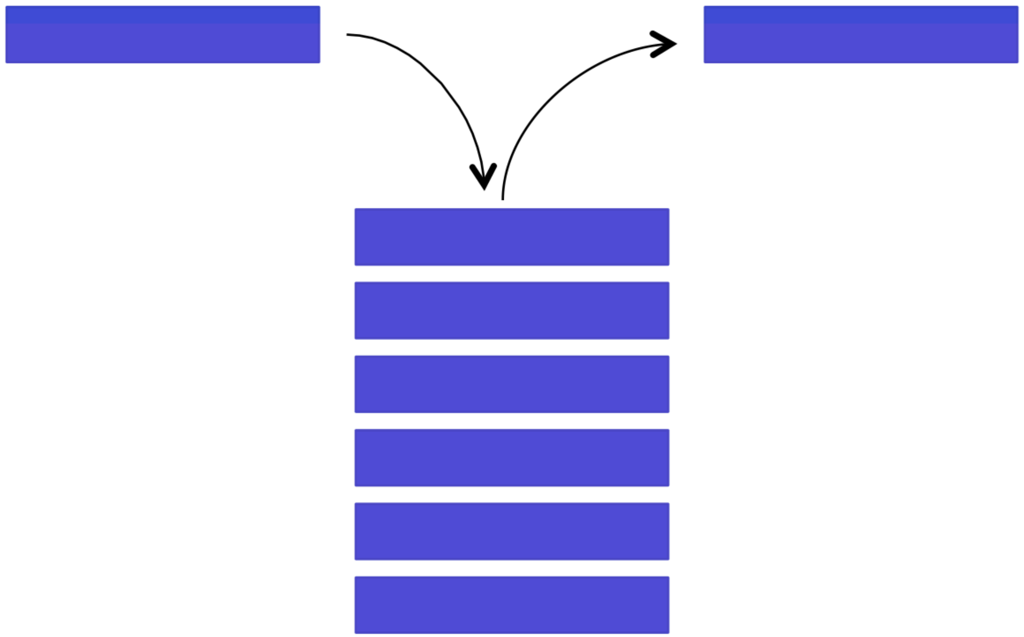


Computer Science example: Stacks!

Stack ADT

- Meaning

- Operations



Stack data structures

- Specific kinds of stacks:

- Example implementation: library “java.util.Stack”

Stack Practice!

- As an array

1. `new Stack`
2. `push(☺)`
3. `push(☆)`
4. `pop()`

- As a linked list

Stacks are used a lot!

- Undo / redo
- Back / forward on browsers
- Recursion
- Matching braces

{ ((a + b) * c – (d / (e + f)) }

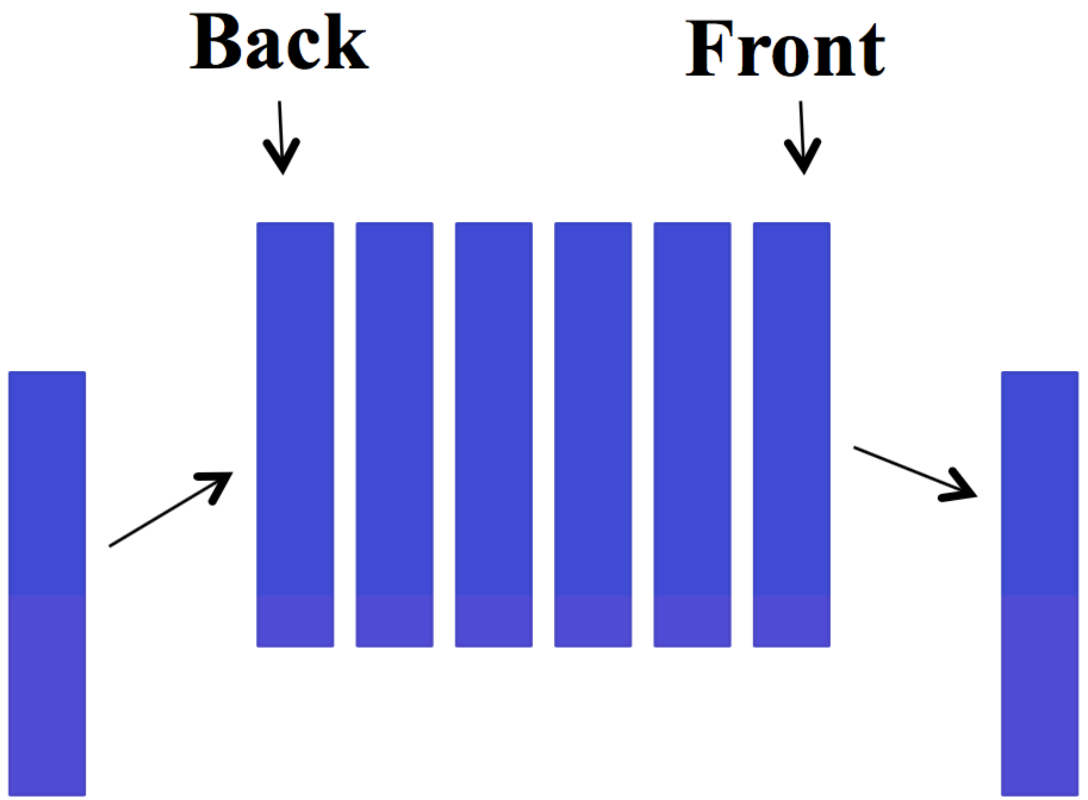
- … and much more!

Another example: Queues!

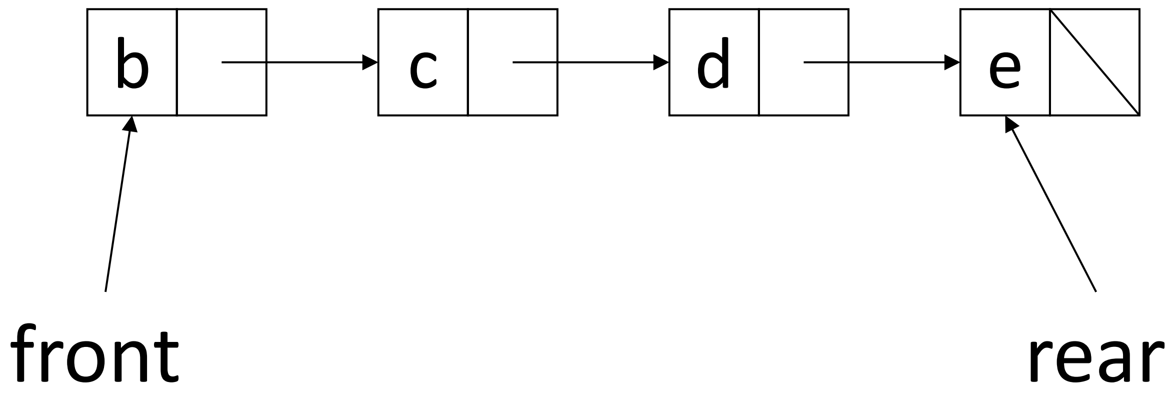
Queue ADT

- Meaning

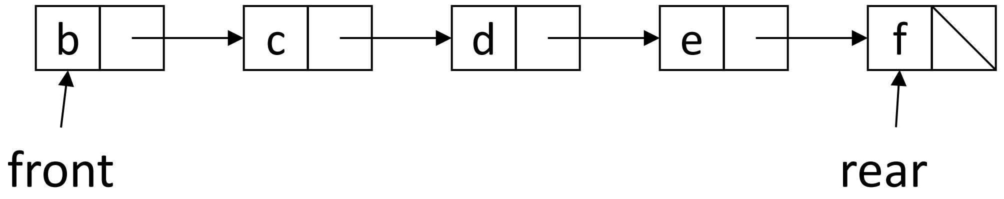
- Operations



Queue Data Structure: Linked List



Queue Data Structure: Linked List

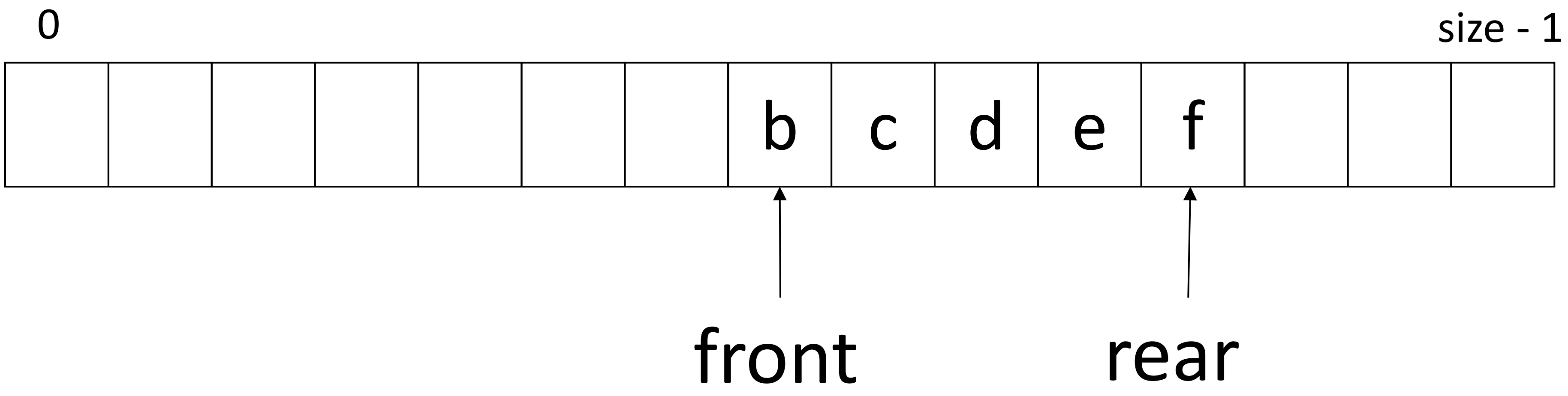



```
// Basic idea only!
enqueue(x) {
  rear.next = new Node(x);
  rear = rear.next;
}
```

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

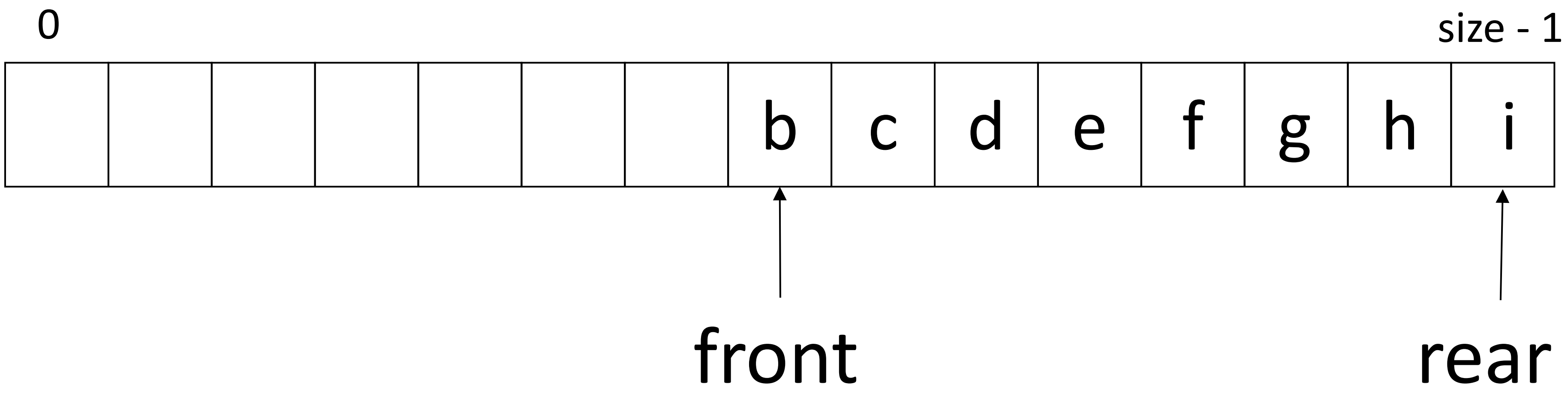
- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- Can you find the kth element in the queue?
- Can ***list*** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?

Queue Data Structure: Array



What happens when we dequeue several times, and *front* catches up to *rear*?

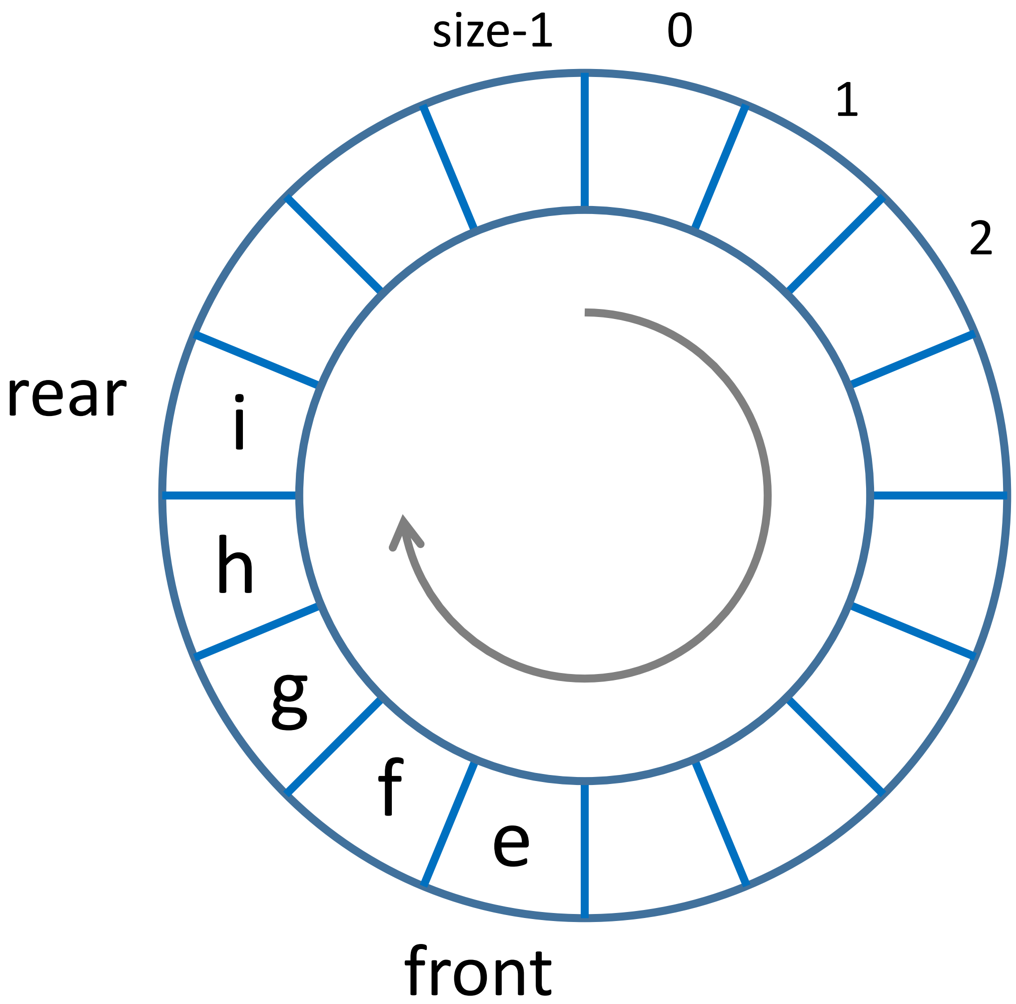
Queue Data Structure: Array



Hmmm…

How do we enqueue to the rear now?

Queue Data Structure: Circular Array!

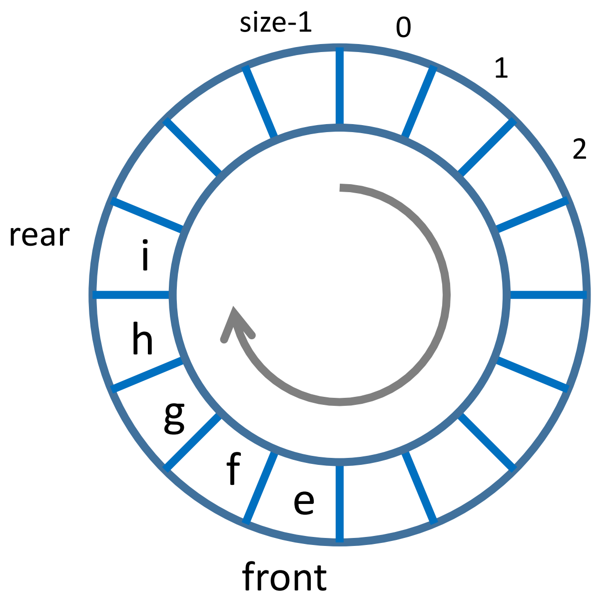


View the array as *circular* and allow both *front* and *rear* to advance through (around) the array

We wouldn’t need to move elements for enqueues and dequeues!

If we can assume the queue is not empty, how can we implement dequeue()?

```
Public E dequeue() {
   size--;
   E e = array[front];
   <Your code here!>
   return e;
}
```



A)
```
front++;
if (front == array.length)
   front = 0;
```

B)
```
rear = rear-1;
if (rear < 0)
   rear = array.length-1;
```

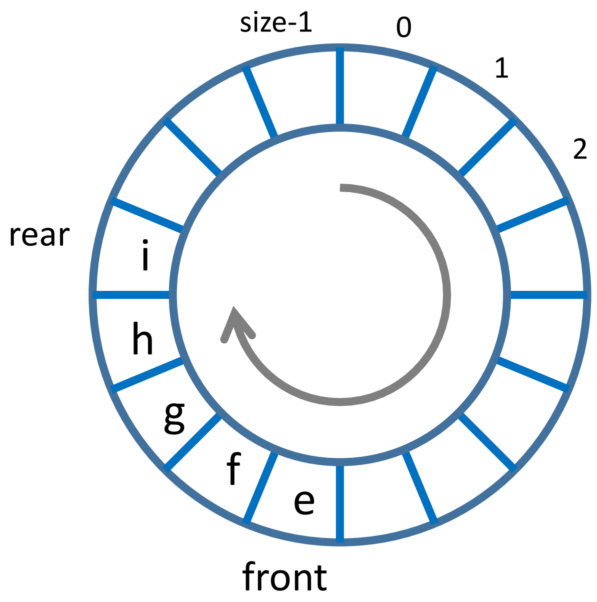
C)
```
for (int i = 0; i < rear; i++) {
   array[i] = array[i+1]
}
front++;
if (front == array.length)
   front = 0;
```

D) None of these are correct

(Notes for yourself)

If we can assume the array is not full, how can we implement enqueue(E e)?

```
Public enqueue(E e) {
    <Your code here!>
    size++;
}
```



A)
```
rear++;
if (rear == array.length)
    rear = 0;
array[rear] = e;
```

B)
```
rear++;
array[rear] = e;
```

C)
```
for (int i=front; i<rear; i++) {
    array[i] = array[i+1]
}
array[rear] = e;
rear++;
```

D) None of these are correct

(Notes for yourself)

Between arrays and linked-lists which one *always* is the fastest at `enqueue`, `dequeue`, and `seeKthElement` operations?

(where `seeKthElement` lets you peek at the kth element in the stack)

Fastest:	`enqueue`	`dequeue`	`seeKthElement`
A)	Arrays	Linked-Lists	Neither
B)	Linked-lists	Neither	Neither
C)	Linked-lists	Neither	Arrays
D)	*They're all the same*		

(Notes for yourself)

Which one’s better?

Arrays

Linked-lists

Trade-offs!

- The ability to choose wisely between trade-offs is why it’s important to understand underlying data structures.
- Common Trade-offs
    - Time vs space
    - One operation’s efficiency vs another
    - Generality vs simplicity vs performance