Lecture 2: Lists

CSE 373: Data Structures and Algorithms

Warm Up

Reminder: Please open www.pollev.com/cse373studentqs to ask questions :)

Please go to www.pollev.com/cse373activity to respond

Which of the following is NOT an "ADT":

- List
- ArrayList
- Set
- Map
- TreeMap
- Stack
- Queue

From our last lecture:

Abstract Data Type (ADT)

- *A definition for expected operations and behavior*
- A mathematical description of a collection with a set of supported operations and how they should behave when called upon
- Describes what a collection does, not how it does it
- Can be expressed as an interface

Data Structure

- *A way of organizing and storing related data points*
- An object that implements the functionality of a specified ADT
- Describes exactly how the collection will perform the required operations

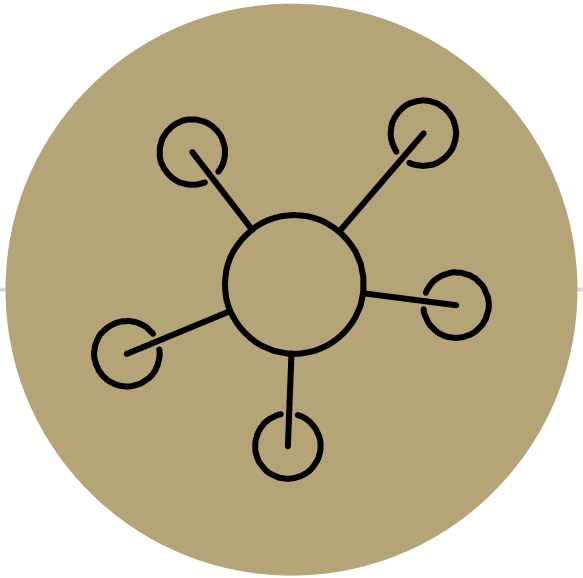
Administration

Project 0 released later today

- Due next **Wednesday April 8th before 11:59pm PST**
- Check “Projects” tab on website, live later today
- Learning Goals
  - Refresh CSE 143 concepts and set the expectations for homework in CSE 373.
  - Set up IntelliJ and other systems we’ll use in this class (Java, GitLab, Git, Checkstyle).
  - Learn how to use JUnit and run unit tests in IntelliJ.

Let’s connect!

- Check out the piazza
- Office hours start next week
  - Limited offering this week to help with set up
- Section
  - Starts tomorrow



Question Break

Review: "Big Oh"

Note: You don't have to understand all of this right now – we'll dive into it soon.

efficiency: measure of computing resources used by code.
- can be relative to speed (time), memory (space), etc.
- most commonly refers to run time

Assume the following:
- Any single Java statement takes same amount of time to run.
- A method call's runtime is measured by the total of the statements inside the method's body.
- A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.

We measure runtime in proportion to the input data size, N.
- **growth rate**: Change in runtime as N gets bigger. How does this algorithm perform with larger and larger sets of data?

```
b = c + 10;

for (int i = 0; i < N; i++) {
   for (int j = 0; j < N; j++) {
      dataTwo[j][i] = dataOne[i][j];
      dataOne[i][j] = 0;
   }
}
for (int i = 0; i < N; i++) {
   dataThree[i] = b;
}
```

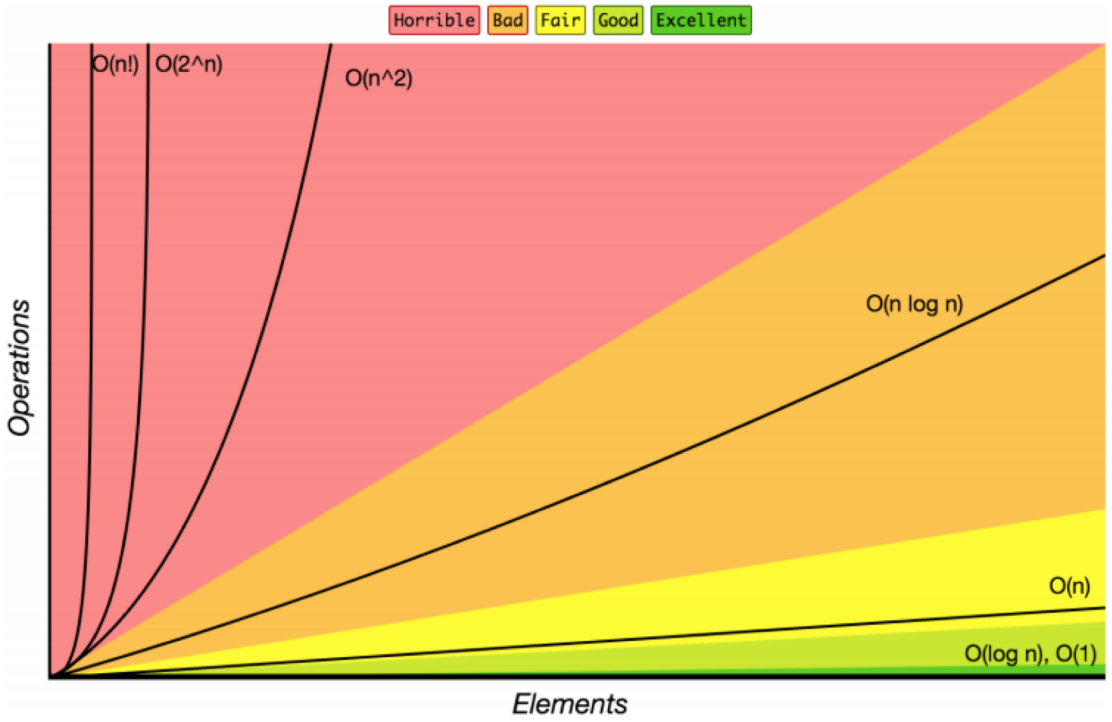
This algorithm runs $\mathbf{2N^2 + N + 1}$ statements.
- We ignore constants like 2 because they are tiny next to N.
- The highest-order term (N^2) "dominates" the overall runtime.
- We say that this algorithm runs "on the order of" N^2.
- or $\mathbf{O(N^2)}$ for short ("**Big-Oh** of N squared")

Review: Complexity Class

Note: You don't have to understand all of this right now – we'll dive into it soon.

complexity class: A category of algorithm efficiency based on the algorithm's relationship to the input size N.

Complexity Class	Big-O	Runtime if you double N	Example Algorithm
constant	O(1)	unchanged	Accessing an index of an array
logarithmic	$O(\log_2 N)$	increases slightly	Binary search
linear	O(N)	doubles	Looping over an array
log-linear	$O(N \log_2 N)$	slightly more than doubles	Merge sort algorithm
quadratic	$O(N^2)$	quadruples	Nested loops!
…	…	…	…
exponential	$O(2^N)$	multiplies drastically	Fibonacci with recursion



Case Study: The List ADT

list: a collection storing an ordered sequence of elements.

- Each item is accessible by an index.
- A list has a variable size defined as the number of elements in the list
- Elements can be added to or removed from any position in the list

Relation to code and our mental image of a list:

```
List<String> names = new ArrayList<>();          // []
names.size();                                    // evaluates to 0
names.add(”Amanda");                             // [“Amanda”]
names.add(”Anish");                              // [“Amanda, Anish”]
names.insert("Brian”, 0);                        // [“Brian”, “Amanda”, “Anish”]
names.size();                                    // evaluates to 3
```

Case Study: List Implementations

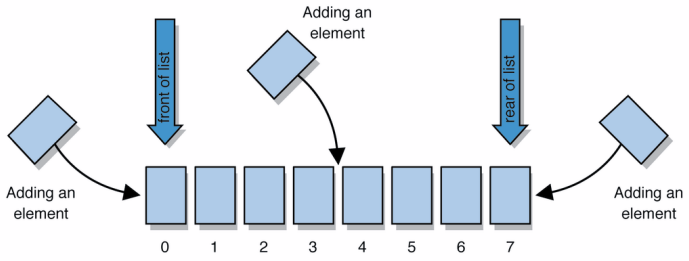
List ADT

state

Set of ordered items
Count of items

behavior

get(index) return item at index
set(item, index) replace item at index
add(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items



ArrayList

uses an Array as underlying storage

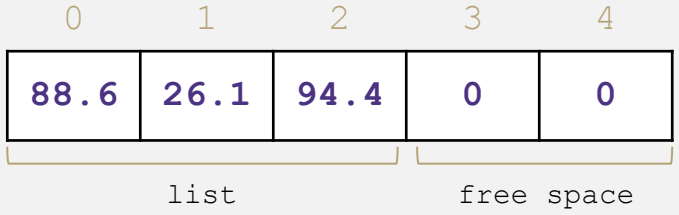
ArrayList<E>

state

data[]
size

behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size



LinkedList

uses nodes as underlying storage

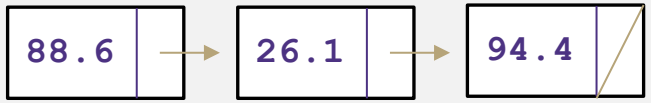
LinkedList<E>

state

Node front
size

behavior

get loop until index, return node's value
set loop until index, update node's value
add create new node, update next of last node
insert create new node, loop until index, update next fields
delete loop until index, skip node
size return size



Case Study: The List ADT: ArrayList

How do Java / other programming languages implement ArrayList to achieve all the List behavior?

On the inside:

- stores the elements inside an array (which has a fixed capacity) that typically has more space than currently used (For example when there is only 1 element in the actual list, the array might have 10 spaces for data),

- stores all of these elements at the front of the array and keeps track of how many there are (the size) so that the implementation doesn't get confused enough to look at the empty space. This means that sometimes we will have to do a lot of work to shift the elements around.

List view

["Brian", "Amanda", "Anish"]

ArrayList view

["Brian", "Amanda", "Anish", null, null, null, null]
(this is the internal array with extra space)

Implementing ArrayList

ArrayList<E>

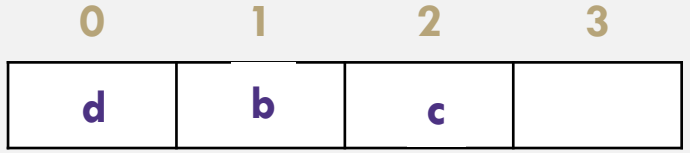
state
- data[]
- size

behavior
- get return data[index]
- set data[index] = value
- add data[size] = value, if out of space make data into a bigger array and copy everything over
- insert shift values to make hole at index, data[index] = value, if out of space grow data
- delete shift following values forward
- size return numberOfItems

Take 1 Minute

`insert(element, index)` with shifting

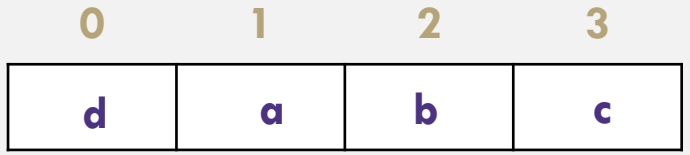
`insert("d", 0)`



`size =` 4

`delete(index)` with shifting

`delete(0)`



`size =` 3

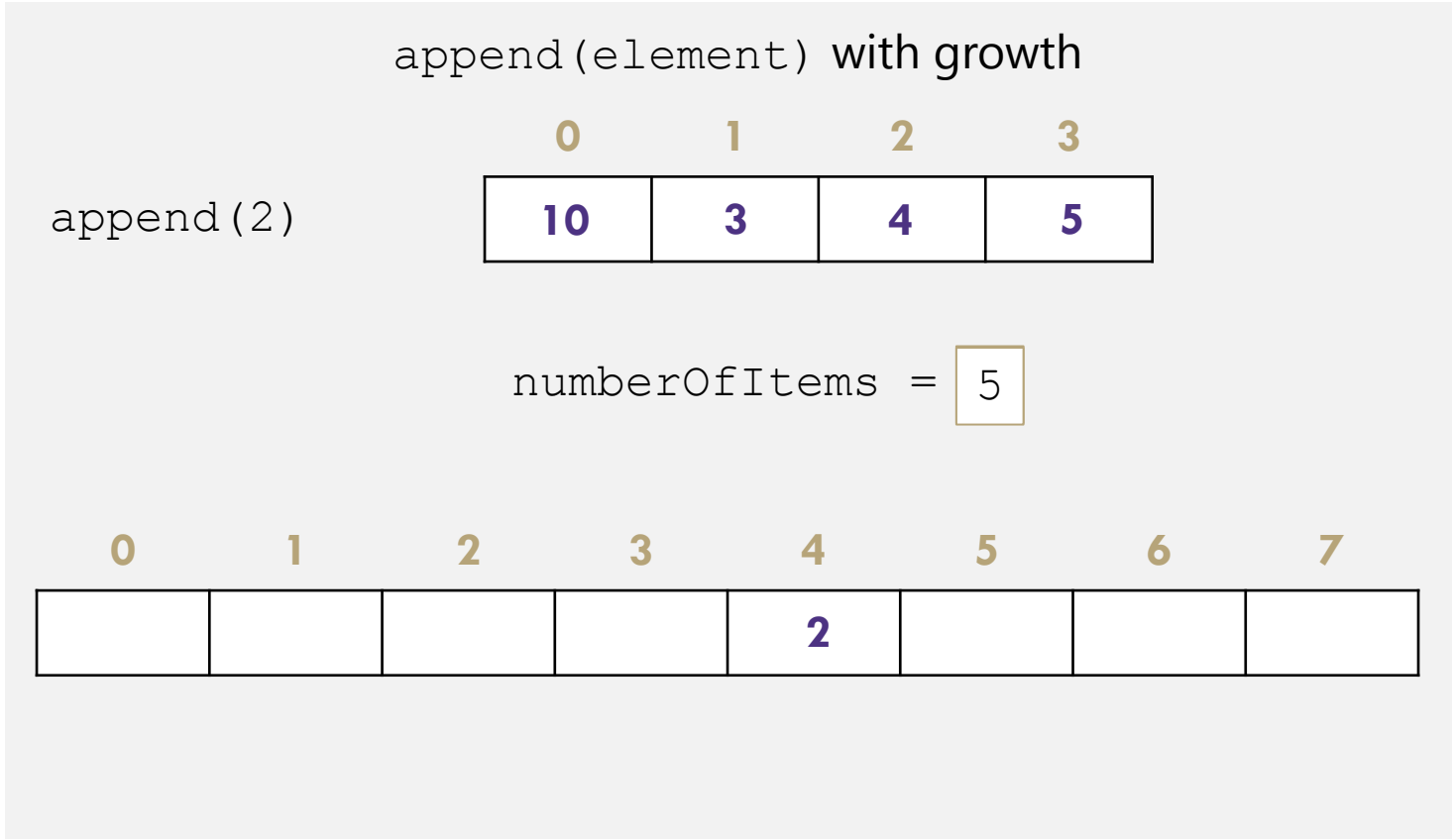
Yes or No: should we overwrite index 3 (the old c) with null? Please go to www.pollev.com/cse373activity to submit a 'Yes' or 'No' answer with a brief explanation why you think so.

Implementing ArrayList

ArrayList<E>

state
- data[]
- size

behavior
- get return data[index]
- set data[index] = value
- add data[size] = value, if out of space grow data
- insert shift values to make hole at index, data[index] = value, if out of space grow data
- delete shift following values forward
- size return size



Case Study: List Implementations

List ADT

state
- Set of ordered items
- Count of items

behavior
- get(index) return item at index
- set(item, index) replace item at index
- add(item) add item to end of list
- insert(item, index) add item at index
- delete(index) delete item at index
- size() count of items

Take 1 Minute

What method/situations will be much faster for LinkedList than for ArrayList?

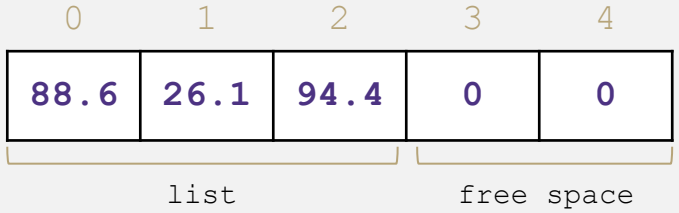
Please go to www.pollev.com/cse373activity to submit a method/situation with a

ArrayList
uses an Array as underlying storage

ArrayList<E>

state
- data[]
- size

behavior
- get return data[index]
- set data[index] = value
- add data[size] = value, if out of space grow data
- insert shift values to make hole at index, data[index] = value, if out of space grow data
- delete shift following values forward
- size return size

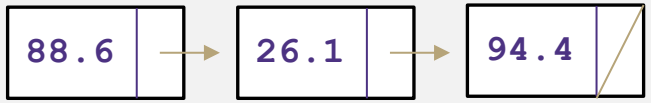


LinkedList
uses nodes as underlying storage

LinkedList<E>

state
- Node front
- size

behavior
- get loop until index, return node's value
- set loop until index, update node's value
- add create new node, update next of last node
- insert create new node, loop until index, update next fields
- delete loop until index, skip node
- size return size



Design Decisions

For every ADT there are lots of different ways to implement them

Based on your situation you should consider:

- Memory vs Speed
- Generic/Reusability vs Specific/Specialized
- One Function vs Another
- Robustness vs Performance

This class is all about implementing ADTs based on making the right design tradeoffs!

> A common topic in interview questions

Design Decisions

Take 3 Minutes

Dub Street Burgers is implementing a new system for ticket (i.e. food order) management.

When a new ticket comes in, it is placed at the end of the set of tickets.

Food is prepared in approximately the same order it was requested, but sometimes tickets are fulfilled out of order.

Let's represent tickets as a list. Which of our ADT implementations should we use?

Why?

We're going to try some more online activities here: please have some mercy

We're going to group you all into groups of ~5 students with the Zoom breakouts feature, click here for some instructions with screenshots about how to use it if you're stuck / not sure. You should be able to just click join, but if you're trouble check out the slides.

With your group, discuss and submit an answer (or multiple if you have different opinions!) and vote at

www.pollev.com/cse373activity

Design Decisions

Let’s represent tickets as a list. Which of our ADT implementations should we use?

Why?

ArrayList

Creating a new ticket is very fast (as long as we don’t resize), and I want the cooks to be able to see all the orders right away.

LinkedList

We’ll mostly be removing from the front of the list, which is much faster for the linkedlist (no shifting), and I want finished orders to be removed fast so they aren’t distracting.

Design Decisions

Both ArrayList and LinkedList implementations have pros and cons.

Neither is strictly better than the other.

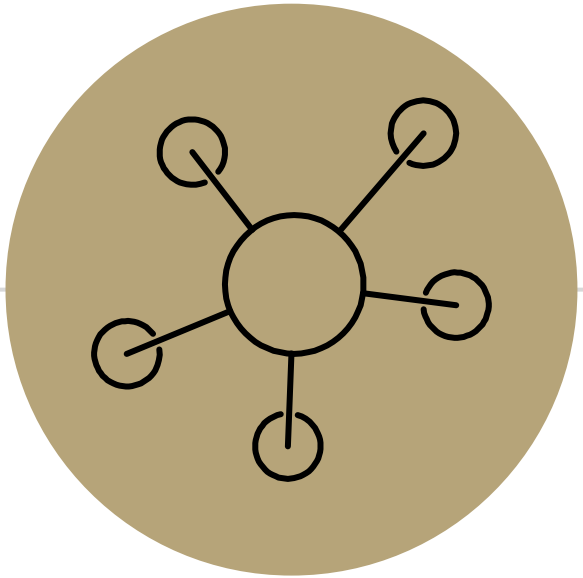
Some major objectives of this course:

Evaluating pros and cons

Deciding on a design

Defending that design decision

Especially when there's more than one possible answer.



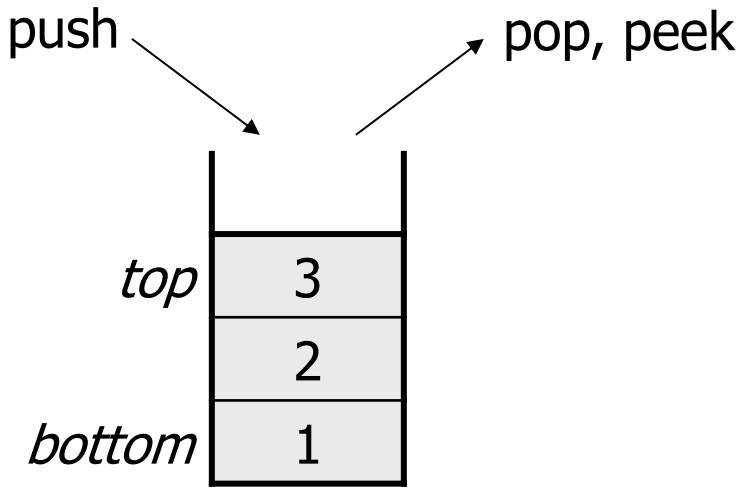
Question Break

This is where we stopped in lecture

Review: What is a Stack?

stack: A collection based on the principle of adding elements and retrieving them in the opposite order.

- Last-In, First-Out ("LIFO")
- Elements are stored in order of insertion.
  - We do not think of them as having indexes.
- Client can only add/remove/examine the last element added (the "top").



Stack ADT
state Set of ordered items Number of items
behavior push(item) add item to top pop() return and remove item at top peek() look at item at top size() count of items isEmpty() count of items is 0?

supported operations:

- **push(item)**: Add an element to the top of stack
- **pop()**: Remove the top element and returns it
- **peek()**: Examine the top element without removing it
- **size():** how many items are in the stack?
- **isEmpty():** true if there are 1 or more items in stack, false otherwise

Implementing a Stack with an Array

Stack ADT

state

Set of ordered items
Number of items

behavior

push(item) add item to top
pop() return and remove item at top
peek() look at item at top
size() count of items
isEmpty() count of items is 0?

ArrayStack<E>

state

```
data[]
size
```

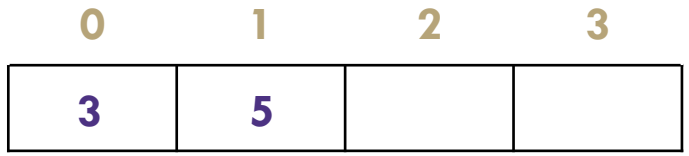
behavior

```
push data[size] = value, if out of room grow data
pop return data[size - 1], size-1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

Big O Analysis

`pop()`	O(1) Constant
`peek()`	O(1) Constant
`size()`	O(1) Constant
`isEmpty()`	O(1) Constant
`push()`	O(1) Constant or worst case O(N) linear

```
push(3)
push(4)
pop()
push(5)
```



`numberOfItems =` 2

Take 1 min to respond to activity

www.pollev.com/cse373activity
What do you think the worst case runtime of the "push()" operation will be?

Implementing a Stack with Nodes

Stack ADT

state

Set of ordered items
Number of items

behavior

push(item) add item to top
pop() return and remove item at top
peek() look at item at top
size() count of items
isEmpty() count of items is 0?

LinkedStack<E>

state

```
Node top
size
```

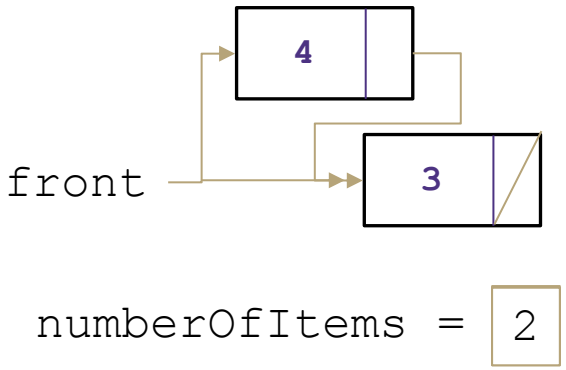
behavior

```
push add new node at top
pop return and remove node at top
peek return node at top
size return size
isEmpty return size == 0
```

Big O Analysis

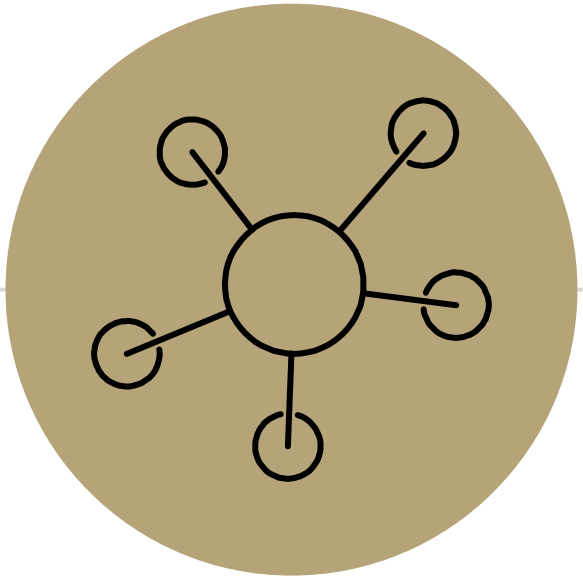
`pop()`	O(1) Constant
`peek()`	O(1) Constant
`size()`	O(1) Constant
`isEmpty()`	O(1) Constant
`push()`	O(1) Constant

```
push(3)
push(4)
pop()
```



Take 1 min to respond to activity

www.pollev.com/cse373activity
What do you think the worst case runtime of the "push()" operation will be?



Question Break

Review: What is a Queue?

queue: Retrieves elements in the order they were added.

- First-In, First-Out ("FIFO")
- Elements are stored in order of insertion but don't have indexes.
- Client can only add to the end of the queue, and can only examine/remove the front of the queue.

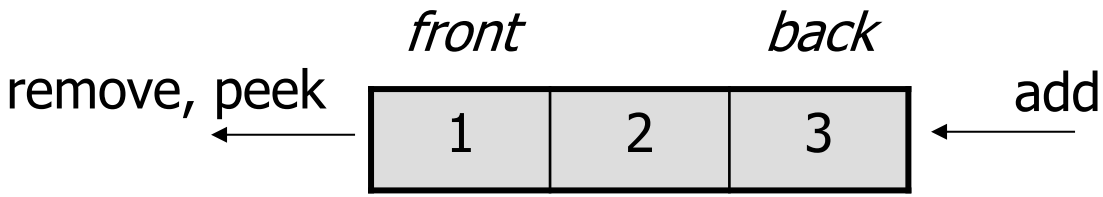
Queue ADT

state

Set of ordered items
Number of items

behavior

add(item) add item to back
remove() remove and return item at front
peek() return item at front
size() count of items
isEmpty() count of items is 0?



supported operations:

- **add(item):** aka "enqueue" add an element to the back.
- **remove():** aka "dequeue" Remove the front element and return.
- **peek()**: Examine the front element without removing it.
- **size():** how many items are stored in the queue?
- **isEmpty():** if 1 or more items in the queue returns true, false otherwise

Implementing a Queue with an Array

Queue ADT

state

Set of ordered items
Number of items

behavior

add(item) add item to back
remove() remove and return item at front
peek() return item at front
size() count of items
isEmpty() count of items is 0?

ArrayQueue<E>

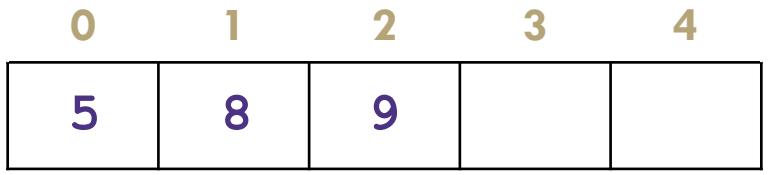
state

```
data[]
Size
front index
back index
```

behavior

```
add - data[size] = value, if out of room grow data
remove - return data[size - 1], size-1
peek - return data[size - 1]
size - return size
isEmpty - return size == 0
```

```
add(5)
add(8)
add(9)
remove()
```



numberOfItems = 3
front = 1
back = 2

Big O Analysis

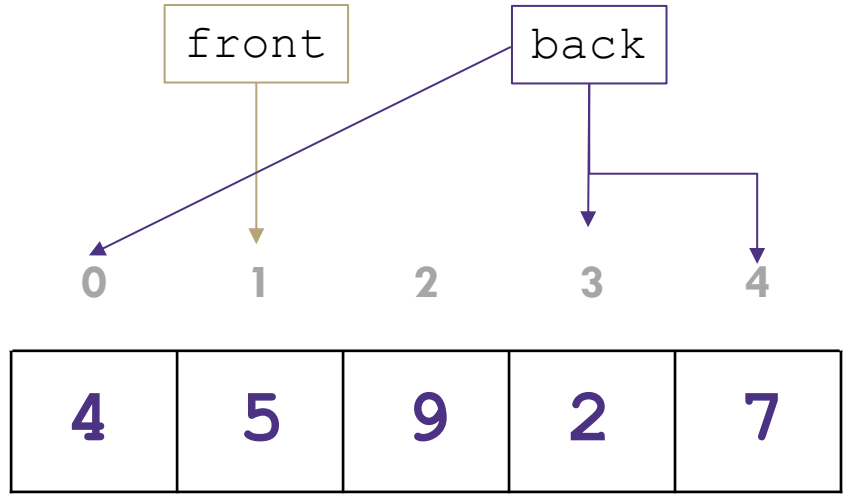
`remove()`	O(1) Constant
`peek()`	O(1) Constant
`size()`	O(1) Constant
`isEmpty()`	O(1) Constant
`add()`	O(1) Constant or worst case O(N) linear

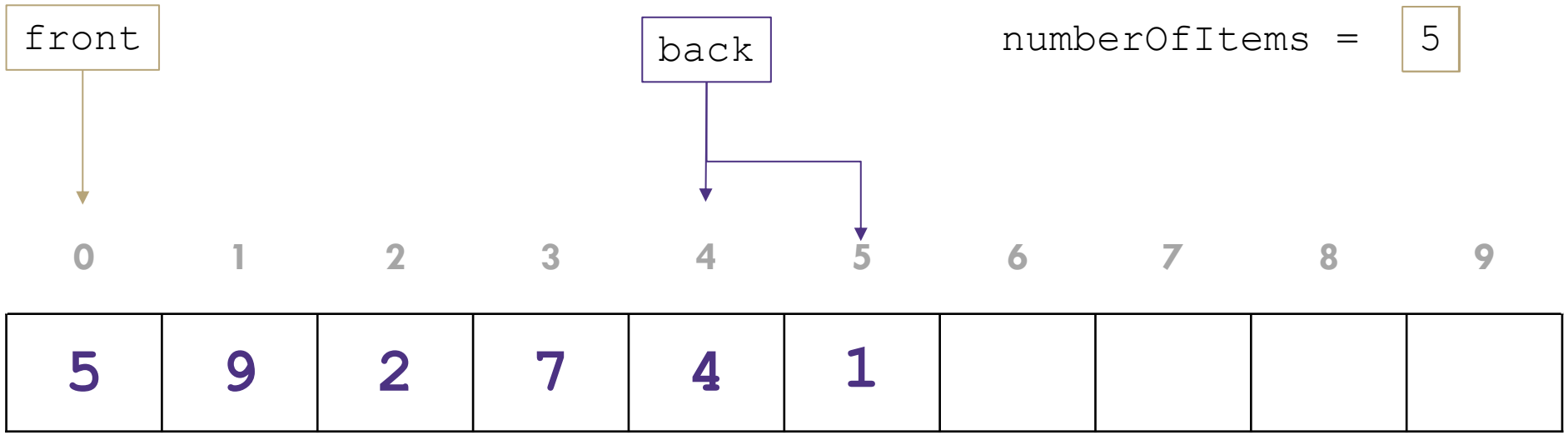
Take 1 min to respond to activity

www.pollev.com/cse373activity
What do you think the worst case runtime of the "add()" operation will be?

Implementing a Queue with an Array
> Wrapping Around

add(7)
add(4)
add(1)





Implementing a Queue with Nodes

Queue ADT

state

Set of ordered items
Number of items

behavior

add(item) add item to back
remove() remove and return item at front
peek() return item at front
size() count of items
isEmpty() count of items is 0?

LinkedQueue<E>

state

```
Node front
Node back
size
```

behavior

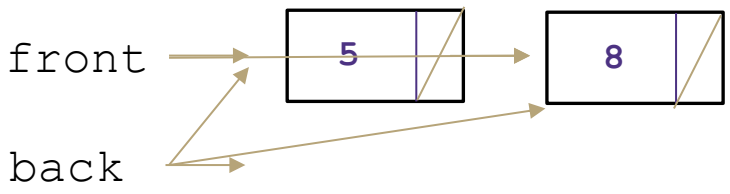
```
add – add node to back
remove – return and remove node at front
peek – return node at front
size – return size
isEmpty – return size == 0
```

Big O Analysis

`remove()`	O(1) Constant
`peek()`	O(1) Constant
`size()`	O(1) Constant
`isEmpty()`	O(1) Constant
`add()`	O(1) Constant

`numberOfItems =` 2

```
add(5)
add(8)
remove()
```



Take 1 min to respond to activity

www.pollev.com/cse373activity
What do you think the worst case runtime of the "add()" operation will be?