

CS151 Intro to Data Structures

Implementing Stacks

Queues

Warmup - what will this code print?

```
public static void main(String[] args) {  
    Stack<Integer> stack = new Stack<Integer>();  
  
    stack.push(10);  
    stack.push(20);  
    stack.push(30);  
  
    int popped = stack.pop();  
    System.out.println("Popped: " + popped);  
  
    int top = stack.peek();  
    System.out.println("Top: " + top);  
  
    stack.push(40);  
    System.out.println("New Top after push: " + stack.peek());  
  
    while (!stack.isEmpty()) {  
        System.out.println("Popped: " + stack.pop());  
    }  
}
```

Important Dates

HW2, Lab3 late deadline tonight at midnight
Lab4 due tonight (manual checkoff)

Lab5 released - due 2/25
HW3 released - due 2/25
Manual checkoff by me or TAs

Midterm after break
If you have testing accommodations let me know immediately

Agenda

- Stack Review
- Stack implementations
- Amortized Analysis
- Queues

Stacks - FILO

- First In Last Out
- *stack* of plates in the dining hall
- Operations:
 - push
 - pop
 - peek
 - isEmpty

Implementing a Stack with an Array

Goal: $O(1)$ operations

Let's try to code it

Our class implementation:

- fixed size array (no expansions!)
- How did we implement push?
- How did we implement pop?
- How did we implement peek?

Now let's implement stack with a linked list!

Goal: $O(1)$ operations.

What to consider:

- When we PUSH where should we insert to?
 - Front, back, middle?
- When we POP where should we remove from?
 - Reminder: Stack should be FILO

Linked List Stack Performance

Space complexity is

- $O(n)$

Runtime Complexity:

- push:

- $O(1)$

- Pop:

- $O(1)$

- Peek:

- $O(1)$

Stack Summary

- FILO wrapper around Array / Linked List
- Allows for limited data structures operations all with $O(1)$ cost
- Real world applications: call stack, browser history, postfix calculator

Amortized Analysis

<https://courses.cs.washington.edu/courses/cse373/17wi/summaries/amortized-runtime.pdf>

Amortized Analysis

average run time complexity of an operation.

Compares the total cost of a series of operations with how many of those operations happened.

Amortized Runtime Analysis

$$\frac{\text{total cost of operations}}{\text{total number of operations}}$$

Where an “operation” is the operation a client is doing through your public interface, like `insert(5)` or `pop()` or `add(3)`.

Exercise: Amortized Analysis of Array Insert

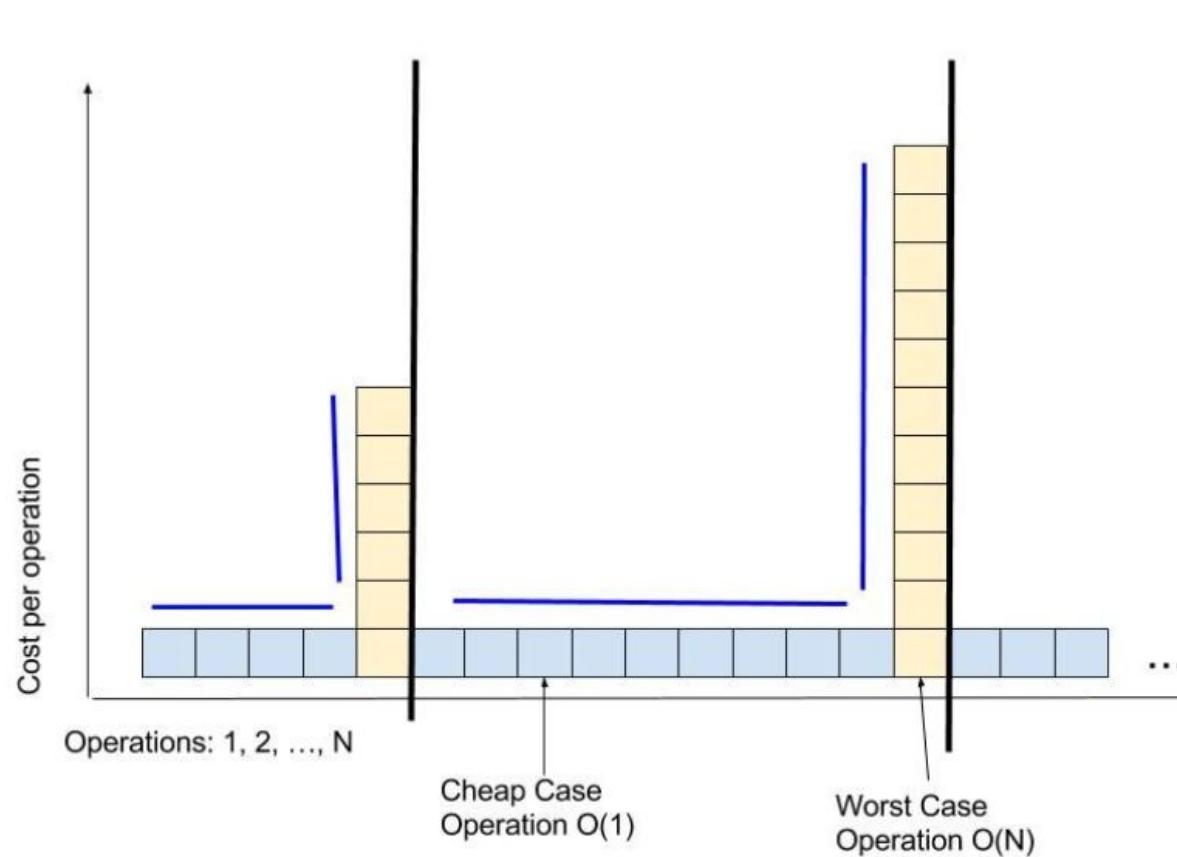
Let's say you have an expandable array with N elements: [1, 2, 3... N]

When you try to add the N+1 th element, a resize occurs.

$$\frac{\text{total cost of operations}}{\text{total number of operations}}$$

Amortized Analysis

Intuition: you want to “build up enough credit” with a series of cheap operations, so that when you have one (or more) expensive operations, you can average out the cost of the expensive one



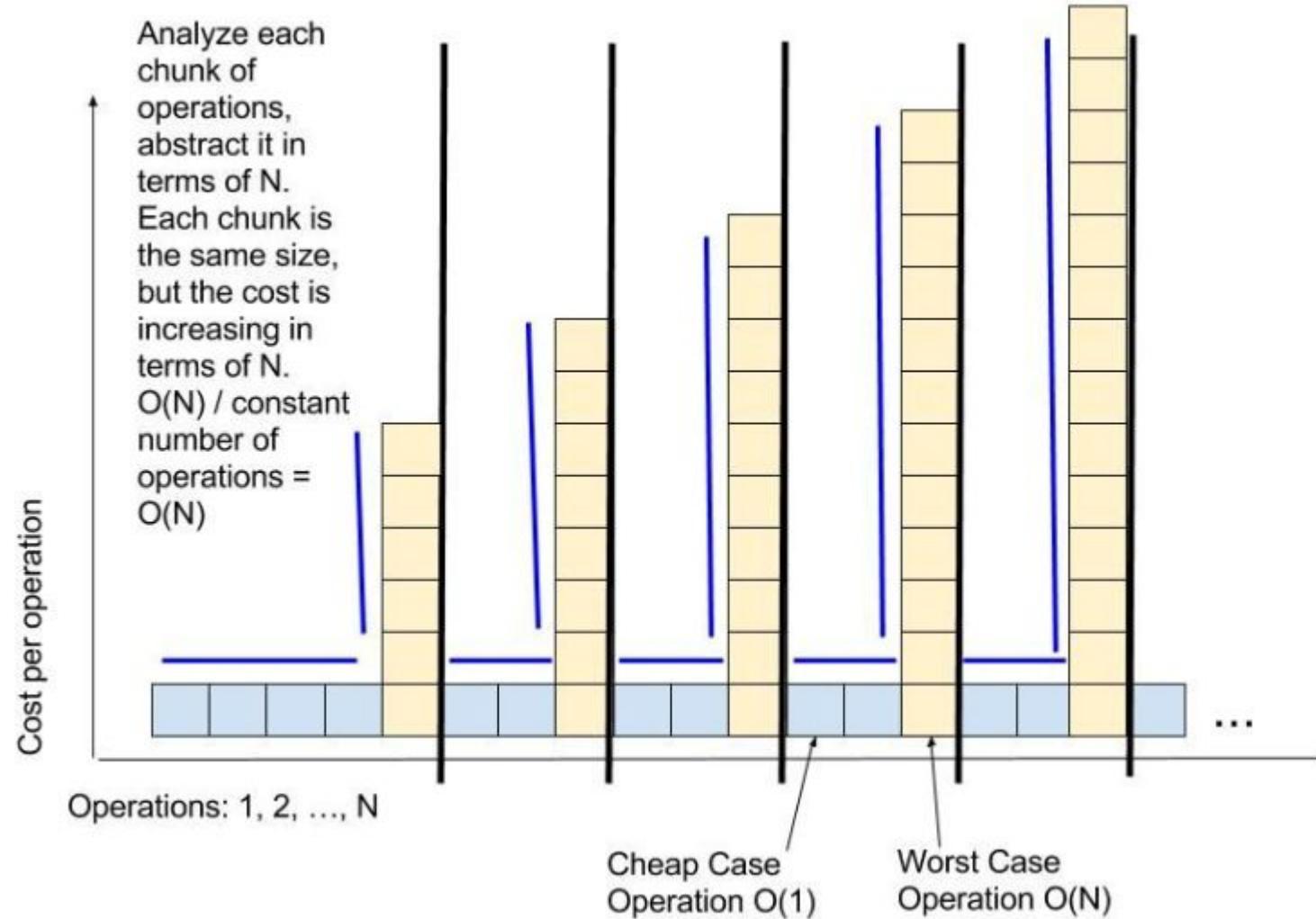
Exercise: Amortized Analysis of Array Insert

Let's say you have an expandable array **which doubles the size when it is full**

Amortized cost of insert? Calculate for 16 inserts with an expandable array that begins with capacity 2

$$\frac{\text{total cost of operations}}{\text{total number of operations}}$$

Amortized Analysis



Amortized Analysis

Average case runtime analysis

Intuition: you want to “build up enough credit” with a series of cheap operations, so that when you have one (or more) expensive operations, you can average out the cost of the expensive one

Comparing the total cost of a series of operations with how many operations happened

Queues

FIFO Stacks

Stack Property

First-in Last-out (FILO)

Where might a FILO stack not make sense?

Line for the cash register

Printer Queue

FIFO: First-in First-out

The first item in, is the first item out

Add-to the back, remove from the front

This is a **Queue**

Inserting – “enqueue”

Removing - “dequeue”

Queue Interface

```
public interface Queue<E> {  
    int size();  
    boolean isEmpty();  
    E first();  
    void enqueue(E e);  
    E dequeue();  
}
```

- **null is returned from `dequeue()` and `first()` when queue is empty**

Queue Example

Cash register code

Example

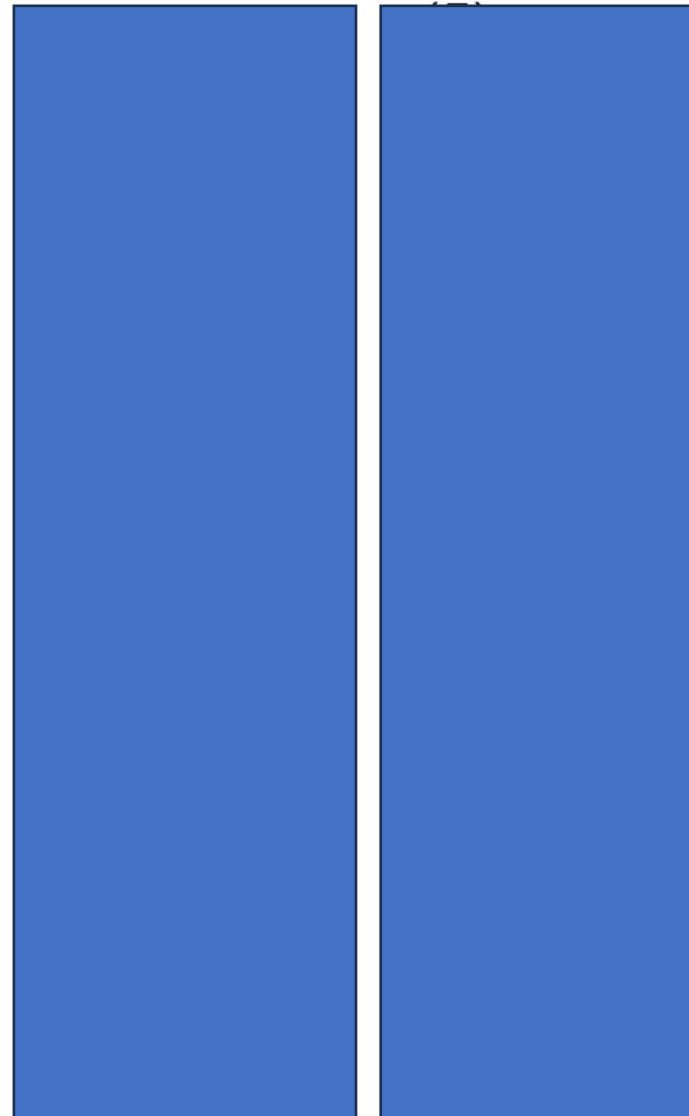
<i>Operation</i>	<i>Output</i>	<i>Q</i>
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	5	(3)
enqueue(7)	–	(3, 7)
dequeue()	3	(7)
first()	7	(7)
dequeue()	7	()
dequeue()	<i>null</i>	()
isEmpty()	<i>true</i>	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	2	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	9	

Example

Operation
enqueue(5)



Output *Q*



Implementing a Queue with an Array

Goal: $O(1)$ operations

How can we achieve this?

1. Fixed size underlying array (no expansions)
1. Where should we insert?
 - a. front, back, middle?
1. Where should we remove from?
 - b. we should preserve first in first out property!

Summary

- Stacks are FILO data structures with $O(1)$ operations
 - can be implemented with an array or LL
- Queues are FIFO data structures with $O(1)$ operations
 - can be implemented with an array or LL
 - saw array today will see LL next week
- Amortized runtime analysis
 - average