Lab 05: Stacks and Queues

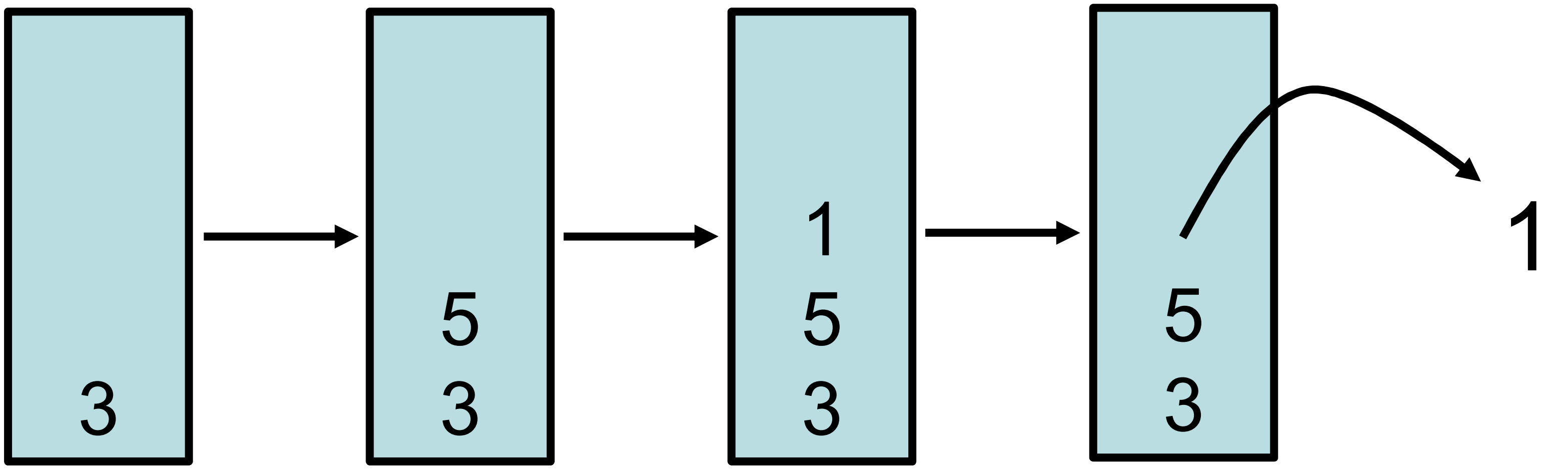
CS 0445: Data Structures

TAs: Jon Rutkauskas
Brian Nixon
http://db.cs.pitt.edu/courses/cs0445/current.term/

October 7, 2019
University of Pittsburgh, Pittsburgh, PA

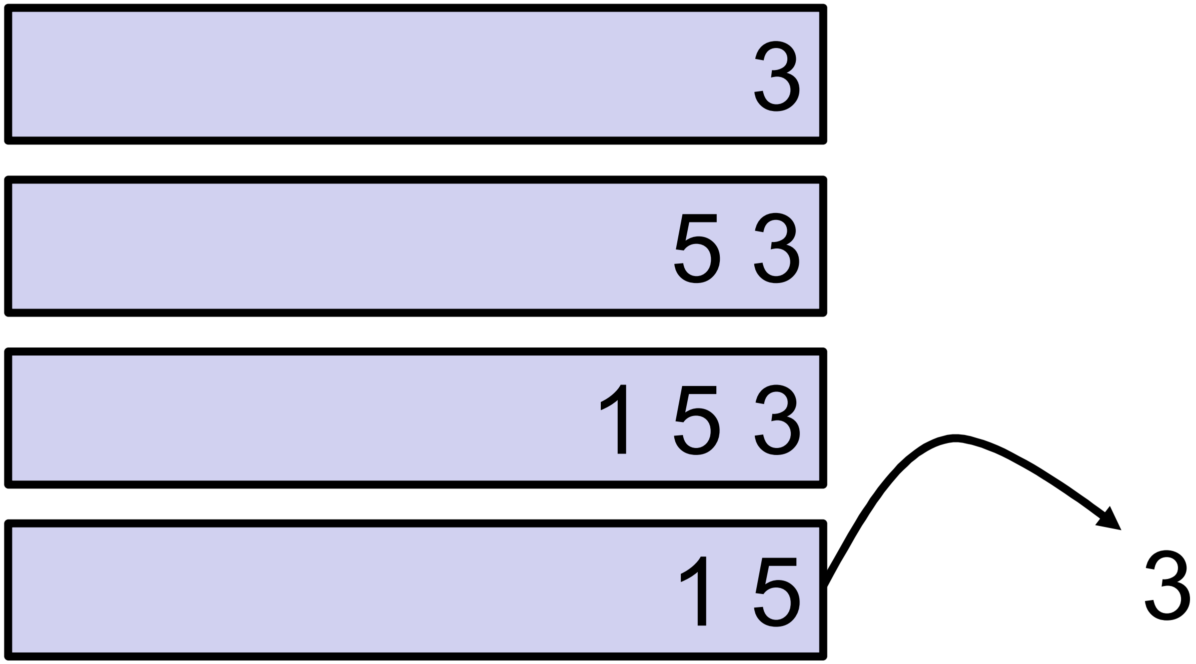
Review: Stacks vs. Queues

- Stack
  - LIFO: Last-in, First Out
  - Push to add, Pop to remove
  - E.g., push(3), push(5), push(1), pop()



Review: Stacks vs. Queues

- Queue
  - FIFO: First-in, first-out
  - Enqueue to add, dequeue to remove
  - E.g., enqueue(3), enqueue(5), enqueue(1), dequeue()



3

5 3

1 5 3

1 5

3

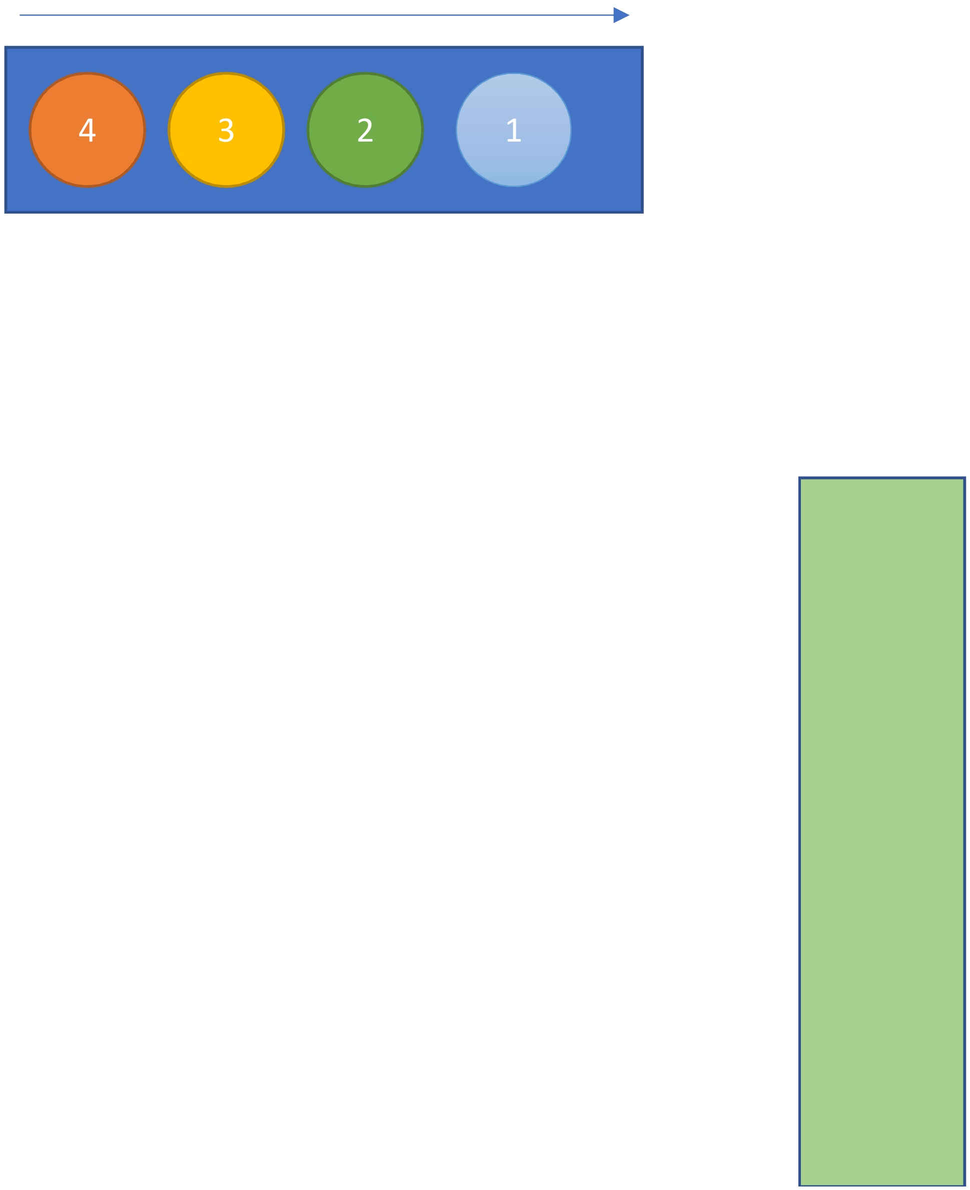
Practical use of Stacks and Queues

- Reversing a queue:
  - How can we reverse the order of the elements in a queue?
    - Iterate over the backing data structure and swap all the positions?
    - What if you don't have access to/don't know the backing data structure. E.g., you just have a Queue<T>
  - We can actually reverse a queue using a stack!

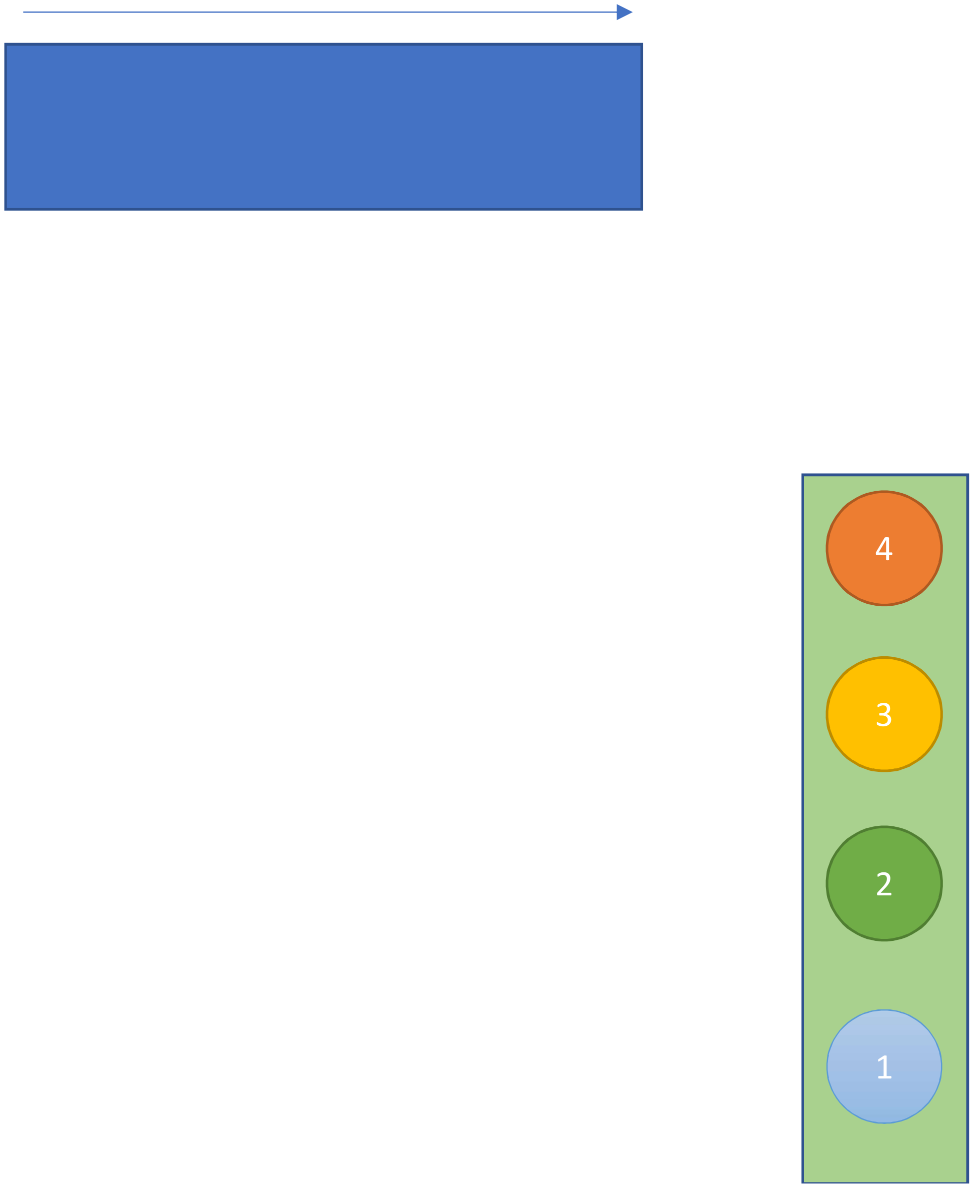
Reversing a queue using a stack

- Basic idea:
  - We take out all the elements of the queue in order, placing them each on a stack
  - Once the queue is empty, we start taking the elements off the stack and adding them back into the queue.
  - Once the stack is empty, the queue will have been filled back up with the original elements, but this time with the order reversed!
  - Much easier to visualize with an example…

Example



Example



Algorithm

- More formally, to reverse the order of the queue with a stack:
  - Make a temporary stack to hold the elements
  - While the queue is not empty:
    - Remove an element from the queue, and add it to the stack
  - While the stack is not empty:
    - Remove an element from the stack, and add it to the original queue

Next Problem

Problem

- Imagine that every day after class, you take your assignments and put them in a big stack on your desk to do later.
- You always work from the top down (LIFO) like a regular stack.
- But, you need to be aware of the deadlines so you don’t turn in an assignment late.
- You need to figure out *which item on the stack has the* ***least*** *amount of time left* so you can work down to it.

Problem

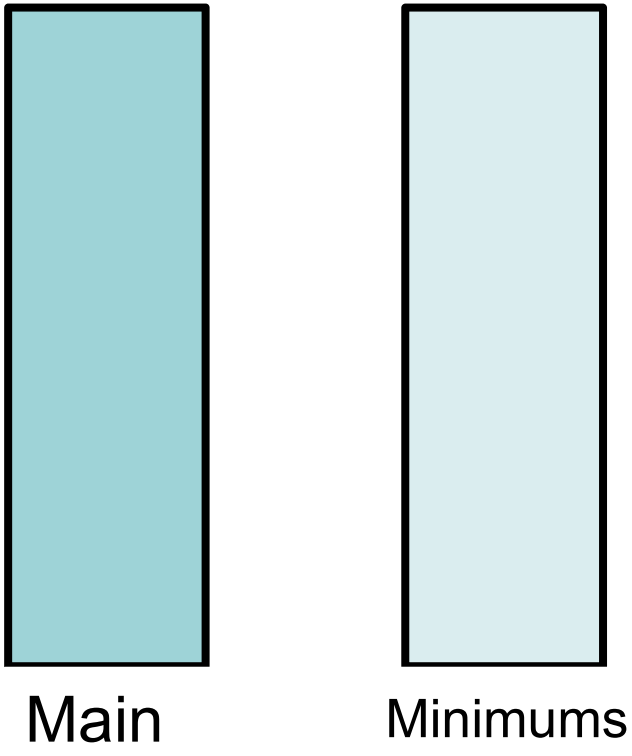
- To generalize this problem:
  - You need to find a way to make a **stack** that lets you always figure out the **minimum value** on the stack.
  - … a MinStack!

MinStack

- MinStack – a stack that can tell you the lowest-value item on it… a getMin() function
- How to implement?
  - Idea: Iterate through the items on the stack on a getMin() call, search for the lowest item.
  - Pros: Very easy to implement and understand. Doesn’t require taking up any extra memory to store minimums anywhere
  - Cons: Runtime?
    - O(n) worst-case and average-case

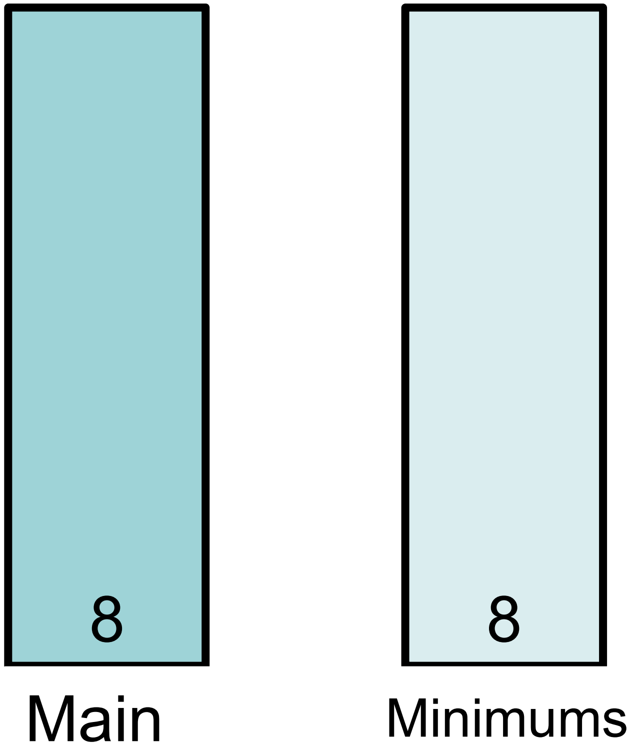
Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, 1, 5



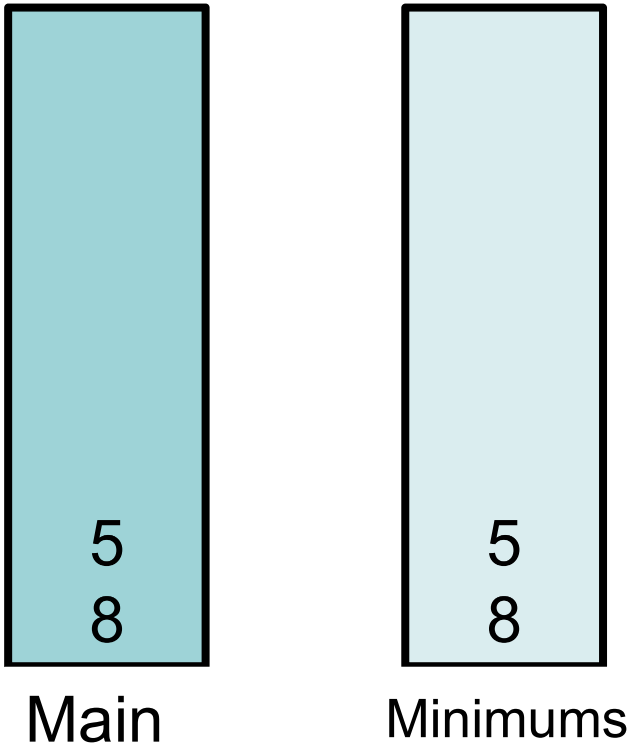
Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: **8**, 5, 2, 6, 1, 3, 1, 5



Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, **5**, 2, 6, 1, 3, 1, 5



5
8

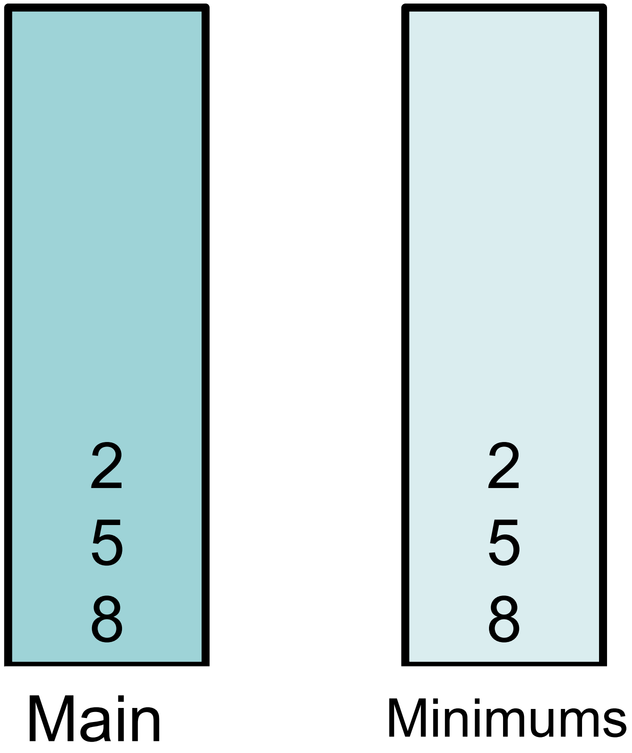
Main

5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, **2**, 6, 1, 3, 1, 5



2
5
8

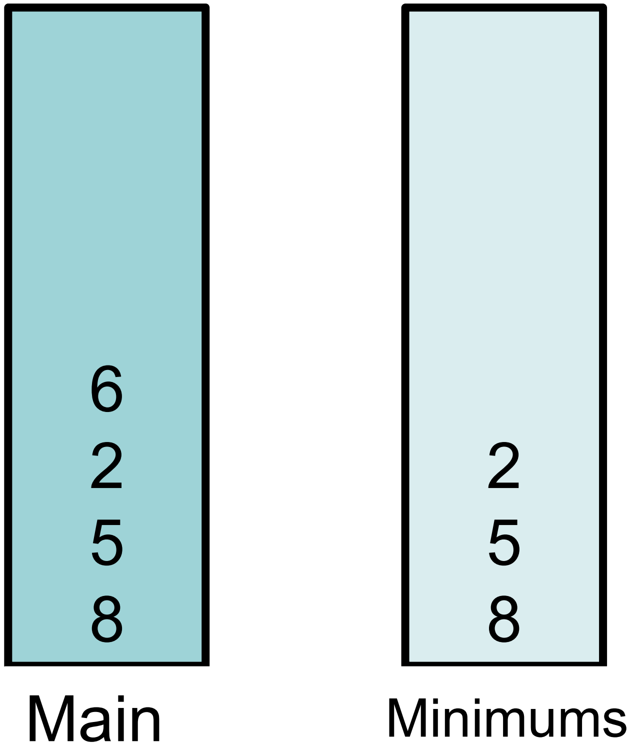
Main

2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, **6**, 1, 3, 1, 5



6
2
5
8

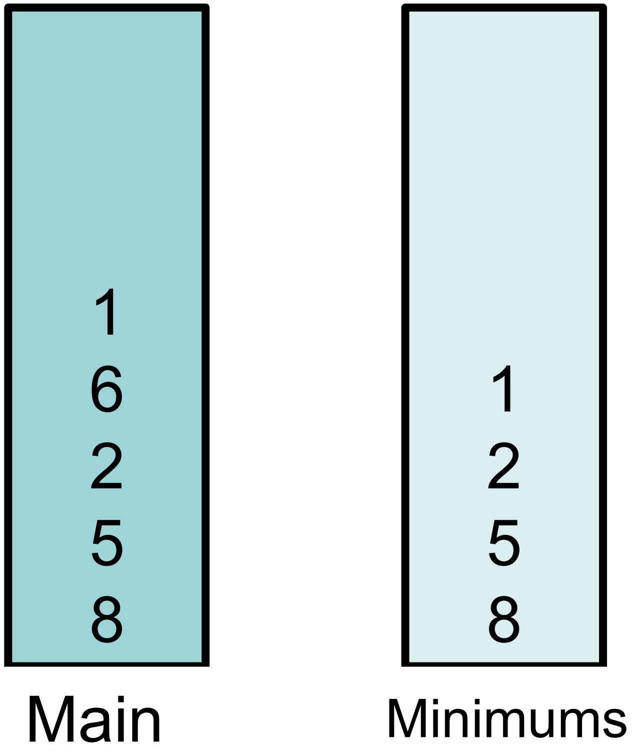
Main

2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, **1**, 3, 1, 5



1
6
2
5
8

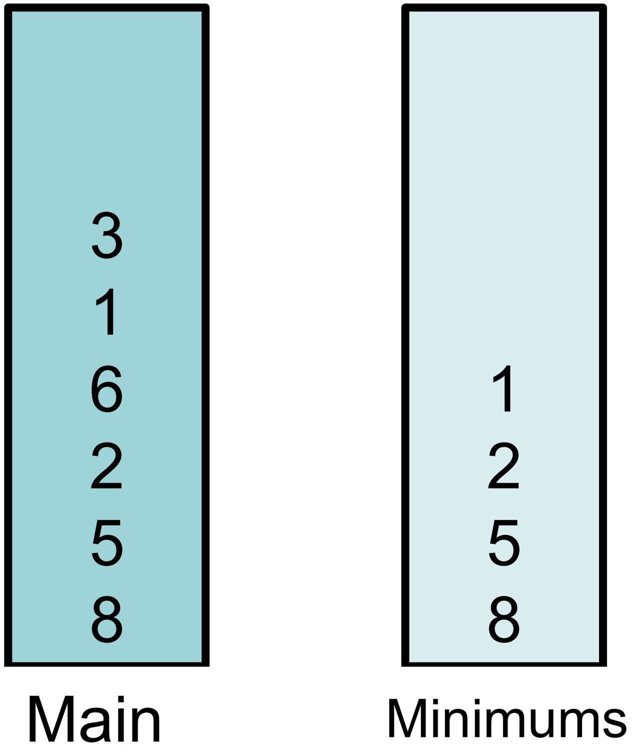
Main

1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, **3**, 1, 5



3
1
6
2
5
8

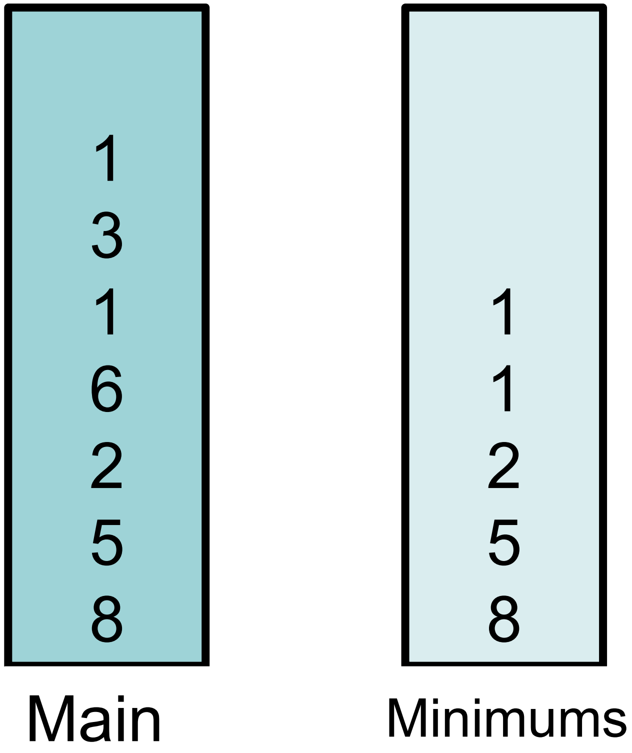
Main

1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, **1**, 5



1
3
1
6
2
5
8

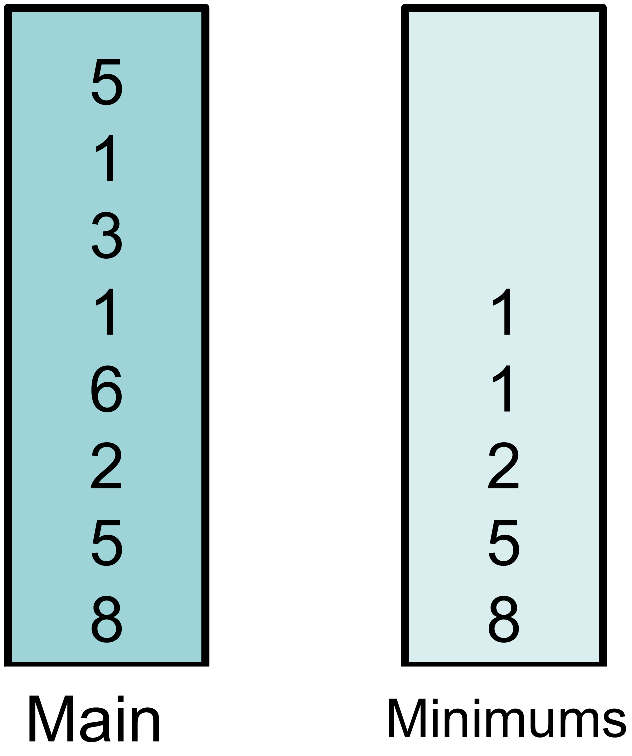
Main

1
1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, 1, **5**



5
1
3
1
6
2
5
8

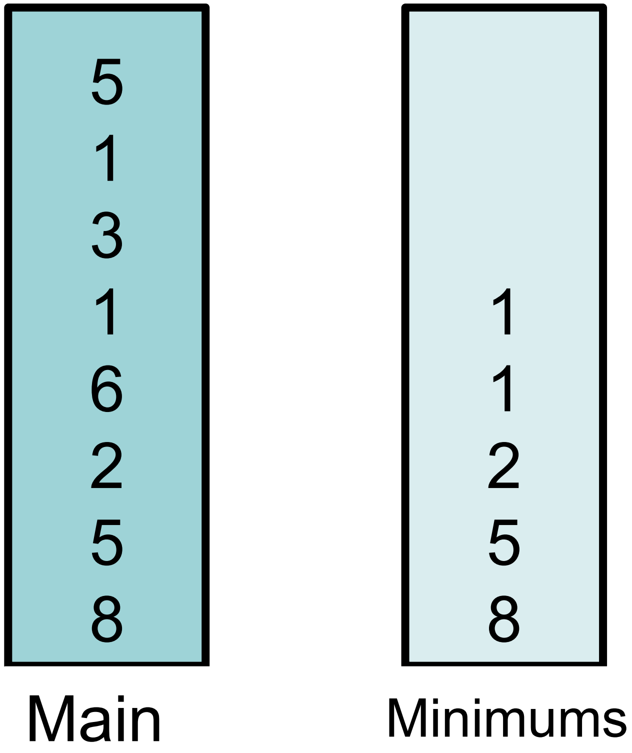
Main

1
1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, 1, 5, then **remove**



5
1
3
1
6
2
5
8

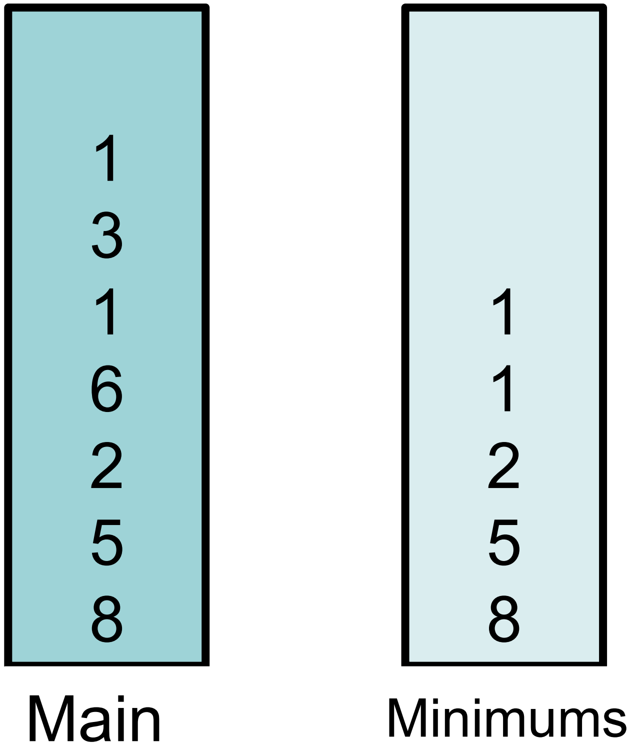
Main

1
1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, 1, **5**, then remove



1
3
1
6
2
5
8

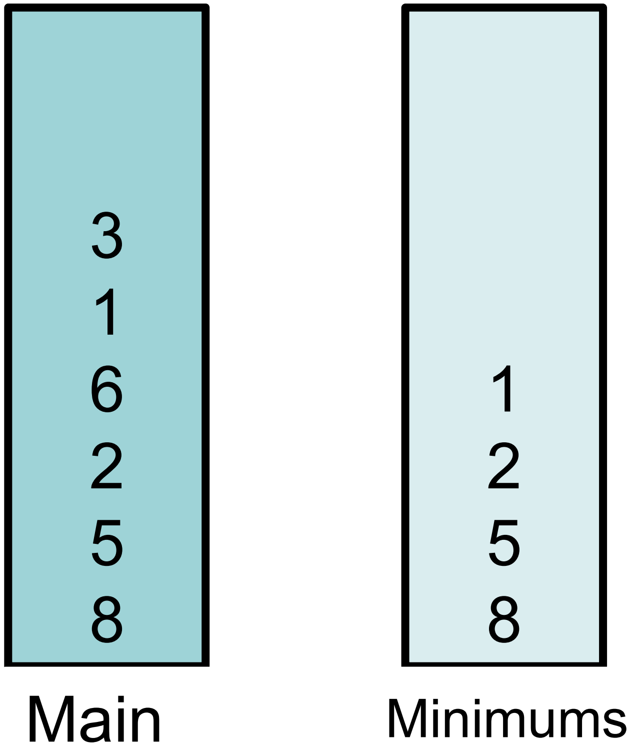
Main

1
1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, 3, **1**, 5, then remove



3
1
6
2
5
8

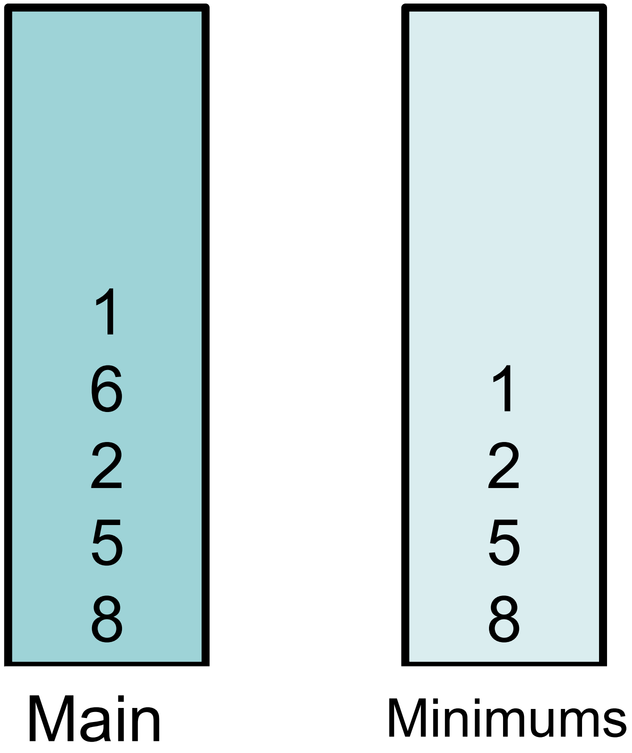
Main

1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, 1, **3**, 1, 5, then remove



1
6
2
5
8

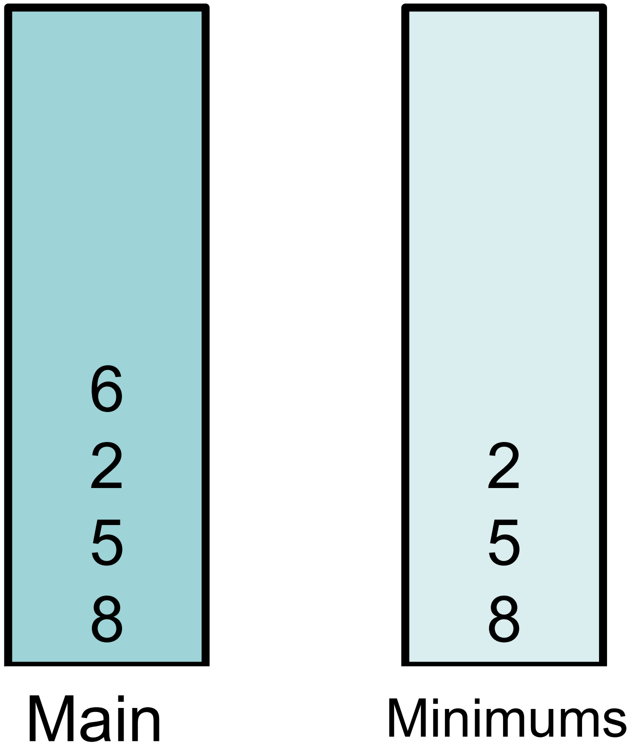
Main

1
2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, 6, **1**, 3, 1, 5, then remove



6
2
5
8

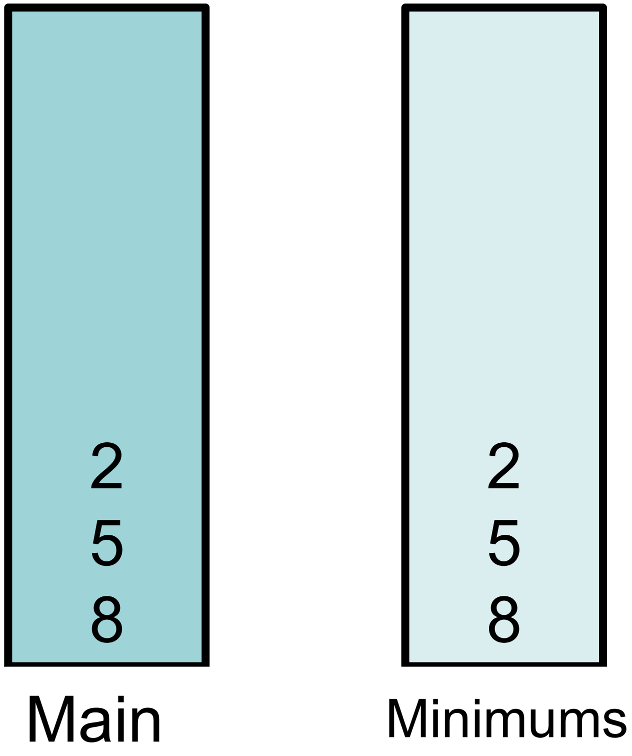
Main

2
5
8

Minimums

Another Implementation

- Use another stack for minimums (like a history of minimums)
  - – Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, 2, **6**, 1, 3, 1, 5, then remove



2
5
8

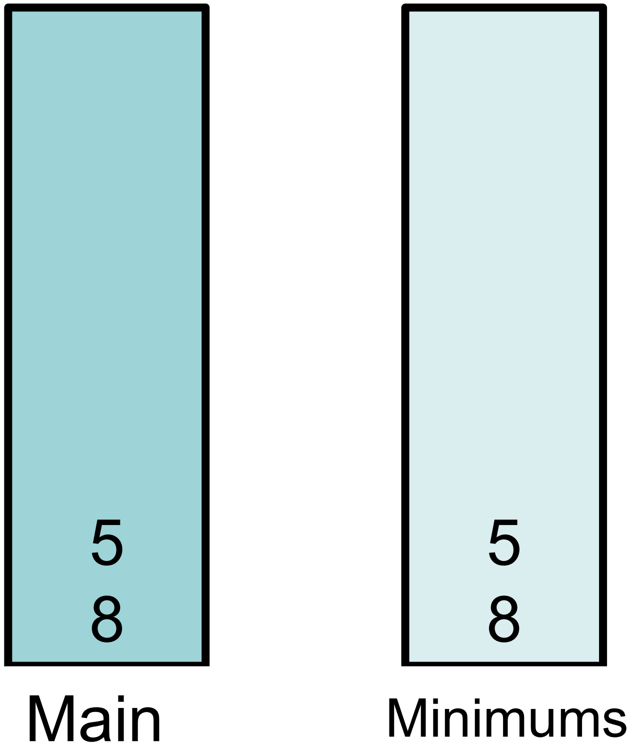
Main

2
5
8

Minimums

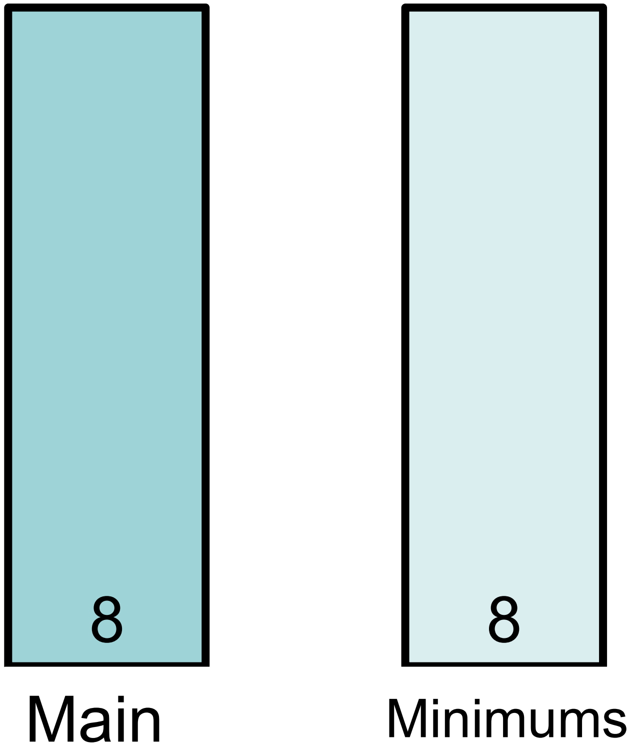
Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there’s a new minimum, remove when that value has been removed from the main stack
- Example: 8, 5, **2**, 6, 1, 3, 1, 5, then remove



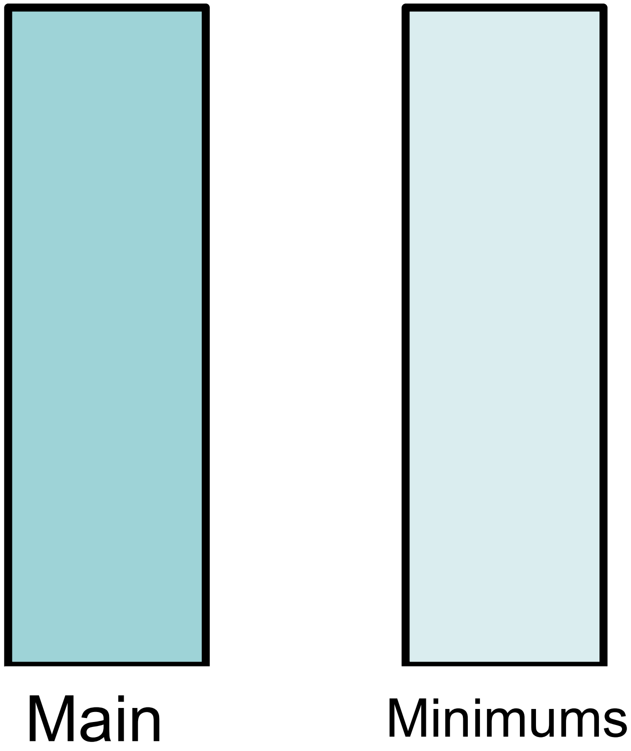
Another Implementation

- Use another stack for minimums (like a history of minimums)
  - Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: 8, **5**, 2, 6, 1, 3, 1, 5, then remove



Another Implementation

- Use another stack for minimums (like a history of minimums)
  – Add to the stack when there's a new minimum, remove when that value has been removed from the main stack
- Example: **8**, 5, 2, 6, 1, 3, 1, 5, then remove



Dual-Stack Implementation

- More formally:
  - Create a second stack to store the history of minimums.
  - On push:
    - Push to the minimums stack if the new value is less than or equal to the current minimum.
    - Push to the main stack.
  - On pop:
    - Pop from the main stack.
    - Pop from the minimums stack if the value popped from the main stack is equal to the value on top of the minimums stack.
  - To getMin():
    - Just return the value on the top of the minimums stack

Dual-Stack Implementation

- Analysis:
  - Runtime?
  - Just returning the value on the top of a stack, plus pushing and popping.
  - O(1) each time
  - Memory?
    - Worst case: have to push to the stack every time – O(n)
  - What order would the values have to be in to be O(n)?
  - What order would the values have to be in to be O(1)?

Implementation Notes

- To best make use of a minStack, we should make sure we can use one anywhere we already use a stack
  - Extend a stack we already implemented, no need to re-implement a stack.
- Also need to work with generic types, not just integers.
  - But how to tell which are bigger than each other?

interface Comparable<T>

- An interface that makes sure we can compare objects,
  - since we can't just write `if(oneObject < otherObject)`; the <, >, == operators don't work.
- Compares objects with the compareTo(object) method
  - Returns a number less than, equal to, or greater than 0
  - a.compareTo(b):
    - a < b → `a.compareTo(b) < 0`
    - a == b → `a.compareTo(b) == 0`
    - a > b → `a.compareTo(b) > 0`

Using super

- When extending a class, you can access the methods and data of the class you extended by using `super`
- E.g.,
  - `super.methodName() –` Calls methodName on the extended object.

Your Tasks

- Download the code from the course website
  - http://db.cs.pitt.edu/courses/cs0445/current.term/
- First, implement the reverseQueue method in QueueReverser.java
  - Fill in the TODOs
- Then, complete the MinStack class in MinStack.java
  - Fill in the TODOs
- Test your code using Lab5Tester
  - Consider adding additional tests
- Ask for help if you get stuck