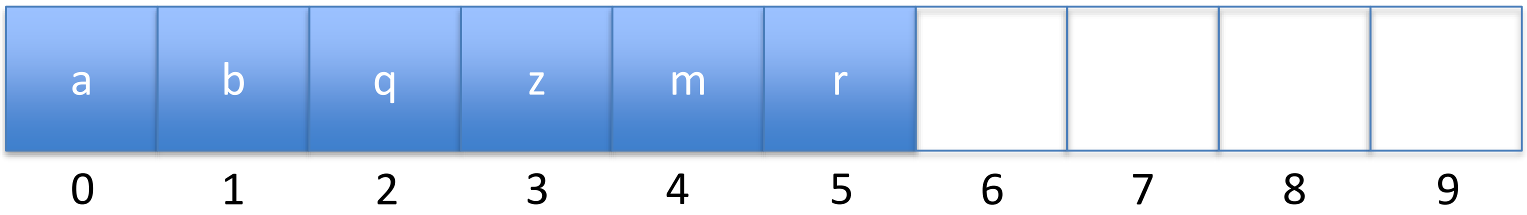
Array-Based Data Structures

COMP2402

Carleton University

Fall 2017

ArrayStack



- List interface, implemented with an array
- Similar to ArrayList from JCF
- Efficient only for stack operations

- Reading:
  - ODS Section 2.1, 2.2

Stack Interface

- push(x)
  - add item x to the top of the stack
- pop()
  - remove/return top item from stack
- size()
  - number of items in stack
- peek()
  - observe top item on stack

Stacks vs. Lists

Stack	List
push(x)	add(n,x)
pop()	remove(n-1)
size()	size()
peek()	get(n-1)

List Interface

- get(i)/set(i,x)
  - Access element *i*, and return/replace it
- size()
  - number of items in list
- add(i,x)
  - insert new item x at position *i*
- remove(i)
  - remove the element from position *i*

ArrayStack

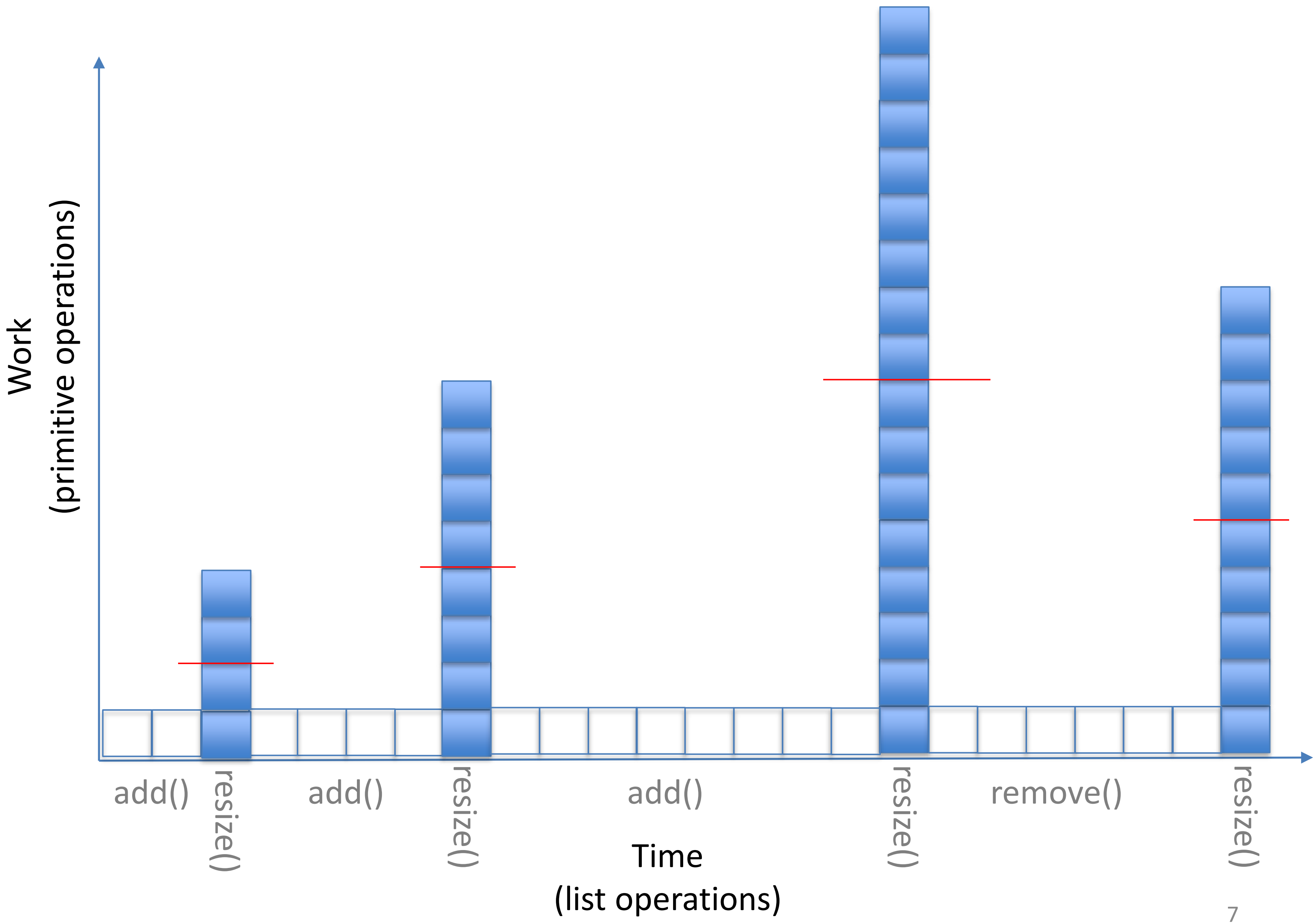
Theorem:

An **ArrayStack** implements the **List** interface. **Ignoring the cost of resize()**, an ArrayStack supports the operations:

- `get(i)` and `set(i,x)` in $O(1)$ time per operation,
- `add(i,x)` and `remove(i)` in $O(1+n-i)$ time per operation.

But can we really just ignore the cost of resize()?

Amortized Cost



Amortized Cost

Lemma:

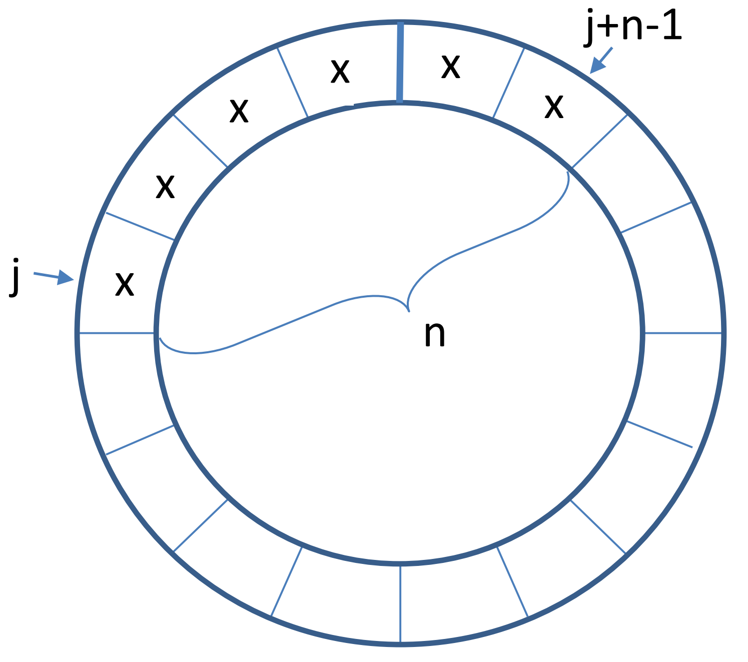
If an ArrayStack is created and any sequence of $\boldsymbol{m} \geq 1$ calls to `add(i,x)` or `remove(i)` are performed, then the **total time spent** during the calls to resize() is $\mathbf{O}(\boldsymbol{m})$.

- For *m* `add/remove` operations, `resize()` will copy at most *2m* elements.
- The amortized cost of resize(), over m calls to add/remove is then: $\frac{2m}{m} = O(1)$

ArrayQueue & ArrayDeque

- Implement the Queue and List interfaces using arrays

- Efficient operations at both ends



- Reading:
  - ODS Section 2.3, 2.4

Queue & Deque Interfaces

- Queue:
  - add(x)/remove(): add to one end, remove from the other

- Deque:
  - addFront(x), removeFront(x), addBack(x), removeBack(x): add/remove from either end

ArrayQueue

Theorem:

An **ArrayQueue** implements the **(FIFO) Queue** interface. Ignoring the cost of resize(), an ArrayQueue supports the operations:

- **add(x)** and **remove()** in $\mathbf{O(1)}$ time per operation.

In addition, starting with an empty ArrayQueue, any sequence of $m \geq 1$ add/remove operations results in $O(m)$ time spent on calls to resize().

ArrayDeque

Theorem:

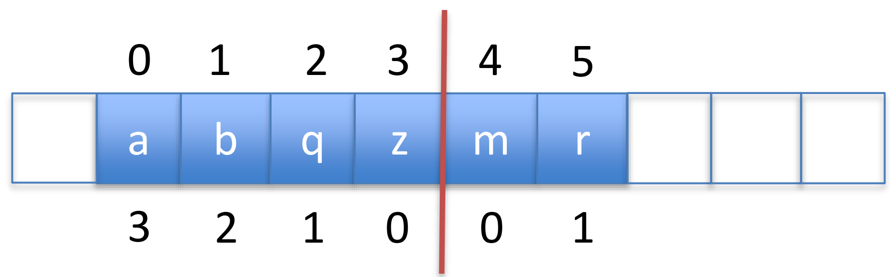
An **ArrayDeque** implements the **List** interface. Ignoring the cost of resize(), an ArrayDeque supports the operations:

- **get(i)**/**set(i,x)** in $\mathbf{O(1)}$ time per operation, and
- **add(i,x)**/**remove(i)** in $\boldsymbol{O(1+\min\{i, n-i\})}$ time per operation

In addition, starting with an empty ArrayDeque, any sequence of $m \geq 1$ add/remove operations results in $O(m)$ time spent on calls to resize()

DualArrayDeque

- Implements a Deque with two ArrayStacks



- Example of using known data structures as building blocks for other data structures!

- Reading:
  - ODS Section 2.5

DualArrayDeque

Theorem:

A **DualArrayDeque** implements the **List** interface. Ignoring the cost of resize() and rebalance(), a DualArrayDeque supports the operations:

- **get(i)**/**set(i,x)** in $\mathbf{O(1)}$ time per operation, and
- **add(i,x)**/**remove(i)** in $\mathbf{O(1+\min\{\mathit{i},\mathit{n}-\mathit{i}\})}$ time per operation

In addition, starting with an empty DualArrayDeque, any sequence of $m \geq 1$ add/remove operations results in $O(m)$ time spent on calls to resize() and rebalance().

Potential Method

Define a **potential function** for the data structure to be the absolute difference of the sizes of the two stacks

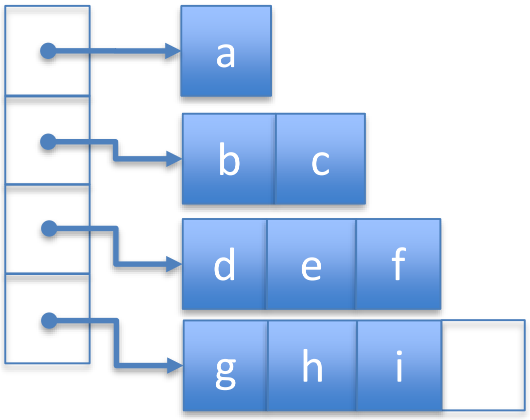
$$\Phi = |f - b|$$

Any add/remove operation can only increase the potential by at most 1. (Other operations do not affect Φ)

Minimum number of add/remove calls between two states is at least $|\Phi(s_1) - \Phi(s_0)|$

RootishArrayStack

- Implements the list interface using multiple backing arrays



- At most $O(\sqrt{n})$ unused array locationses!

- Reading:
  – ODS Section 2.6

RootishArrayStack

Theorem:

A **RootishArrayStack** implements the **List** interface. Ignoring the cost of resize() and rebalance(), a RootishArrayStack supports the operations:

- **get(i)/set(i,x)** in $\mathbf{O(1)}$ time per operation, and
- **add(i,x)/remove(i)** in $\boldsymbol{O(1+n-i)}$ time per operation

In addition, starting with an empty RootishArrayStack, any sequence of $m \geq 1$ add/remove operations results in $O(m)$ time spent on calls to grow() and shrink().

The wasted space in a RootishArrayStack that stores n elements is $O(\sqrt{n})$