

Positional Collection (Chapter 9 pages 107-119)

position $0 \quad 1 \quad \dots \quad n-1$

array-based
implementation

+ constant-time
access to position p

must shift elements

- to add or remove near
middle (linear time)

list-based
implementation

- must traverse list
to reach position p
linear time to get to
middle

+ constant time to
add or remove
once element located

array-based

- Some space (and time) overhead to maintain a tracker

+ space usage can be as small as roughly n references when n elements in collection

- must select a size when allocating (you can resize)

list-based

+ negligible overhead to track an element

- even in a singly-linked list use 3 refs per node (element, next, type)

Roughly $3n$ references to hold same n elements

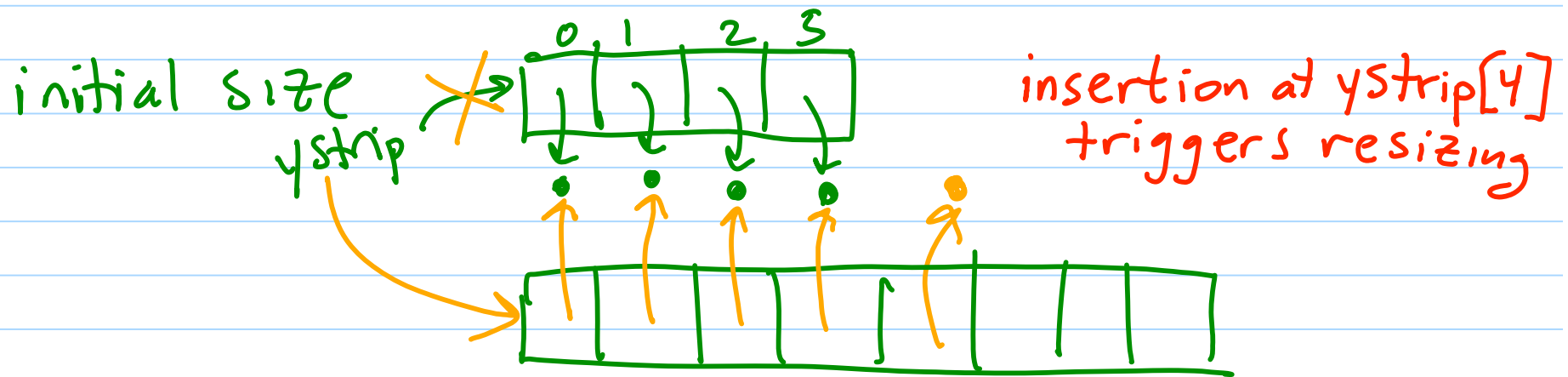
+ naturally can dynamically resize

Positional Collection Data Structures

- Array position p stored in $a[p]$

- Dynamic Array

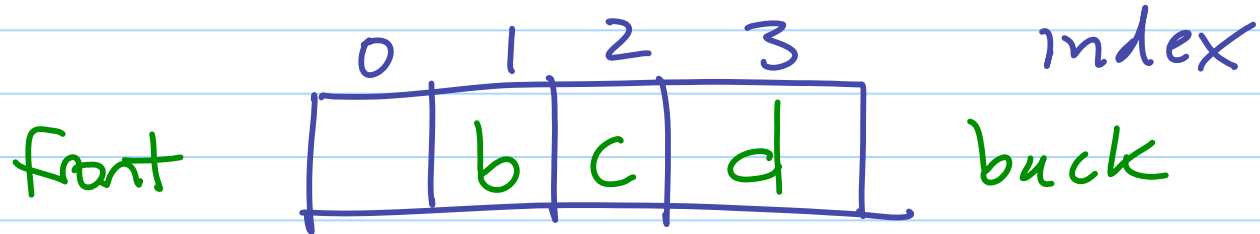
(This is what a Java arraylist is!)



changes when doubling
array size

Circular Array

Queue with at most 4 elements

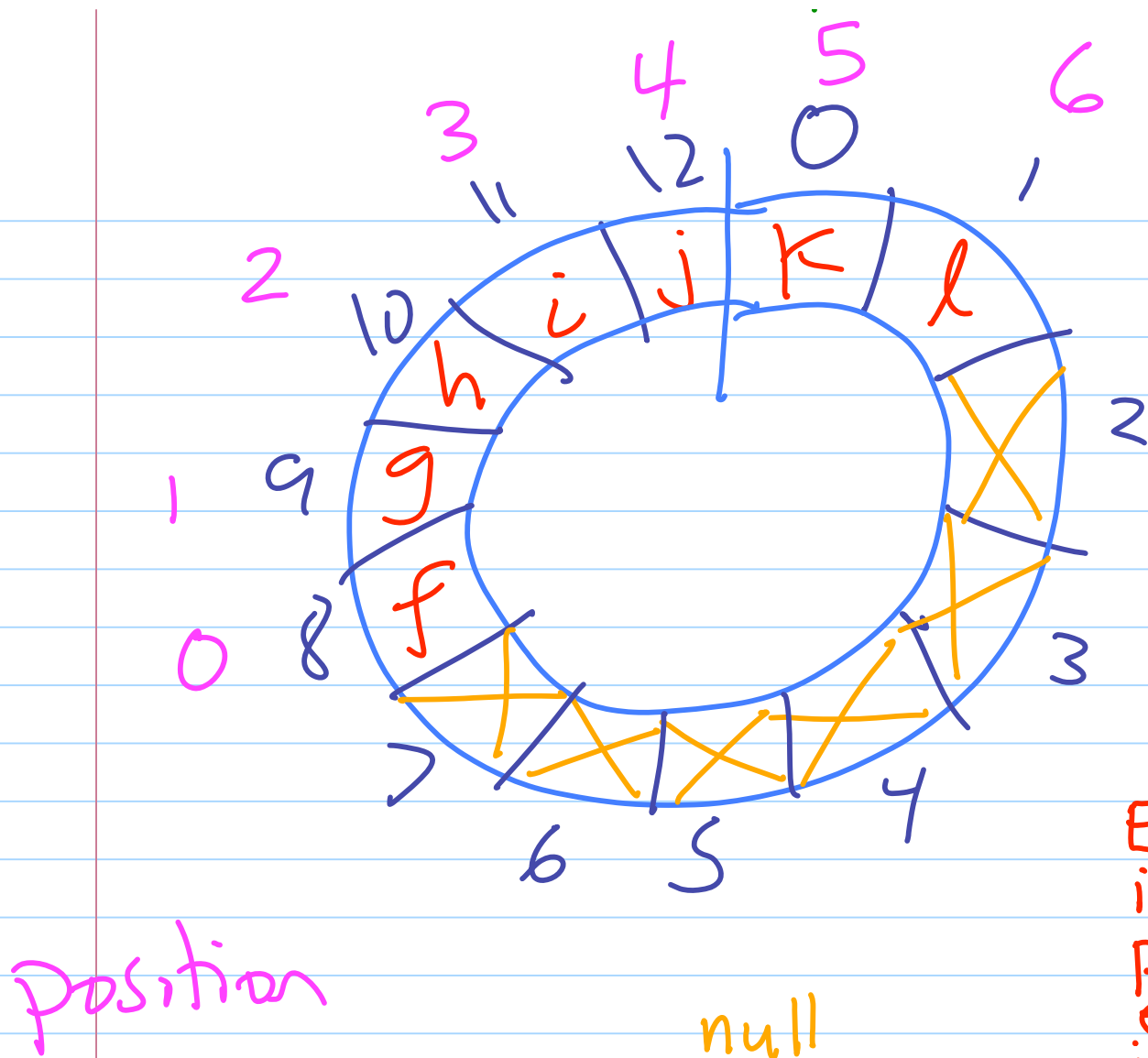


Offset of 1, let variable start = 1 position to mark index for position 0

dequeue (to remove a)

Translation between index + position computed by:

$$\begin{aligned} \text{index} &= \text{position} + \text{start} \\ \text{position} &= \text{index} - \text{start} \end{aligned} \quad \left. \vphantom{\begin{aligned} \text{index} &= \text{position} + \text{start} \\ \text{position} &= \text{index} - \text{start} \end{aligned}} \right\} \begin{array}{l} \text{mod} \\ \text{array} \\ \text{size} \end{array}$$



index

offset into
underlying array for
position 0

start = 8

size = 7

Queue: f, g, h, i, j, k, l

Each dequeue
implicitly changes
positions of remaining
elements by
incrementing start

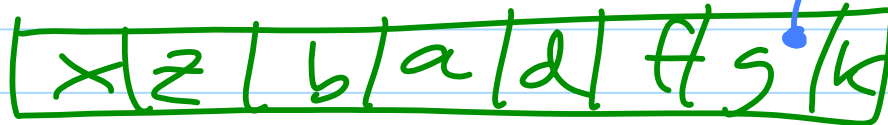
Array

Dynamic Array

Circular Array

Dynamic Circular Array

Tracked Array



index

application program must maintain a reference to the tracker returned by addTracked

Singly Linked List
Doubly Linked List

Key

- ⊙ Excellent
- Good
- Fair
- Not required

For time

$O(1)$
 $\sim O(\min(p+1, n-p))$
 $O(n)$

	Array	Circular Array	Dynamic Array	Dynamic Circular Array	Tracked Array	Singly Linked List	Doubly Linked List
access by position near middle	⊙	⊙	⊙	⊙	⊙	○	○
adding to middle portion (once located)	○	○	○	○	○	⊙	⊙
removing from middle portion (once located)	○	○	○	○	○	○	⊙
access to front	⊙	⊙	⊙	⊙	⊙	⊙	⊙
adding to front	●	⊙	●	⊙	⊙	⊙	⊙
removing at front	●	⊙	●	⊙	⊙	⊙	⊙
access to back	⊙	⊙	⊙	⊙	⊙	⊙	⊙
adding to back	⊙	⊙	⊙	⊙	⊙	⊙	⊙
removing from back	⊙	⊙	⊙	⊙	⊙	●	⊙
determining the position of a locator	⊙	⊙	⊙	⊙	⊙	●	○
support of a tracker					✓	✓	✓
space usage	⊙	⊙	⊙	⊙	○	○	●
ease in adjusting the capacity	●	●	●	●	●	⊙	⊙
automatic resizing			✓	✓	✓	•	•

Table 9.1 in
A Practical
 Guide to Data
 Structures &
 Algorithms in
 Java

Quicksort

Note Title

9/19/2007

First let's review two things from last class.

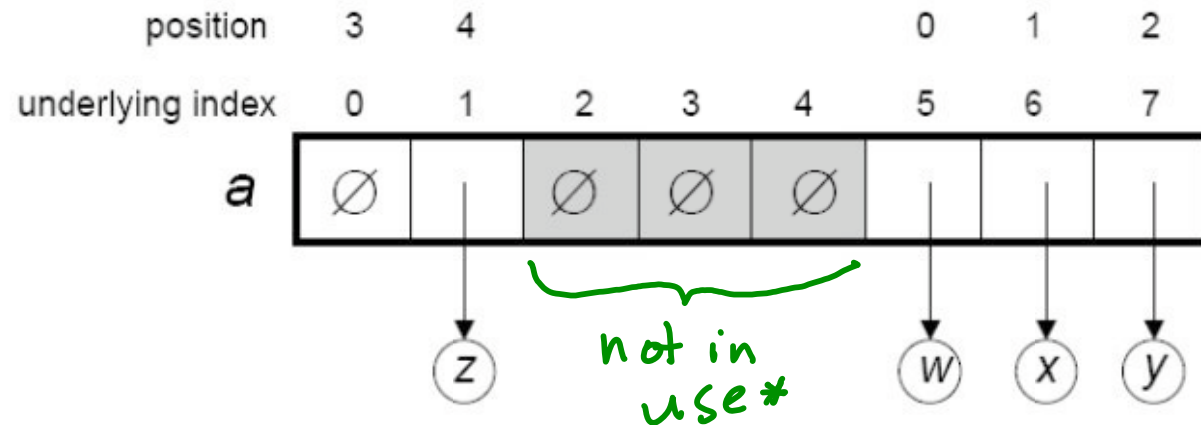


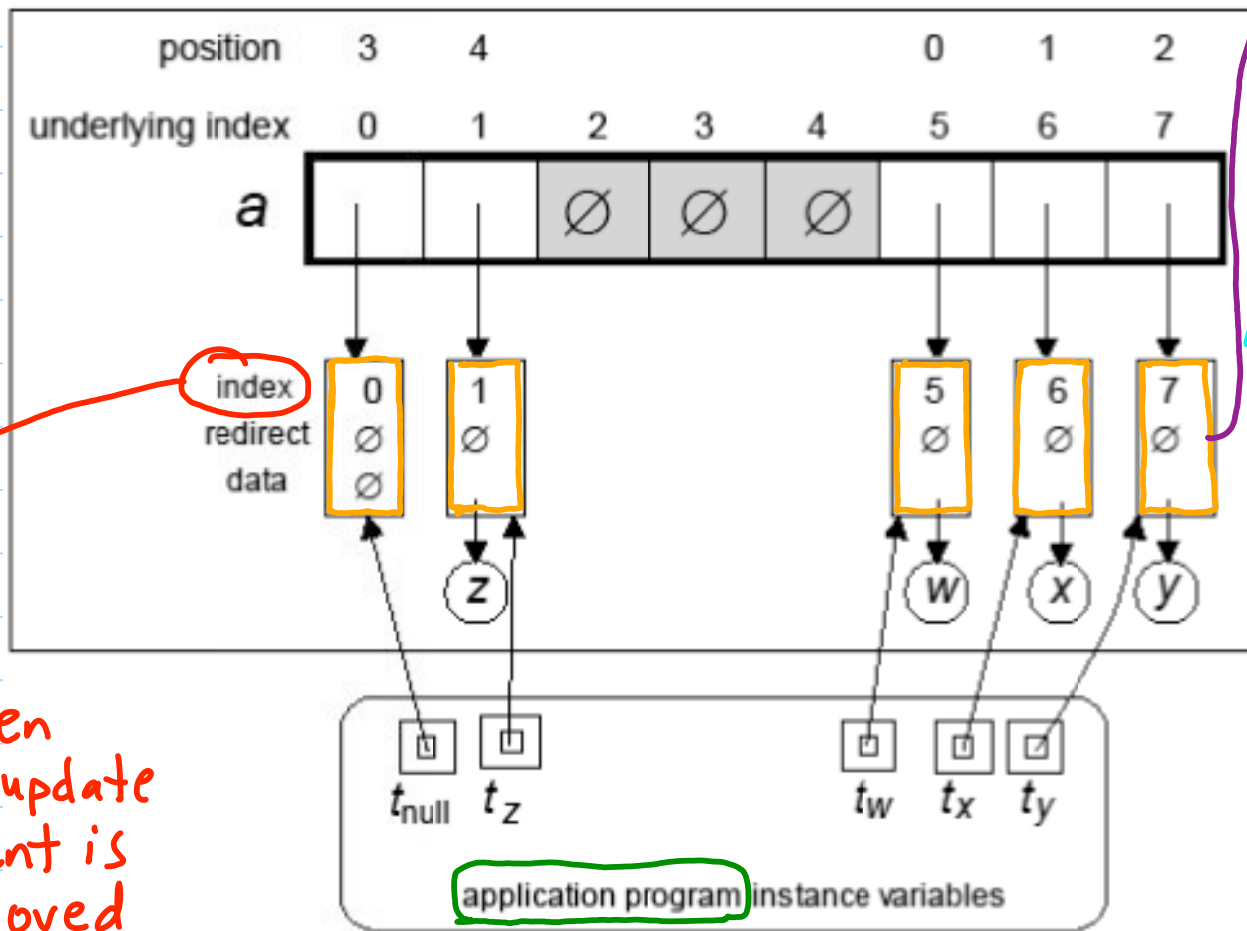
Figure 12.1

A circular array representing the positional collection $\langle w, x, y, \text{null}, z \rangle$ where $\text{start}=5$.

* implementation in text guarantees that array slots not in use are null.

I called this offset last class.

Tracked Array holding $\langle w, x, y, \emptyset, z \rangle$



underlying index (not position) stored. Then only need to update when element is physically moved

redirect used to navigate to next element in iteration order from a tracked element even when it has been deleted