COMP 2402 - Lec 4

how were drills #1 & #2? Drills 3,4 due nextFri;
A1 due next Thu, workshop 4pm today

Review array-based implementations of List, Stack, Queue, Deque

- started with array a, int n, elts stored in a starting at index 0
- this was "good enough" for O(1) (amortized) Stack & ok List (O(n-i+1) add/remove.)
- but add/rem @ index 0 requires O(n) shift, bad for Queue & Deque
- so we added int j to represent the (index of the) 1st elt of our list & stored elts at indices $j, j+1, \ldots, j+n-1$ (% a.length)
- this "circular" array got us $O(1)^A$ ← amortized Queue operations
- circular array + choosing which direction to shift got us $O(1)^A$ Deque operations & $O(1+\min\{i, n-i\})^A$ List add/remove

Today: diff. impl. of Deque/List interface in same time/space.

O(1) get(i), set(i,x), size, isEmpty

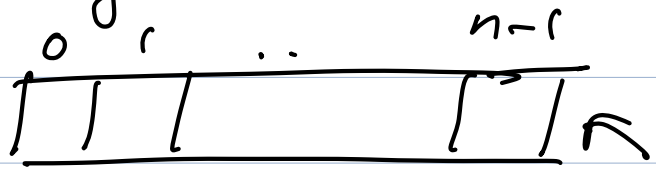
$O(1+\min\{i, n-i\})^A$ add(i,x), remove(i)

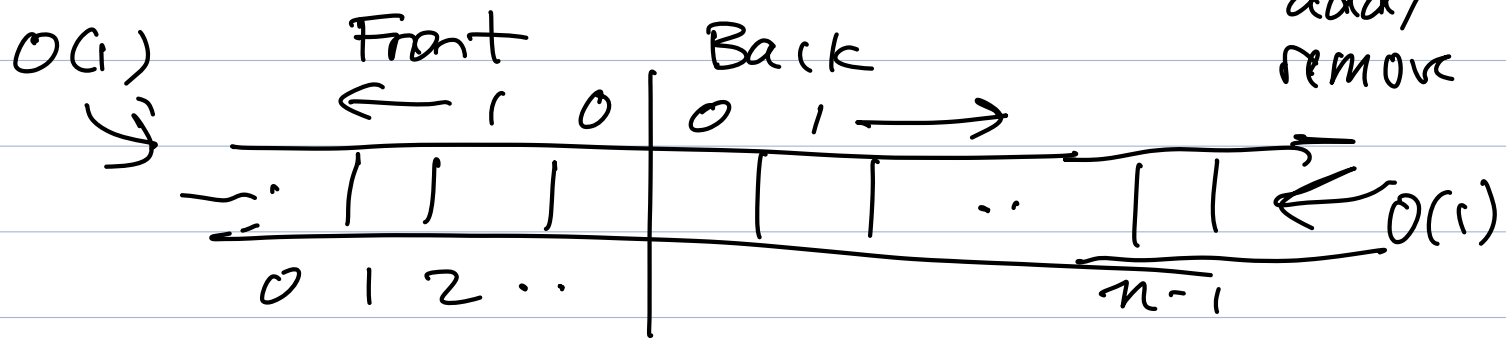
DualArrayDeque

implements List / Deque interfaces with same time & space as ArrayDeque.

idea: use 2 arrayStacks end to end as our backing storage

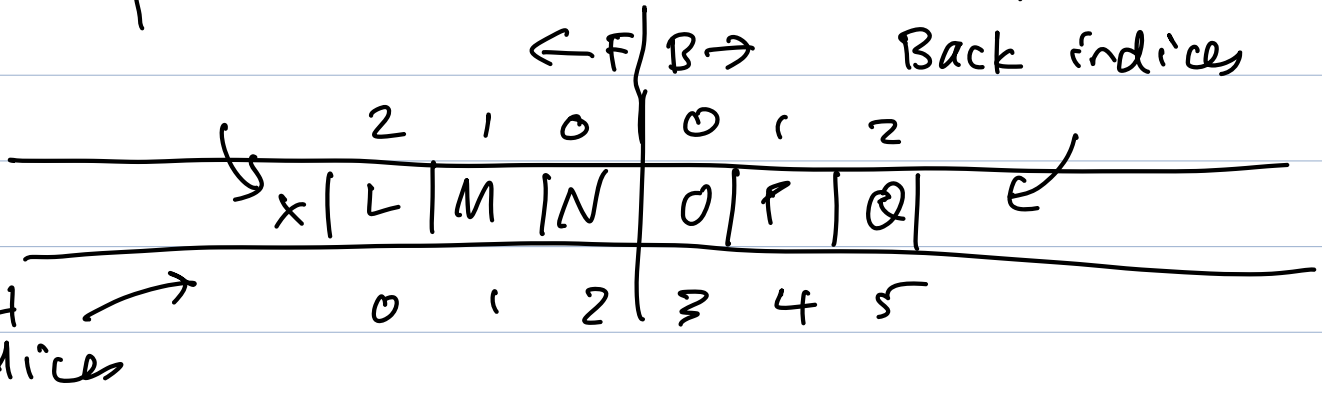
ArrayStack





ArrayStack<T> Front (F) // front "half"

ArrayStack<T> Back (B) // back "half"



addFront(x) ≡ F.add(x) / F.push(x)

removeFront() ≡ F.remove(F.size-1) / F.pop()

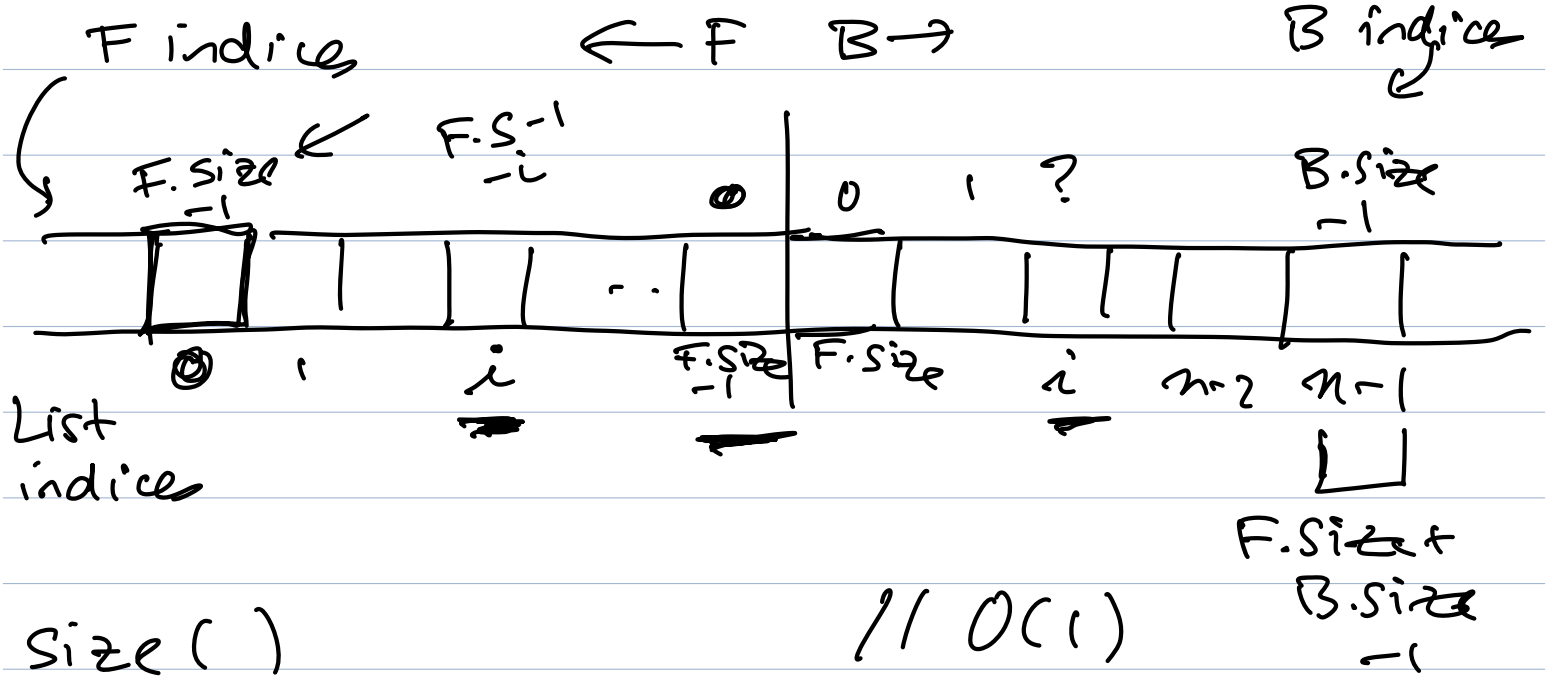
addBack(x) ≡ B.add(x) / B.push(x)

removeBack() ≡ B.remove(B.size-1) / B.pop

↑ these are all $O(1)^A$ b/c push + pop on stack is $O(1)^A$

Q: What if F or B becomes empty?
These won't quite work...

Q: What about List operations that use indices?
Need to deal with these too.



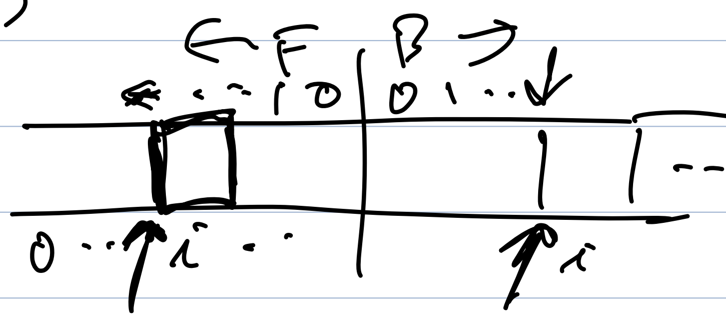

```
size()                          // O(1)
    return F.size() + B.size()

isEmpty()                       // O(1)
    return size() == 0

get(i)  // return ith elt of our List
    if i < F.size               // i is in F
```

```
    return F.get(F.size - 1 - i)
else        // i >= F.size , i is in B
    return B.get(i - F.size)
```

RT of get is RT of AS get() which is $O(1)$

```
set(i, x)
  if i < F.size
      return F.set(F.size - 1 - i, x)
  else
      return B.set(i - F.size, x)
```

RT also $O(1)$




```
add(i, x)
 ① if i < F.size
       F.add(F.size - i, x)      // F.size - i = i_F
 ② else
       B.add(i - F.size, x)      // i - F.size = i_B
 balance()
```

↳ do this later, but for now know that this ensures

$\frac{n}{4} \leq F.s , B.s \leq \frac{3n}{4}$

i.e. $F.s \leq 3 \cdot B.s$

& $B.s \leq 3 \cdot F.s$

RT of add if we ignore balance RT:

2 cases:

① if $i < F.size$:

RT of F's add is

$O(n_F - i_F + 1) = O(F.size - (F.size - i) + 1)$

$= O(i+1)$

② if $i \geq F.size$

RT of B's add

$O(n_B - i_B + 1) = O(B.size - (i - F.size) + 1)$

$= O(F.size + B.size - i + 1)$

$= O(n - i + 1)$

Note that when / if $i < \frac{n}{4} \leq F.S$

So in this case we're in case 1.

i.e.

when $i < \frac{n}{4}$ RT is $O(i+1)$ $\leq \frac{n}{4} + 1$

And if $i \geq \frac{3n}{4} \geq F.S$

So in this case we're in case 2

i.e. when $i \geq \frac{3n}{4}$ RT is $O(n-i+1)$ $\leq \frac{n}{4} + 1$

If $\frac{n}{4} \leq i < \frac{3n}{4}$ then we could be in either case, but both $O(i+1)$ and $O(n-i+1)$ is $O(n)$

in this range.

$$\therefore \text{RT of add}(i,x) = \begin{cases} O(i+1) & \text{if } i < \frac{n}{4} \\ O(n) & \text{if } \frac{n}{4} \le i < \frac{3n}{4} \\ O(n-i+1) & \text{if } i \ge \frac{3n}{4} \end{cases}$$

This is $O(1 + \min\{i, n-i\})^A$

```
remove(i)
  if i < F.size
     x = F.remove(F.size - i - 1)
  else
     x = B.remove(i - F.size)
  balance()
  return x
```

RT analysis very similar to the add : $O(1 + \min\{i, n-i\})^A$ without balance.

```
balance
  if n ≤ 1  or if  F.size ≤ 3.B.size
                and  B.size ≤ 3.F.size
```

then return // do nothing

In $O(n)$ time:

put all elements of F & B into a 3rd arraystack in correct list order using get/set

ex. $AS[0] \in F.get(F.size()-1)$

In $O(n)$ time:

put 1st $\lfloor \frac{n}{2} \rfloor$ elements of AS into F (in reverse order)

ex. $AS(0) \Rightarrow F.set(\lfloor \frac{n}{2} \rfloor - 1, \cdot)$

put last $\lceil \frac{n}{2} \rceil$ elements of AS into B (in forward order)

Note this leaves $\lfloor \frac{n}{2} \rfloor$ elts in F, and $\lceil \frac{n}{2} \rceil$ elts in B & does this in $O(n)$ time (it does use tertiary $O(n)$ space)

Lemma 2.2 If start with an empty Dual Array Deque, perform $m \geq 1$ adds/removes, then total time spent in balance is $O(m)$

(ie amortized $O(1)$ add/remove)

Proof. As with Arraystack resize analysis, we'll show that if balance has to do $O(n)$ work, then there were $\geq \frac{n}{2}$ adds or removes since the last balance.

Define "potential function"
$\Phi(F, B) = |F.size - B.size|$

① add / removes change Φ by $= 1$
(unless there's a rebalance)

② right after a balance $\Phi \leq 1$
b/c $|\lfloor \frac{n}{2} \rfloor - \lceil \frac{n}{2} \rceil| \leq 1$.

③ right before a balance we have either:

$F.s < \frac{n}{4}$, $B.s > \frac{3n}{4}$
or $B.s < \frac{n}{4}$, $F.s > \frac{3n}{4}$
$\Big\}$ $|F.s - B.s| > \frac{n}{2}$

Φ after a balance is ≤ 1
before next balance it's $> \frac{n}{2}$
& each add / remove change Φ by 1
$\therefore$ we have $\geq \frac{n}{2}$ adds / removes to change Φ from 1 to n.

~~resize i~~ , ~~resize $i+1$~~ ...

balance i, $\geq \frac{n}{2}$ A(R

balance $i+1$

<u>**Thm 2.4**</u> DualArrayDeque implements the List interface in

$O(1)$ get, set

$O(1+\min\{i, n-i\})^A$ add, remove