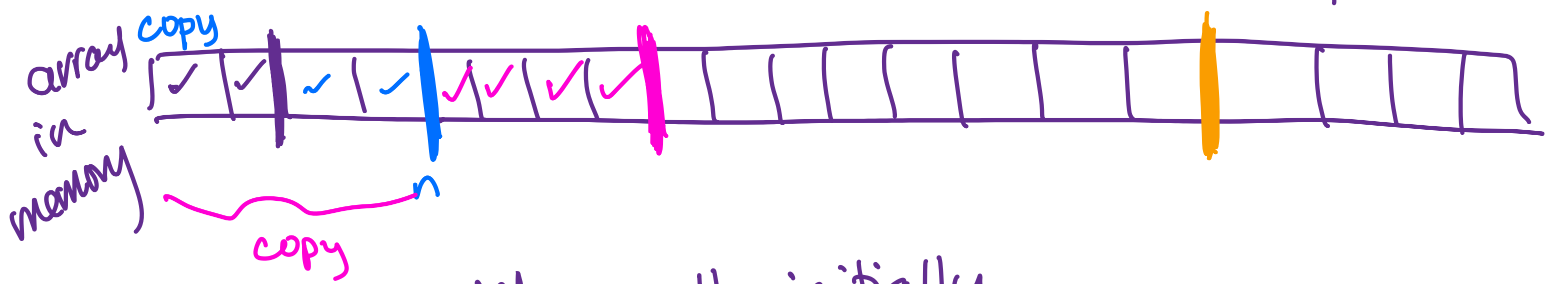
Lecture 12 – ArrayLists and Runtime

Summarize Worst-Case Runtimes (in terms of number of elements in the list)

	LinkList (immutable)	MutableList (Link)	ArrList
size	Constant (if keep field, otherwise linear to compute)		
addFirst	constant	constant	for now, linear
addLast	linear	constant	usually constant
get(index)	linear	linear	constant

unless array is full, then linear

What's with "usually constant"?



Create array with 2 cells initially

· each of the first two addLasts costs constant time

What is the cost of addlast?

- constant if array has space
- otherwise linear in #elts

} By worst case, addlast is linear, even though usually constant

What if instead we ask for the time to make k calls to addlast?

$$k \text{ times} \left\{ \begin{array}{l} \text{addlast} \\ \text{addlast} \\ \vdots \\ \text{addlast} \end{array} \right.$$

what's the average time needed per call among the k calls?

$$\boxed{\frac{\text{total cost of k calls}}{k}}$$

this is where the amount of resize matters.

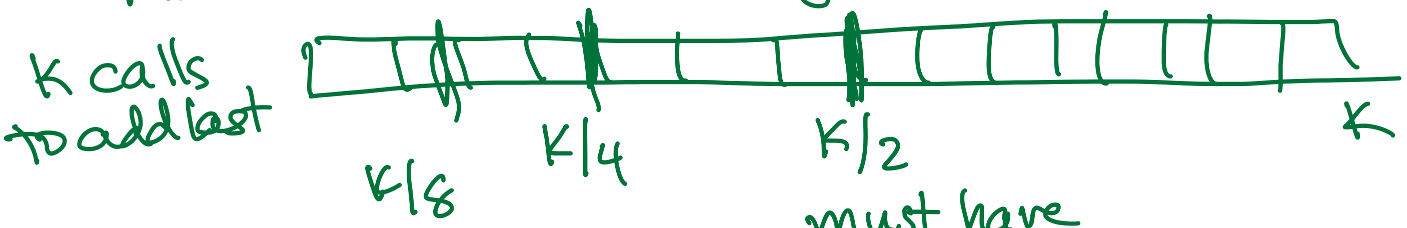
if we add 1 cell per resize, then each call to addlast is linear time.

$$\text{total cost is } \frac{\overset{\text{time per call}}{O(K)} * \overset{\text{each call}}{K}}{K}$$

average it out

get linear time per call

Now let's double the size when out of room.



must have resized

total cost of getting to k elements

$$\underbrace{\ldots + K/8 + K/4 + K/2}_{\text{sums to } K*2 \text{ in total costs}} + K$$

cost for each insert

---

amortized cost is $\frac{2K}{K}$ avg to 2 constant O(1)

addlast has constant amortized time

over mult. calls to addlast

Runtime of AddLast/AddFirst with Resizing

```
public class ArrList {
   String[] theArray;  // the underlying array that stores the elements
   int eltcount;       // how many elements are in the array
   int end;            // the last USED slot in the array

   private void resize(int newSize) {
       // make the new array
       String[] newArray = new String[newSize];
       // copy items from the current theArray to newArray
       for (int index = 0; index < theArray.length; index++) {
           newArray[index] = this.theArray[index];
       }
       // change this.theArray to refer to the new, larger array
       this.theArray = newArray;
   }

   public void addLast(String newItem) {
       if (this.isFull()) {
           // add capacity to the array
           this.resize(this.theArray.length + 1);
           // now that the array has room, add the item
           this.addLast(newItem);
       } else {
           if (!(this.isEmpty())) {
               this.end = this.end + 1;
           }
           this.eltcount = this.eltcount + 1;
           this.theArray[this.end] = newItem;
       }
   }
```

```
public class ArrTest {
   ArrList flavors = new ArrList(2);
   flavors.addLast("mint")
   flavors.addLast("grape")
   new Course("cs1410", 200)
   flavors.addLast("lemon")
   flavors.addLast("cherry")
}

------------------------------------
-
environment
flavors → @1221
```

@1221	**ArrList** theArray: @1222 end: 1 eltcount: 2
@1222	"mint"
@1223	"grape"
@1224	Course("cs1410", 200)
@1225	
@1226	
@1227	
@1228	

How many resizes get done across N calls to addLast? How does this affect runtime?

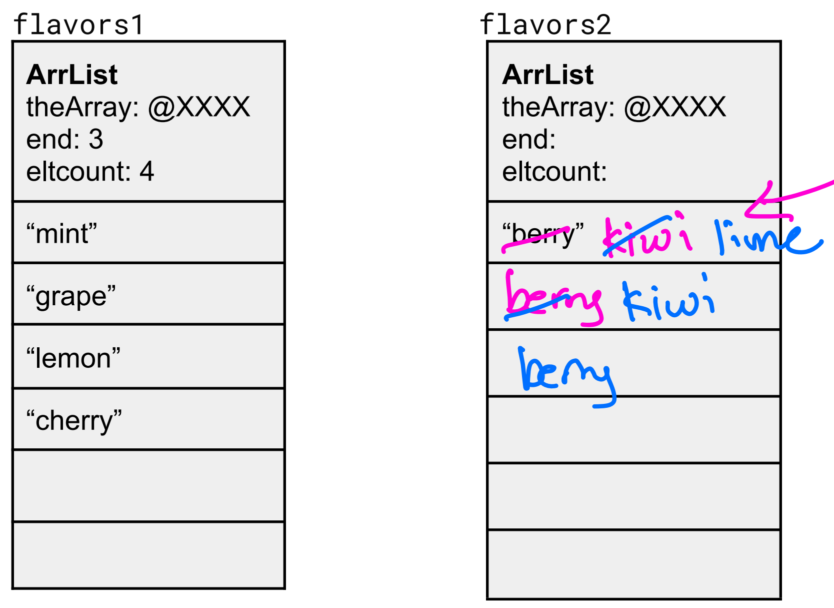
```
ArrList flavors = new ArrList(2);
```

	Resize by 1	Resize by 2	Resize by double
`flavors.addLast("mint")`			
`flavors.addLast("grape")`			
`flavors.addLast("lemon")`			
`flavors.addLast("cherry")`			
`flavors.addLast("mango")`			
`flavors.addLast("orange")`			
`flavors.addLast("coffee")`			

Enabling AddFirst – Leave space at front and end of the array to avoid resizing

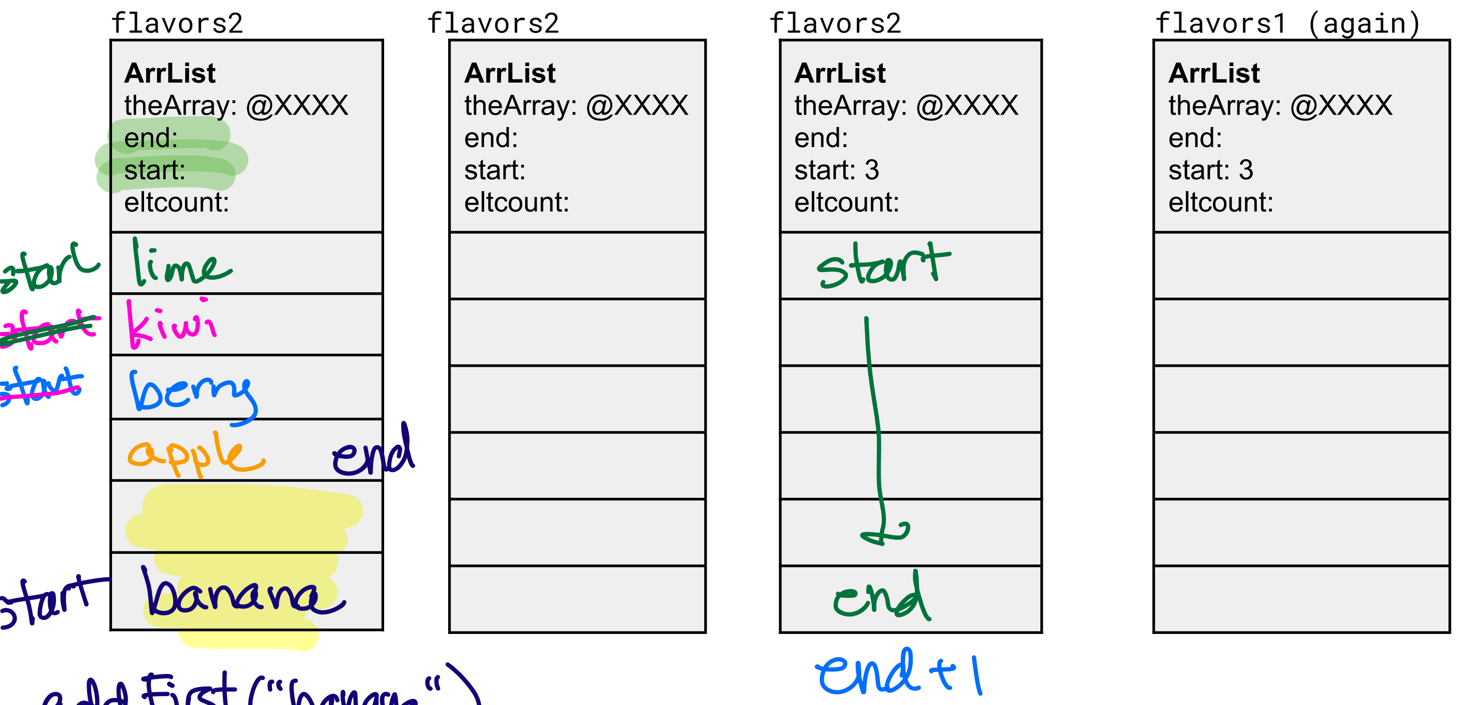
```
public class ArrTest1 {
   ArrList flavors1 = new ArrList(6);
   flavors.addLast("mint")
   flavors.addLast("grape")
   flavors.addLast("lemon")
   flavors.addLast("cherry")
}
```

```
public class ArrTest2 {
   ArrList flavors2 = new ArrList(6);
   flavors.addLast("berry")
   flavors.addFirst("kiwi")
   flavors.addFirst("lime")
   flavors.addLast("apple")
}
```



kiwi needs to be here. suggests that to addFirst, we move everything down by 1 slot

lots of copying ensues!



addFirst("banana")
what needs to happen?

```
class ArrList {
    int start ← where first elts is
    int end   ← where last elts is

  public addlast ()
     if (end+1 == start)        assumes end & start in
                                middle of the array
       \\ oops - array is full, must resize

       (end+1) % capacity   == start
                  ↑
              # of slots
```