Will use this document today

Find it:
• [https://goo.gl/Gzu6At](https://goo.gl/Gzu6At)
• or ICON announcement
• or calendar

sit in a 2-3 person group
Roles:
• Driver: follows the instructions on their computer
• Scribe: writes down answers the questions
• (optional 3rd person) Skeptic: double-check your group's answers as you go
CS 2230
CS II: Data structures

Meeting 15: Amortization, Iterators
Brandon Myers
University of Iowa
Today’s big ideas

• Even if an individual operation is expensive, with the right algorithm, many operations can be efficient when looked at together

• An iterator is an object that allows us to look at elements of a data structure one-at-a-time

• An iterator can return any sequence

• By making Iterator an interface, we can write code that uses an iterator but doesn’t need to know what data structure the elements came from
Return of the ArrayList: analyzing the running time of resizing

/*
A List that is implemented using an array
*/

public class ArrayList implements List {
    private Object[] elements;
    private int numElements;

    @Override
    public void append(Object ele) {
        // copy existing elements to a bigger array if necessary
        if (elements.length == numElements) {
            Object[] n = new Object[elements.length+1];
            for (int i=0; i<elements.length; i++) {
                n[i] = elements[i];
            }
            elements = n;
        }
        // insert ele
        elements[numElements] = ele;
        numElements++;
    }
}
Return of the ArrayList: an analysis of resizing
If it takes one “step” to copy one element, about how many total steps will be taken to call append 1000 times when the initial size was 4?

a) 1000
b) 2000
c) 1,000,000
d) 1,000,000,000

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Will use this document today

Find it:
• [https://goo.gl/Gzu6At](https://goo.gl/Gzu6At)
• or ICON announcement
• or calendar

sit in a 2-3 person group
Roles:
• Driver: follows the instructions on their computer
• Scribe: writes down answers the questions
• (optional 3rd person) Skeptic: double-check your group's answers as you go
<table>
<thead>
<tr>
<th>Nth element</th>
<th>number of steps for append</th>
<th>total steps for appends to this point</th>
</tr>
</thead>
<tbody>
<tr>
<td>5th</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6th</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7th</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>8th</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>9th</td>
<td>9</td>
<td>35</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>999</td>
<td>499490</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>500490</td>
</tr>
</tbody>
</table>
Simulation

Calculate the total time by a simple simulation!

steps = 0
# initially ArrayList's internal array is size 4
arraysize = 4
for n in range(1, 1001):
    if n > arraysize:
        # ArrayList is full so n steps to copy to new array of size n+1
        steps += n
        # new array has +1 size
        arraysize += 1

    # 1 step to copy the new value into the last spot
    steps += 1

print str(n)+"\t"+str(steps)

(not the same as an experiment, where we actually time the ArrayList insertions)
append time is accelerating!
Another approach: Geometric resizing

(double the size when we run out of space)
Simulate this new scenario

Calculate the total time by a simple simulation!

```
steps = 0
# initially ArrayList's internal array is size 4
arraysize = 4
for n in range(1, 1001):
    if n > arraysize:
        # ArrayList is full so n steps to copy to new array of size 2n
        steps += n
        # new array has x2 size
        arraysize *= 2
    # 1 step to copy the new value into the next open spot
    steps += 1

print str(n)+"\t"+str(steps)

(not the same as an experiment, where we actually time the ArrayList insertions)
```
some insertions take a long time, but the total time is growing linearly!