CS 106
INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- **Lab 1 due TOMORROW**
- **Lab 2 due next Wednesday**
- **Before lab on Friday, mostly complete with the pre-lab (okay to finish in lab)**
- **Office Hours TODAY 4:30-6pm (H110)**
- **Review quiz will be collected on Thursday**
  - Full credit for putting your name and answering one question (does not have to be correct!)
  - On the material from today

FEB 11 OUTLINE

- Digital Scholars program
- StringBuilder
- Begin: runtime analysis
- Recap ArrayLists and Generics
- Begin: Singly Linked Lists

FEB 11 OUTLINE

- Digital Scholars program
- **StringBuilder**
- Begin: runtime analysis
- Recap ArrayLists and Generics
- Begin: Singly Linked Lists

STRING BUILDER

Since Strings are immutable, any change to the reference means a new copy has to be created!

```
String greeting = "hello";
greeting += "!";
System.out.println(greeting);
```

STRING BUILDER

Since Strings are immutable, any change to the reference means a new copy has to be created!

```
String greeting = "hello";
greeting += "!";
System.out.println(greeting);
```

StringBuilder provides a way of creating a mutable String.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append("!");
sb.setCharAt(0, 'M');
sb.insert(4, "oooo");
System.out.println(sb);
```

STRING BUILDER

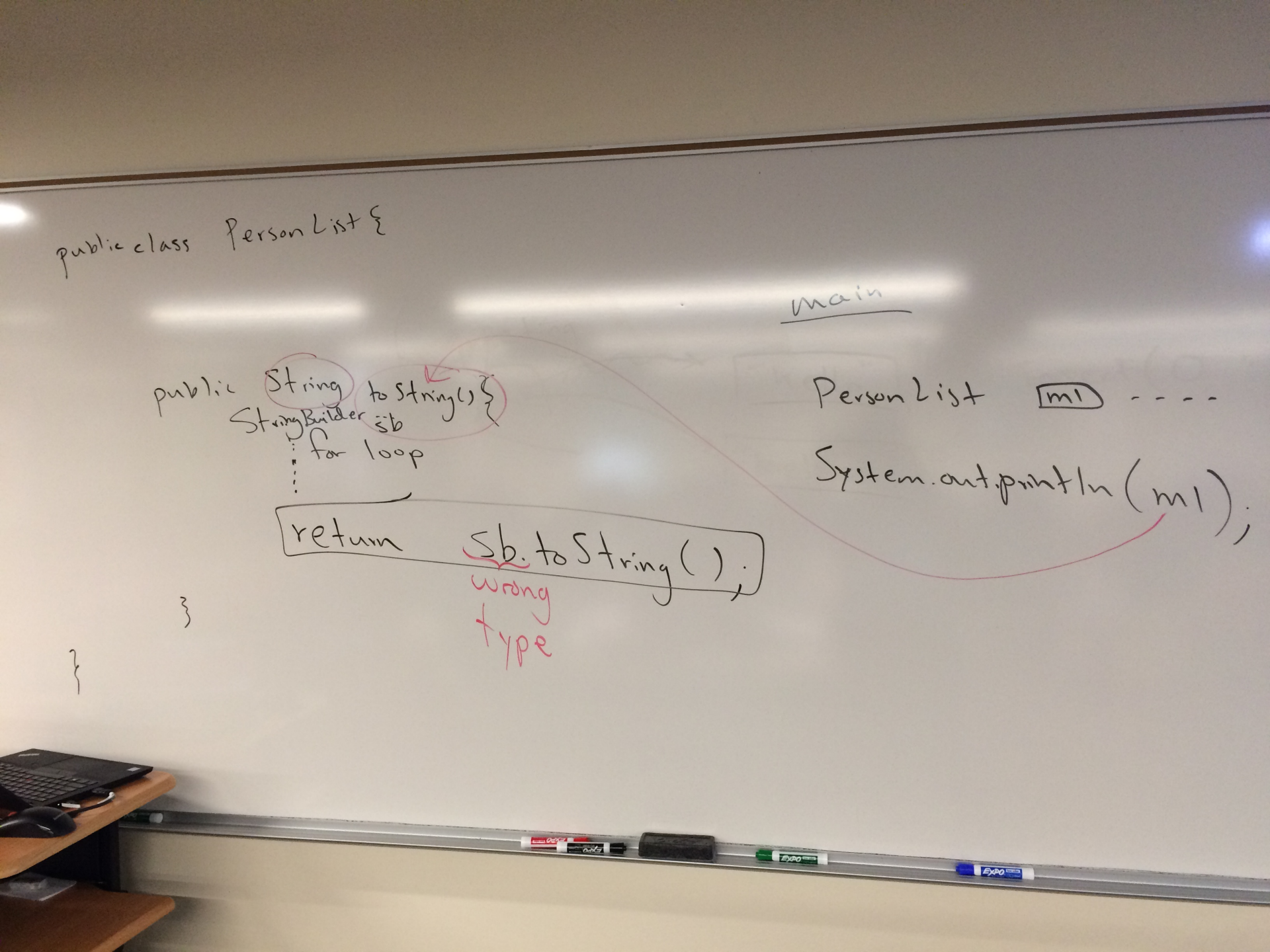
Since Strings are immutable, any change to the reference means a new copy has to be created!

```
String greeting = "hello";
greeting += "!";
System.out.println(greeting);
```

StringBuilder provides a way of creating a mutable String.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append("!");
sb.setCharAt(0, 'M');
sb.insert(4, "oooo");
System.out.println(sb);
```

Output: “Mellooooo!”


public class PersonList {
public String toString() {
StringBuilder sb
for loop
return sb.toString();
wrong type
}
}
main
PersonList ml ----
System.out.println(ml);

FEB 11 OUTLINE

- Digital Scholars program
- StringBuilder
- **Begin: runtime analysis**
- Recap ArrayLists and Generics
- Begin: Singly Linked Lists

RUNTIME: HANDOUT 7, PG 1

1.
```
n = int(input("n: "))
for i in range(n):
    print(i)
```

RUNTIME: HANDOUT 7, PG 1

1.
```
n = int(input("n: "))
for i in range(n):
    print(i)
```

n steps => O(*n*)

linear

2.
```
n = int(input("n: "))
for i in range(100):
    print(i*n)
```

RUNTIME: HANDOUT 7, PG 1

1.
```
n = int(input("n: "))
for i in range(n):
    print(i)
```

n steps => O(*n*)

linear

2.
```
n = int(input("n: "))
for i in range(100):
    print(i*n)
```

100 steps => O(1)

constant

3.
```
n = int(input("n: "))
for i in range(n):
    print(i)
for j in range(n):
    print(j)
```

RUNTIME: HANDOUT 7, PG 1

1.
```
n = int(input("n: "))
for i in range(n):
    print(i)
```

n steps => O(*n*)

linear

2.
```
n = int(input("n: "))
for i in range(100):
    print(i*n)
```

100 steps => O(1)

constant

3.
```
n = int(input("n: "))
for i in range(n):
    print(i)
for j in range(n):
    print(j)
```

2*n* steps => O(*n*)

linear

RUNTIME: HANDOUT 7, PG 1

4.
```
n = int(input("n: "))
for i in range(n):
    for j in range(n):
        print(i, j)
```

RUNTIME: HANDOUT 7, PG 1

4.
```
n = int(input("n: "))
for i in range(n):
    for j in range(n):
        print(i, j)
```

n^2 steps => $O(n^2)$

quadratic

5.
```
n = int(input("n: "))
for i in range(n):
    for j in range(i,n):
        print(i, j)
```

RUNTIME: HANDOUT 7, PG 1

4.
```
n = int(input("n: "))
for i in range(n):
    for j in range(n):
        print(i, j)
```

n^2 steps => $O(n^2)$

quadratic

5.
```
n = int(input("n: "))
for i in range(n):
    for j in range(i,n):
        print(i, j)
```

$n^2/2$ steps => $O(n^2)$

quadratic

6.
```
n = int(input("n: "))
for i in range(n):
    for j in range(10):
        print(i, j)
```

RUNTIME: HANDOUT 7, PG 1

4.
```
n = int(input("n: "))
for i in range(n):
    for j in range(n):
        print(i, j)
```

n^2 steps => $O(n^2)$

quadratic

5.
```
n = int(input("n: "))
for i in range(n):
    for j in range(i,n):
        print(i, j)
```

$n^2/2$ steps => $O(n^2)$

quadratic

6.
```
n = int(input("n: "))
for i in range(n):
    for j in range(10):
        print(i, j)
```

$10n$ steps => $O(n)$

linear

RUNTIME: HANDOUT 7, PG 1

7.
```
n = int(input("n: "))
while n > 1:
  print(n)
  n = n/2
```

RUNTIME: HANDOUT 7, PG 1

7.
```
n = int(input("n: "))
while n > 1:
  print(n)
  n = n/2
```

log(*n*) steps => O(log(*n*))

logarithmic

8.
```
arraylst = {1,2,5,7,13,21,24,25,26,33,34,38,50,57,58,63}
n = arraylst.length
mid = int(n/2)
print lst[mid]
```

RUNTIME: HANDOUT 7, PG 1

7.
```
n = int(input("n: "))
while n > 1:
  print(n)
  n = n/2
```

log(*n*) steps => O(log(*n*))

logarithmic

8.
```
arraylst = {1,2,5,7,13,21,24,25,26,33,34,38,50,57,58,63}
n = arraylst.length
mid = int(n/2)
print lst[mid]
```

3 steps => O(1)

constant

9.
```
n = int(input("n: "))
for i in range(n):
  k = n
  while k > 1:
    print(i, k)
    k = k/2
```

RUNTIME: HANDOUT 7, PG 1

7.
```
n = int(input("n: "))
while n > 1:
  print(n)
  n = n/2
```

log(*n*) steps => O(log(*n*))

logarithmic

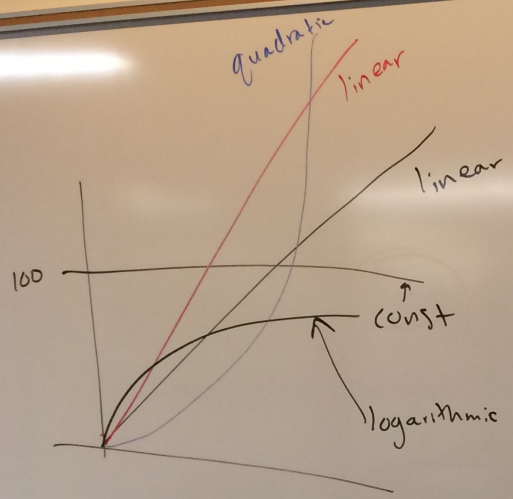
8.
```
arraylst = {1,2,5,7,13,21,24,25,26,33,34,38,50,57,58,63}
n = arraylst.length
mid = int(n/2)
print lst[mid]
```

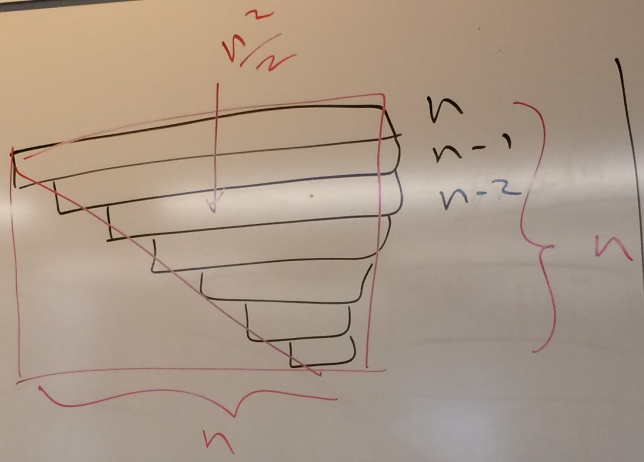
3 steps => O(1)

constant

9.
```
n = int(input("n: "))
for i in range(n):
  k = n
  while k > 1:
    print(i, k)
    k = k/2
```

n log(*n*) steps => O(*n* log(*n*))





$n + (n-1) + \cdots 2 + 1 = S$

$$\frac{n(n+1)}{2}$$

log

$$\frac{n}{2^x} = 1$$

$$\log(n) = \log(2^x)$$

$$\boxed{x = \log(n)}$$

FEB 11 OUTLINE

- Digital Scholars program
- StringBuilder
- Begin: runtime analysis
- **Recap ArrayLists and Generics**
- Begin: Singly Linked Lists

ARRAYLISTS: INITIALIZATION

```
import java.util.ArrayList;
ArrayList<String> myList = new ArrayList<String>();
```

Imports the library for the ArrayList class.

The type that the underlying array will be declared to have. These are known as generics in Java.

No need to give a specific length!

Source code for the class:

```
public class ArrayList<T> {
      private T[] dataArray;
…
```

This is how classes with generic types are created and then used.

https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Slide: Sorelle Friedler

LAST TIME

- Wrote our own ArrayList class!
- Implemented the “add” method

```
public void add(T elem) {
    int l = array.length;

    // no more space: resize
    if (numElem == l) {
        T[] arrayNew = (T[]) new Object[l*2];
        for (int i=0; i < l; i++) {
            arrayNew[i] = array[i];
        }
        ____________________________;
    }

    // add new element
    array[    _________    ] = elem;
    numElem _________;
}
```

LAST TIME

- Wrote our own ArrayList class!
- Implemented the “add” method

```
public void add(T elem) {
    int l = array.length;

    // no more space: resize
    if (numElem == l) {
        T[] arrayNew = (T[]) new Object[l*2];
        for (int i=0; i < l; i++) {
            arrayNew[i] = array[i];
        }
        array = arrayNew; // update reference
    }

    // add new element
    array[numElem] = elem;
    numElem += 1;
}
```

Why is adding an element to the end of an ArrayList O(1) on average?

initial space = a

scaling factor = r

Example: a = 2, r = 2

Q: how much work to get to size n?

generic

runtime: $n\left(\frac{1}{1-\frac{1}{r}}\right)$ (constant)

$\Rightarrow$ linear

runtime $= 2+4+8+16+32+\cdots n$

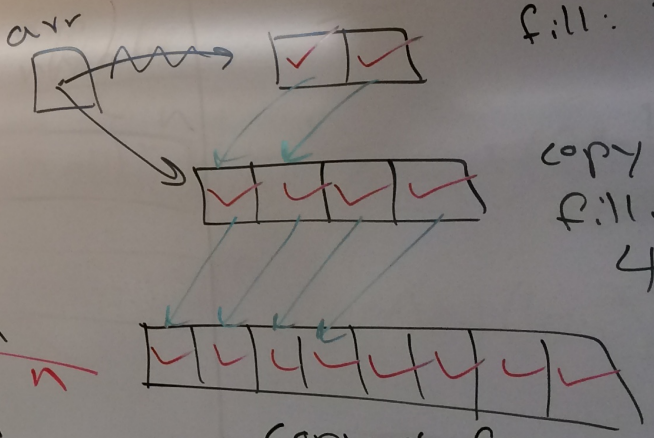
$2\ 4\ 6\ 8\ 10\ \cdots\ n$

$= 2 + \cdots + \frac{n}{4} + \frac{n}{2} + n$

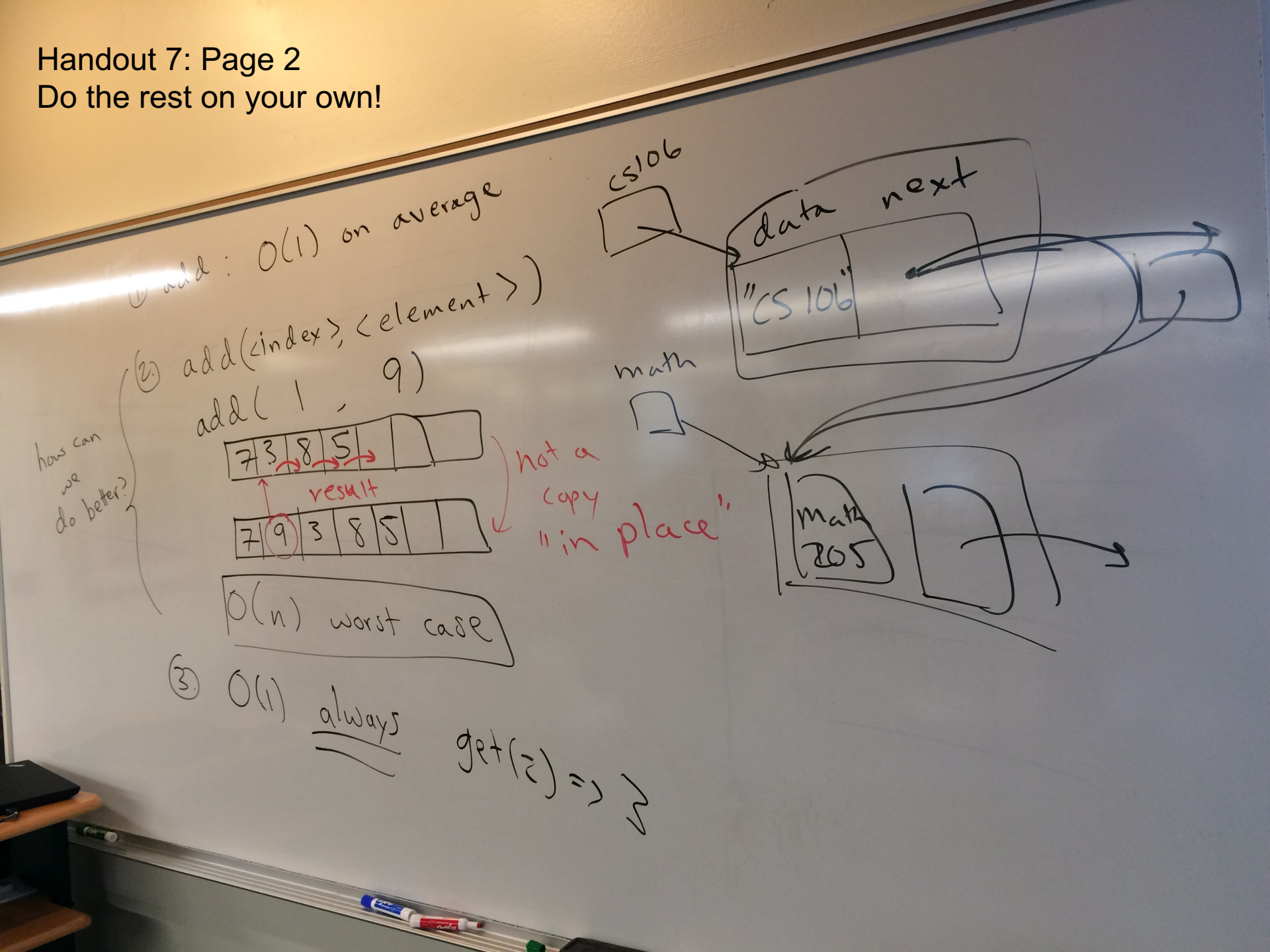
$= n\left(1+\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\cdots \frac{2}{n}\right)$

geometric series

$= \frac{n}{1-\frac{1}{2}} = 2n$



Handout 7: Page 2
Do the rest on your own!



FEB 11 OUTLINE

- Digital Scholars program
- StringBuilder
- Begin: runtime analysis
- Recap ArrayLists and Generics
- **Begin: Singly Linked Lists**

SINGLY LINKED LISTS

Arrays and ArrayLists need to do work to store more items than anticipated.

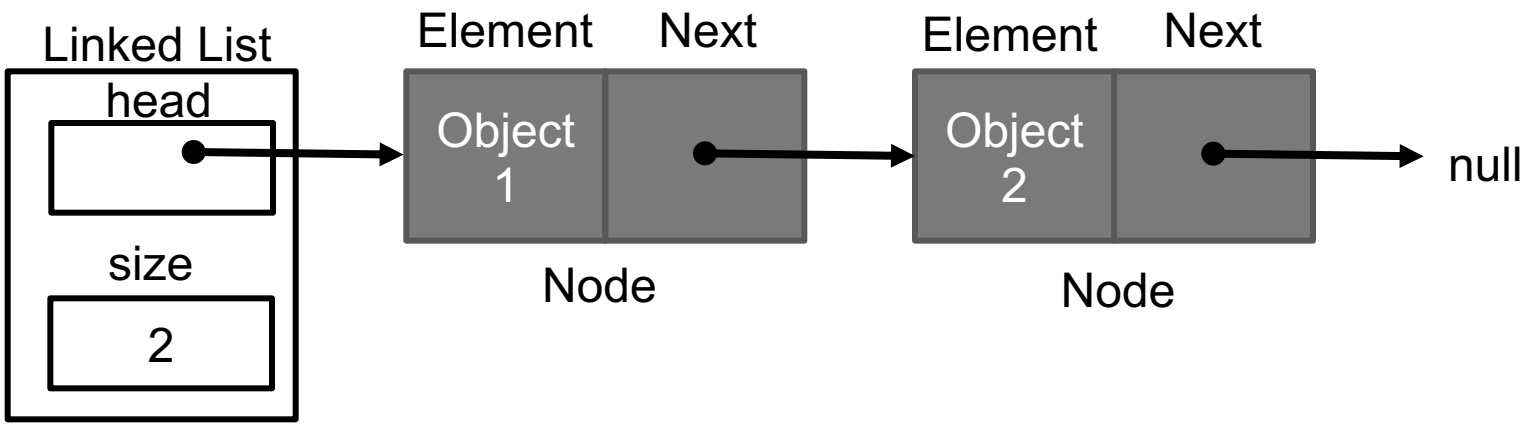
What if we want our list storage to grow and shrink quickly and only store as much as we're using?

We can use a linked list! Every item stores a pointer to the next item.

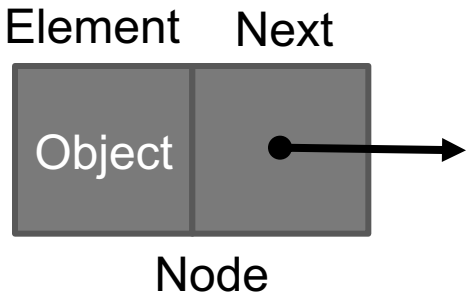
Want more info? See book section 3.2.

SINGLY LINKED LISTS

Singly Linked Lists store items in order in a list by storing the head and size of the list. Each item (Node) then stores a pointer to the next item.

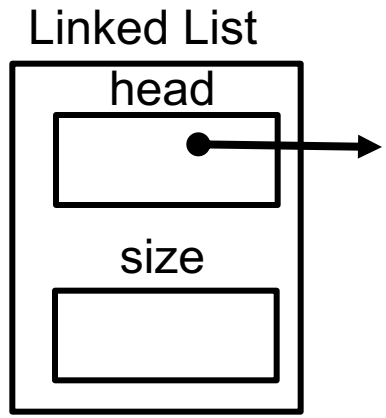


CODING EXERCISE



Make a Node class that can be used as part of a Linked List:

1. **Make a Node class that takes a generic type E**
2. **Add the relevant private fields to the Node class**
3. **Add a constructor that takes these fields as input**
4. **Add needed getters and setters**



Make a LinkedList class:

1. **Add and initialize needed instance variables**
2. **Make a constructor that takes no input**

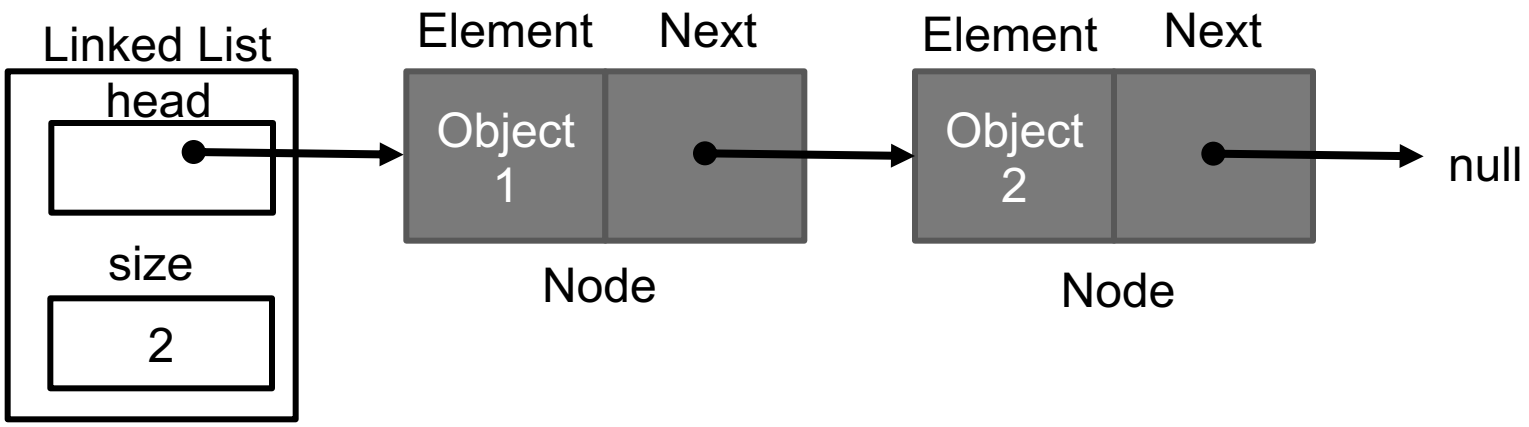
Slide: Sorelle Friedler

TRAVERSING A LINKED LIST

start at the head pointer and let the current node = head

while the current node is not null

- get the element
- let the current node = next



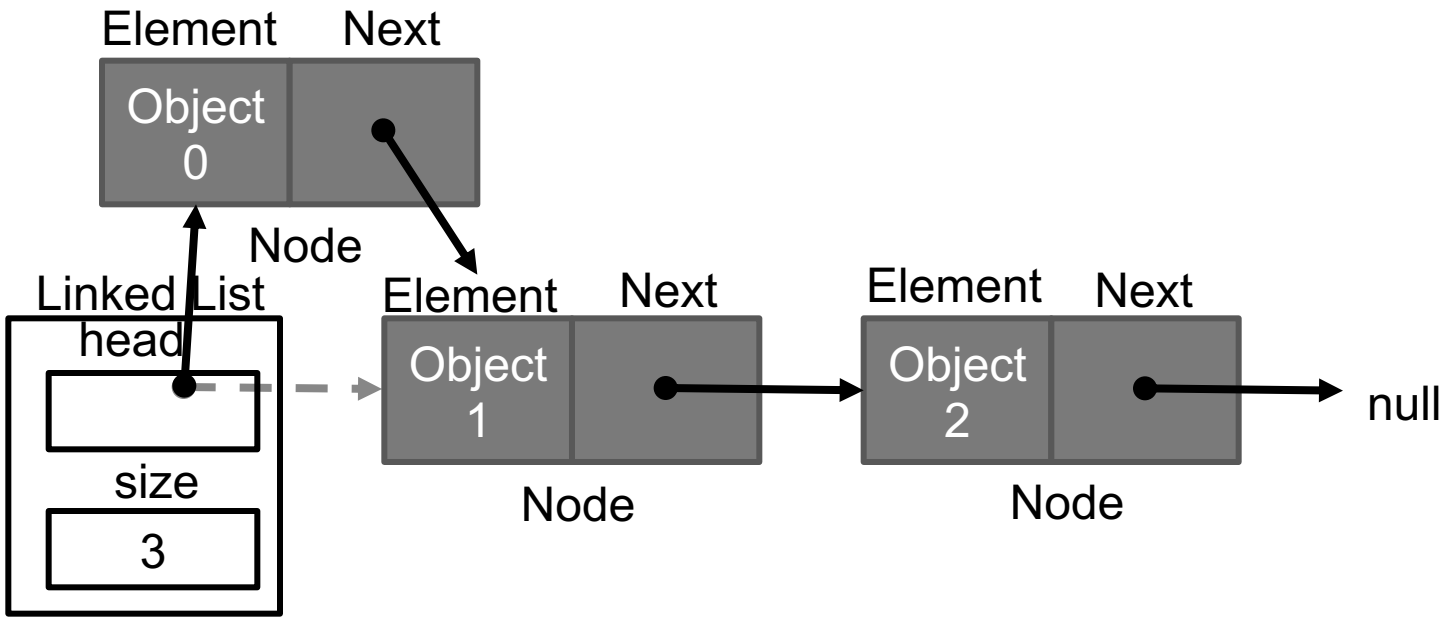
Slide: Sorelle Friedler

ADDING AN ELEMENT (AT THE HEAD)

Create the Node and set a next pointer equal to head

Let head point to the created Node

Add 1 to the size



ADDING AN ELEMENT (NOT AT THE HEAD)

Slow!

Traverse the list until the Node before the one being added

Create a new Node, link it to the Node that will come after it in the list, and point to it from the Node before it

Add 1 to the size