administrivia...
-assignment 2 is due tomorrow at midnight (11:59pm)

-next assignment goes out today
  -due Thursday at midnight
  -requires a partner!

-tutoring experiment
NEW! As a pilot program we will be offering tutoring sessions in two forms -- 30 minute small group sessions, and 1 hour discussion sessions. The small group sessions are to help students understand the concepts covered in class and in the assignments. These sessions are aimed to help those who don’t have programming background or who feel that they are lagging in the concepts as compared to their classmates. In these sessions you can discuss specific concepts that you are interested in with the TA, but not these sessions WILL NOT BE to debug your homework. Up to 3 students can sign up in single slot. In the discussion sessions we will discuss topics proposed by you. When you sign up for a discussion session slot you can propose topics for discussion by leaving a comment -- you must do this at least one day before the session. Discussion sessions will be with 10-12 students for an hour. Doodle pools for signing up for any of these sessions will be linked under the weekly lectures below.

W3. ALGORITHM ANALYSIS & DATA STRUCTURES | JAN 27 & 29

reading - Data Structures & Problem Solving Using Java, Chapters 5 and 6.
lab - Lab 1: timing experiments

tutoring - sign-up for small group sessions, located in MEB 3423
- sign-up for discussion sessions, located in MEB 3485

slides - L05-algo-analysis.pdf
CLICKERS!!!
...again <sigh>
Channel 26
Session ID 283755
ARE YOU HERE?

1) yes
2) no
feedback on Assignment 1
-grade is strongly correlated with the amount, and quality, of your testing code
  -write many tests! keep them organized!
  -will get practice with JUnit testing in lab on Monday

-include name (and partner’s name) on every file submitted
  -good practice is to include them in a comment header for each class

-do NOT change the signatures for the methods we give you… this will break the grader

-we will be requiring Javadoc comments starting with Assignment 3
  -TAs will review Javadocs in lab on Monday

-many people lost points on toString method
  -read specs carefully!
How many hours did you spend on Assignment 1?

A) <5
B) 5-10
C) 10-15
D) 15-20
E) >20
last time...
algorithm analysis
- correctness is only half the battle

- programs are expected to terminate in a reasonable amount of time

- running time of a program is strongly correlated to the choice of algorithms used in problem solving

- how much time and space does an algorithm require?
typical run-time complexities

$N^2$ and $N^3$ are typically not acceptable for moderate input sizes!
-for sufficiently large $N$, a function’s growth rate is determined by its dominate term

$10N^2 + 40N + 760$ → **WHAT IS THE DOMINATE TERM?**

- $c$ constant
- $\log N$ logarithmic
- $N$ linear
- $N \log N$ linearithmic
- $N^2$ quadratic
- $N^3$ cubic

increasing growth rate
how to get log growth?

- how many bits are needed to represent \( N \) consecutive integers?

- starting at \( x=1 \), how many iterations of \( x*2 \) before \( x\geq N \)?
  - the repeated doubling principle

- starting at \( x=N \), how many iterations of \( x/2 \) before \( x\leq 1 \)?
  - the repeated halving principle
- **big-O notation** (O) is used to capture the dominate term in an algorithm
  - assuming large N!

- for example, the running time of a quadratic algorithm is \( N^2 \) is specified \( O(N^2) \)
  - pronounced “order N squared”

- this notation allows us to establish a relative order among algorithms
  - \( O(N \log N) \) is better than \( O(N^2) \)
analyze the running time

for(int i=0; i<n; i+=2)
    sum++;

for(int i=0; i<n; i++)
    for(int j=0; j<n*n; j++)
        sum++

for(int i=0; i<n; i*=2)
    sum++;
today...
-collections

-iterators
Collection interface
- A Collection is a data structure that holds items very unspecific as to how the items are held - ie. the data structure

- Supports various operations:
  - add, remove, contains, ...

- Examples:
  - ArrayList
  - PriorityQueue
  - LinkedList
  - TreeSet
-you’ll be working with Collections for assignment 3
  -backed by an array for storage that you will implement yourselves
  -items will be sorted as they are inserted
  -no duplicates allowed

-what are some of the issues with using an array?
add

```java
int[] data = new int[6];
data.add(5);
data.add(17);
data.add(9);
data.add(12);
data.add(1);
data.add(33);
```

DON'T FORGET `size++`

WHAT IS THE COMPLEXITY OF `add`?

- A) O
- B) \(\log N\)
- C) N
- D) N \(\log N\)
- E) \(N^2\)
- F) \(N^3\)

```
data.add(22);
```

NOW WHAT???
-we need to grow our array!

-avoid allocating slightly larger arrays
  -you will most likely need to grow again soon

-good rule of thumb is to double the size
  -**con:** wastes up to 2x space
  -**pro:** growth will be rare
grow

data → 5 17 9 12 1 33

tmp = new int[data.length*2];
tmp →
copy all from data to tmp
tmp → 5 17 9 12 1 33
data = tmp;

data → 5 17 9 12 1 33

WHAT IS THE COMPLEXITY OF GROWING?

A) c
B) log N
C) N
D) N log N
E) N^2
F) N^3
remove

5 17 9 12 1 33 size = 6

data.remove(9);

5 17 12 1 33 size = 5

IS THIS CORRECT?
A) yes
B) no
what is the complexity of `remove`?

A) \( c \)  
B) \( \log N \)  
C) \( N \)  
D) \( N \log N \)  
E) \( N^2 \)  
F) \( N^3 \)
wait, what were we talking about?
Collection

- A Collection is an object that groups multiple elements into a single unit
  - Used to store, retrieve, manipulate, and communicate stored data
  - Is like an array except their size can change dynamically, and have more advanced behaviors

-The Java Collections API provides a set of classes and interfaces for storing data in different types of data structures
  - You will be implementing many of these on your own in assignments!
iterators
-not all data structures are guaranteed to use an array
  -thus, we can’t just do:
    
    ```java
    for(i=0; i<size; i++)
        data[i]...
    ```

-the **Iterator interface** provides generic retrieval of items from a data structure

-the **Collection interface** requires an **Iterator**
  -for example, `ArrayCollection` has `iterator()` method that returns an **Iterator**
**Iterator**

-an **Iterator** is specific to a data structure, and knows how to traverse the structure

- **hasNext**: determines if iteration is complete

- **next**: gets the next item

- **remove**: removes the last seen item

-internally, keeps track of where the next item is (as well as other state)

-actually points to **between** items
next

```
5 17 9 12 1 33
```

iterator

```
iterator.next(); // returns 9
iterator.next(); // returns 12
iterator.next(); // returns 1
```
**remove** removes the last item seen

```
iterator.remove();
```

**MUST BE PRECEDED BY A CALL TO next()**
enhanced for-loop

- allows for simplification of code by representing a visit to each element of an array or Collection without explicitly expressing how you get from element to element.

**Standard Way:**
```java
for(int i=0; i < things.length; i++)
    // do something with things[i]
```

**Enhanced Loop:**
```java
for(Thing t : things)
    // do something with t
```

- uses an Iterator behind the scenes!
next time...
-reading
  - chapters 8.1 - 8.4

-homework
  - assignment 2 due tomorrow at 11:59pm
  - assignment 3 out today, due next Thursday