TAFTD (Take Aways for the Day)

- Graded work this week:
  - APT Quiz, details and overview
  - Markov assignment, details and overview

- Concepts: Empirical and Analytical Analysis
  - Terminology, mathematics, analytical analyses

- Software Engineering: Unit Testing and Junit
  - Concepts and Practices
Empirical and Analytical Analysis

- We can run programs to look at "efficiency"
  - Depends on machine, environment, programs

- We can analyze mathematically to look at efficiency from a different point of view
  - Depends on being able to employ mathematics

- We will work on doing both, leading to a better understanding in many dimensions
Analytical Analysis

- **Since LinkedList is roughly linear**
  - Time to remove first element is constant, but must be done N times
  - Vocabulary, time for one removal is $O(1)$ --- constant and doesn't depend on N
  - Vocabulary, time for all removals is $O(N)$ -- linear in N, but slope doesn't matter

- **For ArrayList, removing first element entails ...**
  - Shifting N-1 elements, so this is $O(N)$

- **All**: $(N-1) + (N-2) + ... + 3 + 2 + 1 = O(N^2)$
  - Sum is $(N-1)N/2$
Interfaces

● What is an interface? What does Google say?
  ➢ Term overloaded even in English
  ➢ What is a Java Interface?

● Abstraction that defines a contract/construct
  ➢ Implementing requires certain methods exist
    • For example, Comparable interface?
  ➢ Programming to the interface is enabling
    • What does Collections.sort actually sort?

● IDE helps by putting in stubs as needed
  ➢ Let Eclipse be your friend
Why use Interfaces?

- **Implementation can vary without modifying code**
  - Code relies on interface, e.g., `addFront` or `removeMiddle`
  - Argument passed has a concrete type, but code uses the interface in compiling

- **Actual method called determined at runtime!**

- **Similar to API, e.g., using the Twitter API**
  - Calls return JSON, the format is specified, different languages used to interpret JSON
Markov Interlude: JUnit and Interfaces

- How do we design/code/test EfficientMarkov?
  - Note: it implements an Interface!
  - Note: MarkovTest can be used to test it!

- How do we design/code/test WordGram?
  - Can we use WordGram tester when first cloned?
  - Where is implementation of WordGram?
  - How do you make your own?
JUnit tests

- To run these must access JUnit library, jar file
  - Eclipse knows where this is, but ...
  - Must add to build-path aka class-path, Eclipse will do this for you if you let it

- Getting all green is the goal, but red is good
  - You have to have code that doesn't pass before you can pass
  - Similar to APTs, widely used in practice

- Testing is extremely important in engineering!
  - See also QA: quality assurance
JUnit Interlude

- Looking at PointExperiment classes:
  - https://git.cs.duke.edu/201fall16/pointExperiment/tree/master/src

- Create JUnit tests for some methods, see live run through and summary

- JUnit great for per-method testing in isolation from other methods
Remove Middle Index

```java
public double removeMiddleIndex(List<String> list) {
    double start = System.nanoTime();
    while (list.size() != 1) {
        list.remove(list.size() / 2);
    }
    double end = System.nanoTime();
    return (end - start) / 1e9;
}
```

● What operations could be expensive here?
   ➢ Explicit: size, remove (only one is expensive)
   ➢ Implicit: find \( n^{th} \) element
Remove Middle 2011

<table>
<thead>
<tr>
<th>size</th>
<th>link</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.105</td>
<td>0.023</td>
</tr>
<tr>
<td>20</td>
<td>0.472</td>
<td>0.09</td>
</tr>
<tr>
<td>30</td>
<td>0.984</td>
<td>0.192</td>
</tr>
<tr>
<td>40</td>
<td>1.83</td>
<td>0.343</td>
</tr>
<tr>
<td>50</td>
<td>3.026</td>
<td>0.534</td>
</tr>
<tr>
<td>60</td>
<td>4.288</td>
<td>0.767</td>
</tr>
<tr>
<td>70</td>
<td>6.078</td>
<td>1.039</td>
</tr>
<tr>
<td>80</td>
<td>7.885</td>
<td>1.363</td>
</tr>
</tbody>
</table>
Remove Middle 2016

<table>
<thead>
<tr>
<th>size</th>
<th>link</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.0635</td>
<td>0.0057</td>
</tr>
<tr>
<td>20</td>
<td>0.2644</td>
<td>0.0131</td>
</tr>
<tr>
<td>30</td>
<td>0.4808</td>
<td>0.0345</td>
</tr>
<tr>
<td>40</td>
<td>0.8524</td>
<td>0.0531</td>
</tr>
<tr>
<td>50</td>
<td>1.4025</td>
<td>0.0844</td>
</tr>
<tr>
<td>60</td>
<td>1.8418</td>
<td>0.1245</td>
</tr>
<tr>
<td>70</td>
<td>2.9064</td>
<td>0.1777</td>
</tr>
<tr>
<td>80</td>
<td>3.7237</td>
<td>0.2224</td>
</tr>
<tr>
<td>90</td>
<td>4.6833</td>
<td>0.3102</td>
</tr>
<tr>
<td>100</td>
<td>7.8717</td>
<td>0.3824</td>
</tr>
</tbody>
</table>
ArrayList and LinkedList as ADTs

● As an ADT (abstract data type) ArrayList supports
  ➢ Constant-time or O(1) access to the k-th element
  ➢ Amortized linear or O(n) storage/time with add
    • Total storage used in n-element vector is approx. 2n, spread over all accesses/additions (why?)
  ➢ Add/remove in middle is "expensive" O(n), why?

● What's underneath here? How Implemented?
  ➢ Concrete: array – contiguous memory, must be contiguous to support random access
  ➢ Element 20 = beginning + 20 x size of a pointer
ArrayList and LinkedList as ADTs

- LinkedList as ADT
  - Constant-time or O(1) insertion/deletion anywhere, but...
  - Linear or O(n) time to find where, sequential search

- Linked good for add/remove at front
  - Splicing into middle, also for 'sparse' structures

- What's underneath? How Implemented
  - Low-level linked lists, self-referential structures
  - More memory intensive than array: two pointers
Remove Middle in Pictures

- **Find middle element**: happens instantly or $O(1)$
  - $\text{alist(location)} + n/2 \times \text{sizeof(pointer)}$ since ArrayList holds pointers

- **Shifting requires moving $n/2$ pointers, but they are all contiguous in memory**: cache performance
Remove Middle in Pictures

- **Find middle element**: have to follow pointers between elements
  - Follow n/2 pointers, but all over memory, so takes time to move from memory->cache->use

- **Removing middle**: instantaneous, no shifting, just re-assign a couple of pointers (back pointers too)
  - Blue points to Yellow
Inheritance and Interfaces

● Interfaces provide method names and parameters
  ➢ The method signature we can expect and use!
  ➢ What can we do to an ArrayList? To a LinkedList?
  ➢ What can we do to a Map or Set or a MarkovInterface?
  ➢ java.util.Collection is an interface

● New in Java 8: Interfaces can have code!
Nancy Leveson: Software Safety

Founded the field

- Mathematical and engineering aspects
  - Air traffic control
  - Microsoft word

"C++ is not state-of-the-art, it's only state-of-the-practice, which in recent years has been going backwards"

- Software and steam engines once deadly dangerous?
  - [http://sunnyday.mit.edu/steam.pdf](http://sunnyday.mit.edu/steam.pdf)

- THERAC 25: Radiation machine killed many people
Analytical Analysis

- Creating random text in Markov takes time proportional to TN where T is #characters generated randomly and N is size of text
  - Rescan text for follows each time: BruteMarkov
  - We say this is "order NT" or $O(NT)$

- For EfficientMarkov, replace N with constant time map.get --- independent of N or $O(1)$
  - So generating random text is $TxO(1)$ or $O(T)$
Big-Oh, O-notation: concepts & caveats

- **Count how many times “simple” statements execute**
  - In the body of a loop, what matters? (e.g., another loop?)
  - Assume statements take a second, cost a penny?
    - What's good, what's bad about this assumption?

- **If a loop is inside a loop:**
  - Tricky because the inner loop can depend on the outer, use math and reasoning

- **In real life:** cache behavior, memory behavior, swapping behavior, library gotchas, things we don’t understand,…
More on O-notation, big-Oh

- Big-Oh hides/obscures some empirical analysis, but is good for general description of algorithm
  - Allows us to compare algorithms in the limit
  - $20N$ hours vs $N^2$ microseconds: which is better?

- O-notation is an upper-bound, this means that $N$ is $O(N)$, but it is also $O(N^2)$; we try to provide tight bounds (see next slide)
More on O-notation, big-Oh

- O-notation is an upper-bound, this means that $N$ is $O(N)$, but it is also $O(N^2)$; we try to provide tight bounds. Formally:
  - A function $g(N)$ is $O(f(N))$ if there exist constants $c$ and $n$ such that $g(N) < cf(N)$ for all $N > n$
Notations for measuring complexity

- **O-notation/big-Oh**: $O(n^2)$ is used in algorithmic analysis, e.g., Compsci 330 at Duke. Upper bound in the limit
  - Correct to say that linear algorithm is $O(n^2)$, but useful?

- **Omega** is lower bound: $\Omega(n \log n)$ is a lower bound for comparison based sorts
  - Can't do better than that, a little hard to prove
  - We can still engineer good sorts: TimSort!
Simple examples of array/loops: O?

```java
for(int k=0; k < list.length; k += 1) {
    list[k] += 1;   // list.set(k, list.get(k)+1);
}
//-----
```

```java
for(int k=0; k < list.length; k += 1)
    for(int j=k+1; j < list.length; j += 1)
        if (list[j].equals(list[k]))
            matches += 1;
//---
```

```java
for(int k=0; k < list.length; k += 1)
    for(int j=k+1; j < list.length; j *= 2)
        value += 1;
```
Loops explained

- Let N be the # elements in list
  - Loop iterates N times
  - Each time does O(1) work – not dependent on N
- Complexity of code or runtime analysis is: O(N)

```
for(int k=0; k < list.length; k += 1) {
    list[k] += 1;
}
```
Loops explained II

- Let N be the # elements in list
  - Outer loop iterates N times
  - Each time does the work of the inner loop
- Inner loop statement is O(1), the inner loop iterates exactly N-(k+1) times, so inner most statement:
  - \((N-1) + (N-2) + \ldots + 2 + 1 = O(N^2)\)

```java
for(int k=0; k < list.length; k += 1)
    for(int j=k+1; j < list.length; j += 1)
        if (list[j].equals(list[k]))
            matches += 1;
```
Loops explained III

● Let N be the # elements in list
  - Outer loop iterates N times
  - Each time does the work of the inner loop
● Inner loop statement is O(1), the inner loop iterates exactly \( \log_2(N-(k+1)) \) times
  - \( \log_2(N) \times N \) is an upper bound, \( O(N \log N) \)
  - \( \log(N-1) + \log(N-2) + \ldots + \log(1) = \log((N-1)!) = \log(N \log N) \)
● http://stackoverflow.com/questions/2095395/is-logn-%CE%98n-%CE%98n-logn

```java
for(int k=0; k < list.length; k += 1)
    for(int j=k+1; j < list.length; j *= 2)
        value += 1;
```
Big-O questions

http://bit.ly/201fall16-sept30-1

- How do check work? Look for patterns? Use concrete values for N
Multiplying and adding big-Oh

- Suppose we do a linear search then do another one
  - What is the complexity? $O(n) + O(n)$
  - If we do 100 linear searches? $100\cdot O(n)$
  - If we do $n$ searches on an array of size $n$? $n \cdot O(n)$

- Binary search followed by linear search?
  - What are big-Oh complexities? Sum?
  - What about 50 binary searches? What about $n$ searches?
What is big-Oh about?

- **Intuition:** avoid details when they don’t matter, and they don’t matter when input size (N) is big enough
  - Use only leading term, ignore coefficients
    - \( y = 3x \)
    - \( y = 6x - 2 \)
    - \( y = 15x + 44 \)
    - \( y = x^2 \)
    - \( y = x^2 - 6x + 9 \)
    - \( y = 3x^2 + 4x \)

- **The first family is** \( O(n) \), **the second is** \( O(n^2) \)
  - Intuition: family of curves, generally the same shape
  - Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time
Some helpful mathematics

● \(1 + 2 + 3 + 4 + ... + N\)
  - \(N(N+1)/2, \text{ exactly } = N^2/2 + N/2 \text{ which is } O(N^2) \text{ why?}\)

● \(N + N + N + ... + N \text{ (total of } N \text{ times)}\)
  - \(N*N = N^2 \text{ which is } O(N^2)\)

● \(N + N + N + ... + N + ... + N + ... + N \text{ (total of } 3N \text{ times)}\)
  - \(3N*N = 3N^2 \text{ which is } O(N^2)\)

● \(1 + 2 + 4 + ... + 2^N\)
  - \(2^{N+1} - 1 = 2 \times 2^N - 1 \text{ which is } O(2^N) \text{ - in terms of last term, call it } X, \text{ this is } O(X)\)