Plan for the week

- Comparing objects
- Towards algorithm analysis
- Review

- Assignment
  - Markov
- Lab: Review for Midterm

From Comparable to Comparator

- When a class implements `Comparable` then ...
  - Instances are comparable to each other
    - "apple" < "zebra", 6 > 2
  - Sorting Strings, Sorting WordNgrams, ...
  - Method `compareTo` invoked when ...
  - `Comparable<...>` types the parameter to `compareTo`
  - Return < 0, == 0, > 0 according to results of comparison

- Suppose we want to change how Strings compare
  - Or change class `Foo` implements `Comparable<Foo>`
  - What if we need more than one way to compare Foo’s?

java.util.Comparator

- How does sorting work in general and in Java?
  - Characteristics of Java library sort methods
  - What can be sorted?
  - How do you change how sorting works?
- APT ClientsList: example to explore Comparator
  - Creating new Comparator: nested class
    - Should it be public? Private? Matter?
  - Comparator could anonymous, but then issues.

- What does it mean to `implement Comparable`?
  - Other Java interfaces: cloneable, serializable, ...

Dropping Glass Balls

- Tower with N Floors
- Given 2 glass balls
- Want to determine the `lowest` floor from which a ball can be dropped and will break
- How?

- What is the most efficient algorithm?
- How many drops will it take for such an algorithm (as a function of N)?
Glass balls revisited (more balls)

- Assume the number of floors is 100
- In the best case how many balls will I have to drop to determine the lowest floor where a ball will break?
  1. 1
  2. 2
  3. 10
  4. 16
  5. 17
  6. 18
  7. 20
  8. 21
  9. 51
  10. 100

If there are $n$ floors, how many balls will you have to drop? (roughly)

- In the worst case, how many balls will I have to drop?
  1. 1
  2. 2
  3. 10
  4. 16
  5. 17
  6. 18
  7. 20
  8. 21
  9. 51
  10. 100

What is big-Oh about? (preview)

- Intuition: avoid details when they don’t matter, and they don’t matter when input size ($N$) is big enough
  - For polynomials, use only leading term, ignore coefficients
    \[
    y = 3x \quad y = 6x - 2 \quad y = 15x + 44 \\
    y = x^2 \quad y = x^2 - 6x + 9 \quad y = 3x^2 + 4x
    \]
  - The first family is $O(n)$, the second is $O(n^2)$
    - Intuition: family of curves, generally the same shape
    - More formally: $O(f(n))$ is an upper-bound, when $n$ is large enough the expression $cf(n)$ is larger
    - Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time

More on O-notation, big-Oh

- Big-Oh hides/obscures some empirical analysis, but is good for general description of algorithm
  - Intuition: 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. Formally:
    - A function $g(N)$ is $O(f(N))$ if there exist constants $c$ and $n$ such that $g(N) < cf(N)$ for all $N > n$
    - $g(N)$

Which graph is “best” performance?
Big-Oh calculations from code

• Search for element in an array:
  - What is complexity of code (using O-notation)?
  - What if array doubles, what happens to time?

```java
for(int k=0; k < a.length; k++) {
    if (a[k].equals(target)) return true;
} return false;
```

• Complexity if we call N times on M-element vector?
  - What about best case? Average case? Worst case?

Amortization: Expanding ArrayLists

• Expand capacity of list when add() called
• Calling add N times, doubling capacity as needed

<table>
<thead>
<tr>
<th>Item #</th>
<th>Resizing cost</th>
<th>Cumulative cost</th>
<th>Resizing Cost per item</th>
<th>Capacity After add</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3-4</td>
<td>4</td>
<td>6</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>5-8</td>
<td>8</td>
<td>14</td>
<td>1.75</td>
<td>8</td>
</tr>
</tbody>
</table>

2^m+1 - 2^m+1
2^m+2-2 around 2 2^m+1

• Big-Oh of adding n elements?
• What if we grow size by one each time?

Some helpful mathematics

• 1 + 2 + 3 + 4 + ... + N
  - N (N+1)/2, exactly = N^2/2 + N/2 which is O(N^2)  why?
• N + N + N + ... + N (total of N times)
  - N*N = N^2 which is O(N^2)
• N + N + N + ... + N + ... + N + ... + N (total of 3N times)
  - 3N*N = 3N^2 which is O(N^2)
• 1 + 2 + 4 + ... + 2^N
  - 2^{m+1} - 1 = 2 * 2^m - 1 which is O(2^N)

• Impact of last statement on adding 2^N+1 elements to a vector
  - 1 + 2 + ... + 2^N + 2^{N+1} = 2^{m+2} - 1 = 4x2^N - 1 which is O(2^N)

resizing + copy = total (let x = 2^N)

Running times @ 10^6 instructions/sec

<table>
<thead>
<tr>
<th>N</th>
<th>O(log N)</th>
<th>O(N)</th>
<th>O(N log N)</th>
<th>O(N^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.000003</td>
<td>0.0001</td>
<td>0.000033</td>
<td>0.0001</td>
</tr>
<tr>
<td>100</td>
<td>0.000007</td>
<td>0.0010</td>
<td>0.000664</td>
<td>0.1000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.00010</td>
<td>0.0100</td>
<td>0.010000</td>
<td>1.0</td>
</tr>
<tr>
<td>10,000</td>
<td>0.00013</td>
<td>0.0100</td>
<td>0.132900</td>
<td>1.7 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.00017</td>
<td>0.1000</td>
<td>1.661000</td>
<td>2.78 hr</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.00020</td>
<td>1.0</td>
<td>19.9</td>
<td>11.6 day</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.00030</td>
<td>16.7 min</td>
<td>18.3 hr</td>
<td>318 centuries</td>
</tr>
</tbody>
</table>