Announcements

1. HW3 posted... have you started?
2. Tutor hours: VERY CROWDED near deadline
3. Tutor hours rules:
   1. Your code must be indented properly or the tutor will not help you (and you will have to go to the end of the list!)
   2. If you put your name in the queue more than once ALL of your entries will be removed
Let $f(n) = 2^n + 14n^2 + 4n^3$

Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^3)$
E. Other/none/more
Let\( f(n) = 100 \)

Which of the following is true?

A. \( f(n) = O(2^n) \)
B. \( f(n) = O(n^2) \)
C. \( f(n) = O(n) \)
D. \( f(n) = O(n^{100}) \)
E. Other/none/more
Extracting time cost from example code

Algorithm analysis starting with the algorithm
A student has counted how many times we perform each line of code.

Is the count $3n+5$:

A. the best case?
B. the worst case?
C. the average case?
D. Other/none/more

<table>
<thead>
<tr>
<th>Statements</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 float findAvg ( int []grades ){</td>
<td></td>
</tr>
<tr>
<td>2 float sum = 0;</td>
<td>1</td>
</tr>
<tr>
<td>3 int count = 0;</td>
<td>1</td>
</tr>
<tr>
<td>4 while ( count &lt; grades.length ) {</td>
<td>$n + 1$</td>
</tr>
<tr>
<td>5 sum += grades[count];</td>
<td>$n$</td>
</tr>
<tr>
<td>6 count++;</td>
<td>$n$</td>
</tr>
<tr>
<td>7 }</td>
<td></td>
</tr>
<tr>
<td>8 if ( grades.length &gt; 0 )</td>
<td>1</td>
</tr>
<tr>
<td>9 return sum / grades.length;</td>
<td></td>
</tr>
<tr>
<td>10 else</td>
<td></td>
</tr>
<tr>
<td>11 return 0.0f;</td>
<td></td>
</tr>
<tr>
<td>12 }</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>$3n+5$</td>
</tr>
</tbody>
</table>
A student has counted how many times we perform each line of code.

Do you need to count the while loop header?

A. Yes
B. No

<table>
<thead>
<tr>
<th>Statements</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 float findAvg ( int []grades ){</td>
<td></td>
</tr>
<tr>
<td>2 float sum = 0;</td>
<td>1</td>
</tr>
<tr>
<td>3 int count = 0;</td>
<td>1</td>
</tr>
<tr>
<td>4 while ( count &lt; grades.length ) {</td>
<td>( n + 1 )</td>
</tr>
<tr>
<td>5 \quad \text{sum += grades[count];}</td>
<td>( n )</td>
</tr>
<tr>
<td>6 \quad \text{count++;}</td>
<td>( n )</td>
</tr>
<tr>
<td>7 }</td>
<td></td>
</tr>
<tr>
<td>8 \quad \text{if ( grades.length &gt; 0 )}</td>
<td>1</td>
</tr>
<tr>
<td>9 \quad \text{return sum / grades.length;}</td>
<td></td>
</tr>
<tr>
<td>10 \quad \text{else}</td>
<td>1</td>
</tr>
<tr>
<td>11 \quad \text{return 0.0f;}</td>
<td></td>
</tr>
<tr>
<td>12 }</td>
<td></td>
</tr>
</tbody>
</table>

ALL \( 3n+5 \)

HW3: Counting the header is up to you.
Count how many times each line executes, then which \( O(\) ) most tightly and correctly characterizes the growth?

\[
\begin{align*}
\text{int maxDifference(int[]} \ arr) \{ & \\
\text{max = 0;} & 1 \\
\text{for (int i=0; i<arr.length; i++)} \{ & 2 \\
\text{for (int j=0; j<arr.length; j++)} \{ & 3 \\
\text{if (arr[i] - arr[j] > max)} & 4 \\
\text{max = arr[i] - arr[j];} & 5 \\
\} & 6 \\
\} & 7 \\
\text{return max;} & 8 \\
\}
\end{align*}
\]

A. \( f(n) = O(2^n) \)  
B. \( f(n) = O(n^2) \)  
C. \( f(n) = O(n) \)  
D. \( f(n) = O(n^3) \)  
E. Other/none/more  
(assume \( n = arr.length \))
Count how many times each line executes, then say which $O(\cdot)$ statement(s) is(are) true.

```java
int maxDifference(int[] arr) {
    max = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            if (arr[i] - arr[j] > max)
                max = arr[i] - arr[j];
        }
    }
    return max;
}
```

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^3)$
E. Other/none/more
   (assume $n = \text{arr.length}$)
Count how many times each line executes, then say which $O(\ )$ statement(s) is(are) true.

```java
int maxDifference(int[] arr) {
    max = 0;
    for (int i=0; i<arr.length; i++) {
        for (int j=0; j<arr.length; j++) {
            if (arr[i] - arr[j] > max)
                max = arr[i] - arr[j];
        }
    }
    return max;
}
```

A. $f(n) = \Theta(2^n)$  
B. $f(n) = \Theta(n^2)$  
C. $f(n) = \Theta(n)$  
D. $f(n) = \Theta(n^3)$  
E. Other/none/more  

(assume $n = arr.length$)
### Some Common Cost Function Classes

<table>
<thead>
<tr>
<th>$\log_2 n$</th>
<th>$n$</th>
<th>$n \log_2 n$</th>
<th>$n^2$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>65,536</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1,024</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4,096</td>
<td>$1.84 \times 10^{19}$</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>896</td>
<td>16,384</td>
<td>$3.40 \times 10^{38}$</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>2,048</td>
<td>65,536</td>
<td>$1.16 \times 10^{77}$</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>4,608</td>
<td>262,144</td>
<td>$1.34 \times 10^{154}$</td>
</tr>
<tr>
<td>10</td>
<td>1,024</td>
<td>10,240</td>
<td>1,048,576</td>
<td>$1.80 \times 10^{308}$</td>
</tr>
</tbody>
</table>

Facebook users: 1,060,000,000

5GHz = $5 \times 10^9$ cycles per second

$6.4\text{ s} \quad 224,720,000\text{ s (or 7.12 years)}$
Benchmarking code
An alternative/supplement to big-O style analysis
Basic procedure for benchmarking

for problem size $N = \text{min, ... max}$
1. initialize the data structure
2. get the current (starting) time
3. run the algorithm on problem size $N$
4. get the current (finish) time
5. timing = finish time – start time
Timer methods in Java

- Java has two static methods in the System class:
  ```java
  /** Returns the current time in milliseconds. */
  static long System.currentTimeMillis()

  /** Returns the current value of the most precise available system timer, in nanoseconds */
  static long System.nanoTime()
  
  - If the algorithm can take less than a millisecond to run, you should use System.nanoTime()!
One way to improve the accuracy of measuring the running time of an algorithm is by

A. Using a faster computer
B. Averaging the measurement results
C. Starting up more applications to have a larger set of running applications
D. Improving the time the algorithm takes to finish executing
IMPROVED procedure for benchmarking

for problem size $N = \text{min, ... max}$

1. initialize the data structure
2. for $K$ runs:
   1. get the current (starting) time
   2. run the algorithm on problem size $N$
   3. get the current (finish) time
   4. $\text{timing} = \text{finish time} - \text{start time}$
3. Average timings for $K$ runs for this $N$
Explain these results...

Timings for `findMax()` on an array of `int`s and an array of `Integer`s (times are in milliseconds)

<table>
<thead>
<tr>
<th>n</th>
<th>array of int</th>
<th>array of Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>800,000</td>
<td>2.314</td>
<td>4.329</td>
</tr>
<tr>
<td>4,000,000</td>
<td>11.363</td>
<td>21.739</td>
</tr>
<tr>
<td>8,000,000</td>
<td>22.727</td>
<td>42.958</td>
</tr>
</tbody>
</table>

From these measurements, what is the likely big-O time cost of `findMax()` on array of `int`?

A. \( f(n) = O(\log_2 n) \)       D. \( f(n) = O(n) \)
B. \( f(n) = O(n \log_2 n) \)    E. Other/none/more
C. \( f(n) = O(n^2) \)
Explain these results...

Timings for `findMax()` on an array of `int`s and an array of `Integer`s (times are in milliseconds)

<table>
<thead>
<tr>
<th>n</th>
<th>array of int</th>
<th>array of Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>800,000</td>
<td>2.314</td>
<td>4.329</td>
</tr>
<tr>
<td>4,000,000</td>
<td>11.363</td>
<td>21.739</td>
</tr>
<tr>
<td>8,000,000</td>
<td>22.727</td>
<td>42.958</td>
</tr>
</tbody>
</table>

From these measurements, what is the likely big-O time cost of `findMax()` on array of `Integer`?

A. $f(n) = O(\log_2 n)$
B. $f(n) = O(n\log_2 n)$
C. $f(n) = O(n^2)$
D. $f(n) = O(n)$
E. Other/none/more
Explain these results...

Timings for `findMax()` on an array of `int`s and an array of `Integer`s (times are in milliseconds)

<table>
<thead>
<tr>
<th>$n$</th>
<th>array of int</th>
<th>array of Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>800,000</td>
<td>2.314</td>
<td>4.329</td>
</tr>
<tr>
<td>4,000,000</td>
<td>11.363</td>
<td>21.739</td>
</tr>
<tr>
<td>8,000,000</td>
<td>22.727</td>
<td>42.958</td>
</tr>
</tbody>
</table>

Discussion:
- Why would `Integer` take more time?
- If `Integer` takes more time, why does it have the same Big-O cost as `int`?
Benchmarking pitfalls (Demo)

- Not running enough reps
- Running on too small a problem
- Strange behavior with the first run