Last Time

- ThreeSum.java
- ThreeSumWithTiming.java
- ThreeSumFast.java

- Binary Search
- Calculating theoretical runtime
- Timing program execution
What Can Complicate Our Analysis?

• Large constants
• Non-dominant loops
• Instruction time
• System considerations
• Too close to call
• Dependence on inputs
• Multiple problem parameters
Large Constants

• I told you to ignore lower-order terms...
  
  $\frac{1}{2} n^2 + cn = \sim \frac{1}{2} n^2$

  - What if $c = 1000$, or $c = 1000000$, or $c = 10^{100}$?
    • It’s still better than a $\sim \frac{n^2}{2}$ function for a large enough $n$...
  
  - Consider a program that runs in linear time, vs a program that does the same thing in constant time. But the constant is $1000000$...
    • Constant is still better with a large enough $n$...

• If the constant is unknown it’s *generally* safe to ignore.
Non-dominant Loops

```c
int sum = 0;
for (int i = 1; i < n; i *= 2) {
    for (int j = 0, j < i; j++) {
        sum++;
    } //for
} //for
```

**Geometric sum:** $1 + 2 + 4 + 8 + \cdots + n = 2n - 1 \sim 2n$

**Harmonic sum:** $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{n} = \ln(n) \sim \log(n)$
Instruction Time

- We cannot assume that every instruction takes the same amount of time to execute. (Remember the $t_0$ through $t_4$ variables from the table?)

```c
int sum = 2, product = 2, i;
for (i = 0; i < n; i++) {
    sum++;
} //for

for (i = 0; i < n; i++) {
    product = product * product;
} //for
```
System Considerations

![Windows Task Manager](image)

<table>
<thead>
<tr>
<th>Image Name</th>
<th>PID</th>
<th>User Name</th>
<th>CPU</th>
<th>CPU Time</th>
<th>Memory (Pr.)</th>
<th>Page Faults</th>
<th>PF Delta</th>
<th>Threads</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firefox.exe *32</td>
<td>11996</td>
<td>John</td>
<td>11</td>
<td>06:28:18</td>
<td>1,679,036 K</td>
<td>339,052,647</td>
<td>9,870</td>
<td>94</td>
<td>Firefox</td>
</tr>
<tr>
<td>BTSync.exe *32</td>
<td>4884</td>
<td>John</td>
<td>00</td>
<td>07:19:22</td>
<td>295,224 K</td>
<td>15,259,178</td>
<td>26</td>
<td>29</td>
<td>BitTorrent Sync</td>
</tr>
<tr>
<td>svhost.exe</td>
<td>1328</td>
<td>SYSTEM</td>
<td>00</td>
<td>04:07:36</td>
<td>201,968 K</td>
<td>10,148,472</td>
<td>0</td>
<td>0</td>
<td>Host Process for Windows Services</td>
</tr>
<tr>
<td>foo2bar.exe *32</td>
<td>7136</td>
<td>John</td>
<td>00</td>
<td>00:11:07</td>
<td>84,600 K</td>
<td>111,113</td>
<td>4</td>
<td>10</td>
<td>foobar2000</td>
</tr>
<tr>
<td>javaw.exe</td>
<td>2068</td>
<td>John</td>
<td>00</td>
<td>00:10:11</td>
<td>69,508 K</td>
<td>446,906</td>
<td>0</td>
<td>26</td>
<td>Java(TM) Platform SE binary</td>
</tr>
<tr>
<td>Dropbox.exe *32</td>
<td>1272</td>
<td>John</td>
<td>00</td>
<td>00:29:11</td>
<td>53,088 K</td>
<td>9,778,617</td>
<td>0</td>
<td>97</td>
<td>Dropbox</td>
</tr>
<tr>
<td>dwm.exe</td>
<td>3820</td>
<td>John</td>
<td>00</td>
<td>01:41:15</td>
<td>45,852 K</td>
<td>4,077,465</td>
<td>0</td>
<td>5</td>
<td>Desktop Window Manager</td>
</tr>
<tr>
<td>POWERPNT.EXE *32</td>
<td>10848</td>
<td>John</td>
<td>00</td>
<td>00:00:18</td>
<td>42,180 K</td>
<td>283,244</td>
<td>0</td>
<td>13</td>
<td>Microsoft PowerPoint</td>
</tr>
<tr>
<td>googledrivesync.exe *32</td>
<td>5616</td>
<td>John</td>
<td>00</td>
<td>00:44:23</td>
<td>38,404 K</td>
<td>457,979</td>
<td>0</td>
<td>29</td>
<td>Google Drive</td>
</tr>
<tr>
<td>explorer.exe</td>
<td>2488</td>
<td>John</td>
<td>00</td>
<td>00:34:00</td>
<td>37,716 K</td>
<td>84,605,508</td>
<td>2</td>
<td>34</td>
<td>Windows Explorer</td>
</tr>
<tr>
<td>aim.exe *32</td>
<td>4692</td>
<td>John</td>
<td>00</td>
<td>04:40:56</td>
<td>37,440 K</td>
<td>560,909</td>
<td>0</td>
<td>48</td>
<td>AOL Instant Messenger</td>
</tr>
<tr>
<td>Steam.exe *32</td>
<td>4980</td>
<td>John</td>
<td>00</td>
<td>00:30:07</td>
<td>28,056 K</td>
<td>8,798,280</td>
<td>7</td>
<td>27</td>
<td>Steam Client Bootstrapper</td>
</tr>
<tr>
<td>svhost.exe</td>
<td>1384</td>
<td>SYSTEM</td>
<td>00</td>
<td>08:49:18</td>
<td>25,400 K</td>
<td>42,276,929</td>
<td>3</td>
<td>47</td>
<td>Host Process for Windows Services</td>
</tr>
<tr>
<td>FlashPlayerPlugin_20_0_...</td>
<td>12868</td>
<td>John</td>
<td>00</td>
<td>00:17:09</td>
<td>23,328 K</td>
<td>820,983</td>
<td>0</td>
<td>15</td>
<td>Adobe Flash Player 20.0 r0</td>
</tr>
<tr>
<td>miktex-texworks.exe *32</td>
<td>6812</td>
<td>John</td>
<td>00</td>
<td>00:00:19</td>
<td>20,208 K</td>
<td>169,044</td>
<td>0</td>
<td>5</td>
<td>TeXworks</td>
</tr>
</tbody>
</table>

Processes: 127  CPU Usage: 20%  Physical Memory: 81%
System Considerations

• Caching
• Java garbage collector
• Internet download
• Just-in-time compiler

• Scientific method: “Experiments should be reproducible!”
  – What is happening in your system during a program run is unlikely to be produced again.
  – Other system processes should in principle be negligible or under your control.
Multiple Problem Parameters

- It is not unusual to have more than one input parameter.
  - Example: an algorithm that builds a data structure, then performs operations on that data structure.
    - Parameters: size of the structure (m), number of operations on the structure (n).
    - Is there a difference between the runtime $O(m \times \log(n))$ and $O(n \times \log(m))$?
      - What if $n = 2^m$?
Too Close to Call

• Program A is faster on Input 1, but Program B is faster on Input 2. Which is the better algorithm?
• It depends...
• Don’t devote an extreme amount of energy towards finding the “best” implementation.
  – Order of growth is most important to our analysis in this class.
Dependence on Inputs

```java
foreach item in items {
    for (i = item, i < 100; i++) {
        System.out.println(i);
    } //for
} //foreach
```

- Input 1: 99, 115, 107, 98, 100, 101, 110, 111
- Input 2: 7, 1, 12, 16, 17, 21, -8325827
Dependence on Inputs

• We cannot *always* make the assumption that running time will be insensitive to inputs.

• Consider a modified version of ThreeSum, which reports true/false whether or not a triple exists, rather than counting them.
  – If the first three numbers are a triple, the runtime will be $O(1)$.
  – If the last three numbers are a triple, the runtime will be $O(n^3)$.
Memory

• Analyzing memory usage is simultaneously easier and harder than analyzing time complexity.
  – Every integer is 4 bytes.
  – Every double is 8 bytes.
  – Every String is...... well......

• As an additional complication, Java’s memory allocation system is both implementation dependent and hardware dependent.
Memory and Objects

- Objects typically have an overhead of 16 bytes.
- Memory usage is padded to a multiple of 8 bytes. (why?)
- Things get more complicated when we delve into data structures.
Memory and Arrays

• An array of n integers needs 24 bytes of overhead (16 for the object, 4 for the length, 4 for padding before the data).
  – Total memory requirement = 24+4n, rounded up to the next multiple of 8, so ~4n. (for doubles?)

• A two-dimensional array (array of arrays) of integers of size $M \times N$ needs $\sim 4MN$:
  – 24 bytes for the overall overhead,
  – $8M$ bytes of overhead referencing each of the row arrays,
  – $24M$ bytes of overhead for each of the row arrays,
  – $4MN$ bytes for each integer (with possibly more padding).
Memory and Strings

• Java 7 and later: (ALERT error in book page 202)
  – Two instance variables:
    • Reference to a char array,
    • An int value hash code.
  – A String of length n takes 56+2n bytes (16 object overhead, 8 reference to char[], 4 hash code, 4 padding, 24+2n for the char[] itself.

• Java 6 and earlier:
  – As above, but with two extra integer variables, offset and count, used to make substring computation more efficient. New total = 64+2n bytes.
Memory and Substrings

• In Java 7+, implementing the `substring()` method involves creating a new `char[]`, taking linear time and linear space.

• In Java 6-, we can alias the new substring using the original object – the `offset` and `count` fields identify the substring, taking constant time and constant space.

• Extracting a substring takes either constant or linear memory depending on the underlying implementation.
Any Questions?