Week 2 - Friday

# COMP 2100

# Last time

- What did we talk about last time?
- Java Collections Framework (JCF)
- Computational complexity
- Big Oh notation

# Assignment 1

# Assignment 2

# Project 1

# Questions?

# Back to complexity

# Mathematical issues

- What's the running time to factor a large number $N$?
- How many edges are in a completely connected graph?
- If you have a completely connected graph, how many possible tours are there (paths that start at a given node, visit all other nodes, and return to the beginning)?
- How many different $n$-bit binary numbers are there?

# Formal definition of Big Oh

- Let $f(n)$ and $g(n)$ be two functions over integers

- $f(n)$ is $O(g(n))$ if and only if
  - $f(n) \leq c \cdot g(n)$ for all $n > N$
  - for **some** positive real numbers $c$ and $N$

- In other words, past some arbitrary point, with some arbitrary scaling factor, $g(n)$ is always bigger

# What's the running time?

```
int count = 0;
for (int i = 0; i < n; i += 2) {
  for (int j = 0; j < n; j += 3) {
     ++count;
  }
}
```

# What's the running time?

```
int count = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    if (j == n - 1) {
        i = n;
    }
    ++count;
  }
}
```

# Hierarchy of complexities

- Here is a table of several different complexity measures, in ascending order, with their functions evaluated at ***n*** = 100

| Description | Big Oh | *f*(100) |
|---|---|---|
| Constant | $O(1)$ | 1 |
| Logarithmic | $O(\log n)$ | 6.64 |
| Linear | $O(n)$ | 100 |
| Linearithmic | $O(n \log n)$ | 664.39 |
| Quadratic | $O(n^2)$ | 10000 |
| Cubic | $O(n^3)$ | 1000000 |
| Exponential | $O(2^n)$ | $1.27 \times 10^{30}$ |
| Factorial | $O(n!)$ | $9.33 \times 10^{157}$ |

# What's log?

- The log operator is short for logarithm
- Taking the logarithm means **de-exponentiating** something

$$\log 10^7 = 7$$
$$\log 10^x = x$$

- What's the $\log 1{,}000{,}000$?

# What's the running time?

```
int count = 0;
for (int i = 1; i <= n; i *= 2) {
    ++count;
}
```

# Logarithms

- Formal definition:
  - If $b^x = y$
  - Then $\log_b y = x$ (for positive $b$ values)
- Think of it as a de-exponentiator
- Examples:
  - $\log_{10}(1{,}000{,}000) =$
  - $\log_3(81) =$
  - $\log_2(512) =$

# log base 2

- In the normal world, when you see a $\log$ without a subscript, it means the logarithm base 10
  - "What power do you have to raise **10** to to get this number?"
- In computer science, a $\log$ without a subscript usually means the logarithm base 2
  - "What power do you have to raise **2** to to get this number?"

$$\log 2^8 = 8$$
$$\log 2^y = y$$

- What's the $\log 2{,}048$? (Assuming $\log$ base 2)

# log math

- $\log_b(xy) = \log_b(x) + \log_b(y)$
- $\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$
- $\log_b(x^y) = y\log_b(x)$
- Base conversion:
  - $\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$
- As a consequence:
  - $\log_2(n) = \frac{\log_{10}(n)}{c_1} = \frac{\log_{100}(n)}{c_2} = \frac{\log_b(n)}{c_3}$ for $b > 1$
  - $\log_2 n$ *is* $O(\log_{10} n)$ and $O(\log_{100} n)$ and $O(\log_b n)$ for $b > 1$

# More on $\log$

- Add one to the logarithm in a base and you'll get the number of digits you need to represent that number in that base
- In other words, the $\log$ of a number is related to its **length**
  - Even big numbers have small logs
- If there's no subscript, $\log_{10}$ is assumed in math world, but $\log_2$ is assumed for CS
  - Also common is $\ln$, the natural log, which is $\log_e$

# Log is awesome

- As we said, the logarithm of the number is related to the number of digits you need to write it
- That means that the log of a very large number is still pretty small
- An algorithm that runs in $\log n$ time is very fast

| Number | $\log_{10}$ | $\log_2$ |
|---:|:---:|:---:|
| 1,000 | 3 | 10 |
| 1,000,000 | 6 | 20 |
| 1,000,000,000 | 9 | 30 |
| 1,000,000,000,000 | 12 | 40 |

# Big Oh, Big Omega, Big Theta

# Formal definition of Big Oh

- Let $f(n)$ and $g(n)$ be two functions over integers

- $f(n)$ is $O(g(n))$ if and only if
  - $f(n) \leq c \cdot g(n)$ for all $n > N$
  - for **some** positive real numbers $c$ and $N$

- In other words, past some arbitrary point, with some arbitrary scaling factor, $g(n)$ is always bigger

# Different kinds of bounds

- We've been sloppy so far, saying that something is $O(n)$ when its running time is proportional to $n$
- Big Oh is actually an **upper bound**, meaning that something whose running time is proportional to $n$ (like $42n + 7$)
  - Is $O(n)$
  - But is also $O(n^2)$
  - And is also $O(2^n)$
- If the running time of something is actually proportional to $n$, we should say it's $\Theta(n)$
- We often use Big Oh because it's easier to find an upper bound than to get a tight bound

# All three are useful measures

- $O$ establishes an upper bound
  - $f(n)$ is $O(g(n))$ if there exist positive numbers $c$ and $N$ such that $f(n) \leq c \cdot g(n)$ for all $n > N$
- $\Omega$ establishes a lower bound
  - $f(n)$ is $\Omega(g(n))$ if there exist positive numbers $c$ and $N$ such that $f(n) \geq c \cdot g(n)$ for all $n > N$
- $\Theta$ establishes a tight bound
  - $f(n)$ is $\Theta(g(n))$ if there exist positive numbers $c_1$, $c_2$, and $N$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n > N$

# Tight bounds

- O and Ω have a one-to-many relationship with functions
  - $4n^2 + 3$ is $O(n^2)$, but it's also $O(n^3)$ and $O(n^4 \log n)$
  - $6n \log n$ is $\Omega(n \log n)$ but it's also $\Omega(n)$
- Θ is one-to-many as well, but it has a much tighter bound
- Sometimes it's hard to find Θ
  - Upper bounding isn't too hard, but lower bounding is difficult for many real problems

# Facts

1. If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$
2. If $f(n)$ is $O(h(n))$ and $g(n)$ is $O(h(n))$, then $f(n) + g(n)$ is $O\big(h(n)\big)$
3. $an^k$ is $O(n^k)$
4. $n^k$ is $O(n^{k+j})$, for any positive $j$
5. If $f(n)$ is $cg(n)$, then $f(n)$ is $O\big(g(n)\big)$
6. $\log_a n$ is $O(\log_b n)$ for integers $a$ and $b > 1$
7. $\log_a n$ is $O(n^k)$ for integer $a > 1$ and real $k > 0$

# Binary search example

- Implement binary search
- How much time does a binary search take at most?
- What about at least?
- What about on average, assuming that the value is in the list?

# Complexity practice

- Give a tight bound for $n^{1.1} + n \log n$
- Give a tight bound for $2^{n+a}$ where $a$ is a constant
- Give functions $f_1$ and $f_2$ such that $f_1(n)$ and $f_2(n)$ are $O(g(n))$ but $f_1(n)$ is not $O(f_2(n))$

# Upcoming

# Next time…

- Abstract data types (ADTs)
- Bags and `ArrayList`

# Reminders

- **No class Monday!**
- Read section 1.3
- Finish Assignment 1
  - Due tonight by midnight!
- Start Assignment 2
  - Due next Friday by midnight
- Keep working on Project 1
  - Due Friday, September 12 by midnight