Algorithmic Analysis

Algorithms

- Algorithm
  - A *clear* and *concise* sequence of steps to solve a *class* of problems
- Every operation we've discussed in this class is an algorithm for modifying a data structure
- We have characterized these operations as fast , ok , and slow in terms of speed
  - You will now see what we've meant by these terms throughout the course

Characterizing Algorithms

- Algorithms can be characterized in terms of:
  - Runtime (speed)
  - Space
- Accurately doing this is difficult
  - Computers are different
  - We want to characterize these independent on hardware
  - We want to look at the worst case that is normally encountered
- Time as a function of size
  - How fast is an algorithm based on the size of the problem?
    - The definition is based on what the algorithm calls for
    - For data structures, the number of elements is the basis for the size of the program

Ram Model of Computation

- A very simple and crude way to approximate the speed of an algorithm is to count operations
  - Simple operations (arithmetic, assignment, memory access) take 1 unit of time
  - Loops, functions, etc are *not* simple operations
- Consider the following code:

```
for (int i=0; i<10; i++)
    System.out.println("hi");
```

How many operations are performed?

	10	increments
	10	prints
+	10	comparisons
	30	operations
	1	initialize i
+	1	comparison to terminate
	32	operations

If, instead of `i<10`, it was changed to `i<n`, then there would be $3n + 2$ operations.
This is more complicated than we need and should be simplified.
We want to know how the algorithm *scales*

Big Oh

- Complexity is denoted with Big-Oh notation: $O(f(n))$
  - big-oh of n
  - order n
- To go from the counted number of operations to big-oh notation, take only the largest term (in n) and drop the coefficients
  - $3n + 2 = O(n)$, $n^2 + 3n + 2 = O(n^2)$, etc

Examples

```
for i = 1 .. n:                                      -|
    for j = 1 .. n:                        -| * n     | * n
        k += i + j // 2 operations         -|        -|
```

$$2 * n * n = 2n^2 = O(n^2)$$

```
for i = 1 .. n:                                      -|
    for j = 1 .. 20                        -| * 20    | * n
        k += i + j // 2 operations         -|        -|
```

$$2 * 20 * n = 40n = O(n)$$

But Why Drop Small Terms?

- Consider $3n + 2$
  - $n = 10, \frac{2}{32} = 6.25\%$
  - $n = 100, \frac{2}{302} = .662\%$
- As n gets large, the small terms contribute much less to the runtime

Growth Rates

- Consider $O(n)$ and $O(n^2)$
  - $O(n)$: Worst time $= cn$
  - $O(n^2)$: Worst time $= cn^2$
- What if we double n?
  - $O(n)$: $c(2n) = 2cn$
  - $O(n^2)$: $c(2n)^2 = 4cn \leftarrow$ The problem size quadruples!
- What about $lg(n)$ (Worst time: $c * lg(n)$
  - $clg(2n) = clg(2) + clg(n) = c(1 + lg(n)) \leftarrow$ Grows slower than $O(n)$!

So What Does That Mean?

- Everything we have characterized as fast is $O(1)$
- Everything we have characterized as slow is $O(n)$
- Everything we have characterized as ok is $O(lg(n))$