Chapter 2 (Algorithms Analysis)

1. RAM model of computation
2. Best, Worst and Average-Case Complexity
3. Big-Oh Notation
4. Working with Asymptotics
5. Growth Rates and Dominance
6. Reasoning about Efficiency
7. Logarithms and their Applications

Discussion

Why do we have to learn all of this stuff?
Why not just use the actual run time/memory requirements to evaluate an algorithm?

RAM Model of Computation

- Assume that each simple statement (arithmetic op, memory op, assignment, etc) requires 1 unit of time per execution.
- Figure out how many times each simple statement is executed.
- Add up for all statements.
- Loops and function calls are NOT simple.
- This model is useful and accurate in the same sense as the flat-earth model (which *is* useful)!

Do Worksheet

- Pseudo-code for insertion sort.
- Three instances supplied.
- Seven distinct "statements", one on each line.
- Figure out how many times each statement is executed (touched).
- Add 'em up.
- A formula in terms of n?

Lessons learned from Worksheet

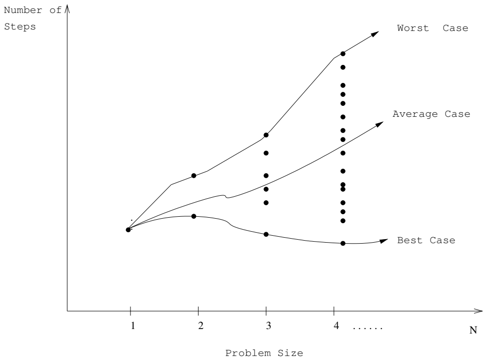
- # steps depends on n.
- # steps depends on input permutation, even for the same n.
- # steps would probably change slightly if INSERTION_SORT was written up differently.
- Do # steps predict algorithm's runtime with 100% accuracy? Identify some sources of inaccuracy.

Problem-specific metrics

- Very common in analysis of sorting or searching to count # data comparisons.
- Matrix multiplication: count # scalar multiplications.

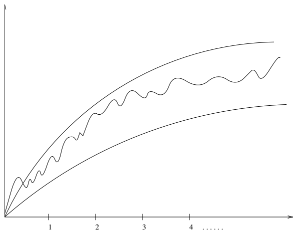
Worst-/Average-/Best-Case Complexity

The worst case complexity of an algorithm is the function defined by the maximum # steps taken on any instance of size n.
best-case complexity $\Rightarrow$ minimum
avg-case complexity $\Rightarrow$ average



Exact Analysis is Hard!

Best, worst, and average are difficult to deal with precisely because the details are very complicated:



So we talk about *upper* & *lower* bounds of the function. Enter ... the dreaded asymptotic notation!

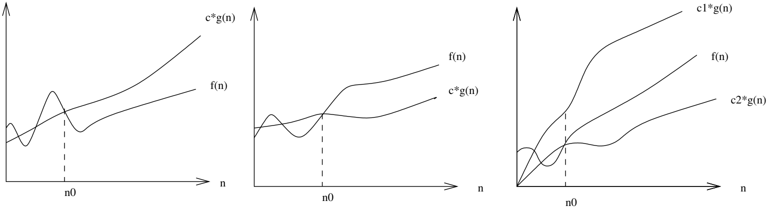
Definitions: O, Ω, Θ

1. $O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.
2. $\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.
3. $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0\}$.

Observations (bit painstaking!)

- All of the above define sets. Thus, $O(g(n))$ is a set of functions.
- If $f(n)$ belongs to this set, we should write it as $f(n) \in O(g(n))$.
- Instead, the convention is to write it as $f(n) = O(g(n))$.
- To show that $f(n)$ belongs to one of these sets, all we need to do is to find one set of constants that make the inequalities work.
- $n \geq n_0$ says we don't worry about what happens at lower values of n.

Graphical View



(a) (b) (c)

Bounding Functions View

I like to think of

- $f(n) = O(g(n))$ as meaning $f(n)$ "$\leq$" $g(n)$.
- $f(n) = \Omega(g(n))$ as meaning $f(n)$ "$\geq$" $g(n)$.
- $f(n) = \Theta(g(n))$ as meaning $f(n)$ "$=$" $g(n)$.

In-class Exercise 1

Show that $2n^2 + 3n = \Theta(n^2)$

- Rule of Thumb: To get the Θ of a function, just drop its lower order terms and constants.
- So, in the above example, drop the "2" and the "$3n$".

In-class Exercise 2

Figure out # steps in worst case for Insertion Sort (extrapolate from worksheet).

Answer:

$$f(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4 = \Theta(n^2)$$

Relationship between O, Ω, and Θ

Observe that one can also show for the above exercise that

1. $2n^2 + 3n = O(n^2)$
2. $2n^2 + 3n = \Omega(n^2)$

Theorem

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Lots of other theorems on asymptotics (see book).

Asymptotic Dominance in Action

n	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10^4	0.013 μs	10 μs	130 μs	100 ms		
10^5	0.017 μs	0.10 ms	1.67 ms	10 sec		
10^6	0.020 μs	1 ms	19.93 ms	16.7 min		
10^7	0.023 μs	0.01 sec	0.23 sec	1.16 days		
10^8	0.027 μs	0.10 sec	2.66 sec	115.7 days		
10^9	0.030 μs	1 sec	29.90 sec	31.7 years		

Implications of Dominance

- Exponential algorithms (2^n and $n!$) get hopeless fast.
- Quadratic algorithms (n^2) get hopeless at or before 1,000,000.
- $O(n \log n)$ is possible to about one billion.
- $O(\log n)$ never sweats.

Testing Dominance

- I implied earlier that we wouldn't need any calculus ... I lied ... a little.
- $f(n)$ dominates $g(n)$ if $\lim_{n\to\infty} g(n)/f(n) = 0$, which is the same as saying $g(n) = o(f(n))$.
- Note the little-oh – it means "grows strictly slower than".

Implications of Dominance

- n^a dominates n^b if $a > b$ since

$$\lim_{n\to\infty} n^b/n^a = n^{b-a} \to 0$$

- $n^a + o(n^a)$ doesn't dominate n^a since

$$\lim_{n\to\infty} n^a/(n^a + o(n^a)) \to 1$$

Dominance Rankings

You must internalize the dominance ranking of the basic functions:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

Advanced Dominance Rankings

Additional functions arise in more sophisticated analysis than we will do in this course:

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg$$
$$\log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

Animation Break

Reasoning About Efficiency

Grossly reasoning about the running time of an algorithm is usually easy given a precise-enough written description of the algorithm.
When you *really* understand an algorithm, this analysis can be done in your head.
However, recognize there is always implicitly a written algorithm/program we are reasoning about.

Selection Sort

```
selection_sort(int s[], int n) {
        int i,j;
        int min;

        for (i=0; i<n; i++) {
                min=i;
                for (j=i+1; j<n; j++)
                        if (s[j] < s[min]) min=j;
                swap(&s[i],&s[min]);
        }
}
```

Worst Case Analysis

The outer loop goes around n times.
The inner loop goes around at most n times for each iteration of the outer loop
Thus selection sort takes at most $n \times n \rightarrow O(n^2)$ time in the worst case.

More Careful Analysis

An exact count of the number of times the *if* statement is executed is given by:

$$S(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} (n - i - 1)$$

$$S(n) = (n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = n(n-1)/2$$

Thus the worst case running time is $\Theta(n^2)$.

Exercise

Is $(x+y)^2 = O(x^2+y^2)$?

Big-Oh examples

For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$, or none of the above.

- $f(n) = n^{100}, g(n) = 2^n$
- $f(n) = 5^{n+1}, g(n) = 5^n$
- $f(n) = \log_5 n!, g(n) = n \lg n$
- $f(n) = \log 2^n, g(n) = n$
- $f(n) = 100^n, g(n) = 10^n$

Logarithms

It is important to understand deep in your bones what logarithms are and how they end up being used in the analysis of algorithms.

A logarithm is simply an inverse exponential function. Saying $b^x = y$ is equivalent to saying that $x = \log_b y$.

Logarithms reflect how many times we can double something until we get to n, or halve something until we get to 1.

Binary Search

In binary search we throw away half the possible number of keys after each comparison.
Thus twenty comparisons suffice to find any name in the million-name Manhattan phone book!
How many times can we halve n before getting to 1?
Answer: $\lceil \lg n \rceil$.

Visual Demos

Logarithms and Trees

How tall a binary tree do we need until we have n leaves?
The number of potential leaves doubles with each level.
Starting with 1, how many times can we double until we get to n?
Answer: $\lceil \lg n \rceil$.

Logarithms and Bits

How many bits do you need to represent the numbers from 0 to $2^i - 1$?

Each bit you add doubles the possible number of bit patterns, so the number of bits equals $\lg(2^i) = i$.

Logarithms and Multiplication

Recall that

$$\log_a(xy) = \log_a(x) + \log_a(y)$$

This is how people used to multiply before calculators (been there done that), and remains useful for analysis.
What if $x = a$?

Log table simulation

The Base is not Asymptotically Important

Recall that

$$\log_b a = \frac{\log_c a}{\log_c b}$$

Thus,

$$\log_2 n = \frac{\log_{100} n}{\log_{100} 2}$$

$1/\log_{100} 2 = 6.643$ is just a constant.
So $\log_2$ vs. $\log_{10}$ doesn't matter in the Big Oh.

More log simulations

Logs of polynomial functions of n

Asymptotically, does not matter!
Note that

$$\log(n^{473} + n^2 + n + 96) = O(\log n)$$

since $n^{473} + n^2 + n + 96 = O(n^{473})$ and $\log n^{473} = 473 * \log n$.

Logs of exponential functions?

- 2^n (or more generally c^n)
- $n!$

Federal Sentencing Guidelines

2F1.1. Fraud and Deceit; Forgery; Offenses Involving Altered or Counterfeit Instruments other than Counterfeit Bearer Obligations of the United States.(a) Base offense Level: 6

Loss(Apply the Greatest)	Increase in Level
(A) $2,000 or less	no increase
(B) More than $2,000	add 1
(C) More than $5,000	add 2
(D) More than $10,000	add 3
(E) More than $20,000	add 4
(F) More than $40,000	add 5
(G) More than $70,000	add 6
(H) More than $120,000	add 7
(I) More than $200,000	add 8
(J) More than $350,000	add 9
(K) More than $500,000	add 10
(L) More than $800,000	add 11
(M) More than $1,500,000	add 12
(N) More than $2,500,000	add 13
(O) More than $5,000,000	add 14

Make the Crime Worth the Time

- The federal sentencing guidelines are designed to help judges be consistent in assigning punishment.
- The time-to-serve is a roughly linear function of the total *level*.
- The increase in punishment level grows logarithmically in the amount of money stolen.
- Thus it pays to commit one big crime rather than many small crimes totalling the same amount.

Consolidation

- Review problems from the first assignment.
- Problem 2-39(d). Prove that

$$x^{\log_b y} = y^{\log_b x}$$

- Prove that $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1)$ (pyramidal numbers)