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

Ram Model of Computation

- Assume that each simple statement (arithmetic, memory, assignment, etc) requires 1 unit of time per execution.
- Figure out how many times each simple statement is executed.
- Add up for all statements.
- Loop and function calls are **NOT** simple statements.

This model is useful and accurate in the same sense as the flat-earth model (useful but not the truth).

Ram Model Worksheet

- # steps depends on n (for loop, single execution in for loop).
- # steps depends on input permutation, even for the same n.
- # steps would change slightly with another INSERTION SORT algorithm

Number of steps don't predict algorithms runtime with 100% accuracy

- Some instructions are more expensive to do

Problem-specific metrics

- Common in analysis of sorting or searching to count # of data comparisons
  - number of comparison in while loop on worksheet
- Matrix multiplication: count # of scalar multiplication

Hone in on the important lines when doing ram model worksheet problems.

Worst, Average, Best Case Complexity

- Worst case complexity: maximum # of steps taken on instance of size n
- Best case complexity: minimum
- Avg case complexity: average (lolnoshit)

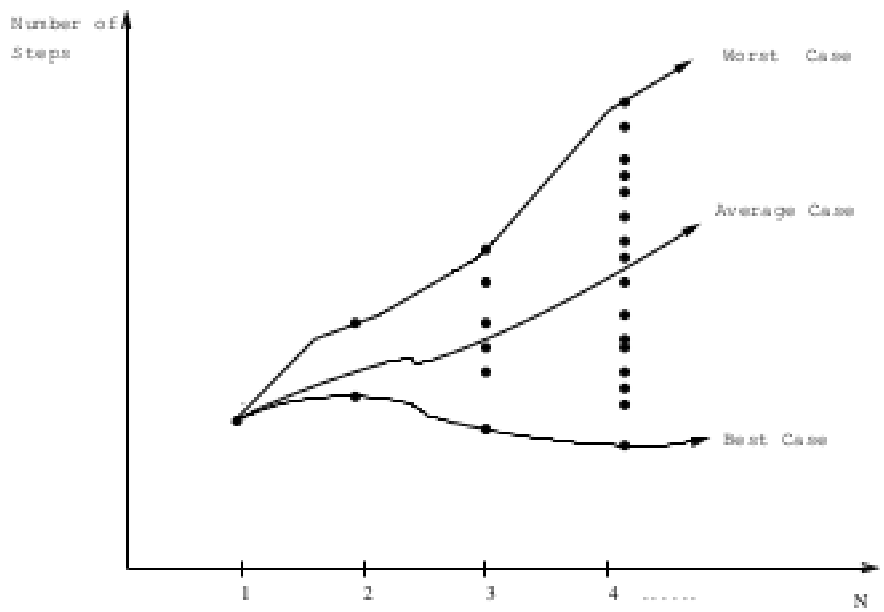

Figure 1: Complexity Graph

Exact analysis is hard, so use upper and lower bounds of functions (Asymptotic notation)

Definitions: O, Ω, Θ

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

- All of the above define sets. (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 find **one set of constants** that make the inequalities work (c)
- $n \geq n_0$ says don't worry about what happens at lower values of n (Care about things as they approach infinity)

Know how to prove an equation is a certain N complexity time (common exam question)

Using Big-Oh and Θ

Is n = O(n^2), YES

Is n = O(n^3), YES

Is n = Θ(nˆ2), NO

Think of these equations as bounds and averages. Makes more sense that way.

Also keep in mind to make sensible bounds

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		

Figure 2: Asymptotic Dominance in Action

Implication of Dominance

- Exponential (2^n and n!): goes bad fast
- Quadratic (n^2): goes bad at or before 1,000,000
- O(nlogn) is possible to 1 billion
- O(logn) never sweats

Testing Dominance

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))

n^a dominates n^b if a > b

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

Dominance Ranking

$n! >> 2^n >> n^3 >> n^2 >> nlogn >> n >> logn >> 1$

Advanced Dominance Ranking

$n! >> c^n >> n^3 >> nlogn >> n >> \sqrt{n} >> logn >> loglogn >> 1$

Epsilon is a small fraction in $n^{1+\epsilon}$

Reasoning About Efficiency

Selection Sort Example

```
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]);
        }
}
```

The outer loop takes n times, the inner loop at worst is n times

Nested loops means n x n which is $O(n^2)$ worst case

Logarithms

Logarithm is an inverse exponential function

$b^x = y$ is equivalent to $x = \log_b y$

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

- How many doubling from 1 to n is $\log_2 n$

Binary Search

Binary search throws away half the elements after each comparison.

- Binary Search = how many halves on n before getting 1 (LOG)

How tall a binary tree do we need until we have n leaves?

- Number of leaves double per level
- How many times we do need to double until we have n? (LOG)

Logarithms and Bits

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

- Each bit doubles the possble number of bit patterns, so $LOG(2^i) = i$

Logarithms and Multiplication

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

Base is not Asymptotically Important

Recall. . .

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

Changing base is like adding a constant to the expression, so in Big Oh, changing base is negligible.

Log of Polynomial Functions of n

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

- This is because $\log(n^{472}) = 472 * \log(n)$. Constant melts away

Logs of Exponential Functions

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

$\log(n!) = O(n\log(n))$