

Algorithm Design Manual Notes

Zachary William Grimm

Notes for ADM by Skiena

zwgrimm@gmail.com

February 14, 2019

Contents

1	Introduction To Algorithm Design	3
1.1	Robot Tour Optimization	4
1.2	Selecting the Right Jobs	4
1.3	Reasoning about Correctness	4
1.3.1	Expressing Algorithms	4
1.3.2	Problems and Properties	4
1.3.3	Demonstrating Incorrectness	4
1.3.4	Induction and Recursion	5
1.3.5	Summations	5
1.4	Modeling The Problem	5
1.4.1	Combinatorial Objects	5
1.4.2	Recursive Objects	5
1.5	About the War Stories	6
1.6	War Story: Psychic Modeling	6
2	Algorithm Analysis	6
2.1	The RAM Model of Computation	6

2.1.1	Best, Worst, and Average-Case Complexity	6
2.2	The Big Oh Notation	6
2.3	Growth Rates and Dominance Relations	7
2.3.1	Dominance Relations	7
2.4	Working with the Big Oh	8
2.4.1	Adding Functions	8
2.4.2	Multiplying Functions	8
2.5	Reasoning About Efficiency	9
2.5.1	Selection Sort	9
2.5.2	Insertion Sort	9
2.5.3	String Pattern Matching	9
2.5.4	Matrix Multiplication	10
2.6	Logarithms and Their Applications	10
2.6.1	Logarithms and Binary Search	10
2.6.2	Logarithms Trees	11
2.6.3	Logarithms and Bits	11
2.6.4	Logarithms and Multiplication	11
2.6.5	Fast Exponentiation	11
2.6.6	Logarithms and Summations	11
2.6.7	Logarithms and Criminal Justice	12
2.7	Properties of Logarithms	12
2.8	War Story: Mystery of the Pyramids	12
2.9	Advanced Aanalysis (*)	12
2.10	Esoteric Functions	12
2.11	Limits and Dominance Relations	12

1 Introduction To Algorithm Design

the algorithmic *problem* known as *sorting* is defined as follows:

Problem: Sorting

Input: A sequence of n keys a_1, \dots, a_n .

Output: The permutation (reordering) of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_{n-1} \leq a'_n$

```

I|N S E R T I O N S O R T
I N|S E R T I O N S O R T
I N S|E R T I O N S O R T
E I N|S R T I O N S O R T
E I N R|S T I O N S O R T
E I N R S|T I O N S O R T
E I I N R S T|I O N S O R T
E I I N R S T|O N S O R T
E I I N O R S T|N S O R T
E I I N N O R S T|S O R T
E I I N N O R S S T|O R T
E I I N N O O R S S T|R T
E I I N N O O R R S S T|T
E I I N N O O R R S S T T
  
```

Figure 1: Animation of insertion sort in action (time flows down)

```

insertion_sort(item s[], int n)
{
    int i,j; /* counters */

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

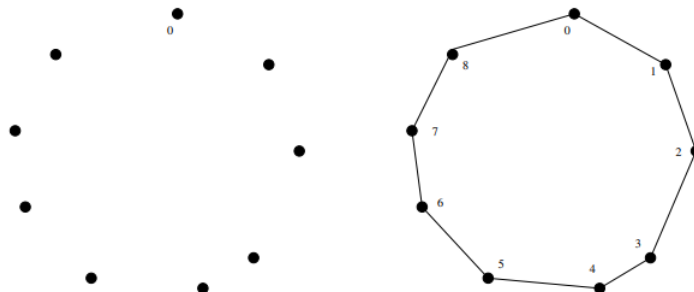


Figure 2: A good instance for the nearest neighbor heuristic

1.1 Robot Tour Optimization

Problem: Robot Tour Optimization

Input: A set S of N points in the plane.

Output: What is the shortest cycle tour that visits each point in the set S ?

1.2 Selecting the Right Jobs

Problem: Movie Scheduling Problem

Input: A set I of n intervals on the line.

Output: What is the largest subset of mutually non-overlapping intervals which can be selected from I ?

Take-Home Lesson: Reasonable-looking algorithms can easily be incorrect. Algorithm correctness is a property that must be carefully demonstrated.

1.3 Reasoning about Correctness

1.3.1 Expressing Algorithms

Take-Home Lesson: The heart of any algorithm is an *idea*. If your idea is not clearly revealed when you express an algorithm, then you are using too low-level a notation to describe it.

1.3.2 Problems and Properties

Take-Home Lesson: An important and honorable technique in algorithm design is to narrow the set of allowable instances until there *is* a correct and efficient algorithm. For example, we can restrict a graph problem from general graphs down to trees, or a geometric problem from two dimensions down to one.

1.3.3 Demonstrating Incorrectness

- *Verifiability*
- *Simplicity*
- *Think small*
- *Think exhaustively*
- *Hunt for the weakness*
- *Seek extremes*

Take-Home Lesson: Searching for counterexamples is the best way to disprove the correctness of a heuristic.

1.3.4 Induction and Recursion

Take-Home Lesson: Mathematical induction is usually the right way to verify the correctness of a recursive or incremental insertion algorithm.

1.3.5 Summations

- *Arithmetic progressions*
- *Geometric series*

1.4 Modeling The Problem

1.4.1 Combinatorial Objects

- *Permutations*
- *Subsets*
- *Trees*
- *Graphs*
- *Points*
- *Polygons*
- *Strings*

Take-Home Lesson: Modeling your application in terms of well-defined structures and algorithms is the most important single step towards a solution.

1.4.2 Recursive Objects

- *Permutations*
- *Subsets*
- *Trees*
- *Graphs*
- *Points*
- *Polygons*
- *Strings*

1.5 About the War Stories

1.6 War Story: Psychic Modeling

2 Algorithm Analysis

Our two most important tools are

1. *The RAM model of computation*
2. *The asymptotic analysis of worst-case complexity*

2.1 The RAM Model of Computation

Take-Home Lesson: Algorithms can be understood and studied in a language and machine-independent manner..

2.1.1 Best, Worst, and Average-Case Complexity

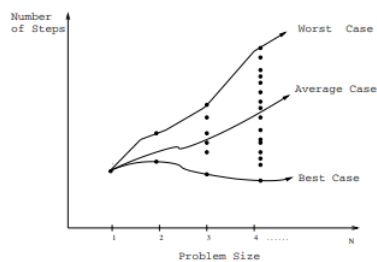


Figure 3: Best, Worst and average case complexity

2.2 The Big Oh Notation

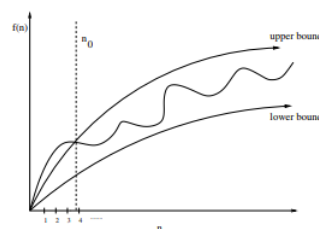


Figure 4: Upper and lower bounds valid for $n > n_0$ smooth out the behavior of complex functions

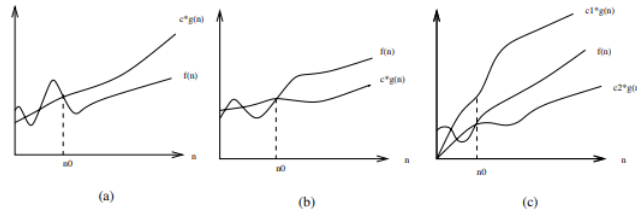


Figure 5: Illustrating the big (a) O , (b) Ω , and (c) Θ notations

Stop and Think: Hip to the Squares

Problem: Is $(x + y)^2 = O(x^2 + y^2)$

Stop and Think: Back to the Definition

Problem: Is $2^{n+1} = \Theta(2^n)$?

2.3 Growth Rates and Dominance Relations

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

Figure 6: Growth rates of common functions measured in nanoseconds

2.3.1 Dominance Relations

Take-Home Lesson: Although esoteric functions arise in advanced algorithm analysis, a small variety of time complexities suffice and account for most algorithms that are widely used in practice.

2.4 Working with the Big Oh

2.4.1 Adding Functions

$$O(f(n)) + O(g(n)) \longrightarrow O(\max(f(n), g(n)))$$

$$\Omega(f(n)) + \Omega(g(n)) \longrightarrow \Omega(\max(f(n), g(n)))$$

$$\Theta(f(n)) + \Theta(g(n)) \longrightarrow \Theta(\max(f(n), g(n)))$$

2.4.2 Multiplying Functions

$$O(c * f(n)) \longrightarrow O(f(n))$$

$$\Omega(c * f(n)) \longrightarrow \Omega(f(n))$$

$$\Theta(c * f(n)) \longrightarrow \Theta(f(n))$$

$$O(f(n)) * O(g(n)) \longrightarrow O(f(n) * g(n))$$

$$\Omega(f(n)) * \Omega(g(n)) \longrightarrow \Omega(f(n) * g(n))$$

$$\Theta(f(n)) * \Theta(g(n)) \longrightarrow \Theta(f(n) * g(n))$$

Stop and Think: Hip to the Squares Transitive Experience

Show that Big Oh relationships are transitive. That is, if $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$

2.5 Reasoning About Efficiency

2.5.1 Selection Sort

```
S E L E C T I O N S O R T
C E L E S T I O N S O R T
C E L E S T I O N S O R T
C E E L S T I O N S O R T
C E E I S T L O N S O R T
C E E I L T S O N S O R T
C E E I L N S O T S O R T
C E E I L N O S T S O R T
C E E I L N O T S S R T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
```

Figure 7: Animation of selection sort in action.

```
selection_sort(int s[], int n)
{
    int i, j;           /* counters */
    int min;            /* index of minimum */

    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]);
    }
}
```

2.5.2 Insertion Sort

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

2.5.3 String Pattern Matching

Problem: Substring Pattern Matching

Input: A text string *t* and a pattern string *p*

Output: Does *t* contain the pattern *p* as a substring, and if so where?

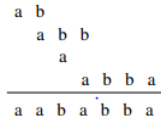


Figure 8: Searching for the substring abba in the text aababba

```
int findmatch(char *p, char*t)
{
    int i, j;          /* counters */
    int m, n;          /* string lengths */

    m = strlen(p);
    n = strlen(t);

    for (i=0; i<(n-m); i=i+1) {
        j = 0;
        while((j<m) && (t[i+j] == p[j]))
            j = j + 1;
        if(j ==m) return (i);
    }

    return(-1)
}
```

2.5.4 Matrix Multiplication

Problem: Matrix Multiplication

Input: Two matrices, A (of dimension $x \times y$) and B (dimension $y \times z$).

Output: An $x \times z$ matrix C where $C[i][j]$ is the dot product of the i th row of A and the j th column of B.

2.6 Logarithms and Their Applications

$$b^x = y \leftrightarrow x = \log_b y$$

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

2.6.1 Logarithms and Binary Search



Figure 9: A height h tree with d children per node as d^h leaves. Here $h = 2$ and $d = 3$

2.6.2 Logarithms Trees

2.6.3 Logarithms and Bits

2.6.4 Logarithms and Multiplication

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

$$\log_a n^b = b \cdot \log_a n$$

$$a^b = e^{(\ln(a^b))} = e^{(b(\ln(a)))}$$

2.6.5 Fast Exponentiation

2.6.6 Logarithms and Summations

Harmonic Numbers:

$$H(n) = \sum_{i=1}^n \frac{1}{i} \sim \ln(n)$$

2.6.7 Logarithms and Criminal Justice

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
(P) More than \$10,000,000	add 15
(Q) More than \$20,000,000	add 16
(R) More than \$40,000,000	add 17
(Q) More than \$80,000,000	add 18

Figure 10: The Federal Sentencing Guidelines for fraud

Take-Home Lesson: Logarithms arise whenever things are repeatedly halved or doubled/

2.7 Properties of Logarithms

2.8 War Story: Mystery of the Pyramids

2.9 Advanced Aanalysis (*)

2.10 Esoteric Functions

2.11 Limits and Dominance Relations