

# Algorithm Design Manual Notes

Zachary William Grimm

Notes for ADM by Skiena

zwgrimm@gmail.com

February 10, 2019

## Contents

<b>1</b>	<b>Introduction To Algorithm Design</b>	<b>3</b>
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
<b>2</b>	<b>Algorithm Analysis</b>	<b>6</b>
2.1	The RAM Model of Computation . . . . .	7

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

# 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;
        }
    }
}
  
```

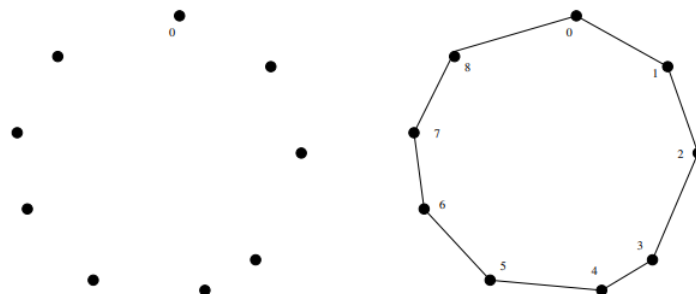


Figure 2: A good instance for the nearest neighbor heuristic

## 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**

algorithms

## **2.1 The RAM Model of Computation**

### 2.1.1 Best, Worst, and Average-Case Complexity

## **2.2 The Big Oh Notation**

## **2.3 Growth Rates and Dominance Relations**

### 2.3.1 Dominance Relations

## **2.4 Working with the Big Oh**

### 2.4.1 Adding Functions

### 2.4.2 Multiplying Functions

## **2.5 Reasoning About Efficiency**

### 2.5.1 Selection Sort

### 2.5.2 Insertion Sort

### 2.5.3 String Pattern Matching

### 2.5.4 Matrix Multiplication

## **2.6 Logarithms and Their Applications**

### 2.6.1 Logarithms and Binary Search

### 2.6.2 Logarithms Trees

### 2.6.3 Logarithms and Bits

### 2.6.4 Logarithms and Multiplication

### 2.6.5 Fast Exponentiation

### 2.6.6 Logarithms and Summations

### 2.6.7 Logarithms and Criminal Justice