Contents

[General Notes 1](#_Toc153774130)

[What is an algorithm? 1](#_Toc153774131)

[The algorithm Designer's Mantra 1](#_Toc153774132)

[How to Design an Algorithm 1](#_Toc153774133)

[Preliminaries 1](#_Toc153774134)

[Pattern for Pseudocode 2](#_Toc153774135)

[How to Analyze an Algorithm 2](#_Toc153774136)

[The scientific method (Algorithms 4th edition, Sedgwick et al.) 2](#_Toc153774137)

[Observations 2](#_Toc153774138)

[Principle 1: Worst Case Analysis 2](#_Toc153774139)

[Principle 2: Ignoring Constants 2](#_Toc153774140)

[Principle 3: Asymptotic Analysis 3](#_Toc153774141)

[Mathematical Concepts 3](#_Toc153774142)

[What is the mathematical proof? 3](#_Toc153774143)

[Propositions 3](#_Toc153774144)

[Concepts 4](#_Toc153774145)

[loop invariant 4](#_Toc153774146)

[Definitions 4](#_Toc153774147)

# General Notes

## What is an algorithm?

An algorithm is a set of well-defined rules or a recipe to solve a problem. For example, if we have a bunch of tasks that need to be completed according to their deadline, an algorithm will solve that problem.

Routing and Communication depend on the shortest path algorithm.

Algorithms are used in quantum mechanics and finance.

## The algorithm Designer's Mantra

"Perhaps the most important principle for the good algorithm designer is to refuse to be content" Aho , Hopcroft, and Ullman (the Design and analysis of Computer Algorithm, 1974)

Can we do better?

# How to Design an Algorithm

## Preliminaries

* Understand the problem. Try to frame it as a pragmatic solution. Specify the problem in a precise manner.
* Understand any scientific problem assumptions, the data type (float, integers, complex numbers), etc.
* Understand the requirements in terms of time and space complexity.
* Understand the needed data structure, as data structures go hand in hand with algorithm design.
* Understand the input and output and how to map the input to output.
* Try shorthand examples with no edge cases; solve them until the algorithm pops up
* If the algorithm doesn’t popup rewrite in simpler terms
* Write the algorithm in the most straightforward manner possible.
* Iterate
* Optimize

## Pattern for Pseudocode

* Define a computation problem.
* Define Input
* Define output.
* Solution that transforms input to output

# How to Analyze an Algorithm

## The scientific method (Algorithms 4th edition, Sedgwick et al.)

### Observations

Observe some features of the natural world, generally with precise measurements. Try it yourself, collect data, simulate, shorthand observation, or any means necessary to generate data consistently and robustly. In general, algorithms are analyzed on how much time they need to process input data of size n. therefore it is always a good idea to generate data observation data of a certain algorithm by simulating the time needed to perform a specific operation. Some books

* Hypothesize a model that is consistent with the observations.
* Predict events using the hypothesis.
* Verify the predictions by making further observations.
* Validate by repeating until the hypothesis and observations agree.

Be fancy and apply mathematical analysis to compress all this in a concise model, but most importantly, make sure you understand the hypothesis on a very intuitive level and let the math hinder that comprehension.

In our quest to find a fast algorithm “whose worst-case running time grows slowly with the input size”, according to Roughgarden, Linear time is the holy grail. Also, the Fast algorithm can be considered for free primitives, like merge sort, which are used in preprocessing the data.

## Principle 1: Worst Case Analysis

* Mathematically more Tractable than other kind of analysis
* Fewer assumptions
* Doesn’t require domain expertise compared to average case analysis.

## Principle 2: Ignoring Constants

* For mathematic tractability
* constants can vary from one platform to another.

## Principle 3: Asymptotic Analysis

* Asymptotic analysis studies the growth rate of running time with input length.

# Mathematical Concepts

## What is the mathematical proof?

The proof verifies that a proposition is confirmed through a series of steps called "logical deductions" from a base set of axioms (a set of self-evident truths); they are the starting point of logical reasoning. Poor axioms (shaky truth) can lead to unreliable mathematical reasoning. (paraphrased from Mathematics for Computer Science, Eric Lehman, and Tom Leighton)

## Types mathematical proof?

### Proof by induction

P(k) is an assertion for every positive integer. We use induction to prove a property in an algorithm that works for every single input to that algorithm (specific to general). Induction is analogous to recursion.

**Base:** Prove that P(n=1) is true

**Induction:** Prove that for every other integer n > 1 that if P(1), P(2) … P(n-1) is true that means P(n) is true for all n. we assume that P(k) is true .

# Definitions

### loop invariant

a loop invariant is a property of an algorithm or functionality that satisfies the following conditions.

Initialization: is true before the first iterations of the loop

Maintenance: is true before the loop iterations and before the next iterations

Termination: When the loop terminates, the invariant should help establish the correctness of the algorithm (as an example, reducing (the inclusive sum of the array's element is a loop invariant)

### Axioms

an axioms is a proposition that you believe is true. Axioms should be consistent and complete.

* Consistent, no proposition is self-contradictory.
* Complete, if every proposition can prove or disproved.

### Logical deductions

Also called inference rules, combines axioms and true proposition to generate more true propositions

* **Modus Ponens ,**if P is true and P==> Q is true it means Q is true
  + **Tautology** is one true proposition and, each tautological proposition there is an associated inference rule

### Propositions

In our logic system, Propositions are binary statements that are true or false. However, a likelihood can be attached to another system's proposition (wholly true or entirely false). A particular case of proposition is called a predicate whose truth depends on the value of one or more variables.

implications (==>) have the following truth tables

P Q P ==> Q

T T T

T F F

F T T (accepted mathematical convention)

T T T

for if and only ( <==>)

P Q P==>Q Q ==> P P<==>Q

T T T T T

T F F T F

F T T F F

F F F F T