Notes for Operations Research & More

Lan Peng, PhD Student

Department of Industrial and Systems Engineering
University at Buffalo, SUNY
lanpeng@buffalo.edu

February 16, 2020

February 16, 2020

To My Beloved Motherland

Contents

Part I

Algorithms

Chapter 1

Computational Complexity

1.1 Asymptotic Notation

1.1.1 Asymptotic Analysis

Definition 1.1.1 (asymptotically positive function)**.** $f : \mathbb{N} \to \mathbb{R}$ is an asymptotically positive function if $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

1.1.2 O-Notation, Ω-Notation and Θ-Notation

Definition 1.1.2 (O-Notation)**.** For a function $g(n)$, $O(g(n)) = \{f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0\}$

O-Notation also known as asymptotic upper bound.

Definition 1.1.3 (Ω-Notation)**.** For a function $g(n)$, $\Omega(g(n)) = \{f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}$

Ω-Notation also known as asymptotic lower bound.

Definition 1.1.4 (Θ-Notation)**.** For a function $g(n)$, $\Theta(g(n)) = \{f : \exists c > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$

Ω-Notation and Θ-Notation are not used very often when we talk about running times.

Definition 1.1.5 (o-Notation)**.** For a function $g(n)$, $o(g(n)) = \{f : \exists c > 0, n_0 > 0 \text{ such that } f(n) <\leq> cg(n), \forall n \geq n_0\}$

Definition 1.1.6 (ω-Notation)**.** For a function $g(n)$, $\omega(g(n)) = \{f : \exists c > 0, n_0 > 0 \text{ such that } f(n) > cg(n), \forall n \geq n_0\}$

Notice: $O(g)$, $\Omega(g)$ and $\Theta(g)$ are sets, we use "=" to represent "$\in$". In here "=" is asymmetric. Equality such as $O(n^3) = n^3 + n$ is incorrect.

Theorem 1.1. *Let f and g be two functions that*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0 \tag{1.1}$$

Then $f(n) = \Theta(g(n))$

Proof. Since $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists and positive, there is some n_0 beyond which the ratio is always between $\frac{1}{2}c$ and 2c. Thus,

$$\forall n > n_0, f(n) \leq 2cg(n) \Rightarrow f(n) = O(g(n)) \tag{1.2}$$

$$\forall n > n_0, f(n) \geq \frac{1}{2}cg(n) \Rightarrow f(n) = \Omega(g(n)) \tag{1.3}$$

□

A set of properties:

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n)) \tag{1.4}$$
$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } g(n) = O(f(n)) \tag{1.5}$$

Another set of properties:

$$f = O(g), g = O(h) \Rightarrow f = O(h) \tag{1.6}$$
$$f = \Omega(g), g = \Omega(h) \Rightarrow f = \Omega(h) \tag{1.7}$$
$$f = \Theta(g), g = \Theta(h) \Rightarrow f = \Theta(h) \tag{1.8}$$

Theorem 1.2. *If $f_i = O(h)$, for finite number of $i \in K$, then $\sum_{i \in K} f_i = O(h)$*

Proof is trivia.

Notice: Function f and g not necessarily have relation $f = O(g)$ or $g = O(f)$. E.g., for $f = \sqrt{n}$ and $g = n^{\sin n}$, $f \notin O(g)$ and $g \notin O(f)$

1.2 Common Running Times

1.2.1 $O(n)$

Example. Scan through a list to find a element matching with input

1.2.2 $O(\lg n)$

Example. Binary-search

1.2.3 $O(n^2)$

Example. Scan through every pair of elements, $\binom{n}{2}$

1.2.4 $O(n^3)$

Example. Matrix-multiplication by definition

1.2.5 $O(n \lg n)$

Example. Many divide and conquer algorithm which in each step iteratively divide the problem into two part and solve the sub-problem, for example merge-sort.

1.2.6 $O(n!)$

Example. Enumerate all permutation, for example Hamiltonian Cycle Problem

1.2.7 $O(c^n)$

Example. Enumerate all elements in power set. $(O(2^n))$

1.2.8 $O(n^n)$

Example. Enumerate all combinations

1.2.9 Comparison between Running Times

$\lg n < n < n \lg n < n^2 < n^{\sqrt{n}} < 2^n < e^n < n! < n^n$

Chapter 2

General Strategies

2.1 Greedy Algorithms

2.2 Divide and Conquer

2.3 Dynamic Programming

Chapter 3

Sorting

3.1 Bubble Sort

3.2 Insertion Sort

3.3 Selection Sort

3.4 Heap-sort

3.5 Quick-sort

3.6 Merge-sort

3.7 Sorting in Linear Time

Chapter 4

Mathematical Algorithm

4.1 Matrices Multiplication

4.2 Gaussian Elimination

4.3 Curve Fitting

4.4 Integration

4.5

Chapter 5

Searching

Chapter 6

String

6.1 String Searching

6.2 Pattern Matching

6.3 Parse

6.4 File Compression

6.5 Cryptology

Chapter 7

Geometric Algorithm

7.1 Convex Hull

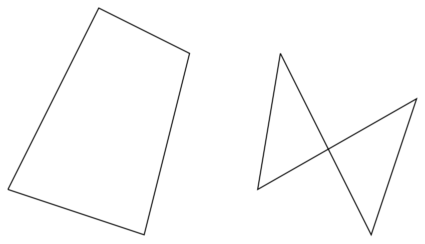
7.2 Geometric Intersection

7.3 Closest Point

7.4 Polygon Triangulation

7.4.1 Types of Polygons

Definition 7.4.1 (simple polygon)**.** A **simple polygon** is a closed polygonal curve without self-intersection.



Simple Polygon Non-simple Polygon

Polygons are basic building blocks in most geometric applications. It can model arbitrarily complex shapes, and apply simple algorithms and algebraic representation/manipulation.

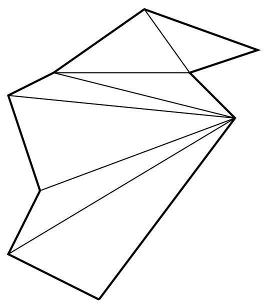
7.4.2 Triangulation

Definition 7.4.2 (Triangulation)**. Triangulation** is to partition polygon P into non-overlapping triangles using diagonals only. It reduces complex shapes to collection of simpler shapes. Every simple n-gon admits a triangulation which has $n - 2$ triangles.

Theorem 7.1. *Every polygon has a triangulation*

Lemma 7.2. *Every polygon with more than three vertices has a diagonal.*

Proof. (by Meisters, 1975) Let P be a polygon with more than three vertices. Every vertex of a P is either *convex* or *concave.* W.L.O.G.(any polygon must has convex corner) Assume p is a convex vertex. Denote the neighbors of p as q and r. If $\bar{qr}$ is a diagonal, done, and we call $\triangle pqr$ is an *ear.* If $\triangle pqr$ is not an ear, it means at least one vertex is inside $\triangle pqr$, assume among those vertexes inside $\triangle pqr$, s is a vertex closest to p, then $\bar{ps}$ is a diagonal. $\square$



Triangulation

7.4.3 Art Gallery Theorem

Theorem 7.3. *Every n-gon can be guarded with $\lfloor \frac{n}{3} \rfloor$ vertex guards*

Lemma 7.4. *Triangulation graph can be 3-colored.*

Problem 7.1. The floor plan of an art gallery modeled as a simple polygon with n vertices, there are guards which is stationed at fixed positions with 360 degree vision but cannot see through the walls. How many guards does the art gallery need for the security? (Fun fact: This problem was posted to Vasek Chvatal by Victor Klee in 1973).

Proof. - P plus triangulation is a planar graph
- 3-coloring means there exist a 3-partition for vertices that no edge or diagonal has both endpoints within the same set of vertices.
- Proof by Induction:
- Remove an ear (there will always exist ear)
- Inductively 3-color the rest
- Put ear back, coloring new vertex with the label not used by the boundary diagonal. □

7.4.4 Triangulation Algorithms

Chapter 8

Data Structures

8.1 Elementary Data Structures

8.2 Hash Tables

8.3 Binary Search Trees

8.4 Red-Black Trees

8.5 B-Trees

8.6 Fibonacci Heaps

8.7 van Emde Boas Trees

Chapter 9

NP and PSPACE