1.4.18Gauses/Karatsuba Multiplication is $O(n^{\log_2 3})$tcb@cnt@claim.1.4.1 e1.4.29Strassen's Matrix Multiplication is $O(n^{\log_2 7})$tcb@cnt@claim.1.4.2

Design and Analysis of Algorithms

This document is a collection of notes on the design and analysis of algorithms.

Contents

Chapter 1

Asymptotic Notation

1.1 Big-O

Definition 1.1.1: Let $f(n)$ and $g(n)$ be functions from the set of positive integers to the set of positive real numbers. We say that $f(n)$ is $O(g(n))$ if there exist positive constants c and n_0 such that $0 \leqslant f(n) \leqslant cg(n)$ for all $n \geqslant n_0$.

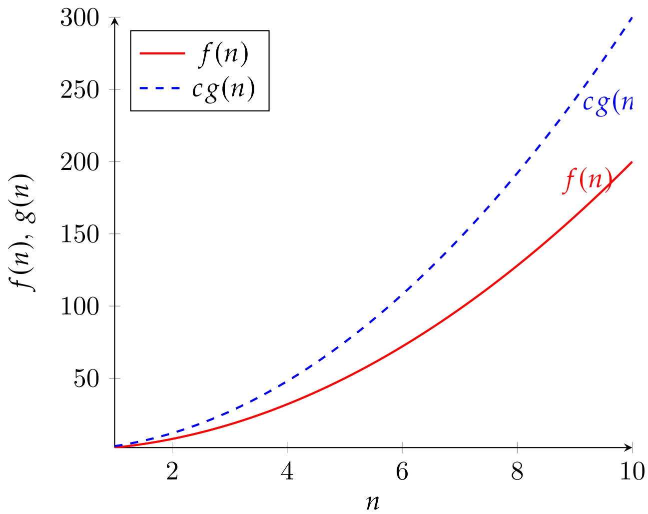


Figure 1.1: Graph showing $f(n) = O(g(n))$

1.2 Big-Ω

Definition 1.2.1: Let $f(n)$ and $g(n)$ be functions from the set of positive integers to the set of positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \leqslant cg(n) \leqslant f(n)$ for all $n \geqslant n_0$.

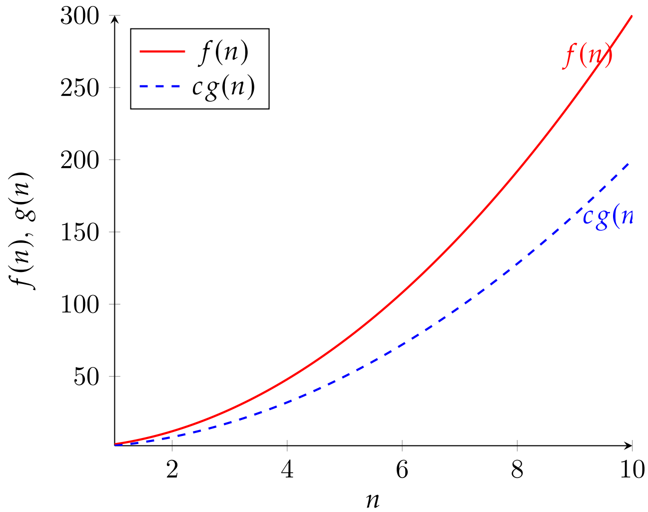


Figure 1.2: Graph showing $f(n) = \Omega(g(n))$

1.3 Big-Θ

Definition 1.3.1: Let $f(n)$ and $g(n)$ be functions from the set of positive integers to the set of positive real numbers. We say that $f(n)$ is $\Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ for all $n \geqslant n_0$.

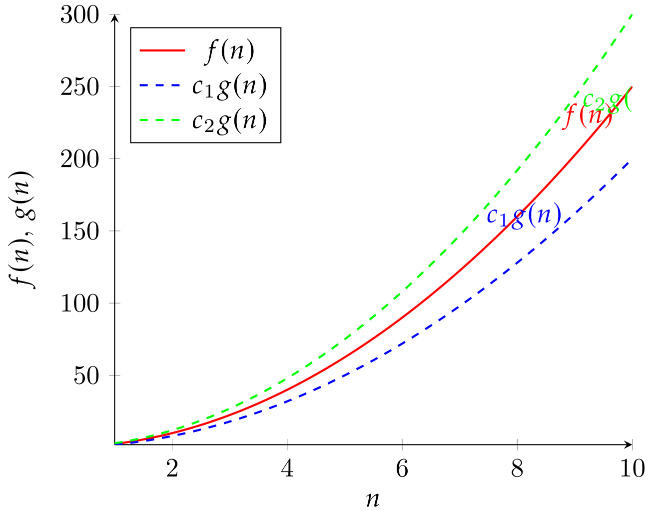


Figure 1.3: Graph showing $f(n) = \Theta(g(n))$

1.4 Master Theorem

Theorem 1.4.1 Let $a \geqslant 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$T(n) = aT(\frac{n}{b}) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } \log_b a > d \\ O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \end{cases}$$

(assuming dividing into subproblems takes constant/polynomial time units)

Note:-

Prove the Master Theorem using the following steps:

assuming $n = b^k$ for some $k \in \mathbb{N}$

$$\begin{aligned} & n \longrightarrow \text{size of main problem} \\ & \downarrow \\ & \frac{n}{b}, \frac{n}{b}, \ldots, \frac{n}{b} \longrightarrow b \text{ subproblems} \\ & \vdots \\ & \frac{n}{b^k}, \frac{n}{b^k}, \ldots, \frac{n}{b^k} \longrightarrow \text{each of the final subproblems can be solved in} \\ & \qquad \Theta(1) \text{ time as they are trivial} \end{aligned}$$

Let $n = \frac{a}{b^d}$

$$T(n) = \sum_{i=0}^{k} a^i \left(\frac{1}{b^i}\right)^d = \sum_{i=0}^{k} n^d \left(\frac{a}{b^d}\right)^i = n^d \sum_{i=0}^{k} n^i$$

If $n < 1$, $\sum_{i=0}^{k} n^i = 1 + n + \ldots + n^k \approx 1$

$$\Rightarrow n^d \sum_{i=0}^{k} n^i \approx n^d \text{ when } b^d > a$$
$$\Rightarrow T(n) \in O(n^d) \text{ when } d \geqslant \log_b a.$$

If $n = 1$, $\sum_{i=0}^{k} n^i = 1 + k$

$$\Rightarrow n^d \sum_{i=0}^{k} n^i \approx kn^d \text{ when } b^d = a$$
$$\Rightarrow T(n) \in O(n^d \log n) \text{ when } d = \log_b a.$$

If $n > 1$, $\sum_{i=0}^{k} n^i = 1 + n + \ldots + n^k \approx n^k$

$$\Rightarrow n^d \sum_{i=0}^{k} n^i \approx n^d n^k \text{ when } b^d < a$$
$$\Rightarrow T(n) \in O(n^{\log_b a}) \text{ when } \log_b d < \log_b a.$$

Example 1.4.1

Applying Merge sort on elemnts of a toset

Sol.

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + O(n)$$
$$T(n) = 2T(\frac{n}{2}) + O(n)$$
$$a = 2, b = 2, d = 1$$
$$\implies \log_b a = log_2 2 = 1$$
$$\therefore T(n) = O(n \log n)$$

Example 1.4.2

Addition of two n-bit integers, x and y

Sol.

Every bitwise addition results in at most 2 digits (e.g., $1 + 1 = 10$).
In such a case, the first digit is the carry.
The sum of three digits in any base has at most two digits.
For example: $9 + 9 + 9 = 27,\ (1) + (1)$ (in base 2 is 10), etc.
Bitwise addition of x and y requires at most $2n - 1$ addition operations
(carry forward after every addition in the extreme case).
Thus, addition of $x + y$ is in $O(n)$.

Example 1.4.3

Binary search on $A[1, 2, \ldots, n]$ for K

Sol.

$$T(N) = T\left(\frac{N}{2}\right) + O(1) \quad \text{(Time to compare } K \text{ with } A\left(\frac{N}{2}\right) + \text{middle element)}$$
$$a = 1, \quad b = 2, \quad d = 0$$
$$\log_b a = \log_2 1 = 0$$
$$\therefore T(N) = O(\log N)$$

Example 1.4.4

Multiplying 2 n-bit integers, x and y

Sol.

$$X = \sum_{i=1}^{n} x_i 2^{i-1}$$

$$Y = \sum_{i=1}^{n} y_i 2^{i-1}$$

Assume $n = 2^k$ for some $k \in \mathbb{N}$

$$X = \sum_{i=1}^{2^k} x_i 2^{i-1} + 2^{\frac{n}{2}} \sum_{i=1}^{\frac{n}{2}} n_{\frac{n}{2}+1} 2^{i-1}$$

$$X = X_R + 2^{\frac{n}{2}} X_L$$

$$Y = Y_R + 2^{\frac{n}{2}} Y_L$$

$$X \cdot Y = (X_L Y_R + X_R Y_L) + 2^n X_L Y_L + X_R Y_R$$

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

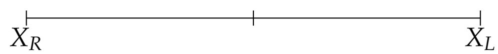
$$a = 4, b = 2, d = 1$$

$$\log_b a = \log_2 4 = 2 \geqslant 1$$

$$\therefore T(n) = O(n^2)$$

Claim 1.4.1 Gauses/Karatsuba Multiplication is $O(n^{\log_2 3})$

$$
\begin{aligned}
X &= X_L X_R, \quad Y = Y_L Y_R, \\
P_1 &= X_R \cdot Y_R \\
P_2 &= X_L \cdot Y_L \\
P_3 &= X_R \cdot Y_R + X_L \cdot Y_L \\
X \cdot Y &= P_1 + 2^{\frac{n}{2}} P_3 + 2^n P_2 \\
T(n) &= 4T\left(\frac{n}{2}\right) + O(n) \\
\therefore T(n) &= O(n^2)
\end{aligned}
$$



This is the same as the trivial method. We have to improve this by reducing the number of subproblems.

$$
\begin{aligned}
X &= X_L X_R, \quad Y = Y_L Y_R, \\
P_1 &= X_R \cdot Y_R \\
P_2 &= X_L \cdot Y_L \\
P_3 &= X_R \cdot Y_R + X_L \cdot Y_L \\
&= (X_L + X_R)(Y_L + Y_R) - P_1 - P_2 \\
X \cdot Y &= P_1 + 2^{\frac{n}{2}} P_3 + 2^n P_2 \\
T(n) &= 3T\left(\frac{n}{2}\right) + O(n) \\
a &= 3,\ b = 2,\ d = 1 \\
\log_b a &= \log_2 3 > 1 \\
\therefore T(n) &= O(n^{\log_2 3})
\end{aligned}
$$

Note:-

We can also apply this to numbers of any base:

$$
\begin{aligned}
X &= X_R \cdot B^M + X_L \\
Y &= Y_R \cdot B^M + Y_L \\
X \cdot Y &= P_1 + B^{2M} P_2 + B^M (P_3 - P_1 - P_2)
\end{aligned}
$$

Example 1.4.5
Matrix Multiplication of 2 n x n matrices

Sol.

$$
\begin{aligned}
X_{n\times n} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\
Y_{n\times n} &= \begin{bmatrix} E & F \\ G & H \end{bmatrix} \\
X \cdot Y &= \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix} \\
T(n) &= 8T\left(\frac{n}{2}\right) + O(n^2) \\
a &= 8,\ b = 2,\ d = 2 \\
\log_b a &= \log_2 8 = 3 > 2 \\
\therefore T(n) &= O(n^3)
\end{aligned}
$$

Claim 1.4.2 Strassen's Matrix Multiplication is $O(n^{\log_2 7})$

$$
\begin{aligned}
X_{n\times n} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\
Y_{n\times n} &= \begin{bmatrix} E & F \\ G & H \end{bmatrix} \\
M_1 &= (A+D)(E+H) \\
M_2 &= (C+D)E \\
M_3 &= A(F-H) \\
M_4 &= D(G-E) \\
M_5 &= (A+B)H \\
M_6 &= (C-A)(E+F) \\
M_7 &= (B-D)(G+H) \\
XY &= \begin{bmatrix} M_1+M_4-M_5+M_7 & M_3+M_5 \\ M_2+M_4 & M_1-M_2+M_3+M_6 \end{bmatrix} \\
T(n) &= 7T\left(\frac{n}{2}\right) + O(n^2) \\
a &= 7, b = 2, d = 2 \\
\log_b a &= \log_2 7 > 2 \\
\therefore T(n) &= O(n^{\log_2 7})
\end{aligned}
$$

Example 1.4.6

Finding median of an unordered list of n elements

Sol.

$$\text{Median of } n \text{ elements} = \begin{cases} \text{if n is odd} & \text{median is } \left(\frac{n}{2}+1\right)^{th} \text{ element} \\ \text{if n is even} & \text{median is } \left(\frac{n}{2}\right)^{th} \text{ element} \end{cases}$$

Finding Kth smallest element in an unordered list of n elements

Given $A[1,2,\ldots,n]$ and k where $k(<n) \in \mathbb{N}$.

$$S_L = \{x \in A : x < m\}$$
$$S_E = \{x \in A : x = m\}$$
$$S_R = \{x \in A : x > m\}$$

Case 1: If $|S_L| \geqslant k$,

$$\text{Selection}(A[1,2,\ldots,n],k) = \text{Selection}(S_L,k).$$

Case 2: Else if $|S_L| + |S_E| \geqslant k$,

$$\text{Selection}(A[1,2,\ldots,n],k) = m.$$

Case 3: Else,

$$\text{Selection}(A[1,2,\ldots,n],k) = \text{Selection}(S_R, k - |S_L| - |S_E|).$$

If the randomly chosen number is indeed the median,

$$T(n) = T(\frac{n}{2}) + O(n)$$

$a = 1$, $b = 2$, $d = 1$

$\log_b a = \log_2 1 = 0 < 1 = d$

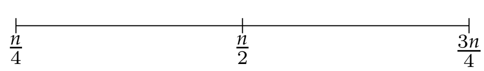
$\Rightarrow T(n) \in O(n)$.

Imagine $T(n)$ if m isn't the median...

A method to ensure m is close to median: Find the median of the first 5 elements, the next 5 elements and so on. Assign the median of these medians to m.

(This needs a deterministic algorithm to find median of n numbers in $O(n)$ time. Here, finding the median of 5 elements takes $O(1)$ time. There are $\lceil \frac{n}{5} \rceil$ such medians to be computed. Therefore, this method takes $O(n)$ time.)

Yet another method to find the k^{th} smallest element:



If m is a number greater than $\frac{n}{4}$ numbers and lesser than $\frac{n}{4}$ other numbers, then in the worst possible case, m is either the greatest or the least smallest number satisfying the above condition.

Then either S_R or S_L is containing $\frac{n}{4}$ numbers, so discarded. What remains is S_L, S_M or S_M, S_R.

$$T(n) = T(\frac{3n}{4}) + O(n)$$

$a = 1$, $b = \frac{4}{3}$, $d = 1$

$$\log_b a = \log_{\frac{4}{3}} 1 < \log_{\frac{4}{3}} \frac{4}{3} = 1 = d$$

$\Rightarrow T(n) \in O(n)$.

The probability of getting such an m is $\frac{1}{2}$.

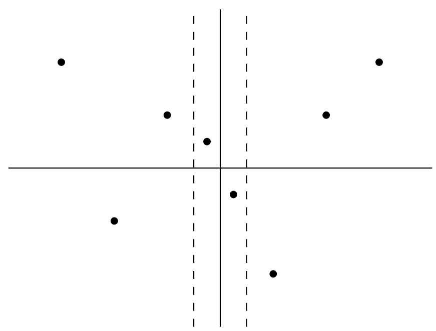
Example 1.4.7

Closest Pair of Points

Sol.

Given: n points $p_1(x_1, y_1), p_2(x_2, y_2), \ldots, p_n(x_n, y_n)$.
We have to find the smallest distance among all possible pairs.
Trivially, $\binom{n}{2}$ distances can be computed in $O(n^2)$ time, and the least distance can be found in $O(n)$ time (totally $O(n^2)$ time).

We can do better than this.



- Sort the given n points according to their abscissae. $\rightarrow O(n \log n)$
- Recursively apply the algorithm on the left and right halves to find the closest pair smallest distance in their respective halves (each half includes the point with x_{median} as its abscissa). $\rightarrow 2T(\frac{n}{2})$
- D = minimum of the smallest distances of the left and the right.
- There could be a point on the left and a point on the right forming the closest pair (distance $< d$). Hence, isolate the strip containing points whose abscissae lie in the range $[x_{median} - d, x_{median} + d]$. Such a pair can only lie in this strip. $\rightarrow O(n)$
- Sort the points in the strip according to their ordinates. $\rightarrow O(n \log \frac{n}{2^l})$ (per subcase $l \in \mathbb{N}$)
- The distance between two points in the strip is at least D, with this it can be stated that in a rectgular region of size $2D \times D$, there can be at most 8 points.
- $\therefore$a point only has to check the distance with the next subsequent 7 points. $\rightarrow O(1)$ per point, $O(n)$ per subcases

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$
$$\therefore T(n) = O(n \log n^2)$$