

COMP3711

Design and Analysis of Algorithms

Larry128

A summary notes for revision

Fall 2024-2025

Contents

1	Prerequisites	2
1.1	Input size of Problems	2
1.2	Asymptotic Notation	2
1.3	Introduction to Algorithms	6

1 Prerequisites

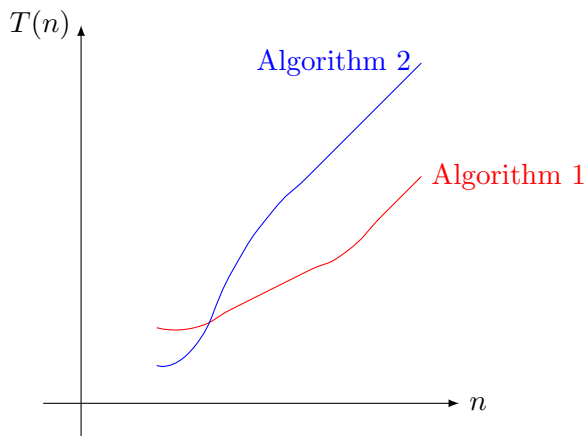
1.1 Input size of Problems

- Input size** how large the input is.
- Assumption**
1. any number can be stored in a computer word
 2. each arithmetic operation takes constant time
- Examples**
- Sorting: Size of the list or array
- Graph problems: Numbers of vertices and edges
- Searching: Number of input keys

1.2 Asymptotic Notation

1. Running time/ Cost of algorithms
 - i. a function of input size: $T(n)$
 - ii. number of operations (e.g., comparisons between two numbers)
 - iii. using **asymptotic notation**, which ignores constants and non-dominant growth terms

2. Intuitions

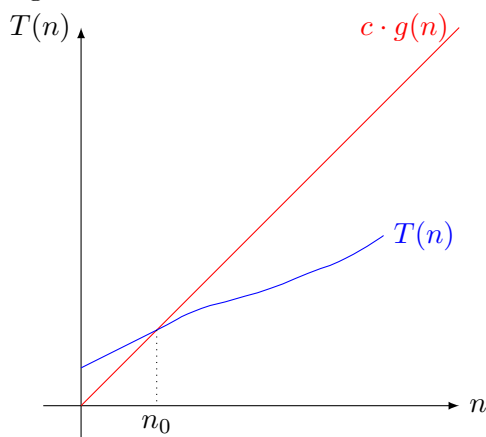


From the figure above, Algorithm 1 is better for large n .

3. Rigorous definition of asymptotic notation

Upper bound $T(n) = O(f(n))$	if $\exists c > 0$ and $n_0 \geq 0$ such that $\forall n \neq n_0, T(n) \leq cf(n)$
Lower bound $T(n) = \Omega(f(n))$	if $\exists c > 0$ and $n_0 \neq 0$ such that $\forall n \neq n_0, T(n) \geq cf(n)$
Tight bound $T(n) = \Theta(f(n))$	if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$

4. Big-O Notation



$$T(n) = O(g(n)) \iff \exists n_0 > 0 \text{ s.t. } \forall n \geq n_0, 0 \leq T(n) \leq c \cdot g(n)$$

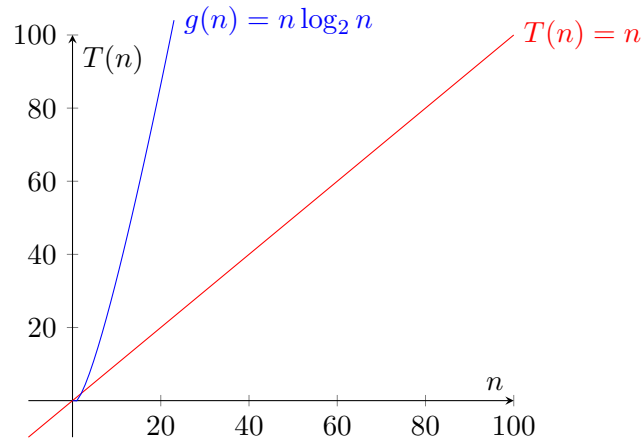
Below are some examples of Big-O notation proofs

(a) $T(n) = n, g(n) = n \log_2 n$

We wish to prove $T(n) = n \in O(n \log_2 n)$.

Choose $c = 1, n_0 = 2$, for all $n \geq 2 = n_0$,

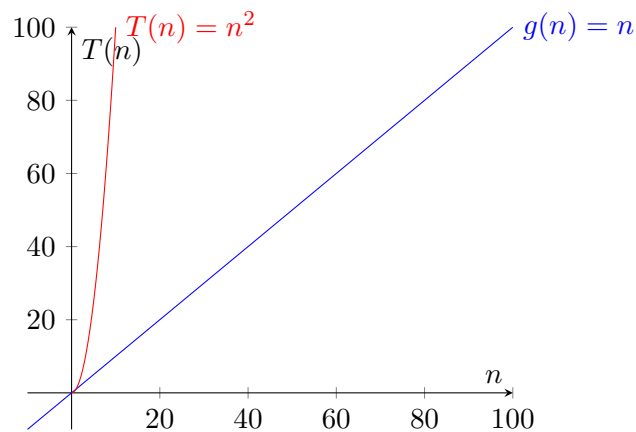
$$1 \leq \log_2 n \iff n \leq n \log_2 n \iff n \leq c \cdot n \log_2 n$$



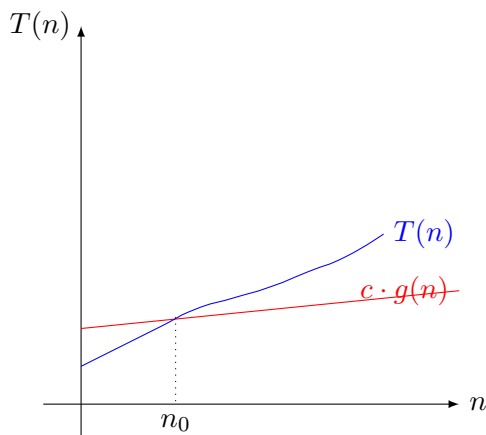
(b) $T(n) = n^2, g(n) = n$

We wish to prove $T(n) = n^2 \notin O(g(n))$ by contradiction.

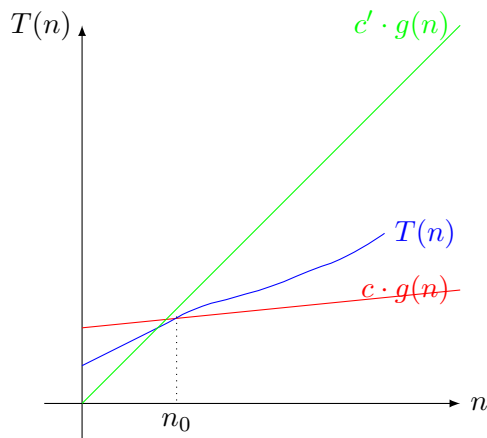
Suppose there exists some c and n_0 such that for all $n \geq n_0$, $n^2 \leq c \cdot n$. Then, $n \leq c$, $\forall n \geq n_0$, which is not possible as c is a constant and n can be arbitrarily large.



5. Big- Ω Notation



$$T(n) = \Omega(g(n)) \iff \exists n_0 > 0 \text{ s.t. } \forall n \geq n_0, 0 \leq c \cdot g(n) \leq T(n)$$

6. Big- Θ Notation

$$T(n) = \Theta(f(n)) \iff T(n) = O(f(n)) \text{ and } T(n) = \Omega(f(n))$$

7. Implementation and experimentation are needed sometimes

If algorithm A is $T_1(n) = 10n \in \Theta(n)$, algorithm B is $T_2(n) = 1000n \in \Theta(n)$, but algorithm A is superior in practice. In this case, Implementation and experimentation are needed.

8. Basic facts on exponents and logarithms

(a) $2^{2n} \neq \Theta(2^n)$, proof: set $x = 2^n$, then $x^2 \neq \Theta(x)$

(b) $2^{n+2} = 4 \cdot 2^n = \Theta(2^n)$

(c) $\log_a(n^b) = \frac{b \log n}{\log a} = \Theta(\log n)$

(d) $\log_b a = \frac{1}{\log_a b}$

(e) $a^{\log_b n} = n^{\log_b a}$

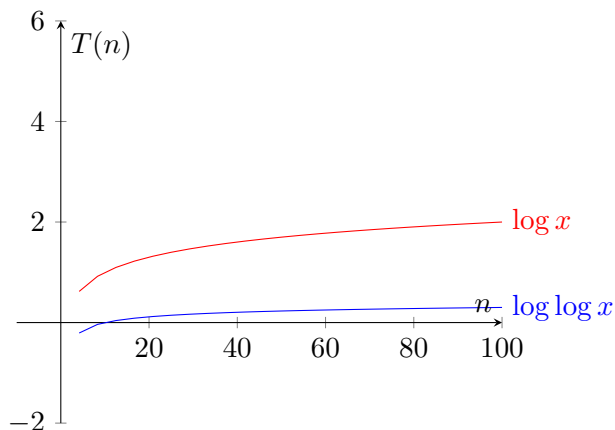
9. Important note on growth of functions

$$k < \log n < n^a < n \log n < n^b < c^n$$

,where $k, c \in \mathbb{R}, 0 < a < 2, b \geq 2$ are constants

(a) $999^{999^{999}} = \Theta(1)$

(b) $\log \log n = O(\log n)$, proof: for $n \geq 2$, $\log \log n \leq \log n$



(c) $n \log n = O\left(\frac{n^2}{\log n}\right)$

proof: To show $n \log n = O\left(\frac{n^2}{\log n}\right)$, it suffices to show that there exists a $C > 0$, such that

$$n \log n < C \cdot \frac{n^2}{\log n} \text{ for sufficiently large } n.$$

$$\begin{aligned} n \log n &< C \cdot \frac{n^2}{\log n} \\ \iff (\log n)^2 &< C \cdot n \end{aligned}$$

It's obvious that for large n , $\log(n) < n^\epsilon$ for $\epsilon > 0$, then we can pick $\epsilon = \frac{1}{2}$

$$\begin{aligned} \log n &< n^{\frac{1}{2}} \\ (\log n)^2 &< n \end{aligned}$$

Since $C > 0$, we can see $(\log n)^2 < n < C \cdot n$. We are done.

10. Extra Examples

- (a) $1000n + n \log n = O(n \log n)$
- (b) $n^2 + n \log(n^3) = n^2 + 3n \log n = O(n^2)$
- (c) $n^3 = \Omega(n)$
- (d) $n^3 = O(n^{10})$
- (e) Let $f(n)$ and $g(n)$ be non-negative functions. Using basic definition of Θ -notation, proof that $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$

- i. Step 1: proof $\max\{f(n), g(n)\} = O(f(n) + g(n))$
For all n , $\max\{f(n), g(n)\}$ is either equal to $f(n)$ or equal to $g(n)$. So we can deduce that $\max\{f(n), g(n)\} \leq f(n) + g(n)$. Therefore, $\max\{f(n), g(n)\} = O(f(n) + g(n))$.

- ii. Step 2: proof $\max\{f(n), g(n)\} = \Omega(f(n) + g(n))$
Note that $\max\{f(n), g(n)\} \geq f(n)$ and $\max\{f(n), g(n)\} \geq g(n)$. So

$$\begin{aligned} \max\{f(n), g(n)\} + \max\{f(n), g(n)\} &\geq f(n) + g(n) \\ 2 \cdot \max\{f(n), g(n)\} &\geq f(n) + g(n) \\ \max\{f(n), g(n)\} &\geq \frac{1}{2}(f(n) + g(n)) \end{aligned}$$

Then, we have $\max\{f(n), g(n)\} = \Omega(f(n) + g(n))$

- (f) if $A = \log \sqrt{n}$, $B = \sqrt{\log n}$, then $A = \Omega(B)$

proof: $A = \log \sqrt{n} = \frac{1}{2} \log n = \Theta(\log n)$, $B = \sqrt{\log n} = \Theta(\log n)$. We can simply deduce that $\log \sqrt{n} = \Omega(\sqrt{\log n})$

- (g) Bounds of series - Arithmetic Series

Proof that $\sum_{i=1}^n i = 1 + 2 + 3 + \dots + (n-1) + n = \Theta(n^2)$

- i. Approach 1: use formula $\sum_{i=1}^n i = \frac{n(1+n)}{2} = \Theta(n^2)$

- ii. Approach 2

A. Step 1: proof $\sum_{i=1}^n i = O(n^2)$

$$\begin{aligned} \sum_{i=1}^n i &= 1 + 2 + 3 + \dots + (n-1) + n \\ &\leq n + n + \dots + n \\ &= \sum_{i=1}^n n \\ &= n \cdot n \\ &= n^2 = O(n^2) \end{aligned}$$

B. Step 2: proof $\sum_{i=1}^n i = \Omega(n^2)$

$$\begin{aligned}
 \sum_{i=1}^n i &= 1 + 2 + 3 + \dots + (n-1) + n \\
 &\geq 0 + 0 + \dots + 0 + \dots + \frac{n}{2} + \left(\frac{n}{2} + 1\right) + \dots + n \\
 &\geq \frac{n}{2} \cdot \frac{n}{2} \\
 &= \frac{n^2}{4} = \Omega(n^2)
 \end{aligned}$$

Then, we can say that $\sum_{i=1}^n i = \Theta(n^2)$

(h) Bounds of series - Polynomial Series

Proof that $\sum_{i=1}^n i^c = 1^c + 2^c + 3^c + \dots + (n-1)^c + n^c = \Theta(n^{c+1})$

(The proof is more or less the same as the approach 2 of arithmetic series.)

(i) Bounds of series - Harmonic Series H_n

Proof that $H_n = \sum_{i=1}^n \frac{1}{i} = \Theta(\log n)$.

Let $k = \log_2 n$, then $n = 2^k$.

index	lower bound	parts of H_n	upper bound
0	$\frac{1}{2}$	1	1
1	$2 \times \frac{1}{4} = \frac{1}{2}$	$\frac{1}{2} + \frac{1}{3}$	$2 \times \frac{1}{2} = 1$
2	$4 \times \frac{1}{8} = \frac{1}{2}$	$\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}$	$4 \times \frac{1}{4} = 1$

k-1	$2^{k-1} \times \frac{1}{2^k} = \frac{1}{2}$	$\frac{1}{2^{k-1}} + \frac{1}{2^{k-1}+1} \dots + \frac{1}{2^k-1}$	$2^{k-1} \times \frac{1}{2^{k-1}} = 1$
k	0	$\frac{1}{2^k} = \frac{1}{n}$	1

Therefore, $H_n < \sum_{i=0}^k 1 = k + 1 = \log_2 n + 1 = O(\log n)$ and $H_n > \sum_{i=0}^{k-1} \frac{1}{2} + 0 = \frac{k}{2} = \frac{\log_2 n}{2} = \Omega(\log n)$. So, $H_n = \sum_{i=1}^n \frac{1}{i} = \Theta(\log n)$.

1.3 Introduction to Algorithms

1. What is an algorithm?

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions.

2. Examples of algorithms

(a) adding two numbers

Input: 2 numbers $x = \overline{x_n x_{n-1} \dots x_1}, y = \overline{y_n y_{n-1} \dots y_1}$.

Output: A number $z = \overline{z_{n+1} z_n \dots z_1}$, such that $z = x + y$.

```

1 /*We assume x, y are arrays of length n, z is of length n+1 */
2 int c = 0; // offset
3 for (int i = 0; i < n; ++i){
4     z[i] = x[i] + y[i] + c;
5     if (z[i] >= 10) {
6         c = 1;
7         z[i] = z[i] - 10;
8     }else c = 0;
9 }
10 z[n] = c;

```