CSE331 Introduction to Algorithms
Lecture 3: Asymptotic Notations II

Antoine Vigneron
antoine@unist.ac.kr

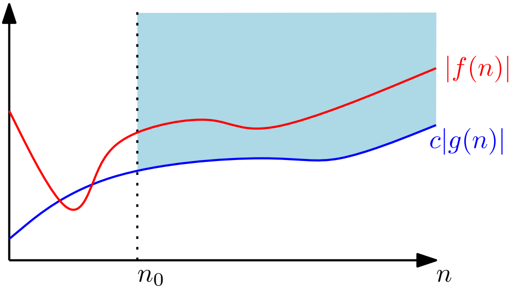
Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

- References for this lecture:
  - Lecture notes posted on blackboard.
  - Chapter I-3: *Growth of Functions* of the textbook presents it differently.

Big-Ω Notation



Definition

We write $f(n) = \Omega(g(n))$ if there exist two constants $c > 0$ and $n_0 \in \mathbb{N}$ such that $n \geqslant n_0$ implies $|f(n)| \geqslant c|g(n)|$. In other words, $f(n) = \Omega(g(n))$ means that $g(n) = O(f(n))$.

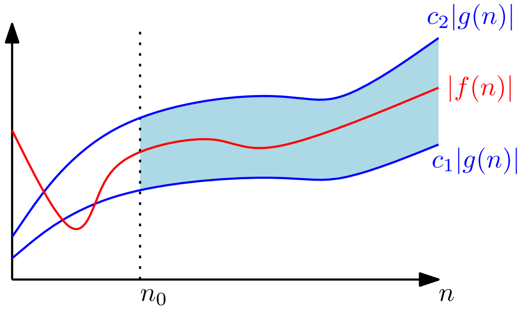
Big-Ω Notation

- It means that $|f(n)|$ is *at least* a constant factor times $|g(n)|$ for large enough n.
- $g(n)$ is an *asymptotic lower bound* on $f(n)$.

Example

It can be proved (see later this semester) that any sorting algorithm takes $\Omega(n \log n)$ time in the worst case. It means that the worst-case running time of any sorting algorithm is at least $cn \log n$ for some constant $c > 0$, and for large enough n.

Big-Θ Notation



Definition

We write $f(n) = \Theta(g(n))$ if there exist three constants $c_1 > 0$, $c_2 > 0$ and $n_0 \in \mathbb{N}$ such that $n \geqslant n_0$ implies $c_1|g(n)| \leqslant |f(n)| \leqslant c_2|g(n)|$. In other words, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Big-Θ Notation

- Interpretation: $f(n) = \Theta(g(n))$ if $f(n)$ and $g(n)$ are within a constant factor from each other for large enough n.
- We say that $g(n)$ is an *asymptotically tight bound* for $f(n)$.
- Let $T_1(n)$ and $T_2(n)$ be defined as in Lecture 2:

$$T_1(n) = 4n^2 - 3n + 6 \text{ and } T_2(n) = 3n^2 + 6n - 2.$$

  Then we have $T_1(n) = \Theta(T_2(n))$.

Properties

Proposition

The relation Θ is an equivalence relation:

- $f(n) = \Theta(f(n))$. *(Reflexivity)*
- $f(n) = \Theta(g(n))$ *iff* $g(n) = \Theta(f(n))$. *(Symmetry)*
- $f(n) = \Theta(g(n))$ *and* $g(n) = \Theta(h(n))$ *implies* $f(n) = \Theta(h(n))$. *(Transitivity)*

Proof.

Follows immediately from the definition. □

Properties

Proposition

1. *Let* $\lambda \neq 0$ *be a constant. Then* $\lambda f(n) = \Theta(f(n))$.
2. *If* $f_1(n) = \Theta(g_1(n))$ *and* $f_2(n) = \Theta(g_2(n))$, *then* $f_1(n) f_2(n) = \Theta(g_1(n) g_2(n))$.
3. *If* $f(n) = o(g(n))$, *then* $f(n) + g(n) = \Theta(g(n))$.
4. *If there exists* n_0 *such that* $0 \leqslant f(n) \leqslant g(n)$ *for all* $n \geqslant n_0$, *then* $f(n) + g(n) = \Theta(g(n))$.
5. *If* $f(n)$ *is a degree-*d *polynomial, then* $f(n) = \Theta(n^d)$.

- Proofs in lecture notes.

Limits and $\Theta(\cdot)$ notation

Proposition

If $g(n) \neq 0$ for large enough n, and

$$\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| = \ell$$

for some $0 < \ell < \infty$, then $f(n) = \Theta(g(n))$.

Example

If

$$f(n) = n\log^2(n) - n\log(n) + 2,$$
$$g(n) = 2n\log^2(n) - 3n,$$

then $\lim_{n\to\infty} f(n)/g(n) = 1/2$ so $f(n) = \Theta(g(n))$.

Limits and $\Theta(\cdot)$ notation

- But the converse is not true: $f(n) = \Theta(g(n))$ does not imply that $\left|\frac{f(n)}{g(n)}\right|$ has a finite limit.
- Example?

Absolute Values

- The definitions of $O(\cdot)$, $\Theta(\cdot)$, and $\Omega(\cdot)$ use absolute values $|f(n)|$ and $|g(n)|$ instead of $f(n)$ and $g(n)$.
- In this course, most functions are running times, so they are always positive.
- Only lower-order terms can be negative.
- So you may think of all the function as being positive, and ignore absolute values.

Lower Order Terms

- $f(n) = O(n^2)$ means exactly the same as $f(n) = O(7n^2 - 5n + 12)$.
- Similarly, $\Theta(n^3)$ is the same as $\Theta(2n^3 + 3n \log n - 5)$.
- $o(5)$ is the same as $o(1)$.
- In practice: Always remove constant factors and *lower order terms*, as they play no role, and can lead to mistakes.

Asymptotic Notation in Equations

Example

- The recurrence relation for the worst-case running time $T(n)$ of MERGE SORT can be written:

$$T(n) = 2T(n/2) + \Theta(n).$$

- It means that there is a function f such that $f(n) = \Theta(n)$ and

$$T(n) = 2T(n/2) + f(n).$$

Example

- $S(n) = \sum_{i=1}^{n} O(i)$
- It means that there is a function $g(n) = O(n)$ such that $S(n) = \sum_{i=1}^{n} g(i)$.
- So $S(n) = O(n^2)$.

Algorithms Analysis

- We will use asymptotic notations to analyze algorithms running times.

Examples

Algorithm	Worst-case running time on input of size n
BINARY SEARCH	$\Theta(\log n)$
LINEAR SEARCH	$\Theta(n)$
MERGE SORT	$\Theta(n \log n)$
INSERTION SORT	$\Theta(n^2)$

- Unknown constants and lower order-terms disappear.
- $\Theta(\cdot)$ says that there is a tight bound. For instance, the table above shows that there are two constants $c_1, c_2 > 0$ such that the worst-case running time of INSERTION SORT is at least $c_1 n^2$ and at most $c_2 n^2$ for large enough n.
- Often these bounds are given using $O(\cdot)$, so only an upper bound is claimed.

Algorithms Analysis: Example

Algorithm	Worst-case running time	Bound
Insertion sort	$T_1(n) = 4n^2 - 3n + 6$	$\Theta(n^2)$
Bubble sort	$T_2(n) = 3n^2 + 6n - 2$	$\Theta(n^2)$
Merge sort	$T_3(n) = 8n \log n + 7n$	$\Theta(n \log n)$

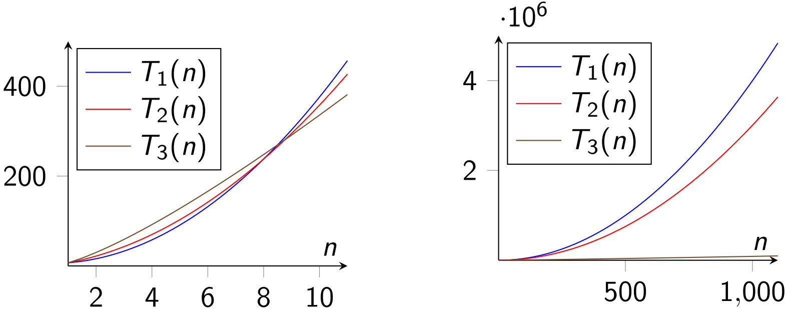


Figure: $T_3(n) = o(T_1(n))$, $T_3(n) = o(T_2(n))$, $T_1(n) = \Theta(T_2(n))$

Classes of Running Times

Time bound	Class name
$T(n) = O(1)$	constant
$T(n) = O(\log n)$	logarithmic
$T(n) = O(n)$	linear
$T(n) = O(n^2)$	quadratic
$T(n) = O(n^k)$	polynomial
$T(n) = O(2^{n^k})$	exponential

Table: n is the input size, k is a constant.

Example

We say that the running time of INSERTION SORT is *quadratic*.

- Next slide shows that these classes are very different.

Classes of Running Times

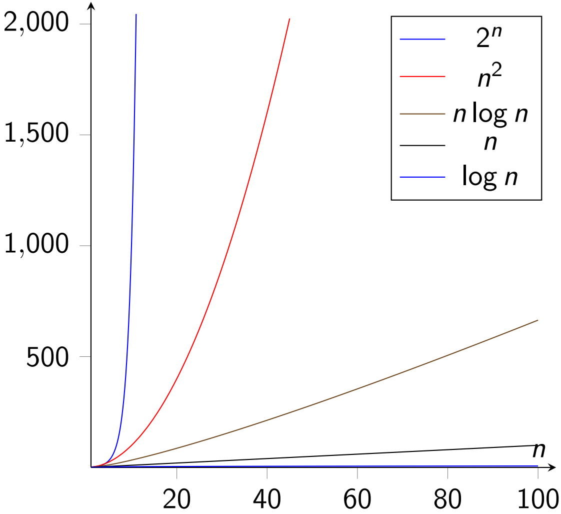


Figure: $\log n \prec n \prec n \log n \prec n^2 \prec 2^n$.

Classes of Running Times

- It is often desirable to find a *polynomial-time* algorithm for the problem being considered.
  - That is, we want the worst-case running time to be $O(n^k)$ for some constant k.
- On the other hand, *exponential time* algorithms are often considered too slow. In particular, if the worst case running time is $\Omega(2^n)$, we can only solve small instances of the problem.
- Unfortunately, for many problems, the best known algorithms are exponential. In particular, it is true for **NP**-hard problems. (Will be described later this semester.)