CSE530 Algorithms & Complexity
Lecture 2: Asymptotic Notations

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

February 26, 2018

Introduction

- References:
  - Lecture notes posted on blackboard.
  - Chapter I-3: *Growth of Functions* of the textbook presents it differently.
- We will consider real-valued functions of integer variables.
  - Can also be seen as *sequences*.
- Usually values will be positive as we are interested in running times.
- We will study their *asymptotic* behavior, i.e. at infinity.
- In particular, we will introduce the *asymptotic notations*

$$o(\cdot),\ O(\cdot),\ \Theta(\cdot),\ \Omega(\cdot), \omega(\cdot)$$

Little-o Notation

Definition (Little-o notation)

We write $f(n) = o(g(n))$ if and only if there exists a function $\rho(n)$ such that $f(n) = \rho(n)g(n)$ for all n, and $\lim_{n\to\infty} \rho(n) = 0$.

- Simpler definition if $g(n) \neq 0$ for all n:

$$f(n) = o(g(n)) \text{ iff } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

- We do not even need $g(n) \neq 0$ for all n. It suffices, for instance, that $g(n) \neq 0$ for all $n > 100$.
- Example: $n = o(n^2)$, because $n/n^2 = 1/n \to 0$.

Motivation

- Worst case running times:

$$\text{Insertion Sort:} \quad T_1(n) = a_1 n^2 + b_1 n + c_1$$
$$\text{Merge Sort:} \quad T_3(n) = a_3 n \log n + b_3 n$$

  where $a_1 > 0$, $a_3 > 0$, b_1, b_3, c_1 are unknown constants.
- For instance, we could have $T_1(n) = 4n^2 - 3n + 6$ and $T_3(n) = 8n \log n + 7n$.
- In any case, regardless of the values of these constants, $T_3(n)/T_1(n) \to 0$ and thus $T_3(n) = o(T_1(n))$.
- It shows that Merge Sort is much faster than Insertion Sort for large values of n, i.e. for sorting a large array.
- So the $o(\cdot)$ notation allows us to compare these functions without even knowing the constants.

Examples

- What does $f(n) = o(1)$ mean?
- It means $\lim_{n\to\infty} f(n) = 0$.

$$
\begin{aligned}
1/n + 1/n^2 &= o(1) \\
2n + 5 &= o(n^2) \\
10n\log n + 7n + 5 &= o(n^2) \\
n^2 + 2n + 1 &= o(n^3) \\
n^{10} + 5n^3 &= o(2^n) \\
2^n + 3^n + 4^n &= o(n!)
\end{aligned}
$$

Reformulation of the Definition

Definition

We say that $f(n) = o(g(n))$ if, for every real number $\varepsilon > 0$, there exists an integer n_0 such that $n \geqslant n_0$ implies $|f(n)| \leqslant \varepsilon |g(n)|$.

- Using quantifiers:

$$\forall \varepsilon > 0 \ \ \exists n_0 : \ \ n \geqslant n_0 \Rightarrow |f(n)| \leqslant \varepsilon |g(n)|$$

- Example: Prove that $2n + 5 = o(n^2)$.

Properties

Proposition

For all real numbers α, β,

(a) $\log n = o(n^\alpha)$ *whenever* $\alpha > 0$.

(b) $n^\alpha = o(n^\beta)$ *whenever* $\alpha < \beta$.

(c) $n^\beta = o(\gamma^n)$ *whenever* $\gamma > 1$.

(d) $\gamma^n = o(n!)$

The relation $f(n) = o(g(n))$ is sometimes denoted $f(n) \prec g(n)$. (*Hardy* notation.) So the proposition above can be rewritten:

$$\log n \prec n^\alpha \prec n^\beta \prec \gamma^n \prec n! \qquad \text{whenever } 0 < \alpha < \beta \text{ and } \gamma > 1$$

Properties

Proposition

For every functions f, g and h, and for every constant $\lambda > 0$,

(i) $f(n) = o(h(n))$ *implies* $\lambda f(n) = o(h(n))$.

(ii) $f(n) = o(h(n))$ *and* $g(n) = o(h(n))$ *implies* $f(n) + g(n) = o(h(n))$.

(iii) $f(n) = o(h(n))$ *implies* $f(n)g(n) = o(g(n)h(n))$.

(iv) $f(n) = o(g(n))$ *and* $g(n) = o(h(n))$ *implies* $f(n) = o(h(n))$.

- Example: Prove that $5n^2 + 2n + 4 = o(n^3)$ using these properties.

O-Notation

- Suppose Insertion Sort and Bubble Sort have running times

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

- Do we have $T_1(n) = o(T_2(n))$ or $T_2(n) = o(T_1(n))$?
  - No, both statements are wrong, because $T_1(n)/T_2(n) \to 4/3$.
- The big-O notation allows to compare these two functions:

Definition

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

- Difference with $o(\cdot)$:
  "There *exists* a constant c" instead of "For all $\varepsilon > 0$"

O-notation

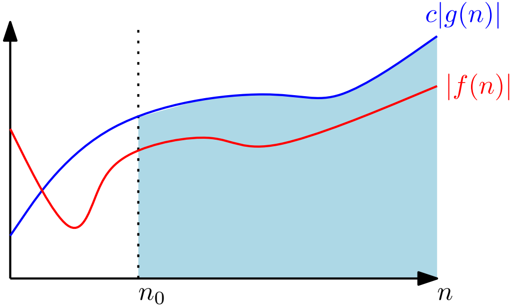


Figure: $f(n) = O(g(n))$

- The relation $f(n) = O(g(n))$ gives an *asymptotic upper bound* on $f(n)$.
- Intuition: $f(n)$ is at most a constant factor times $g(n)$ for large enough n.

Examples

- Let $T_1(n) = 4n^2 - 3n + 6$ and $T_2(n) = 3n^2 + 6n - 2$.
- Prove that $T_1(n) = O(T_2(n))$.
- Remark: $T_2(n) = O(T_1(n))$ is also true.

- Caveat: $f(n) = O(g(n))$ does *not* mean that there exists a constant ℓ such that $\lim_{n\to\infty} f(n)/g(n) = \ell$.
- Example?

- $f(n) = O(1)$ means that $f(n)$ is *bounded*: there exists a constant $C > 0$ such that $|f(n)| \leqslant C$ for all $n \in \mathbb{N}$. In computer science, we often say that $f(n)$ is *constant* when $f(n) = O(1)$.

Properties

Proposition

For all functions f, g, h, φ and for all constant $\lambda > 0$,

(i) $f(n) = O(f(n))$

(ii) $f(n) = o(g(n))$ *implies* $f(n) = O(g(n))$.

(iii) $f(n) = O(h(n))$ *implies* $\lambda f(n) = O(h(n))$.

(iv) $f(n) = O(h(n))$ *and* $g(n) = O(h(n))$ *implies* $f(n) + g(n) = O(h(n))$.

(v) $f(n) = O(h(n))$ *and* $g(n) = O(\varphi(n))$ *implies* $f(n)g(n) = O(h(n)\varphi(n))$.

(vi) $f(n) = o(g(n))$ *and* $g(n) = O(h(n))$ *implies* $f(n) = o(h(n))$.

(vii) $f(n) = O(g(n))$ *and* $g(n) = o(h(n))$ *implies* $f(n) = o(h(n))$.

(viii) $f(n) = O(g(n))$ *and* $g(n) = O(h(n))$ *implies* $f(n) = O(h(n))$.

Interpretation

- Hardy's notation: We write

$$f(n) \prec g(n) \text{ iff } f(n) = o(g(n))$$
$$f(n) \preccurlyeq g(n) \text{ iff } f(n) = O(g(n))$$

- Then (i) means: $f(n) \preccurlyeq f(n)$.
- (viii) means: $f(n) \preccurlyeq g(n)$ and $g(n) \preccurlyeq h(n)$ implies $f(n) \preccurlyeq h(n)$.
- (vii) means: $f(n) \preccurlyeq g(n)$ and $g(n) \prec h(n)$ implies $f(n) \prec h(n)$.
- So $o(\cdot)$ is similar with $<$, and $O(\cdot)$ is similar with $\leqslant$.

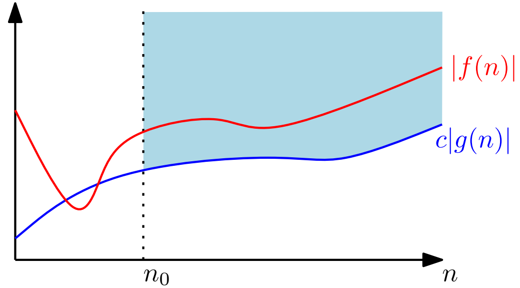
Applications

- Exercise: Prove that $f(n) = 5n^3 - 2n^2 + 5$ satisfies $f(n) = O(n^3)$ using the properties above.
- More generally:

Proposition

We say that $f(n)$ is a degree-d polynomial in n if there are constants $a_0, \ldots, a_d$ such that $a_d \neq 0$, and $f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$ for all n. If $f(n)$ satisfies these conditions, then $f(n) = O(n^d)$.

Big-Ω Notation



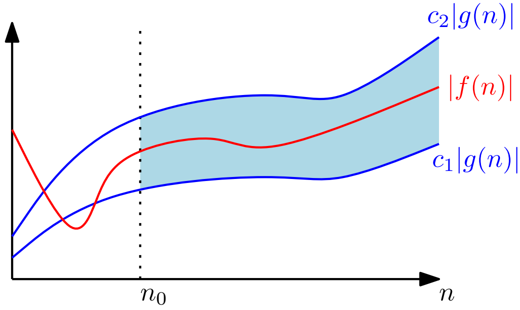
Definition

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 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 $c > 0$.

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 12:

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

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

(i) *Let* $\lambda \neq 0$ *be a constant. Then* $\lambda f(n) = \Theta(f(n))$.

(ii) *If* $f_1(n) = \Theta(g_1(n))$ *and* $f_2(n) = \Theta(g_2(n))$, *then* $f_1(n)f_2(n) = \Theta(g_1(n)g_2(n))$.

(iii) *If* $f(n) = o(g(n))$, *then* $f(n) + g(n) = \Theta(g(n))$.

(iv) *If there exists* n_0 *such that* $0 \leqslant f(n) \leqslant g(n)$ *for all* $n \geqslant n_0$, *then* $f(n) + g(n) = \Theta(g(n))$.

(v) *If* $f(n)$ *is a degree-d polynomial, then* $f(n) = \Theta(n^d)$.

- Proofs in lecture notes.

ω-Notation

Definition (Little-ω notation)

We write $f(n) = \omega(g(n))$ if and only if $g(n) = o(f(n))$.

- Simpler definition if there exists n_0 such that $f(n) \neq 0$ for all $n \geqslant n_0$, then

$$f(n) = \omega(g(n)) \text{ iff } \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

- So $f(n) = \Omega(g(n))$ means that $f(n)$ grows much faster than $g(n)$.

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 1
  - 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 2
  - $S(n) = \sum_{i=1}^{n} O(i)$
  - 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

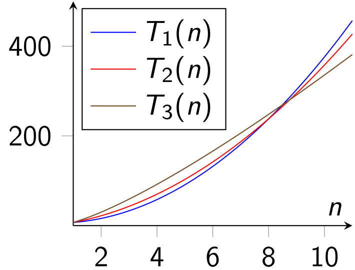
- We will use asymptotic notations to analyze algorithms running times.
- Examples:

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 $c_1 > 0$, c_2 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	$T_3(n) = 8n \log n + 7n$	$\Theta(n \log n)$



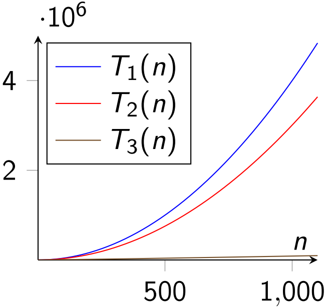


Figure: $T_3(n) = o(T_1(n))$, $T_3(n) = o(T_2(n))$, $T_1(n) = \Theta(T_2(n))$

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

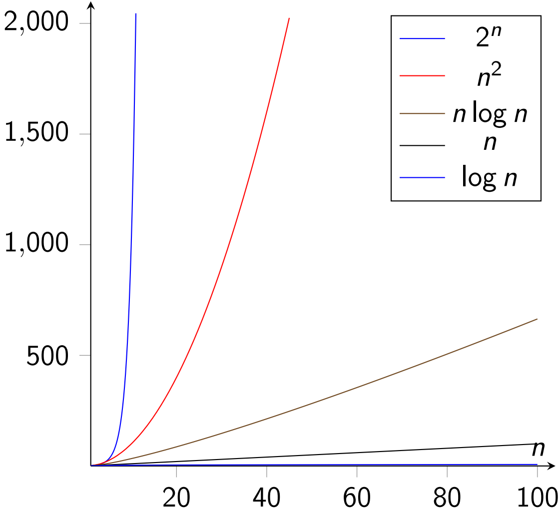


Figure: $\log n \prec n \prec n \log n \prec n^2 \prec 2^n$.

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 $O(n^d)$ for some constant d.
- 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.)