Introduction to Algorithm Analysis
Part 2

David Fernández-Baca

Computer Science 311
Iowa State University

January 20, 2022

Examples and Further Properties of O-Notation

Example 1

Proposition

$5n^2 + 3n \log n + 2n + 5$ *is* $O(n^2)$.

Proof.

$$\log_2 n \le n, \text{ for } n \ge 1.$$

Therefore,

$$5n^2 + 3n \log n + 2n + 5 \le (5 + 3 + 2 + 5)n^2 = 15n^2, \text{ for } n \ge 1.$$

Hence, choose $c = 15$ and $n_0 = 1$. □

Example 2

Proposition

$2n + 70 \log n$ *is* $O(n)$.

Proof.

$$2n + 70 \log n \leq 72n, \text{ for } n \geq 1.$$

Hence, choose $c = 72$ and $n_0 = 1$. □

Example 3

Proposition

$3 \log n + 2$ *is* $O(\log n)$.

Proof.

$$3 \log n + 2 \leq 5 \log n, \text{ for } n \geq 2.$$

Hence, choose $c = 5$ and $n_0 = 2$.

Note. We use $n_0 = 2$ instead of 1, because $\log n = 0$ for $n = 1$.

□

Example 4

Proposition

2^{n+2} is $O(2^n)$.

Proof.

$$2^{n+2} = 2^2 \cdot 2^n = 4 \cdot 2^n, \text{ for } n \geq 1$$

Hence, choose $c = 4$ and $n_0 = 1$. □

Example 5

Proposition

n^2 *is not* $O(n)$.

Proof.

- Recall: If $f(n) = O(g(n))$, then $\exists c > 0$ such that

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \le c.$$

- But here $f(n) = n^2$, $g(n) = n$ and

$$\lim_{n \to \infty} \frac{n^2}{n} = n \not\le c \quad \text{for any fixed } c > 0.$$

□

Example 6

Proposition

3^n *is not* $O(2^n)$.

Proof.

$$\lim_{n\to\infty} \frac{3^n}{2^n} = \left(\frac{3}{2}\right)^n \not\leq c \quad \text{for any fixed } c > 0.$$

□

Properties of O-notation

Polynomials

Proposition (A polynomial is big-O of its leading term)

If $f(n)$ is a polynomial of degree d, that is,

$$f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d,$$

and $a_d > 0$, then $f(n)$ is $O(n^d)$.

Proof.

$$1 \leq n \leq n^2 \leq \cdots \leq n^d, \quad \text{for } n \geq 1.$$

Thus,

$$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \leq (|a_0| + |a_1| + |a_2| + \cdots + |a_d|) n^d.$$

Hence, choose

$$c = |a_0| + |a_1| + |a_2| + \cdots + |a_d| \quad \text{and} \quad n_0 = 1.$$

□

Theorem (Properties of O-notation)

1. *(Reflexivity)* f *is* $O(f)$.
2. *(Constants)* *If* f *is* $O(g)$ *and* $c > 0$, *then* $c \cdot f$ *is* $O(g)$.
3. *(Products)* *If* f_1 *is* $O(g_1)$ *and* f_2 *is* $O(g_2)$, *then* $f_1 \cdot f_2$ *is* $O(g_1 \cdot g_2)$.
4. *(Sums)* *If* f_1 *is* $O(g_1)$ *and* f_2 *is* $O(g_2)$, *then* $f_1 + f_2$ *is* $O(\max\{g_1, g_2\})$.
5. *(Transitivity.)* *If* f *is* $O(g)$ *and* g *is* $O(h)$, *then* f *is* $O(h)$.

Proof of 3: $f_1 = O(g_1)$ and $f_2 = O(g_2) \Rightarrow f_1 \cdot f_2 = O(g_1 \cdot g_2)$.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that
$$0 \leq f_1(n) \leq c_1 \cdot g_1(n), \text{ for all } n \geq n_1.$$
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that
$$0 \leq f_2(n) \leq c_2 \cdot g_2(n), \text{ for all } n \geq n_2.$$
- Then,
$$0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n), \text{ for all } n \geq \max\{n_1, n_2\}.$$
- Big-O bound follows by taking $c = c_1 \cdot c_2$ and $n_0 = \max\{n_1, n_2\}$.

$\square$

Implication 1: Consecutive Operations

Property

An algorithm that consists of a $O(f)$-time step followed by an $O(g)$-time step, takes $O(\max\{f, g\})$ time.

Example

If $f(n) = n^2$ and $g(n) = n$, then total time is $O(n^2)$.

Implication 2: Loops

Property

If a $O(f)$ operation is repeated $O(g)$ times, the total time is $O(f \cdot g)$.

Example

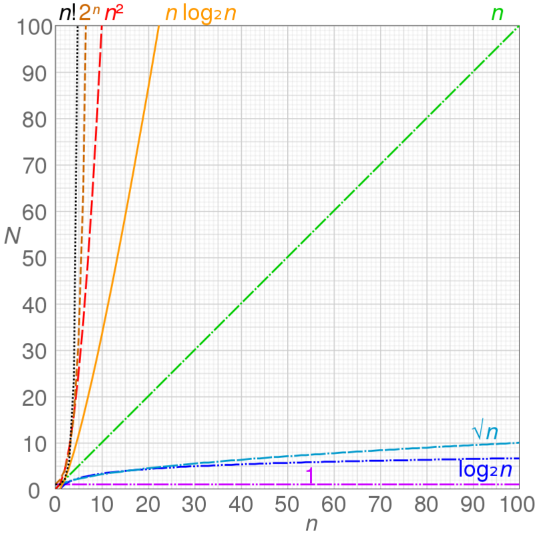
If an $O(n^2)$ operation is performed $O(n \log n)$ times, the total time is $O(n^3 \log n)$.

Classifying Functions by Growth Rate

A Hierarchy or Running Times

- Constant, $O(1)$, functions don't grow at all.
  - Any operation that does not depend on input size is $O(1)$; e.g., assignments, comparisons, and increments.
- Logarithmic, $O(\log n)$, functions grow more slowly than
- Linear, $O(n)$, functions, which grow more slowly than
- Linearithmic, $O(n \log n)$, functions, which grown more slowly than
- Quadratic, $O(n^2)$, functions, which are a special case of
- Polynomial functions — i.e., $O(n^k)$ functions, k constant — which grow more slowly than
- Exponential functions — i.e., α^n functions, $\alpha > 1$.

A Hierarchy of Running Times



Source: Wikipedia

A Hierarchy of Running Times

Clock rate:	1,000,000,000
seconds/day	86400
seconds/year	31536000

size	log n	n	n log n	n^2	n^3	2^n
10	3 ns	0.00001 ms	0.00003 ms	0.0001 ms	0.0010 ms	0.00102 ms
20	4 ns	0.00002 ms	0.00009 ms	0.0004 ms	0.0080 ms	1.04858 ms
30	5 ns	0.00003 ms	0.00015 ms	0.0009 ms	0.0270 ms	1.0737 s
50	6 ns	0.00005 ms	0.00028 ms	0.0025 ms	0.1250 ms	13.0312 days
100	7 ns	0.00010 ms	0.00066 ms	0.0100 ms	1.0000 ms	4.0E+13 years
1000	10 ns	0.00100 ms	0.00997 ms	1.0000 ms	1000.0000 ms	3.4E+284 years
10000	13 ns	0.01000 ms	0.13288 ms	0.1000 s	1000.0000 s	#NUM!
100000	17 ns	0.10000 ms	1.66096 ms	10.0000 s	11.5741 days	#NUM!
1000000	20 ns	1.00000 ms	19.93157 ms	1000.0000 s	31.7098 years	#NUM!

Source: Steve Kautz

Exponential Time

Subset Sum

Input: An array A with n elements and a number K.

Question: Does A contain a subset that adds up to exactly K?

Notes

- Fastest known algorithms for Subset Sum take time exponential in n.
- Subset Sum is NP-complete, and thus unlikely to have a polynomial-time algorithm.

An Exponential Time Algorithm for Subset Sum

Algorithm

- Enumerate all subsets of the elements of A.
- For each subset, see if its elements add up to K.

Analysis

- Two choices for each i: include or exclude $A[i]$.

$$\Rightarrow \text{number of subsets to enumerate} = \underbrace{2 \cdot 2 \cdot 2 \cdots 2}_{n \text{ times}} = 2^n$$

- Time to add up each set is $O(n)$.

$$\Rightarrow \quad \text{Algorithm takes } O(n \cdot 2^n) \text{ time.}$$

More Asymptotic Notation

Beyond O-notation

Reminder

O-notation expresses upper bounds.

Other useful notations

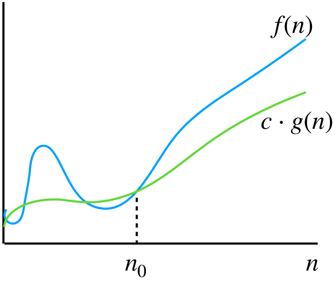
- Ω-notation: For lower bounds.
- Θ-notation: For exact bounds.
- o-notation: For strict upper bounds.
- ω-notation: For strict lower bounds.

Ω-notation

Definition

$f(n)$ is $\Omega(g(n))$ if and only if there exist positive constants c and n_0 such that

$$0 \leq c \cdot g(n) \leq f(n), \quad \text{for all } n \geq n_0.$$



$f(n) = \Omega(g(n))$

Ω-notation

Example

$3n \log n - 2n$ is $\Omega(n \log n)$.

Justification

$$3n \log n - 2n = n \log n + 2n(\log n - 1) \geq n \log n, \quad \text{for } n \geq 2.$$

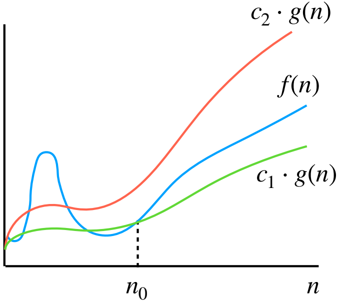
Thus, we can take $c = 1$ and $n_0 = 2$.

Θ-notation

Definition

$f(n)$ is $\Theta(g(n))$ if and only if there exist positive constants c_1, c_2, and n_0 such that

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \quad \text{for all } n \geq n_0.$$



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

Θ-notation

Example

$3n \log n + 4n + 5 \log n$ is $\Theta(n \log n)$.

Justification

$$3n \log n \leq 3n \log n + 4n + 5 \log n \leq (3 + 4 + 5) n \log n, \quad \text{for } n \geq 2.$$

Thus, we can take $c_1 = 3$, $c_2 = 12$, and $n_0 = 2$.

Θ, O, and Ω

Theorem

For any two functions $f(n)$ and $g(n)$,

$$f(n) = \Theta(g(n)) \quad \Leftrightarrow \quad f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

Little-o and Little-ω
(Not covered in class)

o-notation

Definition

$f(n)$ is $o(g(n))$ if and only if for every positive constant c, there exists a constant n_0 such that

$$0 \leq f(n) < c \cdot g(n), \quad \text{for all } n \geq n_0.$$

Note that

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

I.e., $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity.

Using o-notation

o-notation expresses upper bounds that are not asymptotically tight.

Example

$$2n = o(n^2), \quad \text{but} \quad 2n \neq o(n).$$

ω-notation

Definition

$f(n)$ is $\omega(g(n))$ if and only if for every positive constant c, there exists a constant n_0 such that

$$0 \le c \cdot g(n) < f(n), \quad \text{for all } n \ge n_0.$$

Note that

$$f(n) = \omega(g(n)) \quad \Rightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

and

$$f(n) = \omega(g(n)) \quad \Leftrightarrow \quad g(n) = o(f(n)).$$

Using ω-notation

ω-notation expresses lower bounds that are not asymptotically tight.

Example

$$\frac{n^2}{2} = \omega(n), \quad \text{but} \quad \frac{n^2}{2} \neq \omega(n^2).$$

Bibliography

References

[CLRS] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms* (3rd edition), MIT Press, 2009.

[KT] Jon Kleinberg and Éva Tardos, *Algorithm Design*, Addison-Wesley, 2006.