Contents

1. What is asymptotic notation?

In computer science and related disciplines, asymptotic notation describes the behavior of a function as its input size (often denoted by n) tends towards infinity. It focuses on the dominant terms that influence the function's growth rate, ignoring constant factors and lower-order terms.

2. Why use asymptotic notation?

It allows researchers to compare and evaluate algorithms' efficiency without getting bogged down by specific hardware or implementation details. By focusing on asymptotic behavior, researchers can make general statements about how well algorithms scale with increasing input sizes.

3. Types of asymptotic notation

3.1. Big O notation (O-notation)

O-notation provides an asymptotic **upper bound**.

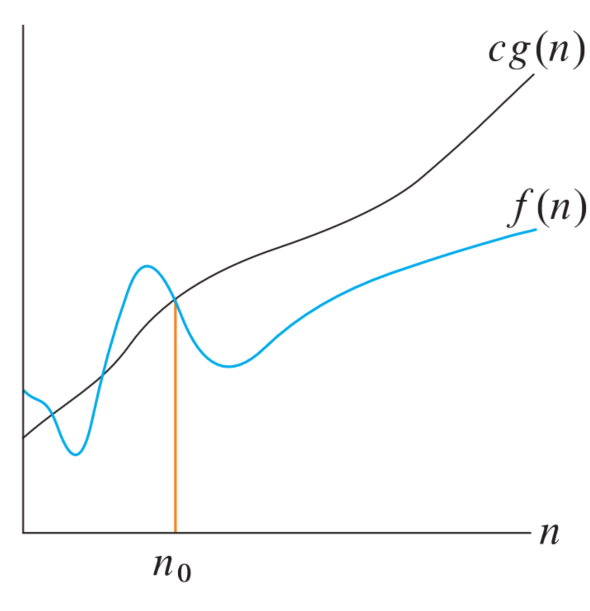


Figure 1: $f(n) = O(g(n))$

Definition 3.1.1:

$$O(g(n)) \coloneqq \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n),\ \forall n \geq n_0\}$$

Definition 3.1.2:

$$f(n) \coloneqq O(g(n)) \Leftrightarrow f(n) \in O(g(n))$$

Example: $\ln(n) = O(n)$

3.2. Big Omega notation (Ω-notation)

Ω-notation provides an asymptotic **lower bound**.

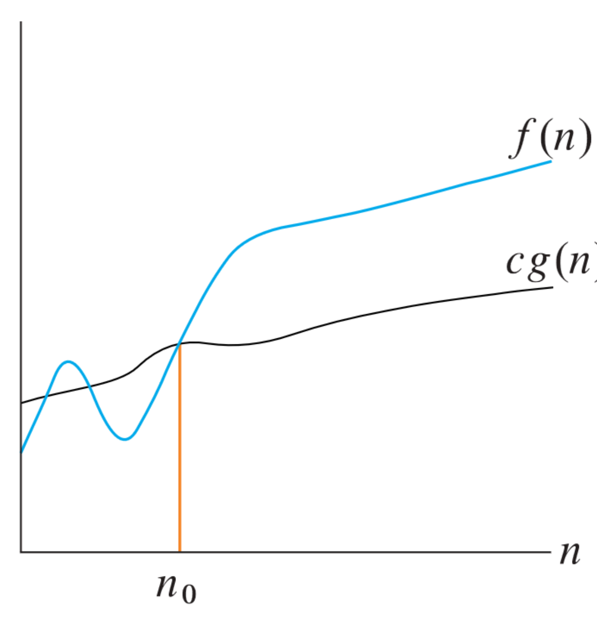


Figure 2: $f(n) = \Omega(g(n))$

Definition 3.2.1:

$$\Omega(g(n)) \coloneqq \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n),\ \forall n \geq n_0\}$$

Definition 3.2.2:

$$f(n) \coloneqq \Omega(g(n)) \Leftrightarrow f(n) \in \Omega(g(n))$$

Example: $n^2 + n = \Omega(n^2)$

3.3. Theta notation (Θ-notation)

Θ-notation provides an asymptotic **tight bound**.

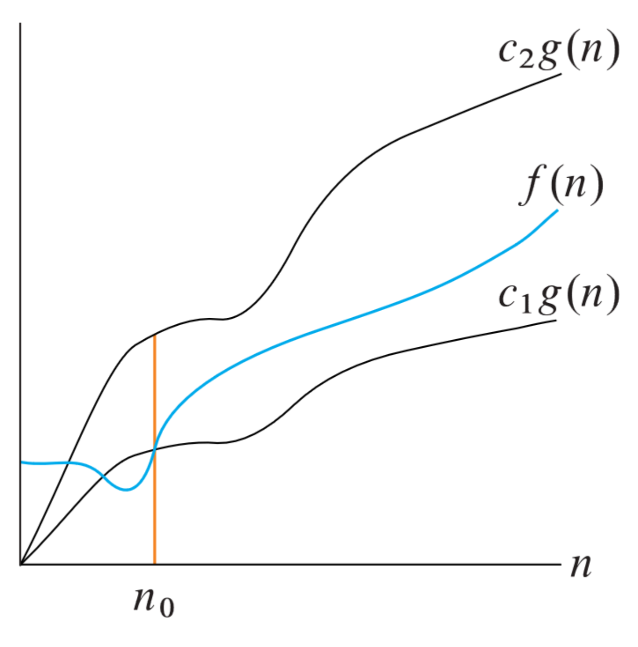


Figure 3: $f(n) = \Theta(g(n))$

Definition 3.3.1:

$$\Theta(g(n)) \coloneqq \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n),\ \forall n \geq n_0\}$$

Definition 3.3.2:

$$f(n) \coloneqq \Theta(g(n)) \Leftrightarrow f(n) \in \Theta(g(n))$$

Example: $\Theta(n^2) = n^2$

3.4. Little o notation (o-notation)

o-notation denotes an **upper bound** that is **not asymptotically tight**

Definition 3.4.1:

$$o(g(n)) \coloneqq \{f(n) : \forall \varepsilon > 0 : \exists n_0 > 0 \text{ such that } 0 \leq f(n) < \varepsilon g(n),\ \forall n \geq n_0\}$$

Proposition 3.4.1:

$$g(n) > 0 \Rightarrow o(g(n)) = \left\{f(n) : f(n) \geq 0 \text{ and } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0\right\}.$$

Definition 3.4.2:

$$f(n) \coloneqq o(g(n)) \Leftrightarrow f(n) \in o(g(n))$$

Example: $\ln(n) = o(n)$

3.5. Little omega notation (ω-notation)

ω-notation denotes an **lower bound** that is **not asymptotically tight**

Definition 3.5.1:

$$\omega(g(n)) \coloneqq \{f(n) : \forall \varepsilon > 0 : \exists n_0 > 0 \text{ such that } 0 \leq \varepsilon g(n) < f(n),\ \forall n \geq n_0\}$$

Definition 3.5.2:

$$f(n) \coloneqq \omega(g(n)) \Leftrightarrow f(n) \in \omega(g(n))$$

Proposition 3.5.1:

$$f(n) \coloneqq \omega(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \text{ , if the limit exists.}$$

Example: $n^2 = \omega(n)$

4. Properties

4.1. Transitivity

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

4.2. Reflexivity

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

4.3. Symmetry

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

4.4. Transpose symmetry

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

4.5. Some useful identities

$$\Theta(\Theta(f(n))) = \Theta(f(n))$$
$$\Theta(f(n)) + O(f(n)) = \Theta(f(n))$$
$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$
$$\Theta(f(n)) \cdot \Theta(g(n)) = \Theta(f(n) \cdot g(n))$$

5. Common types of asymptotic bound

$$p(n) := \sum_{k=0}^{d} a_k n^k,\ \forall k \geq 0 : a_k > 0$$

1. $p(n) = O\left(n^k\right),\ \forall k \geq d$
2. $p(n) = \Omega\left(n^k\right),\ \forall k \leq d$
3. $p(n) = \Theta\left(n^k\right) \text{if } k = d$
4. $p(n) = o\left(n^k\right),\ \forall k > d$
5. $p(n) = \omega\left(n^k\right),\ \forall k < d$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$\log(n!) = \Theta(n \log(n))$$

6. Methods for proving asymptotic bounds

6.1. Using definitions

Example:

$$\ln(n) \le n \ , \forall n \ge 1 \ (c = 1, \, n_0 = 1) \\ \Rightarrow \ln(n) = O(n)$$

Example:

$$0 \le n^2 \le n^2 + n \ , \forall n \ge 1 \ (c = 1, \, n_0 = 1) \\ \Rightarrow n^2 + n = \Omega(n^2)$$

Example:

$$0 \le n^2 \le n^2 + n \le 2n^2, \, \forall n \ge 1 \ (c_1 = 1, \, c_2 = 2, \, n_0 = 1) \\ \Rightarrow \Theta(n^2) = n^2$$

Example:

$$\left.\begin{aligned} &\ln(n) \ge 0, \, \forall n \ge 1 \\ &\lim_{n\to\infty} \tfrac{\ln(n)}{n} = \lim_{n\to\infty} \tfrac{1}{n} = 0 \end{aligned}\right\} \Rightarrow \ln(n) = o(n)$$

Example:

$$\forall \varepsilon > 0 : 0 \le \varepsilon n < n^2 \ , \forall n \ge \varepsilon + 1 \ (n_0 = \varepsilon + 1) \\ \Rightarrow n^2 = \omega(n)$$

6.2. Substitution method

The substitution method comprises two steps:
- Guess the form of the solution using symbolic constants.
- Use mathematical induction to show that the solution works, and find the constants.

This method is powerful, but it requires experience and creativity to make a good guess.

Example:

$$T(n) \coloneqq \begin{cases} \Theta(1), \, \forall n : 4 > n \ge 2 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + d \ (d > 0) \ , \forall n \ge 4 \end{cases}$$

To guess the solution easily, we will assume that: $T(n) = T\left(\frac{n}{2}\right) + d$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + d \\ &= T\left(\frac{n}{4}\right) + 2d \\ &= T\left(\frac{n}{2^k}\right) + (k-1)d \\ &= T(c) + \left(\log\left(\frac{n}{c}\right) - 1\right)d \\ &= d\log(n) + (T(c) - \log(c) - d) \end{aligned}$$

So we will make a guess: $T(n) = O(\log(n))$

Define $c \coloneqq \max\{T(2), T(3), d\}$
Assume $T(n) \le c\log(n) \ , \forall n : k > n$

$$\begin{aligned} T(k) &= T\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + d \\ &\le c\log\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + d \\ &\le c\log\left(\frac{k}{2}\right) + d \\ &\le c\log(k) - c + d \\ &\le c\log(k) \ (1) \end{aligned}$$

$$T(n) \le c\log(n) \forall n : 4 > n \ge 2 \quad (2)$$

From (1), (2) $\Rightarrow T(n) = O(\log(n))$

6.3. Master theorem

Theorem 6.3.1 (Master theorem):

$$T(n) \coloneqq aT\left(\frac{n}{b}\right) + f(n)$$

where:
- $a > 0$
- $b > 1$
- $\exists n_0 > 0 : f(n) > 0, \, \forall n \ge n_0$

$$\Rightarrow T(n) = \begin{cases} \Theta(n^{\log_b a}), \text{ if } \exists \varepsilon > 0 : f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a}\log(n)^{k+1}\right), \text{ if } \exists k \ge 0 : f(n) = \Theta\left(n^{\log_b a}\log(n)^k\right) \\ \Theta(f(n)), \text{ if } \begin{cases} \exists \varepsilon > 0 : f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \\ \exists n_0 > 0, \, c < 1 : af\left(\frac{n}{b}\right) \le cf(n), \, \forall n \ge n_0 \end{cases} \end{cases}$$

Example: Solve the recurrence for merge sort: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

We have $f(n) = \Theta(n) = \Theta\left(n^{\log_2 2}\log(n)^0\right)$, hence $T(n) = \Theta\left(n^{\log_2 2}\log(n)^1\right) = \Theta(n\log(n))$ (according to 2nd case of Theorem 6.3.1)

6.4. Akra-Bazzi method

Theorem 6.4.1 (Akra-Bazzi method):

$$T(x) \coloneqq g(x) + \sum_{i=1}^{k} a_i T(b_i x + h_i(x))$$

where:
- $a_i > 0, \, \forall i \ge 1$
- $0 < b_i < 1, \, \forall i \ge 1$
- $\exists c \in \mathbb{N} : |g'(x)| = O(x^c)$
- $|h_i(x)| = O\left(\frac{x}{\log(x)^2}\right)$

$$\Rightarrow T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}} \, \mathrm{d}u\right)\right)$$

where: $\sum_{i=1}^{k} a_i b_i^p = 1$

Example: Solve the recurrence: $T(x) = T\left(\frac{x}{2}\right) + T\left(\frac{x}{3}\right) + T\left(\frac{x}{6}\right) + x\log(x)$

$$|(x\log x)'| = |\log x + 1| \le x \ , \forall x \ge 1 \\ \Rightarrow |(g(x))'| = O(x) \quad (1)$$

$$|h_{i(x)}| = 0 = O\left(\frac{x}{\log(x)^2}\right) \quad (2)$$

$$\left(\frac{1}{2}\right)^1 + \left(\frac{1}{3}\right)^1 + \left(\frac{1}{6}\right)^1 = 1 \quad (3)$$

From (1), (2), and (3), we can apply Theorem 6.4.1 to get:

$$\begin{aligned} T(x) &= \Theta\left(x\left(1 + \int_1^x \frac{u\log(u)}{u^2} \, \mathrm{d}u\right)\right) \\ &= \Theta\left(x\left(1 + \int_1^x \frac{\log(u)}{u} \, \mathrm{d}u\right)\right) \\ &= \Theta\left(x\left(1 + \frac{1}{2}\log(u)^2\Big|_1^x\right)\right) \\ &= \Theta\left(x\left(1 + \frac{1}{2}\log(x)^2\right)\right) \\ &= \Theta\left(x + \frac{1}{2}x\log(x)^2\right) \\ &= \Theta\left(x\log(x)^2\right) \end{aligned}$$

7. Finding asymptotic bound of an algorithm

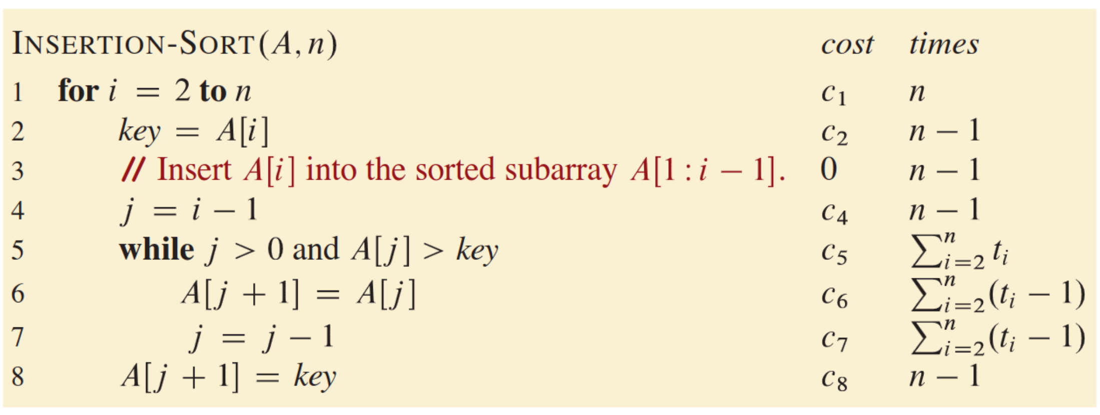
7.1. Exact step-counting analysis

The asymptotic bound of an algorithm can be calculated by following the steps below:

- Break the program into smaller segments
- Find the number of operations performed in each segment
- Add up all the number of operations, call it T(n)
- Find the asymptotic bound of T(n)

Example: Analyze insertion sort

We have the following analysis:



INSERTION-SORT(A, n)	*cost*	*times*
1 **for** $i = 2$ **to** n	c_1	n
2 $key = A[i]$	c_2	$n-1$
3 // Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
4 $j = i - 1$	c_4	$n-1$
5 **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6 $A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7 $j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8 $A[j+1] = key$	c_8	$n-1$

Figure 4: Pseudo code for insertion sort with analysis

where:

- c_k denotes the cost of k^{th} line
- t_i denotes the number of times the while loop test in line 5 is executed for given i

From the analysis, we can see that:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i \\ &+ c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) \end{aligned}$$

In the best case (when the array is already sorted), we have $t_i = 1$ for all i.

$$\begin{aligned} \Rightarrow T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} 1 \\ &+ c_6 \sum_{i=2}^{n} (1-1) + c_7 \sum_{i=2}^{n} (1-1) + c_8(n-1) \\ &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 n + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - c_2 - c_4 - c_8 \\ \Rightarrow T(n) &= \Omega(n) \end{aligned}$$

In the worst case, we have $t_i = i$ for all i.

$$\begin{aligned} \Rightarrow T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} i \\ &+ c_6 \sum_{i=2}^{n} (i-1) + c_7 \sum_{i=2}^{n} (i-1) + c_8(n-1) \\ &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) \\ &+ c_6 \frac{n(n-1)}{2} + c_7 \frac{n(n-1)}{2} + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 \\ &+ \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - c_2 - c_4 - c_5 - c_8 \\ \Rightarrow T(n) &= O(n^2) \end{aligned}$$

In conclusion, we have $T(n) = \Omega(n)$ and $T(n) = O(n^2)$

7.2. Recurrence relation

Example: Calculate asymptotic bound of merge sort

Define $T(n)$ as the running time of the algorithm.
From the implementation of merge sort, we have: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$
Applying Theorem 6.3.1, we can conclude that $T(n) = \Theta(n \log(n))$ (2nd case with $b = 2$, $a = 2$, $k = 0$)

8. Asymptotic notation and running time

When using asymptotic notation to characterize an algorithm's running time, make sure that the asymptotic notation used is as precise as possible without overstating which running time it applies to.

> *Example*:
> In average case, quick sort runs in $\Theta(n \log n)$, so it also runs in $O(n^k),\ \forall k \geq 2$
> In the same way, merge sort's running time can be $O(n^l),\ \forall l \geq 2$
>
> Take $k = 2$ and $l = 3$, it's intuitive to conclude that quick sort is faster than merge sort for large enough n since $n^2 < n^3$
> However, they both have the same asymptotic behavior (both runs in $\Theta(n \log n)$)
>
> The error occurs due to the inaccuracy of the asymptotic notation used to compare 2 algorithms.

Asymptotic notations only give a bound for the running time of an algorithm when n is large enough. Hence, comparing algorithms with asymptotic notation is only applicable for large enough n.

> *Example*:
> Linear search's running time is $\Theta(n)$ and binary search's running time is $\Theta(\log n)$ so one may attempt to conclude that binary search is faster than linear search for all n, which is a **wrong** statement.
> Consider the following benchmark:

9. References

- https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/
- https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)
- https://en.wikipedia.org/wiki/Akra%E2%80%93Bazzi_method
- https://ocw.mit.edu/courses/6-042j-mathematics-for-computer-science-fall-2010/b6c5cecb1804b69a6ad12245303f2af3_MIT6_042JF10_rec14_sol.pdf
- https://www.geeksforgeeks.org/asymptotic-notations-and-how-to-calculate-them/
- https://www.geeksforgeeks.org/step-count-method-for-time-complexity-analysis/