Asymptotic Analysis

RAM Model:

The RAM (Random Access Machine) model is a theoretical model for analyzing algorithms. It simplifies how a computer works so we can focus on the efficiency of algorithms (like time and space complexity) without getting distracted by hardware details. This model assumes a single processor and instructions execute one after another, with no concurrent operations. The RAM model assumes that each instruction (operations, data access, data movement) takes the same amount of time as any other instruction.

The RAM model contains instructions commonly found in real computers: Arithmetic (add, subtract, multiply), Data Movement (assign), Control Flow (return), and Comparison (logical operators) take constant time.

```
temp = A[0]    → 2  (data access and data assign)
if temp > 10:  → 1 (comparison)
   return temp → 1 (control flow)

Total Operations = 4
```

To compare algorithms, let us define a few objective measures:

- Execution times? It's not a good measure as execution times are specific to a particular computer.
- Number of statements executed? Not a good measure, since the number of statements varies with the programming language as well as the style of the individual programmer.
- Ideal solution? Let us assume that we express the running time of a given algorithm as a function of the input size n (i.e., f(n)) and compare these different functions corresponding to running times. This kind of comparison is independent of machine time, programming style, etc.

The running time of an algorithm on a particular input is the number of instructions and data accesses executed. How we account for these costs should be independent of any particular computer. A constant amount of time is required to execute each line. One line might take more or less time than another line, but we'll assume that each execution of the k^{th} line takes c_k time, where c_k is a constant. The running time of the algorithm is the sum of running times for each

statement executed. We usually denote the running time of an algorithm on an input of size n by T(n).
We'll usually concentrate on finding only the worst-case running time, that is, the longest running time for any input of size n. Because:

- The worst-case running time of an algorithm gives an upper bound on the running time for any input. If you know it, then you have a guarantee that the algorithm never takes any longer.
- For some algorithms, the worst case occurs fairly often. For example, in searching a database for a particular piece of information, the searching algorithm's worst case often occurs when the information is not present in the database.

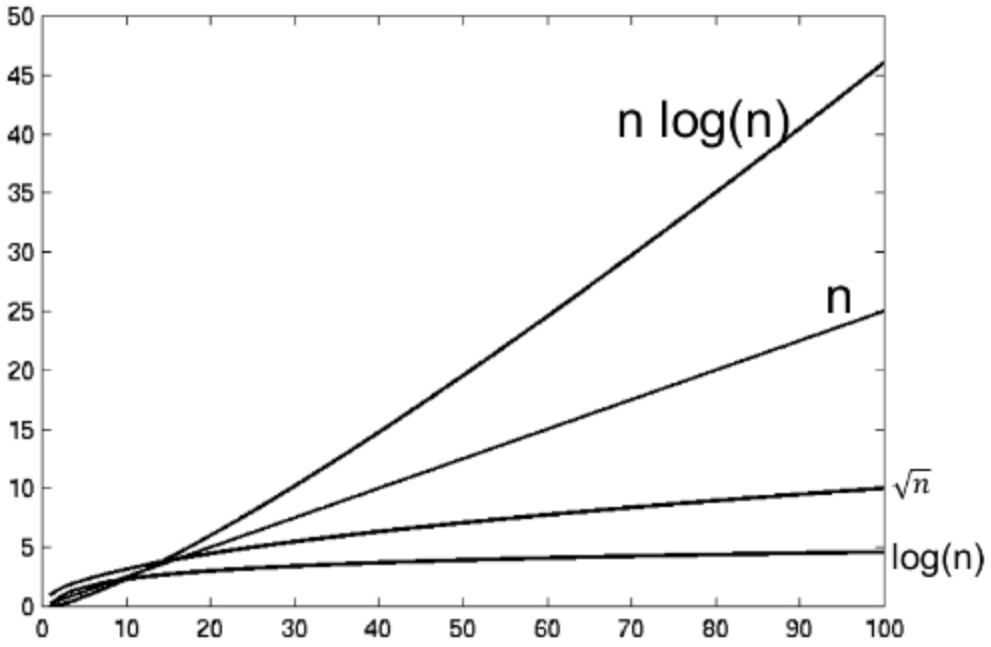
Order of growth:

We will consider only the leading term of a running time formula (e.g. an^2) since the lower-order terms are relatively insignificant for large values of n. We also ignore the leading term's constant coefficient, since constant factors are less significant than the rate of growth in determining computational efficiency for large inputs.

Typical Running Time Functions:

- 1 (constant running time): Instructions are executed once or a few times
- log N (logarithmic): A big problem is solved by cutting the original problem into smaller sizes, by a constant fraction at each step
- N (linear): A small amount of processing is done on each input element
- N log N: A problem is solved by dividing it into smaller problems, solving them independently, and combining the solution
- N^2 (quadratic): Typical for algorithms that process all pairs of data items (double nested loops)
- N^3 (cubic): Processing of triples of data (triple nested loops)
- N^K (polynomial)
- 2^N (exponential): Few exponential algorithms are appropriate for practical use

n	1	lgn	n	nlgn	n^2	n^3	2^n
1	1	0.00	1	0	1	1	2
10	1	3.32	10	33	100	1,000	1024
100	1	6.64	100	664	10,000	1,000,000	1.2×10^{30}
1000	1	9.97	1000	9970	1,000,000	10^9	1.1×10^{301}



Comparison of Orders:

$\log(\log n) < \log n < \sqrt{n} < n < n \log n < n^{3/2} < n^2 < n^2 \log n < n^3 < 2^n < e^{n+1} < n! < n^n$

Asymptotic Analysis:

Asymptotic analysis is a method for analyzing functions as their arguments tend toward infinity. Asymptotic analysis is used to approximate functions that cannot be exactly calculated. The three most common types of asymptotic analysis are:

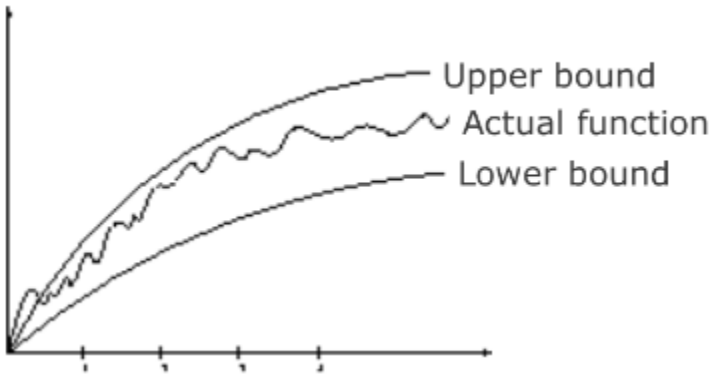
Worst Case Complexity: The function is defined by the maximum number of steps taken on any instance of size n. It represents the upper bound of an algorithm's complexity. It is represented by Big O notation (O()).

Best Case Complexity: The function is defined by the minimum number of steps taken on any instance of size n. It represents the lower bound of an algorithm's complexity. It is represented by Omega notation (Ω()).

Average Case Complexity: The function is defined by the average number of steps taken on any instance of size n. It represents the tight bound of an algorithm's complexity. It is represented by Theta notation (Θ()).

It's hard to estimate the exact running time. The best case depends on the input whereas the average case is difficult to compute. So we usually focus on worst-case analysis which is easier to compute and usually close to the actual running time

Strategy: find a function (an equation) that, for large n, is an upper bound to the actual function

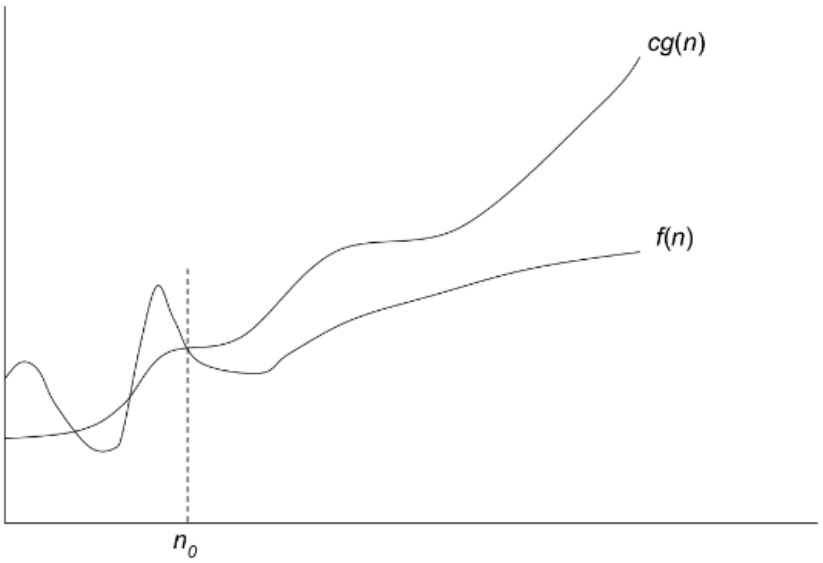


We can sometimes determine the exact running time of an algorithm but the extra precision is rarely worth the effort of computing it. For large enough inputs, the multiplicative constants and lower-order terms of an exact running time are dominated by the effects of the input size itself.

Big Oh, O: Asymptotic Upper Bound:

A function f(n) can be represented in the order of g(n) that is O(g(n)), if there exists a value of positive integer n as n_0 and a positive constant c such that −

f(n) = O(g(n)) when $0 \leq f(n) \leq c.g(n)$ for $n \geq n_0$ in all case



Prove that $2n + 3$ is $O(n)$

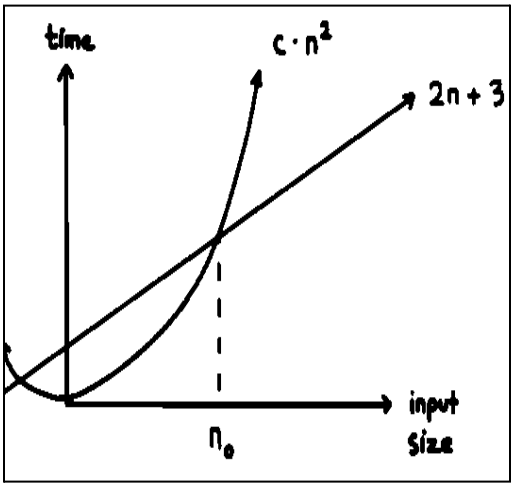
$$2n + 3 \leq 2n + 3n$$
$$2n + 3 \leq 5n$$

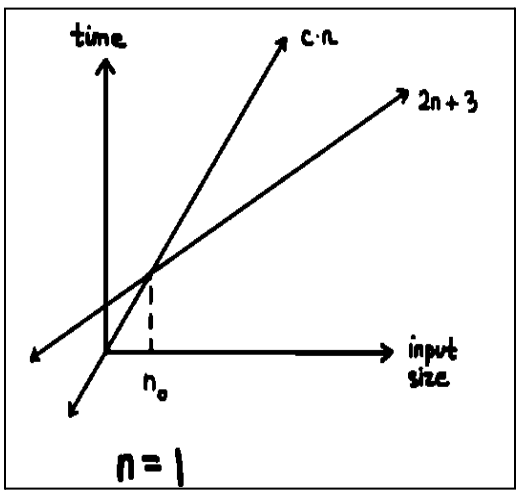
By the definition of Big O, with $c = 5$ and $n_0 = 1$,
Therefore, $2n + 3$ is $O(n)$ since $2n + 3 \leqslant 5n$ for $n \geqslant 1$

Prove that $2n + 3$ is $O(n^2)$

$$2n + 3 \leq 2n^2 + 3n^2$$
$$2n + 3 \leq 5 \cdot n^2$$

By the definition of Big O with $c = 5$ and $n_0 = 1$,
Therefore, $2n + 3$ is $O(n^2)$ since $2n + 3 \leqslant 5n^2$ for $n \geqslant 1$





Prove that 2^{n+2} is $O(2^n)$

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

By the definition of Big O, with $c = 4$ and $n_0 = 1$
Therefore, 2^{n+2} is $O(2^n)$ since $2^{n+2} \leqslant 4 \cdot 2^n$ for $n \geqslant 1$

Prove that $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$

$$5n^2 + 3n\log n + 2n + 5 \leq 5n^2 + 3n^2 + 2n^2 + 5n^2$$
$$5n^2 + 3n\log n + 2n + 5 \leq 15n^2$$

let $n_0 = 1$
$5 + 0 + 2 + 5 \leq 15$
$12 \leq 15$

By the definition of Big O, with $c = 15$ and $n_0 = 1$
Therefore, $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$
since $5n^2 + 3n\log n + 2n + 5 \leq 15n^2$ for all $n \geqslant 1$

Prove that 5^n is NOT $O(4^n)$

Suppose 5^n is $O(4^n)$
Then, there exists a positive constant c and a positive integer n_0 such that

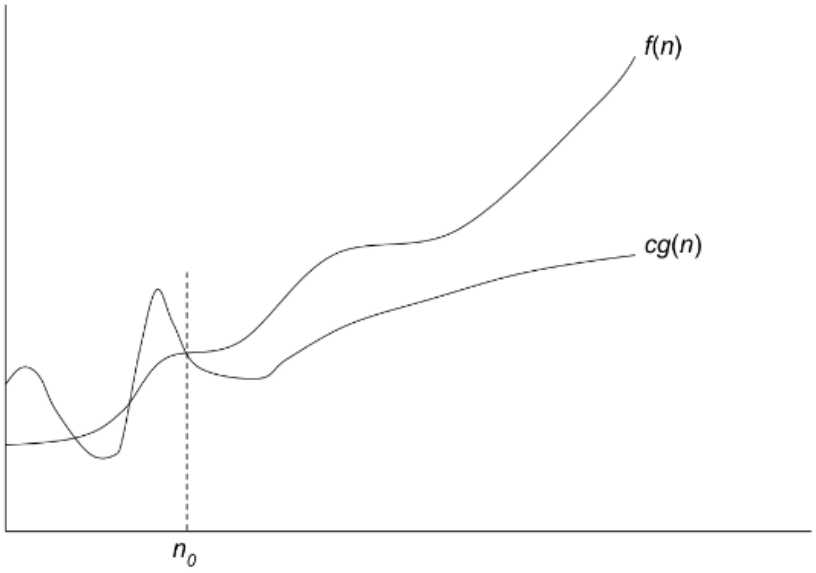
$$5^n \leq c \cdot 4^n \text{ for } n \geqslant n_0$$
$$\left(\frac{5}{4}\right)^n \leqslant c$$
$$1.25^n \leqslant c$$

This is a contradiction as 1.25^n grows infinitely so it cannot be bounded by a constant. Thus, 5^n is not $O(4^n)$

Big Omega, Ω: Asymptotic Lower Bound:

A function f(n) can be represented in the order of g(n) that is Ω(g(n)), if there exists a value of positive integer n as n_0 and a positive constant c such that −

f(n) = Ω(g(n)) when 0 ≤ f(n) ≥ c.g(n) for n ≥ n_0 in all case



Prove that $8n^3 + 5n^2 + 7$ is $\Omega\left(n^3\right)$

$$8n^3 + 5n^2 + 7 \geq 8n^3$$

$$\text{let } n_0 = 1$$
$$8 + 5 + 7 \geqslant 8$$
$$20 \geqslant 8$$

By definition of Big Ω, with $c = 8$ and $n_0 = 1$
Therefore, $8n^3 + 5n^2 + 7$ is $\Omega\left(n^3\right)$
since $8n^3 + 5n^2 + 7 \geqslant 8n^3$ for all $n \geqslant 1$

Prove that 3^n is $\Omega\left(2^n \cdot n^4\right)$

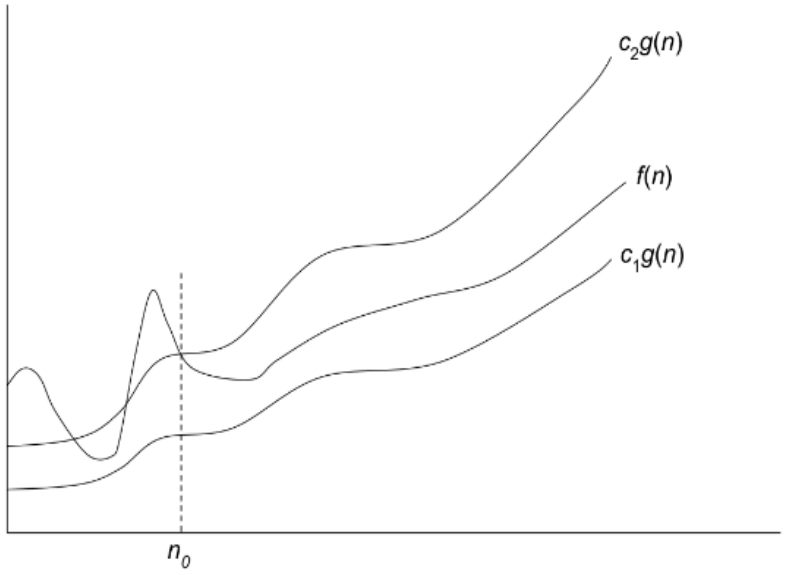
$$3^n \geqslant c \cdot 2^n \cdot n^4$$
$$\frac{3^n}{2^n} \geqslant c \cdot \frac{2^n \cdot n^4}{2^n}$$
$$1 \cdot 5^n \geqslant c \cdot n^4$$

As exponential always grows larger than polynomial
Therefore, 3^n is $\Omega\left(2^n \cdot n^4\right)$

Theta, θ: Asymptotic Tight Bound:

A function f(n) can be represented in the order of g(n) that is Θ(g(n)), if there exists a value of positive integer n as n_0 and a positive constant c such that −

$f(n) = \Theta(g(n))$ when $0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$ for $n \geq n_0$ in all case
Or
$f(n) = \Theta(g(n))$ when $f(n) = O(g(n))$ AND $f(n) = \Omega(g(n))$



Prove that $20n^3 + 7n + 1000$ is $\theta\left(n^3\right)$

Upper Bound:

$$20n^3 + 7n + 1000 \leq 20n^3 + 7n^3 + 1000n^3$$
$$20n^3 + 7n + 1000 \leq 1027n^3$$

By the definition of Big O, with $c = 1027$ and $n_0 = 1$,
Therefore, $20n^3 + 7n + 1000$ is $O\left(n^3\right)$
since $20n^3 + 7n + 1000 \leqslant 1027n^3$ for $n \geqslant 1$

Lower Bound:

$$20n^3 + 7n + 1000 \geq 20n^3$$

By the definition of Big Ω, with $c = 20$ and $n_0 = 1$,
Therefore, $20n^3 + 7n + 1000$ is $\Omega\left(n^3\right)$
since $20n^3 + 7n + 1000 \geqslant 20n^3$ for $n \geqslant 1$

Since, both are true, $20n^3 + 7n + 1000$ is $\theta\left(n^3\right)$

Prove that $n^2 + 15n - 3$ is $\theta\left(n^2\right)$

Upper Bound:

$$
\begin{aligned}
n^2 + 15n - 3 &\leq n^2 + 15n \\
n^2 + 15n - 3 &\leq n^2 + 15n^2 \\
n^2 + 15n - 3 &\leq 16n^2
\end{aligned}
$$

By the definition of Big O , with $c = 16$ and $n_0 = 1$,
Therefore, $n^2 + 15n - 3$ is $O\left(n^2\right)$
since $n^2 + 15n - 3 \leq 16n^2$ for $n \geqslant 1$

Lower Bound:

$$n^2 + 15n - 3 \geq n^2$$

By the definition of Big Ω, with $c = 1$ and $n_0 = 1$,
Therefore, $n^2 + 15n - 3$ is $\Omega\left(n^2\right)$
since $n^2 + 15n - 3 \geqslant n^2$ for $n \geqslant 1$

Since, both are true, $n^2 + 15n - 3$ is $\theta\left(n^2\right)$