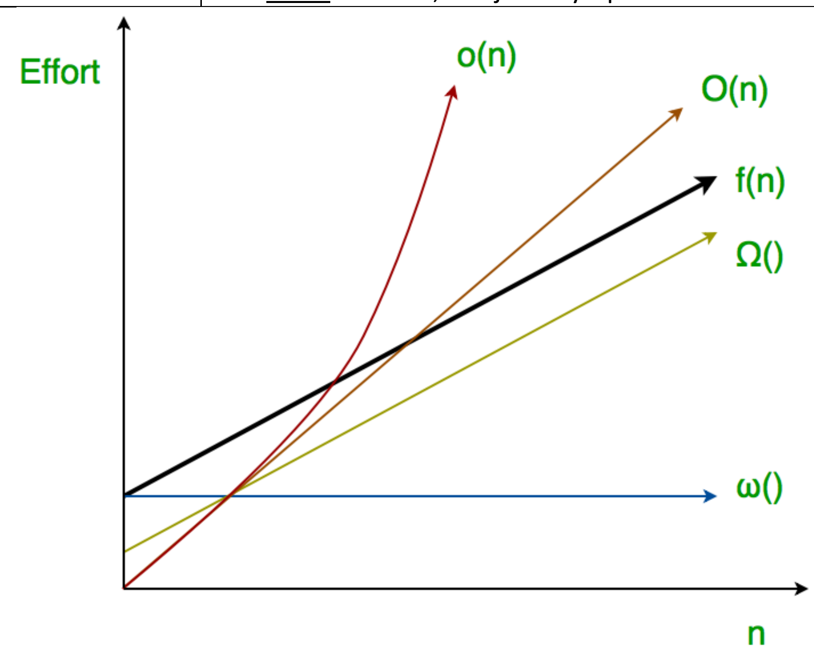
Harold's Big O
Cheat Sheet
22 September 2025

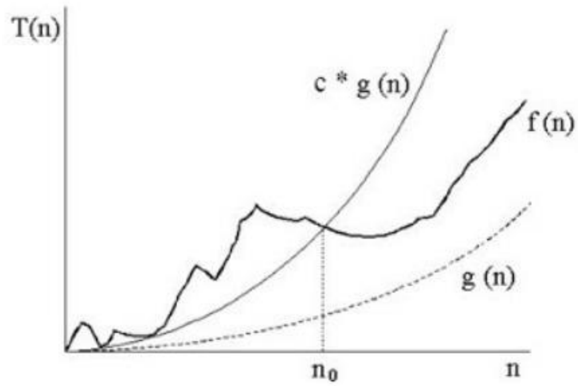
AKA Analysis of Algorithms

Asymptotic Notations

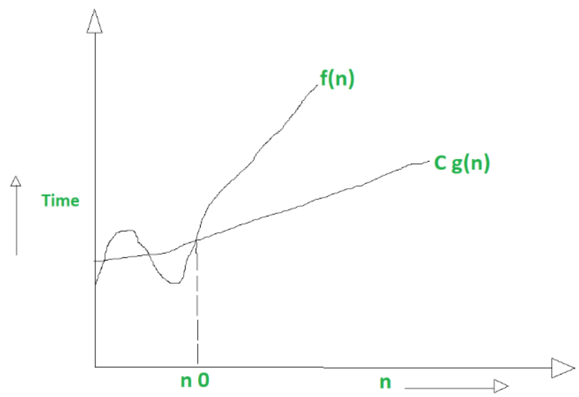
Term	Definition
Bachmann–Landau Notation	• A family of asymptotic mathematical notations that describe the limiting behavior of a function as the argument tends towards infinity. • Includes O, o, Ω, ω, and Θ. • Omits constant factors (a_n), lower-order terms, and constants (c).
Big O (O)	The tight upper bound asymptotic growth rate of $f(n)$. GOOD
Big Omega (Ω)	The tight lower bound asymptotic growth rate of $f(n)$.
Theta (Θ)	The tight bound asymptotic growth rate of $f(n)$. BETTER
Little O (o)	The loose upper bound asymptotic growth rate of $f(n)$.
Little Omega (ω)	The loose lower bound asymptotic growth rate of $f(n)$.
Closed Form	The exact solution, not just asymptotic. BEST



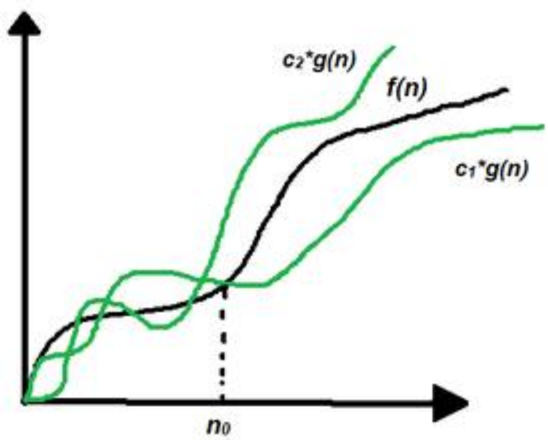
Big O (O) – Tight Upper Bound

Term	Definition	
What it Means	• The asymptotic tight <u>upper bound</u> of a function is represented by Big O notation (**O**). • Means "is of the same order as". • The rate of growth of an algorithm is $\leq$ a specific value. • $f(n)$ grows no faster than $g(n)$. • We are concerned with how f grows when n is large.	
Definition	$f(n) = \boldsymbol{O}(g(n))\ as\ n \to \infty$ If there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)\ \ for\ all\ n \geq n_0$.	
Graph	$\lvert f(n)\rvert$ is asymptotically bounded above by $g(n)$ up to a constant factor C.  	
Examples	$f(n) = 6n^4 - 2n^3 + 5 = O(n^4)$	Since $\lvert 6n^4 - 2n^3 + 5\rvert \leq 13n^4$
	$f(n) = n^{-3} + n^{-2} + n^{-1} = O(n^{-1})$	n^{-1} is the largest exponential

Big Omega (Ω) – Tight Lower Bound

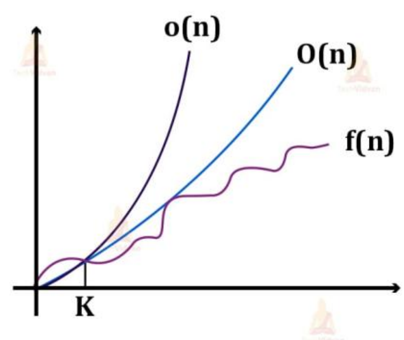
Term	Definition
What it Means	• The asymptotic tight <u>lower bound</u> of a function is represented by Big Omega notation ($\boldsymbol{\Omega}$). • The rate of growth of an algorithm is ≥ to a specific value. • Big-Omega **Ω** notation is the least used notation for the analysis of algorithms because it can make a **correct but imprecise** statement over the performance of an algorithm.
Definition	$f(n) = \boldsymbol{\Omega}(g(n))\ as\ n \to \infty$ If there exist positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)\ \ for\ all\ n \ge n_0$.
Graph	
Examples	$f(n) = sin(n) = \boldsymbol{\Omega}(1)$

Theta (Θ) – Tight Bound

<table>
<tr><th>Term</th><th colspan="2">Definition</th></tr>
<tr><td>What it Means</td><td colspan="2">The <u>exact asymptotic</u> behavior, both upper and lower, is represented by Theta notation (Θ).
The rate of growth of an algorithm is $=$ to a specific value.
Provides the average time complexity of an algorithm.</td></tr>
<tr><td>Definition</td><td colspan="2">$f(n) = \boldsymbol{\Theta}(g(n))\ as\ n \to \infty$
If there exist positive constants c_1, c_2, and n_0 such that
$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\ \ for\ all\ n \geq n_0$.</td></tr>
<tr><td>Graph</td><td colspan="2">
</td></tr>
<tr><td>Example</td><td>Linear search</td><td>Average case time complexity:
$= \frac{\sum_{i=1}^{n+1} \boldsymbol{\Theta}(i)}{n+1}$
$\Rightarrow \frac{\boldsymbol{\Theta}(n+1)\ \cdot\ \frac{(n+2)}{2}}{n+1}$
$\Rightarrow \boldsymbol{\Theta}\left(1 + \frac{n}{2}\right)$
$\Rightarrow \boldsymbol{\Theta}(n)$</td></tr>
</table>

Little O (o) – Loose Upper Bound

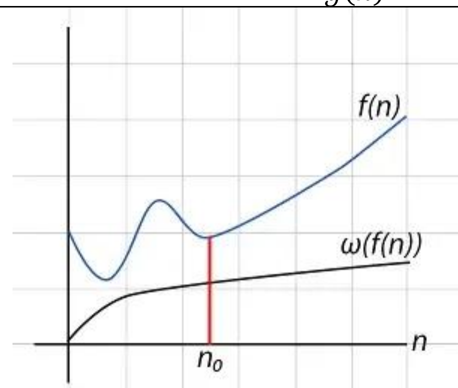
<table>
<tr><th>Term</th><th colspan="2">Definition</th></tr>
<tr><td>What it Means</td><td colspan="2">The asymptotic loose <u>upper bound</u> of a function is represented by Little O notation (o).
Means “is ultimately smaller than”.
o is a <u>rough estimate</u> of the maximum order of growth whereas O is more accurate and may be the actual order of growth.
$g(x)$ grows strictly faster than, or grows at least as fast as, $f(x)$.
Is a stronger statement than Big-O since it is not asymptotically tight.</td></tr>
<tr><td>Definition</td><td colspan="2">$f(n) \in \boldsymbol{o}\big(g(n)\big)$
If there exist positive constants c and n_0 such that
$0 \leq f(n) \leq c \cdot g(n) \;\; for\ all\ n \geq n_0$.

$f(n) \in \boldsymbol{o}\big(g(n)\big)\ if \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$</td></tr>
<tr><td>Graph</td><td colspan="2">
</td></tr>
<tr><td rowspan="2">Examples</td><td>$f(n) = \frac{1}{n} = \boldsymbol{o}(1)$</td><td>$\lim_{n\to\infty} \frac{\left(\frac{1}{n}\right)}{1} = 0$</td></tr>
<tr><td>$f(n) = 7n + 8 = \boldsymbol{o}(n^2)$</td><td>$\lim_{n\to\infty} \frac{7n+8}{n^2} = 0$</td></tr>
</table>

Little Omega (ω) – Loose Lower Bound

<table>
<tr><th>Term</th><th colspan="2">Definition</th></tr>
<tr><td>What it Means</td><td colspan="2">

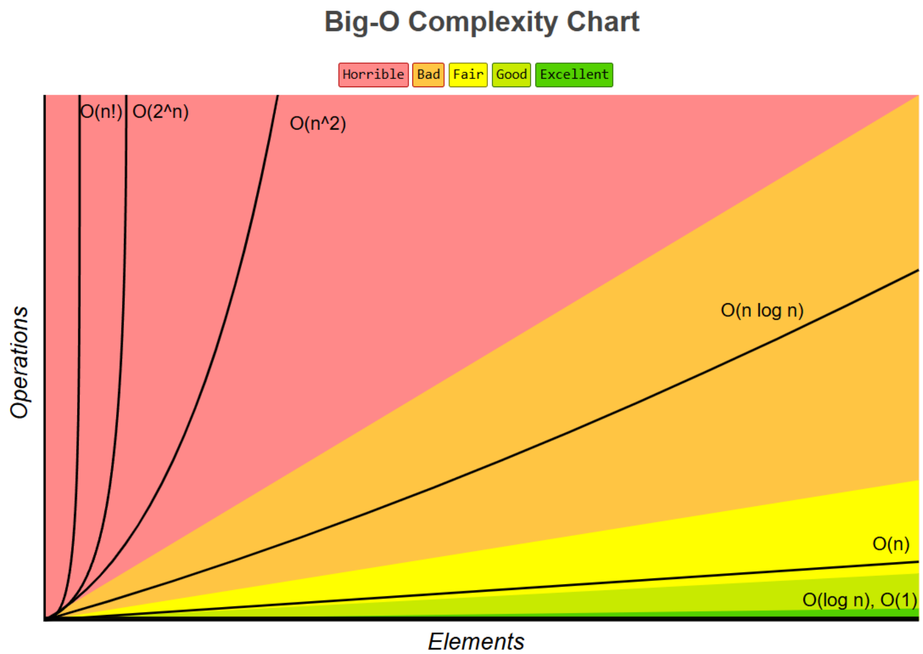
The asymptotic loose lower bound of a function is represented by Little Omega notation ($\boldsymbol{\omega}$).
Means "is ultimately larger than".
$\boldsymbol{\omega}$ is a rough estimate of the minimum order of growth whereas $\boldsymbol{\Omega}$ is more accurate and may be the actual order of growth.
$f(x)$ grows strictly faster than, or grows at least as fast as, $g(x)$.
$\boldsymbol{\omega}$ is a stronger statement than $\boldsymbol{\Omega}$ since it is not asymptotically tight.
</td></tr>
<tr><td>Definition</td><td colspan="2">$f(n) \in \boldsymbol{\omega}(g(n))$
If there exist positive constants c and n_0 such that
$0 \le c \cdot g(n) \le f(n) \;\; for\ all\ n \ge n_0.$

$f(n) \in \boldsymbol{\omega}(g(n))\ if \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$</td></tr>
<tr><td>Graph</td><td colspan="2">

f(n) = ω(g(n))</td></tr>
<tr><td rowspan="2">Examples</td><td>$f(n) = 4n + 6 = \boldsymbol{\omega}(1)$</td><td>$\lim_{n\to\infty} \frac{4n+6}{1} = \infty$</td></tr>
<tr><td>$f(n) = 6n^2 - 4n + 6 = \boldsymbol{\omega}(n)$</td><td>$\lim_{n\to\infty} \frac{6n^2 - 4n + 6}{n} = \infty$</td></tr>
</table>

Complexity

<table>
<tr><th>Term</th><th>Definition</th></tr>
<tr><td>Comparing Complexity</td><td>



</td></tr>
<tr><td>Complexity Classes</td><td>

Ordered from smallest to largest impact.

Notation	Name
$O(1)$	Constant
$O(\alpha(n))$	Inverse Ackermann function
$O(\log(\log(n)))$	Double logarithmic
$O(\log(n))$	Logarithmic
$O((\log(n))^c)$ where $c > 1$	Polylogarithmic
$O(n^c)$ where $0 < c < 1$	Fractional power
$O(n)$ where $c = 1$	Linear
$O(n \log^*(n))$	n log-star n
$O(n \log(n)) = O(\log(n!))$	Linearithmic
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(n^c)$ where $c > 1$	Polynomial or algebraic
$O(c^n)$	Exponential
$O(n!)$	Factorial

</td></tr>
<tr><td>Examples</td><td>

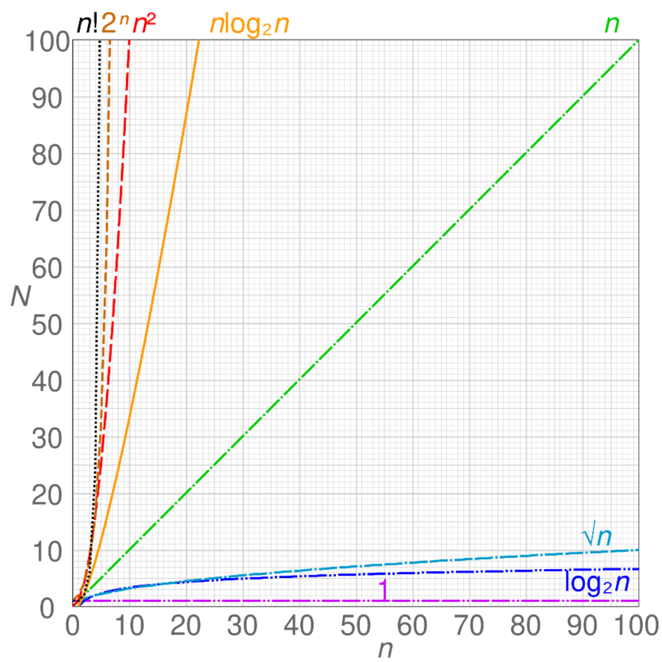
Ordered from smallest to largest.

Big O	Justification
$O(10^{100})$	It is a constant (rather large, but still a constant).
$O(\log_{10}(n))$	Logs make large numbers small.
$O(ln(n))$	$ln(x)$ is larger than $\log_{10}(n)$, but is still a log so the same as above.
$O(n^3 + n^2)$	Polynomials can be large.

</td></tr>
</table>

	$O(n^3)$	Same as $O(n^3 + n^2)$ since Big-O only cares about the largest polynomial degree.
	$O(n^{100})$	Similar to $O(n^3)$, but is much larger.
	$O(1.1^n)$	Exponentials are larger than polynomials.
	$O(3^n)$	Similar to $O(1.1^n)$ but is larger.
	$O(n2^n)$	Larger than the exponential $O(3^n)$ since multiplied by n.
	$O(n!)$	Factorials grow fastest of all.

Computer Science Application

Term	Definition
Usage	Analysis of algorithms.
Asymptotic Growth Rates	Used to analyze and classify algorithms according to how their run time or space requirements grow as the input size grows.
	
Master Theorem	Provides an asymptotic analysis for many recurrence relations that occur in the analysis of divide-and-conquer algorithms.
General Recurrence Relation Form	$\boldsymbol{T(n) = a\,T\left(\frac{n}{b}\right) + f(n)}$ n: Input size $T(n)$: Total time for the algorithm a: Number of subproblems b: Factor by which the subproblem size is reduced in each recursive call $(b > 1)$ $f(n)$: Amount of time taken at the top level of the recurrence

Define c_{crit}	$c_{crit} = log_b\, a = \frac{log(\#\ of\ subproblems)}{log(relative\ subproblem\ size)}$

Master Theorem Cases

Case	Description	Condition on $f(n)$ in relation to c_{crit}, i.e., $\log_b a$	Master Theorem bound	Notational examples
1	Work to split / recombine a problem is dominated by subproblems. i.e., the recursion tree is **leaf-heavy**.	When $f(n) = \boldsymbol{O}(n^c)$ where $c < c_{crit}$ (upper-bounded by a lesser-exponent polynomial)	... then $T(n) = \boldsymbol{\Theta}(n^{c_{crit}})$ (The splitting term does not appear; the recursive tree structure dominates.)	If $b = a^2$ and $f(n) = \boldsymbol{O}(n^{\frac{1}{2}-\epsilon})$, then $T(n) = \boldsymbol{\Theta}(n^{\frac{1}{2}})$.
2	Work to split / recombine a problem is comparable to subproblems.	When $f(n) = \boldsymbol{\Theta}(n^{c_{crit}}\, (\log n)^{\,k})$ for a $k \geq 0$ (rangebound by the critical-exponent polynomial, times zero or more optional logs)	... then $T(n) = \boldsymbol{\Theta}(n^{c_{crit}}\, (\log n)^{\,k+1})$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(n) = \boldsymbol{O}(n^{\frac{1}{2}})$, then $T(n) = \boldsymbol{\Theta}(n^{\frac{1}{2}} \log n)$. If $b = a^2$ and $f(n) = \boldsymbol{O}(n^{\frac{1}{2}} \log n)$, then $T(n) = \boldsymbol{\Theta}(n^{\frac{1}{2}}\, (\log n)^2)$.
3	Work to split / recombine a problem dominates subproblems. i.e., the recursion tree is **root-heavy**.	When $f(n) = \boldsymbol{\Omega}(n^c)$ where $c > c_{crit}$ (lower-bounded by a greater-exponent polynomial)	... this doesn't necessarily yield anything. Furthermore, if $af\left(\frac{n}{b}\right) \leq kf(n)$ for some constant $k < 1$ and all sufficiently large n (often called the *regularity condition*) then the total is dominated by the splitting term $f(n)$: $T(n) = \boldsymbol{\Theta}(f(n))$	If $b = a^2$ and $f(n) = \boldsymbol{O}(n^{\frac{1}{2}+\epsilon})$, and the regularity condition holds, then $T(n) = \boldsymbol{\Theta}(f(n))$.

Generating Functions	• $T(n)$ represents time, or the number of steps it takes, to complete a problem of size n. • Assume $T(1) = 1$. • $\boldsymbol{\Theta}(f(n)) \approx$ exact solution.

Examples

Recursive Form	Closed Form Exact Solution
$T(n) = 4T\left(\frac{n}{2}\right) + n$	$T(n) = 2n^2 - n$
$T(n) = 2T\left(\frac{n}{2}\right) + 10n$	$T(n) = n + 10n\log_2 n$
$T(n) = 2T\left(\frac{n}{2}\right) + n^2$	$T(n) = 2n^2 - n$
$T(n) = 4T\left(\frac{n}{2}\right) + n^2$	$T(n) = n^2 \cdot log_2(n) + n^2 + n - 2$
$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$	$T(n) = 1001n^3 - 1000n^2$
$T(n) = 4T\left(\frac{n}{2}\right) + n^2\, log_2(n)$	$T(n) = \frac{1}{2}n^2 \cdot (log_2(n))^2 + \frac{1}{2}n^2 \cdot log_2(n) + n^2$

Closed Form Tool

Use my Big O spreadsheet to iteratively help you find the exact closed-form solution from a recursive generating function $T(n)$.

[Harolds_Big_O_Calculator.xlsx]

$$T(n) = \boldsymbol{A}n! + \boldsymbol{B}3^n + \boldsymbol{C}2^n + \boldsymbol{D}n^3 + \boldsymbol{E}(n\log_2(n))^2 + \boldsymbol{F}n^2\log_2(n) + \boldsymbol{G}n^2\log_2(\log_2(n)) + \boldsymbol{H}n^2 + \boldsymbol{I}(n\log_2(n)) + \boldsymbol{J}(n\log_2(\log_2(n))) + \boldsymbol{K}(\log_2(n))^2 + \boldsymbol{L}n + \boldsymbol{M}\sqrt[2]{n} + \boldsymbol{N}\sqrt[3]{n} + \boldsymbol{O}\log_2(n) + \boldsymbol{P}1$$

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Mathematics Application

Term	Definition
Usage	Is commonly used to describe how closely a finite series approximates a given function, especially in the case of a truncated Taylor series.
Taylor Series	$f(x) = P_n(x) + R_n(x)$ $P_n(x) = \sum_{n=0}^{+\infty} \frac{f^{(n)}(c)}{n!} (x-c)^n$ $R_n(x) = \frac{f^{(n+1)}(x^*)}{(n+1)!} (x-c)^{n+1}$ where $x \leq x^* \leq c$ and $\lim_{x \to +\infty} R_n(x) = 0$ $R_n(x) = \boldsymbol{O}(f(x))$
Maclaurin Series	Taylor Series centered about $x = 0$. $f(x) \approx P_n(x) = \sum_{n=0}^{+\infty} \frac{f^{(n)}(0)}{n!} x^n$
Example	$f(x) = e^x$ $f(x) = P_8(x) + R_8(x)$ $f(x) \approx P_8(x)$ $R_8(x) = P_8(x)'s\ Error\ Upper\ Bound$ $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ for all $x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \frac{x^8}{8!} + \cdots$ $P_8(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \frac{x^8}{8!}$ $R_8(\max x^*\ in\ range) = \frac{(x^*)^9}{9!}$ $\boldsymbol{R_8(x) = O(x^9)}$

Sources

- Dev (2025), Asymptotic Notations: A Comprehensive Guide. https://dev.to/princem/asymptotic-notations-a-comprehensive-guide-30i8
- Geeks for Geeks (20 Mar 2015).
    - Big O vs Theta Θ vs Big Omega Ω Notations. https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/
    - Analysis of algorithms | little o and little omega notations. https://www.geeksforgeeks.org/analysis-of-algorithems-little-o-and-little-omega-notations/
- Rowell, Eric (2025). The Big-O Algorithm Complexity Cheat Sheet. https://www.bigocheatsheet.com/
- Wikipedia (2025).
    - Big O notation. https://en.wikipedia.org/wiki/Big_O_notation
    - Master theorem (analysis of algorithms). https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)