**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 (), lower-order terms, and constants (). |
| **Big O (O)** | The tight upper bound asymptotic growth rate of . GOOD |
| **Big Omega (Ω)** | The tight lower bound asymptotic growth rate of . |
| **Theta (Θ)** | The tight bound asymptotic growth rate of . BETTER |
| **Little O (o)** | The loose upper bound asymptotic growth rate of . |
| **Little Omega ()** | The loose lower bound asymptotic growth rate of . |
| **Closed Form** | The exact solution, not just asymptotic. BEST |
| Lightbox | |

**Big O (O) – Tight Upper Bound**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **What it Means** | * The asymptotic tight upper bound 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 a specific value. * grows no faster than . * We are concerned with how grows when is large. | |
| **Definition** | If there exist positive constants and such that | |
| **Graph** | is asymptotically bounded above by up to a constant factor .  A graph of a function  AI-generated content may be incorrect. | |
| **Examples** |  | Since |
|  | is the largest exponential |

**Big Omega (Ω) – Tight Lower Bound**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **What it Means** | * The asymptotic tight lower bound of a function is represented by Big Omega notation (). * 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** | If there exist positive constants and such that | |
| **Graph** | Lightbox | |
| **Examples** |  |  |

**Theta (Θ) – Tight Bound**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **What it Means** | * The exact asymptotic 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. | |
| **Definition** | If there exist positive constants and such that | |
| **Graph** | Lightbox | |
| **Example** | Linear search | Average case time complexity: |

**Little O (o) – Loose Upper Bound**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **What it Means** | * The asymptotic loose upper bound of a function is represented by Little O notation (**o**). * Means “is ultimately smaller than”. * **o** is a rough estimate of the maximum order of growth whereas **O** is more accurate and may be the actual order of growth. * grows strictly faster than, or grows at least as fast as, . * Is a stronger statement than Big-O since it is not asymptotically tight. | |
| **Definition** | If there exist positive constants and such that | |
| **Graph** | Data Structures Asymptotic Analysis - TechVidvan | |
| **Examples** |  |  |
|  |  |

**Little Omega () – Loose Lower Bound**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **What it Means** | * The asymptotic loose lower bound of a function is represented by Little Omega notation (). * Means “is ultimately larger than”. * is a rough estimate of the minimum order of growth whereas is more accurate and may be the actual order of growth. * grows strictly faster than, or grows at least as fast as, . * is a stronger statement than since it is not asymptotically tight. | |
| **Definition** | If there exist positive constants and such that | |
| **Graph** |  | |
| **Examples** |  |  |
|  |  |

**Complexity**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | |
| **Comparing Complexity** | A diagram of a complexity  AI-generated content may be incorrect. | |
| **Complexity Classes** | Ordered from smallest to largest impact. | |
| **Notation** | **Name** |
|  | Constant |
|  | Inverse Ackermann function |
|  | Double logarithmic |
|  | Logarithmic |
| where | Polylogarithmic |
| where | Fractional power |
| where | Linear |
|  | n log-star n |
|  | Linearithmic |
|  | Quadratic |
|  | Cubic |
| where | Polynomial or algebraic |
|  | Exponential |
|  | Factorial |
| **Examples** | Ordered from smallest to largest. | |
| **Big O** | **Justification** |
|  | It is a constant (rather large, but still a constant). |
|  | Logs make large numbers small. |
|  | is larger than , but is still a log so the same as above. |
|  | Polynomials can be large. |
|  | Same as since Big-O only cares about the largest polynomial degree. |
|  | Similar to , but is much larger. |
|  | Exponentials are larger than polynomials. |
|  | Similar to but is larger. |
|  | Larger than the exponential since multiplied by 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. |
| undefined | |
| **Master Theorem** | Provides an asymptotic analysis for many recurrence relations that occur in the analysis of divide-and-conquer algorithms. |
| **General Recurrence Relation Form** | : Input size  : Total time for the algorithm  : Number of subproblems  : Factor by which the subproblem size is reduced in each recursive call  :f(n) : Amount of time taken at the top level of the recurrence |
| **Define** |  |
| **Master Theorem Cases**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Case** | **Description** | **Condition on  in relation to ,**  **ccriti.e., logb⁡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)=O(nc) where  c<ccrit(upper-bounded by a lesser-exponent polynomial) | ... then    (The splitting term does not appear; the recursive tree structure dominates.) | If and f(n)=O(n1/2−ϵ), then  T(n)=Θ(n1/2). | | **2** | Work to split / recombine a problem is comparable to subproblems. | When f(n)=Θ(nccrit(log⁡n)k)for a  (rangebound by the critical-exponent polynomial, times zero or more optional loglogs) | ... then  (The bound is the splitting term, where the log is augmented by a single power.) | If and f(n)=O(n1/2−ϵ), then  T(n)=Θ(n1/2).  If and f(n)=O(n1/2−ϵ), then  T(n)=Θ(n1/2). | | **3** | Work to split / recombine a problem dominates subproblems.  i.e., the recursion tree is **root-heavy**. | When f(n)=Ω(nc)where c>ccrit  (lower-bounded by a greater-exponent polynomial) | ... this doesn't necessarily yield anything.  Furthermore, if for some constant and all sufficiently large  (often called the *regularity condition*)  then the total is dominated by the splitting term : T(n)=Θ(f(n)) | If and f(n)=O(n1/2−ϵ),  and the regularity condition holds, then T(n)=Θ(f(n))  T(n)=Θ(n1/2). | | |

|  |  |  |
| --- | --- | --- |
| **Generating Functions** | * represents time, or the number of steps it takes, to complete a problem of size . * Assume . * exact solution. | |
| **Examples** | **Recursive Form** | **Closed Form Exact Solution** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Closed Form Tool** | Use my Big O spreadsheet to iteratively help you find the exact closed-form solution from a recursive generating function . | |
| [Harolds\_Big\_O\_Calculator.xlsx](https://www.toomey.org/tutor/harolds_cheat_sheets/Harolds_Big_O_Calculator.xlsx) | |
|  | |
| **A screenshot of a computer  AI-generated content may be incorrect.** | | |
| **A screen shot of a computer program  AI-generated content may be incorrect.** | | |

**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** | where and |
| **Maclaurin Series** | Taylor Series centered about |
| **Example** |  |

**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](https://x.com/ericdrowell) (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)>