# **Big O notation cheat sheet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Notation** | **Description** | **Example code** | **Example use** |
|  | O(1) | **Constant**. An algorithm that always executes in the same time regardless of the size of the data set. Efficient with any data set. | random\_num = data\_set(x) | Extracting data from any element from an array.  Hashing algorithm. |
|  | O(log N) | **Logarithmic**. An algorithm that halves the data set in each pass. Opposite to exponential. Efficient with large data sets. | While Found = False And LowerBound <= UpperBound  MidPoint = LowerBound + (UpperBound - LowerBound) \ 2  If data\_set (MidPoint) = searchedFor Then  Found = True  ElseIf data\_set (MidPoint) < searchedFor Then  LowerBound = MidPoint + 1  Else  UpperBound = MidPoint - 1  End If  End While | Binary search. |
|  | O(N) | **Linear**. An algorithm whose performance declines as the data set grows. Reduces efficiency with increasingly large data sets. | For x = 1 To y  data\_set(x) = counter Next | A loop iterating through a single dimension array.  Linear search. |
|  | O(n log N) | **Linearithmic**. Algorithms that divide a data set but can be solved using concurrency on independent divided lists. |  | Quick sort.  Merge sort. |
|  | O(N2) | **Polynomial**. An algorithm whose performance is proportional to the square of the size of the data set. Significantly reduces efficiency with increasingly large data sets. Deeper nested iterations result in O(N3), O(N4) etc. depending on the number of dimensions. | For x = 1 To w  For y = 1 To z  data\_set(x, y) = 0  Next  Next | A nested loop iterating through a two dimension array.  Bubble sort. |
|  | O(2N) | **Exponential**. An algorithm that doubles with each addition to the data set in each pass. Opposite to logarithmic. Inefficient. | Function fib(x)  If x <= 1 Then Return x  Return fib(x - 2) + fib(x - 1)  End Function | Recursive functions with two calls. Fibonacci number calculation with recursion. |

# **Big O notation cheat sheet**

Big O notation also known as Landau's symbol is used to describe the complexity of an algorithm. As the data set on which an algorithm executes grows, so too can the number of cycles of processing time and the memory space requirements. This is known as scalability. Big O notation describes this effect, considering best, worst and average case situations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Searching algorithms** | **Time complexity** | | |
| **Best** | **Average** | **Worst** |
| Linear search | O(1) | O(n) | O(n) |
| Binary search array | O(1) | O(log n) | O(log n) |
| Binary search tree | O(1) | O(log n) | O(n) |
| Hashing | O(1) | O(1) | O(n) |
| Breadth/Depth first search | O(1) | O(V+E) No. vertices + No. edges | O(V2) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sorting algorithms** | **Time complexity** | | | **Space complexity** |
| **Best** | **Average** | **Worst** |
| Bubble sort | O(n) | O(n2) | O(n2) | O(1) |
| Insertion sort | O(n) | O(n2) | O(n2) | O(1) |
| Merge sort | O(n log n) | O(n log n) | O(n log n) | O(n) |
| Quick sort | O(n log n) | O(n log n) | O(n2) | O(log n) |