Big O notation cheat sheet

Big O notation is used to describe the complexity of an algorithm in terms of how well it scales. As a data set grows, so too can the number of cycles of processing time and memory space requirements – this is known as scalability. Big O notation describes this effect, considering best-, worst- and average-case scenarios.

	Notation	Description	Example code	Example use
	O(1)	**Constant**. An algorithm that always executes in the same amount of 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.

	$O(N^2)$	**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 $O(N^3)$, $O(N^4)$, 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-dimensional array. Bubble sort.

	$O(2^N)$	**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

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 of graph	O(1)	O(V+E) No. vertices + No. edges	$O(V^2)$

Sorting algorithms	Time complexity			Space complexity
	Best	Average	Worst	
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	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)	$O(n^2)$	O(log n)