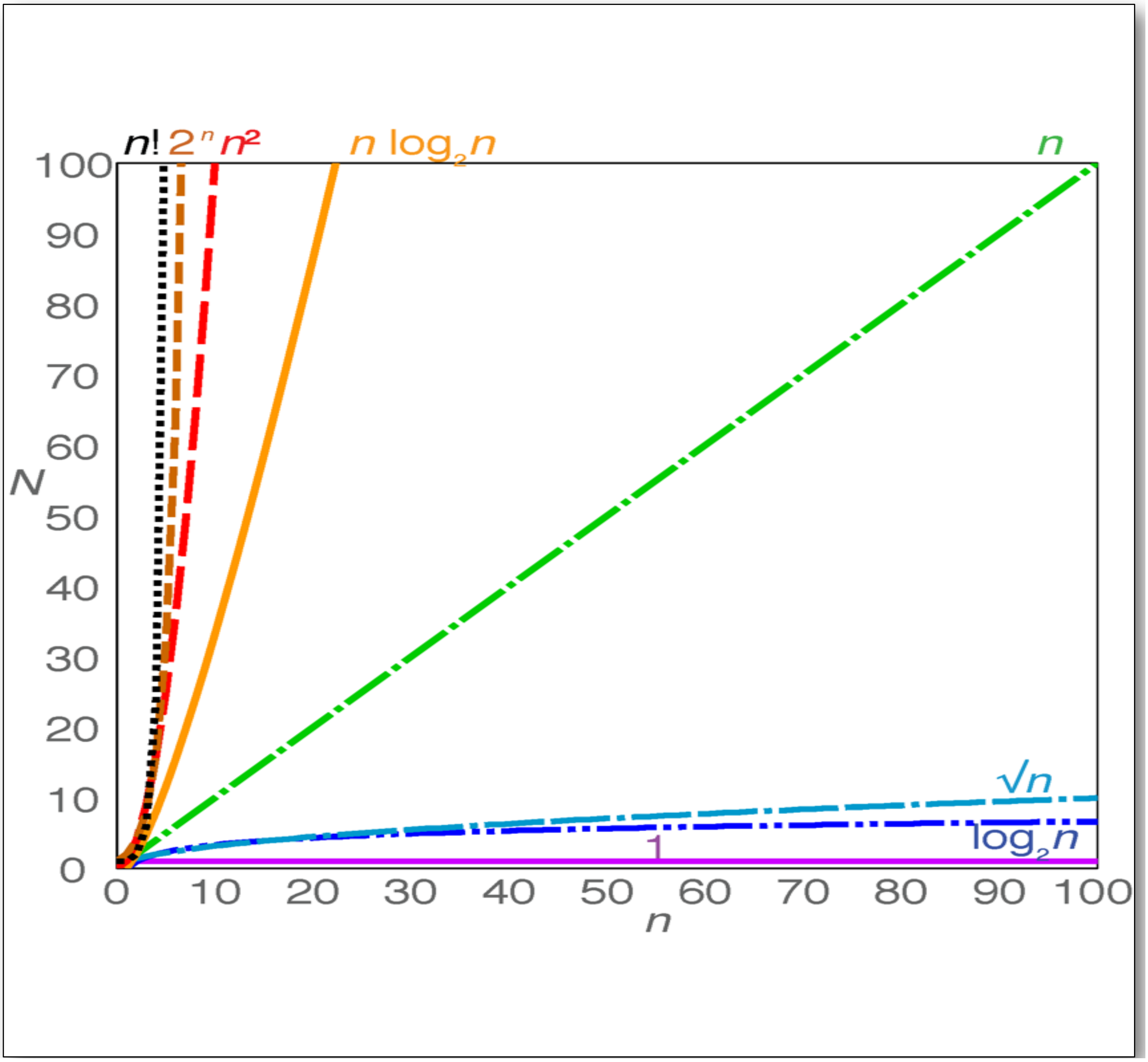
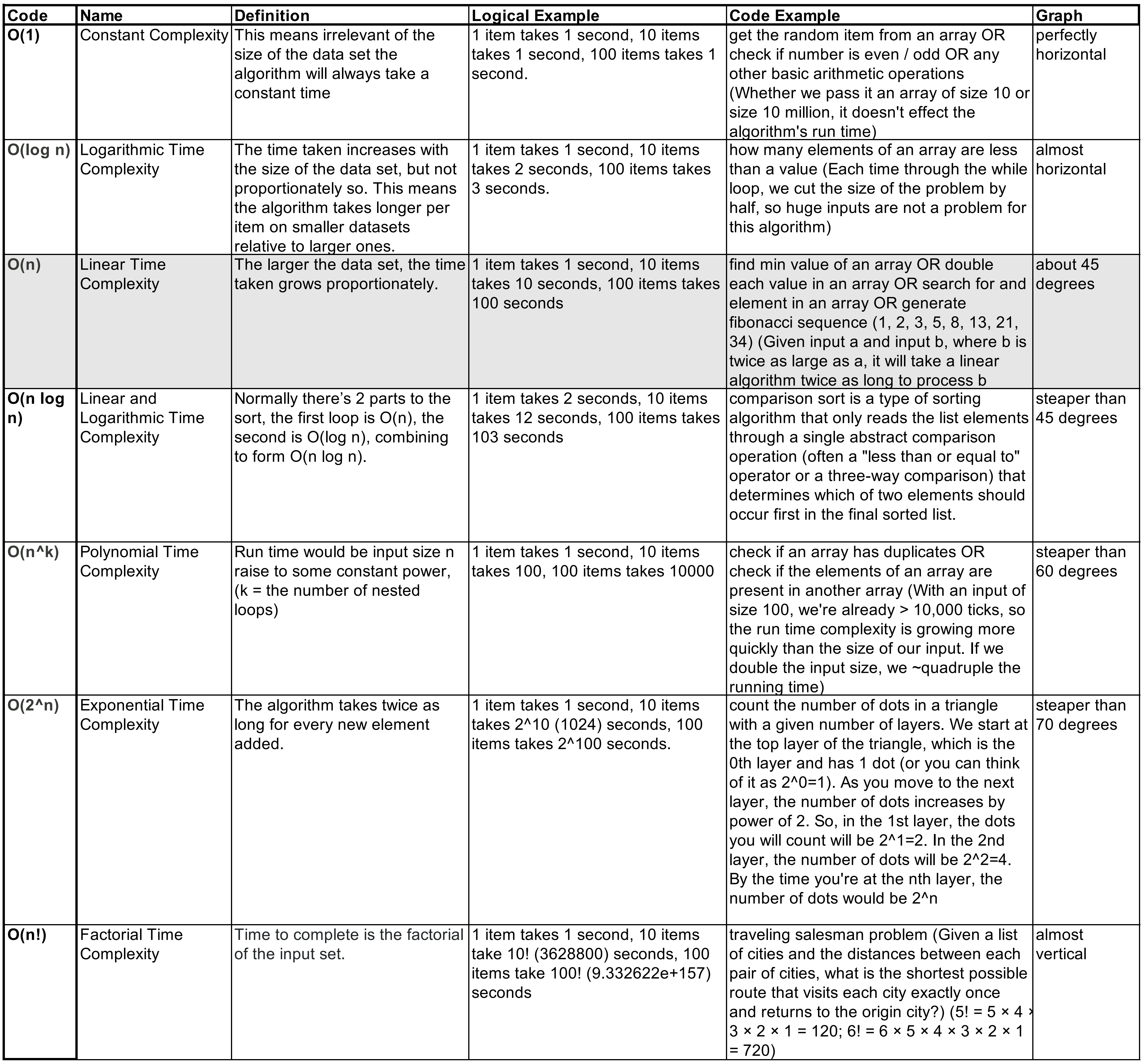
Big-O Notations for Interviews





Code	Name	Definition	Logical Example	Code Example	Graph
O(1)	Constant Complexity	This means irrelevant of the size of the data set the algorithm will always take a constant time	1 item takes 1 second, 10 items takes 1 second, 100 items takes 1 second.	get the random item from an array OR check if number is even / odd OR any other basic arithmetic operations (Whether we pass it an array of size 10 or size 10 million, it doesn't effect the algorithm's run time)	perfectly horizontal
O(log n)	Logarithmic Time Complexity	The time taken increases with the size of the data set, but not proportionately so. This means the algorithm takes longer per item on smaller datasets relative to larger ones.	1 item takes 1 second, 10 items takes 2 seconds, 100 items takes 3 seconds.	how many elements of an array are less than a value (Each time through the while loop, we cut the size of the problem by half, so huge inputs are not a problem for this algorithm)	almost horizontal
O(n)	Linear Time Complexity	The larger the data set, the time taken grows proportionately.	1 item takes 1 second, 10 items takes 10 seconds, 100 items takes 100 seconds	find min value of an array OR double each value in an array OR search for and element in an array OR generate fibonacci sequence (1, 2, 3, 5, 8, 13, 21, 34) (Given input a and input b, where b is twice as large as a, it will take a linear algorithm twice as long to process b	about 45 degrees
O(n log n)	Linear and Logarithmic Time Complexity	Normally there’s 2 parts to the sort, the first loop is O(n), the second is O(log n), combining to form O(n log n).	1 item takes 2 seconds, 10 items takes 12 seconds, 100 items takes 103 seconds	comparison sort is a type of sorting algorithm that only reads the list elements through a single abstract comparison operation (often a "less than or equal to" operator or a three-way comparison) that determines which of two elements should occur first in the final sorted list.	steaper than 45 degrees
O(n^k)	Polynomial Time Complexity	Run time would be input size n raise to some constant power, (k = the number of nested loops)	1 item takes 1 second, 10 items takes 100, 100 items takes 10000	check if an array has duplicates OR check if the elements of an array are present in another array (With an input of size 100, we're already > 10,000 ticks, so the run time complexity is growing more quickly than the size of our input. If we double the input size, we ~quadruple the running time)	steaper than 60 degrees
O(2^n)	Exponential Time Complexity	The algorithm takes twice as long for every new element added.	1 item takes 1 second, 10 items takes 2^10 (1024) seconds, 100 items takes 2^100 seconds.	count the number of dots in a triangle with a given number of layers. We start at the top layer of the triangle, which is the 0th layer and has 1 dot (or you can think of it as 2^0=1). As you move to the next layer, the number of dots increases by power of 2. So, in the 1st layer, the dots you will count will be 2^1=2. In the 2nd layer, the number of dots will be 2^2=4. By the time you're at the nth layer, the number of dots would be 2^n	steaper than 70 degrees
O(n!)	Factorial Time Complexity	Time to complete is the factorial of the input set.	1 item takes 1 second, 10 items take 10! (3628800) seconds, 100 items take 100! (9.332622e+157) seconds	traveling salesman problem (Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?) (5! = 5 × 4 × 3 × 2 × 1 = 120; 6! = 6 × 5 × 4 × 3 × 2 × 1 = 720)	almost vertical

https://en.wikipedia.org/wiki/Time_complexity