Big-O Notation

Review

What is time complexity?

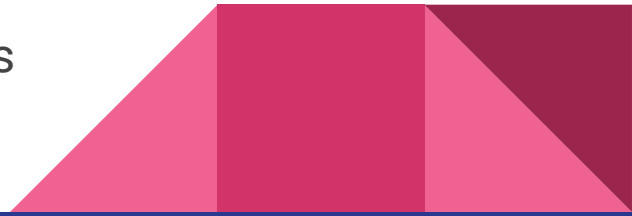
It is the number of steps involved to run an algorithm.

How do we measure the time complexity of an algorithm?

Looking at:

1. The worst case?
2. The best case?
3. The average case?

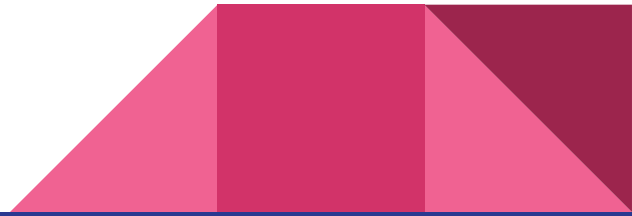
A: The worst case. Looking at the worst case can help us to understand what to expect from the algorithm.



Constant Time: O(1)

An algorithm does not depend on the input size n. The run time will always be the same regardless of the input size.

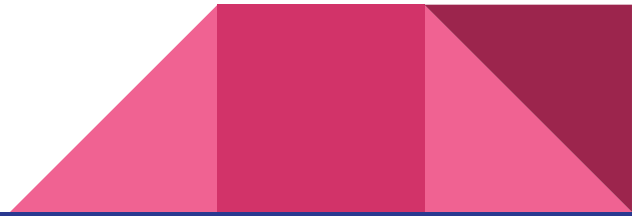
Example: Return the first element of an array. Even if the array has 1 million elements, the time complexity will be constant.



Linear Time: O(n)

When the running time of an algorithm increases linearly with the size of the input (a function iterates over the input size of n).

Example: An algorithm that calculates the factorial of any inputted number. If you input 5 then you are to loop through and multiply 1 by 2 by 3 by 4 and by 5 and then output 120.



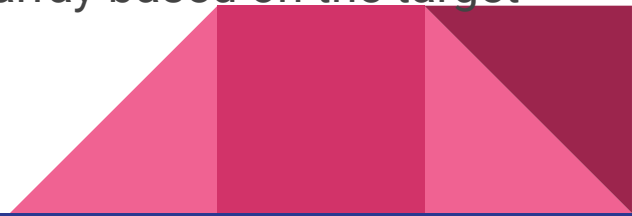
Logarithm Time O(log n)

Similar to linear time complexity, except that the runtime does not depend on the input size but rather on half the input size. The input size decreases on each iteration or step.

This method is the second best because an algorithm runs for half the input size rather than the full size.

The input size decreases with each iteration.

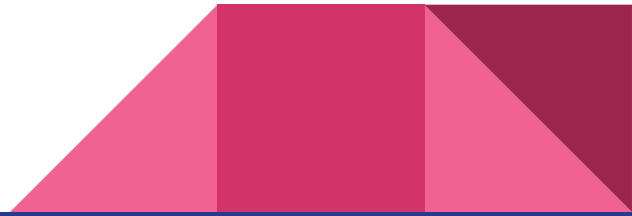
Example: Binary search functions, which divide a sorted array based on the target value.



Quadratic Time O(n^2)

When an algorithm has nested iteration (nested loops).

Example: An outer loop runs n times, and the inner loop will run n times for each iteration of the outer loop, which will give total n^2 prints. If the array has ten items, ten will print 100 times (10^2).

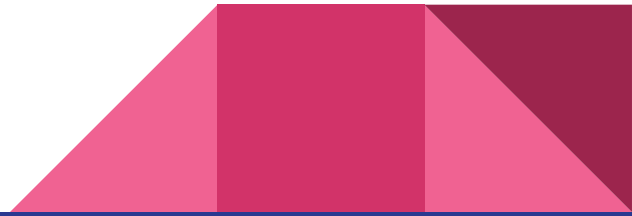


Exponential Time O(2^n)

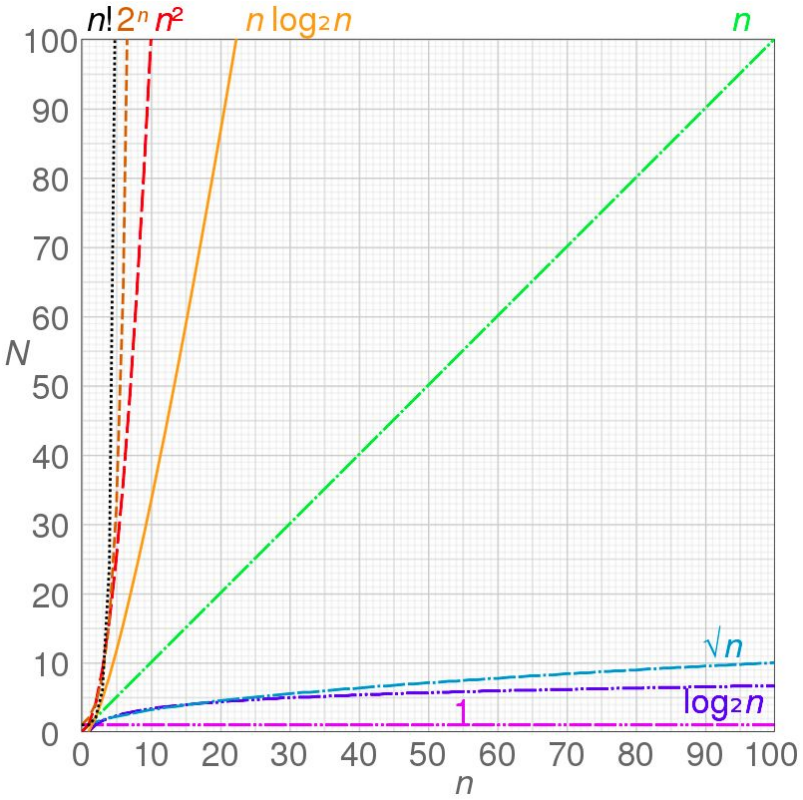
When the growth rate doubles with each addition to the input (n), often iterating through all subsets of the input elements.

Example: The recursive Fibonacci sequence. The Fibonacci sequence is a mathematical sequence in which each number is the sum of the two preceding numbers, where 0 and 1 are the first two numbers. The third number in the sequence is 1, the fourth is 2, the fifth is 3, and so on.

If you pass in 6, then the 6th element in the Fibonacci sequence would be 8:



Big O Notation



Graphs of functions commonly used in the analysis of algorithms, showing the number of operations *N* versus input size *n* for each function.

Source: Wikipedia

Big 0 values

Big 0 values		
O(1)	Constant	Excellent
O(logn)	Logarithmic (base2)	Good
O(n)	Linear	Fair
O(nlogn)	n logn	Bad
O(n^2)	Quadratic	Horrible