Data Structures

Organises and stores data

Each has its own strengths and weaknesses

The best data structures depend on :	The type of data you need to store
	How your application needs access to the data
	The operations it will perform the most on the data

Algorithms

Steps performed to accomplish the specified task

Big O Notation

Time complexity is the steps taken to run an algorithm

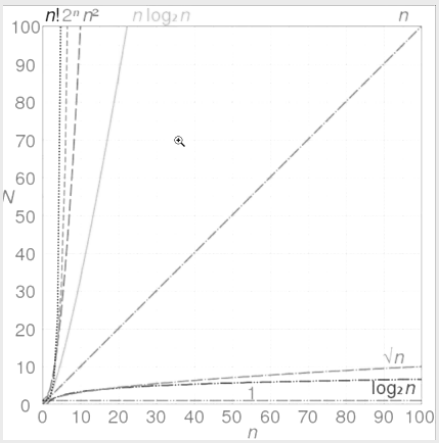
How well an algorithm scales to the number of items that it must deal with

Always look at the worst case scenario

Summary:

O(1)	Constant
O(logn)	Logarithmic
O(n)	Linear
O(nlogn)	n log-star n
O(n²)	Quadratic

Big O Graph



Constant Time Complexity

The number of items have no effect on the number of steps.

Number of steps is always going to be constant

This has a constant time complexity of O(1)

As the number of items inscreases, algorithm doesn't degrade at all.

Retrieving with an index is a constant time O(1)

Linear time complexity

Time complexity increases as n increases

This increase is *linear*

Worst case requires going through the entire array

Fixed size array, not resizable (not dynamic)

>Adding a new element to an array requires a new array big enough for the new element

>Then copy the old elements into it with the new integer

This is also a linear time complexity as creating an array doesn't depend on elements, and adding a new one doesn't depend on it,

> but copying it requires looping over the entire array.

If the array had a space and we knew the index, it would be O(1), because it is similar to retrieving an element.

In conclusion; With a loop, it's O(n), without a loop, its O(1)

Retrieving without an index is Linear time O(n)

Arrays

Arrays (cont)

if you create an array of strings, what you're actually storing in the array is a bunch of object references to the string instances

those object references are all gonna be the same size

That's why you can have an object array and store any type of object in there. It's because the object references to the different instances are always the same size.

Calculating Memory Address Based on Index

If an array starts at memory address x	x
size of each element in the array is y	
calculaing the memory address of element i by using the following expression:	x + i * y

Stored as one contiguous block in memory.
Stored as one block with a static length, not spread out
Each element occupies the same amount of space in memory.
Every value of an int array occupies 4bytes in memory, not differing between elements.
you create an array of objects, what's stored in the array elements is a reference to those objects,
object references are always the same size regardless of the type of object they're referring to.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com