ArrayList

Let's shift into the second half our course: data structures. We're going to start with how arrays are handled. Up to this point we've just been talking about how to sort items in a list but how are the things actually stored in a list? Well, as you guessed, it depends. You'll choose which one you need contextually.

Big O applies here. We're going to be talking about two ways to implement lists and you'd choose depending on what sort of workload you needed.

This is actually a bit of a moot point for JavaScript developers: we have normal arrays and no choice beyond anything in the matter. In other languages, however, there are multiple types of array and you choose which one you need based on the sort operations you intend on doing on that array. Nonetheless I think it's a useful exercise to think beyond just the algorithms you're running but also the cost of the underlying data structures.

ArrayList

Let's pretend for a moment that JavaScript has no array type. No more `const x = []`. We only have one thing: objects. So we'd need to implement the array numbering ourselves. But not just that, we'd have to implment adding numbers, removing numbers, getting numbers, etc. It's a lot of work!

I'm borrowing the Java terms for these, by the way.

For array list, this works more or less how we as humans tend think about it: in memory we'll lay out everything sequentially in memory. We'd say that the array starts *somewhere* and then if we ask for the 2 index, we'd go to the beginning of the array and move over 3 to the 2 index. Since everything is already laid out in the order it's in memory, it makes look-ups really simple. Just by knowing where the start of the array is and the index, we know where the thing we're looking for in memory. This would be a O(1) in terms of complexity. No matter how big the ArrayList is, array lookups take the same amount of time.

Now imagine our list is 15,000 items long and we delete the 1 index. We now have to shift 14,998 items over in memory. This is called compacting and it's painful for ArrayList. Same applies for inserts.

```
[a,b,c,d,e,f,g]
-> delete index 3
-> array is [a,b,c,(blank),e,f,g]
-> shift elements 4,5,6 back one index
-> array is [a,b,c,e,f,g]
-> decrement length
```

Keep in mind you have to do an look-up and a deletion in order to do a deletion. This true regardless. We're measuring these actions independently.

So when are you going to choose an ArrayList? When you need to do lots of lookups! It's the best for that. You're essentially prematurely optimizing for lookups.