So now we're going to talk about the big O of arrays.

And it's important to understand this.

So we can compare arrays to the other data structures that we're going to build.

So I'm going to start with an array.

But I'm going to present it like this.

So we have the indexes there because this will be important for explaining the big O of arrays.

So we're going to start with an array that we've already created that we call my array.

And then we're going to push an item onto the end.

So my array dot push is going to push this number 17 onto the end with that index of four.

The important thing here though is that we do not have to re-index this or this or this all the way down at the beginning.

Likewise, when we remove this item with a pop and we pop that off.

We don't have to re-index anything.

And because of that, push and pop are both o of one operations.

But it's different on the other end of the array.

So let's bring up my array.

And we're going to shift that first item off of the array when we remove this.

The problem we have is that this index is incorrect.

The first item in the array should be zero.

So this has to be re indexed to zero.

And the next item has to be re-indexed.

And the next item.

Likewise, if we do an unshift and we bring that 11 back, we have to re-index everything in the array

all the way down just to be able to bring this back.

Not a big deal when we have a small array like this, but if we had one that had thousands of items,

that's a lot of reindexing.

So this is O of n, where n is the number of items in the array.

Very important with arrays to understand that on this end.

Adding and removing are both o of one.

On the other end.

Removing because of the reindexing.

And bringing it back because of the Reindexing are both o of n.

Now let's look at adding something into the middle of this array.

So we'll say my array dot splice.

And if you haven't seen this before what we're saying with this is we're going to put something in at

the index of one.

This zero means we're not going to remove any items.

And this means we're going to add an item with a string that says hi.

So let's open up that index of one, drop that new item with that string in there at the index of one.

And when we do.

This index is incorrect.

We have to re-index this and this and this all the way down.

Since we have to re index all of this.

This is O of N and you might say, hey, we're putting that in the middle.

Wouldn't that be o of one half n.

Well, there's two problems with that logic.

First is that.

Bingo.

Always measures worst case, not average case.

The other problem is this even if this was a one half n operation, one half is a constant.

And we drop constants.

So either way this is O of n.

So likewise when we go to remove this item we're going to have to re-index everything all the way down.

So it doesn't matter if you're adding or removing somewhere in the middle of the array, that's going

to be O of n.

Now let's look at finding an item.

Let's say we're going to find the number seven in this array.

To do that we'd have to start at the beginning and say is that seven.

No.

No.

And then we finally get to seven and it's.

Yes.

That means that if you search by value.

This is O of N.

But if you search by index and you say, tell me what's at the index of three.

Those indexes allow you to go directly to that place in memory as an O of one operation.

This is one of the big advantages of arrays is that we can find something in an array with a million items in it, and go to item 400,000, say, and it is an O of one.

But those indexes are also a disadvantage for the arrays.

If you're adding something to the beginning, as we just saw, because you're going to have to re-index

Everything.

So the big thing that you have to look at when you're looking at a data structure is what are you using

this for if you need to access things by index.

Arrays are a great data structure, but if you're going to be adding and removing a lot of items from the beginning, maybe not the best data structure for you.

And maybe you should look at a different data structure if that is your use case.

But either way, you're making your decision based on Big-O.