Search arrays

Selecting transcript lines in this section will navigate to timestamp in the video

- We are able to access a specific item in an array because we know its index. We know where it's stored in the array. In searching, we usually want to return a true or false value depending on if the value exists in the array. Sometimes if the item is in the array, we'll also want to know its index. Now, the most obvious solution for searching is to check every item, to go through the entire data structure and check if the item at a given index matches the item we're looking for. If there's a match, we return true. If we've gone through the entire data structure and there is not a match, we return false. To do this in most languages, we can use a for loop, iterating over the data structure and checking each item at every index to see if it matches the item we're looking for. Here we're using a Java example where we go through every index and check if the element at that index equals our specific item. It it does, we return true. And if we go through the whole structure without finding it, we return false. We call this linear search or sequential search. It is an inelegant brute force method. We go through each item from beginning to end. This is the simplest way to understand searching, but it is also very slow, meaning it takes a lot of time to run. The more stuff you have, the longer the search takes. Of course, if the item you're looking for is the first item in the array, then the search is very fast. But in the worst case, the item you are looking for is not in the array at all and you have to check every single item in the data structure. This takes time. In the simplest terms, linear search is a linear time algorithm, meaning that the time it takes for the algorithm to run increases the size of the input. This means if you have an array with five times the number of items, we have five times the search time. With arrays, we also get search functionality. For example, in Java we can create an ArrayList and use the indexOf method to search for the index of a particular within a data structure. If the item is not in the ArrayList, we get back negative one. Let's look at an example of how this works in Java. Again, don't worry too much about the syntax here. Here we create an array with a list of values. Then we print out the index of the item with value 39 in the array and then the index of the item with value nine. 39 is not in the array, so we get back negative one. And nine is in the array, so we get back its appropriate index, which is 10. For the first indexOf query, the inputs were myArrayList and 39. For the second, the inputs were myArrayList and nine. Using the indexOf method looks pretty simple, and it is, but it's important to remember that this search is a linear search. Just because you aren't writing the linear algorithm doesn't meant it isn't happening behind the scenes, so you need to make absolute sure you need this in your code before calling methods like this. Since indexOf has no information about our data besides the ArrayList structure of it, we cannot make any assumptions about the values it contains and optimize it in any way, so we must check every element. Sometimes this linear search is your only option if the items are unsorted and you know nothing about your data. Sorting takes a long time and sometimes may not be worth it. But you can speed up your search time if your data structure is already sorted.