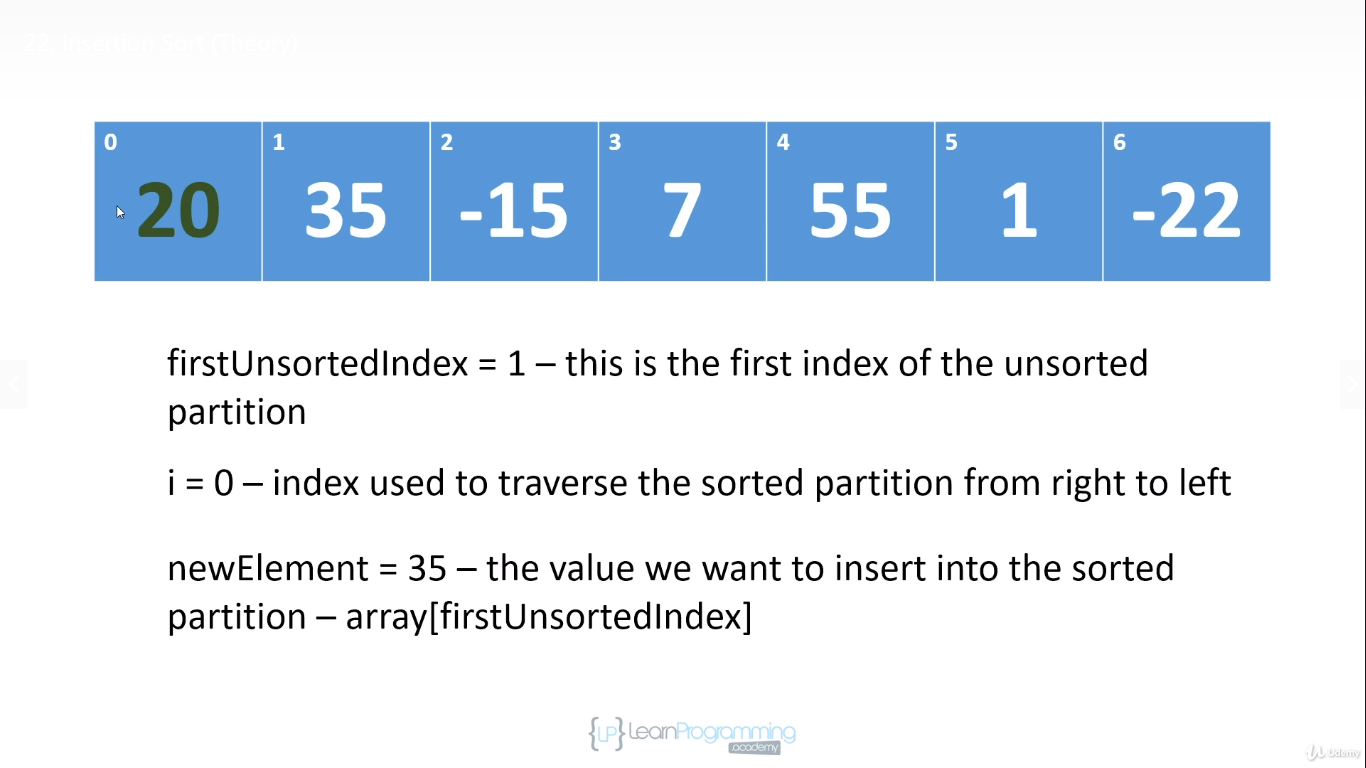
**Insertion Sort**

Like the other algorithms we've seen, its partitions the array into sorted and unsorted partitions. But this time the implementation I'm going to show you, grows as sorted partition from left to right. So it grows a sorted partition from the front of the array. So how does Insertion Sort work?



Well, it starts out by saying that the element position zero is in the sorted partition. And because this sorted partition is of length 1, by default, the element is sorted.

Coz if you have an array of length 1,or a partition of length 1, it's sorted, right?

There's only 1 element. So at the beginning, the elements from position 1 into the right or in the unsorted partition. And so we're going to set a first unsorted index field to 1. Now on each iteration, we take the first element in the unsorted partition which will be the element at array of first unsorted index, and we insert it into the sorted partition. And so at the end of each iteration we will have grown this sorted partition by one.

And so what we'll do on the first iteration is we will take 35, because that's the first unsorted value.

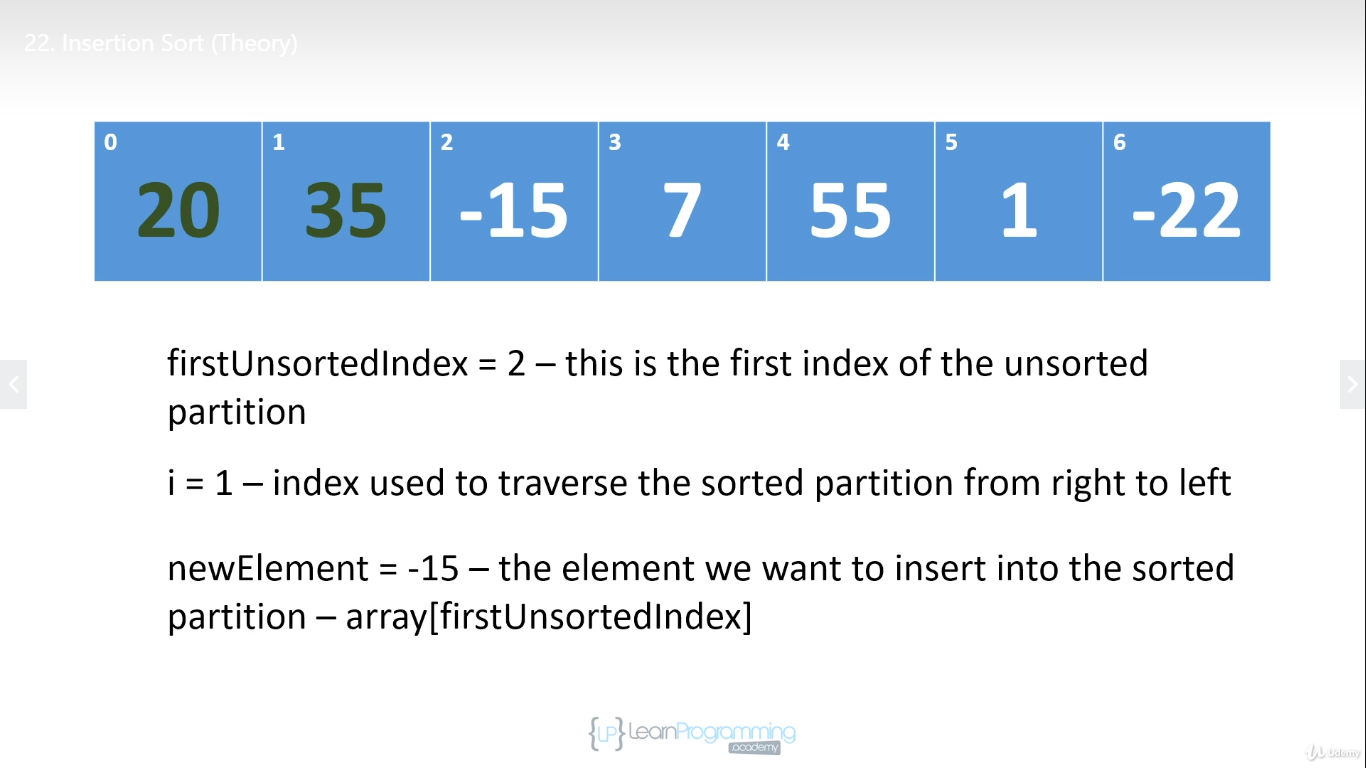
And we will insert it into the sorted partition. And at the end of the iteration the first two elements in the array will be sorted. So how is each element inserted?

Well, what we do is we compare the value we're inserting with the values in the sorted partition. And we traverse the sorted partition from right to left, and we look for a value that is less than or equal to the one we're trying to insert because once we find that value, we can stop looking we have found the correct insertion position for the new value that we're trying to insert.

Because remember, when we're inserting the value we are working within the sorted partition. And so if the element at index i in the sorted partition is less than or equal to the element we're trying to insert, then all of the values to the left of element i will be less than or equal to the value we're trying to insert, because all the values are in sorted order. So as we look for the correct insertion position, we shift values in the sorted partition to the right.

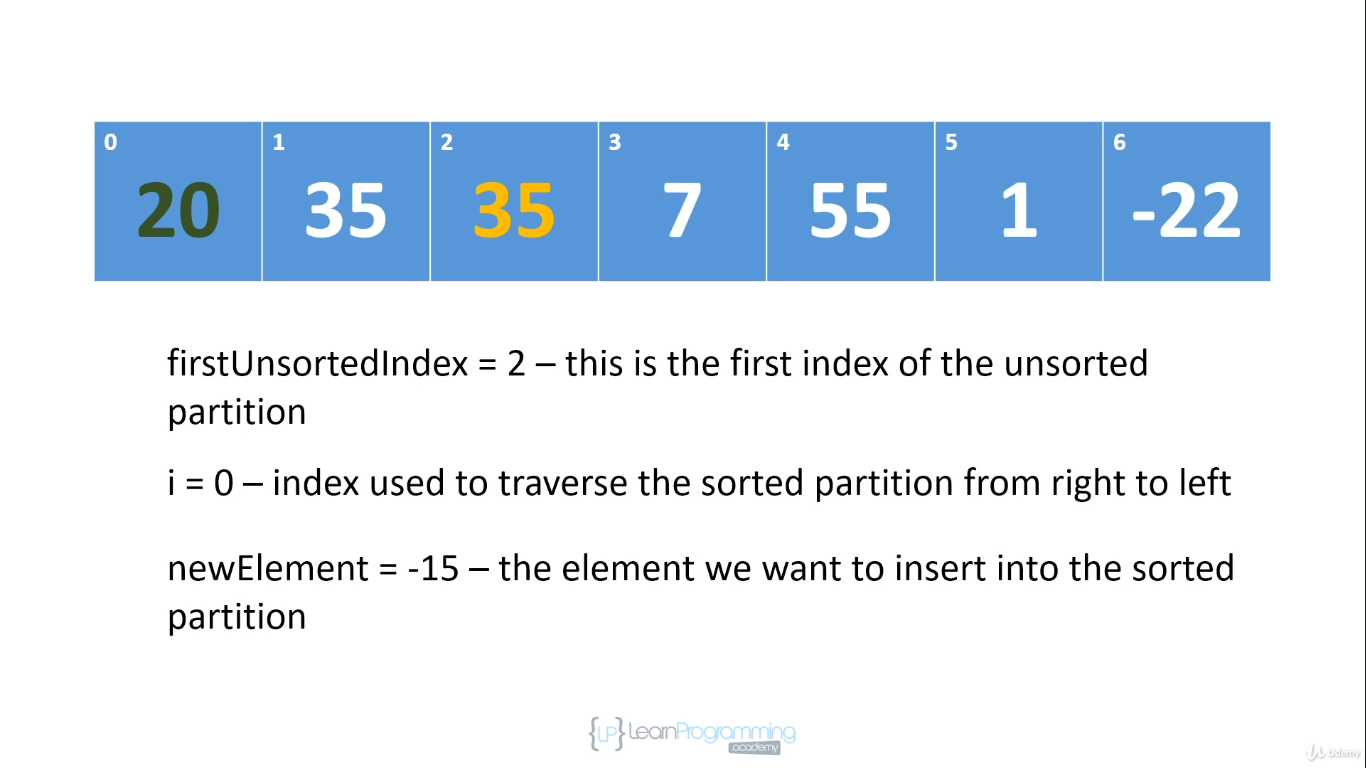
And you'll see this in action right now.

Let's go through this by hand. So on the first iteration, we're going to assign 35 to a new element field because 35 is the first element in the unsorted partition. And then we use i to traverse the sorted partition from right to left. So we compare 35 to 20. Now 35 is greater than 20 so 35 is already in its correct position in the sorted partition. It's not in its correct position in the array, but it is in the sorted partition. So after the first iteration, the sorted partition has been grown to two lengths two and the first two elements are in their correct position. And now the first unsorted index is at index 2 and i is assigned 1, because that's the right most index in the sorted partition.

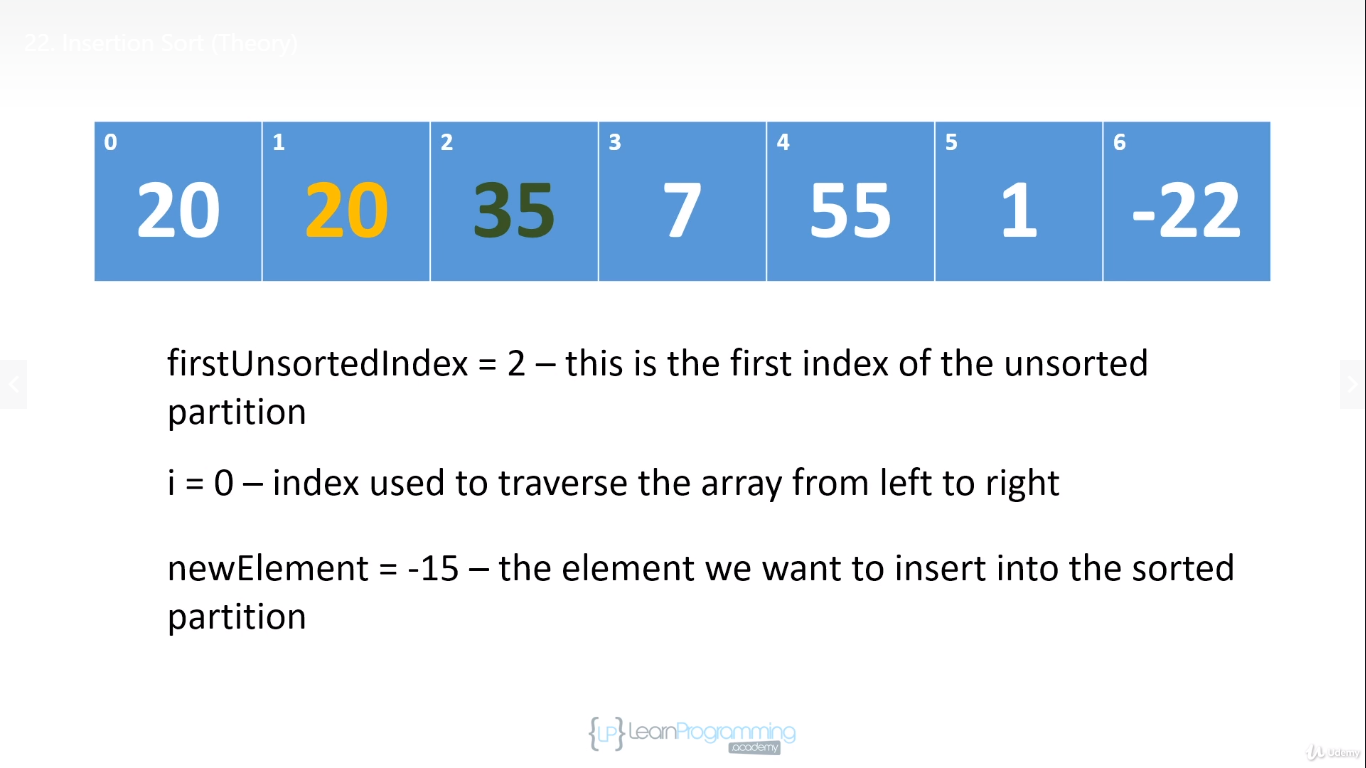


So we assign -15 to new element. And now we need to insert -15 into the sorted partition. So we compare -15 against 35 ,-15 is less than 35. And so we want to shift -15 down the array. But another way of looking at it

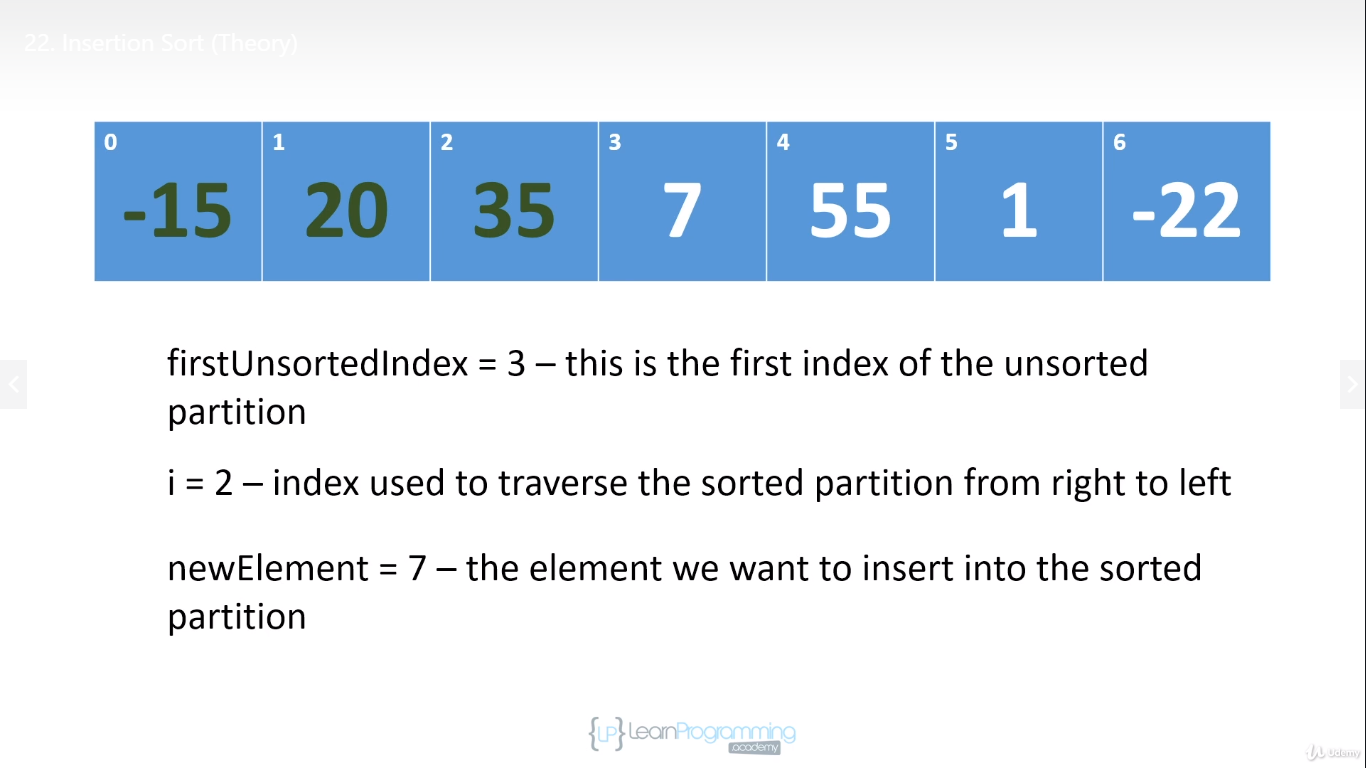
is we're gonna shift 35 to the right to make room for -15. And so we we assign 35 to position 2



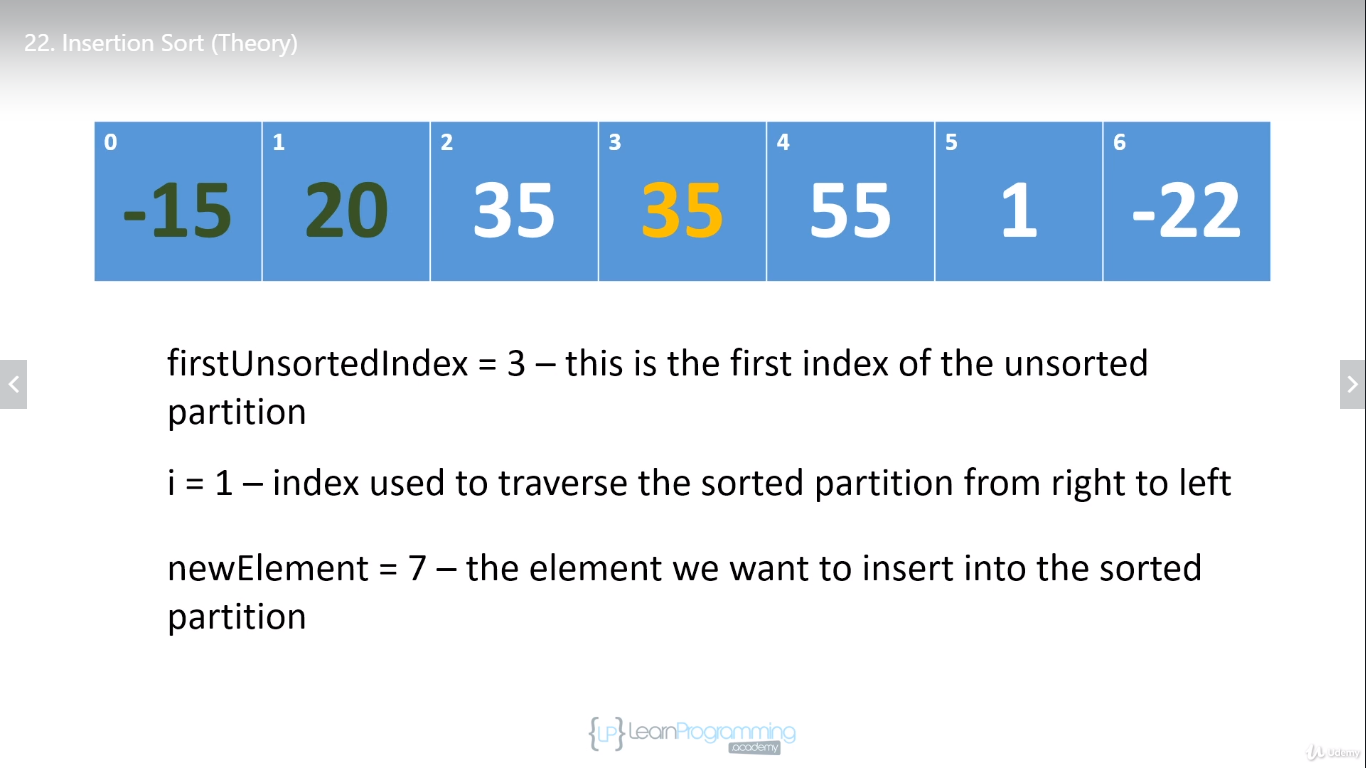
Now don't worry that might we've overwritten -15 because we have saved in a new element field. So now we're going to compare -15 to 20, 15 is less than 20 so we're gonna shift 20 to the right to make room for-15. And at this point, we've hit the front of the array so we have nothing else to compare -15 to basically we have the smallest element in the sorted partition and because we've hit the front of the array



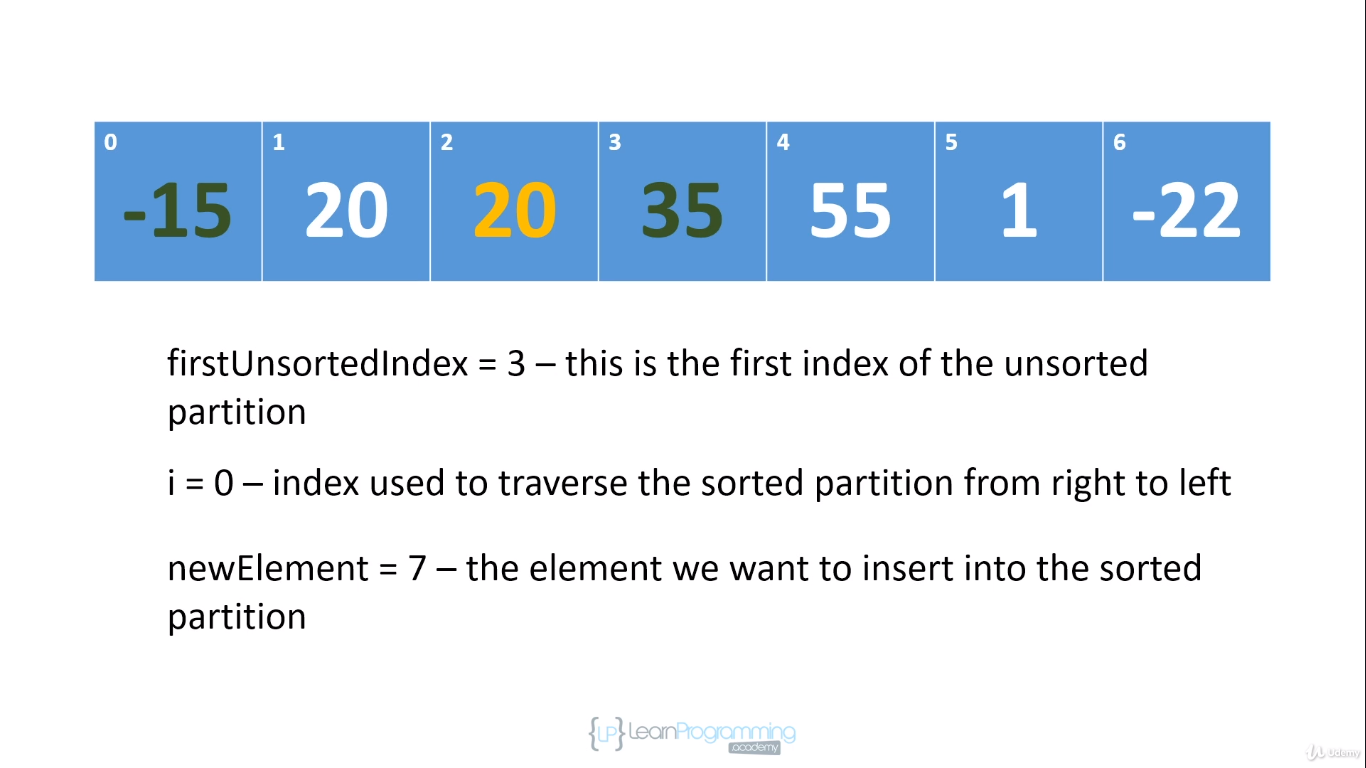
this is where we're going to insert -15. And so we go ahead and do that. And we've ended the second iteration. And at this point, we've grown the sorted partition to three. And the first three elements in the array, which is a sorted partition are in their correct positions in the sorted partition. So now the first unsorted indexes at position 3.



So we're going to assign the value of 7 to new element and we're going to compare 7 against 35. 7 is less than 35. So we're gonna shift 35 to the right.



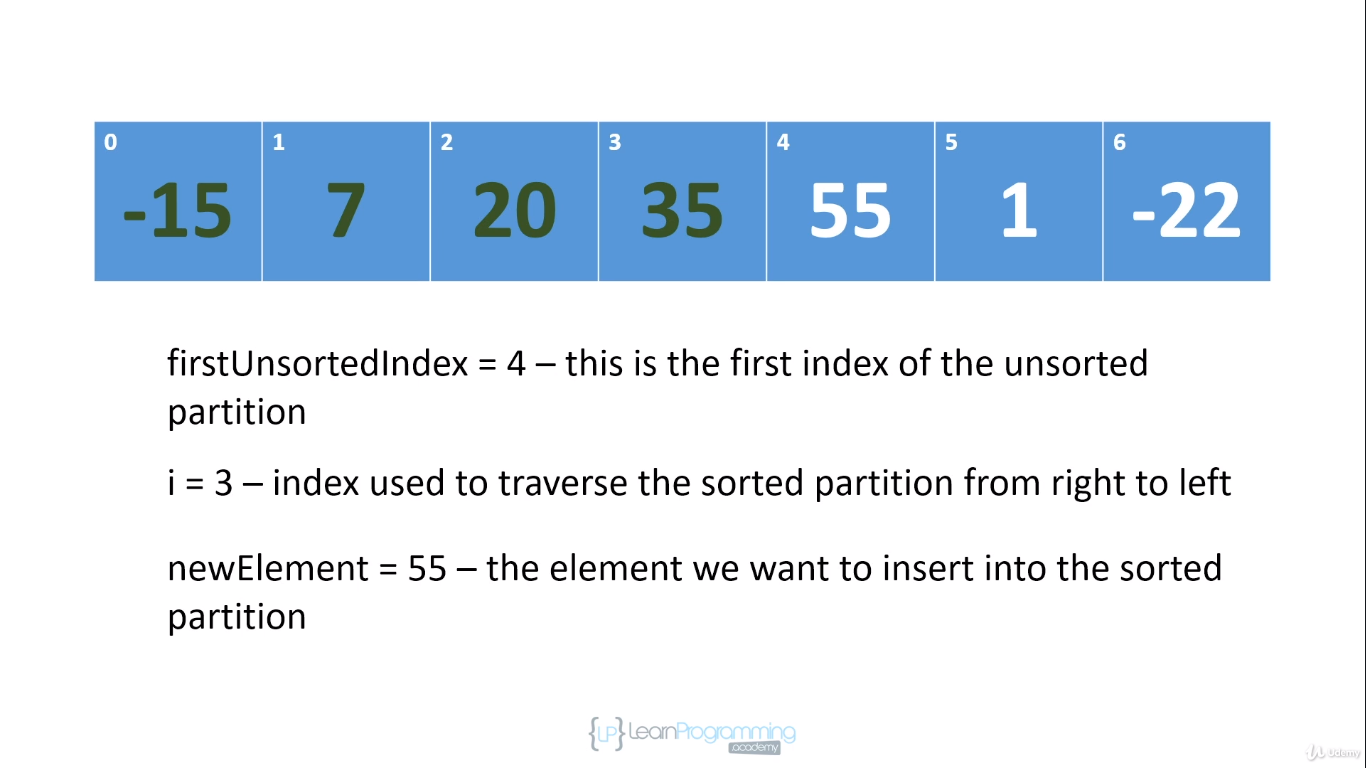
We compare 7 against 20, 7 is less than 20. So we're gonna shift 20 to the right



and then we 7 seven against -15. 7 is greater than -15. So we have found the insertion position.

Remember, we're working within the sorted partition. So if there was anything to the left of -15, all those values would be less than -15. So there's no need to keep traversing the sorted partition the moment we have hit an element

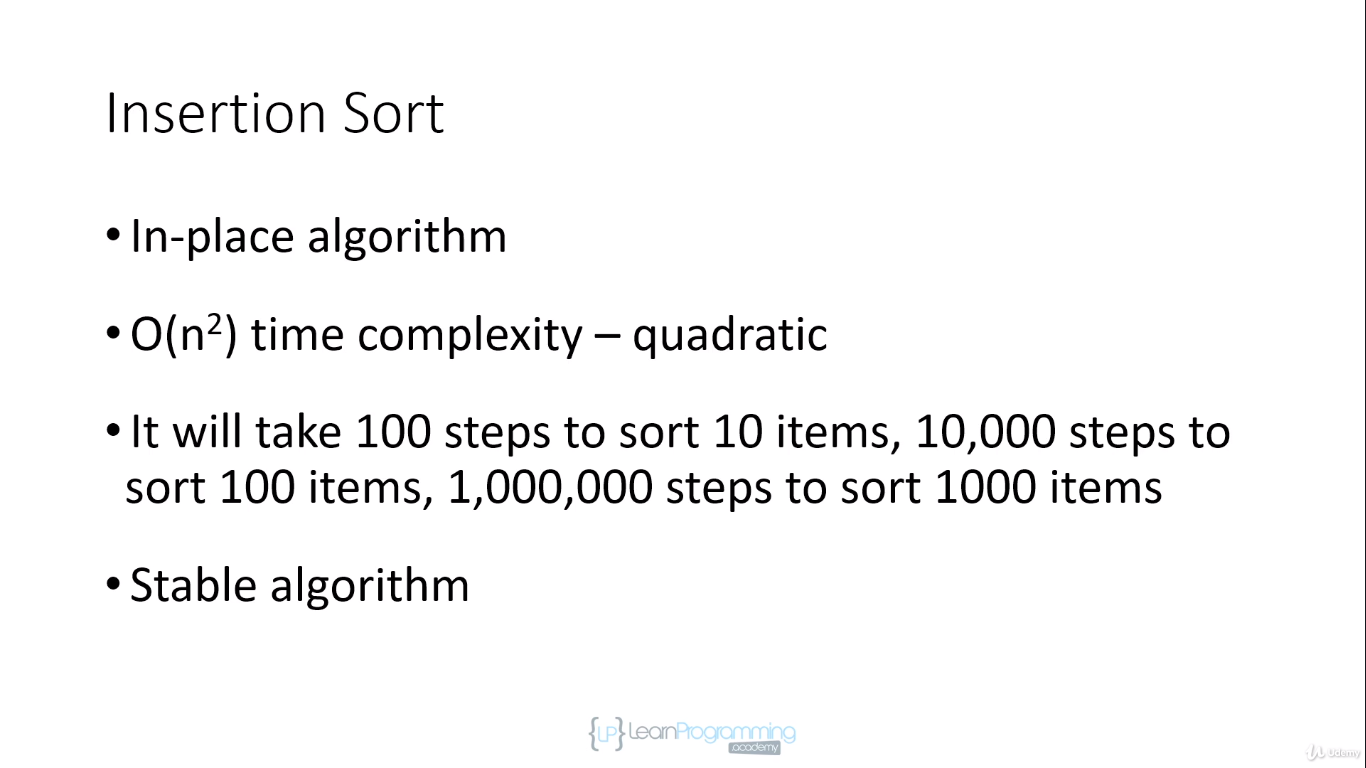
that's less than or equal to the one we're trying to insert, we found that correct insertion position. And so we're going to insert 7 into position 1. And we have completed another iteration, we've grown the sorted partition by one. And all of the elements in the sorted partition are sorted.



And so that's Insertion Sort.

It starts out by saying the first element is sorted. The implementation or at least the implementation that I'm going to show you does that. It grows the sorted partition from left to right. And on each iteration, it picks off the first element in the unsorted partition and it inserts it into the correct position in the sorted partition.

And it does that by shifting elements to the right to make room for the new element.



So Insertion Sort is an In-place algorithm. We don't need to create any temporary arrays, we have a few extra fields. But as I've said before, as long as the extra memory we're using doesn't depend on the number of items were sorted. It's an In-place algorithm.

It's a quadratic algorithm. And it's a stable algorithm. It's stable because when we're picking off elements, we're doing that from left to right. So if let's say there are two nines in the array, we're going to insert the left most nine first and then when we come to the right most nine. Remember that when we're looking for the insertion position, we stop when we hit an element that is less than or equal to the one that we're inserting. So when we're inserting the nine, we're eventually going to hit the first nine, we inserted into the sorted partition. And that second nine will be inserted to the right of the first nine. And so the relative positions of those two nines will be preserved. And so insertion sort is a stable sort algorithm.

