**Selection Sort**

we're going to take a look at the selection sort algorithm. Now this algorithm divides the array into sorted and unsorted partitions just like bubble sort does.

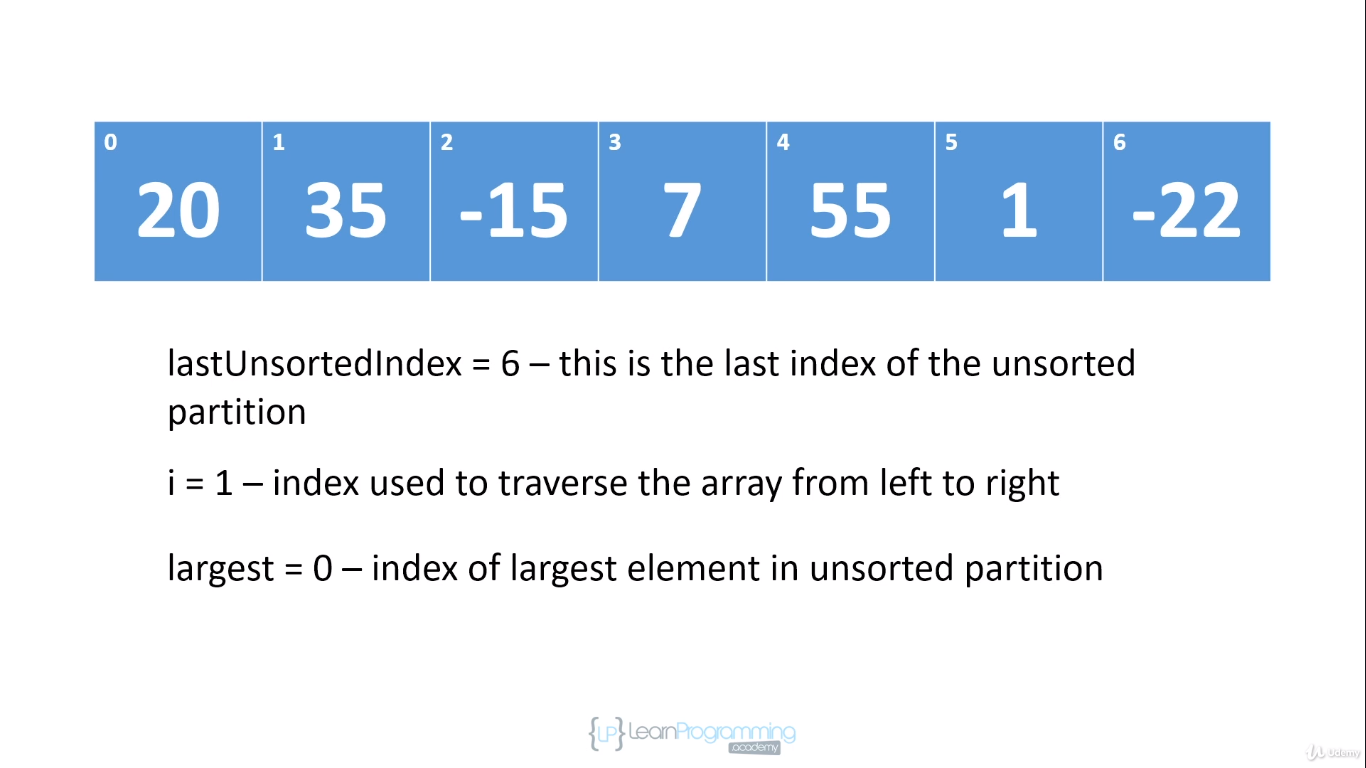
And what we do is we traverse the array and we look for the largest element in the unsorted partition. And when we find it, we swap it with the last element in the unsorted partition. And at that point, that swapped element will be its correct sorted position.

And so just like with bubble sort, at the beginning of the algorithm, the entire array is unsorted so the last unsorted index is six and just like with bubble sort we're going to grow the sorted partition from right to left.

We're going to initialize a largest field to zero so when we start we say you know what, 20 is the largest element that we know about so far, so whatever is at position zero will be the largest element.

And we're going to start by comparing the element at position one to whatever is at position zero,

so we're going to start with i equal to 1. We're going to use i to traverse the unsorted partition and find the largest element and so these are our initial values.

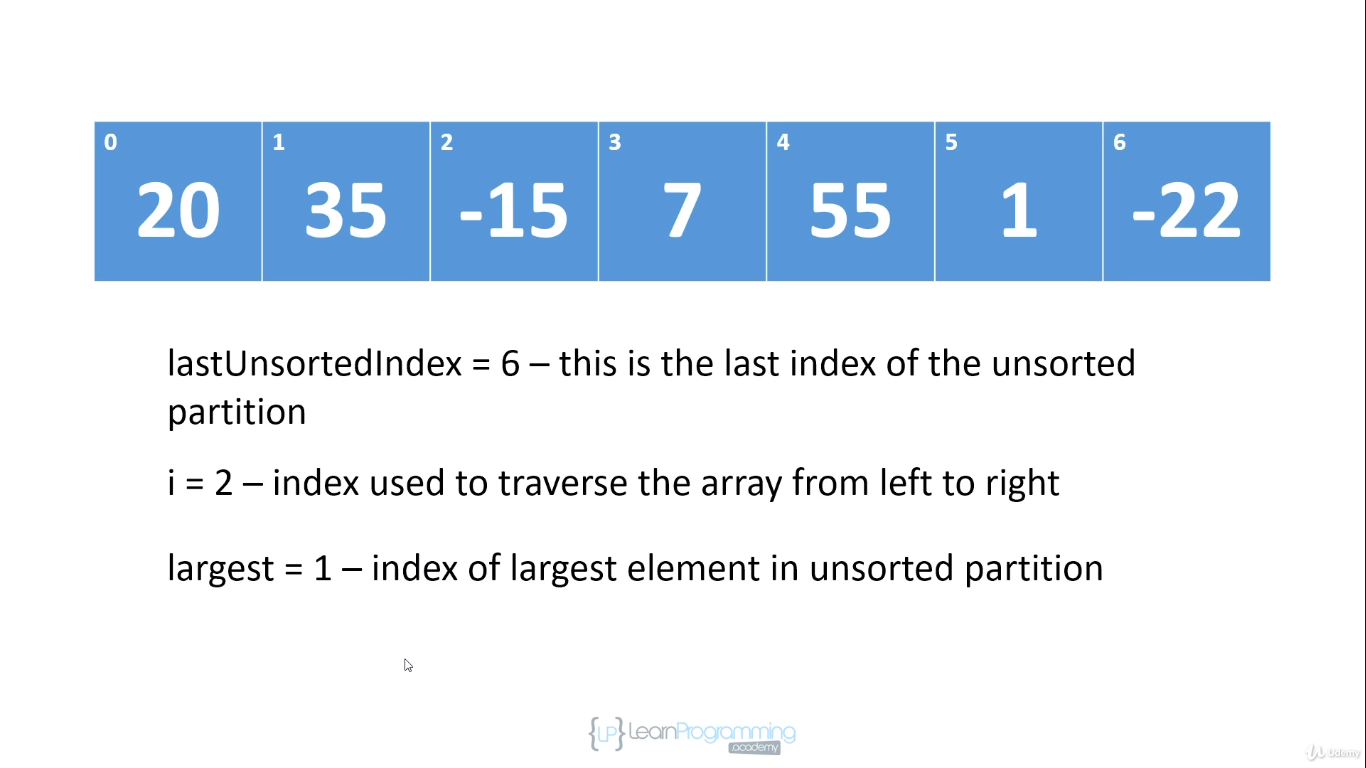


The last unsorted index is 6 because the entire array is unsorted, i is 1.

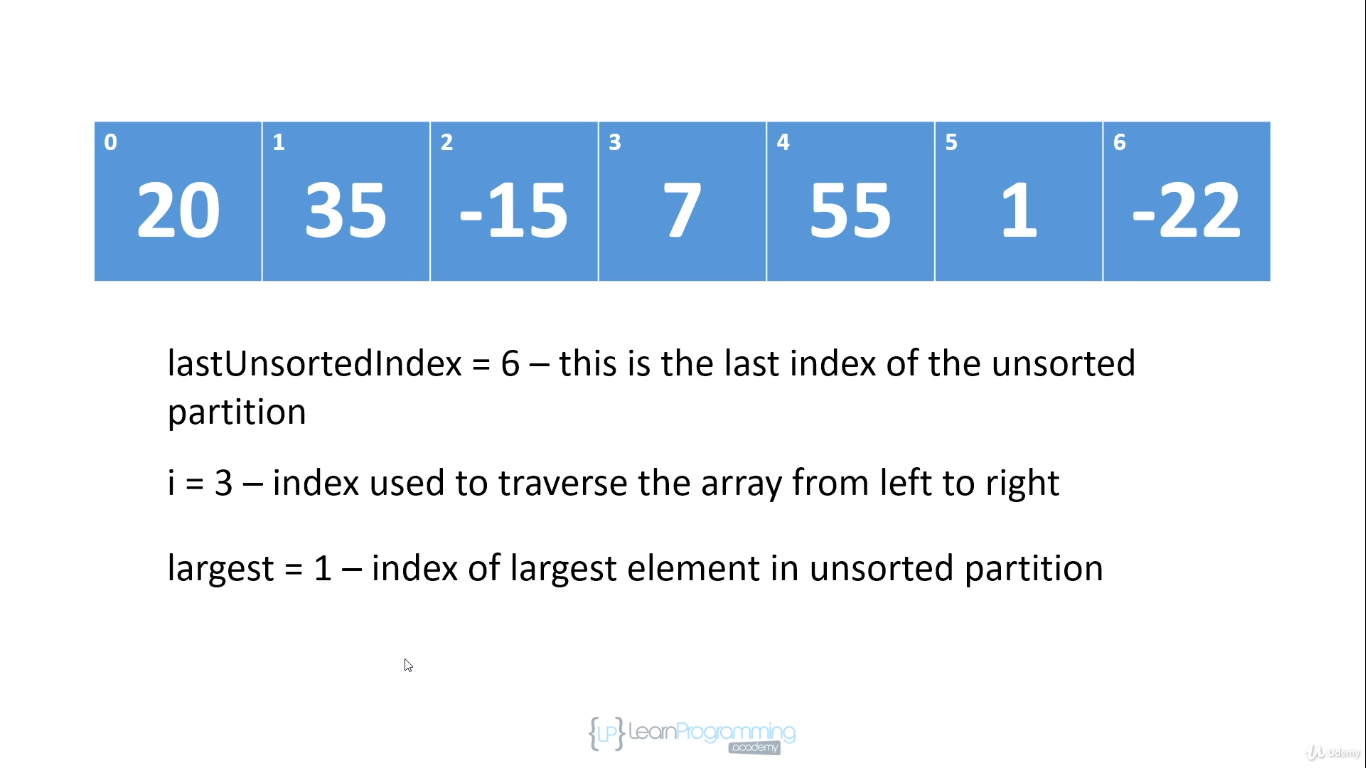
We're gonna start our traversal here and we've initialised the largest.

The largest will contain the index of the largest element in the unsorted partition, that's initialized to 0.

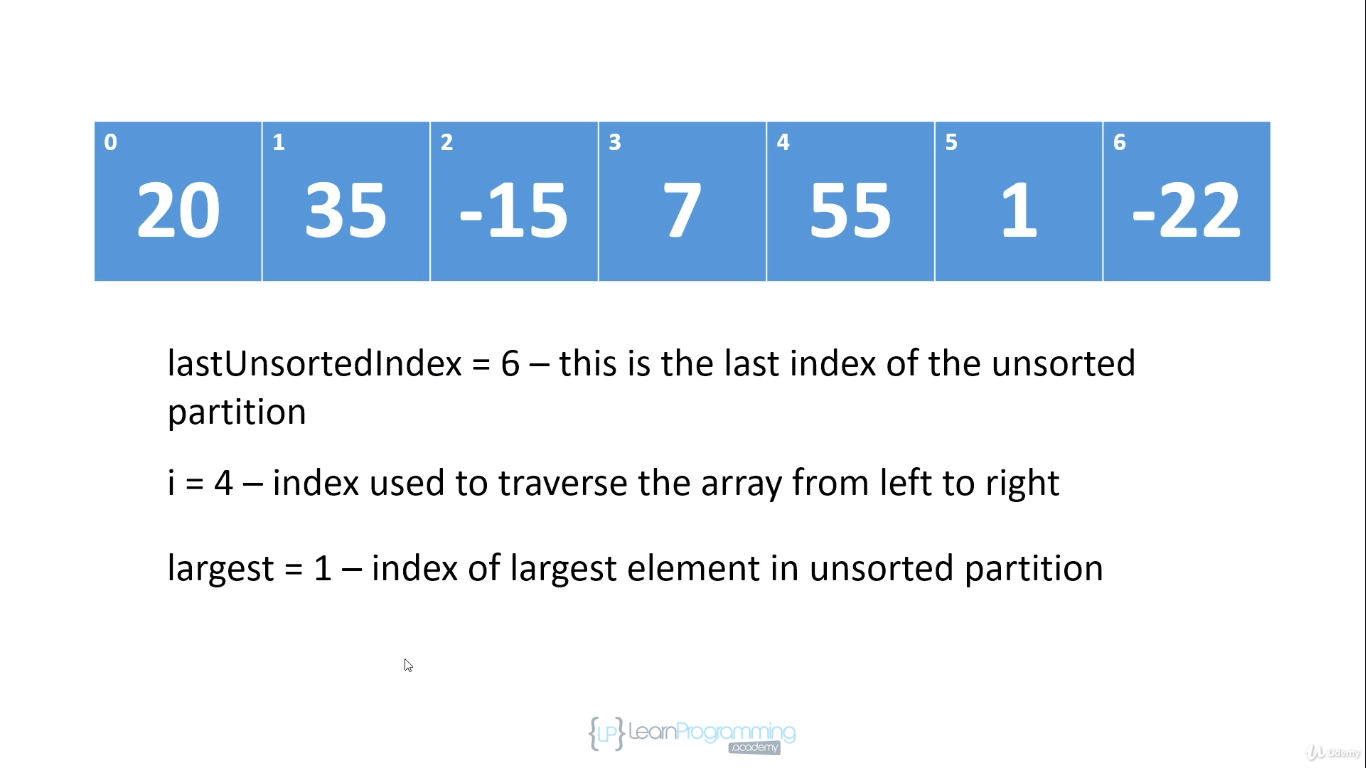
So we're going to compare 35 to 20 and we're gonna say hey, 35 is larger than 20 and because of that, we're going to change largest to 1 and then we're gonna increment i to 2



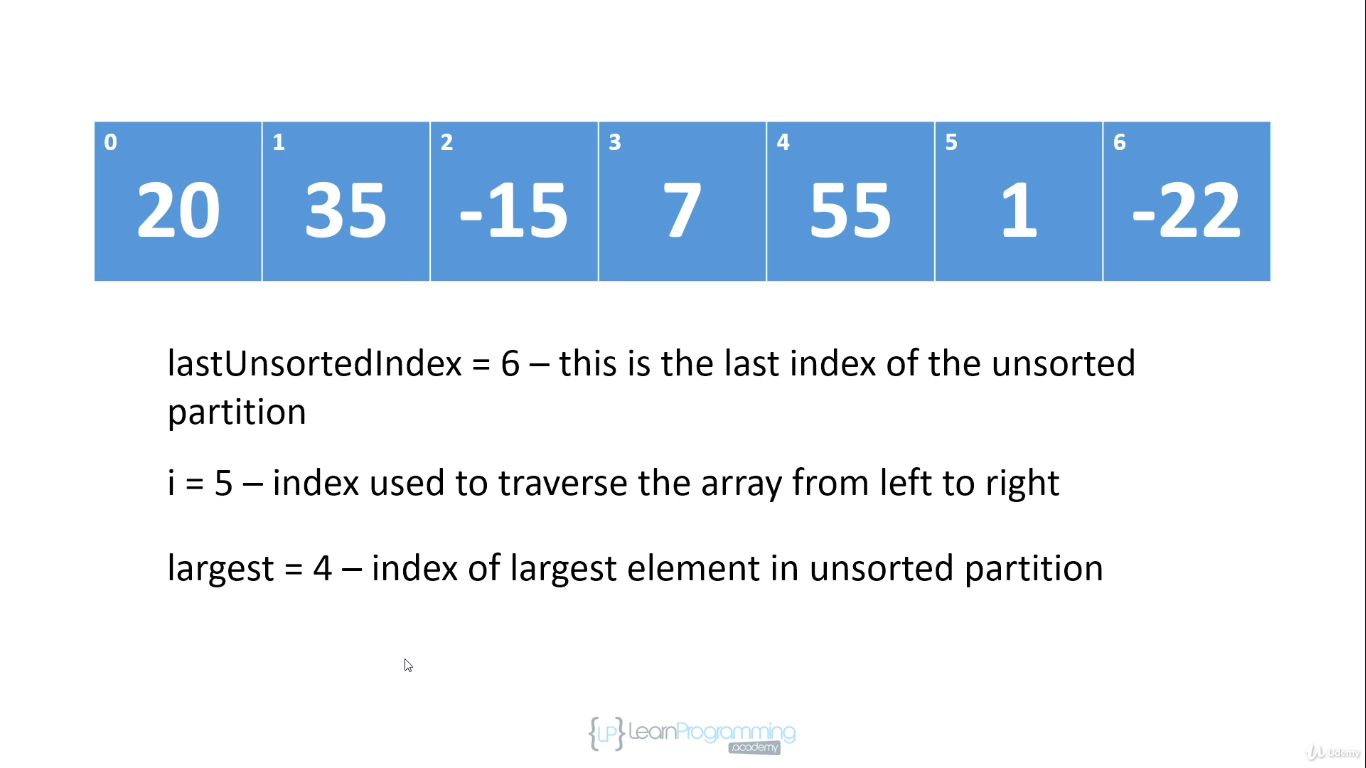
and we're gonna compare -15 to the largest element which is now at position 1 and we're gonna say well -15 is less than 35 so we're just gonna increment i to 3.



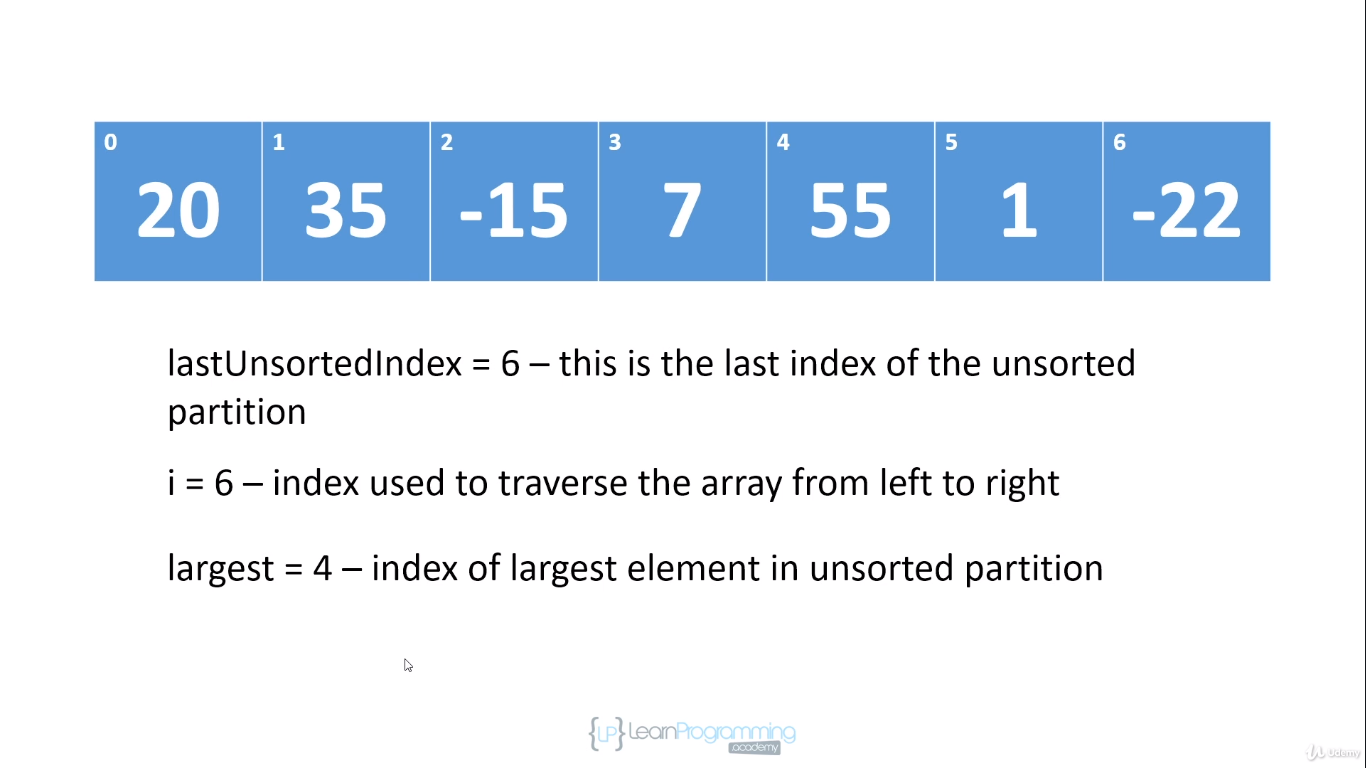
We're gonna compare 7 to the largest element which is still at position 1. 7 is less than 35 so we just increment i to 4.



We're now gonna compare 55 to 35 and 55 is greater than 35 so at this point we're going to change largest to 4 because the largest element we found so far in the unsorted partition is at index 4 and we increment i to 5.

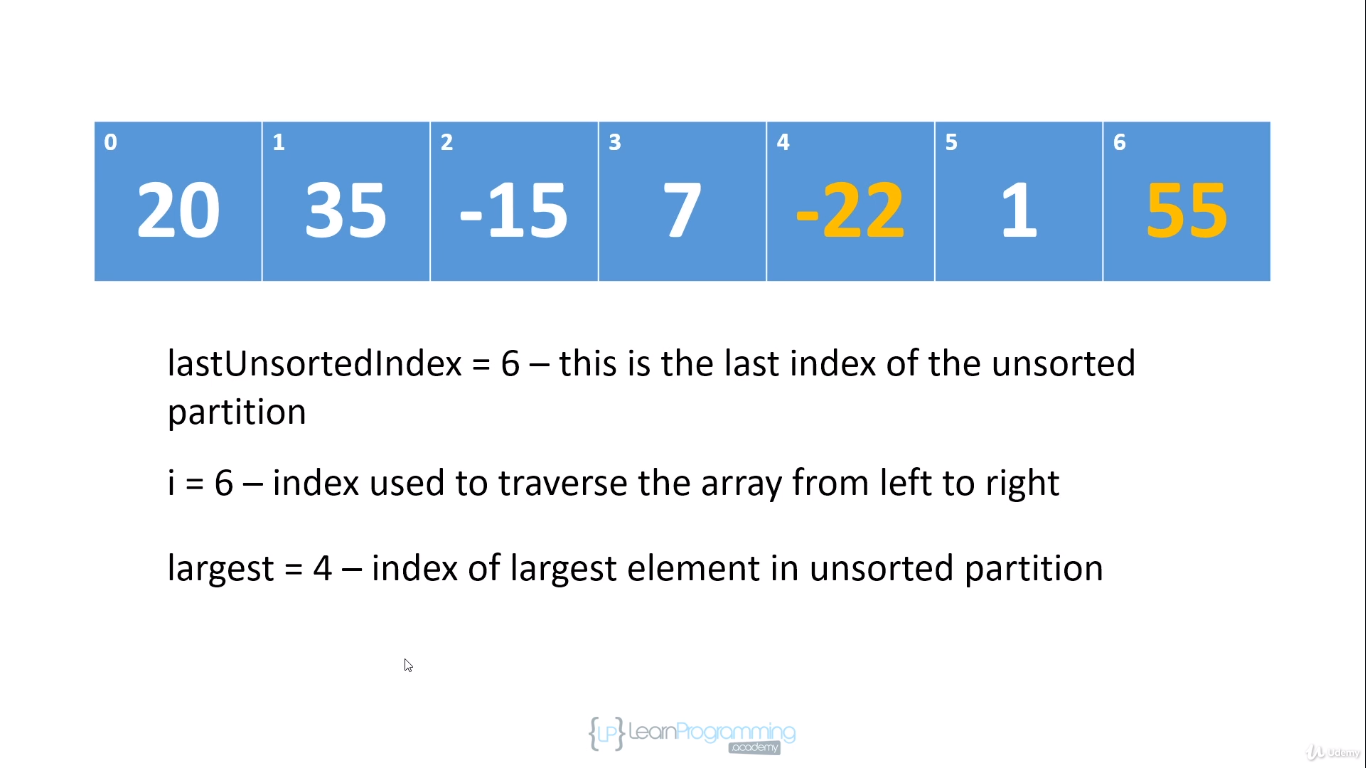


We compare 1 to 55. Well, 1 is less than 55 so we just increment i to 6.



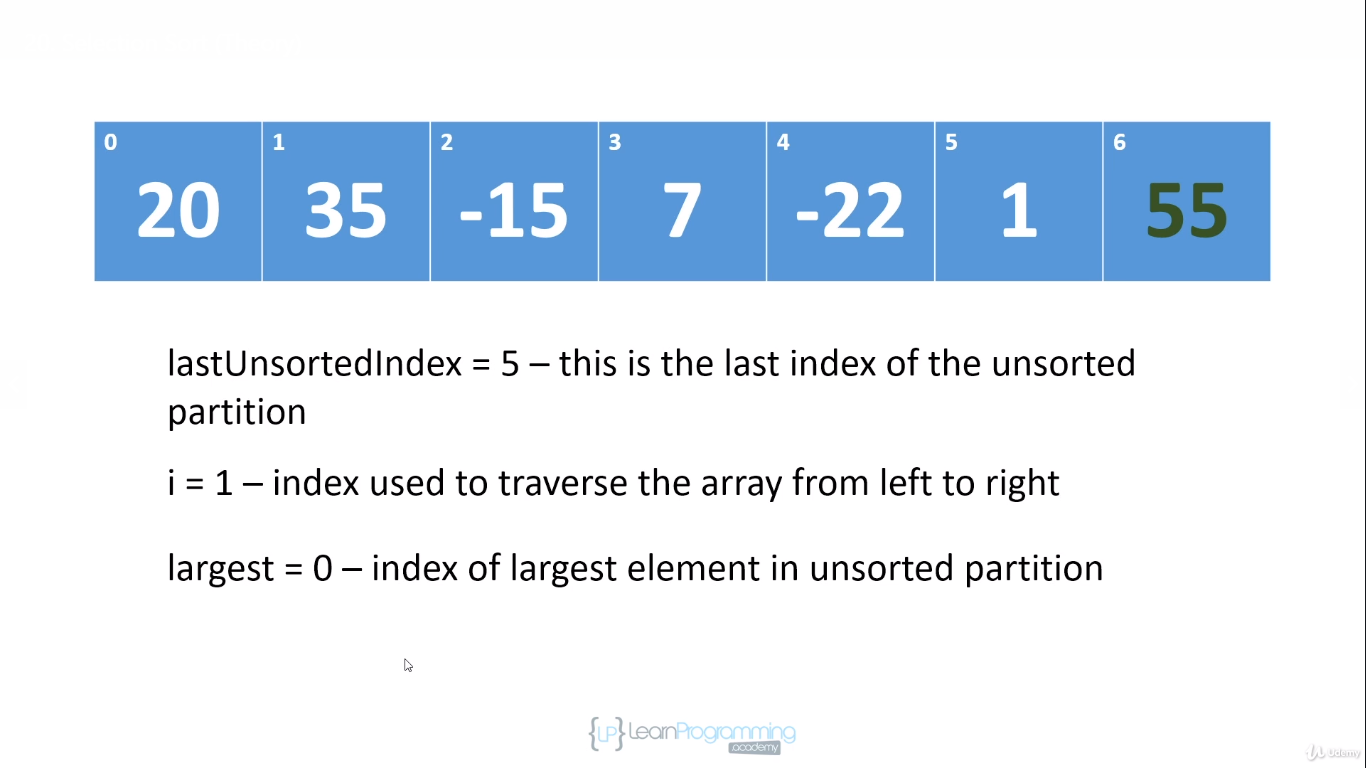
We compare -22 to 55.- 22 is less than 55 so we don't do anything. And at this point, i is equal to the last unsorted index and so we have completed our first traversal of the array.

So we're going to swap the largest element that we found in the unsorted partition and that's at position 4 with the last element in the unsorted partition and that's at position 6. So what we're gonna do is swap 55 and -22

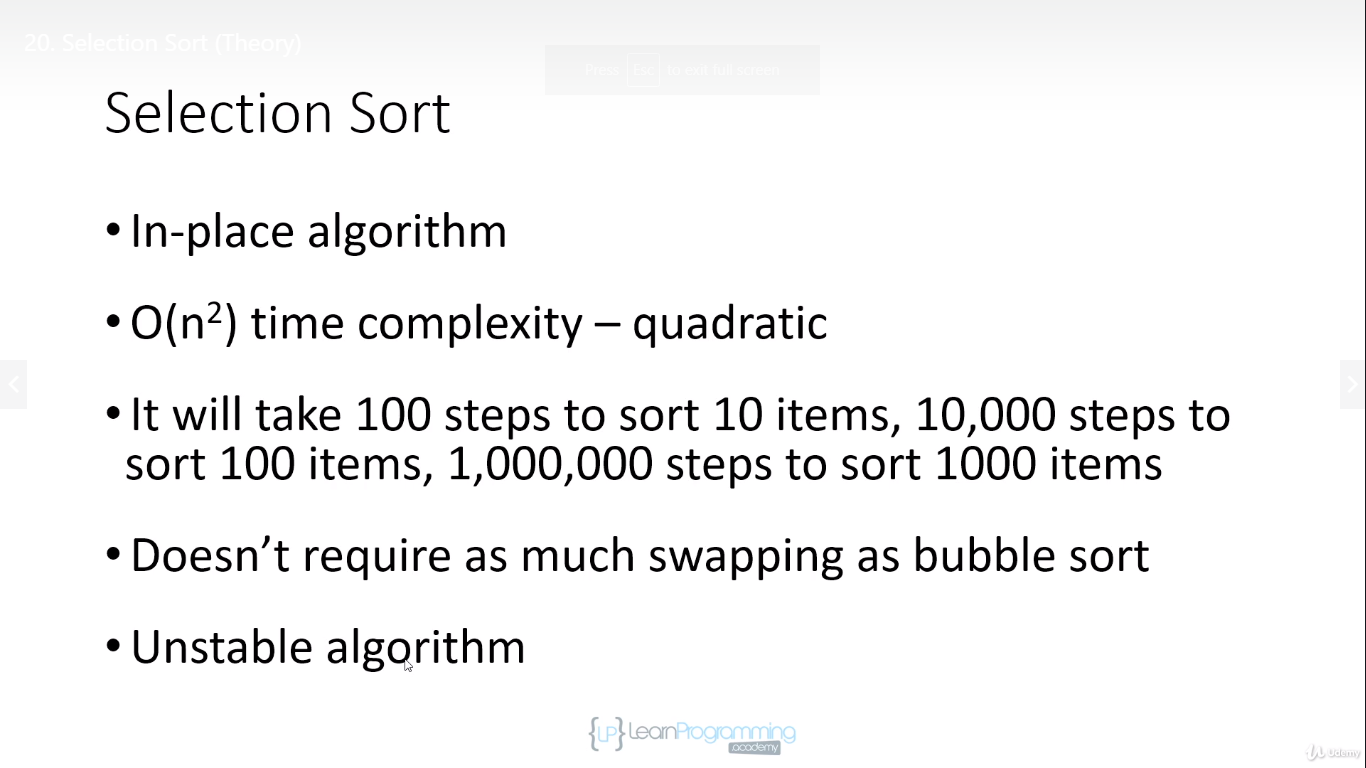


and we have now completed our first traversal. And at this point, 55 is in its correct sorted position and so we're gonna decrement last unsorted index to 5 and we're gonna re-initialize i to 1

and we're going to say the largest element in the unsorted partition is at position 0 and we repeat the process.



so this is how selection sort works and it's called selection sort because on each traversal where it's selecting the largest element and removing it into the sorted partition. So selection sort is an in-place algorithm. It doesn't use any extra memory. As I said with bubble sort, it's okay to use a few extra fields as long as the extra memory you're using doesn't depend on the number of items you're sorting, it's an in-place algorithm.



It's a quadratic algorithm so it has a time complexity of O(n2) because we have n elements in the array

and for each element we traverse n elements, so it's quadratic.

However, it doesn't require as much swapping as bubble sort. You'll notice that we only swap once per traversal and so selection sort will usually perform better than bubble sort. I say usually because depending on you might have an array that's almost sorted and so bubble sort doesn't have to swap that much, but in the average case, all other things being equal, generally selection sort will perform better.

However, selection sort is an unstable algorithm and you can see why, right? Because if we have duplicate elements,

there's no guarantee that their original order relative to each other will be preserved because on each pass, we swapped the largest element with whatever is occupying the last position in the unsorted partition and so it's very possible that we could take the second duplicate value and move it in front of its twin. So because of that, selection sort is an unstable algorithm. So if you need a stable sort algorithm, then you don't wanna be using selection sort. Okay, so that's it for the theory.

Code for Selection Sort

