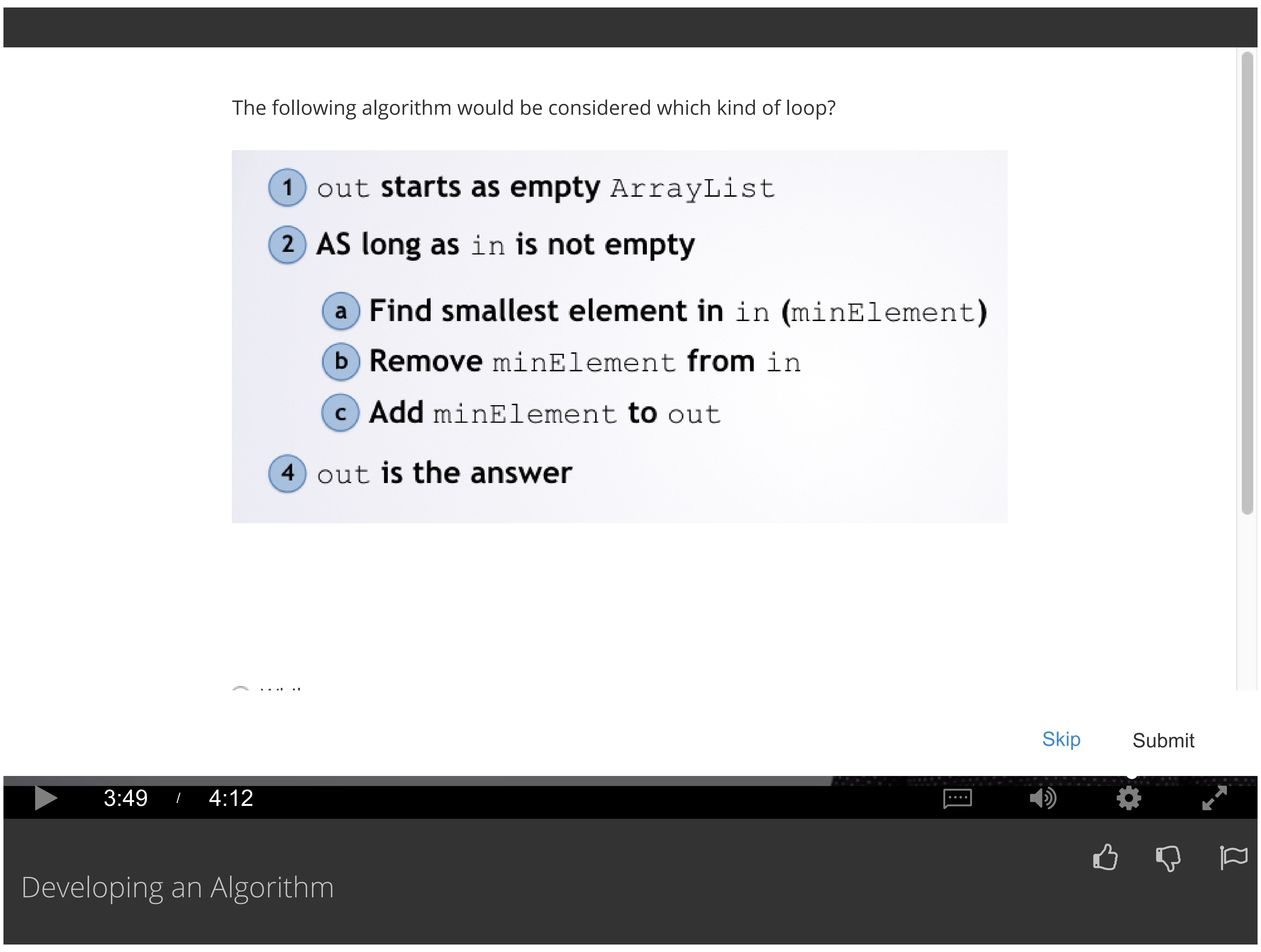
Implementing Selection Sort

Module Learning Outcomes / Resources	10 min
Introduction	1 min
Developing an Algorithm	4 min
Translating to Code	3 min
In Place	9 min
Efficiency	5 min
Summary	1 min
Programming Exercise: Implementing Selection Sort	10 min
Practice Quiz: Implementing Selection Sort	5 questions

Sorting at Scale

Review



The following algorithm would be considered which kind of loop?

```
1 out starts as empty ArrayList
2 AS long as in is not empty
    a Find smallest element in in (minElement)
    b Remove minElement from in
    c Add minElement to out
4 out is the answer
```

Developing an Algorithm

Have a question? Discuss this lecture in the week forums.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?

Interactive Transcript

Search Transcript English

0:03
So, how would you go about sorting data? There are many ways to sort. Some more efficient than others. As with all things you can use the seven steps to devise a solution to the problem. Here we start with a small example using numbers. Sort 56, 17, 4 and 33. Even though you want to sort earthquakes, you can design the algorithm by working with numbers. Then adjust it to extract the particular data, magnitude, distances, depth, or whatever that you want to sort by. Working through this small instance by hand is fairly straight forward, as the list is small and it is easy to see how to put the elements in order. However, as always, you must be careful to do it in a step by step fashion, and not just write down the answer. Now that we have done this, let us go back and carefully think about what we did and write down those steps. First, let us name things in and out to make them easier to refer to clearly.

0:59
You should also explicitly note that out starts as empty ArrayList. That is an easy step to mentally gloss over but it's important to include when you translate the code.

1:10
The next thing we did was find the smallest element it in, which was 4. Then we remove 4 from in and added it to out.

1:18
Next, we found the new smallest element in in, which was 17, so we removed 17 from in and added it to the right end of out.

1:27
Now, the smallest element in in is 33, so we remove that from in and add it to the right end of out. Finally, 56 is the smallest element in in, so we remove it and add it to the right end of out.

1:41
Now, you are done. Out has the data sorted the way you want it so it is your answer.

1:48
Now you have these 14 steps to start this particular data set. Of course, you'd like to be able to sort any set of data so you want to generalize these steps into an algorithm which works on any set of data of any size. As usual, you can see repetition in what you are doing. But, as is also typical, there are some differences between the similar steps. You need to find the patterns in these before proceeding.

2:15
The difference between these similar steps is the particular numbers in each group. Here whatever you found for the minimum element in the first step of the group is the number that is removed from in and then added to out in the other two steps. As usual you should give this a name and use that name to make the repetitive steps completely uniform.

2:35
Here we've named this element, this value, minElement, and made the steps uniform. We are almost ready to express these as repetition, but there's one last detail we need to think about. How do we know when to stop repeating these steps? Unlike many algorithms that you have seen before, we are not doing for each element of the input. Not only are we not going through them in order, but it is generally a bad idea to try to do for each element while removing items from the array list.

3:07
Let's go back to where we worked this problem by hand to think this through. Going back through this you can clearly see that we are not doing a for each element repetition. As we're not working with each element in order. Now that we are done it's pretty clear that we're done, but how can you tell? It is because in is empty. Since this is how you can tell that you're done, it is also what you should express in your algorithm. With that observation, you could write an algorithm that looks like this. Notice how we express the repetition based on what we just doctored. You want to repeat these steps as long as in is not empty. When you go to translate this to code what kind of loop would this be?

3:50
Hopefully, you remember wild loops from before. They're the best way to express this kind of repetition.

3:57
As usual, it's a great idea to test out your algorithm before you translate the code. Try this algorithm out for this input. 9, -3, and 0. Did the algorithm work correctly? Great! It's time to turn it into code.