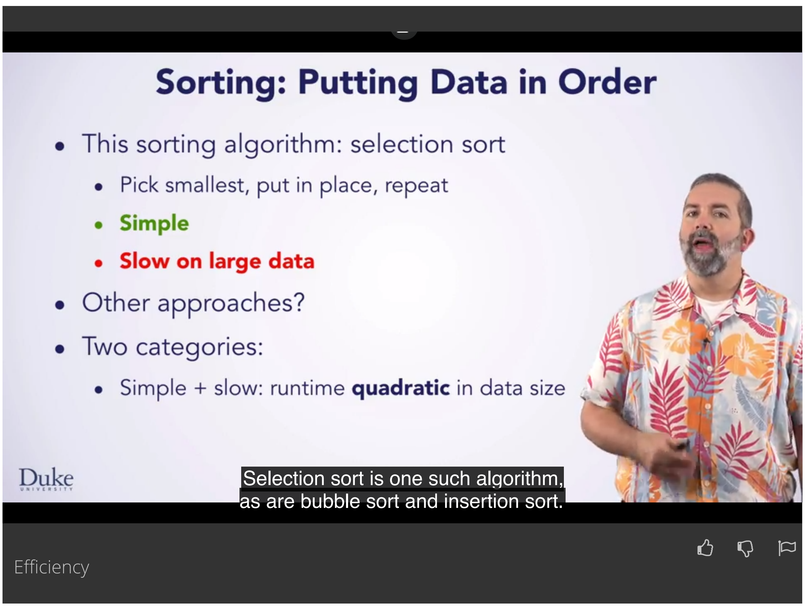
Implementing Selection Sort

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



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

Interactive Transcript

Search Transcript

English

0:03
This sorting algorithm has a name, selection sort, because it selects the smallest element and then adds it to the end of the output.

0:11
One of the great things about this algorithm is that it's conceptionally simple. Many people would come up with it themselves via the seven steps, and it is relatively simple to implement. However as sorting algorithms go, it is rather inefficient. If you were to use it on a small data set this would not be a big problem since computers are fast. However if you had a very large data set it would be quite slow.

0:34
So are there other approaches? Of course there are. In fact, there are dozens of sorting algorithms.

0:41
They generally fall into two categories. The first of these are simple to understand and implement in code, but slow. Selection sort is one such algorithm, as are bubble sort and insertion sort. Their running time is quadratic in the input size. If you double the size of your input, the algorithm will take four times as long to run. The other category of algorithms is those which are more complex to understand but much much faster. Examples of such algorithms include quick sort, merge sort and others. The algorithm used in the Java library collections.sort is a variation of merge sort called tim sort. The runtime for these algorithms is close to linear. If it were linear, doubling the input size would only take twice as long. These efficient algorithms grow slightly more than linear, but are very close to linear.

1:35
For the simple and slow approach, the runtime grows quadratically. So these are called n squared sorts. Both selection and bubble sort are n squared sorts. They have the same general shape, and the algorithms are easy to understand.

1:51
Here, you see a graph of the run times for two of these n squared sorts, taken from sorttimings.java.

2:00
As you can see, these algorithms are reasonable for 20,000 strings, taking two and four seconds respectively.

2:08
For small lists this might be acceptable, but for large lists, the other category of sorting algorithms is much much better.

2:18
Let's look at the timings for even more elements, to understand why n squared sorts, are called inefficient. This graph shows that n squared sorts from 10,000 to 70,000 strings. Time in seconds is labeled on the y axis, and the number of elements being sorted is on the x axis.

2:36
How long would it take to sort many, many more elements? We can use a quadratic fit of the bubble sort, the first equation shown, to extrapolate the sorting for a million, or a billion elements. So how big a difference is there?

2:54
To sort a million strings would require 6.4 hours with bubble sort. Using Collections.sorts, Tim sort function requires less than one second to sort the same million strings.

3:07
To sort a billion elements would require 738 years with bubble sort. We can actually time collections.sort and see that it takes fewer than 20 minutes to sort a billion strings.

3:21
Fortunately, many languages have an efficient sort built in as part of their standard library. Of course, such a sort should work on a variety of types so that programmers can get the most use out of it. In fact, it has to work with data types that the sort's author didn't think about specifically. For example, you want to sort earthquakes. Do you think the author of Java's sorting library was thinking about earthquakes when he wrote that sort? Probably not, and certainly he was not thinking about the particular earthquake class you wrote. So how does this work?

3:56
Remember interfaces? You saw these already as a way to write generic code. An interface is a type that promises certain methods. Code, such as a sorting library, can then use the interface type and call the method it promises.

4:11
Other code can then make instances of classes which implement the interface and pass them to the sorting library. For sorting there are two important interfaces, comparable and comparator, which you will learn about shortly.

4:27
In Java this built in sort is called Collections.sort and it is quite efficient. It is what you should use any time you need to sort data.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

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