Lecture 12: More on selection sort. Proofs by induction.

Doina Precup
With many thanks to Prakash Panagaden and Mathieu Blanchette

January 31, 2014

Last time we started discussing selection sort, our first sorting algorithm, and we looked at evaluation its running time and proving its correctness using loop invariants. We now look at a recursive version, and discuss proofs by induction, which will be one of our main tools for analyzing both running time and correctness.

1 Selection Sort revisited

The algorithm can also be written in a recursive way as follows:

Algorithm selectionSort(a, n)
Input: An array a of n elements
Output: The array will be sorted in place (i.e. after the algorithm, the elements of a will be in nondecreasing order
if $n \leq 1$ **return**
int $indmax \leftarrow$findMaxIndex(a,n)
swap(a,n, $indmax$)
selectionSort(a,$n-1$)

This is an example of **tail recursion**: the recursive call is executed only once, on almost the entire array. Tail recursion in general does not give any speed gains compared with using loops. Indeed, if the compiler is smart, it will convert tail recursion into a loop instead (because running a loop does not require using the execution stack, and hence is faster in practice). We will discuss different types of recursion again later on. The algorithm has the same big-oh as the loop version from last lecture, as the number of comparisons needed is the same (comparisons occur to check the base case, and inside findMaxIndex).

2 Mathematical induction

Let's come back to the equality that we used as the basis for determining the complexity of selection sort. We obtained it based on a calculations - but can we prove more formally that it is correct? To do this, we will use a proof technique called **proof by induction**. This is a very important way

of proving both mathematical statements over sequences of integers, as well as statements about the complexity and correctness of recursive algorithms.

The goal of mathematical induction is to prove that some statement, or proposition $P(n)$ is true for all integers $n \geq a$ for some constant a. For example, we may want to prove that:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

(this is a fact that we will use on several occasions). So $P(n)$ is the fact above, and we want to prove that it is true for any integer $n \leq 1$.

One way to visualize what we want to do is to imagine that we wrote the statement of $P(n)$, for every n, on a piece of domino. Now we arrange these dominoes in a long line, so that we only have access to the first domino in the line. Proving that $P(n)$ is true for all n is the same as trying to topple all the dominoes. First, we need to make sure that we can topple the first one. Then, we need to make sure that any two dominoes are close enough, so that if the nth one falls, the $(n+1)$th will fall as well. Intuitively, if we ensure these two facts, then we can topple the first domino and all of them will go down.

Proofs by induction work exactly based on this intuition. If we want to prove that $P(n)$ is true for any $n \geq a$, we will do it in two steps:

1. **Base case:** Prove that $P(a)$ is true (i.e., we can topple the first domino)
2. **Induction step:** If $P(n)$ is true, then $P(n+1)$ is also true (i.e, if the nth domino falls, then $n+1$th will also fall) . We will call $P(n)$ the **induction hypothesis**.

If we can prove these two things, then, by the principle of induction, $P(n)$ is true for all $n \geq a$.

More generally, in the base case, we may show not only $P(a)$, but several cases: $P(a+1)$, $P(a+2)$, ... Then, in the induction step, we will show that if $P(n)$ holds for any $a \leq n \leq b$ then $P(b+1)$ also holds. In other words, if all the dominoes up to b have fallen, then $b+1$ will also fall.

As an example, suppose that we want to prove:

$$P(n) : \sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \forall n \geq 1$$

The proof goes in two steps:

1. Base case: suppose $n = 1$. Then $\sum_{i=1}^{1} i = 1 = \frac{1 \cdot 2}{2}$. So the base case is correct
2. Induction step: suppose that $P(n)$ is true, i.e. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ (this is the induction hypothesis). Now show that $P(n+1)$ is also true, i.e., $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$. To do this, let us proceed by breaking up the sum into two parts:

$$\begin{aligned}
\sum_{i=1}^{n+1} i &= \left(\sum_{i=1}^{n} i\right) + (n+1) \\
&= \frac{n(n+1)}{2} + (n+1) \text{ (by using the induction hypothesis)} \\
&= \frac{(n+1)(n+2)}{2}
\end{aligned}$$

In most proofs by induction, in the induction step we will try to do something very similar to the approach here; we will try to manipulate $P(n+1)$ in such a way as to highlight $P(n)$ inside it. This will allow us to use the induction hypothesis.

Here are now some more examples of induction:

1. Prove that $2^n < n!, \forall n \geq 4$.
   Proof:
   Base case: For $n = 4$, we have $2^4 = 16$ and $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$
   Induction step: Suppose the $2^n = n!$. Is it true that $2^{n+1} < (n+1)!$? We have:

   $$\begin{aligned} 2^n &< n! \text{ (by the induction hypothesis)} \\ 2 &< (n+1) \text{ (because } n \geq 4\text{)} \end{aligned}$$

   By multiplying the two together, we get the result. This concludes the proof.

2. Prove that $1^3 + 2^3 + \ldots n^3 = (1 + 2 + \ldots + n)^2$
   Proof:
   Base case: $n = 1$ - obvious
   Induction step:

   $$\begin{aligned} \sum_{i=1}^{n+1} i^3 &= \sum_{i=1}^{n} i^3 + (n+1)^3 \\ &= (1 + 2 + \ldots + n)^2 + (n+1)^2(n+1), \text{ using the induction hypothesis} \\ &= (1 + 2 + \ldots n)^2 + (n+1)^2 + 2\frac{n(n+1)}{2}(n+1) \\ &= (1 + 2 + \ldots n)^2 + (n+1)^2 + 2(1 + 2 + \ldots + n)(n+1), \text{ using identity showed above} \\ &= (1 + 2 + \ldots (n+1))^2 \end{aligned}$$

3 Correctness of recursive selection sort

Note that induction proofs have a very similar flavour to recursive algorithms. There too, we have a base case, and then the recursive call essentially makes use of "previous cases". for this reason, **induction will be the main technique to prove correctness and time complexity of recursive algorithms**.

Induction proofs for recursive algorithm will generally resemble very closely the algorithm itself. The base case(s) of the proof will correspond to the base case(s) of the algorithm. The induction step will typically assume that the all recursive calls execute correctly, and then prove that the algorithm itself is correct. In other words, you have to **put your faith in the recursive call**.

For the recursive selection sort, the property we want to prove is that for an array of any input size n, at the end of the algorithm, $a[j] \leq a[i], \forall j \leq i, i < n$.
Base case: If $n = 1$, the statement is trivially true.
Induction step: We will look at the statements that hold true after each line of the algorithm.

int $indmax \leftarrow$findMaxIndex(a,n)
After this step, due to the fact that the findMaxIndex call works correctly (as we proved last time), we have: $a[indmax] \geq a[i], \forall i < n$

swap(a,n, $indmax$)
After this step, due to the effect of the swap, we have $a[n] \geq a[i], \forall i < n$

selectionSort(a,$n-1$)
By the induction hypothesis, we assume the recursive call worked. Hence, at this point, $\forall j \leq i, i < n-1, a[j] \leq a[i]$.

Putting the last two statements together gives the desired conclusion.

Note that in this style of analysis, we established repeatedly what **must be true after a particular piece of code**; this is called a **postcondition**. similarly, a logical statement that must be true before executing a certain piece of code is called a **precondition**. The proof of correctness chains these statements together. Note also that in general, the proof by induction technique is somewhat easier to construct, as one does not need to guess at a loop invariant.