Proving Insertion Sort

Prof. Hans Georg Schaathun

13th September 2013

This documents give typed solutions for both the videos proving insertion sort.

1 Take 1. The iterative case

Exercise 1.1 *Prove that the output array of insertion sort (as given in previous videos) is sorted in increasing order.*

To conduct a proof by induction, we need some predicate describing partial success of the algorith, Loop indices provide a reasonable link. We observe that during the iteration $i = 2$ only A_i is moved (and only to the left). Hence elements to the left must already be sorted, and elements to the right are irrelevant. Thus we can propose the following invariant $P(i)$:

$$P(n) := \text{the subarray } A_1, A_2, \ldots, A_n \text{ is sorted at the start of iteration } n+1$$

If $P(n)$ is true, the entire array is sorted after the last iteration, and the algorithm is correct. Therefore we prove $P(n)$ by induction.

Firstly, we note that for $i = 2$, there is a single element to the left, which makes a trivially sorted subarray. Hence $P(1)$ is true.

To prove $P(n) \Rightarrow P(n+1)$, i.e. that the subarray remains sorted after the iteration, we need to consider the old subarray $A_1, \ldots, A_n$, and the new one, $A'_1, \ldots, A'_{n+1}$. It is sufficient to prove that $A'_{j-1} \leq A'_j$ for any j.

Considering an arbitrary j, there are three possibilities:

```
1        for i = 2 to n
2            j = i
3            while (j ≥ 2) and (A_j < A_{j-1})
4                exchange A_j and A_{j-1}
5                j = j-1
```

Table 1: Iterative algorithm.

```
1   procedure InsertionSort(A_1, A_2, ..., A_n)
2   if n = 0, then return []
3   else
4       InsertionSort(A_1, A_2, ..., A_{n-1})
5       insert(A_n into A_1, A_2, ..., A_{n-1})

1   procedure insert(e into A_1, A_2, ..., A_n)
2   if n = 0, then
3       A_1 = e
4   else if e > A_n, then
5       A_{n+1} = e
6   else
7       A_{n+1} = A_n
8       insert(e into A_1, A_2, ..., A_{n-1})
```

Table 2: Recursive algorithm.

1. A'_{j-1} and A'_j correspond to adjacent elements in the original array. Then $A'_{j-1} \leq A'_j$ as required, because the original array was sorted.

2. $A'_j = A_{n+1}$ is the new element. In this case $A'_j \geq A'_{j-1}$, lest the while-condition be true, and the two elements swapped.

3. $A'_{j-1} = A_{n+1}$ is the new element. In this case $A'_j < A'_{j-1}$, since the two elements have been swapped by the while loop.

In all three cases, the sort order is satisfied, and we conclude that the loop maintains sort order. InsertionSort is correct by mathematical induction.

2 Take 2. The recursive case

Exercise 2.1 *Prove that the output array of insertion sort (see Table 2–3) is sorted in increasing order.*

To conduct a proof by induction, we need some predicate describing partial success of the algorith, The a variable should be in the set of natural numbers. In the case of recursion, we can typically link the predicate to the size of the input, as follows:

$$P(n) := \text{Insertion sort can correctly sort } n \text{ numbers}$$

If we can prove $P(n)$ for all n, then the algorithm is correct for any (valid) input.

We note that for $n = 0$ we have an empty array, which is trivially sorted. Hence $P(0)$ is trivially true. It remains to prove that $P(n-1) \Rightarrow P(n)$.

In the recursive case, the algorithm performs two operations. First, an InsertionSort on $n-1$ elements, which is correct by the inductive hypothesis $P(n-1)$; and secondly

an insert into a sorted array of $n-1$ elements. If we can prove that the insert algorithm is correct, then correctness of InsertionSort will follow.

The proof of the insert algorithm is simular

$Q(n) :=$ The insert algorithm correctly inserts an element into an array of n numbers

Again, we note that $Q(0)$ is true, because $n = 0$ gives a singleton array which is trivially sorted. To prove $Q(n-1) \Rightarrow Q(n)$, we note that there are two cases. If e is larger than A_n, it is larger than every element in A because it is sorted. Thus e is correctly inserted as a last element. If e is not larger than A_n, A_n has to be the last element, at position $n+1$, and e is inserted into the subarray of $n-1$ elements. By the inductive hypothesis, $P(n-1)$, e is correctly inserted, and $P(n)$ follows for any non-negative n by mathematical induction.

Since insert is correct, InsertionSort is correct, as argued above.