CIS 575. Introduction to Algorithm Analysis
Material for February 16, 2024

Correctness of Iterative Insertion Sort

The topic of this note is sketched in *Cormen*'s Section 2.1 (p.21).

1 Correctness Proof for Iterative Insertion Sort

In this note, we shall embark on a rather daunting task: proving the correctness of an algorithm which (unlike the examples given so far) involves a *nested* loop. We shall look at the algorithm that we developed in the first week for *iterative insertion sort*:

> **for** $i \leftarrow 2$ **to** n
> $j \leftarrow i$
>   **while** $j > 1$ and $A[j] < A[j-1]$
> $A[j] \leftrightarrow A[j-1]$; $j \leftarrow j - 1$

which is supposed to implement the postcondition:

> $A[1..n]$ is a permutation of its original values such that $A[1..n]$ is non-decreasing.

The first part of the postcondition will obviously be fulfilled; all modifications of A involve two elements being swapped, and hence A will always be a permutation of its initial value. We shall therefore focus on the second part: that $A[1..n]$ is non-decreasing. For that purpose, we shall first address the outer loop; to prove that its body maintains its invariant, we need to address also the inner loop.

1.1 Outer Loop

We propose the invariant

$$1 \leq i \leq n + 1 \text{ with } A[1..i-1] \text{ non-decreasing} \tag{1}$$

and shall now verify (assuming $n \geq 1$) the various items on the checklist:

Establish Our obligations are to show that $1 \leq 2 \leq n+1$ and that $A[1..1]$ is nondecreasing which is obviously true (we could even have let the outer loop start with $i = 1$).

Correctness At loop exit, $i > n$ (as the guard is false) and $i \leq n+1$ (by the loop invariant), implying $i = n + 1$ which together with $A[1..i-1]$ being non-decreasing (by the loop invariant) entails the desired result: $A[1..n]$ is non-decreasing.

Terminate This is trivial as a `for loop` always terminates.

Maintain Since $i \leq n$ when the loop body is executed, we will have $i + 1 \leq n + 1$ which shows that the first part of (1) is maintained. To prove that also the second part is maintained, we must examine the inner loop, as will be done next.

1.2 Inner Loop

We consider the loop

$j \leftarrow i$
while $j > 1$ and $A[j] < A[j-1]$
  $A[j] \leftrightarrow A[j-1]$; $j \leftarrow j - 1$

and must prove that it implements the "local" specification:

Precondition $A[1..i-1]$ is non-decreasing

Postcondition $A[1..i]$ is non-decreasing.

For that purpose, we need to come up with an invariant for the inner loop. We may find inspiration by simulating the algorithm on some input; for example, with $i = 5$ we may have (the value of j is indicated by having $A[j]$ in boldface)

	1	2	3	4	5
start	2	4	7	8	**3**
step 1	2	4	7	**3**	8
step 2	2	4	**3**	7	8
finish	2	**3**	4	7	8

from which we may *conjecture* the invariant

$A[1..j-1]$ and $A[j..i]$ are both non-decreasing (with $1 \leq j \leq i$)

Let us try to verify the checklist for this purported invariant:

Establish The claim is that $A[1..i-1]$ and $A[i..i]$ are both non-decreasing; the first follows from the local precondition, and the second is trivial.

Terminate is obvious as eventually $j \leq 1$ and the loop will exit (it may exit earlier).

Correctness At loop exit, we have either

- $j = 1$, in which case the invariant gives us directly that $A[1..i]$ is non-decreasing
- $j > 1$ with $A[j-1] \leq A[j]$, in which case the two parts of the invariant can be combined to show that $A[1..i]$ is non-decreasing.

Maintain Even though the invariant is indeed always true, we can **not** show that it is maintained! The problem is that it allows for situations that can *not* occur during regular program execution, for example (with $i = 6$)

	1	2	3	4	5	6
start	5	6	7	**2**	3	4
step 1	5	6	**2**	7	3	4

where after step 1 it is no longer the case that $A[j..i]$ is non-decreasing.

It turns out that we need a *stronger* loop invariant, which not just says that $A[j..i]$ is non-decreasing, but also that everything is in its proper place except possibly $A[j]$:

$$1 \leq j \leq i, \text{ and for all } k_1, k_2 \text{ with } 1 \leq k_1 < k_2 \leq i : \text{ if } k_2 \neq j \text{ then } A[k_1] \leq A[k_2] \quad (2)$$

Let us verify the checklist for that loop invariant:

Establish Our obligation is that $1 \leq i \leq i$ which follows from (1), and that if $1 \leq k_1 < k_2 < i$ then $A[k_1] \leq A[k_2]$ which amounts to $A[1..i-1]$ being non-decreasing which follows from the local precondition.

Termination As for our first attempt.

Correctness We must prove that at loop exit, the local postcondition is fulfilled, that is $A[1..i]$ is non-decreasing; this will mostly follow from (2) except we must also prove

$$\text{for all } k \text{ with } 1 \leq k < j : A[k] \leq A[j].$$

But when the loop exits, we have either

- $j = 1$ and the claim holds *vacuously* (as there is no k with $1 \leq k < 1$), or
- $j > 1$ with $A[j-1] \leq A[j]$, in which case we for a given k with $1 \leq k < j$ have $k \leq j-1$ which by (2) implies $A[k] \leq A[j-1]$ and thus the desired $A[k] \leq A[j]$.

Maintain With j' the new value of j, and A' the new value of A, we have

$$j' = j - 1 \text{ and } A'[j] = A[j-1] \text{ and } A'[j-1] = A[j].$$

We must consider k_1, k_2 with $1 \leq k_1 < k_2 \leq i$ and with $k_2 \neq j-1$, so as to show (relying on the assumption that (2) holds before the iteration) that $A'[k_1] \leq A'[k_2]$. We split into 4 cases:

$k_2 < j - 1$: then $A'[k_1] = A[k_1] \leq A[k_2] = A'[k_2]$.

$k_2 = j, k_1 = j - 1$: then the claim follows from the calculation $A'[k_1] = A'[j-1] = A[j] < A[j-1] = A'[j] = A'[k_2]$.

$k_2 = j, k_1 < j - 1$: then the claim follows from the calculation $A'[k_1] = A[k_1] \leq A[j-1] = A'[j] = A'[k_2]$

$k_2 > j$: then $A'[k_1]$ is a member of the set $\{A[k_1], A[j-1], A[j]\}$ from which we infer $A'[k_1] \leq A[k_2] = A'[k_2]$.

1.3 Summary

We have proved the correctness of iterative insertion sort (much as is done for Theorem 2.7 of *Howell*'s textbook). On the other hand, *Cormen* (p.21 in Section 2.1) considers only the outer loop, and does not attempt to phrase an invariant for the inner loop (since they, perhaps wisely, *prefer not to get bogged down in such formalism*).