Proving Correctness with Invariants

Today in class we talked about how to prove that our insertion sort algorithm is correct using invariants. This document provides a formally written proof of the reasoning we discussed, including the inner while loop.

Loop-1

1. Initialization. By the precondition, `to_sort` contains a multiset of ints. None of the code between the precondition and the first loop-1 iteration touches `to_sort`, so `to_sort` contains all of the original data at the start of the first iteration, satisfying invariant (1). Just before the start of the loop, we set `i=1`, so subarray `to_sort[0...i-1]` consists of just element 0 (i.e., a single element), so it must be sorted, satisfying invariant (2).

2. Maintenance. We want to show that if our invariant is true and the loop condition (i.e, `i<n`) is true, then after running the code in the body of the loop, the invariant remains true. By arguments that we will make below regarding loop-2, we will see that the loop-2 postcondition holds. This says that if `val` were stored in `to_sort[j+1]` then the subarray `to_sort[0...i]` would be sorted and would contain all of the original data. On the next line of code, we store `val` in `to_sort[j+1]`, so the array contains all of the original data and subarray `to_sort[0...i]` is sorted. We then increment `i`, so subarray `to_sort[0...i-1]` is sorted and contains all of the original data. These are exactly our loop-1 invariants.

3. Termination. The loop terminates when `i=n`. The first loop-1 invariant tells us that `to_sort` contains all of the original data, satisfying postcondition (1). The second loop-2 invariant tells us that `\to_sort[0...i-1]` is sorted. But when `i=n`, this subarray is `to_sort[0...n-1]`, which is the entire array. Hence the second postcondition also is satisfied. Note that if we never enter the loop, then we reach the termination state because `n<i=1`. In this case our array consists of 0 or 1 element, which must be sorted. In this case our postconditions also hold.

Loop-2

We will use the loop-1 invariants as preconditions for loop-2.

1. Initialization. The first loop-1 invariant tells us that `to_sort` contains all of the original data. In the following two lines of code, we remove `to_sort[i]` and set `j=i-1`, so we have removed the element at position `j+1`. The first loop-2 invariant follows. Since `j=i-1`, ignoring `to_sort[j+1]` is the same as ignoring `to_sort[i]`. Ignoring `to_sort[i]`, the subarray `to_sort[0...i]` is the same as subarray `to_sort[0...i-1]`, which we know from the second loop-1 invariant is sorted. Hence the second loop-2 invariant holds. At the start of loop-2, `j=i-1`, so `j+2=i+1`. This means that the subarray `to_sort[j+2...i]` is equivalent to subarray `to_sort[i+1...i]`, which is empty, so the third loop-2 invariant holds vacuously.

2. Maintenance. Invariants 1 and 2 follow straightforwardly from the assertions after each line of code in the loop body. All we are doing here is shifting data and shifting the corresponding array indices in our assertions. Invariant 3 is a bit trickier. In the first line of the body of the loop, we shift the element at position `j` to position `j+1`. Since our loop condition checks that `val<to_sort[j]`, we know that the element that we moved into position `j+1` is greater than `val`. Our third invariant says that all of the data in subarray `to_sort[j+1...i]` is greater than `val`, and we just put an element greater than `val` into position `j+1`, so we can now assert that all of the data in subarray `to_sort[j+1...i]` is greater than `val`. Finally we decrement `j`, bringing us back to our third invariant as stated: the data in subarray `to_sort[j+2...i]` is greater than `val`.

3. Termination. The postcondition states that if `val` were stored in `to_sort[i]`, the subarray `to_sort[0...i]` would be sorted and would contain all of the original data. That the array would contain all of the original data follows immediately from the first loop-2 invariant. The fact that the loop terminated means that we are in one of two cases. If the loop terminated because `j=-1<0`, then we know that the subarray `to_sort[1...i]` is sorted (by invariant 2) and all elements in this subarray are greater than `val` (by invariant 3), hence after inserting `val` into position `j+1=0` the subaray `to_sort[0...i]` is sorted. If the loop terminated because `val>=to_sort[j]`, then (a) all data in subarray `to_sort[j+2...i]` is greater than `val` (by invariant 3), (b) all data in subarray `to_sort[0...j]` is smaller than `val` (by the fact that `val>=to_sort[j]`), and (c) the data in each of subarrays `to_sort[0...j]` and `to_sort[j+2...i]` is sorted (by invariant 2). Hence after inserting `val` into position `j+1`, the whole subarray `to_sort[0...i]` is sorted.