15-122: Principles of Imperative Computation

Recitation Week 3 — Rob Simmons

In class, we covered one quadratic sort, *selection sort*, and two $O(n \log n)$ sorts, *quicksort* and *mergesort*. To practice with thinking about sorts in terms of loop invariants, today we'll look at two other quadratic sorts, *insertion sort* and *bubble sort*.

Insertion Sort

```
1  void sort(int[] A, int n)
2  //@requires 0 <= n && n <= \length(A);
3  //@ensures is_sorted(A, 0, n);
4  {
5     for (int i = 0; i < n; i++)
6     //@loop_invariant 0 <= i && i <= n;
7     //@loop_invariant is_sorted(A, 0, i);
8     {
9        int j = i;
10
11       while (_____________________________________________________)
12
13       //@loop_invariant ____________________________________________;
14       //@loop_invariant is_sorted(A, 0, j);
15       //@loop_invariant is_sorted(A, j, i+1);
16       //@loop_invariant le_segs(A, 0, j, A, j+1, i+1);
17       {
18          swap(A, _________________________, _________________________);
19          j--;
20       }
21    }
22 }
```

(a) Assuming inner loop invariants are correct, prove the outer ones are preserved.

(b) What goes wrong if we're missing just the loop invariant on line 13? 14? 15? 16?

Bubble Sort

Different versions of bubble sort use one or both of two different tricks; one trick allows sorting to end early if the array is mostly sorted, and another trick avoids avoids re-scanning the upper part of the array, which we know to be sorted. The next page has two bubble sorts, each uses one of the tricks.

(a) Loop invariants used for safety are on lines 12 (first example) and 7 and 15 (second example). Make sure you justify the safety of the array access in line 16 of the second example!

(b) On line 16 in the first example, `ge_seg(A[j], A, 0, j)` is almost right. What's wrong with it?

(c) What goes wrong in each example without the invariants on line 16?

(d) What goes wrong if one of the loop invariants on lines 9 and 11 of the second example are missing?

```
void sort(int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@ensures is_sorted(A, 0, n);
{
   bool done = false;
   while (!done)
   //@loop_invariant !done || ____________________________________;
   {
      bool swapped = false;
      for (int j = 0; j < n-1; j++)

      //@loop_invariant ____________________________________________;

      //@loop_invariant swapped || is_sorted(A, 0, j);

      //@loop_invariant ____________________________________________;
      {
         if (A[j] > A[j+1]) {
            swapped = true;
            swap(A, j, j+1);
         }
      }
      if (!swapped) done = true;
   }
}
```

```
void sort(int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@ensures is_sorted(A, 0, n);
{
   for (int upper = n; upper > 0; upper--)

   //@loop_invariant ____________________________________________;

   //@loop_invariant ____________________________________________;

   //@loop_invariant ____________________________________________;
   {
      for (int j = 0; j < upper-1; j++)

      //@loop_invariant ____________________________________________;
      //@loop_invariant ge_seg(A[j], A, 0, j);
      {
         if (A[j] > A[j+1]) {
            swap(A, j, j+1);
         }
      }
   }
}
```