15-122: Principles of Imperative Computation

Recitation 5 Solutions — Josh Zimmerman

Basic linear search: recap

(Note: I assume the `is_in` and `is_sorted` functions exist as defined in class.)

```
1 int lin_search(int x, int[] A, int n)
2 //@requires 0 <= n && n <= \length(A);
3 //@requires is_sorted(A, 0, n);
4 /*@ensures (-1 == \result && !is_in(x, A, 0, n))
5       || ((0 <= \result && \result < n) && A[\result] == x);
6   @*/
7 {
8    for (int i = 0; i < n; i++)
9    //@loop_invariant 0 <= i && i <= n;
10   //@loop_invariant !is_in(x, A, 0, i);
11   {
12      if (A[i] == x) {
13         return i; // We found what we were looking for!
14      }
15      else if (x < A[i]) {
16         return -1; // We've passed the last point it could be, so it's not there
17      }
18      //@assert A[i] < x;
19   }
20   return -1;
21 }
```

Now, let's look at this code and see if we can prove that it works. Work on your own or with other people to follow the four-step process to proving that linear search works. (Remember: Show that the loop invariants hold initially, that they are preserved, that the loop invariants and the negation of the loop condition imply the postcondition, and that the loop terminates.)

Solution:

Loop invariants hold initially

Loop invariant 1: we initialize `i` to 0, so `0 <= i`. By the precondition, `0 <= n`, so `i <= n` initially as well.

Loop invariant 2: we initialize `i` to 0, so we're checking to see if anything is in an empty chunk of the array. Nothing is, since it's empty, so the loop invariant holds.

Preservation of loop invariants

Loop invariant 1: By the loop exit condition, `i < n` when we start the iteration, so when we exit the iteration, `i + 1 == i' <= n`. Further, `i' > i >= 0`, so `i' >= 0` (since `i' <= n`, we know there wasn't overflow)

Loop invariant 2: By the loop invariant, $x \notin A[0 \ldots i)$. If `A[i] == x`, we would have exited the loop on line 13. Thus, `A[i] != x` after we finish this iteration of the loop, so $x \notin A[0, i+1)$. Since `i' == i`

`+ 1`, we know that $x \notin A[0, i')$.

Loop invariants imply postcondition

There are several cases in which we can return. We need to address all of them.

Case 1: We return on line 13. In this case, we return a value which by the loop invariant is between 0 and `n`. Further, we know that `A[i] == n` by the condition on line 12.

Thus, the second clause of the postcondition is satisfied, and so the postcondition is satisfied.

Case 2: We return on line 16. We know that `\result == -1`, so we want to show `!is_in(x, A, 0, n)`. We know by the loop invariant that `!is_in(x, A, 0, i)`. Further, we know that `A[i] > x`, and that `A` is sorted. Since `A` is sorted, we know that everything in the segment $A[i, n)$ is also greater than `x`. Thus, `x` is not in the array. We returned -1, so the first clause of the postcondition is satisfied.

Case 3: We return on line 20. In this case, we know we've exited the loop, so `i >= n` by the negation of the loop guard and `i <= n` by the loop invariant. Thus, `i == n`.

So, `!is_in(x, A, 0, i)`, which is equivalent to `!is_in(x, A, 0, n)`. Further, we return -1, so the first clause of the postcondition is satisfied.

Termination

The loop starts with `i` being nonnegative. We increment `i` once per iteration of the loop and terminate once `i >= n`, which must happen eventually since `0 <= n`.

I claim we can search a sorted array faster than this. We'll discuss why in lecture tomorrow, but for now try to think about how you could improve on this search method.