Here is the code from lecture. Let’s look at exactly how it works. Can we use the loop invariants to prove the postcondition?

```c
int binsearch(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A,n);
/*@ensures (-1 == \result && !is_in(x, A, n))
|| ((0 <= \result && \result < n) && A[\result] == x); @*/
{
    int lower = 0;
    int upper = n;
    while (lower < upper)
    //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
    //@loop_invariant (lower == 0 || A[lower-1] < x);
    //@loop_invariant (upper == n || A[upper] > x);
    {
        int mid = lower + (upper-lower)/2;
        //@assert lower <= mid && mid < upper;
        if (A[mid] == x) return mid;
        else if (A[mid] < x) lower = mid+1;
        else /*@assert(A[mid] > x);@*/ upper = mid;
    }
    return -1;
}
```

First, we prove that they’re true before entering the loop:

- **WTS (want to show):** $0 <= lower && lower <= upper && upper <= n$
  Since $lower = 0$ and $upper = n$, we have $0 <= 0 && 0 <= n && n <= n$  ✓

- **WTS: (lower == 0 || A[lower-1] < x)**
  This is true, since $lower = 0$ ✓

- **WTS: (upper == n || A[upper] > x)**
  This is also true, since $upper = n$ ✓

Next we prove that if they’re true at the start of a loop iteration, then they’re true at the end:

- Assume $0 <= lower && lower <= upper && upper <= n$
  **WTS (want to show):** $0 <= lower' && lower' <= upper' && upper' <= n$
  If we return from the loop, we haven’t changed $lower$ or $upper$, so the invariant still holds.
  Suppose we don’t return. Then we case on what happens next:
    - Case $A[mid] < x$:
      Then $lower' = mid + 1 = lower + (upper - lower)/2 + 1 > 0$.
      Also, since $lower < upper$ by the loop condition, then
      $lower + (upper - lower)/2 < lower + (upper - lower) = upper$, so
      $lower + (upper - lower)/2 + 1 <= upper$, and $lower' <= upper$.
      And since $upper' = upper$, then the invariant holds. ✓
Case $A[mid] \geq x$:
Since we didn’t return from the function, we actually know $A[mid] > x$.
Then $upper = mid = lower + (upper - lower)/2$.
Since $lower \leq upper$, then $0 \leq upper - lower$, and $0 \leq (upper - lower)/2$.
So $upper = mid \geq lower$. Also, since $lower' = lower$, the invariant holds. ✓

• Assume $(lower == 0 || A[lower-1] < x)$
WTS: $(lower' == 0 || A[lower'-1] < x)$
If we return from the loop, $lower$ hasn’t changed, so the invariant holds.
Suppose we have not returned from the loop.

– Case $A[mid] < x$:
  By our assumption, we know $A[mid] < x$, so the invariant holds. ✓

– Case $A[mid] \geq x$:
  Like before, since we didn’t return from the function, we actually know $A[mid] > x$.
  Then $lower' = lower$, so the loop invariant still holds, by our assumption. ✓

Now we prove that if the loop invariants are true the function returns then the post-condition is true:
WTS: $(-1 == \result && !is_in(x, A, n)) || ((0 <= \result && \result < n) && A[\result] == x)$

• Case that we returned from line 16:
Then $\result = mid$. Since $mid = lower + (upper - lower)/2$, and we’ve shown that $0 \leq (upper - lower)/2$ and $lower + (upper - lower)/2 \leq upper$, we know that $lower \leq \result && \result \leq upper$.
By the first loop invariant, $0 \leq lower && lower \leq upper && upper \leq n$.
Then $0 \leq \result && \result \leq n$.
We returned $mid$ because $A[mid] == x$, so $A[\result] == x$, and thus the postcondition is true. ✓

• Case that we returned from line 20:
In this case, we return -1. We want to show now that $!is_in(x, A, n)$. Then we know that the negation of the loop condition is true and $lower \geq upper$. Also because of the first loop invariant, since $lower <= upper$, we know that $lower == upper$.

– Case $lower == 0$:
  Then we know $upper = lower = 0$, and, by the third loop invariant, $A[upper] > x$.
  Since our precondition tells us that A is sorted, then we know that even the smallest element in the first $n$ elements of A is larger than x, so x can’t possibly be among them.
  Then $\result == -1 && !is_in(x, A, n)$ is true. ✓

– Case $upper == n$:
  Then we know $lower = upper = n$, and, by the second loop invariant, $A[lower - 1] < x$.
  Since A is sorted, we know the largest of the first $n$ elements of A is smaller than x, so x is not among them.
  Then $\result == -1 && !is_in(x, A, n)$ is true. ✓
Case $lower > 0 \&\& upper < n$:
Then we know that $0 < lower = upper < n$, so the loop terminated with $lower$ and $upper$ somewhere in the middle of $A$.
We also know that $A[lower-1] < x$ by the second loop invariant, and $A[upper] > x$ by the third loop invariant.
Then since the preceding elements are all less than $x$ and the subsequent elements are greater than $x$, and we haven’t found $x$, then we know that $x$ is not in $A$.
Then $\result == -1 \&\& !is\_in(x, A, n)$ is true. ✓

We now prove that the loop eventually terminates:
Going through the loop body, we either return after finding the $x$, or we end up shrinking our interval, since $lower$ to $mid$ and $mid+1$ to $upper$ are strictly smaller than the interval $lower$ to $upper$. Eventually, then, the interval size will eventually become 0, and the loop will exit. ✓.

A Variation of Binary Search

Here’s another way to implement binary search. As an exercise, try proving the postcondition like the above version.

```c
1 int binsearch2(int x, int[] A, int n)
2 //@requires 0 <= n \&\& n <= \length(A);
3 //@requires is_sorted(A, n);
4 /*@ensures ((\result == -1) \&\& !is\_in(x, A, n)) ||
5 ((0 <= \result \&\& \result < n) \&\& A[\result] == x); @*/
6 {
7   int lower = 0;
8   int upper = n-1;
9   while (lower <= upper)
10     //@loop\_invariant 0 <= lower;
11     //@loop\_invariant upper < n;
12     //@loop\_invariant (upper - lower) >= -1;
13     //@loop\_invariant (lower == 0) || (A[lower - 1] < x);
14     //@loop\_invariant (upper == n - 1) || (A[upper + 1] > x);
15     {
16       int mid = lower + ((upper - lower) / 2);
17       if (A[mid] == x) return mid;
18       else if (A[mid] < x) lower = mid + 1;
19       else /*@assert (A[mid] > x) @*/ upper = mid - 1;
20     }
21   return -1;
22 }
```