```
/**
adding a ? to a: array?<int> means that a can be null.
 */
method {:verify false} searchArray(a: array<int>, key: int) returns (found: bool, i: nat)
// requires a!=null such a precondition is no longer needed in our version of Dagny
ensures key in a[..] <==> found //a[..] is a as a sequence of integers. operator "in" is only on sequence
ensures found ==> i < a.Length && a[i] == key
{
    i := a.Length;
    if a.Length == 0
    {
        found := false; //found and i are originally initialized with "garbage" value. because we didn't specify i's value if found is false
        // so we don't need to initialize i.
    }
    else
    {
        i,found := a.Length, false;
        while Guard(a,key,found,i)
        invariant Inv(a[..],key,found,i)
        decreases i
        {
            ProofOfLoopBody(a,key,found,i);
            i,found := i -1,UpdateFound(a,key,found,i);
        }
        StrengthenPostcondition(a[..],key,foumd.i);
    }
}
predicate method UpdateFound (a: array<int>,key: int, found: bool, i: nat)
{


}


lemma {:verify true} StrengthenPostcondition (a: array<int>, key: int, found: bool, i: nat)
    requires Inv(a[..],key,found,i) //sometimes order of preconds is important
    requires !Guard(a,key,found,i)
    ensures key in a[..] <==> found
    ensures found ==> i < a.Length & a[i] == key
    {
        //all this is probably not really needed, it was for our own convicing of the design - guard/inv.
        assert key in a[..] <==> found by {
            assert key in a[i..] <==> found; // from invraiant
            assert found || i == 0; // the negation of the guard
            if i == 0
            {
                assert a[i..] == a[0..] == a[..];
                assert key in a[..] <==> key in a[i..];
            }
            else
            {
                assert found;
                assert key in a[..] by {
                    assert key in a[i..]; //from Inv (when found is true)
                    assert forall k :: k in a[i..] ==> k in a[i..];
                }

            }
        }
        assert found ==> i < a.Length && a[i] == key by {
            if found
            {
                assert i < a.Length && a[i] == key; //from Inv when found is true
            }
            else
            {
                assert false ==> i < a.Length && a[i] == key; // false implies everything
            }

        }
    }

predicate method Guard(a: array<int>, key: int, found: bool, i: nat)
{
    // !found && i>=0 this guard is problematic since if the key isn't in the array, we then do i = i-1 in the loop body and this is a type error
    //since i is a natural number. so the guard isn't sufficient.
    !found && i > 0
}
// not a method since it's called in specification (invriant of the loop).

predicate Inv(q: seq<int>, key: int, found: bool, i: nat)
{
    |q| > 0 && i <= |q| &&
    (key in q[i..] <==> found) && (found ==> i < |q| && q[i] == key)

}
```