1: Contracts

(The following question is Question 1 on Written Homework 2)

Consider the following implementation of the linear search algorithm that finds the first occurrence of $x$ in array $A$:

```c
int find(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
//@answer to 1.b goes here
{
    int i = 0;
    while (i < n && A[i] <= x)
        //answer to 1.a goes here
        {
            if (A[i] == x) return i;
            i = i + 1;
        }
    return -1;
}
```

The function `is_sorted` has the following signature:

```c
bool is_sorted(int[] A, int lower, int upper)
//@requires 0 <= lower && lower <= \length(A) - 1;
//@requires 0 <= upper && upper <= \length(A);
//@requires lower <= upper;
```

and returns true if the array $A$ is sorted in increasing order from $[lower, upper)$.

You may also use the function `is_in` in any of the following questions. Its signature is:

```c
bool is_in(int[] A, int x, int n)
//@requires 0 <= n && n <= \length(A);
```

and it returns true if $A[i] == x$ for some $i$ in $[0, n)$.  


Add loop invariants to the while loop in the code and show that they hold for this loop. Be sure that the loop invariants precisely describe the computation in the loop.

What we were looking for is:

//@loop_invariant 0 <= i && i <= n;
//@loop_invariant !is_in(A, x, i);
OR
//@loop_invariant (i == 0) || (A[i-1] < x);

It’s important to get these correct at this stage because you have to prove that your loop invariant implies the postcondition and if your loop invariant is incorrect or not strong, that part will go wrong.

Add one or more ensures clause(s) to describe the intended postcondition in a precise manner.

/*@ensures (result == -1 && !is_in(A, x, n)) ||
(0 <= result && result < n &&
 A[result] == x && !is_in(A, x, result)); */

OR

/*@ensures (result == -1 && !is_in(A, x, n)) ||
(0 <= result && result < n &&
 A[result] == x && ((result == 0) || (A[result - 1] < x))); */

Both these forms are equivalent. Take a moment to reason why the are equivalent. Even though you’re expected to write just one if these, it’s crucial to know other ways to say the same thing.

The common mistake here was missing out the fact that you were asked to return the first occurrence. Remember to read the question carefully!

Also, whenever you have cases in the postcondition, remember to be as specific as possible. A good way to think about it is independently generate the strongest postcondition for each of the possible cases and then combine them together using the || operator. (WARNING: While using the || operator, remember to take advantage of short-circuiting to prevent errors like array access out of bounds. After writing your post-condition, try it out with a sample output each of both cases, to ensure that it’s saying what you intend it to)
Show that the loop invariant is *strong enough* by using the loop invariant to prove that the postconditions hold when the function returns. You do not need to prove the loop invariant’s correctness (you already did that in 1.a); that is, you assume it is correct for this problem. Make sure to deal with the fact that `find` can return in two different ways.

Consider the loop invariant

```markdown
//@loop_invariant 0 <= i && i <= n;
//@loop_invariant !is_in(A, x, i);
```

and the postcondition

```markdown
/*@ensures (\result == -1 && !is_in(A, x, n)) ||
(0 <= \result && \result < n &&
A[\result] == x && !is_in(A, x, \result)); @*/
```

So we’ll have two cases here:

**When we return `i` from within the loop:**
Then, by the loop invariant, we have `0 <= \result && \result <= n && !is_in(A, x, \result)`
Also, from the loop body, we know that we return when `A[i] == x`, so `A[\result] == x`
Thus the second clause in the postcondition,

```markdown
(0 <= \result && \result < n && A[\result] == x && !is_in(A, x, \result))
```

holds true and since the clauses in the postcondition are connected by an OR, the postcondition holds true

**When we return `-1` after the loop:**
Now, we know that the loop condition is false but the invariant is true
So what we know is `(i >= n) || ((i < n) && A[i] > x)`
In either case, since the array is sorted and we know that `!is_in(A, x, i)`, we can conclude that `!is_in(A, x, n)`
(I’m not justifying this here, but it would be a good idea to)

The important part here is proving the *entire* postcondition true
The common mistake originated from the earlier parts. If your loop-invariant and postcondition are not strong enough, your proof won’t be what we’re looking for
2: Big O Analysis

You are given the following expression -

$$f(n) = 3n^2 + 4n + \log n + 4$$

Give a tight Big O bound on the asymptotic complexity of this expression

$$f(n) \in O(n^2)$$

Using your answer from the previous part, prove that $$f(n) \in O(g(n))$$ using the formal definition of Big O

You should know the formal definition of Big O. it’s really important and we may not give it to you in the exam. If you have trouble memorizing it, be sure to put it on your cheat sheet!

Anyhow, here’s the definition

If $$f(n) \in O(g(n))$$ then, $$\exists$$ constants $$c > 0$$ and $$n_0 \geq 0$$ such that

$$\forall n \geq n_0, f(n) \leq cg(n)$$

So, what is this question asking you to do?

Basically you need to find suitable values of $$c$$ and $$n_0$$ for which $$f(n) \leq c(n^2)$$, whenever $$n \geq n_0$$

You need to come up with this $$c$$ and $$n_0$$ by observation (or any means that you may like)

All that is expected is that you state your values for $$c$$ and $$n_0$$ and then show that they hold

Here, lets take $$c = 8$$ and $$n_0 = 2$$

So we must show that $$f(n) \leq 8(n^2), \forall n \geq 2$$

Base Case: $$n = 2$$

$$f(2) = 3 \times 4 + 4 \times 2 + \log 2 + 4 = 25 < 8 \times 4$$

Induction Step:

Assume that $$f(k) < 8k^2$$, for some $$k \geq 2$$

To show $$f(k + 1)$$:

$$f(k + 1) = 3 \times (k + 1)^2 + 4(k + 1) + \log (k + 1) + 4$$

$$\Rightarrow f(k + 1) = 3k^2 + 6k + 3 + 4k + 4 + \log (k + 1) + 4$$

$$\Rightarrow f(k + 1) = (3k^2 + 4k + \log (k + 1) + 4) + 6k + 7$$

$$\log (k + 1) \leq 2 \log k$$

$$\Rightarrow f(k + 1) \leq (3k^2 + 4k + \log (k + 1) + 4) + 6k + 7 + \log (k)$$

$$\Rightarrow f(k + 1) \leq 8k^2 + 6k + 7 + k$$

Add $$(9k + 1)$$ on the RHS, since $$k > 2$$

$$\Rightarrow f(k + 1) \leq 8k^2 + 16k + 8$$

$$\Rightarrow f(k + 1) \leq 8(k + 1)^2$$
Complete the following code for a function that takes in a queue and returns a reversed queue. Assume that you have all the functions in the interface of stacks and queues.

```c
queue reverse(queue Q)
//@requires __________________;
//@ensures __________________;
{
    _____________________; (HINT: allocate a temporary data structure)
    while (________________)
        //@loop_invariant ______________;
        //@loop_invariant ______________;
        {
        
        }
    _____________________; (HINT: create a new queue)
    while (________________)
        //@loop_invariant ______________;
        //@loop_invariant ______________;
        {
        
        }
    _____________________; (HINT: look at the function prototype)
}
```

The basic idea here is to store the elements on the stack and then add them on to a new queue. Then we return the newly made queue. Remember that in such cases, you are usually expected to create a new queue and not modify the original queue.

The contracts involved are checking that the data structures are valid (is_stack(S) and is_queue(Q)). This is important, especially in the loop-invariants as it is completely possible that certain statements within the loop body modify the data structure in a way that it is no longer valid.