15-122: Principles of Imperative Computation, Spring 2020

Written Homework 3

Due on Gradescope: Monday 3rd February, 2020 by 9pm

Name: ______________________________

Andrew ID: ______________________________

Section: ______________________________

This written homework covers specifying and implementing search in an array and how to reason with contracts. You will use some of the functions from the `arrayutil.c0` library that was discussed in lecture in this assignment.

Preparing your Submission You can prepare your submission with any PDF editor that you like. Here are a few that prior-semester students recommended:

- ***PDFescape*** or ***DocHub***, two web-based PDF editors that work from anywhere.
- ***Preview***, the Mac's PDF viewer.
- ***Acrobat Pro***, installed on all non-CS cluster machines, works on many platforms.
- ***iAnnotate*** works on any iOS and Android mobile device.

There are many more — use whatever works best for you. If you'd rather not edit a PDF, you can always print this homework, write your answers *neatly* by hand, and scan it into a PDF file — *we don't recommend this option, though*.

Submitting your Work Once you are done, submit this assignment on Gradescope. ***Always check it was correctly uploaded.*** You have unlimited submissions.

Question:	1	2	3	4	Total
Points:	3.5	3.5	3	2	12
Score:					

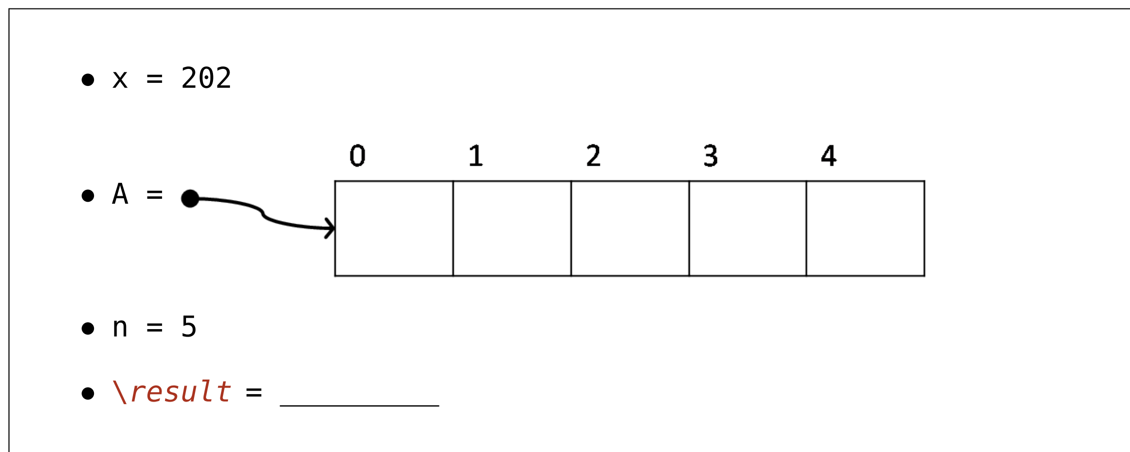
1. **Debugging Preconditions and Postconditions**

Here is an initial, buggy specification of the function `find` that returns the index of the first occurrence of an element `x` in an array `A`. You should assume the `find` function does not modify the contents of the array `A` in any way.

```
1 int find(int x, int[] A, int n)
2 //@requires 0 <= n && n <= \length(A);
3 // (nothing to see here)
4 /*@ensures (\result == -1 && !is_in(x, A, 0, n))
5         || (0 <= \result && \result < n
6             && A[\result] == x
7             && A[\result-1] < x); @*/
```

0.5pts

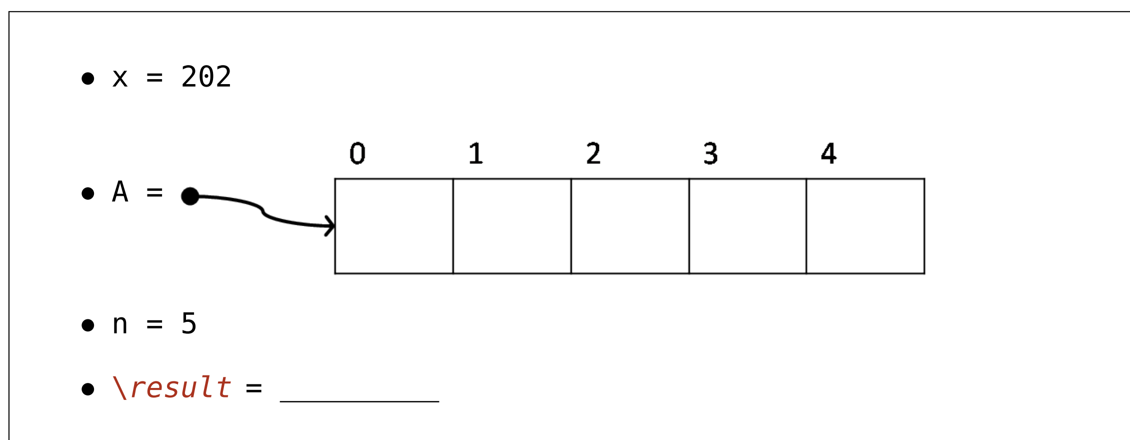
1.1 Give values of `A` and `\result` below, such that the precondition evaluates to `true` and checking the postcondition will cause an array-out-of-bounds exception.



0.5pts

1.2 Notice that the postcondition seems to be relying on `A` being sorted, although the precondition does not specify this. It might be possible, then, that unsorted input will reveal additional bugs in our initial specification.

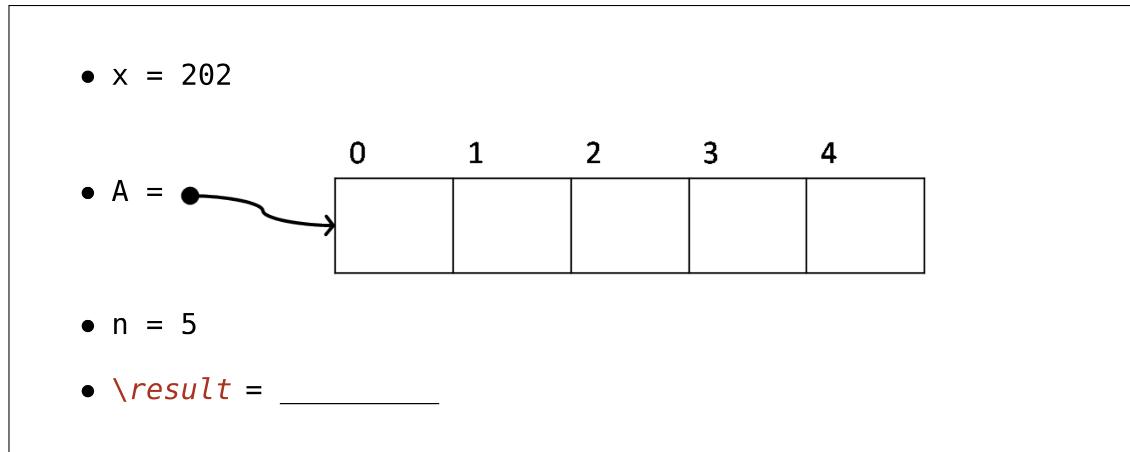
Give values for `A` and `\result` below, such that `\result != -1`, the precondition and the postcondition both evaluate to `true`, and `\result` is *not* the index of the first occurrence of `x` in the array.



© Carnegie Mellon University 2020

0.5pts

1.3 Give values for `A` and `\result` below, such that the precondition evaluates to `true`, the postcondition evaluates to `false`, and `\result` *is* the index of the first occurrence of `x` in the array.



1pt

1.4 Edit line 7 slightly so that, if we added an additional precondition

```
//@requires is_sorted(A, 0, n);
```

the postcondition for `find` would be safe and it would correctly enforce that `A[\result]` is the first occurrence of `x` in `A`. Do *not* use any of the `arrayutil.c0` specification functions.

The addition you make to the postcondition should run in constant time ($O(1)$). (We don't usually care about the complexity of our contracts, of course, but this limits what kinds of answers you can give. In the future, unless we specifically say otherwise, you can assume that the efficiency of contracts doesn't matter.)

7 ____;

1pt

1.5 Edit line 7 so that *whether or not* we require that the array is sorted, the postcondition for `find` is safe and correct. Make the answer as simple as possible. You'll need to use one of the `arrayutil.c0` specification functions.

7 ____;

2. **The Loop Invariant**

   Now we will consider a buggy implementation with a correct specification.

```
1  int find(int x, int[] A, int n)
2  //@requires 0 <= n && n <= \length(A);
3  //@requires is_sorted(A, 0, n);
4  /*@ensures (\result == -1 && !is_in(x, A, 0, n))
5          || (0 <= \result && \result < n
6              && A[\result] == x
7              /* YOUR ANSWER FOR PART (4) OF THE PREVIOUS QUESTION*/); @*/
8  {
9    int lo = 0;
10   int hi = n;
11   while (lo < hi)
12   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
13   //@loop_invariant gt_seg(x, A, 0, lo);
14   //@loop_invariant le_seg(x, A, hi, n);
15   {
       ...
21   }
22   //@assert lo == hi;
23   return -1;
24 }
```

   You should assume that the missing loop body does not write to the array `A` or modify the local variables `x`, `A`, or `n`, but that it might modify `lo` or `hi`.

0.5pts

2.1 In one sentence, explain why `gt_seg(x, A, 0, 0)` and `le_seg(x, A, n, n)` are always `true`, assuming `0 <= n && n <= \length(A)`. Your answer should involve the size of the array segment being tested.

1pt

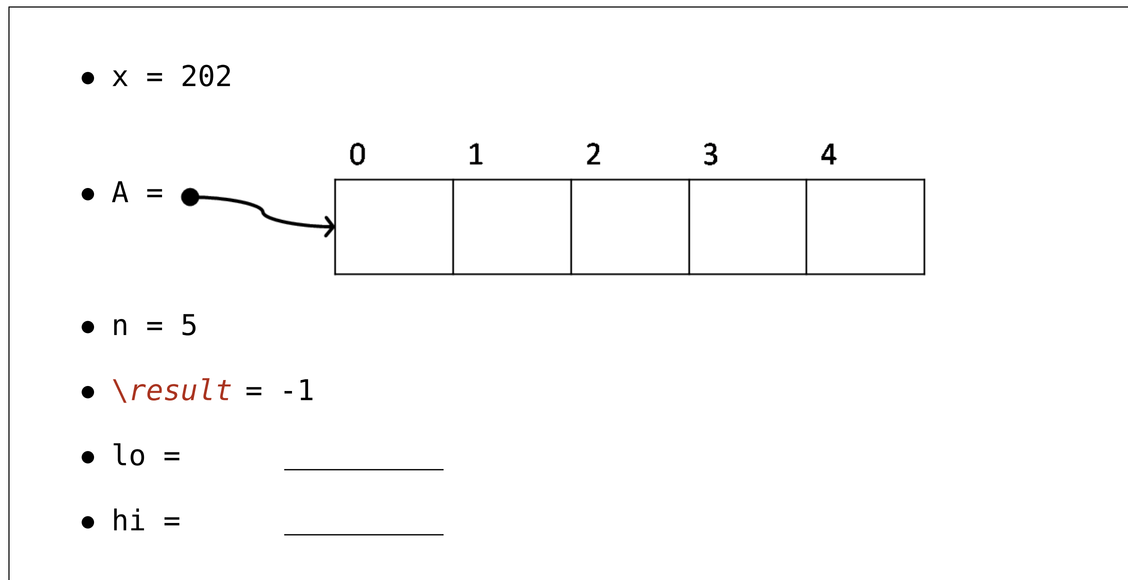
2.2 Prove that the loop invariants (lines 12–14) hold initially.

`0 <= lo`	is true because of line(s) ________
`lo <= hi`	is true because of line(s) ________
`hi <= n`	is true because of line(s) ________
`gt_seg(x, A, 0, lo)`	is true because of line(s) ________
`le_seg(x, A, hi, n)`	is true because of line(s) ________

Take for granted that all the loop invariants are known to be safe. You do need line `n <= \length(A)` from line 2 to reason that the last loop invariant involving `le_seg` is safe (that it satisfies its preconditions). You don't need to include line 2 in your proof that `le_seg(x,A,hi,n)` always evaluates to `true`.

0.5pts

2.3 Danger! These loop invariants do not imply the postcondition when the function exits on line 23. Give specific values for `A`, `lo`, and `hi` such that the precondition evaluates to `true`, the loop guard evaluates to `false`, the loop invariants evaluate to `true`, and the postcondition evaluates to `false`, given that `\result == -1`.



© Carnegie Mellon University 2020

1.5pts

2.4 Modify the code *after* the loop so that, if the loop terminates, the postcondition will always be `true`. The conditional and the return statement should both run in constant time ($O(1)$) and should not use `arrayutil.c0` specification functions.

Take care to ensure that any array access you make is safe! You know that the loop invariants on lines 12–14 are true, and you know that the loop guard is false (which, together with the first loop invariant on line 12, justifies the assertion `lo == hi`).

```
22  /* Loop ends here... */
23  //@assert lo == hi;

25  if (______________________________________________)

27      return ______________________________________;

29  return -1;      // old line 23
30 }                // old line 24
```

3. **Code Revisions**

Here is a loop body that performs linear search. You can use it as an implementation for lines 15–21 on page 3:

```
15  {
16    if (A[lo] == x)
17      return lo;
18    if (A[lo] > x)
19      return -1;
20    lo = lo + 1;
21  }
22  //@assert lo == hi;
```

1pt

3.1 For the loop invariants to hold for this loop body, they must be preserved through each iteration. Prove that the invariant on line 12 on page 3 is preserved by this loop body — you may not need all the provided lines.

A ______________________________ assumption

B ______________________________ by ______________

C ______________________________ by ______________

D ______________________________ by ______________

E ______________________________ by ______________

Therefore we conclude that

______________________________ by ______________

© Carnegie Mellon University 2020

1pt

3.2 Prove that the invariant in line 13 is preserved by this loop body — you may not need all the provided lines. (The proof for line 14 is not required for this answer.)

A	______________________	assumption
B	______________________	by ____________
C	______________________	by ____________
D	______________________	by ____________
E	______________________	by ____________
F	______________________	by ____________

Therefore we conclude that

______________________ by ____________

0.5pts

3.3 You might have noticed in the previous part that `hi` does not actually change during the loop, even though all our reasoning assumes it might. So now, complete this simpler loop invariant for the modified code by writing a line that tells you something about `hi`. The resulting loop invariant should be simple, should be true initially, should be preserved by any iteration of the loop, and should allow you to prove the postcondition *without* the modifications you made in part 4 of the previous question. (You don't have to write the proof.)

```
12  //@loop_invariant 0 <= lo && lo <= hi;
13  //@loop_invariant gt_seg(x, A, 0, lo);
14  //@loop_invariant ______________________________;
```

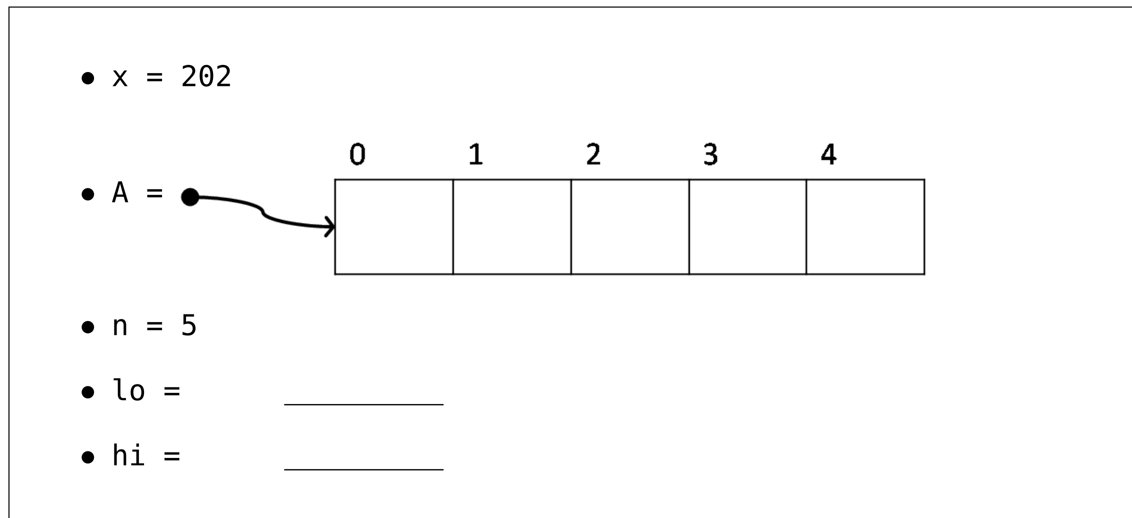
0.5pts

3.4 Here's an alternate loop body that does perform binary search. You can use it as a replacement for code at the beginning of this question.

```
15  {
16    int mid = lo + (hi-lo)/2;
17    if (A[lo] == x) return lo;
18    if (A[mid] < x) lo = mid+1;
19    else { /*@assert(A[mid] >= x); @*/
20      hi = mid;
21    }
22  }
```

© Carnegie Mellon University 2020

Show that your answer for the above task is *not* a loop invariant of a loop with *this* body. Give specific values for all variables such that `n` and `A` satisfy the preconditions, the loop guard `lo < hi` evaluates to `true`, and your loop invariants from the previous question evaluate to `true` before this loop body runs, but those loop invariants evaluate to `false` after this loop body runs.



© Carnegie Mellon University 2020

4. **Timing Code**

The following run times were obtained when using two different algorithms on a data set of size n. You are asked to extrapolate the asymptotic complexity of these algorithms based on this time data. Determine the asymptotic complexity of each algorithm as a function of n. Use big-O notation in its tightest form and briefly explain how you reached the conclusion.

1pt **4.1**

n	Execution Time
1000	0.564 milliseconds
2000	2.271 milliseconds
4000	8.992 milliseconds
8000	36.150 milliseconds

Asymptotic complexity: $O(____________________)$

1pt **4.2**

n	Execution Time
1000	0.043 milliseconds
1000000	43.68 milliseconds
1000000000	43.9 seconds

Asymptotic complexity: $O(____________________)$

© Carnegie Mellon University 2020