CS 1110, LAB 12: SEQUENCE ALGORITHMS

http://www.cs.cornell.edu/courses/cs1110/2014fa/labs/lab12.pdf

First Name: ____________ **Last Name**: ____________ **NetID**: ____________

This last lab is extremely important. It helps you understand how to construct complex algorithms on sequences, such as we have done in class. This is a important part of the final exam (and the only new topic not on Prelim 1 or 2). Therefore, even if you have completed enough labs to "skip one" we recommend that you do not skip this lab.

Getting Credit for the Lab. This is a *long* lab. There are several written exercises. There is also a very, very short programming assignment. However, the programming part consists mainly of changing code that we wrote, not coming up with your implementations. To complete the programming part, you should download the file `lab12.py`. To get get credit for this lab, you will need to show both this worksheet and your file `lab12.py` to your instructor.

While it is okay if you do not finish the lab during class time, remember that there is *no official lab next week* because of Thanksgiving. We will hold lab on Tuesday for students who want the help, but there is no lab on Wednesday. However, this is an important enough lab that we would prefer for you to do it before Thanksgiving. Therefore, **next Tuesday is open to any student that wants to check off the lab, regardless of section**.

Exercise 1: Warm-Up Exercises

Completing Assertions. Each line below contains an assertion P guaranteed to be true. It also contains an assertion R, which we would like to be true. In the righthand column, put a boolean expression that, when true, allows us to conclude R is true. We have filled in the first one for you.

Know P	**Want** R	**Additional Info Needed**
x is the sum of 1..`n`	x is the sum of 1..100	`n == 100`
x is the sum of 1..(`n`−1)	x is the sum of 1..100	
x is the product of `k..n`	x is the product of 1..`n`	
x is smallest element of the segment `s[0..k−1]`	x is smallest element of the segment `s[0..len(s)−1]`	
x is the smallest element of the segment `s[h..]`	x is the smallest element of the segment `s[0..]`	
b is True if nothing in `h..k` divides x; False otherwise	b is True if nothing in `m..k` divides x; False otherwise	

Course authors: D. Gries, L. Lee, S. Marschner, W. White

Preserving Invariants. Below is a precondition P, an assignment to a variable, and P rewritten as a postcondition. At the place indicated, write a statement so that if P is true initially, it will be true afterward (as indicated). The statement can be in English, if you are not sure how to write it in Python, but make it a command to do something. In the exercises below, `v` is a list of `ints`.

```
(a) # P: x is the sum of 1..n
    # Put a statement here:

    n = n + 1
    # P: x is the sum of 1..n
```

```
(b) # P: x is the sum of h..100
    # Put a statement here:

    h = h - 1
    # P: x is the sum of h..100
```

```
(c) # P: x is the minimum of v[0..k-1]
    # Put a statement here:

    k = k + 1
    # P: x is the minimum of v[0..k-1]
```

```
(d) # P: x is the minimum of v[h..100]
    # Put a statement here:

    h = h - 1
    # P: x is the minimum of v[h..100]
```

Drawing Horizontal Diagrams. Draw a diagram for list `b` that satisfies these conditions

```
b[0..i] >= 5, b[i+1..j] = 5, b[j+1..] <= 5
```

Exercise 2: Sequence Algorithms

This part of the lab gives you exercise with using invariants to implement algorithms on sequences, as covered in Lecture 23. If you are having difficulty with this part of the lab, you might want to review the loop invariant handout:

http://www.cs.cornell.edu/courses/cs1110/2014fa/materials/loop_invariants.pdf

In writing your algorithms in the space on the next page, we want the algorithms to be as close to Python as possible, and not high-level "pseudo-code". However, if you want to swap two elements of an list, you are permitted to write

```
swap b[i] and b[j];
```

instead of the the three assignments that perform the swap. This is the only algorithm shortcut that we are permitting.

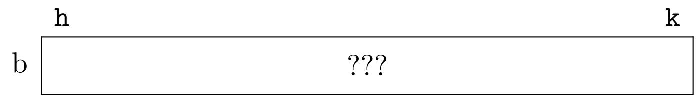
Finding the Minimum of the List. The following are assertions for an algorithm to find the minimum of an list **b[h..k]**:

Precondition: `h <= k < len(b)`

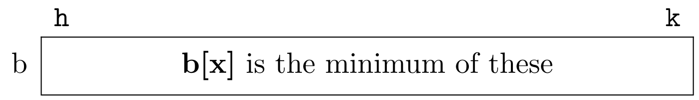
Postcondition: **b**[**x**] is the minimum of **b**[**h..k**]

We represent each of these assertions as the following pictures.

pre:



post:

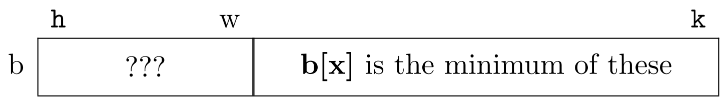


There any many, many different invariants we could construct that would be compatible with these assertions. Consider the following:

invariant: `b[x]` is the minimum of `b[w+1..k]`

Pictorially, we represent it as follows:

P4:



Write your loop for this invariant in the space below:

Partitioning on a Fixed Value. The algorithms below swap the values of list b and store a value in k to make the postcondition is true. List b is not sorted initially. The precondition and postcondition are as follows:

Precondition: `b[0..]` $= ?$ (i.e. nothing is known about the values in b)

Postcondition: `b[0..k]` ≤ 6 and `b[k+1..]` > 6

Below are two different invariants. You should write a loop (with initialization) for each one. You are also to draw pictorial representation of the invariant in each case.

Invariant P1. Written in text form, this invariant is as follows:

P2: `b[0..k]` ≤ 6 and `b[t..]` > 6

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below:

Invariant P3. Written in text form, this invariant is as follows:

P3: `b[0..s-1]` ≤ 6 and `b[k+1..]` > 6

Draw the pictorial representation of this invariant below

Write your loop for this invariant in the space below:

Exercise 3: Implementing Loops With Invariants

Inside of `lab12.py` you will see the specifications of four functions. You are to implement **the first two of the functions in this module**. The last two functions are optional.

Each implementation must contain a while-loop. Write one at a time, implementing the specification that we give you. When a loop invariant has not been given (such as with the last function), write your own.

Make sure each function is correct before proceeding to the next one. Do this by writing suitable calls in the interactive prompt or making a unit test (though we do not ask that you turn in these tests). Use enough different test cases so that you really are sure that the function is correct. If the function uses a string value, make sure that it works on an empty string (one whose length is 0). File `lab12.py` contains additional comments.

When you have completed the first two functions, you are done.