CS 1110, LAB 11: SEQUENCE ALGORITHMS

http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab11.pdf

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

This 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 second prelim.

There are no files to download for this lab. The purpose of this lab is to design loop algorithms, and you are to write the algorithms that you design (together with the diagrams for your pre and postconditions) on this sheet of paper. If you finish during class time, show this handout to your instructor. You instructor will then swipe your ID card to record your success. You do not need to submit the handout.

You only need to finish **the first two exercises**; the last one is optional. If you do not finish during lab, you need to show this lab to an instructor or consultant as usual **by the following Monday**, the day before the prelim.

For each problem you are asked to draw the loop invariant as a picture, which you should do before continuing to work on the problem. If you need help understanding how to draw these pictures, you should ask your TA or consultant. You **do not need to implement these algorithms**. However, if you wish to test your programs in Python, you are welcome to do so. In this case, you may find it useful to print out the contents of the list after each time that you complete the algorithm.

When you implement some of the algorithms in this lab it will be useful to know the idiomatic Python trick for swapping two values: if `a` and `b` are variables you can assign to (local variables, array elements, object attributes—anything that can occur on the left of an assignment operator) then the code "`a, b = b, a`" has the effect of exchanging the two variables. Unlike when using two simple assignment statements "`a = b`" and "`b = a`", the two assignments effectively happen at the same time, so there is no need to use a temporary variable to store one value while it is overwritten. For instance, if `b` is a list then "`b[2], b[3] = b[3], b[2]`" swaps the values at indices 2 and 3 in the array. (Of course, Python is secretly using some kind of temporary storage of its own in order to accomplish this.)

More Practice Material. You can find more worked examples of problems like the ones on this lab in the following handout, which is also linked as additional material to the loop invariants lecture on April 10:

http://www.cs.cornell.edu/courses/cs1110/2013sp/materials/loop_invariants.pdf

Loops and loop invariants will be an important topic on Prelim 2. Be sure you understand the examples in this lab as well as the examples in that handout, so that you're well prepared for the problems on the exam.

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

Exercise 1: Warm-Up Exercises

Counting the values in `b[x..y-1]`. *Without slicing the list `b`*, write a Python expression that evaluates to the number of elements in the segment `b[x..y-1]`.

If you do not know what the formula is, first try to figure it our by looking at segments of size 1 and 2. If you are stuck, check the back of this handout. That formula is a useful tool.

Drawing List Diagrams. Draw a list `b` that satisfies these conditions

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

After you draw it, check your picture against the second-to-last page of this handout.

Exercise 1: Partitioning on a Fixed Value

The algorithms below swap the values in a list `b` and store a value in `k` to make the postcondition true. The idea is to rearrange the values so that all the values greater than 6 are at the end of the array and `k` tells us where the boundary is; 6 is called the "pivot" value. 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 draw the invariant and write a loop (with initialization) for **either one of them**—finish the other one after you've checked off the lab.

Invariant 1. Written in text form, this invariant is as follows:

Inv: `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 2. Written in text form, this invariant is as follows:

Inv: `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 2: General Partitioning Algorithm

This question is a generalization of the previous one. Again `b` is not necessarily sorted initially, and we are putting values smaller than a pivot value in the left part of the list and values larger than that pivot in the right part. The difference is that this time the pivot value is one of the values in the list, rather than being given separately. Develop the partition algorithm (which uses only swap operations) to meet the assertions below:

Precondition: `b[h] =` x for some x and `h` $\leq$ `k` $<$ `len(b)`
(The variable x is here so we can talk about the original value of `b[h]`; x is not a program variable.)

Postcondition: `b[h..j-1]` $\leq x =$ `b[j]` $\leq$ `b[j+1..k]`

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

Inv: `b[h..j-1]` $\leq x =$ `b[j]` $\leq$ `b[q+1..k]`

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below:

Invariant 2. Written in text form, this invariant is as follows:

Inv: `b[h..j-1]` $\leq x =$ `b[j]` $\leq$ `b[j+1..n-1]`

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below:

Exercise 3: Selection Sort (OPTIONAL)

The last exercise is optional—you don't have to do it now but should review it when it comes time to study for the final. The task is to write a selection sort, which sorts the contents of the list b. The postcondition for this problem is straightforward:

Postcondition: `b[0..len(b)-1]` is sorted (in ascending order)

We have provided several invariants for you below. Before you do each one, write the invariant as a picture. Then write the statement(s) you use to maintain the invariant in the body in English. You should state *what* it is you want to do, not *how* to do it. In particular, we do not want to see a nested loop (e.g. a loop within the body of your first loop).

Instead, you should assume the existence of a function `min_pos(h, k)` that contains a solution to Exercise 2 in the worked examples handout mentioned on the first page of this lab. Then use that function as a helper in this part of the lab.

Invariant 1. Written in text form, this invariant is as follows:

Inv: `b[0..k1]` is sorted and `b[0..k1]` $\leq$ `b[k..]`

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below:

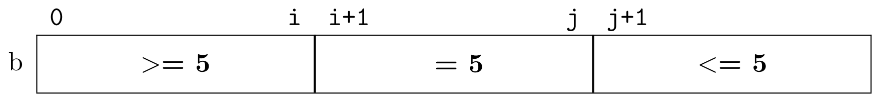
Invariant 2. Written in text form, this invariant is as follows:

Inv: `b[0..h]` is sorted and `b[0..h]` $\leq$ `b[h+1..]`

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below:

The warmup invariant diagram is:



Invariant 3. Written in text form, this invariant is as follows:

Inv: `b[s+1..len(b)-1]` is sorted and `b[0..s]` $\leq$ `b[s+1..]`

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below. Your code will require a helper function which is similar to, but not the same as one of the previous exercises. You do not have to implement the helper function so long as you clearly explain what it does.

The answer to the warmup problem is `y - x`.