CS 1110, LAB 13: MORE SEQUENCE ALGORITHMS

http://www.cs.cornell.edu/courses/cs1110/2014fa/labs/lab13.pdf

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

This is an optional lab; it does not need to be turned in or checked off. The purpose of this lab is to give you extra practice to study for the final exam. This lab is similar to the second half of Lab 12, in that you have to modify code to satisfy invariants. You will be shown several of the sequence algorithms in class. For each one you are given an invariant. You are to draw the invariant and note how this invariant affects the code for the algorithm.

Algorithm Shortcuts. As with the previous lab, 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.

Exercise 1: General Partitioning Algorithm

Assume that list `b` is not necessarily sorted initially. The partition algorithm (which uses only swap operations) is designed to meet the assertions below:

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

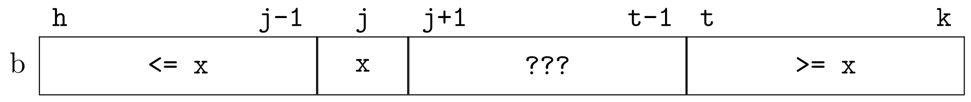
Postcondition: `b[h..j-1]` $\leq x =$ `b[j]` $\leq$ `b[j+1..k]`

Below are three different invariants. We provide the first one as an example. You are to complete the invariant and the algorithm code for the other two.

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

P1: `b[h..j-1]` $\leq x =$ `b[j]` $\leq$ `b[t..k]`

The pictorial representation of this invariant is as follows:



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

The following is the code for this invariant:

```
# Make invariant true at start
j = h
t = k+1

# inv:  b[h..j-1] <= x = b[j] <= b[t..k]
while j < t-1:
    if b[j+1] <= b[j]:
        swap b[j] and b[j+1]
        j = j + 1

    else:
        swap b[j+1] and b[t-1]
        t = t - 1

# post:  b[h..j-1] <= x = b[j] <= b[j+1..k]
```

Invariant P2. Written in text form, this invariant is as follows:

P2: 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 P3. Written in text form, this invariant is as follows:

P3: `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 2: Selection Sort

The function selection sort is an algorithm sorts the contents of the list **b**. The postcondition for this problem is straightforward:

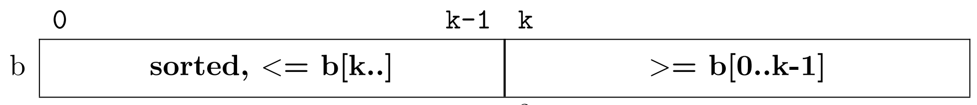
Postcondition: `b[0..len(b)-1]` is sorted (in ascending order)

We have provided several invariants for you below. The first one is completely worked out for you. For the other two, you should draw the invariants and provide the code for the algorithm.

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

P1: `b[0..k-1]` is sorted and `b[0..k-1]` $\leq$ `b[k..]`

The pictorial representation of this invariant is as follows:



When writing the code for this algorithm, 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 take one of the algorithms from Lab 12 and turn it into a helper function. We have done this below.

```
# Make invariant true at start
k = 0

# inv:  b[0..k-1] is sorted and b[0..k-1] <= b[k..]
while k < len(b):
    pos = minpos(b,k,len(b)-1) # Exercise 2 from Lab 12.
    swap b[k] and b[x]         # Put b[x] in sorted region
    k = k + 1

# post:  b[0..len(b)-1] is sorted
```

Invariant P2. Written in text form, this invariant is as follows:

P2: `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:

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

P3: `b[s+1..len(b)-1]` is sorted and `b[0..s]` $\leq$ `b[s+1..]`

Draw the pictorial representation of this invariant below.

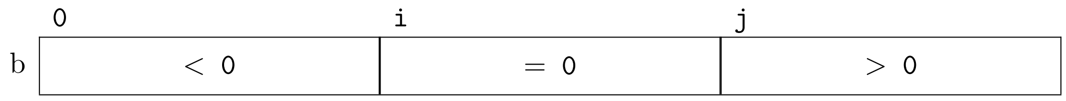
Write your loop for this invariant in the space below. Your code will require a helper function which is similar to, but not the same as one of the previous exercises. You do not have to implement the helper function so long as you clearly explain what it does.

Exercise 3: Dutch National Flag

The last algorithm is like partition in that it groups numbers into general buckets without actually sorting them. While partition groups the numbers into two buckets (those ¡= and ¿= the pivot `x`), Dutch National Flag organizes them into three buckets. Stated as text, the postcondition is

> **Postcondition**: The elements of `b[0..i-1]` are negative, the elements of `b[j..len(b)-1]` are positive, and the elements of `b[i..j-1]` are all zeroes.

The pictorial representation of this postcondition is as follows:

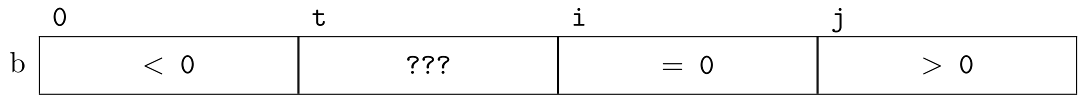


Below are two invariants for this algorithm. We have worked out the first one for you. You are to complete the second, drawing the invariant pictorially and providing the algorithm code.

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

> **P1**: The elements of `b[0..t-1]` are negative, the elements of `b[j..len(b)-1]` are positive, and the elements of `b[i..j-1]` are all zeroes.

The pictorial representation of this invariant is as follows:



The code satisfying this invariant is shown below.

```
# Make invariant true at start
t = 0; j = len(b); i = len(b)

# inv:  b[0..t-1] < 0, b[t..i-1] unknown, b[i..j-1] = 0, and b[j..]  > 0
while t < i:
    if b[i-1] < 0:
        swap b[i-1] and b[t]
        t= t+1
    elif b[i-1] == 0:
        i= i-1
    else:
        swap b[i-1] and b[j]
        i= i-1; j= j-1

# post:  b[0..i-1] < 0, b[i..j-1] = 0, and b[j..]  > 0
```

Invariant P2. Written in text form, this invariant is as follows:

> **P1**: The elements of `b[0..i-1]` are negative, the elements of `b[j..len(b)-1]` are positive, and the elements of `b[i..t-1]` are all zeroes.

Draw the pictorial representation of this invariant below.

Write your loop for this invariant in the space below. When you are done, the lab is finished.