NOTE: ***you do not need to be able to know the algorithms in this lab for the prelim of November 9th*****! But, you will need to know these algorithms for the final; and for prelim 2, you should be familiar with developing and understanding loops using invariants.**

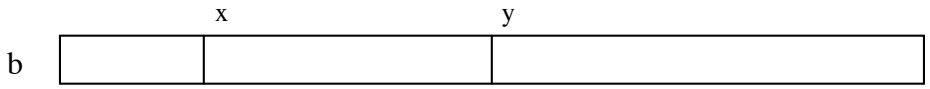
This lab gives you practice with developing loops from invariants and with the array algorithms that you have to know. Your lab instructor and any consultants that are present will help you.

In our sample answers, we don't draw all the pictures that are required. We suggest that if an invariant is not drawn as a picture, *you* draw it as a picture before continuing to work on the problem. If you need help, ask your TA or consultant.

When you are finished, show your TA what you have done. If you finish at least exercises 1-4, your lab instructor will mark you down as having completed the assignment. If you don't finish that much, finish exercises 1-4 by the next week and show them to your TA at the *beginning* of the lab next week. In any case, try to do all the exercises because that will help you understand and gain skill in solving such problems.

Note: if you wish to test your programs in DrJava, you may find it useful to print out the contents of an array. The import statement "import java.util.*;" (either in a program or in the Interactions pane) gives you access to the method Arrays. toString(Object[] arr), which prints out the contents of array arr.

1. Give a formula for the number of values in the segment b[x..y-1]:



Not sure what the formula is? Then ask, and *memorize it*. Also, learn to figure it out by looking at segments of size 1 and 2. The formula is a useful tool. Knowing it will help you develop loops that deal with ranges of integers.

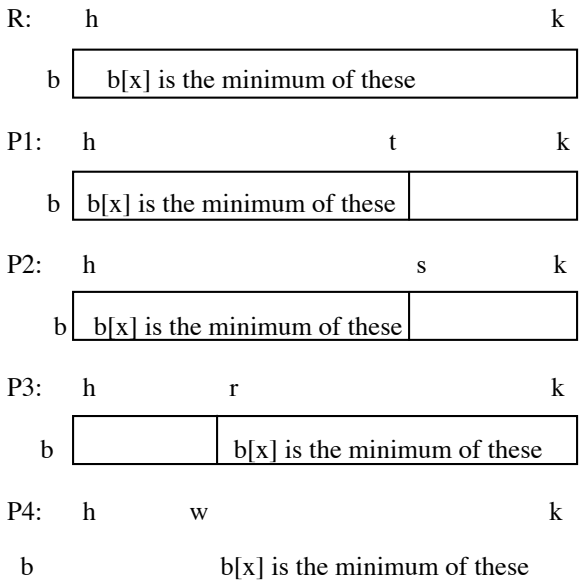
2. Draw an array b that satisfies these conditions: b[0..i] >= 5, b[i+1..j] = 6, b[j+1..] <= 5.

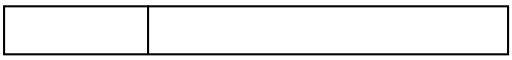
3. Below is a precondition and postcondition for the algorithm to find the minimum of an array b[h..k]. Below that are four different loop invariants for the problem. Write a loop (with initialization) for each one.

Precondition: h <= k
Postcondition R: b[x] is the minimum of b[h..k]

(a) invariant P1: b[x] is the minimum of b[h..t]
(b) invariant P2: b[x] is the minimum of b[h..s–1]
(c) invariant P3: b[x] is the minimum of b[r..k]
(d) invariant P4: b[x] is the minimum of b[w+1..k]

Below are the postcondition and invariants drawn as pictures. Use either the text versions or the pictures.





4. The purpose of the algorithms in this question is to swap the values of array b and to store a value in k so that the postcondition given below is true. Array b is not necessarily sorted initially. Besides the postcondition, we give three different invariants; write a loop (with initialization) for each one. Before you begin, write the precondition, post-condition, and invariant as pictures.

Precondition Q: b[0..] = ? —i.e. nothing is known about the values in b.
Postcondition R: b[0..k] ≤ 6 and b[k+1..] > 6

(a) invariant P1: b[0..h] ≤ 6 and b[k+1..] > 6
(b) invariant P2:...b[0..k] ≤ 6 and b[t..] > 6
(c) invariant P3: .. b[0..s–1] ≤ 6 and b[k+1..] > 6

5. Below is the precondition and postcondition for the partition algorithm. Below that are three different invariants. Develop the partition algorithm (which uses only swap operations) using each of the three invariants. Before you start, write all assertions as pictures. This algorithm manipulates an array segment b[h..k].

Precondition: b[h] = x for some x AND (this is just so we can talk about what is in b[h] initially;
h ≤ k x is not a program variable.)
Postcondition: b[h..j-1] ≤ x = b[j] <= b[j+1..k]
(a) invariant P1: b[h..j-1] ≤ x = b[j] <= b[t..k]
(b) invariant P2: b[h..j-1] ≤ x = b[j] <= b[q+1..k]
(c) invariant P3: b[h..j-1] ≤ x = b[j] <= b[j+1..n-1]

6. Write selection sort, to sort array b, where b.length >= 1, in several ways, using the invariants provided below. Before you do each one, write the invariant as a picture. Write the statement(s) you use to maintain the invariant in the repetend in English —state WHAT it is to do, not HOW it is to do it.

postcondition: b[0..b.length–1] is sorted (in ascending order)

(a) invariant P1: b[0..k–1] is sorted, and b[0..k–1] ≤ b[k..]
(b) invariant P2: b[0..h] is sorted, and b[0..h] ≤ b[h+1..]
(c) invariant P3: b[s+1..b.length–1] is sorted, and b[0..s] ≤ b[s+1..]