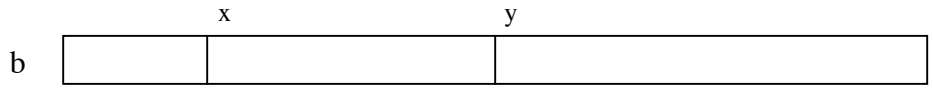
This lab gives you practice with developing loops from invariants and with the array algorithms that you have to know for prelim III. Your lab instructor and any consultants that are present will help you. Answers are at the end of this document, but don't look at an answer until you have done an exercise.

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 a skill in solving such problems.

If you finish these ahead of time and you want something else to do, work on assignment A7.

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



If you are not sure what the formula is, then ask, and *memorize it*. The formula is a useful tool, and knowing it will help you develop loops that deal with ranges of integers.

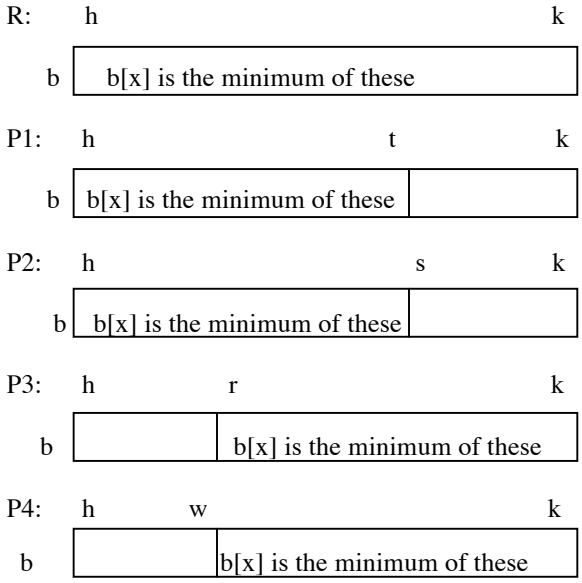
2. Draw an array b that satisfies these conditions b[0..i] >= 5, b[i+1..j] = 6, b[j+1..] <= 5.

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 these algorithms 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, postcondition, and invariant as pictures.

Precondition Q: b[0..] = ? —i.e. we know nothing 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. Write selection sort, to sort array segment b[h..k], in several ways, using the invariants provided below. Before you do each one, write the invariant as a picture.

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..]

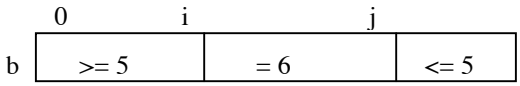
6. Below is the precondition and postcondition for the partition algorithm. Below that are three different invariants. Develop the partition algorithm using each of the three invariants. Before you start, write all assertions as pictures. This algorithm manipulates an array segment b[h..k].

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]

Answers to questions.

1. y – x

2.



3. (a)
```
t= h; x= t;
while (t != k) {
  if (b[t+1] < b[x]) x= t+1;
  t= t + 1;
}
```

(b)
```
s= h+1; x= h;
while (s-1 != k) {
  if (b[s] < b[x]) x= s;
  s= s + 1;
}
```

(c)
```
r= k; x= r;
  while (r != h) {
    if (b[r–1] < b[x]) x= r–1;
    r= r–1;
  }
```

(d)
```
w= k–1; x= k;
  while (w+1 != h) {
    if (b[w] < b[x]) x= w;
    w= w–1;
  }
```

4. (a)
```
h= –1; k= b.length–1;
// invariant: P1
while (h != k) {
  if (b[h+1] <= 6) h= h+1;
  else { Swap b[h+1] and b[k]; k= k–1; }
}
```

(b)
```
k= –1; t= b.length;
// invariant: P2
while (t != k+1) {
  if (b[k+1] <= 6) k= k+1;
  else { Swap b[k+1] and b[t–1]; t= t–1; }
}
```

(c)
```
s= 0; k= b.length–1;
// invariant: P3
while (s–1 != k) {
    if ( b[s] <= 6) s= s+1;
    else { Swap b[s] and b[k]; k= k–1; }
}
```

5. (a)
```
k= 0;
// invariant: P1
while (k != b.length] {
  Set t to the index of the minimum
     of b[k..b.length–1];
  Swap b[k] and b[t];   k= k+1;
}
```

(b)
```
h= –1;
// invariant: P2
while (h != b.length–1]) {
  Set t to the index of the minimum
     of b[h+1..b.length–1];
  Swap b[h+1] and b[t];   h=  h+1;
}
```

(c)
```
s= b.length–1;
// invariant: P3
while (s != –1) {
  Set t to the index of the maximum of b[0..s]
  Swap b[t] and b[s];      s= s–1;
}
```

6. (a)
```
j= h;  t= k+1;
// invariant: P1
while (j != t-1) {
  if (b[j+1] <= b[j]) {
     Swap b[j] and b[j+1];
     j= j+1;
  else {
     t= t – 1;
    Swap b[t] and b[j+1]
   }
}
```

(b)
```
j= h; q= k;
// invariant: P2
while ((j != q){
  if (b[j + 1] <= b[j]) {
     Swap b[j] and b[j + 1];
     j= j + 1;
  else {
    Swap b[q] and b[j+1]
    q= q – 1;
  }
}
```

(c)
```
j= h;  n= j+1;
// invariant: P3
while (n <= k) {
   if (b[n] <= b[j]) {
      Swap b[n] and b[j+1]; n= n+1;
      Swap b[j] and b[j+1]; j= j+1;
   } else {
      n= n+1;
   }
}
```