Practice Questions on Loop Invariants

These questions (and the ones on mathematical prerequisites) are good preparation for Quiz #1. They are not to be handed in. Some hints follow the questions, but don't look at them unless you are stuck after working on the problem for a while.

1. Recall the Boyer-Moore majority algorithm from class.

```
1:  MAJORITY(A[1..n]) : boolean
2:      ▷ Phase 1: find a candidate that might be the majority element
3:      count ← 0
4:      for i ← 1..n do
5:          if count = 0 then cand ← A[i]
6:          if A[i] = cand then count + +
7:          else count − −
8:      ▷ Phase 2: check whether candidate is actually the majority element
9:      count ← 0
10:     for i ← 1..n do
11:         if A[i] = cand then count + +
12:     return count > n/2
```

(a) Adapt the algorithm to detect whether some element appears more than n/3 times. Your algorithm should run in linear time and use $O(1)$ extra memory locations.

(b) Use loop invariants to prove that your algorithm works.

2. We want to evaluate the polynomial $p(x) = \sum_{j=0}^{d} a_j x^j = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_2 x^2 + a_1 x + a_0$, where a_i are given coefficients.

(a) Write loop invariants to prove that each of the following two algorithms compute $p(x)$.

```
P1(x)
    sum ← 0
    power ← 1
    for i ← 0..d do
        sum ← sum + a_i · power
        power ← power · x
    return sum
```

```
P2(x)
    sum ← 0
    for i ← d down to 0 do
        sum ← sum · x + a_i
    return sum
```

(b) Which function is preferable, and why?

3. Let $A[1..n]$ be a sorted array of n distinct integers. We say an index i is *special* if $A[i] = n - i$.

[1] **(a)** Does $A[1..n] = [2, 3, 6, 8, 12]$ contain a special index? If so, what is that index?

[4] **(b)** Fill in the blanks in the following binary search algorithm to find a special index, if there is one.

```
1:  FindSpecial(A[1..n]) : int
2:      precondition: n ≥ 1 and A[1] < A[2] < A[3] < ··· < A[n]
3:      lo ← 1
4:      hi ← n
5:      loop
6:          invariants: (1) 1 ≤ lo ≤ hi ≤ n and
7:                      (2) no index less than lo or greater than hi is special
8:          exit when _____
9:          m ← ⌈(lo+hi)/2⌉
10:         if _____ then
11:             hi ← _____
12:         else
13:             lo ← _____
14:     if A[lo] = n − lo then return lo
15:     else return 0
16:     postcondition: returns a special index if there is one, or 0 if there is none
```

(c) Prove your algorithm terminates.

(d) Prove your algorithm satisfies the postconditions.

4. (Challenging) Before working on this problem, it may be helpful to read the standard (two-way) partition algorithm on page 184 of the textbook.

Design an algorithm to perform a *three-way* partition of an array using pivot p. The goal is to rearrange the elements of $A[1..n]$ so that the first part of the array contains elements less than p, the middle part contains elements equal to p, and the third part contains elements greater than p.

For example, Partition([3 2 4 3 1 3 3 2 6 5], 3) could modify the array to [2 1 2 3 3 3 3 4 6 5]. (Other answers are also possible, but this answer has the nice additional property that the relative order of elements less than p is unchanged, and the relative order of elements greater than p is unchanged.)

Your algorithm should run in linear time, and use $O(1)$ extra memory locations (in addition to the array A).

Hints

1. For Phase 1, keep track of two candidates, $candidate_1$ and $candidate_2$, and two counters, $count_1$ and $count_2$. Think about how to generalize the invariant for the majority problem discussed in class to this situation.

2. The invariant can simply say what the value of each of the local variables is at the beginning (or end) of each iteration of the loop. Use $\sum$ notation to describe the value of sum.

3. Think about what condition you can test that will allow you to decrease hi (while maintaining the invariant). Check that if that condition is false, you can increase lo (while maintaining the loop invariant).

4. Maintain 3 array indices i, j, k so that

- entries in $A[1..i]$ are less than p.
- entries in $A[i+1..j]$ are equal to p.
- entries in $A[j+1..k-1]$ have not been looked at yet.
- entries in $A[k..n]$ are greater than p.

In each iteration of the loop, look at one or two array entries, and maybe swap a pair; make small changes to indices i, j, k so that $k - j$ decreases.

Dijkstra called this problem the Dutch flag problem because the final answer has three segments, like the stripes on the Dutch flag.