Review 7

Required Algorithms
Algorithms on the Final

- One of these is on the final:
  - binary search
  - Dutch national flag
  - partition algorithm
  - insertion sort
  - selection sort
- Will be asked to write one
  - Have to know specifications
  - Develop invariant from spec
  - Develop the loop from inv

Reasons for this:
1. Important algorithms.
2. Forces you to think in terms of specifications.
3. Forces you do learn to develop invariants.
4. Forces you to learn to use the four loopy questions in reading/developing a loop

- Answer is wrong if it
  - Does not give the invariant
  - Does not use the invariant
Algorithms on the Final

- One of these is on the final:
  - binary search
  - Dutch national flag
  - partition algorithm
  - insertion sort
  - selection sort

- Will be asked to write one
  - Have to know specifications
    And be able to use them.
  - Develop invariant from spec
  - Develop the loop from inv

Reasons for this:

1. Important algorithms.
2. Forces you to think in terms of specifications.
3. Forces you do learn to develop invariants.
4. Forces you to learn to use the four loopy questions in reading/developing a loop

- Answer is wrong if it
  - Does not give the invariant
  - Does not use the invariant
Example of an assertion about an sequence b. It asserts that:

1. \( b[0..k-1] \) is sorted (i.e. its values are in ascending order)
2. Everything in \( b[0..k-1] \) is \( \leq \) everything in \( b[k..\text{len}(b)-1] \)

Given index \( h \) of the first element of a segment and index \( k \) of the element that follows that segment, the number of values in the segment is \( k - h \).

\( b[h..k-1] \) has \( k - h \) elements in it.
• DON’T put variables directly above vertical line.

- Where is j?
- Is it unknown or >= x?
Algorithm Inputs

- We may specify that the list in the algorithm is
  - b[0..len(b)-1] or
  - a segment b[h..k] or
  - a segment b[m..n-1]

- **Work with whatever is given!**

- Remember formula for # of values in an array segment
  - Following – First
  - e.g. the number of values in b[h..k] is k+1–h.
Binary Search

• **Vague:** Look for \( v \) in sorted segment \( b[h..k] \).

• **Better:**
  - **Precondition:** \( b[h..k] \) is sorted (in ascending order).
  - **Postcondition:** \( b[h..i-1] < v \) and \( v \leq b[i..k] \)

• Below, the sequence is in non-descending order:

<table>
<thead>
<tr>
<th>pre: ( b )</th>
<th>( h )</th>
<th>?</th>
<th>( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: ( b )</td>
<td>( h )</td>
<td>( i )</td>
<td>( k )</td>
</tr>
<tr>
<td>( &lt; v )</td>
<td>( \geq v )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h )</td>
<td>( i )</td>
<td>( j )</td>
<td>( \geq v )</td>
</tr>
</tbody>
</table>

Called binary search because each iteration of the loop cuts the array segment still to be processed in half.
Dutch National Flag

- Tri-color flag represented by an list
  - Array of 0..n-1 of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

\[
\begin{array}{ccccccc}
0 & & & & & & n \\
\text{pre: } & b & & & & \text{?} & \\
0 & & & & & & n \\
\text{post: } & b & \text{reds} & \text{whites} & \text{blues} & & \\
0 & j & k & l & & n \\
\text{inv: } & b & \text{reds} & \text{whites} & \text{?} & \text{blues} & \\
\end{array}
\]

(values in 0..n-1 are unknown)

Make the red, white, blue sections initially empty:
- Range i..i-1 has 0 elements
- Main reason for this trick

Changing loop variables turns invariant into postcondition.
Invariants are Not Unique

- Invariants come from combining pre-, postconditions
  - Often more than one way to do it (see below)
  - **Do not memorize them.** Work them out on your own

**binary search**

<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>t</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

inv: `b < v ? >= v`

**Dutch National Flag**

<table>
<thead>
<tr>
<th>0</th>
<th>h</th>
<th>k</th>
<th>m</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

inv: `b`  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>whites</th>
<th>blues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partition Algorithm

- Given an segment \( b[h..k] \) with some value \( x \) in \( b[h] \):

\[
\begin{array}{c|c}
\text{pre:} & b \begin{array}{c}
\text{?} \\
x
\end{array} \\
\text{post:} & b \begin{array}{c|c|c}
\leq x & x & \geq x
\end{array}
\end{array}
\]

- Swap elements of \( b[h..k] \) and store in \( j \) to truthify post:

\[
\begin{array}{c|c|c|c|c|c|c}
\text{change:} & b & 3 & 5 & 4 & 1 & 6 & 2 & 3 & 8 & 1 \\
\text{into} & b & 1 & 2 & 1 & 3 & 5 & 4 & 6 & 3 & 8 \\
\text{or} & b & 1 & 2 & 3 & 1 & 3 & 4 & 5 & 6 & 8
\end{array}
\]

- \( x \) is called the **pivot value**
  - \( x \) is not a program variable
  - denotes value initially in \( b[h] \)
Partition Algorithm

- Given an segment $b[h..k]$ with some value $x$ in $b[h]$: 
  
  \[
  \begin{array}{c|c|c|c|c}
  h & x & \text{?} & k \\
  \hline
  \text{pre:} & b & x & \text{?} \\
  \end{array}
  \]

- Swap elements of $b[h..k]$ and store in $j$ to truthify post: 
  
  \[
  \begin{array}{c|c|c|c|c}
  h & i & i+1 & k \\
  \hline
  \text{post:} & b & \leq x & x & \geq x \\
  \end{array}
  \]

- Agrees with precondition when $h = i$, $j = k+1$
- Agrees with postcondition when $j = i+1$
Insertion Sort AND Selection Sort

Insertion Sort:

pre: b

inv: b sorted

post: b sorted

Selection Sort:

inv: b sorted, \( \leq b[i..] \) \( \geq b[0..i-1] \)

DO have to remember difference between the two sorting invariants

First segment always contains smaller values
Insertion Sort vs. Selection Sort

Insertion Sort

Selection Sort

Find minimum

12/4/12
Review 7
## Insertion Sort vs. Selection Sort

### Insertion Sort

\[
i = 0\\
\text{while } i < n:\\
\text{pushdown}(b,i)\\
i = i + 1\\
\]

**def pushdown(b, i):**

# inv: b[j] < b[j+1..i]
\[
j = i\\
\text{while } j > 0:\\
\text{if } b[j-1] > b[j]:\\
\text{swap}(b,j-1,j)\\
j = j - 1
\]

### Selection Sort

\[
i = 0\\
\text{while } i < n:\\
\text{j = minPos}(b,i,n-1)\\
\text{swap}(b,i,j)\\
i = i + 1\\
\]

**def minpos(b, h, k):**

"""Returns: min position in b[h..k]"

# inv: ???
...
# post: ???
Insertion Sort

```python
i = 0
while i < n:
    pushdown(b,i)
    i = i + 1

def pushdown(b, i):
    # inv: b[j] < b[j+1..i]
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

Selection Sort

```python
i = 0
while i < n:
    j = minPos(b,i,n-1)
    swap(b,i,j)
    i = i+1

def minpos(b, h, k):
    """Returns: min position in b[h..k]""
    # inv: b[x] is minimum of b[h..j]
    ... # post: b[x] is minimum of b[h..k]
```
def swap(b, h, k):
    """Swaps b[h] and b[k] in b
    Pre: b is a mutable list, h and k are valid positions in b. """
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
def dutch_national_flag(b, h, k):
    """Use a Dutch National Flag algorithm to arrange the elements of b[h..k] and produce a tuple (i, j). Precondition and postcondition are given above."""
    ...

pre: b  
| h | ? | k |

post: b  
| h | i | j | k |
| <0 | =0 | >0 |
Dutch National Flag (Spring ‘11)

pre:  b

post: b

inv: b

# inv: b[h..t-1] < 0, b[t..i-1] unknown, b[i..j] = 0, and b[j+1..k] > 0
def dutch_national_flag(b, h, k):
    """Use a Dutch National Flag algorithm to arrange the elements of b[h..k] and produce a tuple (i, j). Precondition and postcondition are given above."""
    t = h; j = k; i = k + 1
    # inv: b[h..t-1] < 0, b[t..i-1] unknown, b[i..j] = 0, and b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b[i-1],b[t])
            t = t + 1
        elif b[i-1] == 0:
            i = i - 1
        else:
            swap(b[i-1],b[j])
            i = i - 1; j = j - 1
    return (i, j)
Questions?