Recall: Horizontal Notation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>k</th>
<th>len(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>&lt;=</td>
<td>&gt;=</td>
<td></td>
</tr>
</tbody>
</table>

Example of an assertion about a 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 < everything in b[j..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.

(h+1) - h = 1

Partition Algorithm Implementation

```python
def partition(h, k):
    # Partition list [h:k] around a pivot x = b[h]
    i = h; j = k; x = b[h]
    # invariant: b[h+1] <= x, b[i] = x, b[j+1] >= x
    while i < j:
        if b[i] >= x:
            # Move to end of block
            _swap(b,i+1,j)
            j -= 1
        elif b[j] <= x:
            # Move to end of block
            _swap(b,h+1,i)
            i += 1
        # post: b[h..j+1] = x, b[i] = x, and b[j+1..k] >= x
        return i, j
```

Dutch National Flag Algorithm

```python
def dnf(h, k):
    # Returns: partition points as a tuple (i,j)
    t = h; l = k; j = k
    # invariant: b[l+1] < 0, b[l] = 0, b[l+1..k] >= 0
    while t < l:
        if b[l] < 0:
            # Swap (l+1, l)
            _swap(b,l+1,l)
            l += 1
        else:
            # Swap (l, l+1)
            _swap(b,l,l+1)
            t += 1
        # post: b[l+1..k] >= 0, b[l] = 0, b[l+1..k] >= 0
        return (t, l)
```

Dutch National Flag Variant

- Sequence of integer values
  - 'red' = negatives, 'white' = 0, 'blues' = positive
  - Only rearrange part of the list, not all

```
pre: b
h  k
post: b
h  k
```

```
inv: b
h  k
pre: t=h, i=k+1, j=k
post: t=i
```
Flag of Mauritius

• Now we have four colors!
  * Negatives: ‘red’ = odd, ‘purple’ = even
  * Positives: ‘yellow’ = odd, ‘green’ = even

<table>
<thead>
<tr>
<th>b</th>
<th>h</th>
<th>r</th>
<th>s</th>
<th>i</th>
<th>t</th>
<th>k</th>
</tr>
</thead>
</table>
| <0, odd| <0, e| ≥0, odd| ≥0, e| ?|<0, e| ?

Negatives: ‘red’ = odd, ‘purple’ = even
Positives: ‘yellow’ = odd, ‘green’ = even

Need two swaps for two spaces

See algorithms.py for Python code

Linear Search

def linear_search(b, a, h, k):
    """Returns: first occurrence of c in b[h..]."""
    # Store in i the index of the first c in b[h..]
    i = b
    # Invariant: c is not in b[0..i-1]
    while i < len(b) and b[i] != c:
        i = i + 1
    # post: c is not in b[h..i]
    #    i >= len(b) or b[i] == c
    return i if i < len(b) else -1

Analyzing the Loop
1. Does the initialization make inv true?
2. Is post true when inv is true and condition is false?
3. Does the repeated make progress?
4. Does the repeated keep the invariant inv true?

Precondition: b[h..k-1] is sorted (in ascending order).
Postcondition: b[h..i] <= v and v < b[i+1..k-1]

Below, the array is in non-descending order: