Horizontal Notation for Sequences

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>k</td>
<td>inv(b)</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 \( \leq \) everything in \( b[k..len(b)-1] \)

Developing Algorithms on Sequences

- Specify the algorithm by giving its precondition and postcondition as pictures.
- Draw the invariant by drawing another picture that "generalizes" the precondition and postcondition
  - The invariant is true at the beginning and at the end
- The four loop design questions (memorize them)
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does the body make progress toward termination?
  4. How does the body keep the invariant true?

Generalizing Pre- and Postconditions

- Dutch national flag tri-color
  - Sequence of \( 0..n-1 \) of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

Generalizing Pre- and Postconditions

- Finding the minimum of a sequence.
  - precondition
  - postcondition

- Put negative values before nonnegative ones.
  - precondition
  - postcondition

Partition Algorithm

- Given a sequence \( b[h..k] \) with some value \( x \) in \( b[h] \):
  - Swap elements of \( b[h..k] \) and store in \( j \) to truthify post:

Partition Algorithm

- Given a sequence \( b[h..k] \) with some value \( x \) in \( b[h] \):
  - Swap elements of \( b[h..k] \) and store in \( j \) to truthify post:
**Partition Algorithm Implementation**

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

**Partition Algorithm Implementation**

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

**Dutch National Flag Algorithm**

```python
def dnf(b, h, k):
    """Return partition points tuple (i, j)""
    t = h
    i = h
    j = k
    # invariant: b[i..j] = 0, b[j+1..k] = 0, b[hi..j] > 0
    # post: b[i..j] < 0, b[j+1..k] > 0
    while t < i:
        if b[t] < 0:
            _swap(b, t, j)
            i += 1
            if t + 1 == j:
                _swap(b, t, j)
        else:
            _swap(b, t, i)
            j -= 1
            if t + 1 == i:
                _swap(b, t, i)
        # post: b[i..j] < 0, b[j+1..k] > 0
    return (i, j)
```

**Dutch National Flag Algorithm**

```python
def dnf(b, h, k):
    """Return partition points tuple (i, j)""
    t = h
    i = h
    j = k
    # invariant: b[i..j] = 0, b[j+1..k] = 0, b[hi..j] > 0
    # post: b[i..j] < 0, b[j+1..k] > 0
    while t < i:
        if b[t] < 0:
            _swap(b, t, j)
            i += 1
            if t + 1 == j:
                _swap(b, t, j)
        else:
            _swap(b, t, i)
            j -= 1
            if t + 1 == i:
                _swap(b, t, i)
        # post: b[i..j] < 0, b[j+1..k] > 0
    return (i, j)
```