**Horizontal Notation for Sequences**

- Sequence of 0..n-1 of red, white, blue "pixels"
- Arrange to put reds first, then whites, then blues

**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
    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

**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:

**Generalizing Pre- and Postconditions**

- Finding the minimum of a sequence.
  - Finding the minimum of a sequence.
  - Put negative values before nonnegative ones.

**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 b[h..k] around a pivot x ∈ b[h..k]""
    i = h, j = k, x = b[h]
    # invariant: b[h..i] < x, b[i] == x, b[i+1..k] >= x
    while i < j:
        if b[i] >= x:
            # Move to end of block.
            _swap(b,i,i+1)
            j = j - 1
        else:
            # b[j+1] < x
            _swap(b,j,i)
            i = i + 1
    # post: b[h..i] < x, b[i] == x, and b[i+1..k] >= x
    return i

# post: b[h..i] < x, b[i] == x, and b[i+1..k] >= x

partition(b,h,k), not partition(b[h:k+1])
Remember, slicing always copies the list!
We want to partition the original list
```

Partition Algorithm Implementation

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

# post: b[h..i] < x, b[i] == x, and b[i+1..k] >= x
```

Dutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns partition points as a tuple (t, j)""
    t = h, i = 2, j = k,
    # inv: b[h..i] < 0, b[i..j] > 0, b[j+1..k] >= 0
    while t < i:
        if b[t] < 0:
            _swap(b,t,1)
            t = t + 1
        else:
            _swap(b,t,2)
            i = i + 1
    # post: b[h..i] < 0, b[i] == 0, b[i+1..k] > 0
    return (i, j)
```

Dutch National Flag Variant

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

<table>
<thead>
<tr>
<th>pre</th>
<th>post</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>h</td>
</tr>
<tr>
<td>k</td>
<td>i</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pre</th>
<th>post</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = h, i = k+1</td>
<td></td>
</tr>
<tr>
<td>i = j</td>
<td></td>
</tr>
</tbody>
</table>

Dutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns partition points as a tuple (t, j)""
    t = h, i = 2, j = k,
    # inv: b[h..i] < 0, b[i..j] > 0, b[j+1..k] >= 0
    while t < i:
        if b[t] < 0:
            _swap(b,t,1)
            t = t + 1
        else:
            _swap(b,t,2)
            i = i + 1
    # post: b[h..i] < 0, b[i] == 0, b[i+1..k] > 0
    return (i, j)
```

Dutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns partition points as a tuple (t, j)""
    t = h, i = 2, j = k,
    # inv: b[h..i] < 0, b[i..j] > 0, b[j+1..k] >= 0
    while t < i:
        if b[t] < 0:
            _swap(b,t,1)
            t = t + 1
        else:
            _swap(b,t,2)
            i = i + 1
    # post: b[h..i] < 0, b[i] == 0, b[i+1..k] > 0
    return (i, j)
```