Horizontal Notation for Sequences

<table>
<thead>
<tr>
<th>0</th>
<th>k</th>
<th>len(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= sorted</td>
<td>&gt;=</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 \( b[k..\text{len}(b)–1] \)

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. 
\( (h+1) – h = 1 \)

Generalizing Pre- and Postconditions

- Finding the minimum of a sequence.
  
  **Preconditions:**
  
  - Sequence of \( 0..n–1 \) of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

  **Postconditions:**
  
  - Everything in \( b[0..k–1] \) is \( \leq \) everything in \( b[k..\text{len}(b)–1] \)
  - \( b[0..k–1] \) is sorted (i.e., its values are in ascending order)
  - \( b[0..k–1] \) has \( n \) elements

- Put negative values before nonnegative ones.

  **Preconditions:**
  
  - Values in \( 0..n–1 \) are unknown

  **Postconditions:**
  
  - Everything in \( b[0..k–1] \) is \( \leq \) everything in \( b[k..\text{len}(b)–1] \)
  - \( b[0..k–1] \) is sorted (i.e., its values are in ascending order)
  - \( b[0..k–1] \) has \( n \) elements

Partition Algorithm

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

  **Preconditions:**
  
  - Sequence of \( 0..n–1 \) of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

  **Postconditions:**
  
  - Everything in \( b[h..k] \) is \( \leq \) everything in \( b[0..\text{len}(b)–1] \)
  - \( b[0..k] \) is sorted (i.e., its values are in ascending order)
  - \( b[0..k] \) has \( n \) elements

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

  **Preconditions:**
  
  - Values in \( 0..n–1 \) are unknown

  **Postconditions:**
  
  - Everything in \( b[h..k] \) is \( \leq \) everything in \( b[0..\text{len}(b)–1] \)
  - \( b[0..k] \) is sorted (i.e., its values are in ascending order)
  - \( b[0..k] \) has \( n \) elements

Partition Algorithm Implementation

```python
def partition(h, k):
    """Partition list \( b[h..k] \) around a pivot \( x = b[h] \).""
    i = h; j = k+1; x = b[h]
    # Invariant: \( b[0..i-1] < x \), \( b[i] = x \), \( b[j..k] >= x \)
    while i < j:
        while i < j:
            if b[i] < x:
                # Move to end of block
                x = b[i]; i += 1
            else:
                # Move to end of block
                x = b[j]; j -= 1
        if b[i] < x:
            # post: \( b[i] < x \) and \( b[j=1..k] >= x \)
            i += 1
        else:
            # post: \( b[i=1..j] <= x \) and \( b[j+1..k] >= x \)
            j -= 1
    return i
```
**Dutch National Flag Variant**

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

```
Dutch National Flag Algorithm

def dnf(b, h, k):
    # Returns: partition points as a tuple (i, j)
    t = h;
    i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return i, j
```

**Linear Search**

- Vague: Find first occurrence of v in b[h..k-1].
- Better: Store an integer in i to truthify result condition post:

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

**Binary Search**

- Vague: Look for v in sorted sequence segment b[h..k-1].
- Better:
  - Precondition: b[h..k-1] is sorted (in ascending order).
  - Postcondition: b[i] <= v and v < b[i+1..k-1]
- Below, the array is in non-descending order:

Call `binary_search` because each iteration of the loop cuts the array segment still to be processed in half.

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 repetend make progress?
4. Does the repetend keep inv true?