Recall: Horizontal Notation

<table>
<thead>
<tr>
<th>b</th>
<th>&lt;= sorted</th>
<th>&gt;=</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td></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[k..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.

Partition Algorithm Implementation

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

Dutch National Flag Algorithm

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

Dutch National Flag Variant

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

pre: b

post: b

inv: b

Partition Algorithm Implementation

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

Dutch National Flag Algorithm

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

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

```
Flag of Mauritius
```

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 repend make progress?
4. Does the repend keep the invariant inv true?

Linear Search
```
def linear_search(b, h, i):
    # Store in i the index of the first c in b[h..i]
    h = b
    # invariant: c is not in b[0..h-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 repend make progress?
4. Does the repend keep the invariant inv true?

Binary Search
- Vague: Look for v in sorted sequence segment b[h..k].
- Better:
  - 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:
```
def binary_search(b, h, k):
    # post: b[h..i] <= v and v < b[i+1..k]
    while h < i:
        m = (h + i) // 2
        if b[m] < v:
            h = m + 1
        else:
            i = m
    return i
```

Need two swaps for two spaces
See algorithms.py for Python code

Linear Search
```
def linear_search(b, h, k):
    pre: b
    post: i
```

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 repend make progress?
4. Does the repend keep the invariant inv true?

Binary Search
```
def binary_search(b, h, k):
    while h < k:
        m = (h + k) // 2
        if b[m] < v:
            h = m + 1
        else:
            k = m
    return h
```

Need two swaps for two spaces
See algorithms.py for Python code

Linear Search
```
def linear_search(b, h, k):
    pre: b
    post: i
```

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 repend make progress?
4. Does the repend keep the invariant inv true?