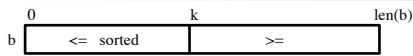
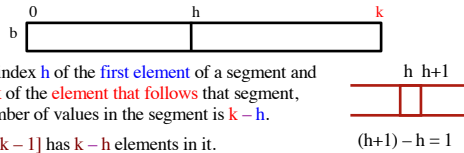


Recall: Horizontal Notation



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 $b[k..len(b)-1]$

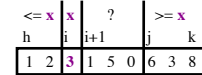


Given index h of the first element of a segment and index k of the element that follows that segment, the number of values in the segment is $k - h$.

$b[h .. k - 1]$ has $k - h$ elements in it.

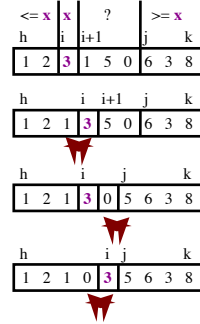
Partition Algorithm Implementation

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



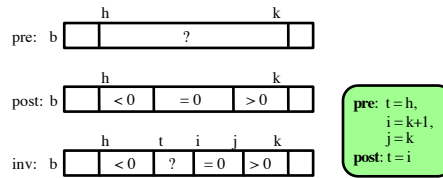
Partition Algorithm Implementation

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



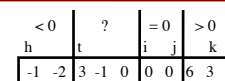
Dutch National Flag Variant

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



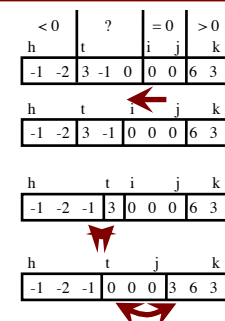
Dutch National Flag Algorithm

```
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] = 0, 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)
```



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] = 0, 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)
```



Flag of Mauritius

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

pre: b


?

post: b

<0 odd	<0 even	≥0 odd	≥0 even
--------	---------	--------	---------

inv: b

<0, o	<0, e	≥0, o	?	≥0, e
-------	-------	-------	---	-------



Flag of Mauritius

$<0, o$	$<0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1	-3	-2	-4	7 5
-5	-6	1	0	2 4

h	r	s	i	t k
-1	-3	-5	-4	7 5
-2	-6	1	0	2 4

Need two swaps for two spaces

Flag of Mauritius

$<0, o$	$<0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1	-3	-2	-4	7 5
-5	-6	1	0	2 4

h	r	s	i	t k
-1	-3	-4	-5	7 5
-2	-6	1	0	2 4

h	r	s	i	t k
-1	-3	-5	-4	7 5
-2	-6	1	0	2 4

See algorithms.py for Python code

Linear Search

pre: b

?

post: b

v not here	v	?
------------	---	---

OR

b

v not here

inv: b

v not here	?
------------	---

Linear Search

```
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: c is not in b[h..i-1]
    # 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 repetend make progress?
4. Does the repetend 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:** $b[h..i] \leq v$ and $v < b[i+1..k-1]$
- Below, the array is in non-descending order:

pre: b

?

post: b

<= v	> v
------	-----

inv: b

< v	?	> v
-----	---	-----

Called **binary search** because each iteration of the loop cuts the array segment still to be processed in half