Announcements for This Lecture

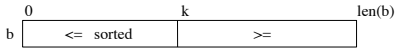
Assignment & Lab

- A6 is not graded yet
  - Done by end of classes
- A7 due **Wed, Dec. 10**
  - Wednesday after classes
  - Keep on top of milestones
  - Is your paddle moving?
- Lab Today: Office Hours
  - Get help on A7 paddle
  - Anyone can go to any lab

Next Week

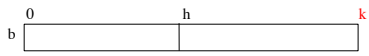
- Last Week of Class!
  - Finish sorting algorithms
  - Special final lecture
- Lab held, but is optional
  - More invariant practice
  - Also use lab time on A7
- Details about the exam
  - Multiple review sessions

Recall: Horizontal Notation



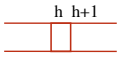
Example of an assertion about an sequence b. It asserts that:

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.

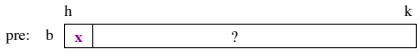
b[h .. k – 1] has k – h elements in it.



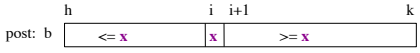
(h+1) – h = 1

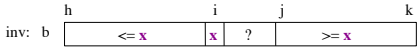
Partition Algorithm

- 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:

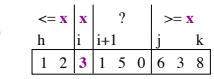




- Agrees with precondition when i = h, j = k+1
- Agrees with postcondition when j = i+1

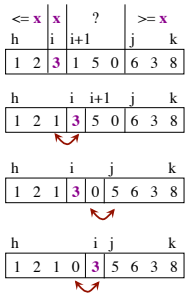
Partition Algorithm Implementation

```
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
```



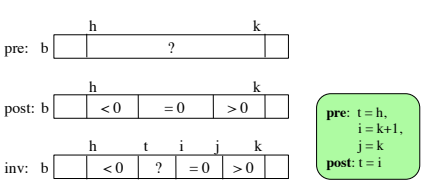
Partition Algorithm Implementation

```
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



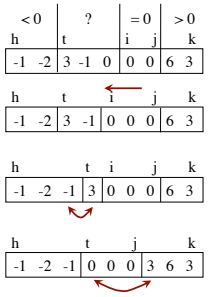
Dutch National Flag Algorithm

```
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] ?, 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)
```

< 0		?		= 0		> 0
h		t		i	j	k
-1	-2	3	-1	0	0	6 3

Dutch National Flag Algorithm

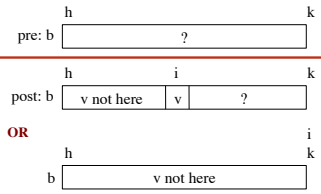
```
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
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



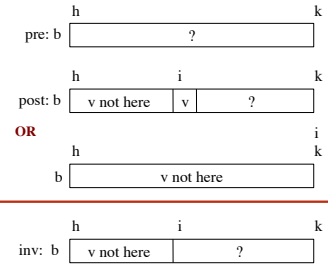
Linear Search

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

  post: 1. v is not in b[h..i-1]
  2. i = k OR v = b[i]



Linear Search



Linear Search

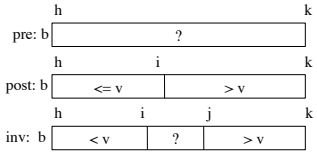
```
def linear_search(b,c,h):
    """Returns: first occurrence of c in b[h..]"""
    # Store in i the index of the first c in b[h..]
    i = h

    # 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] <= v and v < b[i+1..k-1]
- Below, the array is in non-descending order:



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