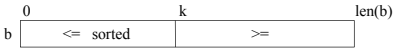
## Announcements

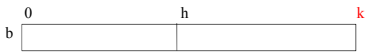
- Final Exam:
  - May 18th, 9am-11:30am
  - **Location**: Barton Hall Central and East
  - Final Exam conflicts are out
    - Watch email if you have not already heard
- Watch for Lab 13 coming out early
- A5 released over the weekend or next week
- No A6

## Horizontal Notation for Sequences




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

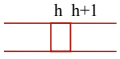
1. b[0..k–1] is sorted (i.e. its values are in ascending order)
2. Everything in b[0..k–1] is ≤ everything in b[k..len(b)–1]




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.

b[h .. h – 1] has 0 elements in it.



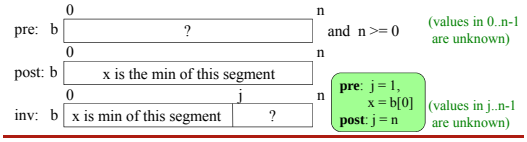

(h+1) – h = 1

## Developing Algorithms on Sequences

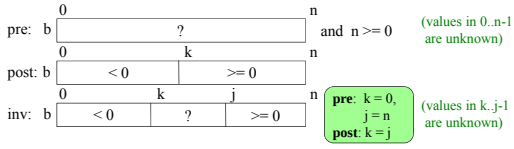
- Specify the algorithm by giving its precondition and postcondition as pictures.
- Draw the invariant by drawing another picture that "generalizes" the precondition and postcondition
  - The invariant is true at the beginning and at the end
- The four loop design questions
  1. How does loop start (how 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

- Find the minimum of a sequence.

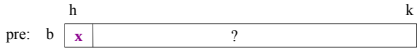



- Put negative values before nonnegative ones and return the split index.

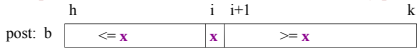


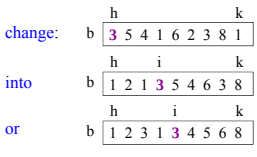

## Partition Algorithm

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




- Swap elements of b[h..k] and store in i to truthify post:

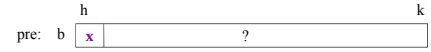


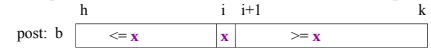

- x is called the pivot value
  - x is not a program variable
  - denotes value initially in b[h]

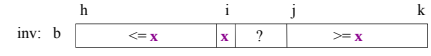
## Partition Algorithm

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




- Swap elements of b[h..k] and store in i to truthify post:

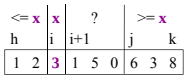



- Agrees with precondition when i = h, j = k+1
- Agrees with postcondition when j = i+1

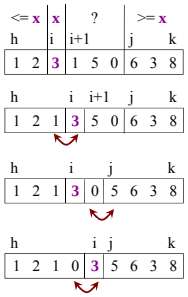
## Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]
       Returns: pivot index"""
    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
```



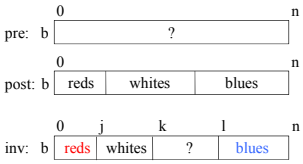

## Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]
       Returns: pivot index"""
    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
```




## 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



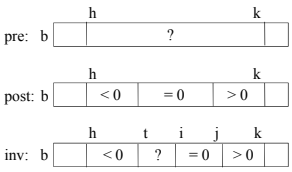

(values in 0..n-1 are unknown)

Make the red, white, blue sections initially **empty**:
- Range i..i-1 has 0 elements

Changing loop variables turns invariant into postcondition.

## Dutch National Flag Variant

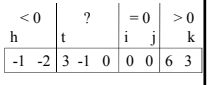
- Sequence of integer values
  - 'red' = negatives, 'white' = 0, 'blues' = positive
  - Only rearrange part of the list, not all




**pre**: t = h, i = k+1, j = k
**post**: t = i

## 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)
```




## 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)
```