

Horizontal Notation for Sequences

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

- $b[0..k-1]$ is sorted (i.e. its values are in ascending order)
- 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.

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
 - How does loop start (how to make the invariant true)?
 - How does it stop (is the postcondition true)?
 - How does the body make progress toward termination?
 - How does the body keep the invariant true?

2

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

pre:

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

Make the **red, white, blue** sections initially **empty**:

- Range $i..i-1$ has 0 elements
- Main reason for this trick: Changing loop variables turns invariant into postcondition.

post:

inv:

3

Generalizing Pre- and Postconditions

- Finding the minimum of a sequence.

pre:

and $n \geq 0$ (values in $0..n$ are unknown)

post:
- Put negative values before nonnegative ones.

pre:

and $n \geq 0$ (values in $0..n$ are unknown)

post:

inv:

pre: $j = 0$
post: $j = n$

(values in $j..n$ are unknown)

4

Partition Algorithm

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

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

post:

change:

into:

or:

x is called the pivot value

- x is not a program variable
- denotes value initially in $b[h]$

5

Partition Algorithm

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

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

post:

inv:

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

6

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(b,h,k), not partition(b[h:k+1])
Remember, slicing always copies the list!
We want to partition the **original** list

7

Partition Algorithm Implementation

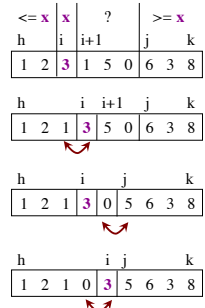
```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            _swap(b,i+1,j-1)
            j = j-1
   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

<= x	x	?	>= x
h	i	i+1	j
1	2	3	1 5 0
			6 3 8

8

Partition Algorithm Implementation

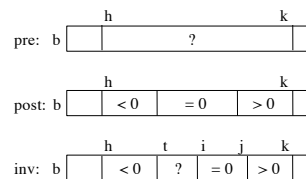
```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            _swap(b,i+1,j-1)
            j = j-1
   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



9

Dutch National Flag Variant

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



10

Dutch National Flag Algorithm

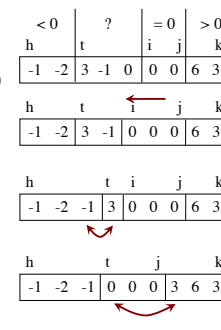
```
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1; j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[t-1] < 0:
            swap(b,t-1,t)
            t = t+1
        elif b[t-1] == 0:
            i = i-1
        else:
            swap(b,t-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
-1	-2	3	-1 0
		0 0	6 3

11

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[t-1] < 0:
            swap(b,t-1,t)
            t = t+1
        elif b[t-1] == 0:
            i = i-1
        else:
            swap(b,t-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)
```



12