Review 7

Required Algorithms

Algorithms on the Final

- **One** of these is on the final:
  - binary search
  - Dutch national flag
  - partition algorithm
  - insertion sort
  - selection sort
- Will be asked to write one
  - **Have** to know specifications
    And be able to **use** them.
  - **Develop** invariant from spec
  - **Develop** the loop from inv

- Reasons for this:
  1. Important algorithms.
  2. Forces you to think in terms of specifications.
  3. Forces you do learn to develop invariants.
  4. Forces you to learn to use the four loopy questions in reading/developing a loop
- Answer is **wrong** if it
  - Does not give the invariant
  - Does not use the invariant

Algorithms on the Final

- **One** of these is on the final:
  - binary search
  - Dutch national flag
  - partition algorithm
  - insertion sort
  - selection sort

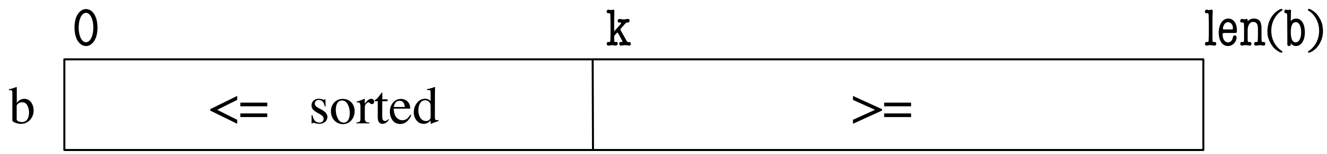
  Hardest

- Will be asked to write one
  - **Have** to know specifications
    And be able to **use** them.
  - **Develop** invariant from spec
  - **Develop** the loop from inv

- Reasons for this:
  1. Important algorithms.
  2. Forces you to think in terms of specifications.
  3. Forces you do learn to develop invariants.
  4. Forces you to learn to use the four loopy questions in reading/developing a loop

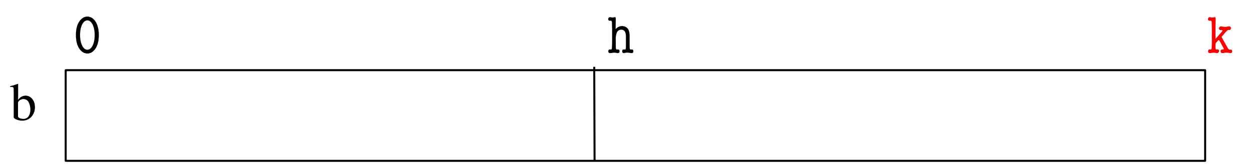
- Answer is **wrong** if it
  - Does not give the invariant
  - Does not use the invariant

Horizontal Notation for Sequences



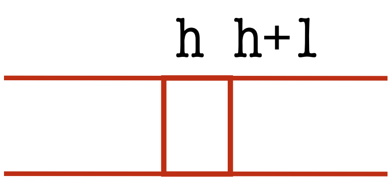
Example of an assertion about an sequence b. It asserts that:

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.

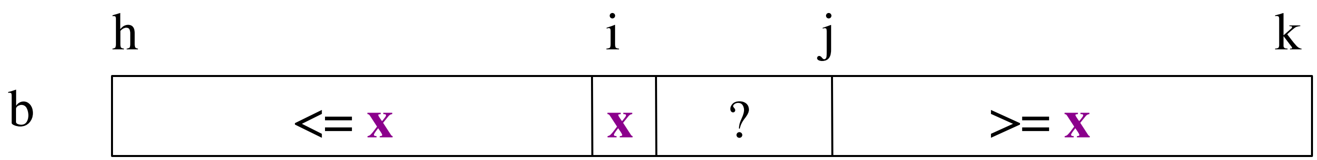
b[h .. k – 1] has k – h elements in it.



(h+1) – h = 1

DOs and DON'Ts #3

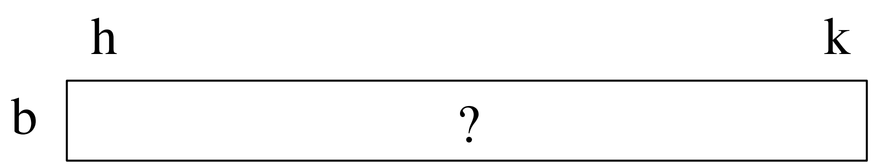
- DON'T put variables directly above vertical line.



  - Where is j?
  - Is it unknown or >= x?

Algorithm Inputs

- We may specify that the list in the algorithm is
  - b[0..len(b)-1] or
  - a segment b[h..k] or
  - a segment b[m..n-1]
- **Work with whatever is given!**



- Remember formula for # of values in an array segment
  - Following – First
  - e.g. the number of values in b[h..k] is k+1–h.

Binary Search

- **Vague:** Look for v in **sorted** segment b[h..k].
- **Better:**
  - Precondition: b[h..k] is sorted (in ascending order).
  - Postcondition: b[h..i-1] < v and v <= b[i..k]
- Below, the sequence is in non-descending order:

	h		k
pre: b	?		

	h	i	k
post: b	< v	>= v	

	h	i	j	k
inv: b	< v	?	>= v	

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

Dutch National Flag

- Tri-color flag represented by an list
  - Array of 0..n-1 of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

		0	n
pre:	b	?	

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

		0			n
post:	b	reds	whites	blues	

		0	j	k	l	n
inv:	b	reds	whites	?	blues	

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.

Invariants are Not Unique

- Invariants come from combining pre-, postconditions
  - Often more than one way to do it (see below)
  - **Do not memorize them**. Work them out on your own

binary search

	h	i	t	k
inv: b	< v	?	>= v	

Dutch National Flag

	0	h	k	m	n
inv: b	reds	?	whites	blues	

Partition Algorithm

- Given an segment b[h..k] with some value x in b[h]:

		h		k
pre:	b	**x**	?	

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

		h	i	i+1	k
post:	b	<= **x**	**x**	>= **x**	

		h								k
change:	b	**3**	5	4	1	6	2	3	8	1
		h			i					k
into	b	1	2	1	**3**	5	4	6	3	8
		h				i				k
or	b	1	2	3	1	**3**	4	5	6	8

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

Partition Algorithm

- Given an segment b[h..k] with some value x in b[h]:

pre: b

h		k
x	?	

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

post: b

h	i	i+1	k
<= **x**	**x**	>= **x**	

inv: b

h	i		j	k
<= **x**	**x**	?	>= **x**	

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

Insertion Sort AND Selection Sort

		0	n
pre:	b	?	

		0	n
post:	b	sorted	

Insertion Sort:

		0	i	n
inv:	b	sorted	?	

DO have to remember difference between the two sorting invariants

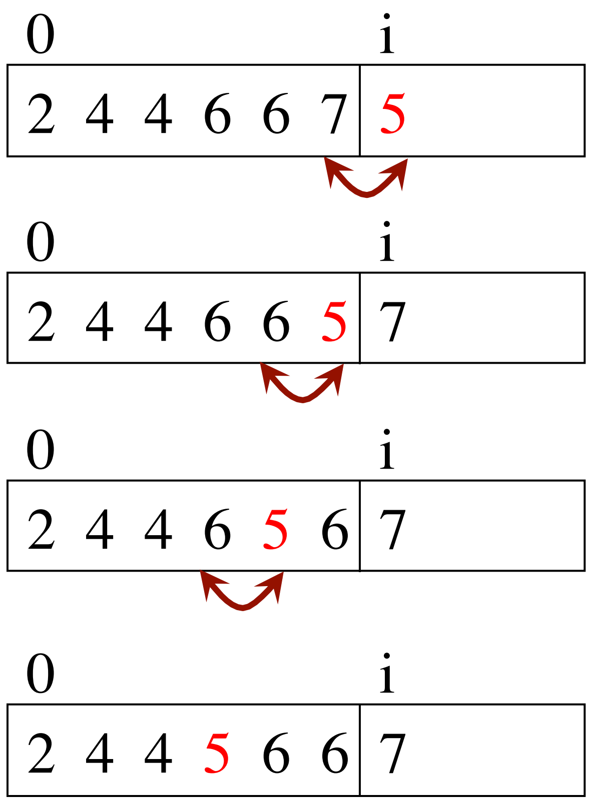
Selection Sort:

		0	i	n
inv:	b	sorted, ≤ b[i..]	≥ b[0..i-1]	

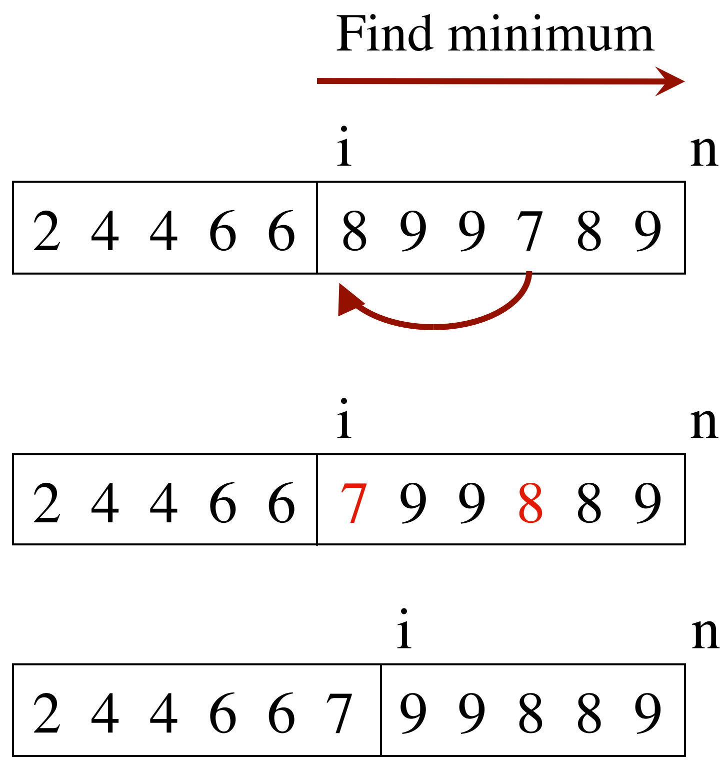
First segment always contains smaller values

Insertion Sort vs. Selection Sort

Insertion Sort



Selection Sort



Insertion Sort vs. Selection Sort

Insertion Sort

```
i = 0
while  i < n:
    pushdown(b,i)
    i = i +1

def pushdown(b, i):
    # inv: b[j] < b[j+1..i]
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

Invariant for inner loop

Selection Sort

```
i = 0
while i < n:
    j = minPos(b,i,n-1)
    swap(b,i,j)
    i = i+1

def minpos(b, h, k):
    """Returns: min position in b[h..k]"""
    # inv: ???
    ...
    # post: ???
```

Insertion Sort vs. Selection Sort

Insertion Sort

```
i = 0
while  i < n:
    pushdown(b,i)
    i = i +1

def pushdown(b, i):
    # inv: b[j] < b[j+1..i]
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

Invariant for inner loop

Selection Sort

```
i = 0
while i < n:
    j = minPos(b,i,n-1)
    swap(b,i,j)
    i = i+1

def minpos(b, h, k):
    """Returns: min position in b[h..k]"""
    # inv: b[x] is minimum of b[h..j]
    ...
    # post: b[x] is minimum of b[h..k]
```

A Word About Swap

- Almost all of these use the swap() function
  - Except for binarySearch
- You may or may not be given it on the exam
  - Should be familiar with it
  - Very easy to write

```
def swap(b, h, k):
    """Swaps b[h] and b[k] in b
    Pre: b is a mutable list, h and
    k are valid positions in b. """
    temp= b[h]
    b[h]= b[k]
    b[k]= temp
```

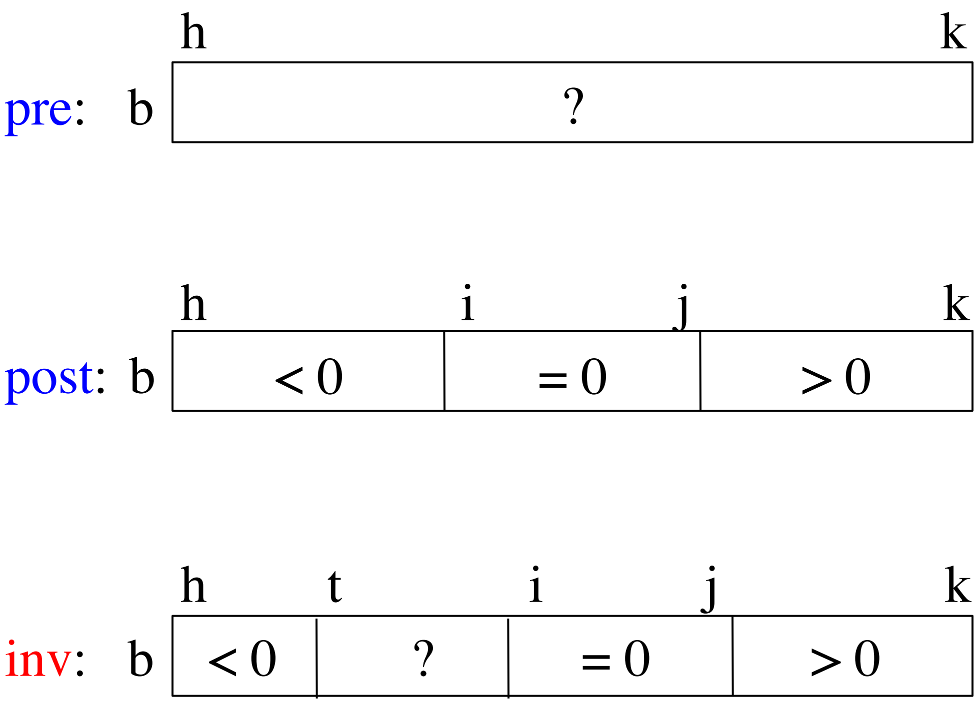
Dutch National Flag (Spring '11)

		h	k
pre:	b	?	

		h	i	j	k
post:	b	< 0	= 0	> 0	

```
def dutch_national_flag(b, h, k):
    """Use a Dutch National Flag algorithm to arrange the elements of b[h..k] and
    produce a tuple (i, j). Precondition and postcondition are given above."""
    ...
```

Dutch National Flag (Spring '11)



inv: b[h..t-1] < 0, b[t..i-1] unknown, b[i..j] = 0, and b[j+1..k] > 0

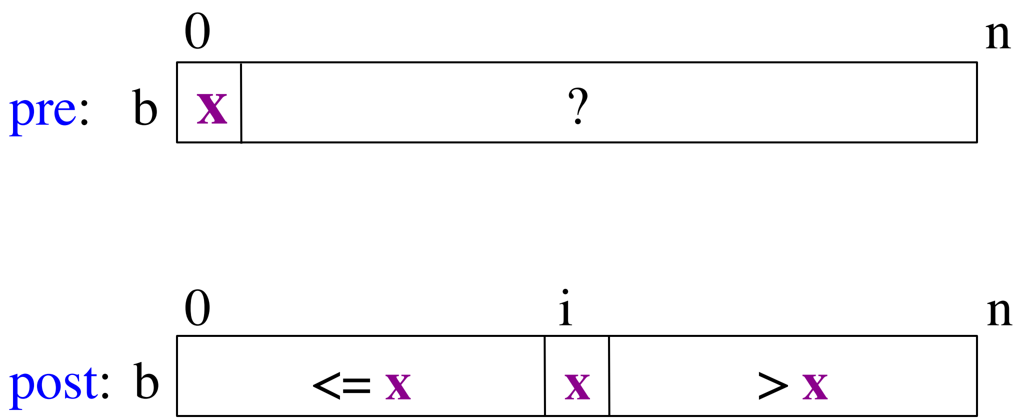
Dutch National Flag (Spring '11)

```
def  dutch_national_flag(b, h, k):
    """Use a Dutch National Flag algorithm to arrange the elements of b[h..k] and
    produce a tuple (i, j). Precondition and postcondition are given above."""
    t = h;    j = k;    i = k+1
    # inv: b[h..t-1] < 0, b[t..i-1] unknown, b[i..j] = 0, and b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b[i-1],b[t])
            t= t+1
        elif b[i-1] == 0:
            i= i-1
        else:
            swap(b[i-1],b[j])
            i= i-1; j= j-1
    return (i, j)
```

inv: b

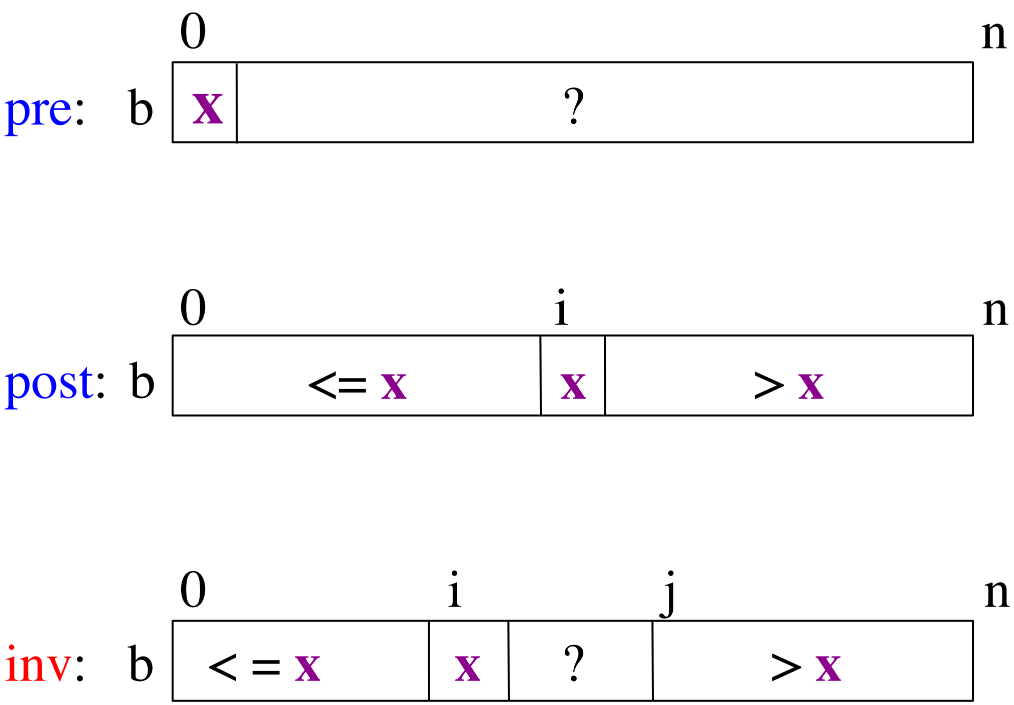
h	t	i	j	k
< 0	?	= 0	> 0	

Partition Algorithm Variant




```
def partition(b, n):
    """Partition the elements b[0..n-1] around pivot b[0]. Return position i.
    Precondition and postcondition are given above."""
    ...
```

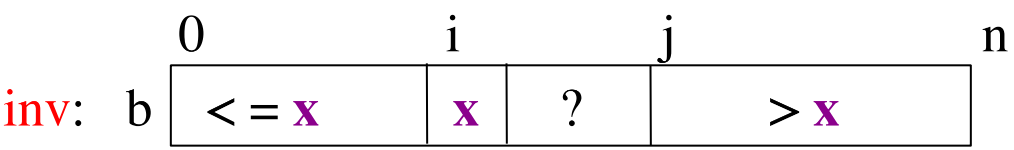
Partition Algorithm Variant



inv: b[0..i-1] <= x, b[i] = x, b[i+1..j-1] unknown, b[j..n-1] > x

Partition Algorithm Variant

```
def partition(b, n):
    """Partition list b[0..n-1] around a pivot x = b[0]"""
    i = 0; j = n; x = b[0]
    # invariant: b[0..i-1] <= x, b[i] = x, b[j..n-1] > 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[0..i-1] <= x, b[i] is x, and b[i+1..n-1] > x
    return i
```



Questions?