CS 1110:
Introduction to Computing Using Python

Lecture 22

Sequence Algorithms

[Andersen, Gries, Lee, Marschner, Van Loan, White]

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

Recall: Sorting

pre: b

0	n
?	

post: b

0	n
sorted	

inv: b

0	i	n
sorted, ≤ b[i..]	≥ b[0..i-1]	

First segment always contains smaller values

```
i = 0
while i < n:
    # Find minimum val in b[i..]
    # Swap min val with val at i
    i = i+1
```

					i					n
2	4	4	6	6	8	9	9	7	8	9

					i					n
2	4	4	6	6	7	9	9	8	8	9

						i				n
2	4	4	6	6	7	9	9	8	8	9

Box Notation for Sequences

	0	k	len(b)
b	<= sorted	>=	

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 $\le$ everything in b[k..len(b)–1]

	0	h	k
b			

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 h+1

(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 "moves from" the precondition to the 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.

x is the min of this segment (b[0])

pre: b — 0 … n: [| ?] and n >= 0 (values in 0..n-1 are unknown)

post: b — 0 … n: [x is the min of this segment]

inv: b — 0 … j … n: [x is min of this segment | ?] (values in j..n-1 are unknown)

pre: j = 1,
x = b[0]
post: j = n

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

pre: b — 0 … n: [?] and n >= 0 (values in 0..n-1 are unknown)

post: b — 0 … k … n: [< 0 | >= 0]

inv: b — 0 … k … j … n: [< 0 | ? | >= 0] (values in k..j-1 are unknown)

pre: k = 0,
j = n
post: k = j

Memory is Limited

- Memory was once *very* limited
- Attempts to use limited memory for multiple purposes led to famous video game bugs:

Pokemon Red and Blue

Pacman

Challenges for Today's Lecture

- Cannot create new lists – ***must swap in place***
- Assume you have a swap function:
  - swap(b, i, j) swaps elements at i and j

Time is Limited

- Some algorithms take more time
- Nesting loops in A3 made it slow

Challenges for Today's Lecture

- Cannot create new lists – ***must swap in place***
- Assume you have a swap function:
  - swap(b, i, j) swaps elements at i and j
- Go through sequence as few times as possible
  - Ideally just once!

Selection Sort

	0	n
pre: b	?	

	0	n
post: b	sorted	

	0	i	n
inv: b	sorted, $\leq$ b[i..]	$\geq$ b[0..i-1]	

First segment always contains smaller values

```
i = 0
while i < n:
    # Find minimum val in b[i..]
    # Swap min val with val at i
    i = i+1
```

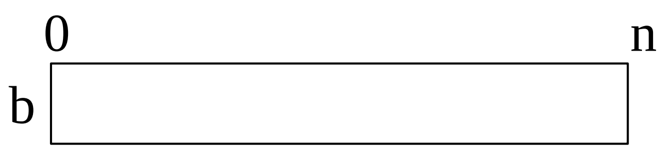
					i						n
2	4	4	6	6	8	9	9	7	8	9	

					i						n
2	4	4	6	6	7	9	9	8	8	9	

						i					n
2	4	4	6	6	7	9	9	8	8	9	

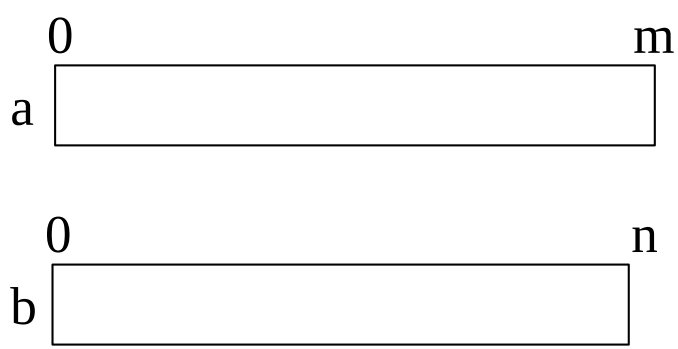
Algorithm Complexity

- Iterating through a sequence of length *n* requires *n* operations:




```
for x in b:
    # process x
```

- Nested loops multiply the # of operations:



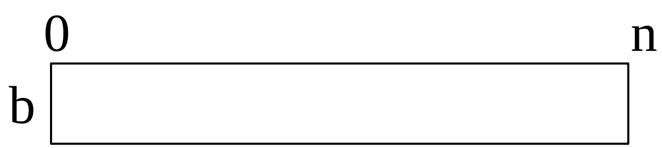

```
for x in a:
    for y in b:
        # process x and y
```

Requires *m***n* operations

Note: This slide was not in 9:05 lecture. Not on Final Exam.

Algorithm Complexity

- Nested loops over the same sequence also multiply # of operations:




```
for x in b:
    for y in b:
        # process x and y
```

Requires *n***n* operations

Note: This slide was not in 9:05 lecture. Not on Final Exam.

Complexity: Selection Sort

Finding the min value requires its own loop.

```
i = 0
while i < n:
    # Find minimum val in b[i..]
    # Swap min val with val at i
    i = i+1
```

How long does this take?

A: ~ n operations

B: ~ n^2 operations **CORRECT**

C: ~ n^3 operations

Note: This slide was not in 9:05 lecture. Not on Final Exam.

QuickSort

0 n

pre: b ?

0 n

post: b sorted

- Idea: Pick a *pivot* element x

We will just pick b[0]

- Partition sequence into <= x and >= x

h i i+1 k

post: b <= **x** | **x** | >= **x**

- Recurse on each partition

Partition Algorithm

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

	h		k
pre: b	**x**	?	

- Swap elements of b[h..k] and then store in i:

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

change:

	h							k
b	**3**	5	4	1	6	2	3	8	1

into

	h			i					k
b	1	2	1	**3**	5	4	6	3	8

- 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]:

	h		k
pre: b	**x**	?	

- Swap elements of b[h..k] and then store in i:

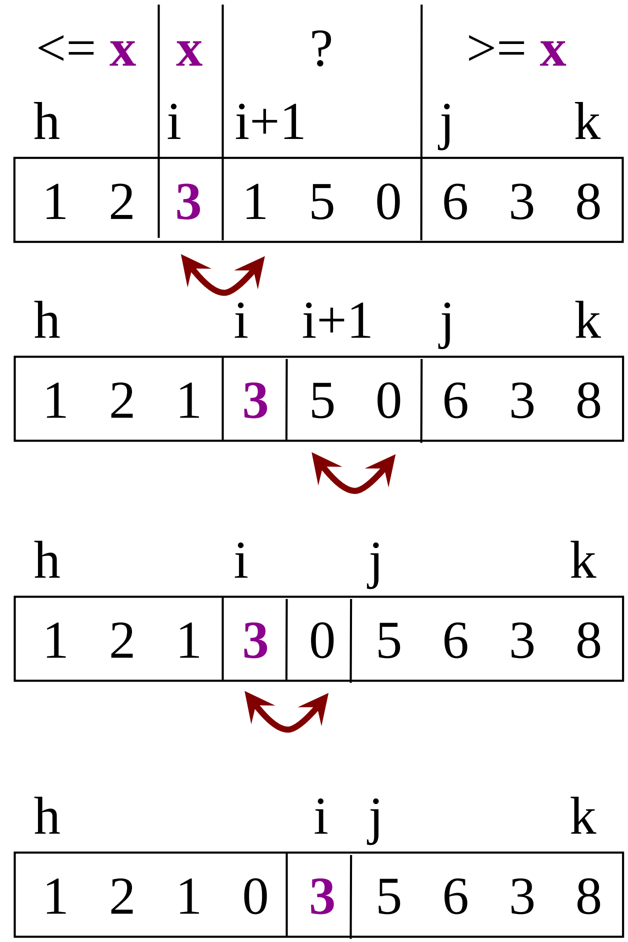
	h	i	i+1	k
post: b	<= **x**	**x**	>= **x**	

	h	i		j	k
inv: b	<= **x**	**x**	?	>= **x**	

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

Partition Algorithm Implementation

```python
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

	0		n
pre: b	?		

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

	0			n
post: b	reds	whites	blues	

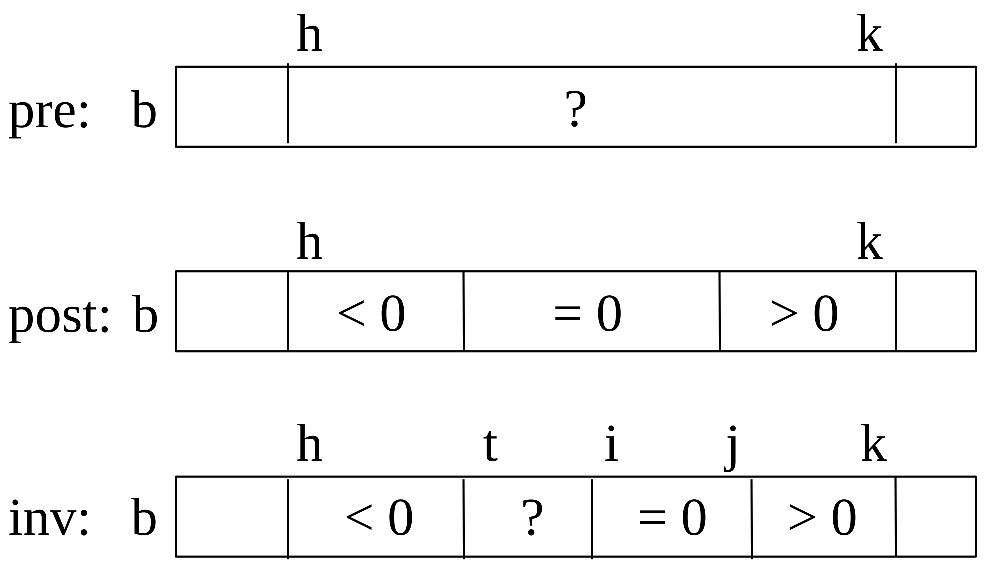
	0	j	k	l	n
inv: b	reds	whites	?	blues	

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.

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)
```