Lecture 27

Sorting

Announcements for This Lecture

Prelim/Finals

- Prelims in **handback room**
  - Gates Hall 216
  - See Piazza for hours
- **Final: Dec 7th 9:00-11:30am**
  - Study guide is posted
  - Announce reviews on Thurs.
- **Conflict with Final time?**
  - Submit to conflict to CMS **by this THURSDAY!**

This Week

- **Lab 13** is optional, final lab
  - Due day before final exam
  - Not part of mandatory 12
  - Best way to study for final
- But **Lab 12** is NOT optional
- **A7** is due **SUNDAY**
  - Extensions to Dec 8 possible
  - Have been granting if ask
  - S/U students get by default

Let's Talk about Assignment 6

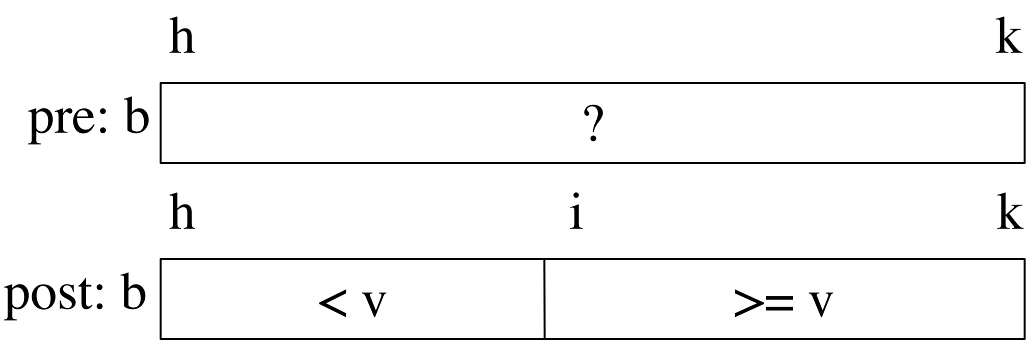
- An extensive redesign of a 2011 assignment
  - Last offered when class was very different
  - Back then majority were engineers & less of them
- We saw a **WIDE** variety of scores/difficulty
  - **Grades**: **Mean** 80, **Median** 84, **SDev** 15
  - **Time**: **Mean** 16.8 hrs, **Median** 15 hrs, **SDev** 7.3 hrs
- Most common rating: Pretty Good
  - But enough students hated to drop to Ok
  - Students who took longer rated lower

Binary Search

- **Vague:** Look for v in **sorted** sequence segment b[h..k].

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



Binary Search

- Look for value v in **sorted** segment b[h..k]

pre: b

h	k
?	

post: b

h	i	k
< v	>= v	

inv: b

h	i	j	k
< v	?	>= v	

New statement of the invariant guarantees that we get leftmost position of v if found

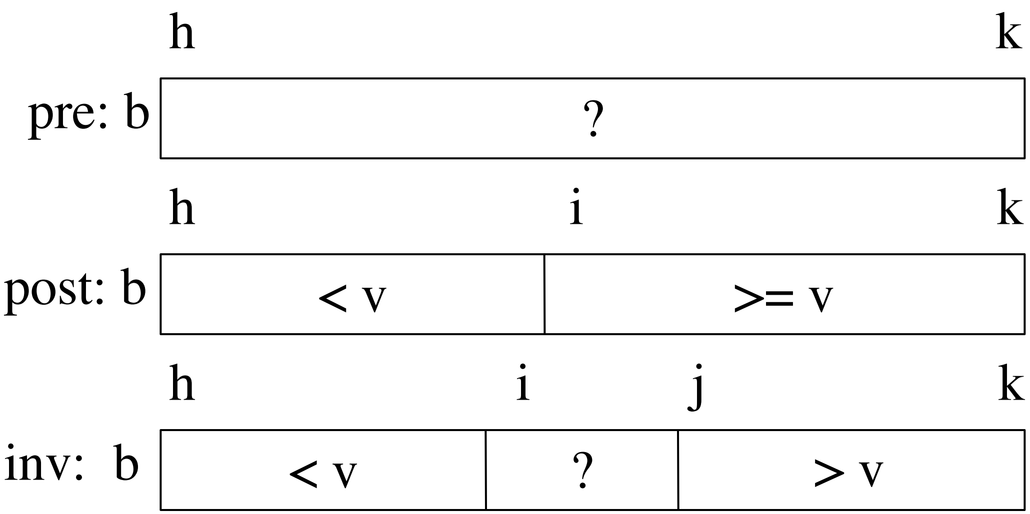
Example b

h									k
0	1	2	3	4	5	6	7	8	9
3	3	3	3	3	4	4	6	7	7

- if v is 3, set i to 0
- if v is 4, set i to 5
- if v is 5, set i to 7
- if v is 8, set i to 10

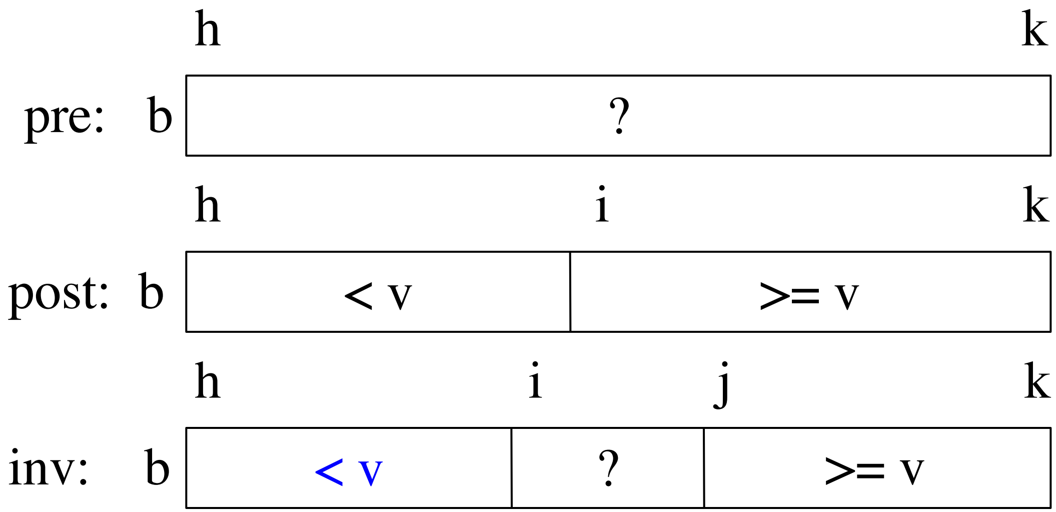
Binary Search

- **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-1] < v and v <= b[i..k]
- 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

Binary Search



New statement of the invariant guarantees that we get leftmost position of v if found

```
i = h; j = k+1;
while i != j:
```

Looking at b[i] gives linear search from left.

Looking at b[j-1] gives linear search from right.

Looking at middle: b[(i+j)/2] gives binary search.

Flag of Mauritius

- Now we have four colors!
  - Negatives: ‘red’ = odd, ‘purple’ = even
  - Positives: ‘yellow’ = odd, ‘green’ = even

	h		k	
pre: b		?		

	h				k	
post: b		< 0 odd	< 0 even	≥ 0 odd	≥ 0 even	

	h	r	s	i	t	k
inv: b		< 0, o	< 0, e	≥ 0, o	?	≥ 0, e

Flag of Mauritius

< 0, o		< 0, e		≥ 0, o		?				≥ 0, e	
h		r		s		i				t	k
-1	-3	-2	-4	7	5	-5	-6	1	0	2	4

h		r		s		i				t	k
-1	-3	-5	-4	7	5	-2	-6	1	0	2	4

One swap is not good enough

Flag of Mauritius

< 0, o		< 0, e		≥ 0, o		?				≥ 0, e	
h		r		s		i				t	k
-1	-3	-2	-4	7	5	-5	-6	1	0	2	4

h		r		s		i				t	k
-1	-3	-5	-4	-2	5	7	-6	1	0	2	4

Need two swaps for two spaces

Flag of Mauritius

< 0, o		< 0, e		≥ 0, o		?				≥ 0, e	
h		r		s		i				t	k
-1	-3	-2	-4	7	5	-5	-6	1	0	2	4

h			r		s		i			t	k
-1	-3	-5	-4	-2	5	7	-6	1	0	2	4

And adjust the loop variables

Flag of Mauritius

< 0, o	< 0, e	?	≥ 0, e
h	r=s	i	t k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4

h	r=s	i	t k
-1 -3 -7	-5 -2 -6	-4 1 0	2 4

BUT NOT ALWAYS!

Flag of Mauritius

< 0, o	< 0, e	?	≥ 0, e
h	r=s	i	t k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4

h	r=s	i	t k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4

BUT NOT ALWAYS!

Have to check if second swap is okay

Sorting: Arranging in Ascending Order

pre: b

0	n
?	

post: b

0	n
sorted	

Insertion Sort:

inv: b

0	i	n
sorted	?	

```
i = 0
while i < n:
    # Push b[i] down into its
    # sorted position in b[0..i]
    i = i+1
```

0						i
2	4	4	6	6	7	5

0						i
2	4	4	5	6	6	7

Insertion Sort: Moving into Position

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

def push_down(b, i):
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

swap shown in the lecture about lists

0						i
2	4	4	6	6	7	5

0						i
2	4	4	6	6	5	7

0						i
2	4	4	6	5	6	7

0						i
2	4	4	5	6	6	7

The Importance of Helper Functions

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

def push_down(b, i):
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

VS

Can you understand all this code below?

```
i = 0
while i < n:
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            temp = b[j]
            b[j] = b[j-1]
            b[j-1] = temp
        j = j -1
    i = i +1
```

Insertion Sort: Performance

```
def push_down(b, i):
    """Push value at position i into
    sorted position in b[0..i-1]"""
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
```

- b[0..i-1]: i elements
- Worst case:
  - i = 0: 0 swaps
  - i = 1: 1 swap
  - i = 2: 2 swaps
- Pushdown is in a loop
  - Called for i in 0..n
  - i swaps each time

Insertion sort is an n^2 algorithm

Total Swaps: 0 + 1 + 2 + 3 + … (n-1) = (n-1)*n/2

Algorithm "Complexity"

- **Given**: a list of length n and a problem to solve
- **Complexity**: *rough* number of steps to solve worst case
- Suppose we can compute 1000 operations a second:

Complexity	n=10	n=100	n=1000
n	0.01 s	0.1 s	1 s
n log n	0.016 s	0.32 s	4.79 s
n^2	0.1 s	10 s	16.7 m
n^3	1 s	16.7 m	11.6 d
2^n	1 s	4×10^{19} y	3×10^{290} y

Major Topic in 2110: Beyond scope of this course

Sorting: Changing the Invariant

pre: b

0	n
?	

post: b

0	n
sorted	

Selection Sort:

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 in b[i..]
    # Move it to position 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	

Sorting: Changing the Invariant

	0	n
pre: b	?	

	0	n
post: b	sorted	

Selection Sort:

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

First segment always contains smaller values

```
i = 0
while i < n:
    j = index of min of b[i..n-1]
    swap(b,i,j)
    i = i+1
```

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

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

Selection sort also is an n^2 algorithm

Partition Algorithm

- Given a list 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]

Sorting with Partitions

- Given a list 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	<= **y**	**y**	>= **y**	**x**	>= **x**	

Partition Recursively

Recursive partitions = sorting
- Called **QuickSort** (why???)
- Popular, fast sorting technique

QuickSort

```
def quick_sort(b, h, k):
    """Sort the array fragment b[h..k]"""
    if  b[h..k] has fewer than 2 elements:
        return
    j = partition(b, h, k)
    # b[h..j–1] <= b[j] <= b[j+1..k]
    # Sort b[h..j–1]  and  b[j+1..k]
    quick_sort (b, h, j–1)
    quick_sort (b, j+1, k)
```

- **Worst Case:**
  array already sorted
  - Or almost sorted
  - n^2 in that case
- **Average Case:**
  array is scrambled
  - n log n in that case
  - Best sorting time!

pre: b

h		k
x	?	

post: b

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

Final Word About Algorithms

- **Algorithm:**
  - Step-by-step way to do something
  - Not tied to specific language
- **Implementation:**
  - An algorithm in a specific language
  - Many times, not the "hard part"
- Higher Level Computer Science courses:
  - We teach advanced algorithms (pictures)
  - Implementation you learn on your own

List Diagrams

Demo Code