Lecture 19

Designing Array Algorithms

Announcements for Today

Reading

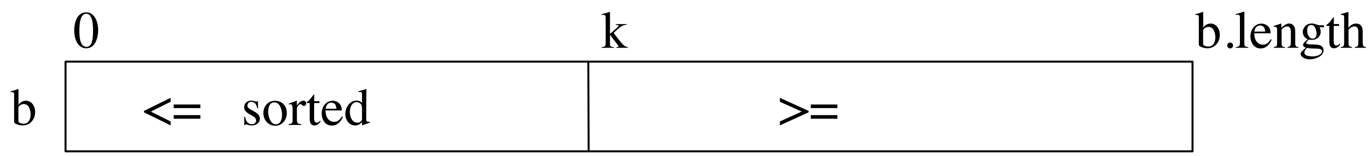
- Sections 8.1 – 8.3
- PLive Lessons 7.5, 7.6

- **Prelim, April 17th 7:30-9:30**
  - **TODAY IS LAST MATERIAL**
  - Review posted this weekend
  - Not the same as previous years
- **Conflict with Prelim time?**
  - Submit to Prelim 2 Conflict assignment on CMS
  - Do not submit if no conflict

Assignments

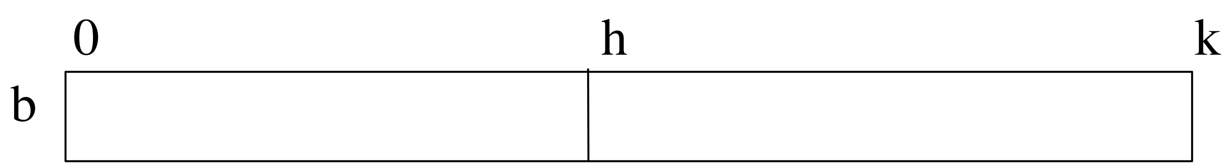
- A5 due tonight by Midnight
  - Will grade this weekend
  - Cannot give extensions
- A6 posted Tonight
  - Get started immediately!
  - Prelim is same week it is due
- Lab for this week & next
  - Made new lab at last minute
  - Original lab is next week
  - Will help with the prelim

Horizontal Notation for Arrays

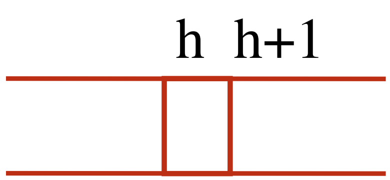


Example of an assertion about an array b. It asserts that:

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..b.length–1]



Given the index h of the First element of a segment and the index k of the element that Follows the 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

Developing Algorithms on Arrays

- 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 loopy questions (memorize them)
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does repetend make progress toward termination?
  4. How does repetend keep the invariant true?

Generalizing Pre- and Postconditions

- Dutch national flag: tri-color
  - 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.

Generalizing Pre- and Postconditions

- Finding the minimum of an array.

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

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

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

pre: j = 0
post: j = n

- Put negative values before nonnegative ones.

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

post: b [0 … k: < 0 | k … n: >= 0]

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

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

Partition Algorithm

- Given an array 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 array 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	i		j	k
inv: b	<= x	x	?	>= x	

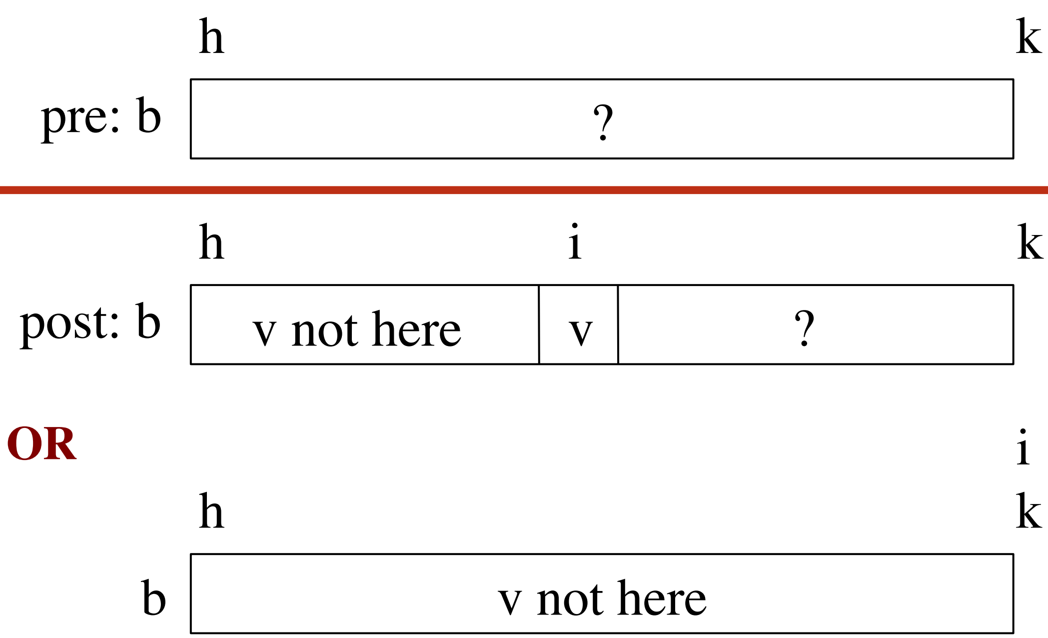
- Agrees with precondition when h = i, j = k+1
- Agrees with postcondition when j = i+1

Linear Search

- **Vague**: Find first occurrence of v in b[h..k-1].
- **Better**: Store an integer in i to truthify result condition post:

post: 1. v is not in b[h..i-1]

2. i = k OR v = b[i]



Linear Search

	h			k
pre: b	?			

	h	i		k
post: b	v not here	v	?	

OR

	h		i k
b	v not here		

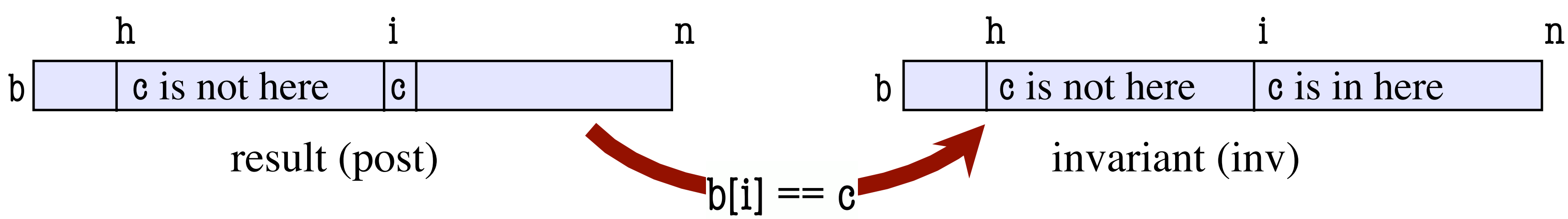
	h	i	k
inv: b	v not here	?	

Linear Search

```
/** Yields: index of first occurrence of c in b[h..]
 *   Precondition: c is in b[h..] */
public static int findFirst(int c, int[] b, int h) {
    // Store in i the index of the first c in b[h..]
    int i= h;
    // inv: c is not in b[h..i–1]
    while (b[i] != c) {
        i= i + 1;
    }
    // post: b[i] == c and c is not in b[h..i-1]
    return i;
}
```

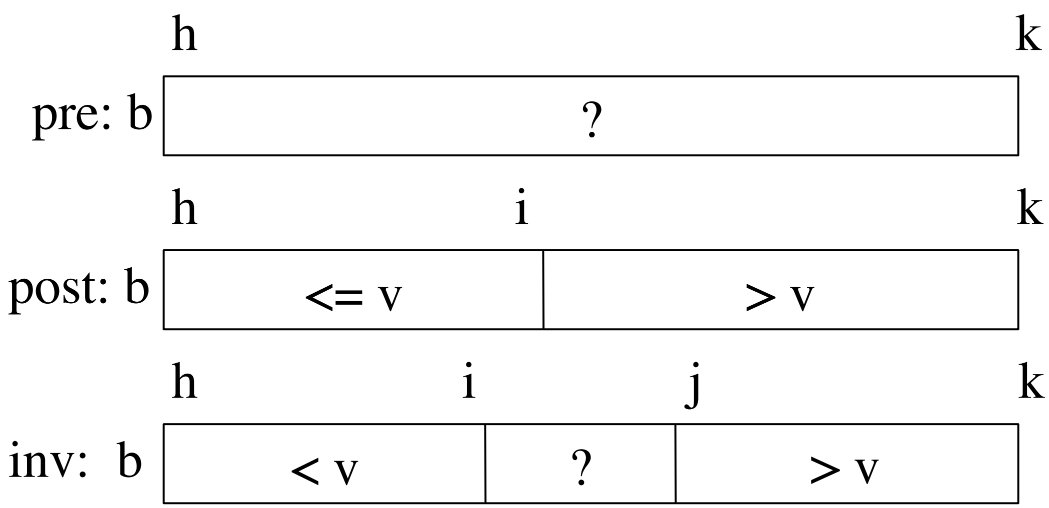
Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?



Binary Search

- **Vague:** Look for v in **sorted** array segment b[h..k].
- **Better:**
  - Precondition: b[h..k-1] is sorted (in ascending order).
  - Postcondition: b[h..i] <= v and v < b[i+1..k-1]
- 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

Loaded Dice

- Array p of length n represents n-sided die
  - Contents of p sum to 1
  - p[k] is probability die rolls the number k

1	2	3	4	5	6
0.1	0.1	0.1	0.1	0.3	0.3

weighted d6, favoring 5, 6

- Goal: Want to "roll the die"
  - Generate random number r between 0 and 1
  - Pick p[i] such that $p[i-1] < r \leq p[i]$

0.1	0.1	0.1	0.1	0.3	0.3
0.1	0.2	0.3	0.4	0.7	1.0

Loaded Dice

- **Want**: Value i such that p[i-1] < r <= [i]

	0		n
pre: b	?		

	0	i	n
post: b	r > sum	r <= sum	

	0	i	n
inv: b	r > sum	?	

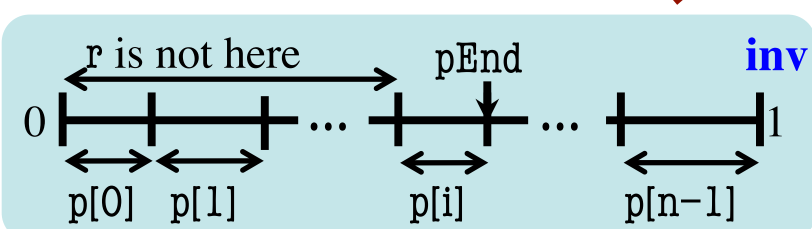
- Same as precondition if i = 0
- Postcondition is invariant + false loop condition

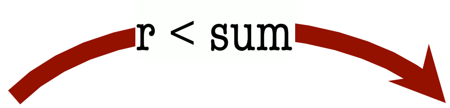
Loaded Dice

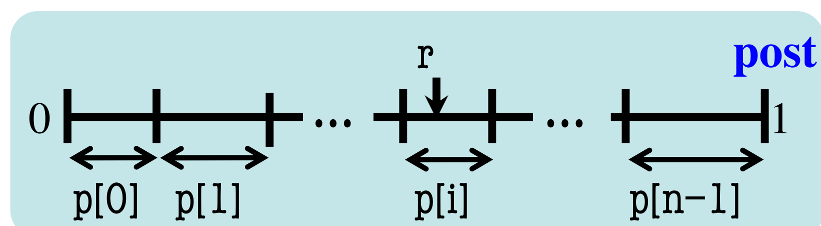
```
/**   Yields: a random int in 0..p.length-1; i is returned with probability p[i].
 *    Precondition: the entries of p are positive and sum to at least 1. */
public static int roll(double[] p) {
  double r= Math.random(); // r in [0,1)
  // Think of interval [0,1] as divided into segments of size p[i]
  // Store into i the segment number in which r falls.
  int i= 0;  double sum= p[0];
  // inv: r >= sum of p[0] .. p[i–1]; pEnd = sum of p[0] .. p[i]
  while ( r >= sum) {
    sum= sum + p[i+1];
    i= i + 1;
  }
  // post: sum of p[0] .. p[i–1] <= r < sum of p[0] .. p[i]
  return i;
}
```

Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?







Reversing an Array

h k

pre: b | not reversed |

h k

post: b | reversed |

h k

change: b | 1 2 3 4 5 6 7 8 9 9 9 9 |

h k

into b | 9 9 9 9 8 7 6 5 4 3 2 1 |

h i j k

inv: b | swapped | not reversed | swapped |