BINARY SEARCH AND LOOP INVARIANTS

Lecture 12A
CS2110 – Spring 2014

Develop binary search in sorted array b for v

pre: b — 0 [?] b.length

post: b — 0 [<= v | h > v] b.length

Example:

pre: b

0				4	5	6	7			b.length
2	2	4	4	4	4	7	9	9	9	9

If v is 4, 5, or 6, h is 5

If v is 7 or 8, h is 6

If v in b, h is index of rightmost occurrence of v.
If v not in b, h is index before where it belongs.

Develop binary search in sorted array b for v

pre: b — 0 [?] b.length

post: b — 0 [<= v | h > v] b.length

Better than Binary search in last lecture because it

(1) Finds not a random occurrence of v but the rightmost one. Useful in some situations

(2) If v is not in b, it gives useful information: it belongs between b[h] and b[h+1]

(3) Works also when array is empty!

Develop binary search in sorted array b for v

pre: b — 0 [?] b.length

Store a value in h to make this true:

post: b — 0 [<= v | h > v] b.length

Get loop invariant by combining pre- and post- conditions, adding variable t to mark the other boundary

inv: b — 0 [<= v | h ? | t > v] b.length

How does it start (what makes the invariant true)?

pre: b — 0 [?] b.length

inv: b — 0 [<= v | h ? | t > v] b.length

Make first and last partitions empty:

```
h= -1;  t= b.length;
```

When does it end (when does invariant look like postcondition)?

post: b — 0 [<= v | h > v] b.length

inv: b — 0 [<= v | h ? | t > v] b.length

```
h= -1;  t= b.length;
while ( h  !=  t-1 ) {

}
```

Stop when ? section is empty. That is when h = t-1.
Therefore, continue as long as h != t-1.

How does body make progress toward termination (cut ? in half) and keep invariant true?

7

inv:

	0	h	t	b.length
b	<= v	?	> v	

	0	h	e	t	b.length
b	<= v	?		> v	

```
h= -1;  t= b.length;
while ( h != t-1 ) {
    int  e= (h+t)/2;
}
```

Let e be index of middle value of ? Section. Maybe we can set h or t to e, cutting ? section in half

How does body make progress toward termination (cut ? in half) and keep invariant true?

8

inv:

	0	h	t	b.length
b	<= v	?	> v	

	0	h	e	t	b.length
b	<= v	?	?	> v	

	0	h	e	t	b.length
b	<= v	<= v	?	> v	

```
h= -1;  t= b.length;
while ( h != t-1 ) {
    int  e= (h+t)/2;
    if (b[e] <= v)  h= e;
}
```

If b[e] <= v, then so is every value to its left, since the array is sorted. Therefore, h= e; keeps the invariant true.

How does body make progress toward termination (cut ? in half) and keep invariant true?

9

inv:

	0	h	t	b.length
b	<= v	?	> v	

	0	h	e	t	b.length
b	<= v	?	?	> v	

	0	h	e	t	b.length
b	<= v	?	> v	> v	

```
h= -1;  t= b.length;
while ( h != t-1 ) {
    int  e= (h+t)/2;
    if (b[e] <= v)  h= e;
    else  t= e;
}
```

If b[e] > v, then so is every value to its right, since the array is sorted. Therefore, t= e; keeps the invariant true.

Loop invariants

10

We used the concept of a loop invariant in developing algorithms to reverse a linked list and do a binary search on a sorted array.

pre:

	0	b.length
b	?	

post:

	0	h	b.length
b	<= v	> v	

inv:

	0	h	t	b.length
b	<= v	?	> v	

Loop invariant: Important part of every formal system for proving loops correct.

11

Extremely useful tool in *developing* a loop. Create (first draft of) invariant from pre- and post-conditions, then develop the parts of the loop from precondition, postcondition, invariant.

pre:

	0	b.length
b	?	

post:

	0	h	b.length
b	<= v	> v	

inv:

	0	h	t	b.length
b	<= v	?	> v	

Loop invariant: Important part of every formal system for proving loops correct.

12

Invariant can be written in English, mathematics, diagrams, or mixtures of these. The important points are precision, clarity.

inv:

	0	h	t	b.length
b	<= v	?	> v	

inv: b[0..h] <= v < b[t..b.length-1]

inv: b[0..h] <= v < b[t..]

inv: everything in b[0..h] is at most v,
everything in b[t..] is greater than v

About notation b[h..k]. b[h..k] has k+1–h elements

13

[h..h+3]	4 elements	
[h..h+2]	3 elements	
[h..h+1]	2 elements	
[h..h]	1 element	
[h..h-1]	How many elements?	

h h+1 h+2 h+3

Use the formula: 0!

Convention: The notation b[h..k] is used only when h <= k+1.
For example, b[0..-2] is not allowed.

When h = k+1, b[h..k] denotes the empty segment starting at b[h].

Developing loop from pre, post, inv: 4 loopy questions

14

```
// pre
init
// inv
while ( b ) {
   // inv && b
   Ensure inv remains true;
   progress
  // inv
}
// inv  && ! b
// post
```

1. How does it start? What init makes invariant true?

2. When can it stop? Choose b so that inv && !b implies post

4. How do we make sure invariant is maintained?

3. How does body make progress toward termination?