BINARY SEARCH AND LOOP INVARIANTS

Lecture 12A
CS2110 – Spring 2014
Develop binary search in sorted array \( b \) for \( v \)

**pre:**

| \( b \) | \_
|---|---|
| 0 | \_
| \_ | ? |

**post:**

<table>
<thead>
<tr>
<th>( b )</th>
<th>_</th>
<th>&lt;= ( v )</th>
<th>&gt; ( v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( h )</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>( b )</th>
<th>_</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

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:**  
\[
\begin{array}{c|c|c|c}
0 & \ldots & \ldots & b.\text{length} \\
\hline
b \quad ?
\end{array}
\]

**post:**  
\[
\begin{array}{c|c|c|c}
0 & h & \ldots & b.\text{length} \\
\hline
b \quad \leq v & > v 
\end{array}
\]

**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

post: b 0 <= v h > v 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:**

| 0 | b | ? | b.length |

**inv:**

| 0 | h | t | b.length |

| <= v | ? | > v |

Make first and last partitions empty:

h = -1; t = b.length;
When does it end (when does invariant look like postcondition)?

```
inv: b
  0  h  t  b.length
  <= v ? > v

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

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?

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?

\[ h = -1; \ t = b.length; \]

\[ \text{while} \ (h != t-1) \ { \]
\[ \text{int} \ e = \frac{h+t}{2}; \]
\[ \text{if} \ (b[e] <= v) \ h = e; \]
\[ } \]

If \(b[e] \leq v\), 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 $b$ in half) and keep invariant true?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>h</th>
<th>t</th>
<th>b.length</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv:</td>
<td>$b$</td>
<td>$\leq v$</td>
<td>?</td>
<td>$&gt; v$</td>
</tr>
<tr>
<td></td>
<td>$0$</td>
<td>$h$</td>
<td>$e$</td>
<td>$t$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$\leq v$</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>$0$</td>
<td>$h$</td>
<td>$e$</td>
<td>$t$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$\leq v$</td>
<td>?</td>
<td>$&gt; v$</td>
</tr>
</tbody>
</table>

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

**while** ($h != t-1$) {
  **int** $e = (h+t)/2$;
  **if** ($b[e] \leq v$) $h = e$;
  **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.
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.

<table>
<thead>
<tr>
<th>pre:</th>
<th>post:</th>
<th>inv:</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>b 0 ?</td>
<td>inv: b &lt;= v ? &gt; v</td>
</tr>
<tr>
<td>post:</td>
<td>b 0 h &lt;= v</td>
<td>post: b &lt;= v ? &gt; v</td>
</tr>
<tr>
<td>post:</td>
<td>b h &lt;= v</td>
<td>post: b &lt;= v ? &gt; v</td>
</tr>
</tbody>
</table>
Loop invariant: Important part of every formal system for proving loops correct.

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.

<table>
<thead>
<tr>
<th>pre:</th>
<th>0</th>
<th>b.length</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>post:</th>
<th>0</th>
<th>h</th>
<th>b.length</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= v</td>
<td>&gt; v</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv:</th>
<th>0</th>
<th>h</th>
<th>t</th>
<th>b.length</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= v</td>
<td>?</td>
<td>&gt; v</td>
<td></td>
</tr>
</tbody>
</table>
Loop invariant: Important part of every formal system for proving loops correct.

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>h</th>
<th>t</th>
<th>b.length</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inv:</strong> b</td>
<td>&lt;= v</td>
<td>?</td>
<td>&gt; v</td>
<td></td>
</tr>
</tbody>
</table>

**inv:** \( b[0..h] \leq v < b[t..b.length-1] \)

**inv:** \( b[0..h] \leq v < b[t..] \)

**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

<table>
<thead>
<tr>
<th>Segment</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>[h..h+3]</td>
<td>4</td>
</tr>
<tr>
<td>[h..h+2]</td>
<td>3</td>
</tr>
<tr>
<td>[h..h+1]</td>
<td>2</td>
</tr>
<tr>
<td>[h..h]</td>
<td>1</td>
</tr>
<tr>
<td>[h..h-1]</td>
<td>How many elements?</td>
</tr>
</tbody>
</table>

Use the formula: 0!

Convention: The notation \(b[h..k]\) is used only when \(h \leq k+1\).

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

// 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
3. How does body make progress toward termination?
4. How do we make sure invariant is maintained?