Develop binary search in sorted array \( b \) for \( v \)

\[
\begin{array}{c|c|c}
\text{pre:} & \text{b} & \text{b.length} \\
\hline
0 & ? & \\
\hline
\text{post:} & \text{b} & \text{b.length} \\
\hline
0 & h & > v \\
\end{array}
\]

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!

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

\[
\begin{array}{c|c|c|c}
\text{pre:} & \text{b} & \text{b.length} \\
\hline
0 & ? & \\
\hline
\text{inv:} & \text{b} & \text{b.length} \\
\hline
0 & h & t & \leq v & ? & > v \\
\end{array}
\]

Make first and last partitions empty:
\( h = -1; \ t = \text{b.length}; \)

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

\[
\begin{array}{c|c|c|c}
\text{post:} & \text{b} & \text{b.length} \\
\hline
0 & h & t & \leq v & ? & > v \\
\hline
\text{inv:} & \text{b} & \text{b.length} \\
\hline
0 & h & t & \leq v & ? & > v \\
\end{array}
\]

\( h = -1; \ t = \text{b.length}; \)
\textbf{while} ( \ h != \ t-1 ) { 
\textbf{stop when} ? \textbf{section is empty. That is when} \ h = t-1. 
\textbf{Therefore, continue as long as} \ h != t-1. 
}
How does body make progress toward termination (cut ? in half) and keep invariant true?

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

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.

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

Loop invariants

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.

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.

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

<table>
<thead>
<tr>
<th>$b[h..h+3]$</th>
<th>4 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b[h..h+2]$</td>
<td>3 elements</td>
</tr>
<tr>
<td>$b[h..h+1]$</td>
<td>2 elements</td>
</tr>
<tr>
<td>$b[h..h]$</td>
<td>1 element</td>
</tr>
<tr>
<td>$b[h..h-1]$</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

1. How does it start? What init makes invariant true?
2. When can it stop? Choose $b$ so that $\text{inv} \&\& \neg b$ implies $\text{post}$
3. How does body make progress toward termination?
4. How do we make sure invariant is maintained?

// pre  
// inv
while (!b) {
    // inv && b
    // post && !b
    // inv remains true: progress
}