Recursive Linear Search

Look at each element of array in range [first, last) until the target is found or run out of elements. Return index of target or last.

Analysis (same as non-recursive linear search)

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>array of values</td>
<td>int[ ]</td>
<td>varying</td>
<td>received</td>
<td>arr</td>
</tr>
<tr>
<td>index of lower bound</td>
<td>int</td>
<td>varying</td>
<td>received</td>
<td>first</td>
</tr>
<tr>
<td>index of upper bound</td>
<td>int</td>
<td>varying</td>
<td>received</td>
<td>last</td>
</tr>
<tr>
<td>search target</td>
<td>int</td>
<td>varying</td>
<td>received</td>
<td>target</td>
</tr>
<tr>
<td>index of target</td>
<td>int</td>
<td>varying</td>
<td>returned</td>
<td>——</td>
</tr>
</tbody>
</table>

Algorithm

1. If first = last then (base case: target not found)
   1.1 Return last

2. If arr[first] = target then (base case: target found)
   2.1 Return first

3. (inductive step: search with range [first+1,last))
   Return RecLinSearch (arr, first+1, last, target)

Code

// Precondition: first <= last
int RecLinearSearch (const int arr[], int first, int last, int target) {
    if (first == last) // base case: target not found
        return last;
    if (arr[position] == target) // base case: target found
        return position;
    // inductive step: search with range [first+1, last)
    return RecLinearSearch (arr, first+1, last, target)
} // end RecLinearSearch