Proofs by induction:

1) Write a (good) template.

2) Think big. Use arbitrary (possibly large) values of $n$. Reduce to a smaller value if you can.

3) Look for holes (base cases).
4) Rewrite everything so it's easier to read & follow.

Don't:
- assume only for \( k = n - 1 \)
- just assume \( n + \) prove for \( n + 1 \)
Recursion:

Reduce large instances of some problem $A$ to smaller instances of the same problem $A$.

If you can't reduce, solve the base case directly.

(don't need to be too clever, usually)
Mergesort von Neumann '45

Input: An array A[1..n] of things to sort by pairwise comparisons (integers, characters, etc.)

1) Divide array into two subarrays of roughly equal size.

2) Recursively merge sort the two subarrays. (magic)

3) Merge the two sorted subarrays quickly.

If \( n \leq 1 \), do nothing instead.

<table>
<thead>
<tr>
<th>Input:</th>
<th>S O R T I N G E X A M P L E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide:</td>
<td>S O R T I N G | E X A M P L E</td>
</tr>
<tr>
<td>Recurse Left:</td>
<td>I N O R S T | G E X A M P L E</td>
</tr>
<tr>
<td>Recurse Right:</td>
<td>I N O R S T A E G L M P X</td>
</tr>
<tr>
<td>Merge:</td>
<td>A E G I L M N O P R S T X</td>
</tr>
</tbody>
</table>
Merge:

1) Use smaller first member of the two halves
   \[ \text{sorted!} \]

2) Recursively sort what remains of the subarrays, fewer elements total, so this works (by induction)

\[ AC[1..m] + A[m+1..] \text{ are sorted} \]

\[
\text{MERGE}(A[1..n], m):
\]
\[
i \leftarrow 1; \quad j \leftarrow m + 1
\]
\[
\text{for } k \leftarrow 1 \text{ to } n
\]
\[
\text{if } j > n
\]
\[
B[k] \leftarrow A[i]; \quad i \leftarrow i + 1
\]
\[
\text{else if } i > m
\]
\[
B[k] \leftarrow A[j]; \quad j \leftarrow j + 1
\]
\[
\text{else if } A[i] < A[j]
\]
\[
B[k] \leftarrow A[i]; \quad i \leftarrow i + 1
\]
\[
\text{else}
\]
\[
B[k] \leftarrow A[j]; \quad j \leftarrow j + 1
\]
\[
\text{for } k \leftarrow 1 \text{ to } n
\]
\[
A[k] \leftarrow B[k]
\]

\[
\text{MERGE}(A[1..n], m):
\]
\[
\text{MERGE}(A[1..m]) \quad \text{ (Recurse!)}
\]
\[
\text{MERGE}(A[m+1..n]) \quad \text{ (Recurse!)}
\]
\[
\text{MERGE}(A[1..n], m)
\]
Correctness:

Thm: Assuming that 
Merge\( (B[1..l], k) \) sorts 
\( B \) if \( B[1..k] + B(k+1..l] \)
are sorted, \( \text{MergeSort}(A[1..n]) \)
sorts \( A \).

(\text{notation note: } C[n+1..n] \text{ is empty})

Proof: Assume \( \text{MergeSort}(D[1..0]) \)
sorts \( D \) whenever \( o < n \).
- if \( n \in \mathbb{N} \), \( A \) is sorted \( \checkmark \)
- o.w., \( m < n \) \( \Rightarrow \) \( n-(m+1)+1 < n \)

\( \uparrow \) otherwise
By assumption (IH)

\[
\text{Merge Sort} + (A(1..m)) \text{ sort} \\
\text{Merge Sort} + (A(m+1..n)) \text{ sort}
\]

Merge does merge the now-sorted subarrays. ✓
Quick sort
Hoare '59

1) Choose a pivot element from the array.
2) Partition array into 3 subarrays stored in this order:
   1) Elements smaller than pivot.
   2) Just the pivot
   3) Elements larger than pivot.
3) Recursively sort 1) and 3).
**QuickSort**($A[1..n]$):

if ($n > 1$)

Choose a pivot element $A[p]$
$r \leftarrow \text{Partition}(A, p)$
QuickSort($A[1..r - 1]$) \text{ (Recurse!)}
QuickSort($A[r + 1..n]$) \text{ (Recurse!)}

\text{Partition}(A[1..n], p):

swap $A[p] \leftarrow A[n]$

$l \leftarrow 0$ \text{ ([#items < pivot])}

for $i \leftarrow 1$ to $n - 1$

if $A[i] < A[n]$

$l \leftarrow l + 1$

swap $A[l] \leftarrow A[i]$

swap $A[n] \leftarrow A[l + 1]$

return $l + 1$
Divide-and-conquer

1) "Divide" given instance to create one or more independent smaller instances of the same problem.

2) Delegate smaller instances to Recursion Fairy.

3) Combine solutions for smaller instances.

If instance cannot be divided solve as a base case.