Proofs by induction:

1) Write a (good) template.

Theorem: $P(n)$ for every positive integer n.

Proof by induction: Let n be an arbitrary positive integer.
Assume that $P(k)$ is true for every positive integer $k < n$.
There are several cases to consider:

- Suppose n is *... blah blah blah ...*
  Then $P(n)$ is true.
- Suppose n is *... blah blah blah ...*
  The inductive hypothesis implies that *... blah blah blah ...*
  Thus, $P(n)$ is true.

In each case, we conclude that $P(n)$ is true. □

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 its 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.)

Goal: Rearrange A's elements so $A[1] \leq A[2] \leq \dots \leq A[n]$

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.

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Divide:	S	O	R	T	I	N	G	E	X	A	M	P	L
Recurse Left:	I	N	O	R	S	T	G	E	X	A	M	P	L
Recurse Right:	I	N	O	R	S	T	A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	R	S	T	X

If $n \leq 1$, do nothing instead.

Merge:

1) Use smaller first member of the two halves

sorted!

2) Recursively sort what remains of the subarrays.

- fewer elements total
- so this works (by induction)

assumes $A[1..m]$ & $A[m+1..n]$ are sorted!

```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])          ⟨⟨Recurse!⟩⟩
      MERGESORT(A[m+1..n])        ⟨⟨Recurse!⟩⟩
      MERGE(A[1..n], m)
```

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i+1
      else if i > m
          B[k] ← A[j]; j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i+1
      else
          B[k] ← A[j]; j ← j+1

  for k ← 1 to n
      A[k] ← B[k]
```

Correctness:

Thm: Assuming that Merge$(B[1..\ell], k)$ sorts B if $B[1..k]$ & $B(k+1..\ell]$ are sorted, MergeSort$(A[1..n])$ sorts A.

(notation note: $C[n+1..n]$ is empty)

Proof: Assume MergeSort$(D[1..o])$ sorts D whenever $o < n$.

- if $n = 1$, A is sorted ✓
- o.w., $m < n$ & $n-(m+1)+1 < n$

otherwise

By assumption (IH)

MergeSort(A[1..m]) &

MergeSort(A[m+1..n]) sort

Merge does merge the now-sorted subarrays. ✓

Quicksort
Hoare '59

1) Choose a <u>pivot</u> element from the array.

2) Partion 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) + 3).

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Choose a pivot:	S	O	R	T	I	N	G	E	X	A	M	$\boxed{P}$	L
Partition:	A	G	O	E	I	N	L	M	$\boxed{P}$	T	X	S	R
Recurse Left:	A	E	G	I	L	M	N	O	$\boxed{P}$	T	X	S	R
Recurse Right:	A	E	G	I	L	M	N	O	$\boxed{P}$	R	S	T	X

takes old index of pivot & returns new index

```
QuickSort(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← Partition(A, p)
      QuickSort(A[1..r − 1])      ⟨⟨Recurse!⟩⟩
      QuickSort(A[r + 1..n])      ⟨⟨Recurse!⟩⟩
```

```
Partition(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                    ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n − 1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ 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.