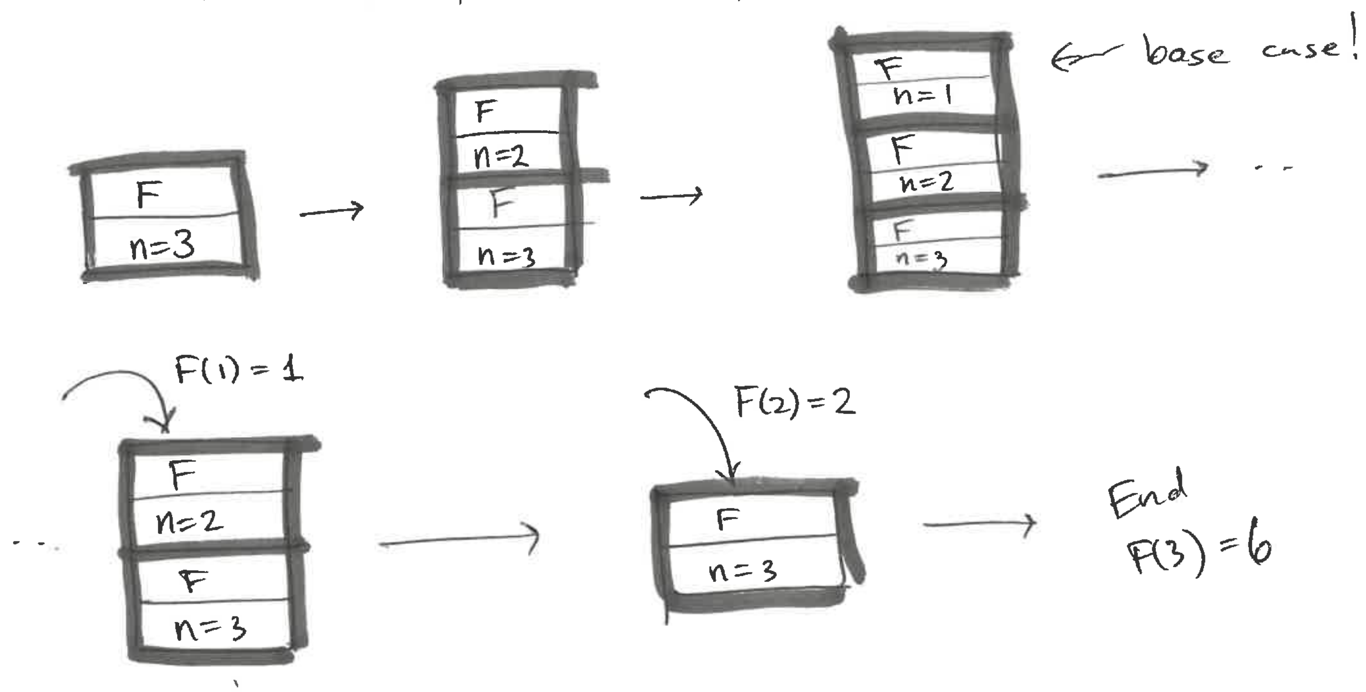
Let's look at factorial2(3). Now called F(3).



Let's look at a different recursive function.
We'll look at a recursive function to determine if a string is a palindrome, that is, the string is the same if read from left to right or right to left.

(palindrome code)

Let's come back to binary search and rewrite that using a recursive function.

(binary search recursive)

Is recursion just a confusing gimmick?

No, but one can "remove" recursion by while loops. The point is that recursion takes the challenge of understanding the entire algorithm to two, often much simpler, pieces: base step and recursion.

Recursion also forms the basis of a technique called "Divide and Conquer."

Although the binary search recursive algorithm divides, I would not call it a DaC approach. DaC is a design where NOT SUPER PRECISE the algorithm divides or splits the problem into smaller and smaller problems until it solves the problem ~~on a small~~ (essentially) directly on trivial input. The solutions are then recombined to solve the original problem.

Analysis of DaC algorithms often requires

1. Mathematical induction
2. Solving recurrence relations.

We will look at two sorting algorithms that use DaC design and are much more efficient than SelectionSort and BubbleSort.

Quick Sort.

At a high level, quick sort partitions the list into three pieces. It does this by choosing a pivot, which is some entry in the list — we'll discuss how to do this soon. With the pivot, the list is split:

$$L = [\text{entries} < \text{pivot}]$$

$$R = [\text{entries} > \text{pivot}].$$

We recursively apply quick sort to both sub-lists. Merging the results is trivial:

$$\text{quicksort}(L) + [\text{pivots}] + \text{quicksort}(R).$$

How to find the pivot? This is important because if too large or small one of L or R is too small and we will not have effectively "divided" — more ~~like~~ like shaved a few terms away.

But we also cannot spend much computational resources here. Let's find a pivot in $O(1)$-time, and then see how we might relax this.

Common approaches:

1. First entry
2. last entry

For lists already sorted (common use case). This is terrible!

3. middle entry
4. random

Fairly reasonable for long lists.

Note: we could select a constant number of entries to look at and ~~the~~ compute the median of the selection.

There is a deep analysis of this process because of the ubiquity of sorting and quick sort in particular. We will stick with one of the first 3 (first, middle, or last). For complexity analysis, it does not matter which we choose. In otherwords, whatever "worst" input we have for a first-pivot can be rearranged to be the worst input for middle-pivot or last-pivot.

The worst input is where the pivot does effectively nothing: either L or R is empty. Let's consider R empty. This implies that the maximal element is the first element of our list. Applying this recursively means that the worst input is a list sorted in a descending sequence.

Suppose n is the length of the starting list, called A. The call stack looks something like:

quicksort(A) ← n comparisons
quicksort(A[1:]) ← $n-1$ comparisons
" " (A[2:]) ← $n-2$ " "

⋮

" " (A[n-1:]) ← base case

We perform
$$\sum_{i=1}^{n} i = O(n^2)$$
comparisons, so this algorithm runs in $O(n^2)$ time.

In the worst-case, quick sort is not asymptotically better than selection sort or bubble sort. However, ~~on average~~ its average-case complexity is $O(n \log n)$. [Thm 11.3] in Open Data Structures.

Let's stay with quick sort a little longer.
Let's use Mathematical Induction to prove the algorithm is correct.

Mathematical Induction:

Suppose we have a statement (a sentence that is either True or False) indexed by the natural numbers.

Ex: n is a non-neg. int: $n \le n^2$. (True)

Ex: " ": $n < \sqrt{n}$. (False)

Ex: " ": $n + 10 \le n^2$. (Either is possible)

How does it work? We have 2 statements to prove:

- Base case: $n = 0$
- Inductive step: assume statement is true for n, then show the statement is true for $n+1$.

If both are proven, then we conclude that the statement is true for all natural numbers!

Ex: Take "$n \le n^2$".

Base case: $0 \le 0^2$. Obvious!

Inductive: Suppose $n \le n^2$. We need to show $n+1 \le (n+1)^2$.

$$\begin{aligned}(n+1)^2 &= n^2 + 2n + 1\\ &\geq n + 2n + 1\\ &= 3n+1\\ &\geq n+1.\end{aligned}$$

This proves the inductive step. Hence, we can conclude that $n \leq n^2$ for all non-neg. integers n. ☺

One more example:

~~Lem~~ Proposition: Let F_n be the n^{th} Fibonacci number with $F_0 = 0$ and $F_1 = 1$. Let

$$\varphi = \tfrac{1}{2}(1+\sqrt{5}) \quad \text{and} \quad \psi = \tfrac{1}{2}(1-\sqrt{5}).$$

Then

$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}.$$

Proof: We will show this by induction.

Base case(s): $\dfrac{\varphi^0 - \psi^0}{\varphi-\psi} = \dfrac{1-1}{\varphi-\psi} = 0$, which is F_0.

~~Inductive Step: $F_{n+2} = F_{n+1} + F_n$~~

$$\frac{\varphi^1 - \psi^1}{\varphi - \psi} = 1 = F_1.$$

<u>Inductive Step</u> We assume that for some $n \geq 1$,

$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi}, \quad F_{n-1} = \frac{\varphi^{n-1} - \psi^{n-1}}{\varphi - \psi}.$$

We need to show that

$$\overbrace{\frac{\varphi^{n+1} - \psi^{n-1}}{\varphi - \psi}}^{\text{"}F_{n+1}\text{"}} = \overbrace{\frac{\varphi^n - \psi^n}{\varphi - \psi}}^{F_n} + \overbrace{\frac{\varphi^{n-1} - \psi^{n-1}}{\varphi - \psi}}^{F_{n-1}}$$

We have

$$\frac{\varphi^n - \psi^n + \varphi^{n-1} - \psi^{n-1}}{\varphi - \psi} = \frac{\varphi^{n-1}(1+\varphi) - \psi^{n-1}(1+\psi)}{\varphi - \psi}.$$

Using the quadratic formula, the quadratic $x^2 - x - 1$ has roots φ and ψ. Thus

$$0 = \varphi^2 - \varphi - 1 = \varphi^2 - (1+\varphi).$$

Hence $\varphi^2 = 1 + \varphi$. Similarly $\psi^2 = 1 + \psi$. Thus,

$$\varphi^{n-1}(1+\varphi) = \varphi^{n+1}$$

similarly with ψ. Thus, we're done. $\square$.

Can we prove that quicksort solves our SORT problem? Yes.

<u>Base Case:</u> ~~Empty lists are~~ Lists of length at most 1 are by definition sorted.

<u>Induction:</u> Suppose both L and R are sorted.

Then L + [pivots] + R is sorted since each entry of L is less than the pivot and each entry of R is greater than the pivot.

Let's analyze worst case behavior by recurrence relations: Let $T(n)$ be the runtime of quicksort on a list of length n. Suppose we have the worst input. Then we have

$$T(n) = O(n) + T(n-1).$$

comparison in the first iteration (when the list has length n). In the next iteration we have

$$T(n-1) = O(n-1) + T(n-2),$$

and so on.

$$T(n) = O(n) + T(n-1) = O(n) + O(n-1) + T(n-2).$$

$$= \sum_{i=1}^{n} O(i). = O(n^2).$$

What if most inputs had good pivots — that is, pivots were always roughly the median? Then we would expect the following timing:

$$T(n) = n + 2 \cdot T\left(\tfrac{n}{2}\right).$$

$$= n + 2\left(\tfrac{n}{2} + 2T\left(\tfrac{n}{4}\right)\right) = 2n + 4T\left(\tfrac{n}{4}\right)$$