Quick Sort

Divide & Conquer

The Great Farm Plot Conundrum

Imagine you're Farmer Bob. You've got this massive, oddly-shaped farm plot that you want to divide into the biggest possible square plots, so you can grow the finest, square-shaped carrots ever! 🥕 But you don't want to waste any space, and you want each plot to be as big as possible. What do you do?

Step 1: Understand the Base Case

The base case is like the ultimate "easy button" for your problem. For our farm, let's say the plot is just a small square of land. The answer is obvious here: you don't need to divide it anymore because it's already a perfect square plot! So, in this case, the largest square plot size is the size of the whole farm.

Step 2: Divide or Decrease

Now, if your farm is not a perfect square and instead is a bit more complicated (like a rectangular plot), you need to use D&C to figure out how to divide it into those big square plots.

1. **Measure Up:** Start by measuring the dimensions of your rectangular farm plot. Suppose it's 20 meters by 15 meters.
2. **Square Size Showdown:**
    - Try to fit the largest possible square in one of the dimensions. In this case, you want to find the biggest size of the square that can fit both dimensions perfectly.
    - To find the largest square size, you need to figure out the Greatest Common Divisor (GCD) of the two dimensions. The GCD will give you the side length of the largest square that can fit evenly into both dimensions.
3. **Recursive Splitting:** If you're not sure how to calculate the GCD, you can use the divide-and-conquer approach to find it:
    - **Base Case:** If you're down to a single number (when the two dimensions are equal), that's your GCD.

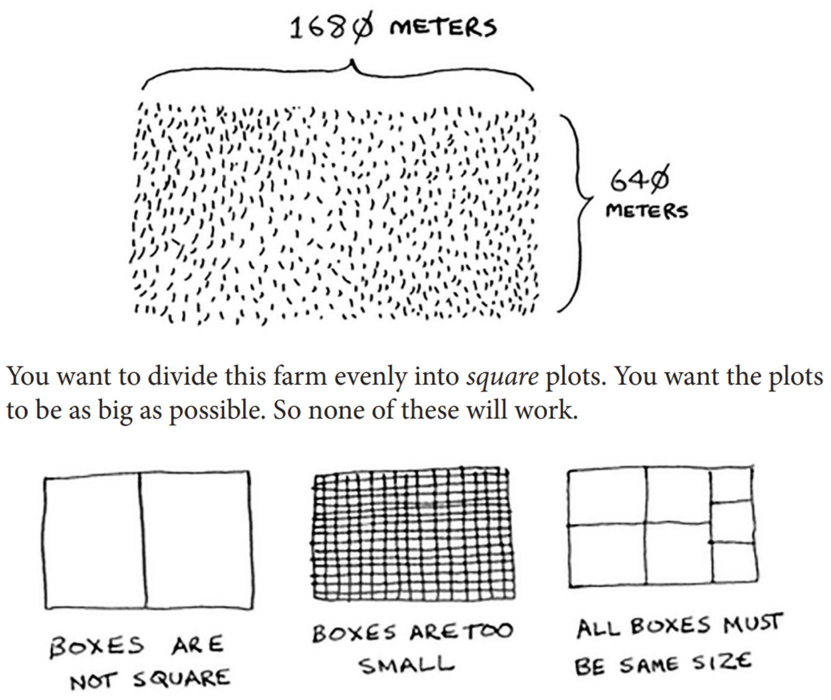
- **Divide and Conquer:** Otherwise, divide the problem into smaller problems. For instance, you can use the Euclidean algorithm, which repeatedly replaces the larger number with its remainder when divided by the smaller number, until you get to the base case.

Example

Imagine you have a farm plot of 20 meters by 15 meters. You want to find the largest square plot size.

1. **Find the GCD of 20 and 15:**
    - Divide 20 by 15. The remainder is 5.
    - Now, find the GCD of 15 and 5:
        - 15 divided by 5 has a remainder of 0.
        - So, the GCD is 5 meters.
2. **Divide Your Farm:**
    - You can now divide your farm into squares that are 5 meters by 5 meters.
    - For a 20×15 plot, you'll get 4 rows and 3 columns of 5×5 square plots.

Voilà! You've successfully used Divide and Conquer to figure out the largest possible square plot size for your farm. Your farm is now perfectly squared away! 🌾



You want to divide this farm evenly into *square* plots. You want the plots to be as big as possible. So none of these will work.

Imagine Farmer Bob has acquired a new, larger rectangular farm plot measuring 1680 × 640 meters. He wants to divide it into the largest possible square plots. Let's walk through the process step by step:

1. **Initial Division:** You can fit two 640 × 640 boxes in the 1680 × 640 farm, leaving a 400 × 640 segment.
2. **The "Aha!" Moment:** Instead of stopping here, apply the same algorithm to the remaining 640 × 400 segment. This reduces the problem from a 1680 × 640 farm to a 640 × 400 farm!
3. **Continuing the Process:**
    - With the 640 × 400m farm, the largest square you can create is 400 × 400 m.
    - This leaves a 400 × 240 m segment.
    - From this, you can create a 240 × 240 m square, leaving a 240 × 160 m segment.
    - Continue this process until you reach the base case.

4. **Reaching the Base Case:** You'll eventually reach a point where one dimension is a factor of the other. In this case, you'll end up with an 80 × 160 m segment, where 80 is a factor of 160.

Final Result: The largest square plot size for the original 1680 × 640 m farm is 80 × 80 m.

This example demonstrates how the Divide and Conquer approach systematically reduces the problem size until it reaches a solvable base case, effectively finding the optimal solution for complex scenarios.

> **Note on Euclid's Algorithm:** This process is closely related to Euclid's algorithm for finding the Greatest Common Divisor (GCD). While the proof is beyond the scope of this explanation, it's worth noting that this method guarantees finding the largest possible square size that evenly divides both dimensions of the original rectangle.

By applying this Divide and Conquer strategy, Farmer Bob can efficiently divide his oddly-shaped farm into the largest possible square plots, maximizing his square-shaped carrot production! 🥕🚜

Example

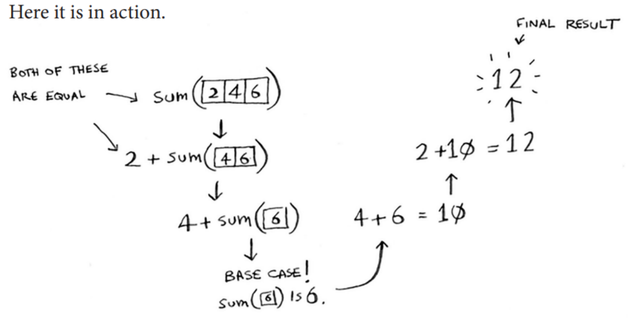
$$\text{sum}\left(\boxed{2}\boxed{4}\boxed{6}\right) = 12$$

It's the same as this.

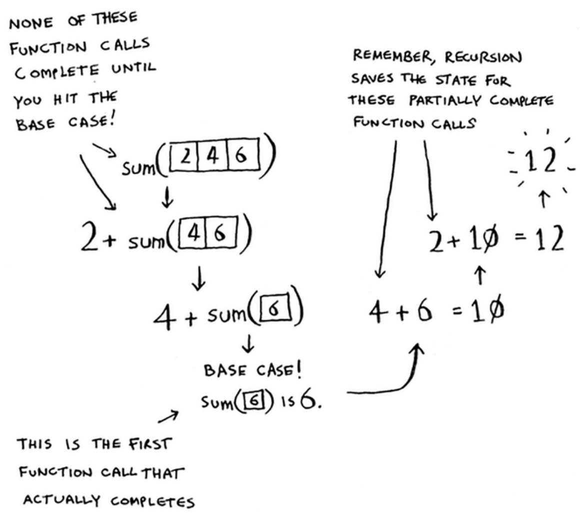
$$2 + \text{sum}\left(\boxed{4}\boxed{6}\right) = 2 + 10 = 12$$

> **Tip:** When you're writing a recursive function involving an array, the base case is often an empty array or an array with one element. If you're stuck, try that first.

Here it is in action.



Remember, recursion keeps track of the state.


NONE OF THESE FUNCTION CALLS COMPLETE UNTIL YOU HIT THE BASE CASE!
sum([2|4|6])
2 + sum([4|6])
4 + sum([6])
BASE CASE! sum([6]) IS 6.
THIS IS THE FIRST FUNCTION CALL THAT ACTUALLY COMPLETES
REMEMBER, RECURSION SAVES THE STATE FOR THESE PARTIALLY COMPLETE FUNCTION CALLS
4 + 6 = 10
2 + 10 = 12
12

Sneak peak at functional programming

Imagine you're at a magical party where loops don't show up—yep, that's right, no loops allowed. Instead, the star of the show is recursion, and everyone's having a blast! In this party, recursion is like that super enthusiastic friend who insists on doing everything in a creative, repetitive way.

Let's look at a fun example: the classic "sum of a list" function, which is like adding up all the snacks at the party.

Recursion Dance-Off

First, let's understand the two moves in our recursion dance-off:

1. **Base Case Boogie**:
   This is the part where the party is over and there are no more snacks to add.

In programming terms, this is when our list of snacks (or numbers) is empty. So, we simply say “0” because there’s nothing left to sum up.

```
sum [] = 0
```

2. **Recursive Rumba**:
   When there are snacks (or numbers) at the party, we take one snack (or number) and then call up our recursion friend to deal with the rest. We keep doing this until we hit the Base Case Boogie. So, it’s like saying, “I’ll handle this snack and then let my recursion buddy handle the rest!”

   ```
   sum (x:xs) = x + (sum xs)
   ```

   Here, `x` is the snack we’re handling right now, and `xs` is the rest of the party (the remaining snacks).

If-Statement Salsa

You can also do a bit of an “if-statement salsa” to achieve the same thing:

```
sum arr = if arr == []
 then 0
 else (head arr) + (sum (tail arr))
```

In this version, if the list of snacks (`arr`) is empty, we simply do the Base Case Boogie and return `0`. Otherwise, we grab the first snack (`head arr`), and pass the rest of the snacks (`tail arr`) to our recursive buddy.

Why Recursion Rocks

Recursion is like a never-ending dance routine where each step is a smaller version of the last one. It’s super cool because it keeps everything neat and tidy, especially when loops aren’t invited to the party. Plus, Haskell, our magical language, makes recursion super easy with its elegant and readable style.

So, if you’re ready to groove to a new beat and join the recursion party, Haskell is waiting with open