Part 1: Two Fibonaccis

$fib(n)$:
  if $n = 0$ or $n = 1$ **then**
    return n
  else
    return $fib(n-1) + fib(n-2)$
  end if

We can state a recurrence for this algorithm:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + O(1) \\ &\geq fib(n-1) + fib(n-2) \\ &= fib(n) \end{aligned}$$

$fib2(n)$:
  if $n = 0$ **then**
    return 0
  end if
  create array $f[0..n]$
  $f[0] \leftarrow 0, f[1] \leftarrow 1$
  for $i \leftarrow 2$ to n **do**
    $f[i] \leftarrow f[i-1] + f[i-2]$
  end for
  return $f[n]$

Addition of two numbers in the preceding algorithm takes constant time until the values exceed the maximum value that can be stored in a word. After which, we need to consider how values of arbitrary length are added.

Part 2 - Integer Multiplication

Let us consider an integer X which is composed of X_L which are the leftmost bits of X, and X_R which are the rightmost bits of X.

$$X = X_L | X_R$$

We can multiply integers X, Y as follows:

$$\begin{aligned} XY &= (2^{n/2}X_L + X_R)(2^{n/2}Y_L + Y_R) \\ &= 2^n X_L Y_L + 2^{n/2} X_L Y_R + 2^{n/2} X_R Y_L + X_R Y_R \\ &= 2^n X_L Y_L + 2^{n/2}(X_L Y_R + X_R Y_L) + X_R Y_R \end{aligned}$$

Which gives the recurrence

$$\begin{aligned} T(n) &= 4T(n/2) + O(n) \\ &\leq 4T(n/2) + cn \\ &\leq 4(4T(n/4) + cn/2) + cn \\ &\leq 4(4(4T(n/8) + cn/4) + cn/2) + cn \\ &\leq 64T(n/8) + cn(1 + 2 + 4) \\ &... \\ &\leq 4^i T(n/2^i) + cn(1 + 2 + ... + 2^{i-1}) \end{aligned}$$

Where i is the number of times we can divide n by 2, or $log_2 n$.

$$\begin{aligned} T(n) &\leq 4^{log_2 n} T(n/2^{log_2 n}) + cn(1 + 2 + ... + 2^{log_2 n - 1}) \\ &\leq n^{log_2 4} T(n/n^{log_2 2}) + cn \sum_{i=0}^{log_2 n - 1} 2^i \\ &\leq n^2 T(1) + cn 2^{log_2 n} \\ &\leq n^2 T(1) + cn n^{log_2 2} \\ &\leq n^2 T(1) + cn^2 \\ &\leq n^2 (T(1) + c) \\ &\leq n^2 (O(1) + c) \\ &\leq O(n^2) \end{aligned}$$

Can we do better? Turns out: yes.

We need: $X_L Y_L, X_R Y_R$, and $X_L Y_R + X_R Y_L$

Observe:

$$\begin{aligned} &(X_L + X_R)(Y_L + Y_R) - X_L Y_L - X_R Y_R \\ &= X_L Y_L + X_R Y_L + X_L Y_R + X_R Y_R - X_L Y_L - X_R Y_R \\ &= X_R Y_L + X_L Y_R \end{aligned}$$

Since we must compute $X_L Y_L$ and $X_R Y_R$ anyway, this saves us an entire multiplication. Reducing our recurrence from $T(n) = 4T(n/2) + O(n)$ to $T(n) = 3T(n/2) + O(n)$.

We can solve this new recurrence as follows:

$$\begin{aligned} T(n) &= 3T(n/2) + O(n) \\ &\leq 3T(n/2) + cn \\ &\leq 3(3T(n/4) + cn/2) + cn \\ &\leq 3^i T(n/2^i) + cn(1 + 3/2 + ... + (3/2)^{i-1}) \\ &\leq 3^{log_2 n} T(n/2^{log_2 n}) + cn(1 + 2 + ... + (3/2)^{log_2 n - 1}) \\ &\leq n^{log_2 3} T(1) + cn \sum_{i=0}^{log_2 n - 1} (3/2)^i \\ &\leq n^{log_2 3} T(1) + cn(3/2)^{log_2 n} \\ &\leq n^{log_2 3} T(1) + cn n^{log_2 (3/2)} \\ &\leq n^{log_2 3} T(1) + cn n^{log_2 3 - log_2 2} \\ &\leq n^{log_2 3} T(1) + cn n^{log_2 3 - 1} \\ &\leq n^{log_2 3} T(1) + cn n^{log_2 3} n^{-1} \\ &\leq n^{log_2 3} (T(1) + cn/n) \\ &\leq n^{log_2 3} (T(1) + c) \\ &\leq O(n^{log_2 3}) \end{aligned}$$

Part 3 - Solving Recurrences

The Master Theorem

If $T(n) = aT(\frac{n}{b}) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d) & \text{if} d > log_b a \\ O(n^d logn) & \text{if} d = log_b a \\ O(n^{log_b a}) & \text{if} d < log_b a \end{cases}$$