- **Fibonacci addition**

Theorem: $Fib(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{\sqrt{5}-1}{2}\right)^n \approx 1.6^n + 0.6^n$

$\log_2 Fib(n) \approx \log_2 1.6^n \approx (\log_2 1.6) \cdot n \in \theta(n)$

$Fib(49) =$ 7778742049

$Fib(50) =$ 12586269025

Numbers are stored in little-endian form, as an array.

(a, b, c below are arrays for digits in little-endian)

```
def add(a, b):
    for i = 1 to max(a.length, b.length):
        c[i] = a[i] + b[i]

def carry(c):
    for i = 1 to c.length:
        c[i + 1] += c[i] / 10
        c[i] %= 10
```

This runs in $O(n^{1.618})$, but can be optimized.

Theorem: we can multiply two n-digit numbers in $O(n^{\log_2 3})$ time.

Every d-digit number in base 10 is actually a sum of polynomials in base 10.

$n = \sum_{i=0}^{d} 10^i$ a$[i]$

```
def naiveMultiply(a, b):
    for i = 0 to a.length:
        for i = 0 to b.length:
            c[i + j] += a[i] * b[j]
    carry(c)
```

a[d] ... a[0] = a[d] ... a[d/2]$\cdot 10^{d/2}$ + a[d/2 - 1] - a[0]

So $1234 \cdot 5678 = (12 \cdot 100 + 34)(56 \cdot 100 + 78) = 12 \cdot 56 \cdot 10^4 + 34 \cdot 78 + 12 \cdot 78 \cdot 100 + 34 \cdot 76 \cdot 100$

- **Karatsuba's algorithm:**

$A^{\uparrow} \times B^{\downarrow} + A^{\downarrow} \times B^{\uparrow} = (A^{\uparrow} + A^{\downarrow}) \times (B^{\uparrow} \times B^{\downarrow}) - A^{\uparrow} \times B^{\uparrow} - A^{\downarrow} \times B^{\downarrow}$

Assume number of digits is d.

```
def multiply(a, b):
    x = multiply(a[d] to a[d/2], b[d] to b[d/2])
    y = multiply(a[d/2 - 1] to a[0], b[d/2 - 1] to b[0])
    z1 = (a[d] to a[d/2]) + (a[d/2 - 1] to a[0])
    z2 = (b[d] to b[d/2]) + (b[d/2 - 1] to b[0])
    z = multiply(z1, z2) - x * y
    return x * (10**d) + y + z * (10**(d/2))
```

Runtime recurrence: $T(d) \le 3 \cdot T(d/2) + O(d)$

The recursive tree has $L = \log_2 d$ layers. The number of calls on the bottom level will be $3^L \le 3^{\log_2 d}$

If there are fewer than 10 layers, it is advisable to switch to the naive algorithm.