- Sequência de Fibonacci
  - ↳ 3 versões (1 T.P.C.)
- Cálculo da Potência
  - ↳ 2 versões
- Problema "Maximum Subarray"
  - ↳ 2 versões (1 T.P.C.)

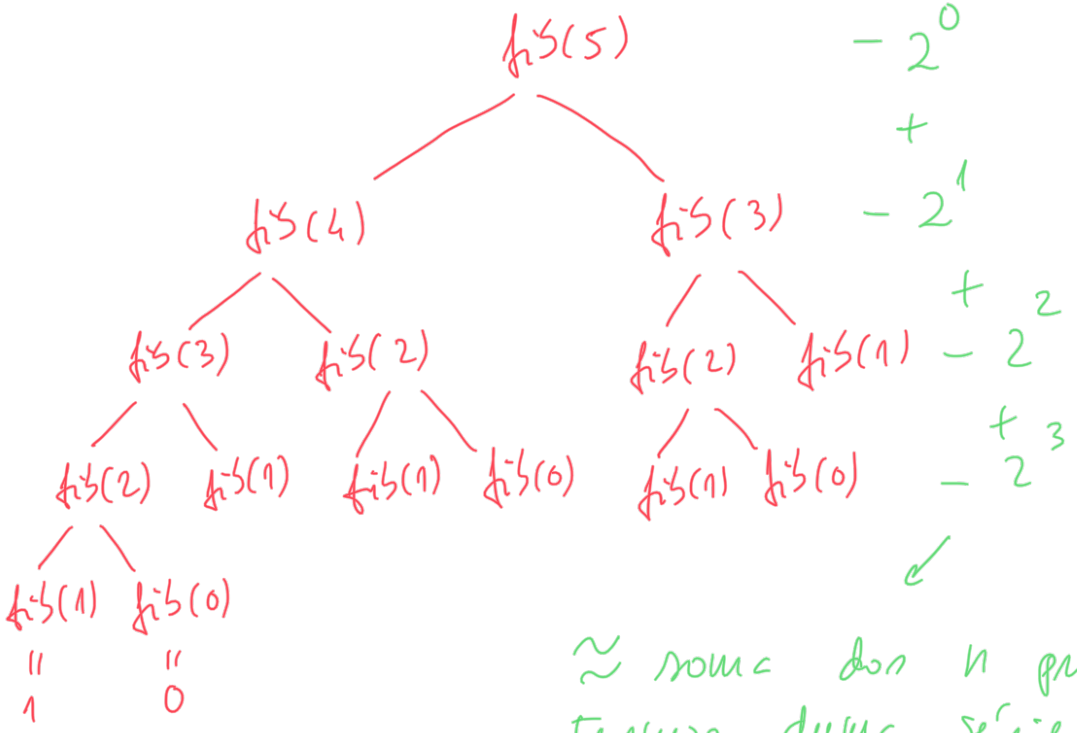
Sequência de Fibonacci

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(n) = F(n-1) + F(n-2), \quad n > 1 \end{cases}$$

0, 1, 2, 3, 5, 8, 13, ...

método 1

```
fun fib_1 (n : Int) : Int =
    if (n <= 1) n
    else fib_1 (n - 1) + fib_1 (n - 2)
```



-2^0

$+$

-2^1

$+$

-2^2

$+$

-2^3

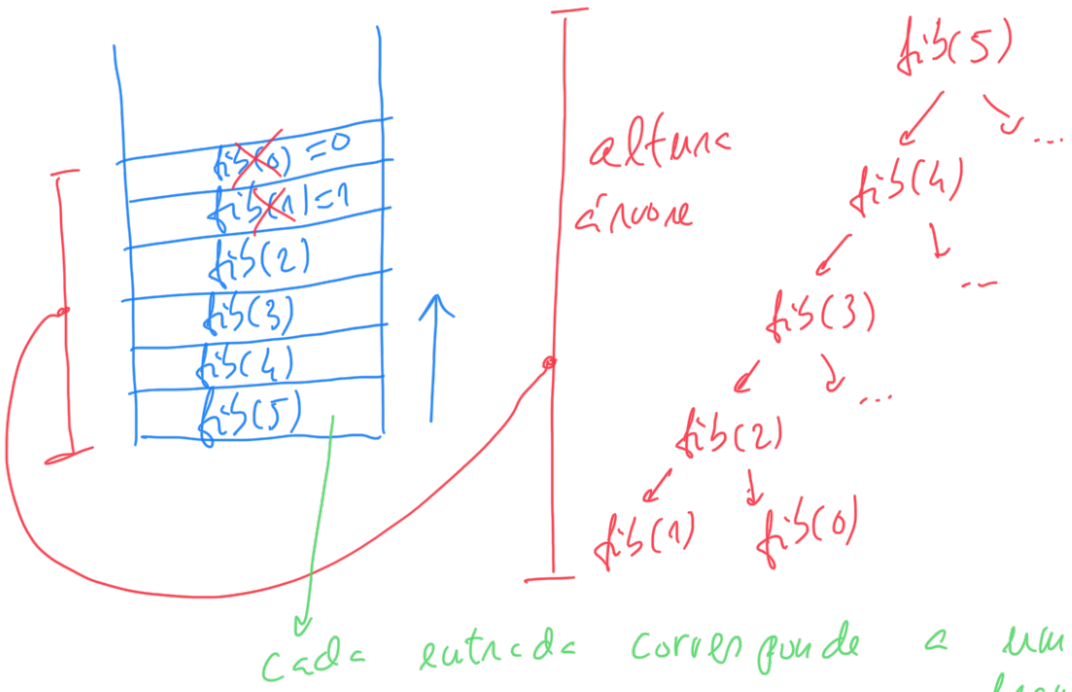
$\approx$ soma dos n primeiros termos duma série geométrica de razão 2

$$= \sum_{i=0}^{n-1} 2^i = 2^0 \times \frac{2^n}{2-1} = 2^n$$

Complexidade em termos de tempo: $O(2^n)$

Notação O-grande ou Big-Oh

Stack



Cada entrada corresponde a um stack frame de uma função

Complexidade em termos de espaço: $O(n)$

Tendo em conta o stack size

"profundidade máxima de chamadas recursivas"

análise quando $n \to \infty$ (assintótica)

Método 2

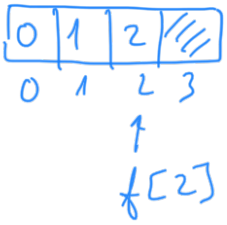
↳ Programação dinâmica: técnica de memorização

Solução: utilizar um array para armazenar os nºs calculados até agora

```
fun fib_2(n:Int) : Int {
    val f = IntArray(n+2) // +1 para gerar o espaço extra no caso do 0.
                          // size
    var i : Int

    f[0] = 0
    f[1] = 1
    i = 2
    while (i <= n) {
        f[i] = f[i-1] + f[i-2]
        ++i
    }
    return f[n]
}
```

fib(2)



Complexidade em termos de tempo: $O(n)$

Complexidade em termos de espaço: $O(n)$

T.P.C.

usar apenas a memorização dos últimos 2 termos ⇒ Complex. espacial reduz para $O(1)$ ↙ constante

Cálculo da potência

a^n, com $a \geq 0$ e $n \geq 0$

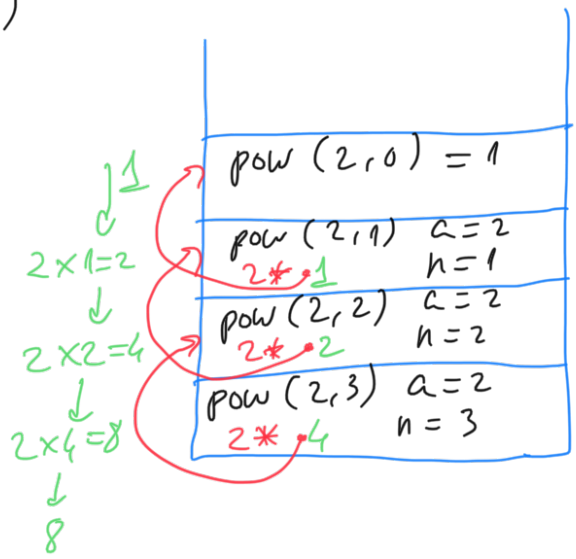
Método 1

```
fun pow_1 (a: Int, n: Int): Int =
    if (n==0) 1   // Caso de Paragem
    else a * pow_1 (a, n-1)
```

Complexidade Temporal: $O(n)$

e Espacial: $O(n)$

- Linear
- nº funções expandidas // stack size

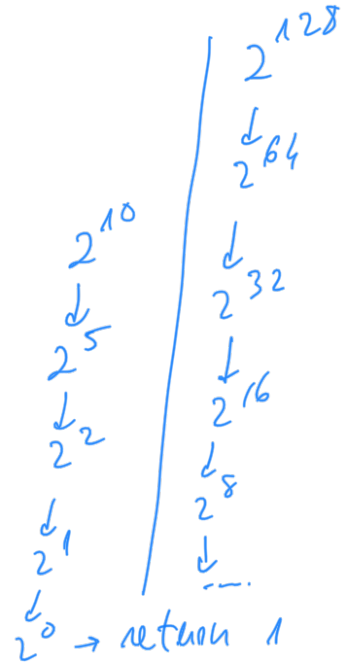


Método 2

$$a^n = \begin{cases} 1, & n = 0 \\ a^{n/2} * a^{n/2}, & \text{se } n \text{ par} \\ a^{n/2} * a^{n/2} * a, & \text{se } n \text{ ímpar} \end{cases}$$

(divisão inteira)

"Divide to Conquer"

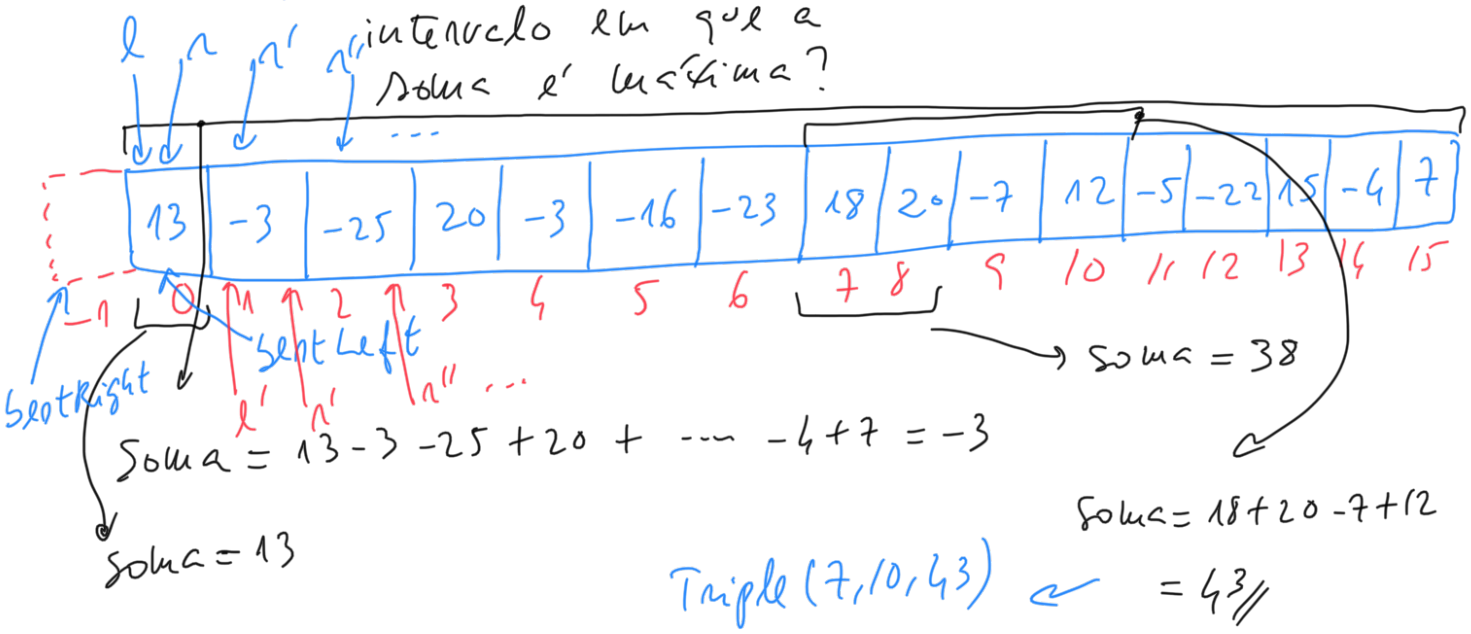



```
fun pow_2 ( a:Int, n:Int):Int {
    if ( n==0) return 1
    val z = pow_2(a, n/2)
    return if (n % 2 == 0) z*z else z*z*a
}
```

Complexidade temporal: $O(\log_2 N)$

" espacial: $O(\log_2 N)$ ↓ Logarítmico

Maximum Subarray Problem




```
fun maxSubArrayQuadratic(array: DoubleArray,
                         left: Int,
                         right: Int): Triple {
    ...
    for (...) {
        for (...) {
            ...
        }
    }
}

data class Triple(val left,
                  val right,
                  val sum)
```

T.P.C.