1.

- Sequência de Fibonacci
  ↳ 3 versões (1 T.P.C.)

- Cálculo da Potência
  ↳ 2 versões

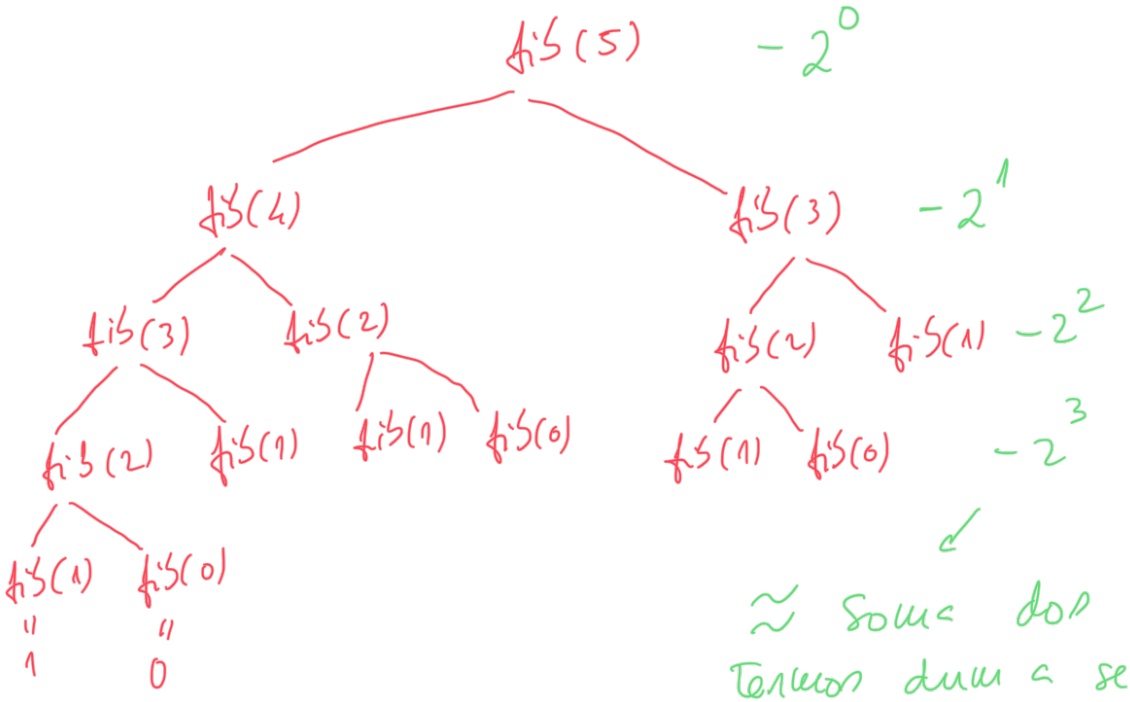
- Problema "Maximum - subarray"
  ↳ 2 versões (1 T.P.C.)

Sequência de Fibonacci

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

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

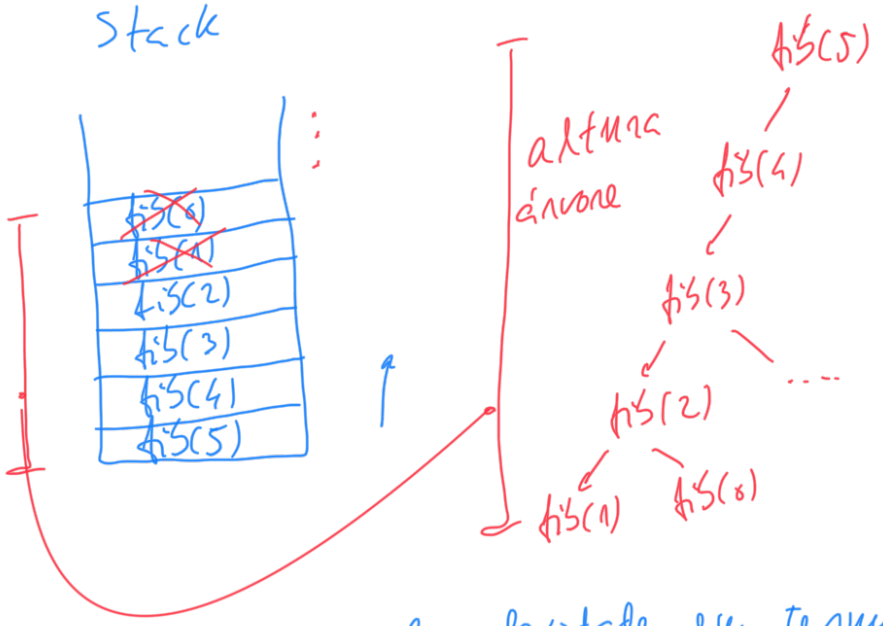
$\approx$ soma dos n primeiros termos duma série geométrica

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

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

Notação O'-grande ou Big-Oh

4.



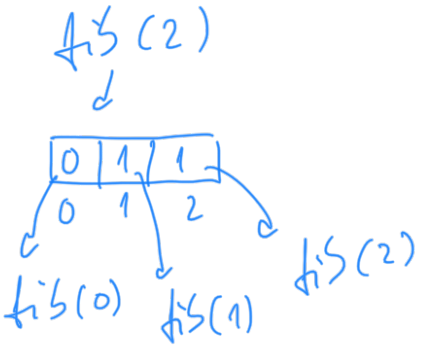
Complexidade em termos de espaço: $O(n)$ → tendo em conta o stack size nº de chamadas recursivas

Método 2

↳ Programação dinâmica: técnica de memorização

Solução: utilização de um array para armazenar os n+1 cálculos até agora

```
fun fib_2 (n: Int): Int {
    val f = IntArray (n+2) // +1 para guardar o espaço extra no caso do 0.
    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]
}
```



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

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

T.P.C.

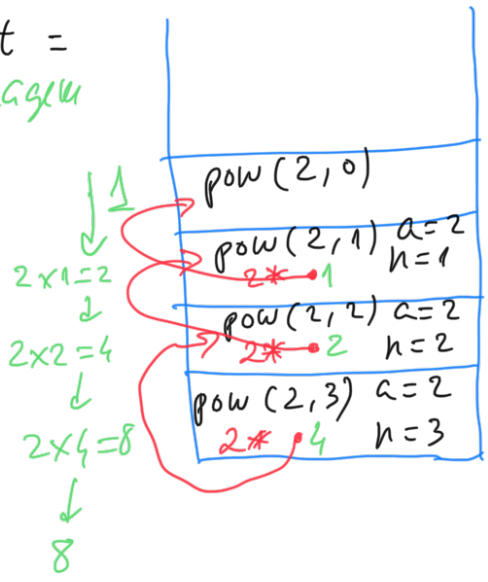
Usar apenas a memorização dos últimos 2 termos ⇒ compl. espacial reduz para $O(1)$

Cálculo de potência

a^n, com $a > 0$ 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: $O(n)$

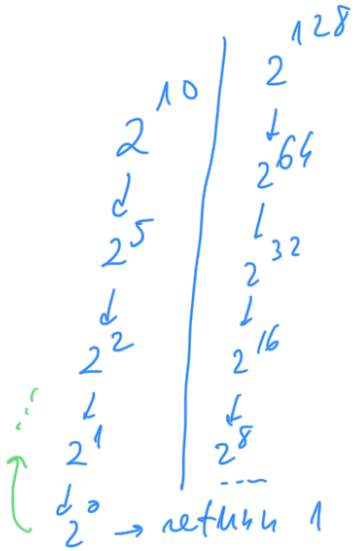
" espacial: $O(n)$

- Linear
- nº de funções expandidas " stack size

Métodos 2

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

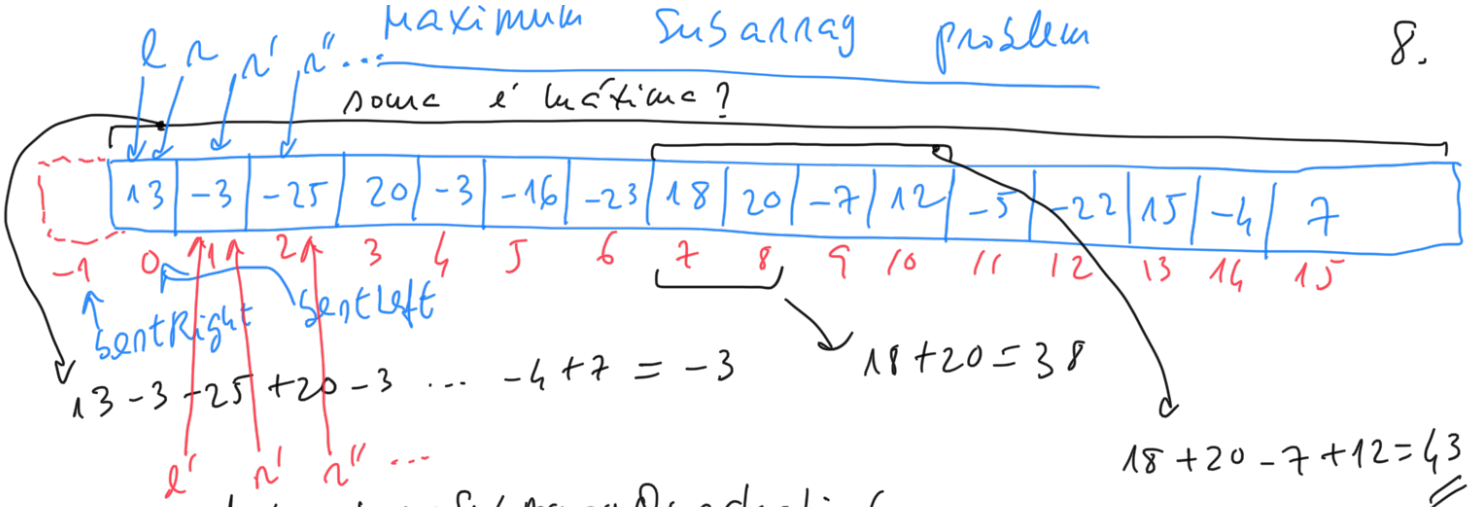
```
fun pow_2(a: Int, n: Int) {
    if (n==0) return 1
    val z = pow_2(a, n/2)
    return if (n%2==0) z*z else z*z*a
}
```



Complexidade Temporal: $O(\log_2 N)$ → Logarítmico

" Espacial: $O(\log_2 N)$

Maximum Subarray problem




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

T.P.C. 1 → for (...) for(...)

T.P.C. 2 → Linear → método de Kadane