Aula 1

1. Calcular os nºs de Fibonacci
2. Notação Assimptótica
3. Divide-para-Conquistar
4. Merge Sort
5. Teorema Mestre

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 1

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{Se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Esta implementação é eficiente?

Implementação 1

```
Fib(n):
    if n==0 || n==1
      return 1
    else
      return Fib(n-1) + Fib(n-2)
```

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 1

```
Fib(n):
    if n==0 || n==1
      return 1
    else
      return Fib(n-1) + Fib(n-2)
```

Esta implementação é eficiente?

$T(n) \rightarrow$ nº de instruções executadas p/ calcular Fib(n)

$$T(n) = \begin{cases} c_0 & \text{se } n = 0 \vee n = 1 \\ c_1 + T(n-1) + T(n-2) & \text{c.c.} \end{cases}$$

$T(n) > Fib(n)$

$Fib(n) \geq (3/2)^n$ for $n \geq 1$

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 2

```
Fib(n)
  let c[0..n] be a new array
```

Esta implementação é eficiente?

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 2

```
Fib(n)
  let c[0..n] be a new array
  c[0] = 1
  c[1] = 1
  for i=2 to n
    c[i] = c[i-1] + c[i-2]
  return c[i]
```

Esta implementação é eficiente?

Exemplo 1 - Nos de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{Se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 2

```
Fib(n)
  let c[0..n] be a new array
  c[0] = 1
  c[1] = 1
  for i=2 to n
      c[i] = c[i-1] + c[i-2]
  return c[i]
```

Esta implementação é eficiente?

$$T(n) = c_0 \cdot (n-1) + c_1$$
$$= O(n)$$

$$S(n) = O(n)$$

Conseguimos melhorar?

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{Se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 3

```
Fib(n)
    if n == 0 || n == 1
        return 1
    else
```

Esta implementação é eficiente?

Exemplo 1 - Nºs de Fibonacci

$$fib(n) = \begin{cases} 1 & \text{Se } n = 0 \text{ ou } n = 1 \\ fib(n-1) + fib(n-2) & \text{c.c.} \end{cases}$$

Implementação 3

```
Fib(n)
    if n == 0 || n == 1
        return 1
    else
        prev = 1
        cur = 1
        for i=2 to n
            temp = cur
            cur = prev + cur
            prev = temp
        return cur
```

Esta implementação é eficiente?

$T(n) = O(n)$

$S(n) = O(1)$

Notação Assimptótica

Definição 1 [Majorante / Minorante Assimptótico]

Majorante:

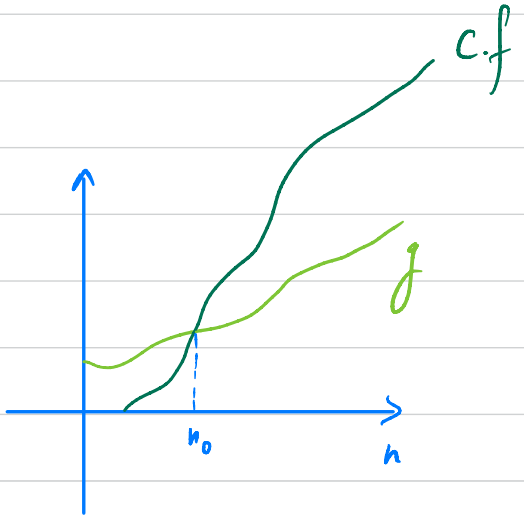
$$g \in \mathcal{O}(f) \text{ sse } \exists c.\ \exists n_0.\ \forall n \geq n_0.\ g(n) \leq c \cdot f(n)$$

Notação Assimptótica

Definição 1 [Majorante / Minorante Assimptótico]

Majorante:

$$g \in \mathcal{O}(f) \text{ sse } \exists c.\ \exists n_0.\ \forall n \geq n_0.\ g(n) \leq c \cdot f(n)$$

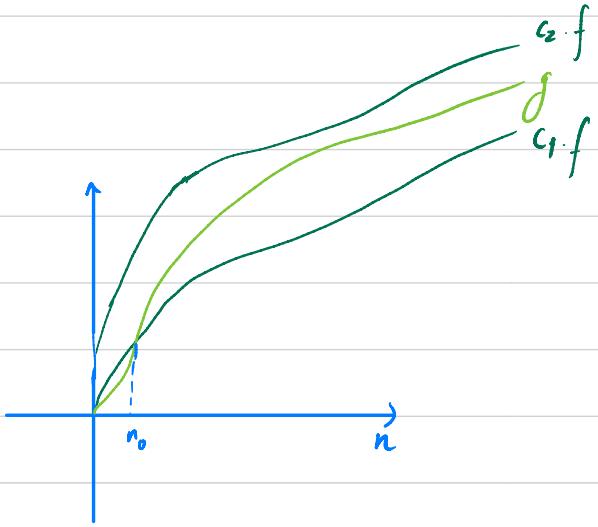


Minorante:

$$g \in \Omega(f) \text{ sse } \exists c.\ \exists n_0.\ \forall n \geq n_0.\ g(n) \geq c \cdot f(n)$$

Tight-Bound:

$$g \in \Theta(f) \text{ sse } \exists c_1, c_2.\ \exists n_0.\ \forall n \geq n_0.\ c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$$



Notação Assimptótica

Lema 1

$g \in \Theta(f)$ sse $g \in O(f) \wedge g \in \Omega(f)$

Prova

Notação Assimptótica

Lema 1

$g \in \Theta(f)$ sse $g \in O(f) \wedge g \in \Omega(f)$

Prova

$\boxed{\Rightarrow}$

$g \in \Theta(f) \Rightarrow g \in O(f) \wedge g \in \Omega(f)$

- $g \in \Theta(f)$ (hyp)
- $\exists n_0, c_1, c_2 . \forall n \geq n_0 . \; c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$
  - $\exists n_0, c . \forall n \geq n_0 . \; g(n) \geq c \cdot f(n) \iff g \in \Omega(f)$
  - $\exists n_0, c . \forall n \geq n_0 . \; g(n) \leq c \cdot f(n) \iff g \in O(n)$

Notação Assimptótica

Lema 1

$g \in \Theta(f)$ sse $g \in O(f) \wedge g \in \Omega(f)$

Prova

$\Leftarrow$

$$g \in O(f) \wedge g \in \Omega(f) \Rightarrow g \in \Theta(f)$$

- $g \in O(f) \Leftrightarrow \exists c_1, n_0 . \forall n \geq n_0 . g(n) \leq c_1 \cdot f(n)$
- $g \in \Omega(f) \Leftrightarrow \exists c_2, n_0' . \forall n \geq n_0' . g(n) \geq c_2 \cdot f(n)$
- $\exists n_0'', c_1, c_2 . \forall n \geq n_0'' . c_2 \cdot f(n) \leq g(n) \leq c_1 \cdot f(n)$

  $\hookrightarrow \max(n_0, n_0')$

Notação Assimptótica

Lema 1 [Transitividade]

i) $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$

ii) $f \in \Omega(g) \wedge g \in \Omega(h) \Rightarrow f \in \Omega(h)$

iii) $f \in \Theta(g) \wedge g \in \Theta(h) \Rightarrow f \in \Theta(h)$

Prova

Notação Assimptótica

Lema 1 [Transitividade]

i) $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$

ii) $f \in \Omega(g) \wedge g \in \Omega(h) \Rightarrow f \in \Omega(h)$

iii) $f \in \Theta(g) \wedge g \in \Theta(h) \Rightarrow f \in \Theta(h)$

Prova i)

- Hipóteses: $\underbrace{f \in O(g)}_{(*)} \wedge \underbrace{g \in O(h)}_{(**)}$

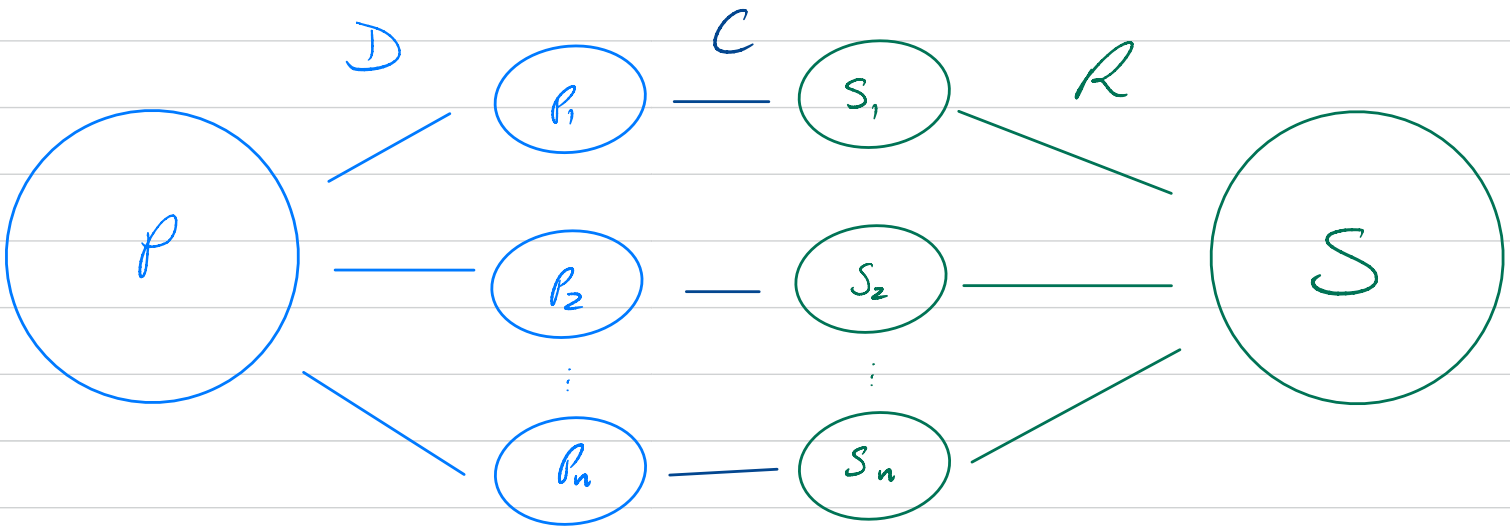
$(*)$ $\exists c, n_0. \forall n \geq n_0. f(n) \leq c \cdot g(n)$

$(**)$ $\exists c', n_0'. \forall n \geq n_0'. g(n) \leq c' \cdot h(n)$

$\forall n \geq \max(n_0, n_0'). f(n) \leq c \cdot c' \cdot h(n) \rightarrow f \in O(h)$

Metodologia Dividir-para-Conquistar

1. Dividir o problema a resolver num conjunto de subproblemas
2. Resolver (recursivamente) cada um dos subproblemas
3. Combinar as soluções dos subproblemas para obter a solução do problema original



MergeSort

```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

Dividir: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

Combinar: $C(n) = ?$ (merge)

Exemplo:

$\langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$ $\quad l = 1, \; r = 8$

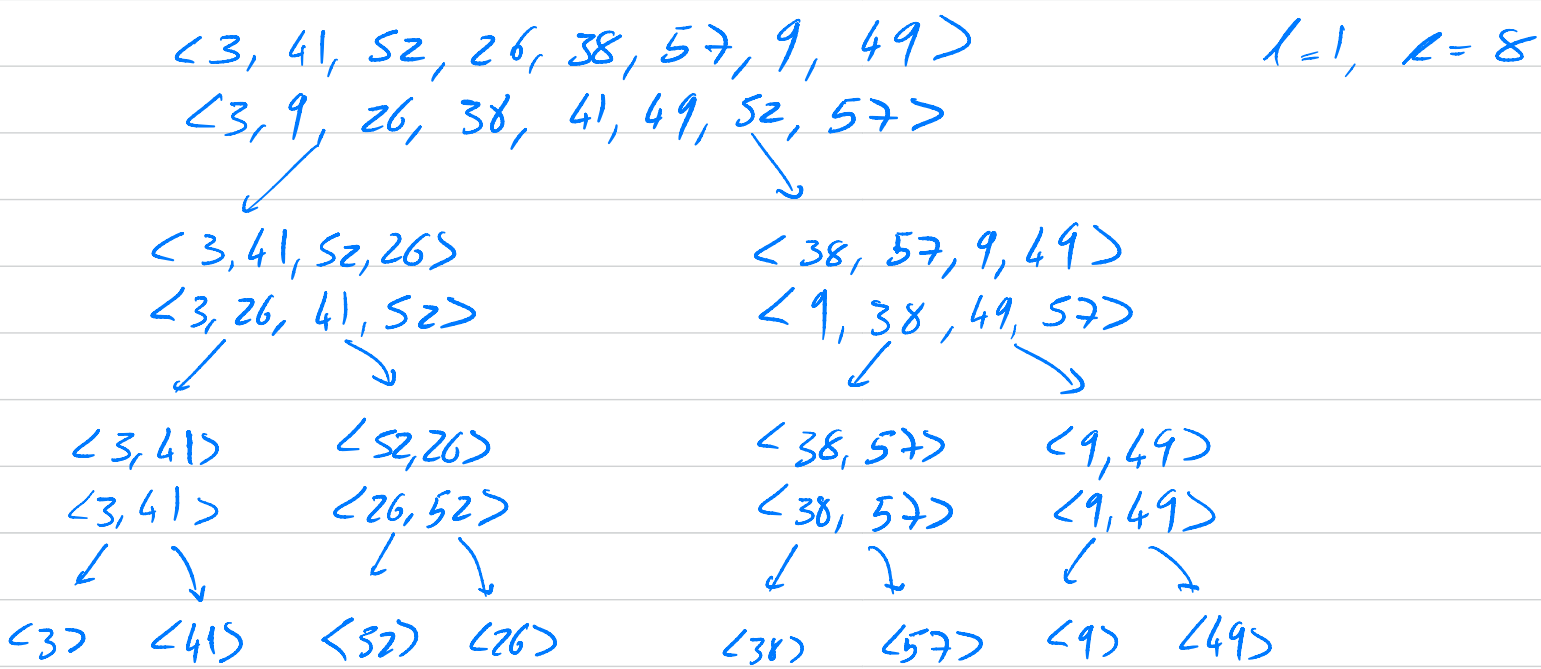
Merge Sort

```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

Dividir: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

Combinar: $C(n) = ?$ merge



$l = 1, \quad r = 8$

Merge Sort

```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

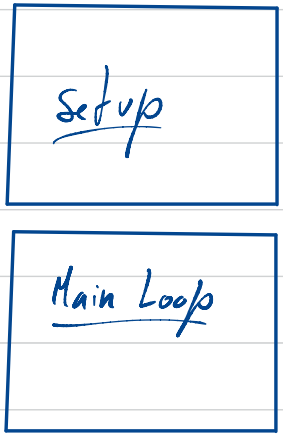
Dividir: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

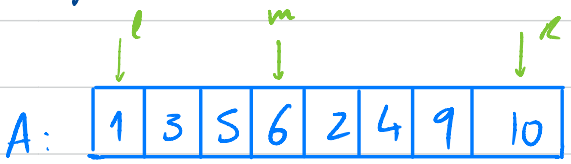
Combinar: $C(n) = ?$ merge

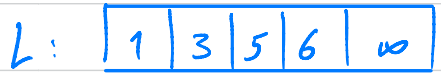
Merge(A, l, m, r)

Merge (A, l, m, r)

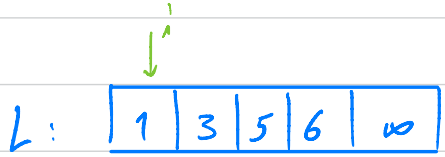


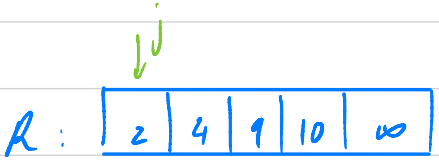
Setup:



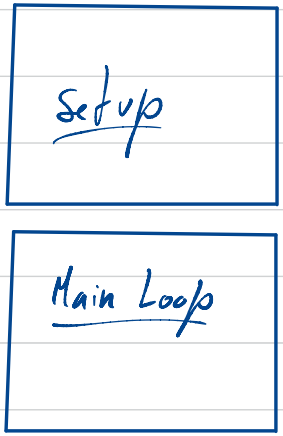


Main Loop:

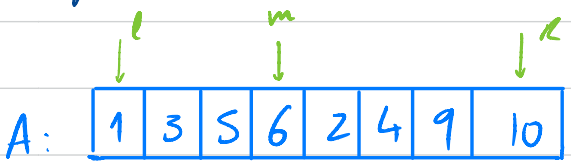


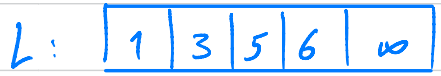


Merge (A, l, m, r)

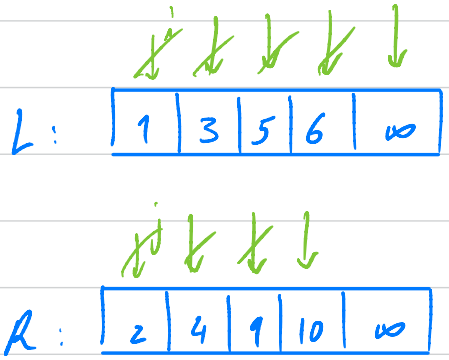


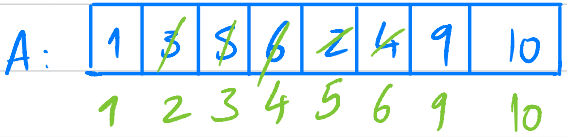
Setup:





Main Loop:





Merge Sort

```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

Divide: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

Combinar: $C(n) = ?$ merge

Merge(A, l, m, r)

Merge Sort

```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

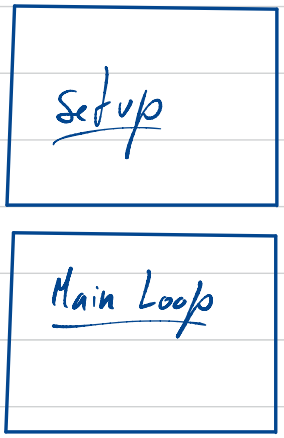
Dividir: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

Combinar: $C(n) = ?$ (merge)

Set-Up: Alocar os arrays L e R e preencher c/ os valores de A

Merge(A, l, m, r)




```
let L[1..(m-l)+2] be a new array
let R[1..(r-m)+1] be a new array
for i = 1 to (m-l)+1
    L[i] = A[i]
for j = 1 to (r-m)
    R[j] = A[j]
L[m-l+2] = ∞
R[m-l+2] = ∞
```

Merge Sort

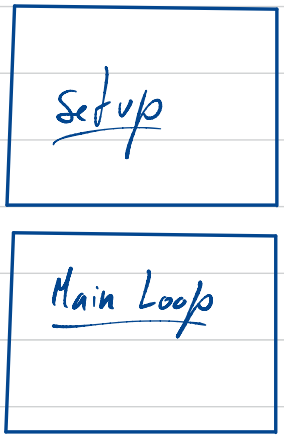
```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

Divide: $D(n) = O(1)$

Resolve: $R(n) = 2 \cdot T(n/2)$

Combine: $C(n) =$? merge

Merge(A, l, m, r)



Set-Up: Main Loop

```
i = 1 ; j = 1
for k = l to r
  if L[i] ≤ R[j]
    A[k] = L[i]
    i++
  else
    A[k] = R[j]
    j++
```

Merge - Complexidade

Merge (A, l, m, r)

```
let L[1.. (m-l)+2] be a new array
let R[1.. (r-m)+1] be a new array
for i=1 to (m-l)+1
    L[i] = A[i]
for j=1 to (r-m)
    R[j] = A[j]
L[m-l+2] = ∞
R[m-l+2] = ∞

i = 1; j = 1
for k=l to r
    if L[i] ≤ R[j]
        A[k] = L[i]; i++
    else
        A[k] = R[j]; j++
```

MergeSort

TRC: $O(n) + O(n) = O(n)$

$f \in O(n) \wedge g \in O(n) \Rightarrow f+g \in O(n)$

```
Merge(A, l, m, r)

let L[1..(m-l)+2] be a new array
let R[1..(r-m)+1] be a new array
for i=1 to (m-l)+1                    ||  -> (m-l)+1
    L[i] = A[i]
for j=1 to (r-m)                      ||  -> (r-m)
    R[j] = A[j]
L[m-l+2] = ∞
R[m-l+2] = ∞

i = 1; j = 1
for k=l to r                          |   (r-l)+1
    if L[i] ≤ R[j]                    |
        A[k] = L[i]; i++              |
    else                              |
        A[k] = R[j]; j++              |
```

$(m-l)+1 + (r-m) = r-m+m-l+1$

$(r-l)+1$ — n

$O(n)$

$O(n)$

$O(n)$

MergeSort

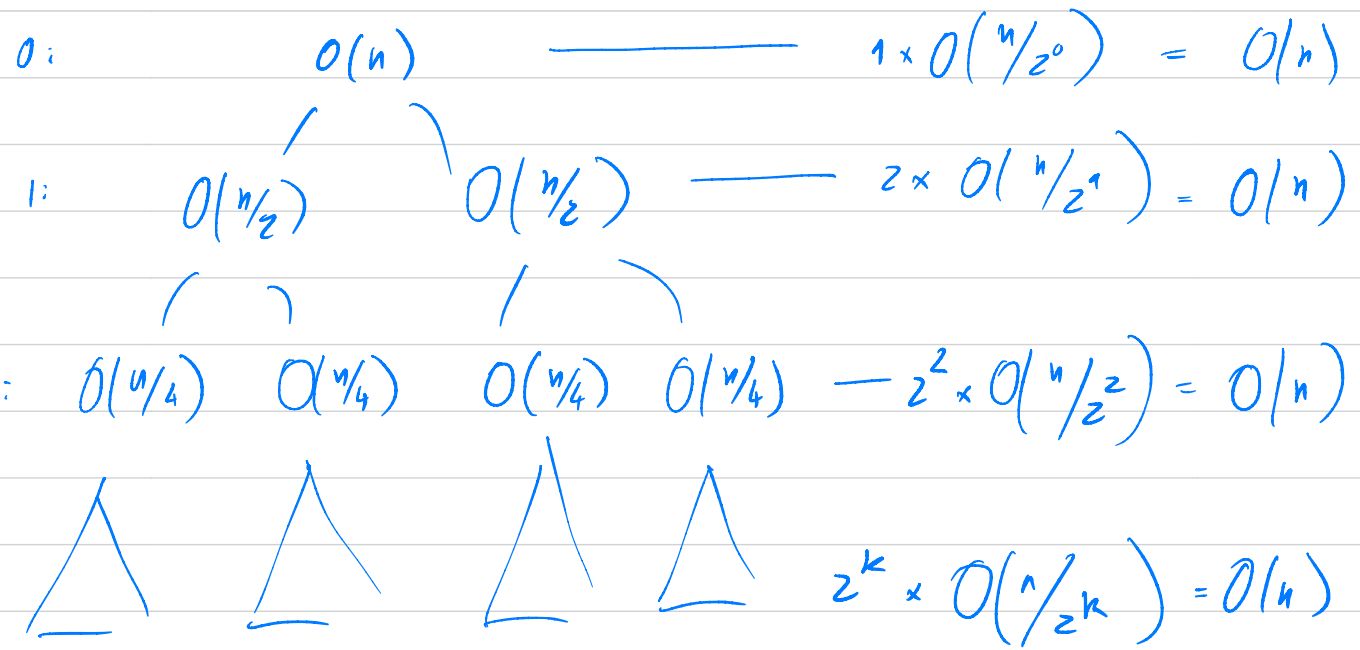
```
MergeSort(A, l, r)
  if l < r
    m = ⌊(l + r)/2⌋
    MergeSort(A, l, m)
    MergeSort(A, m+1, r)
    Merge(A, l, m, r)
```

Dividir: $D(n) = O(1)$

Resolver: $R(n) = 2 \cdot T(n/2)$

Combinar: $C(n) = O(n)$

$$T(n) = T(n/2) + O(n)$$



Quem é k?

$n/2^k = 1$

$k = \log_2 n$

$$T(n) = O(n \cdot \log n)$$

Método de Substituição

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = O(n \lg n)$$

Base Case

$n = 2$

$$T(2) = 2T(2/2) + O(2)$$
$$= 2 \cdot T(1) + O(1)$$
$$= O(1)$$

$n > 2$

$$T(n+1) = 2 \cdot T\left(\frac{n+1}{2}\right) + O(n+1)$$

$$= 2 \cdot O\left(\left(\frac{n+1}{2}\right) \cdot \log_2\left(\frac{n+1}{2}\right)\right) + O(n+1)$$

$$= 2 \cdot \frac{n+1}{2} \cdot O\left(\log_2\left(\frac{n+1}{2}\right)\right) + O(n+1)$$

$$= (n+1) \cdot O(\log_2(n+1)) - O((n+1) \cdot \log_2 2) + O(n+1)$$

$$= (n+1) \cdot O(\log_2(n+1))$$