

“Maus programadores se preocupam com código. Bons programadores se preocupam com as estruturas de dados e seus relacionamentos” (Linus Torvalds).

Merge Sort

Paulo Ricardo Lisboa de Almeida

Merge Sort

Ideia: particionar o vetor em duas partes de (quase) mesmo tamanho

Recursivamente, para cada uma das partes, repetir o processo, até que o vetor particionado seja **trivialmente** ordenável

Juntar novamente os vetores através de uma função de merge

Merge Sort - Exemplo

42	15	23	8	16	4
----	----	----	---	----	---

42	15	23	8	16	4
----	----	----	---	----	---



A diagram illustrating a memory copy operation. At the top, a horizontal array of six boxes contains the values 42, 15, 23, 8, 16, and 4. Two arrows point downwards from the second and third boxes to a separate horizontal array below. This second array, which has three boxes, contains the values 42, 15, and 23. The boxes are separated by thin vertical lines.

42	15	23
----	----	----



A diagram illustrating a memory copy operation. It shows the same initial array at the top. Two arrows point downwards from the fourth and fifth boxes to a separate horizontal array below. This second array, which has three boxes, contains the values 8, 16, and 4. The boxes are separated by thin vertical lines.

8	16	4
---	----	---

42	15	23	8	16	4
----	----	----	---	----	---

42	15	23
----	----	----

8	16	4
---	----	---

42

15	23
----	----

8

16	4
----	---

42	15	23	8	16	4
----	----	----	---	----	---

42	15	23
----	----	----

8	16	4
---	----	---

42

15	23
----	----

8

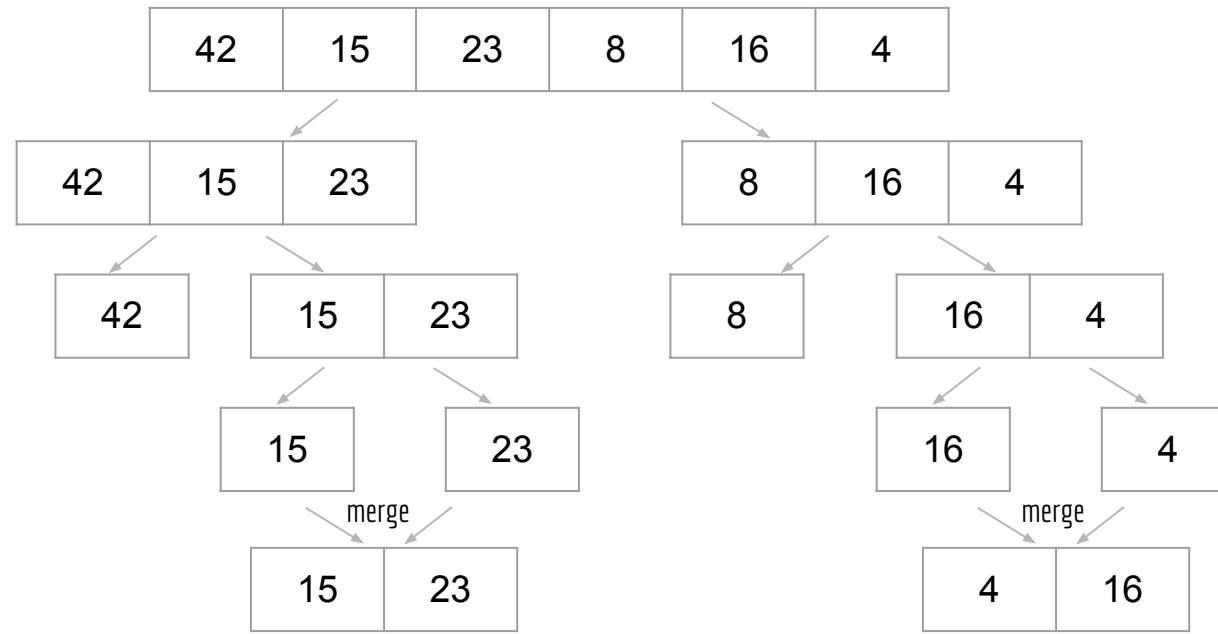
16	4
----	---

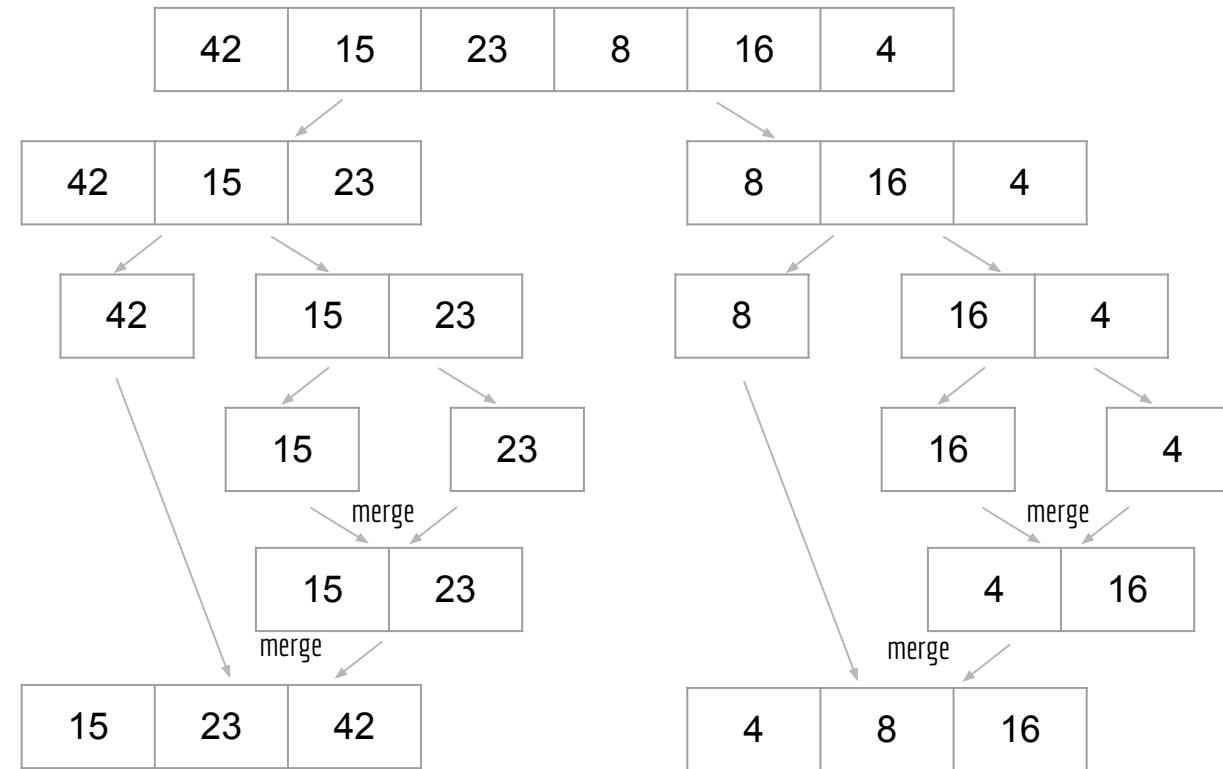
15

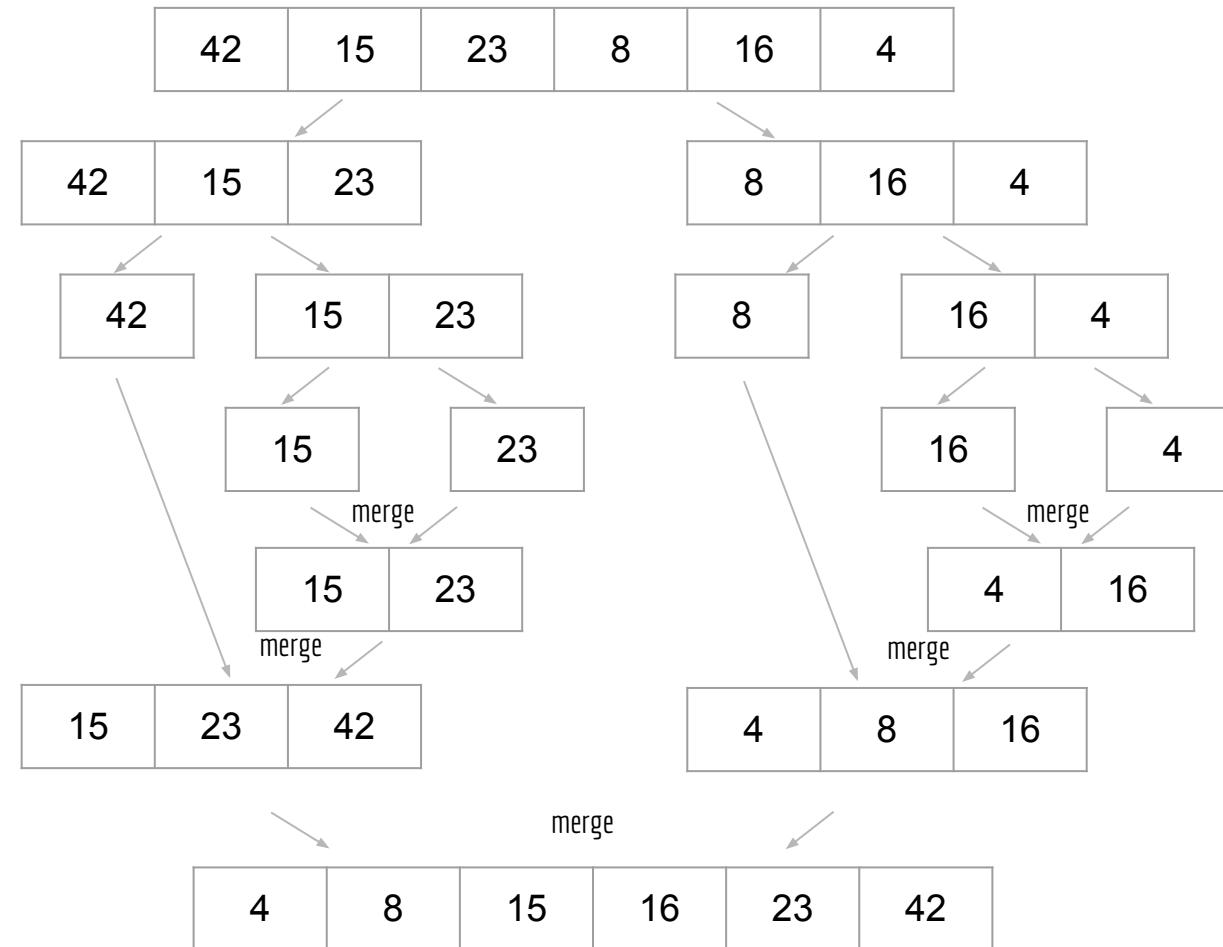
23

16

4







Merge-Sort

função mergeSort (v,a,b)

entrada: vetor v, indexado por [a..b]

saída: o vetor v modificado de forma que v[a..b] é um vetor ordenado.

se $a \geq b$

 retorne v

$m \leftarrow \lfloor (a+b)/2 \rfloor$

mergeSort(v,a,m)

mergeSort(v,m + 1,b)

retorne merge(v,a,m,b)

Teste de Mesa

i	1	2	3	4
v[i]	42	15	23	8

```
função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

mergeSort		
a	b	m
1	4	

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorne v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorne v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

mergeSort		
a	b	m
1	4	2



mergeSort		
a	b	m
1	2	

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2



mergeSort		
a	b	m
1	2	1

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2



mergeSort		
a	b	m
1	2	1

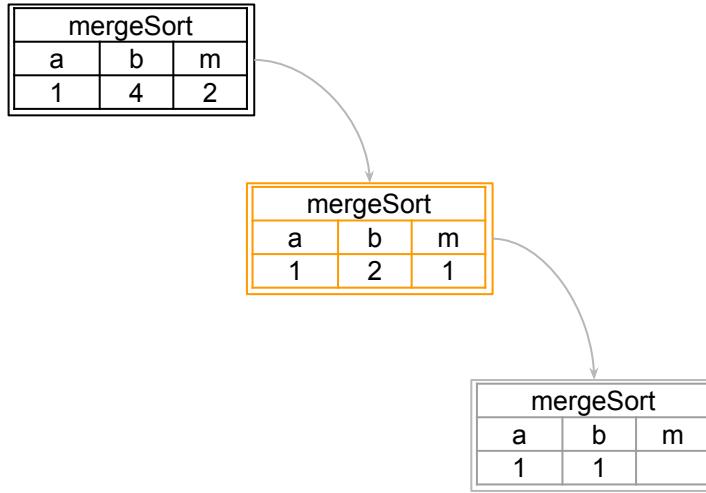
i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```



i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
1	2	1

mergeSort		
a	b	m
1	1	

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

mergeSort

a	b	m
1	2	1

i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
1	2	1

mergeSort		
a	b	m
2	2	

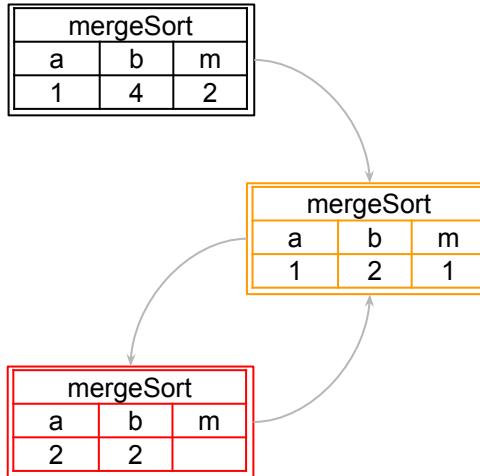
i	1	2	3	4
v[i]	42	15	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```



i	1	2	3	4
v[i]	42	15	23	8

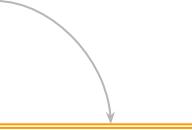
```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2



mergeSort		
a	b	m
1	2	1

i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

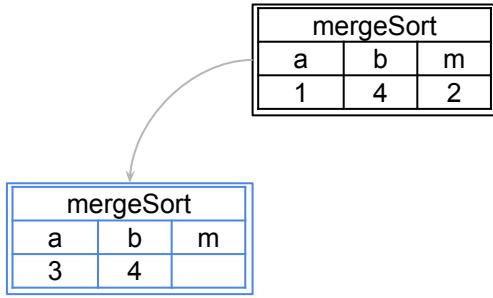
mergeSort		
a	b	m
1	4	2

i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```



i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

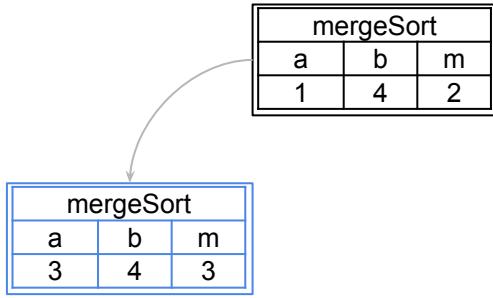
mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
3	4	3

i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
  
```



i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
3	4	3

mergeSort		
a	b	m
3	3	

i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
  
```

mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
3	4	3

mergeSort		
a	b	m
3	3	

i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
  
```

mergeSort		
a	b	m
1	4	2

mergeSort		
a	b	m
3	4	3

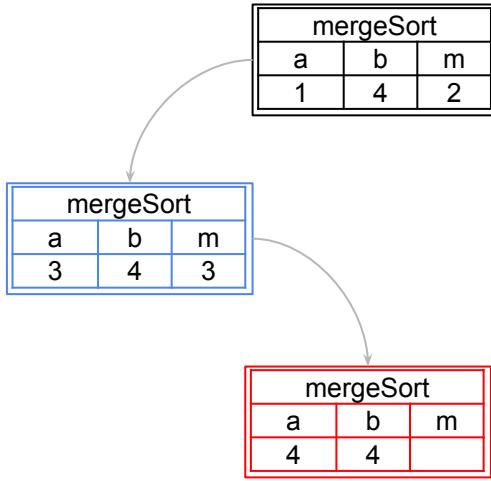
i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```



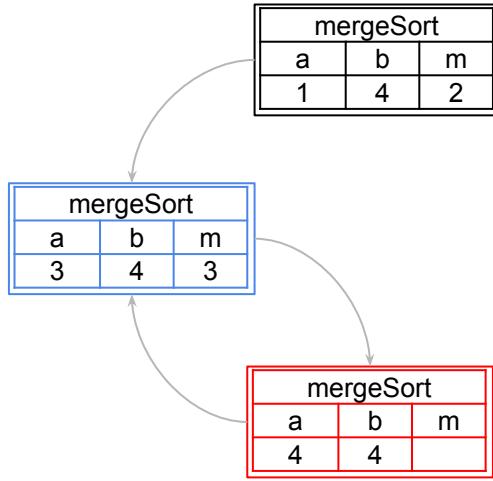
i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```



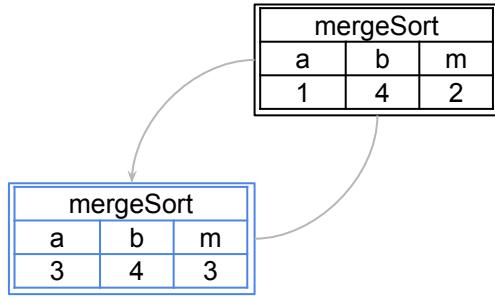
i	1	2	3	4
v[i]	15	42	23	8

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋

mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```



i	1	2	3	4
v[i]	15	42	8	23

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorne merge(v,a,m,b)

```

mergeSort		
a	b	m
1	4	2

i	1	2	3	4
v[i]	8	15	23	42

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```

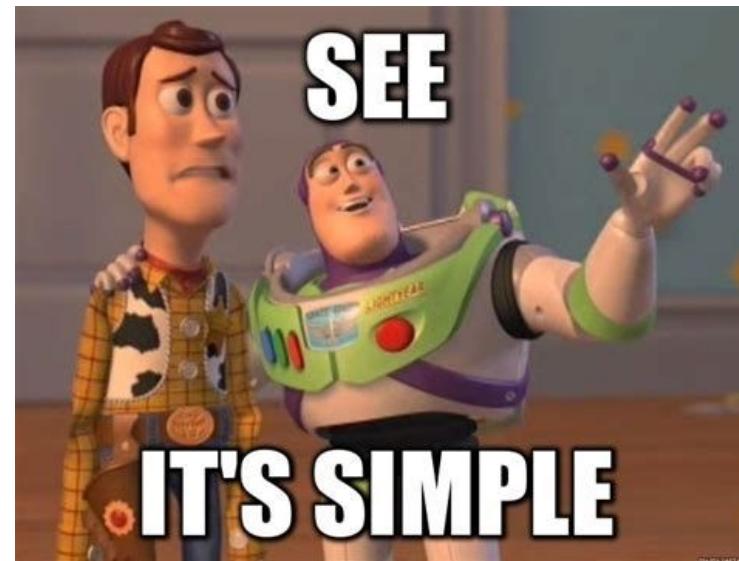
mergeSort		
a	b	m
1	4	2

i	1	2	3	4
v[i]	8	15	23	42

```

função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)

```



Análise

Considerando o número de comparações com elementos do vetor

$$C(n) = \dots$$

```
função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

Análise

Considerando o número de comparações com elementos do vetor

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ C(\lfloor \frac{n}{2} \rfloor) + C(\lceil \frac{n}{2} \rceil) + C_m(n), & \text{se } n > 1 \end{cases}$$

```
função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

Análise

Da aula passada, o custo do merge para um vetor de tamanho n é n

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ C(\lfloor \frac{n}{2} \rfloor) + C(\lceil \frac{n}{2} \rceil) + n, & \text{se } n > 1 \end{cases}$$

```
função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

Análise

Para facilitar (muito) a análise, considerando que $\lfloor \frac{n}{2} \rfloor \approx \frac{n}{2}$ e $\lceil \frac{n}{2} \rceil \approx \frac{n}{2}$

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ C\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

```
função mergeSort (v,a,b)
se a ≥ b
    retorno v
m ← ⌊(a+b)/2⌋
mergeSort(v,a,m)
mergeSort(v,m + 1,b)
retorno merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n$$

```
função mergeSort (v,a,b)
  se a ≥ b
    retorne v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorne merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n$$

$$C(n) = 2\left(2C\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2C\left(\frac{n}{2^2}\right) + 2n$$

```
função mergeSort (v,a,b)
    se a ≥ b
        retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n$$

$$C(n) = 2\left(2C\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2C\left(\frac{n}{2^2}\right) + 2n$$

$$C(n) = 2^2\left(2C\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n = 2^3C\left(\frac{n}{2^3}\right) + 3n$$

```
função mergeSort (v,a,b)
    se a ≥ b
        retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n$$

$$C(n) = 2\left(2C\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2C\left(\frac{n}{2^2}\right) + 2n$$

$$C(n) = 2^2\left(2C\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n = 2^3C\left(\frac{n}{2^3}\right) + 3n$$

$$C(n) = 2^\mu C\left(\frac{n}{2^\mu}\right) + \mu n$$

```
função mergeSort (v,a,b)
    se a ≥ b
        retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2^\mu C\left(\frac{n}{2^\mu}\right) + \mu n \quad \frac{n}{2^\mu} = 1 \iff \mu = \log_2 n$$

```
função mergeSort (v,a,b)
    se a ≥ b
        retorne v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorne merge(v,a,m,b)
```

Análise

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1, \\ 2C\left(\frac{n}{2}\right) + n, & \text{se } n > 1 \end{cases}$$

$$C(n) = 2^\mu C\left(\frac{n}{2^\mu}\right) + \mu n \quad \frac{n}{2^\mu} = 1 \iff \mu = \log_2 n$$

$$C(n) = 2^{\log_2 n} C(1) + \log_2(n)n = n \log_2 n$$

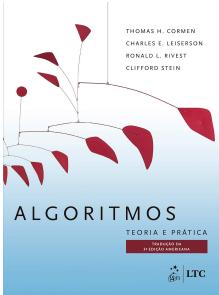
```
função mergeSort (v,a,b)
    se a ≥ b
        retorno v
    m ← ⌊(a+b)/2⌋
    mergeSort(v,a,m)
    mergeSort(v,m + 1,b)
    retorno merge(v,a,m,b)
```

Exercícios

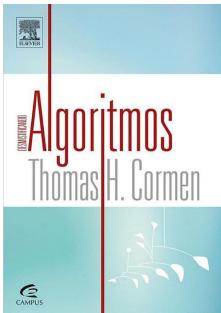
1. Plote um gráfico onde o eixo x possui valores para n , e no eixo y plote $n^2/2$ e $n\log_2 n$. Compare a diferença entre as curvas, especialmente para valores grandes de n .
2. Mostre que o custo de memória do algoritmo é aproximadamente $M(n) = n + n * \log_2(n)$
3. Implemente o merge sort em C

Referências

T. Cormen, C. Leiserson,
R. Rivest, C. Stein.
Algoritmos: Teoria e
Prática. 3a ed. 2012



T. Cormen.
Desmistificando
algoritmos. 2017.

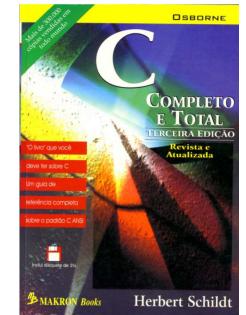


Renato Carmo. Algoritmos e
Estruturas de Dados.
www.inf.ufpr.br/renato

R. Sedgewick, K. Wayne.
Algorithms Part I. 4a ed.
2014



H. Schildt. C completo e
total. 1996



Licença

Este obra está licenciado com uma Licença [Creative Commons Atribuição 4.0 Internacional.](#)

