"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

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Merge Sort

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

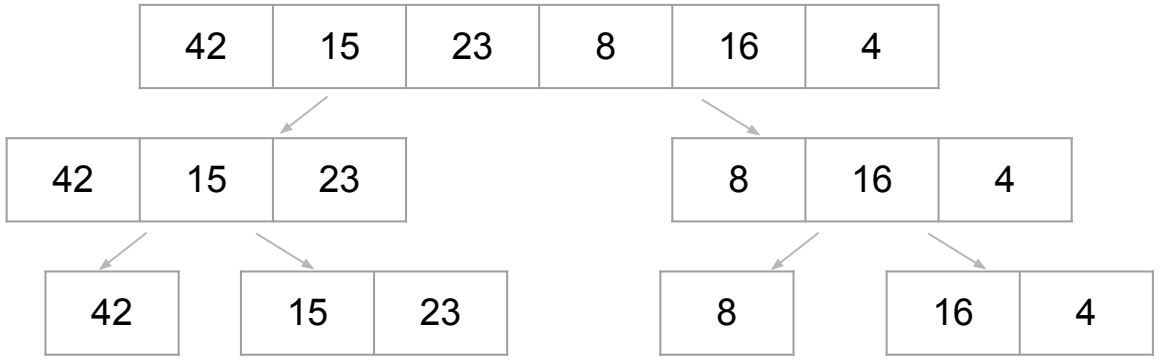
Recursivamente, para cada uma das partes, repetir o processo, até que o vetor particionado seja **trivialmente** ordenável

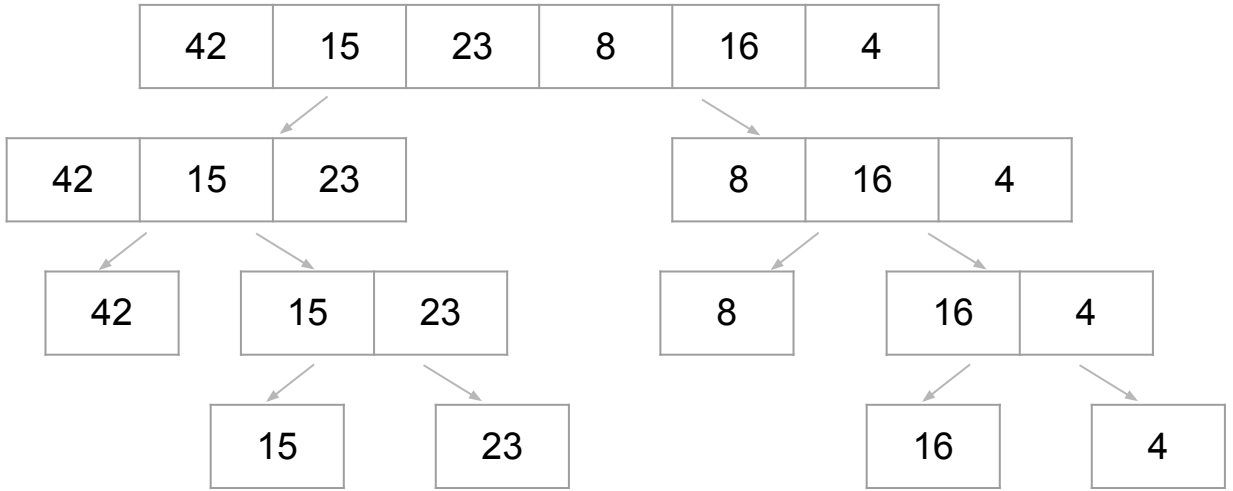
Juntar novamente os vetores através de uma função de merge

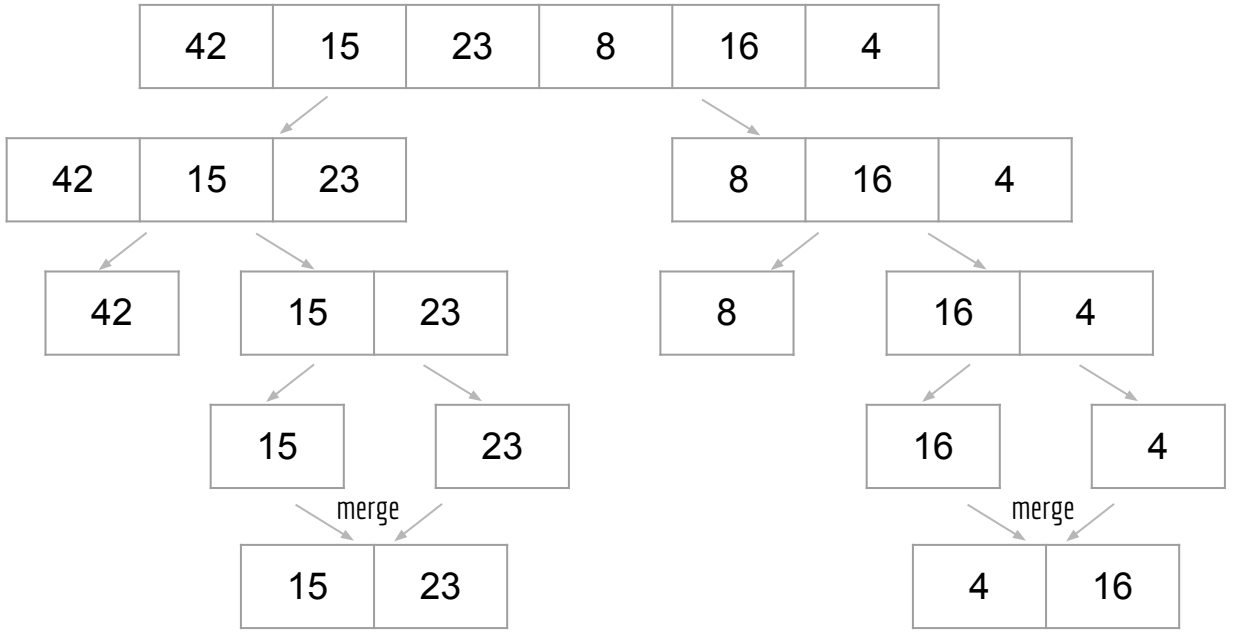
Merge Sort - Exemplo

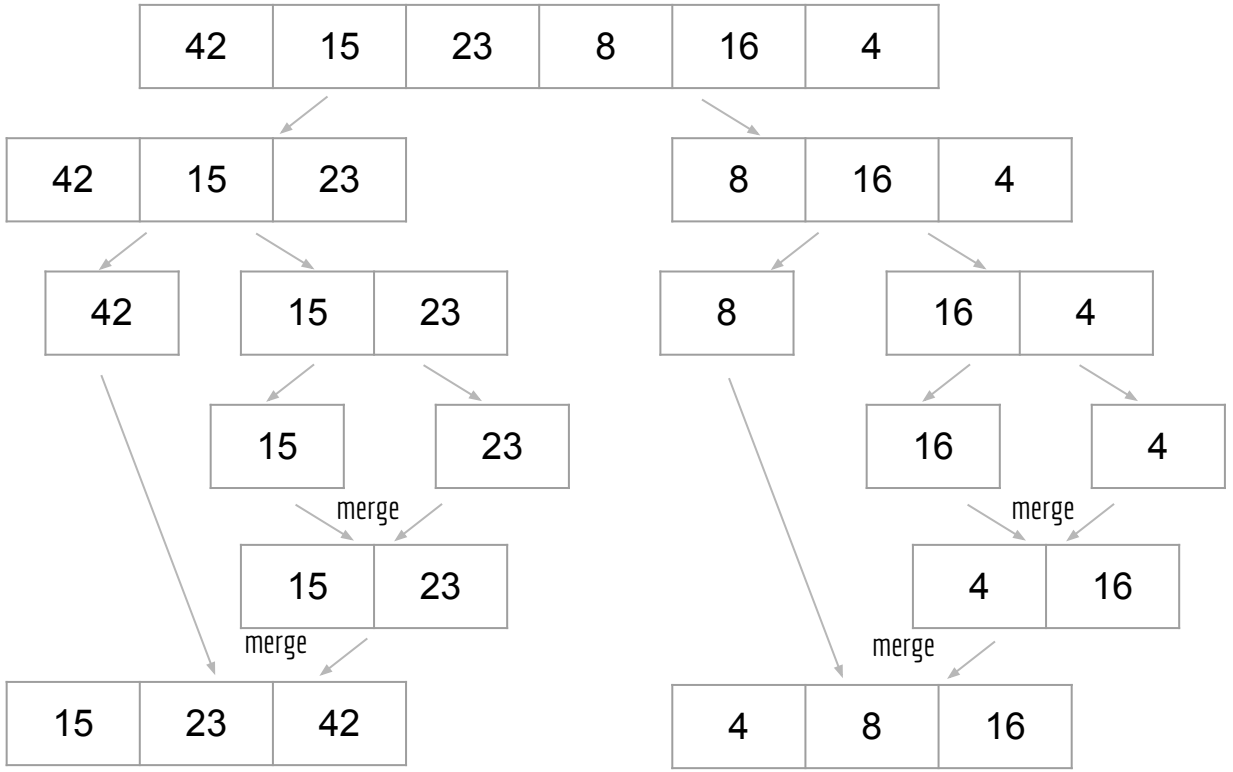
42	15	23	8	16	4

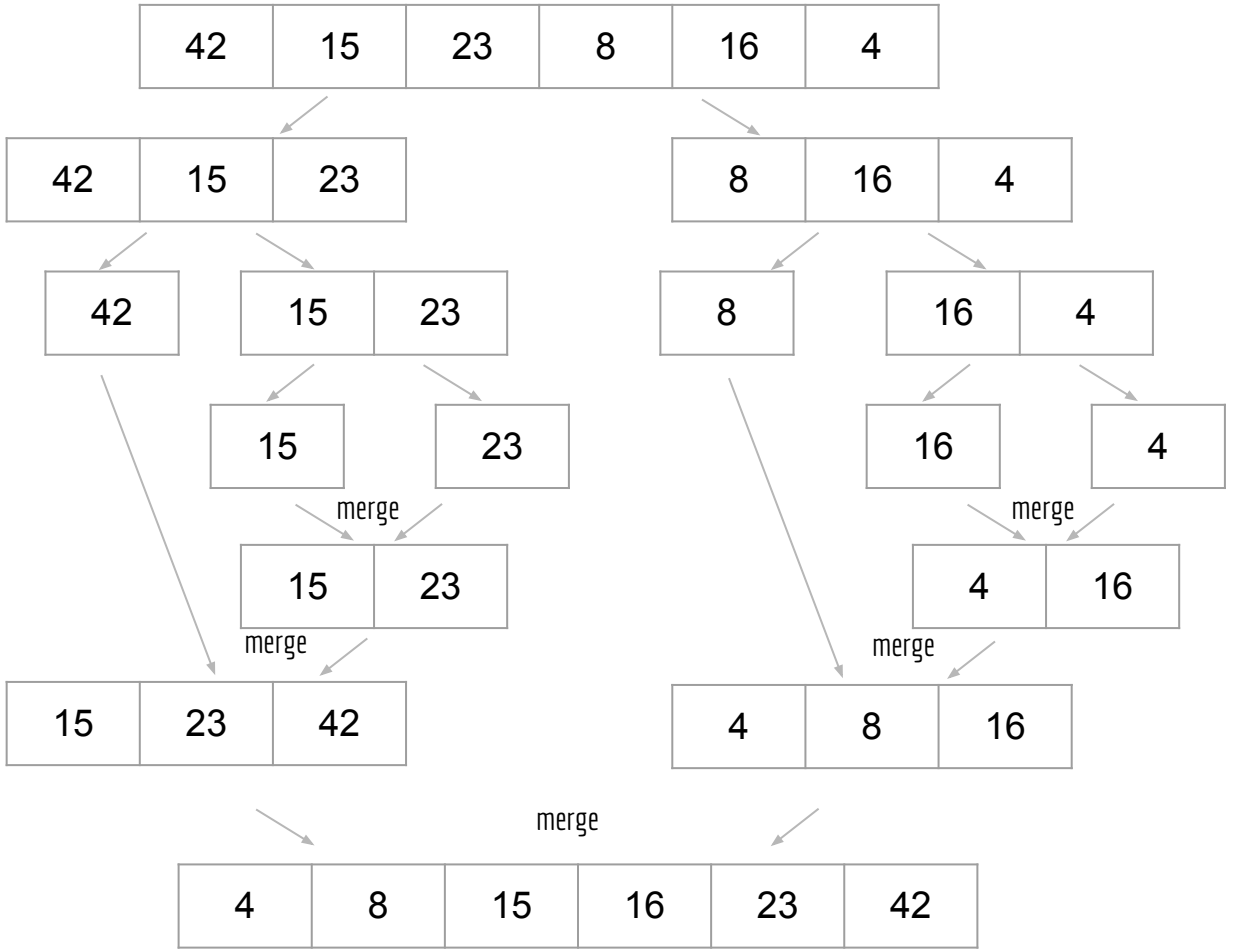

42
15
23
8
16
4
42
15
23
8
16
4


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
42
15
23
8
16
4
15
23
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
merge
15
23
4
16


42 15 23 8 16 4
42 15 23
8 16 4
42
15 23
8
16 4
15
23
16
4
merge
merge
15 23
4 16
merge
merge
15 23 42
4 8 16


42 15 23 8 16 4
42 15 23
8 16 4
42
15 23
8
16 4
15
23
16
4
merge
merge
15 23
4 16
merge
merge
15 23 42
4 8 16
merge
4 8 15 16 23 42

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 ≥ b
      retorne v
m ← ⌊(a+b)/2⌋

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
      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	

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

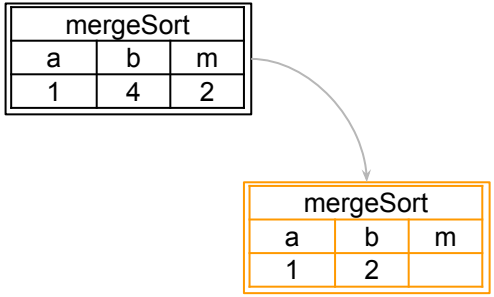
i	1	2	3	4
v[i]	42	15	23	8

```
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

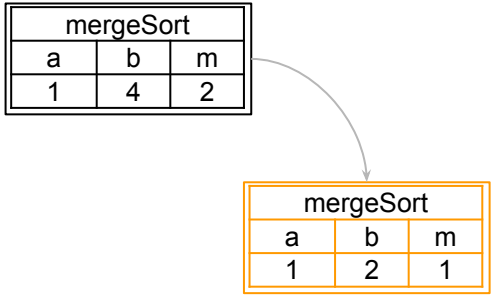
i	1	2	3	4
v[i]	42	15	23	8

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

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



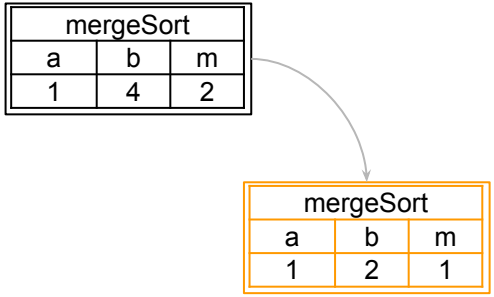
i	1	2	3	4
v[i]	42	15	23	8

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

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



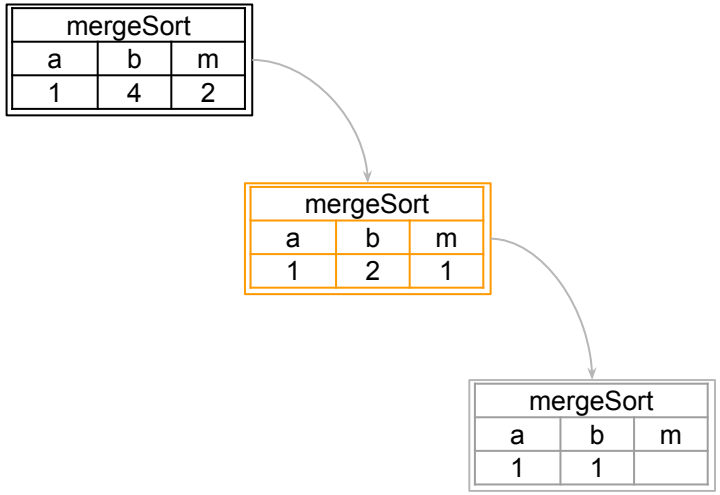
i	1	2	3	4
v[i]	42	15	23	8

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

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



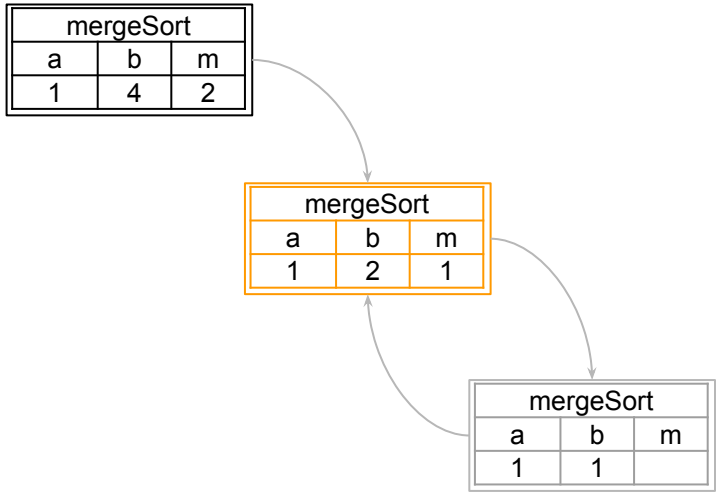
i	1	2	3	4
v[i]	42	15	23	8

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

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



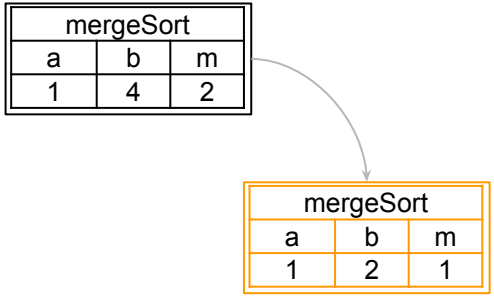
i	1	2	3	4
v[i]	42	15	23	8

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

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



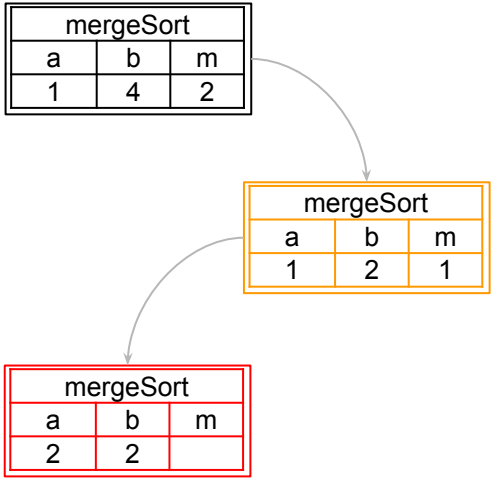
i	1	2	3	4
v[i]	42	15	23	8

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

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



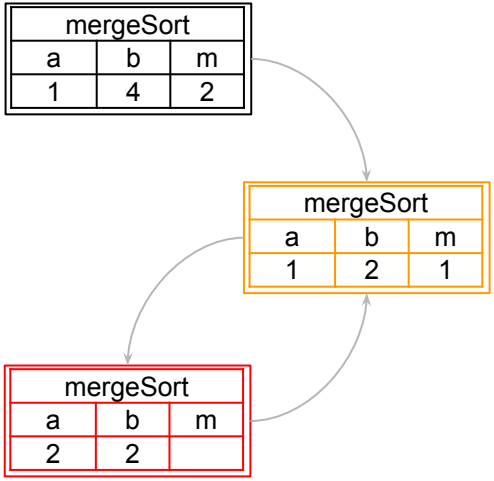
i	1	2	3	4
v[i]	42	15	23	8

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

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



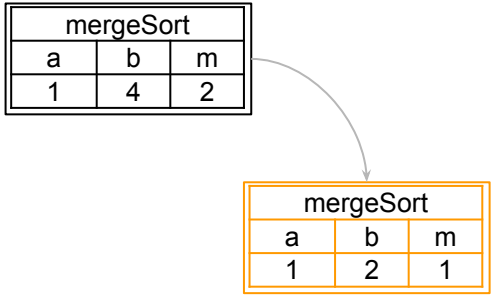
i	1	2	3	4
v[i]	42	15	23	8

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

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



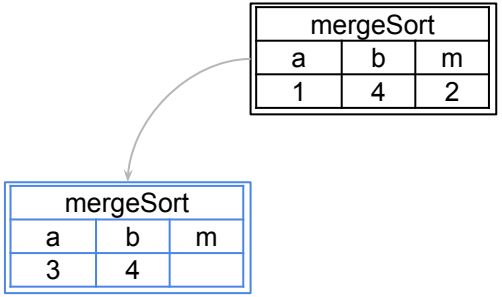
i	1	2	3	4
v[i]	15	42	23	8

```
função mergeSort (v,a,b)
se a ≥ b
      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

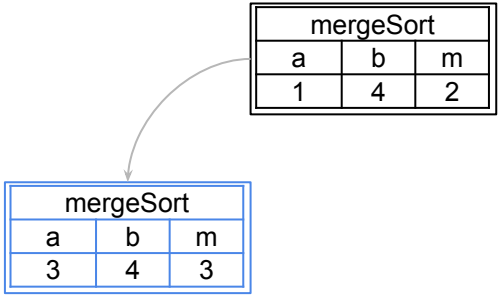
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



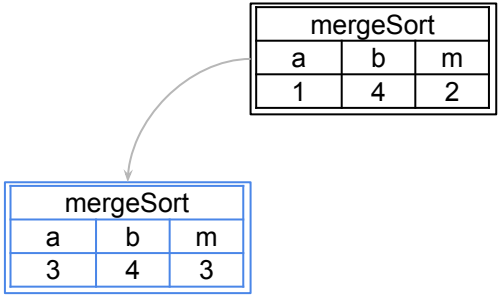
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



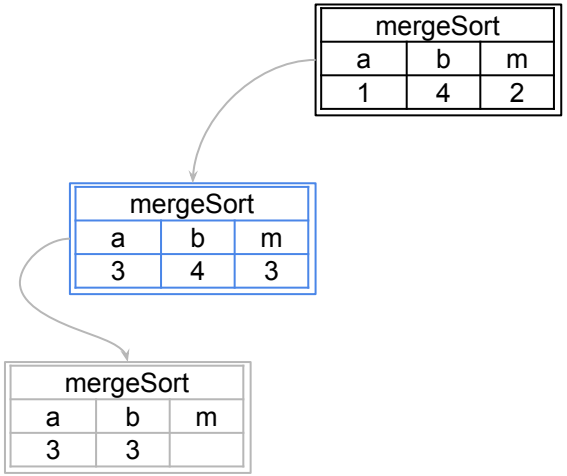
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



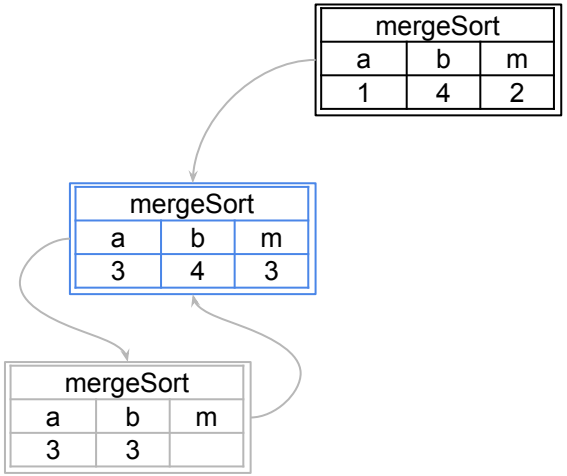
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



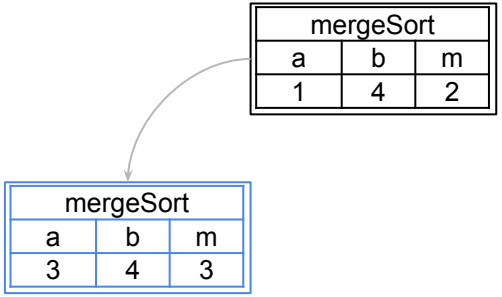
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



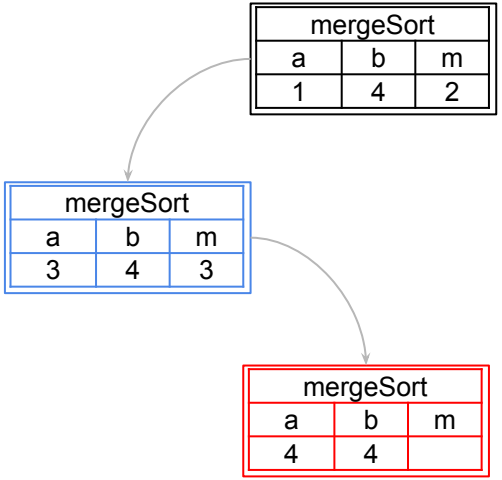
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



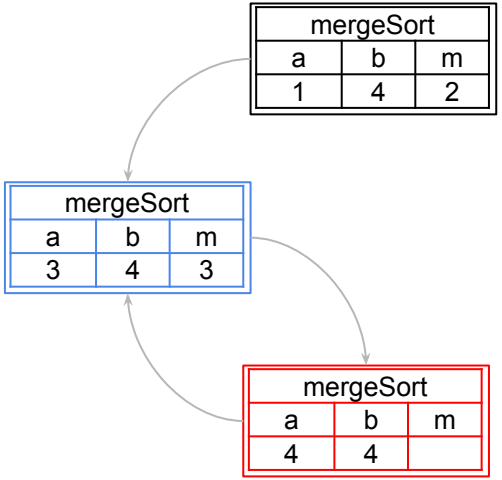
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



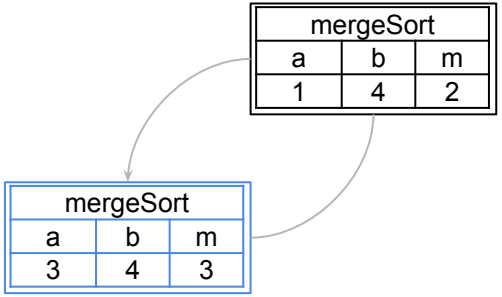
i	1	2	3	4
v[i]	15	42	23	8

```
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)
```



i	1	2	3	4
v[i]	15	42	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)
```



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

```
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

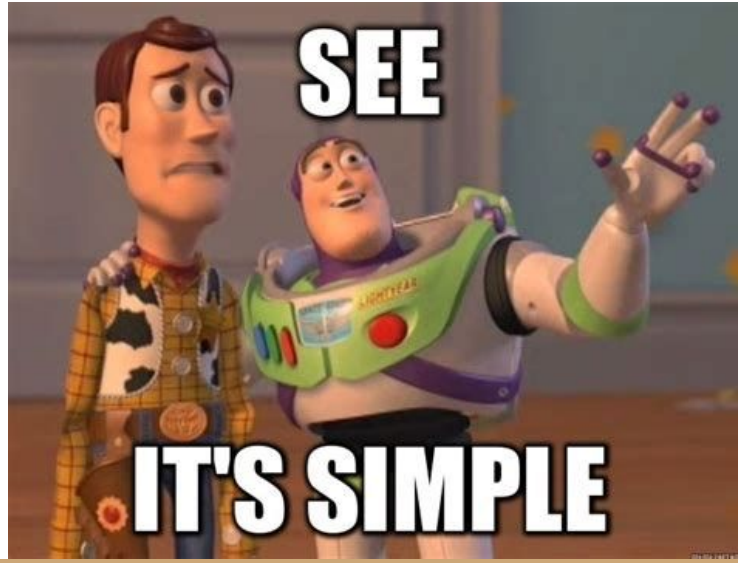
i	1	2	3	4
v[i]	8	15	23	42

```
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]	8	15	23	42

```
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

Considerando o número de comparações com elementos do vetor

$$C(n) = \ldots$$

```
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

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
      retorne v
m ← ⌊(a+b)/2⌋

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

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

Análise

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

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

```
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

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

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

```
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

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

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

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

```
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

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

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

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

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

```
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

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

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

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

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

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

```
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

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

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

```
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

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

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

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

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

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

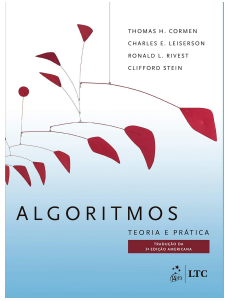
Exercícios

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*log2(n)
3. Implemente o merge sort em C

Referências

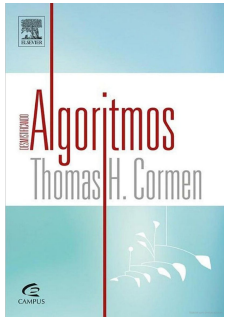
Renato Carmo. Algoritmos e Estruturas de Dados. www.inf.ufpr.br/renato

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

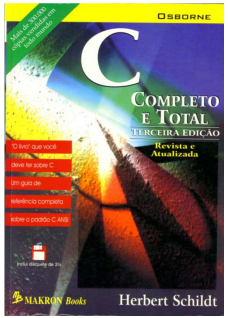


R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2014

T. Cormen. Desmistificando algoritmos. 2017.



H. Schildt. C completo e total. 1996



Licença