



Disciplina: Análise de Algoritmo  
Professor: Herbert Oliveira Rocha

Universidade Federal de Roraima  
Centro de Ciência e Tecnologia  
Departamento de Ciência da Computação



# Mergesort

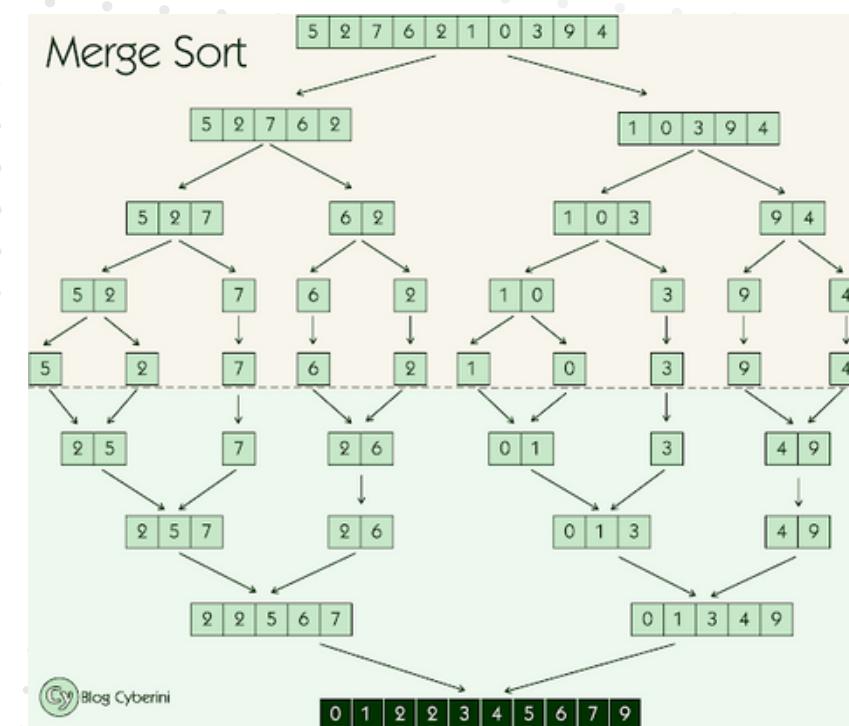
Rafael Nóbrega de Lima

Boa Vista/RR  
2023

# Algoritmo Mergesort

COMO ELE FUNCIONA:

O merge sort, ou ordenação por mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar. Sua ideia básica consiste em **Dividir** (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e **Conquistar** (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas).



# Pseudocódigo

```
MERGE-SORT( $A, p, r$ )
1   if  $p < r$ 
2     then  $q = \lfloor (p + r)/2 \rfloor$ 
3       MERGE-SORT( $A, p, q$ )
4       MERGE-SORT( $A, q + 1, r$ )
5       MERGE( $A, p, q, r$ )
```

```
MERGE( $A, p, q, r$ )
1    $n_1 = q - p + 1$ 
2    $n_2 = r - q$ 
3   sejam  $L[1..n_1 + 1]$  e  $R[1..n_2 + 1]$  novos arranjos
4   for  $i = 1$  to  $n_1$ 
5      $L[i] = A[p + i - 1]$ 
6   for  $j = 1$  to  $n_2$ 
7      $R[j] = A[q + j]$ 
8    $L[n_1 + 1] = \infty$ 
9    $R[n_2 + 1] = \infty$ 
10   $i = 1$ 
11   $j = 1$ 
12  for  $k = p$  to  $r$ 
13    if  $L[i] \leq R[j]$ 
14      then  $A[k] = L[i]$ 
15       $i = i + 1$ 
16    else  $A[k] = R[j]$ 
17       $j = j + 1$ 
```

# Código em C

```
1 #include <stdio.h>
2
3 void merge(int arr[], int left[], int left_size, int right[], int right_size)
4 {
5     int i = 0, j = 0, k = 0;
6
7     // Comparando e mesclando as duas metades
8     while (i < left_size && j < right_size) {
9         if (left[i] <= right[j]) {
10             arr[k] = left[i];
11             i++;
12         } else {
13             arr[k] = right[j];
14             j++;
15         }
16         k++;
17     }
18
19     // Copiando os elementos restantes de left[] (se houver)
20     while (i < left_size) {
21         arr[k] = left[i];
22         i++;
23         k++;
24     }
25
26     // Copiando os elementos restantes de right[] (se houver)
27     while (j < right_size) {
28         arr[k] = right[j];
29         j++;
30         k++;
31     }
32
33 void merge_sort(int arr[], int size) {
34     if (size < 2) {
35         return;
36     }
37
38     int mid = size / 2;
39     int left[mid], right[size - mid];
40
41     // Dividindo o array em duas metades
42     for (int i = 0; i < mid; i++) {
43         left[i] = arr[i];
44     }
45     for (int i = mid; i < size; i++) {
46         right[i - mid] = arr[i];
47     }
48
49     // Ordenando recursivamente as duas metades
50     merge_sort(left, mid);
51     merge_sort(right, size - mid);
52
53     // Mesclando as duas metades ordenadas
54     merge(arr, left, mid, right, size - mid);
55 }
```

A blurred background image showing a close-up of a computer screen displaying code in various programming languages like JavaScript, Python, and C.

# Complexidade e Custo

---

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 2T(n/2) + \Theta(n), & \text{se } n > 1 \end{cases}$$

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

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

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

$$T(n) = 2\left(2^2T\left(\frac{n}{2^3}\right) + \frac{2n}{2^2} + 2\right) + \frac{n}{2} + 1 + n + 1$$

$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 3n + 4 + 2 + 1$$

Substituindo por K, pois encontramos o padrão

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn + 2^{k-1} + 2^{k-2} + 2^0$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn + \sum_{i=0}^{K-1} 2^i$$

Voltando para o Caso Base, n sendo  $2^k$

$$T(n) = 2^k T\left(\frac{2^k}{2^k}\right) + kn + \sum_{i=0}^{K-1} 2^i$$

# Complexidade e Custo

---

E por último substituindo  $k = \log n$

$$T(n) = 2^{\log n} + (\log n)n + \sum_{i=0}^{\log n - 1} 2^i$$

$$T(n) = 2^{\log n} + n \log n + \sum_{i=0}^{\log n - 1} 2^i$$

$$T(n) = n + n \log n + \sum_{i=0}^{\log n - 1} 2^i$$

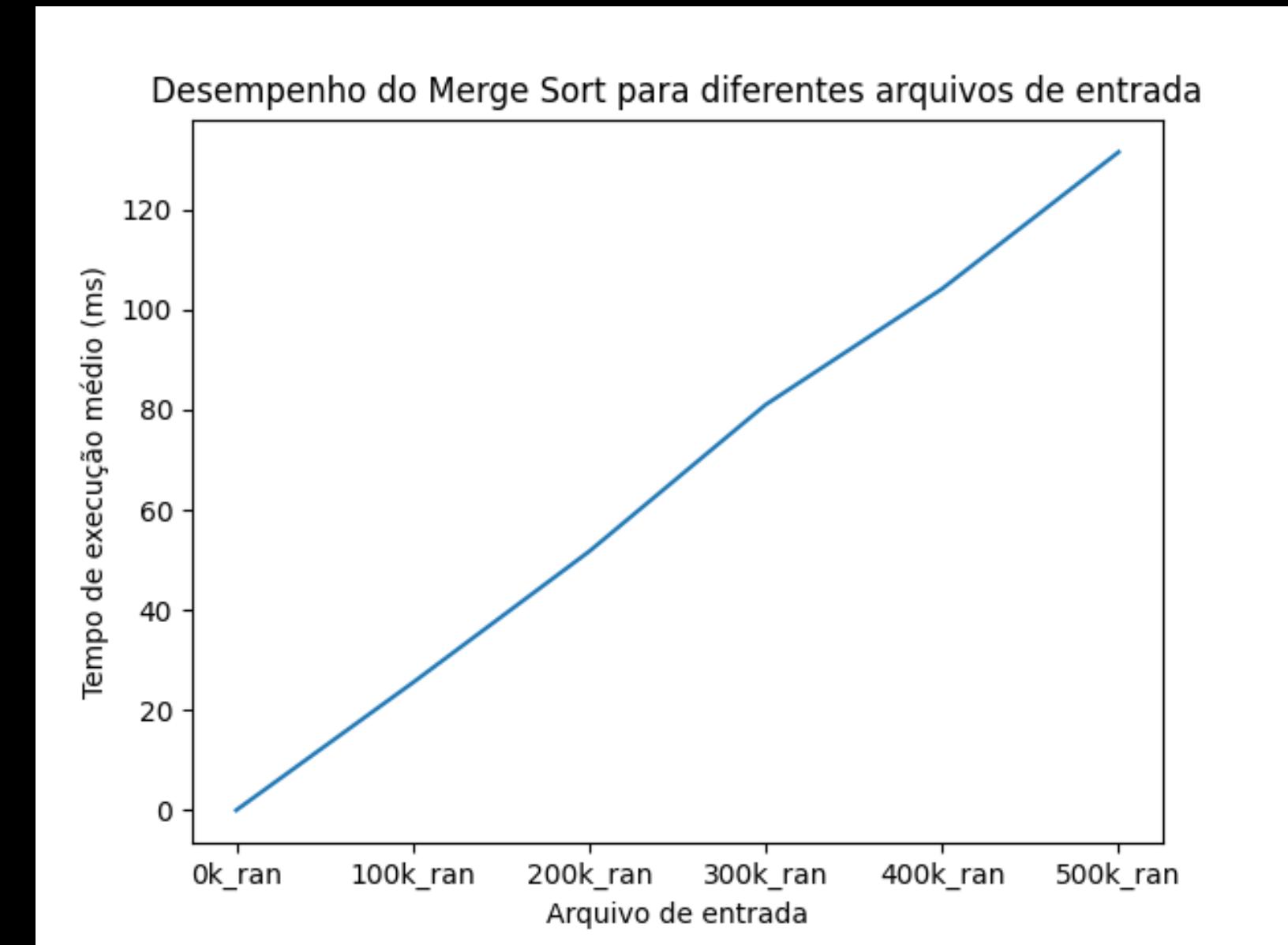
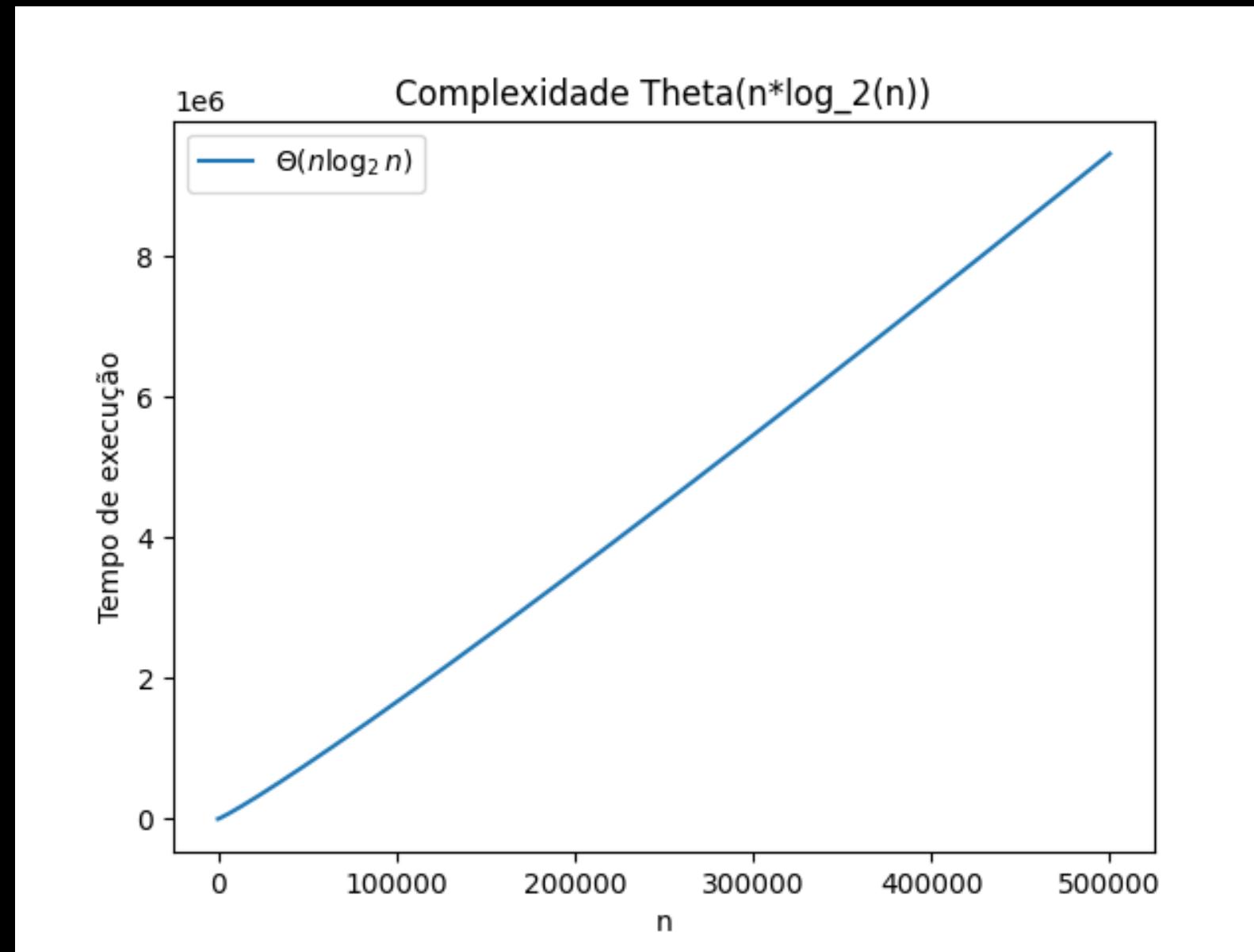
Resolvendo o Somatório que é uma PG

$$T(n) = n + n \log n + \frac{n}{2} - 1 = \text{custo}$$

Complexidade =  $n \log n$

---

# Gráficos



# Algoritmos melhores

**Quicksort:**O Quicksort é geralmente mais rápido do que o Mergesort em casos médios, mas pode ser menos eficiente em casos extremos.

**Complexidade para o tempo médio**  $O(n \log n)$

**Heapsort:**O Heapsort é geralmente mais eficiente do que o Mergesort em termos de espaço de memória, mas pode ser menos eficiente em casos extremos.

**Complexidade de tempo**  $O(n \log n)$ .

**OBRIGADO!!**