Merge Sort

e Suas Variações

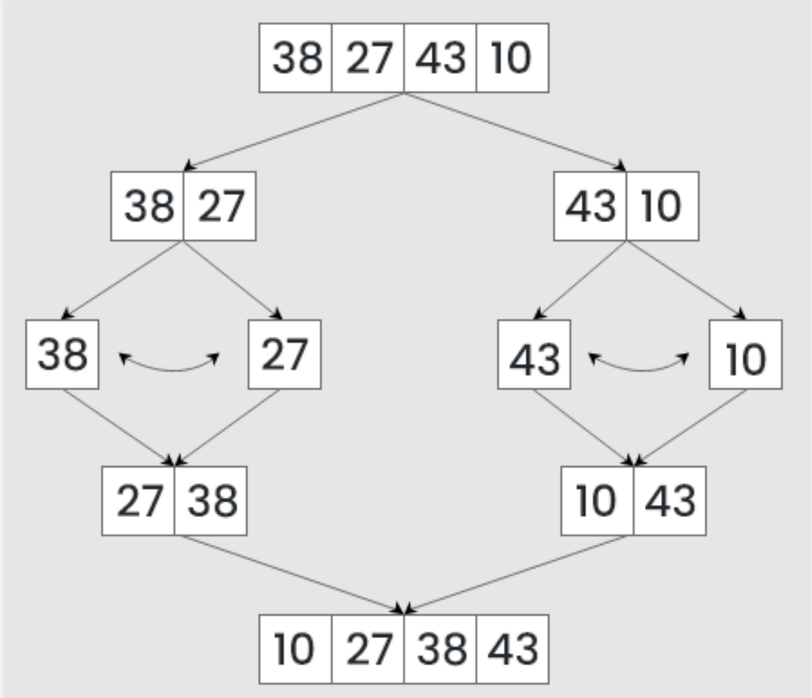
Comparando MergeSort clássico e Merge3Sort

Victor Hugo Souza Costa
Giovana Oliveira Moraes de Lima

MergeSort Clássico

Merge Sort Clássico - Introdução

Merge Sort é um algoritmo de ordenação por comparação, baseado em recursão e divisão do problema.



Etapas:

- Dividir o vetor em duas metades.
- Ordenar recursivamente.
- Mesclar as duas partes ordenadas.

Vantagens:

- Estável, eficiente para grandes entradas.
- Complexidade: O(nlogn)

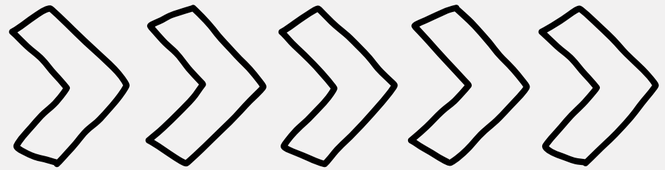
MergeSort Clássico - Código

MergeSort - Pseudocódigo

```
1  MERGE-SORT(A, p, r)
2      if p < r then
3      m ← (p + r) / 2;
4      MERGE-SORT(A, p, m);
5      MERGE-SORT(A, m + 1, r);
6      MERGE(A, p, m, r);
7  end if
```

MergeSort - Código em C

```c
1  void merge_sort(int *arr, int l, int r) {
2      if (l < r) {
3          int m = l + (r - l) / 2;
4          merge_sort(arr, l, m);
5          merge_sort(arr, m + 1, r);
6          merge(arr, l, m, r);
7      }
8  }
```



Merge3Sort

Merge3Sort - Introdução

Algoritmo de ordenação por comparação baseado em divisão e conquista

Etapas:

- Dividir o vetor em 3 partes aproximadamente iguais
- Ordenar recursivamente cada parte
- Mesclar as três partes ordenadas

Merge3Sort - Código

Merge3Sort - Pseudocódigo

```
1  MERGE3-SORT(A, p, r)
2      if p < r then
3      d ← (r + p ) / 3;
4      MERGE3-SORT(A, p, p + d);
5      MERGE3-SORT(A, p + d+1, r - d);
6      MERGE3-SORT(A, r - d + 1, r);
7      MERGE3(A, p, d, r);
8  end if
```

Merge3Sort - Código em C

```c
1  void merge3_sort(int A[], int p, int r)
2  {
3      if (p < r)
4      {
5          int d = (r + p) / 3;
6          merge3_sort(A, p, p + d);
7          merge3_sort(A, p + d + 1, r-d);
8          merge3_sort(A, r-d + 1, r);
9
10         merge3(A, p, d, r);
11     }
12 }
13
```

Análise de Complexidade

Análise de Complexidade

Análise:

- Método de iteração
- Função de custo: T(n) = nlogn + n
- Complexidade: O(nlogn)

Diferenças:

- Log na base 3
- 2 vetores temporários extras

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 3\,T\left(\frac{n}{3}\right) + \mathrm{n}, & \text{se } n > 1 \end{cases}$$

$$\begin{aligned} T\left(\frac{n}{3}\right) &= 3\left[3T\left(\frac{n/3}{3}\right) + \frac{n}{3}\right] + \mathrm{n} \\ &= 3^2 T\left(\frac{n}{3^2}\right) + 2\mathrm{n} \end{aligned}$$

$$\begin{aligned} T\left(\frac{n}{3^2}\right) &= 3^2\left[3T\left(\frac{n/3^2}{3}\right) + \frac{n}{3^2}\right] + 2\mathrm{n} \\ &= 3^3 T\left(\frac{n}{3^3}\right) + 3\mathrm{n} \end{aligned}$$

Padrão geral: $T(n) = 3^k T\left(\frac{n}{3^k}\right) + \mathrm{kn}$

Quando $n = 3^k$, temos $k = \log_3 n$

$$\begin{aligned} T(n) &= 3^k T\left(\frac{3^k}{3^k}\right) + \mathrm{kn} \\ &= 3^k + (\log_3 n)\cdot n \\ &= 3^{\log_3 n} + \mathrm{n}\log_3 n \end{aligned}$$

$$\begin{aligned} T(n) &= n + \mathrm{n}\log_3 n \\ &= \Theta(n \log n) \end{aligned}$$

Experimentação

Experimentação

Entradas:

- Python para geração
- Vetores de 1k à 100M
- Random, sorted e decrescent

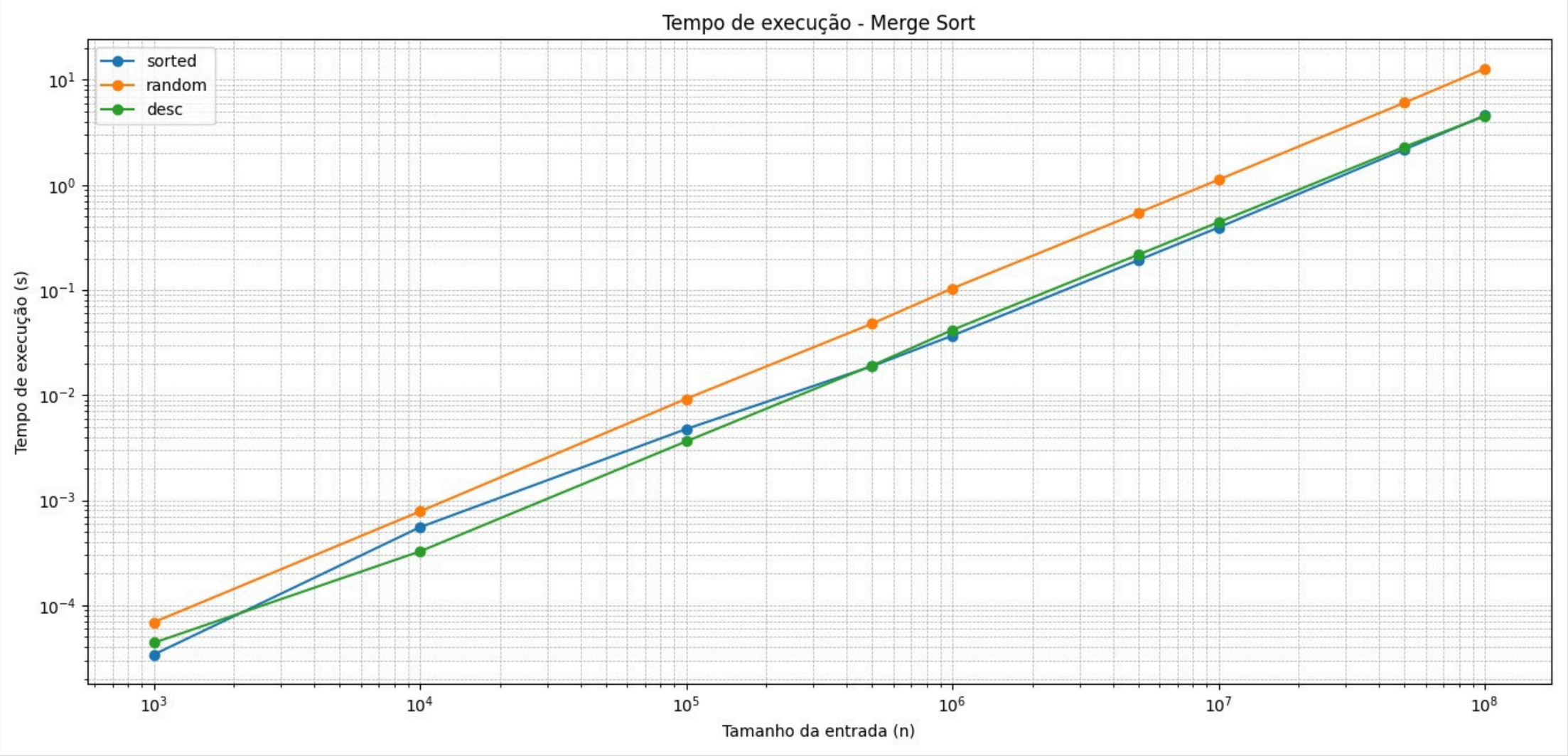
Medição:

- Merge_sort e Merge3_sort
- Programa em C para medição
- Resultado salvo em csv

```
Testes concluídos!
Resultados salvos em: resultados/resultados.csv

Resumo comparativo dos resultados:
==================================
Tipo     Tamanho       Merge     Merge3    Speedup
-------------------------------------------------
sorted       1000  0.000058s  0.000033s     1.75x
sorted      10000  0.000610s  0.000269s     2.26x
sorted     100000  0.007269s  0.003825s     1.90x
sorted     500000  0.027838s  0.015557s     1.78x
sorted    1000000  0.041701s  0.023350s     1.78x
sorted    5000000  0.222624s  0.122381s     1.81x
sorted   10000000  0.521821s  0.245919s     2.12x
sorted   50000000  2.717070s  1.301558s     2.08x
sorted  100000000  5.664248s  2.841486s     1.99x
random       1000  0.000118s  0.000061s     1.93x
random      10000  0.001821s  0.000419s     4.34x
random     100000  0.019087s  0.006637s     2.87x
random     500000  0.059534s  0.015531s     3.83x
random    1000000  0.112920s  0.030330s     3.72x
random    5000000  0.786103s  0.168505s     4.66x
random   10000000  1.456160s  0.351230s     4.14x
random   50000000  7.805046s  1.797473s     4.34x
random  100000000 16.169606s  3.668801s     4.40x
desc         1000  0.000056s  0.000030s     1.86x
desc        10000  0.000547s  0.000248s     2.20x
desc       100000  0.007017s  0.002734s     2.56x
desc       500000  0.027546s  0.014711s     1.87x
desc      1000000  0.040708s  0.022397s     1.81x
desc      5000000  0.218068s  0.115151s     1.89x
desc     10000000  0.448526s  0.233297s     1.92x
desc     50000000  2.426276s  1.246683s     1.94x
desc    100000000  4.876225s  2.680758s     1.81x
```

MergeSort Clássico - Gráfico



Merge3Sort - Gráfico

Algoritmos Mais Eficientes

Aloritmos mais Eficientes

Algoritmos:

- Tim sort:
  - Complexidade: Θ(nlogn)
- Radix Sort:
  - Complexidade: Θ(nk)
- Bucket sort:
  - Complexidade: Θ(n+k)