Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros

Avaliação Teórica e Experimental de Algoritmos de Ordenação: Shell Sort e Cycle Sort

Discentes: Luciano Sousa Barbosa; Pedro Henrique Silva Rodrigues; Tiago Lima de Moura.

Docente: Raí Araújo de Miranda
Disciplina: Projeto e Análise de Algoritmos

Sumário

Introdução

- A ordenação de dados é uma operação fundamental em Ciência da Computação;
- A eficiência dos algoritmos de ordenação impacta diretamente o desempenho dos sistemas, especialmente com grande volume de dados;
- Diferentes algoritmos apresentam comportamentos distintos conforme uma série de fatores.

Introdução

- Objetivos:
  - Realizar uma avaliação comparativa entre o *Shell Sort* e *Cycle Sort*;
  - Relacionar resultados empíricos com previsões teóricas;
  - Analisar desempenho, complexidade assintótica e sensibilidade à ordenação inicial.

Referencial Teórico - Shell Sort

- Proposto por Shell em 1959;
- Generalização do *Insertion Sort*;
- Objetivo: Reduzir o número de deslocamentos durante a ordenação;
- Busca minimizar o impacto de entradas grandes e desordenadas.

Referencial Teórico - Shell Sort

- Princípio de funcionamento:
  - Utiliza comparações entre elementos distantes;
  - O vetor é dividido em subsequências definidas por um intervalo (*gap*);
  - O valor do *gap* é reduzido progressivamente até 1.

Referencial Teórico - Shell Sort

- A escolha da sequência de *gaps* é um fator crítico;
- Existem diversas sequências propostas na literatura.

Referencial Teórico - Shell Sort

- Complexidade Assintótica:
  - Em cenários práticos: $O(n^{3/2})$.
  - No pior caso teórico: $O(n^2)$.

Referencial Teórico - Shell Sort

- Características Operacionais:
  - Algoritmo *in-place*;
  - Não estável;
  - Implementação relativamente fácil;
  - Bom desempenho em entradas parcialmente ordenadas.

Demonstração - Shell Sort

Salto: ?

Comparações: 0
Trocas: 0

4	6	1	7	0	5	2	3

Demonstração - Shell Sort

Sequência utilizada:

$$h = \frac{3^k - 1}{2}$$

Onde h é o valor do salto e k é igual ao número da varredura.

O Número de varreduras é dado por:

$$\log_3(N + 1)$$

Onde N é o tamanho do vetor.

Demonstração - Shell Sort

Definindo o número de varreduras no nosso algoritmo:

$$\log_3(8+1) = 2$$

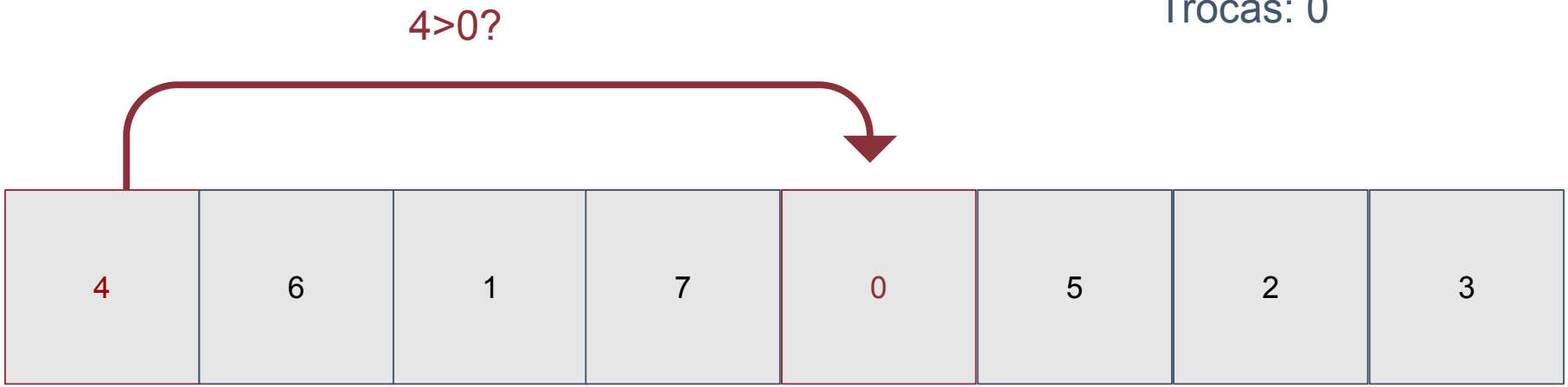
O valor do salto na varredura inicial será:

$$salto = \frac{3^2 - 1}{2} = 4$$

Demonstração - Shell Sort

Salto: 4

Comparações: 1
Trocas: 0

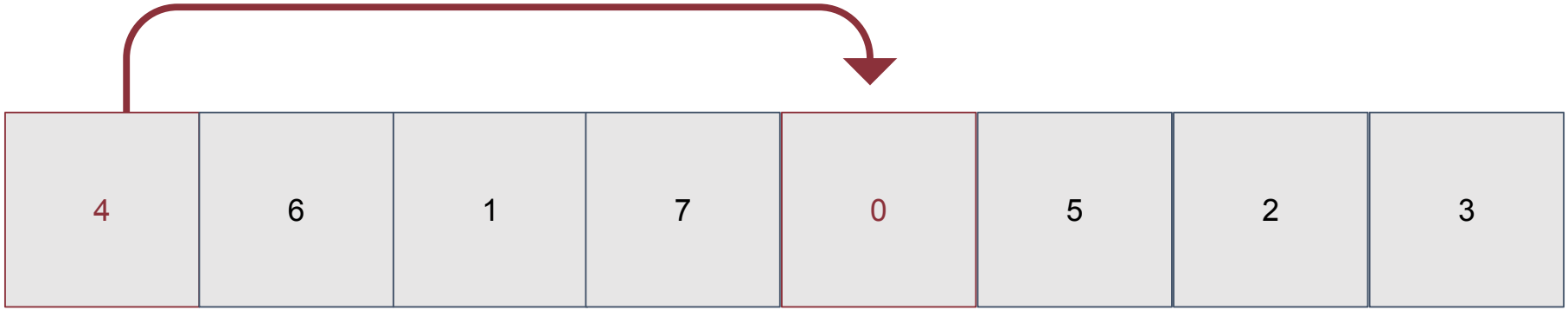


Demonstração - Shell Sort

Salto: 4

Comparações: 1
Trocas: 0

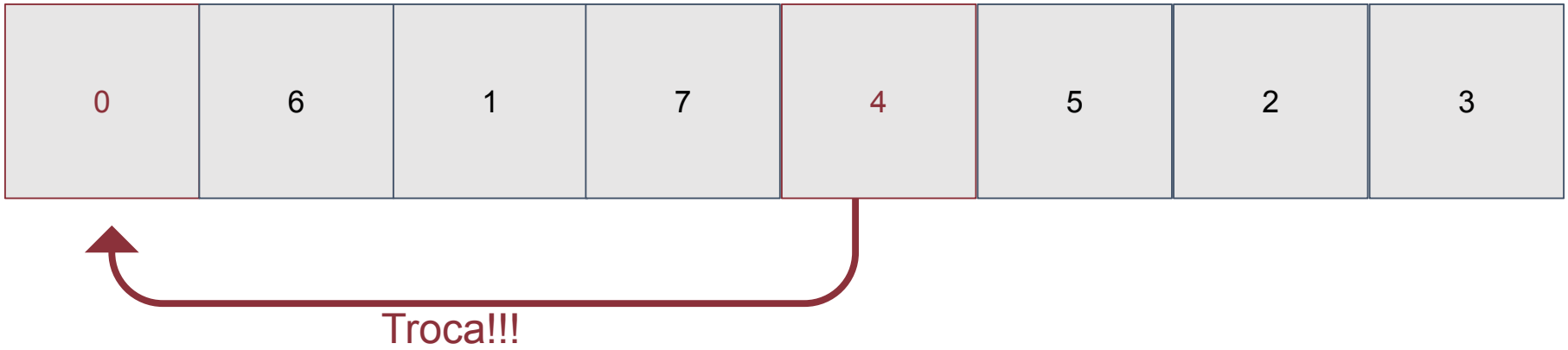
4>0? sim !!!



Demonstração - Shell Sort

Salto: 4

Comparações: 1
Trocas: 1

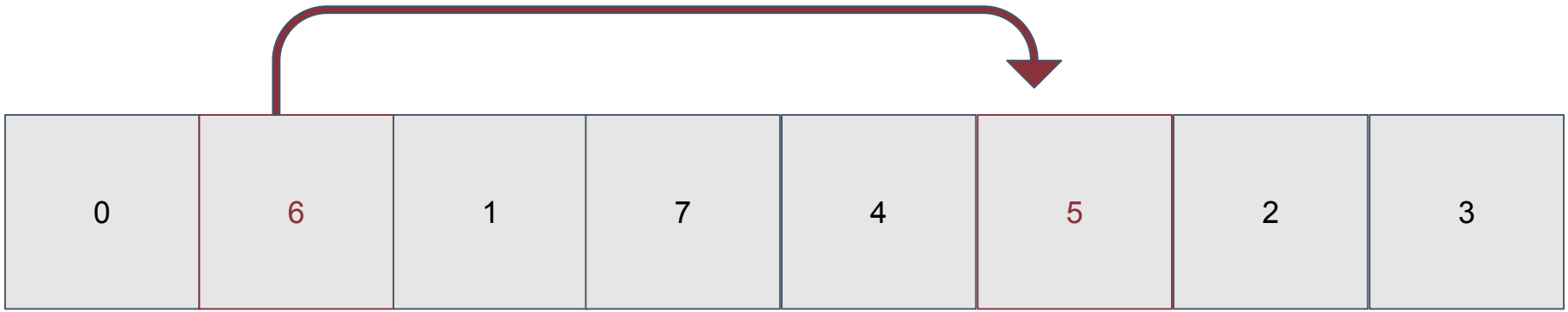


Demonstração - Shell Sort

Salto: 4

Comparações: 2
Trocas: 1

6>5?

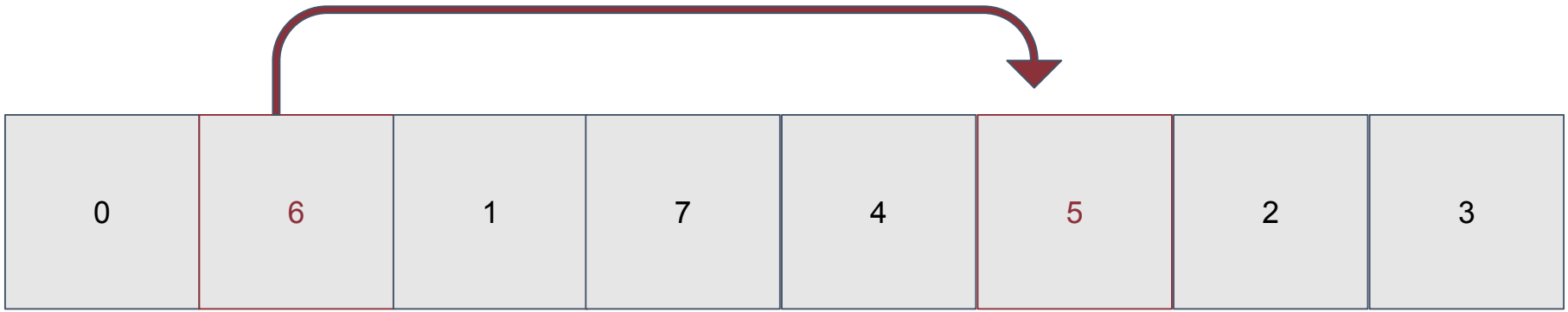


Demonstração - Shell Sort

Salto: 4

Comparações: 2
Trocas: 1

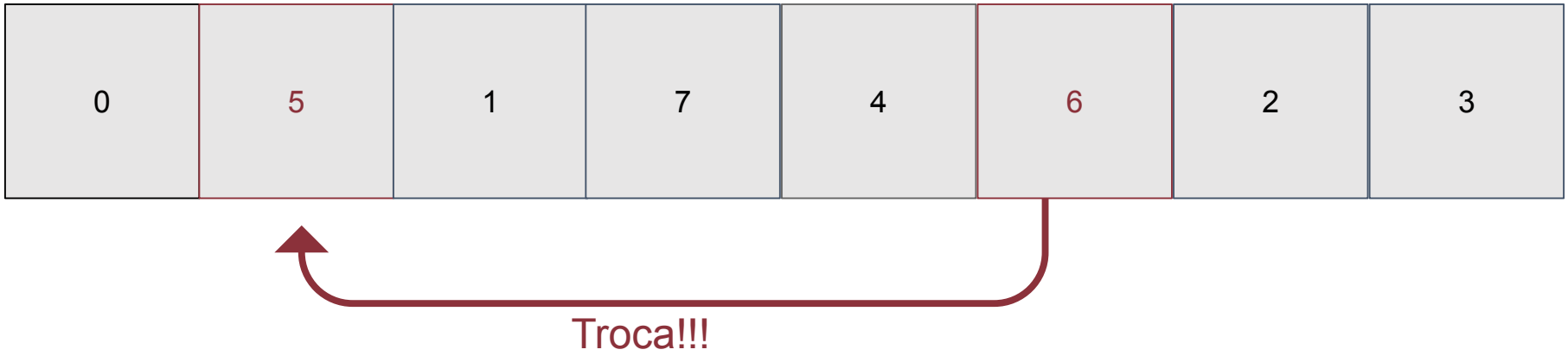
6>5? sim!!!



Demonstração - Shell Sort

Salto: 4

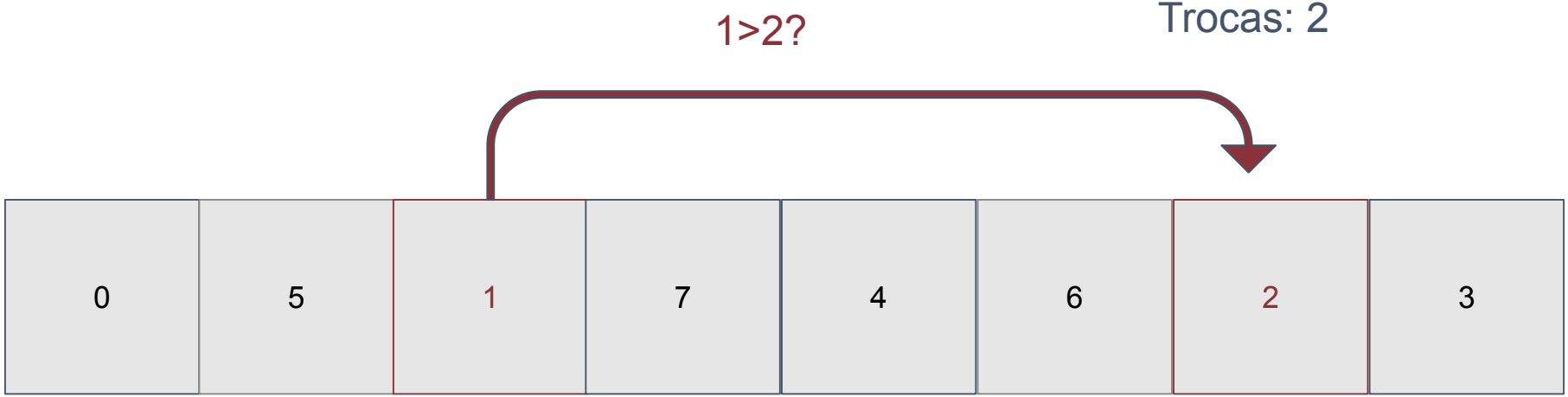
Comparações: 2
Trocas: 2



Demonstração - Shell Sort

Salto: 4

Comparações: 3
Trocas: 2

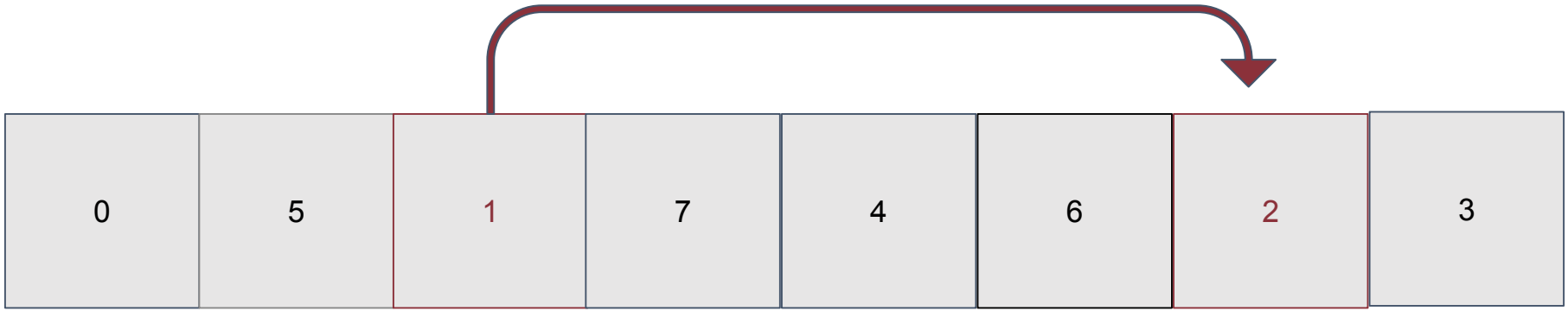


Demonstração - Shell Sort

Salto: 4

Comparações: 3
Trocas: 2

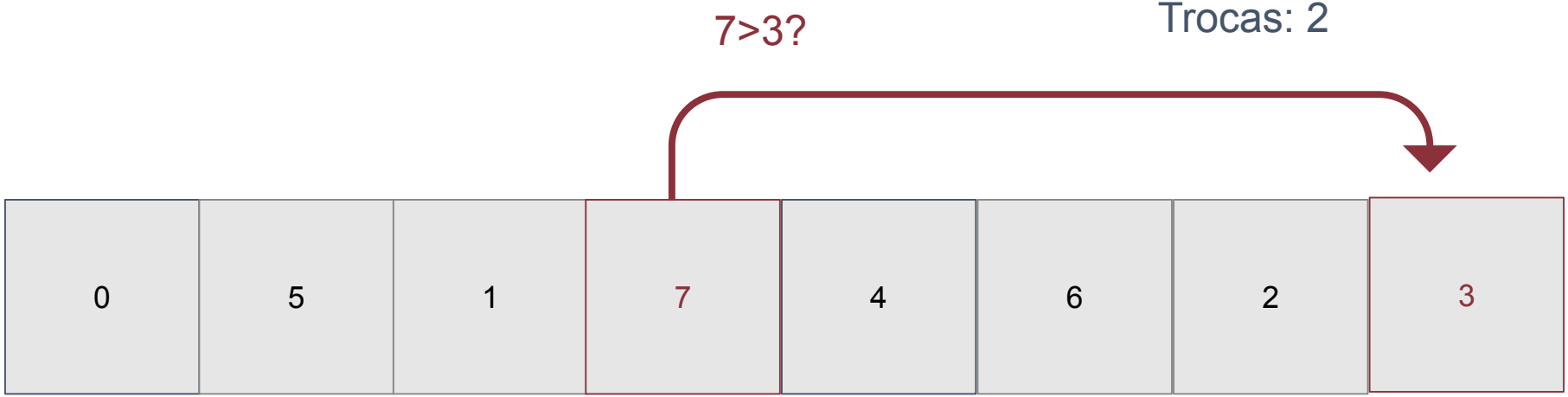
1>2? não!!!



Demonstração - Shell Sort

Salto: 4

Comparações: 4
Trocas: 2

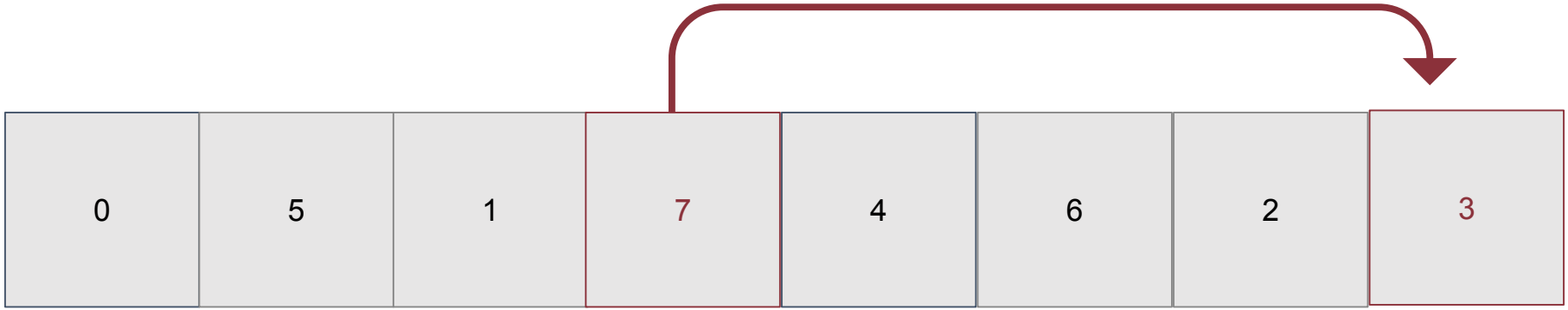


Demonstração - Shell Sort

Salto: 4

Comparações: 4
Trocas: 2

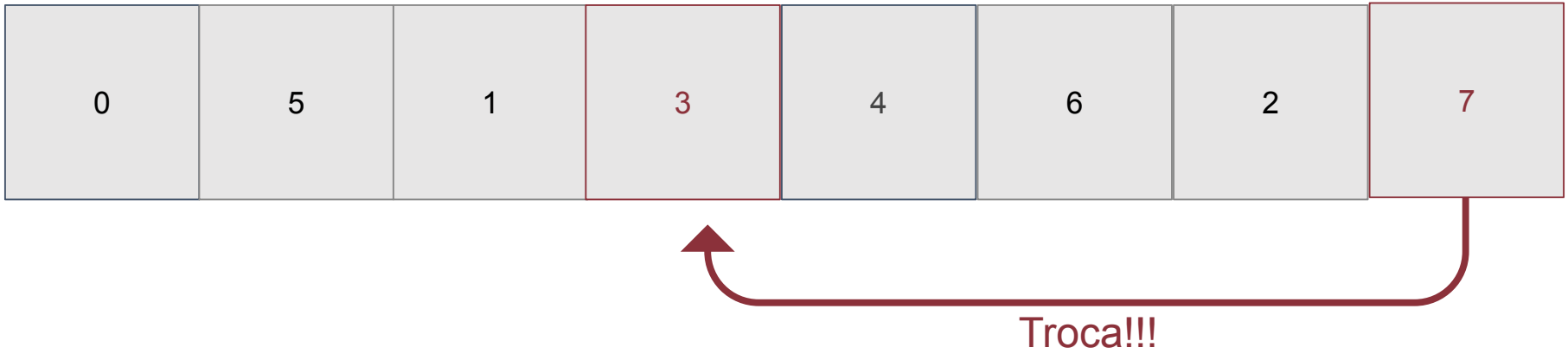
7>3? sim!!!



Demonstração - Shell Sort

Salto: 4

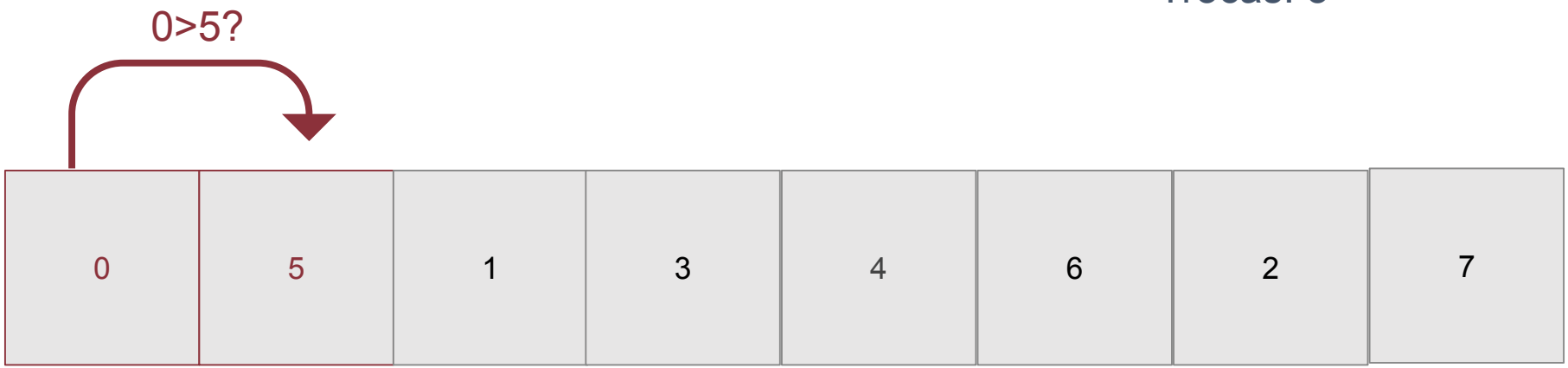
Comparações: 4
Trocas: 3



Demonstração - Shell Sort

Salto: 1

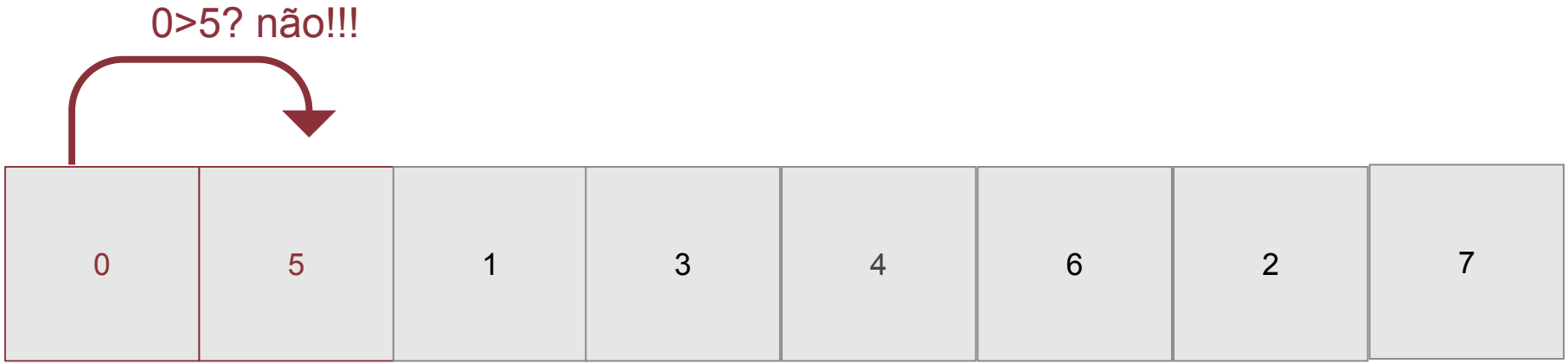
Comparações: 5
Trocas: 3



Demonstração - Shell Sort

Salto: 1

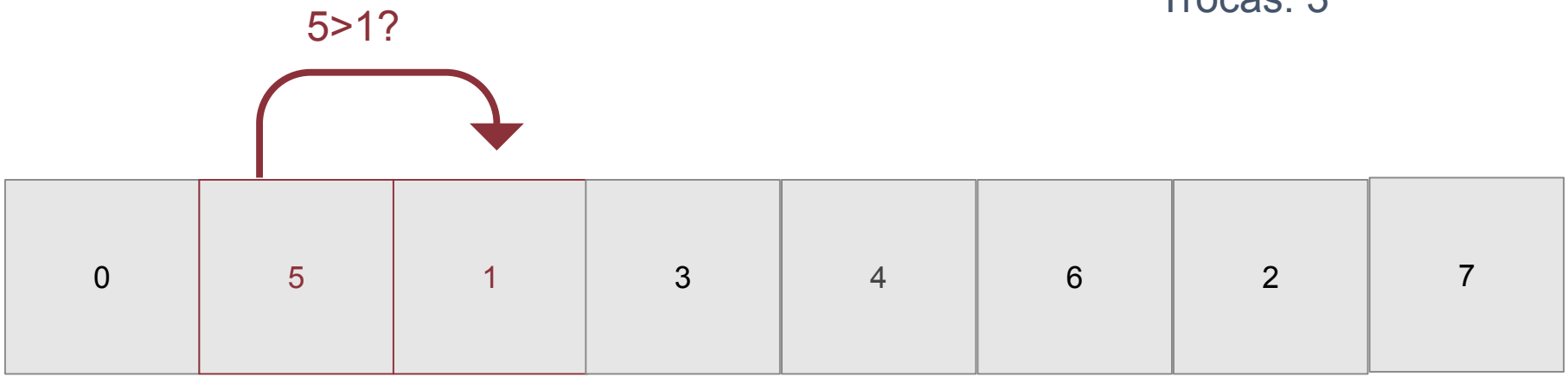
Comparações: 5
Trocas: 3



Demonstração - Shell Sort

Salto: 1

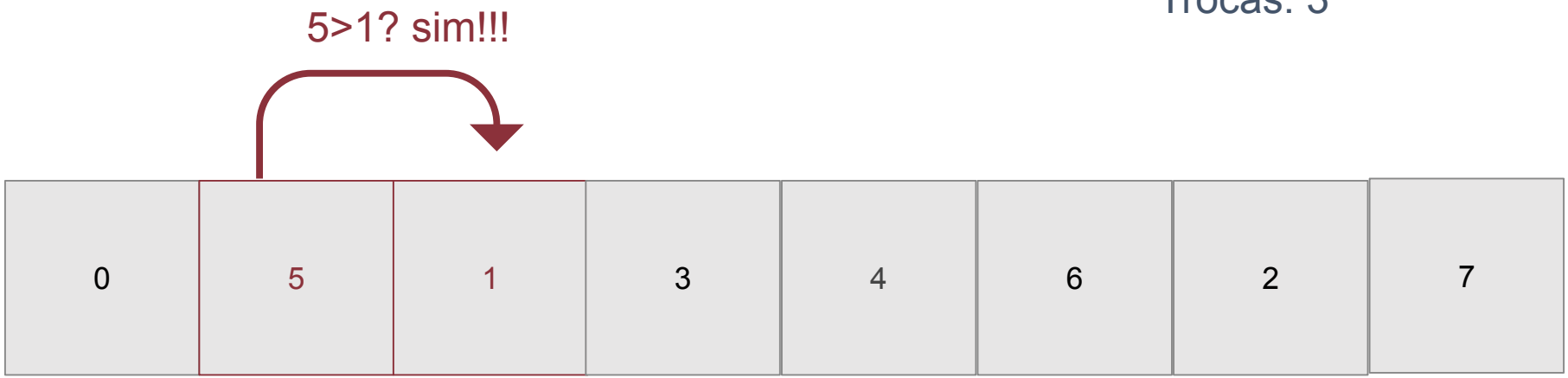
Comparações: 6
Trocas: 3



Demonstração - Shell Sort

Salto: 1

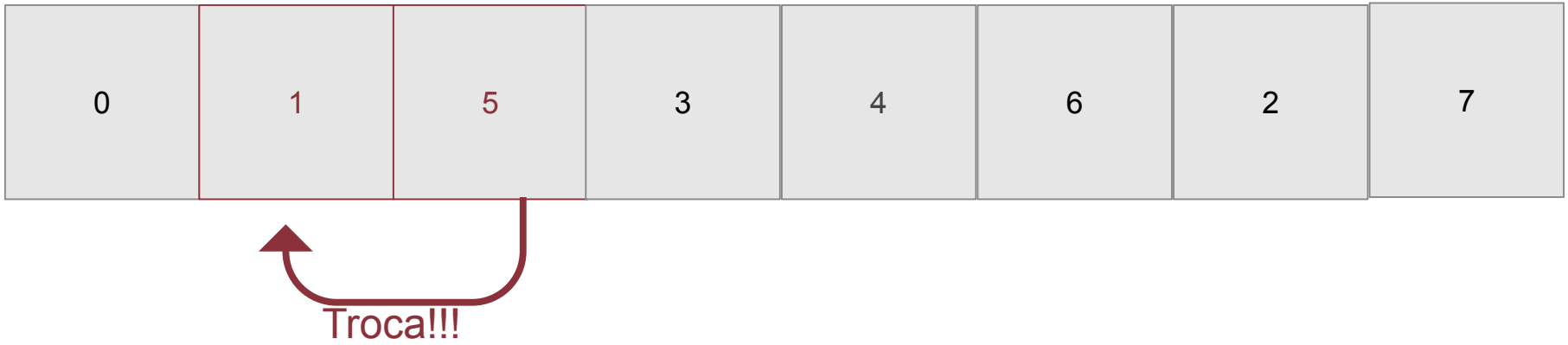
Comparações: 6
Trocas: 3



Demonstração - Shell Sort

Salto: 1

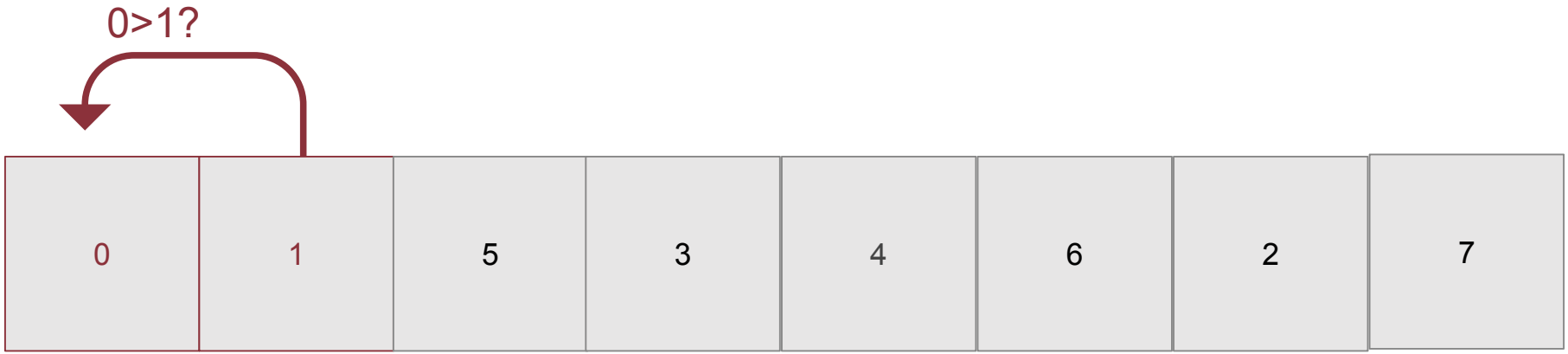
Comparações: 6
Trocas: 4



Demonstração - Shell Sort

Salto: 1

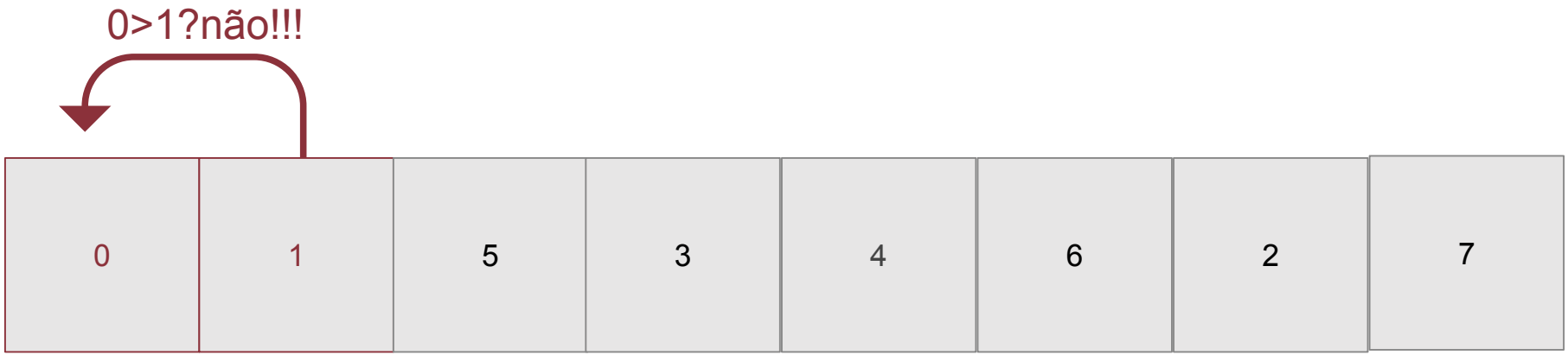
Comparações: 7
Trocas: 4



Demonstração - Shell Sort

Salto: 1

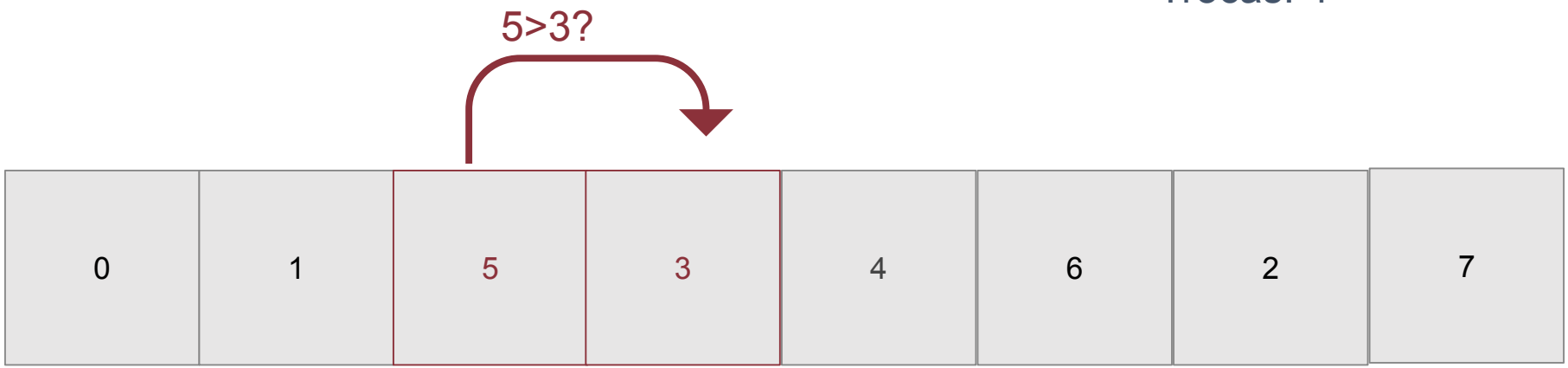
Comparações: 7
Trocas: 4



Demonstração - Shell Sort

Salto: 1

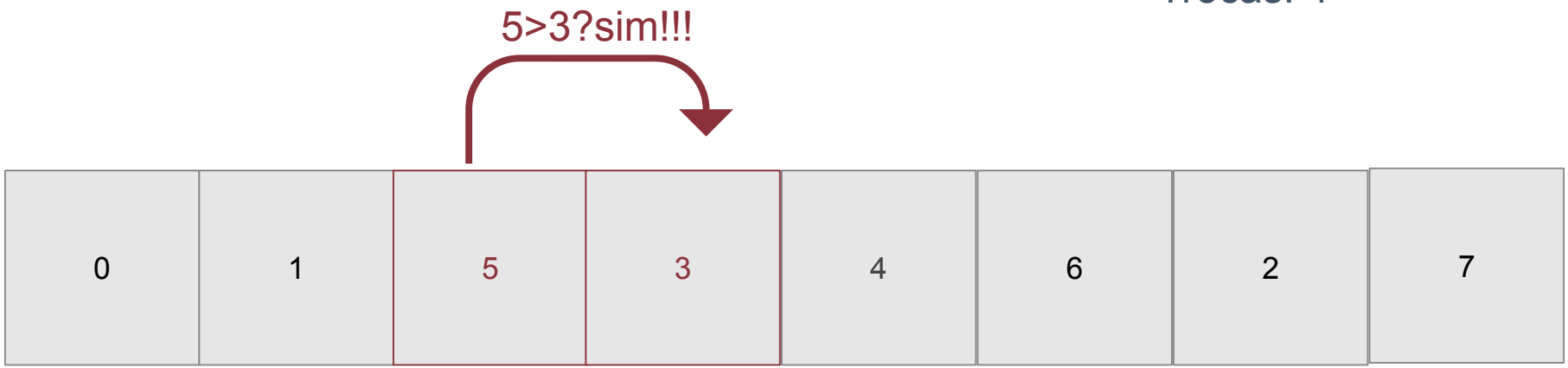
Comparações: 8
Trocas: 4



Demonstração - Shell Sort

Salto: 1

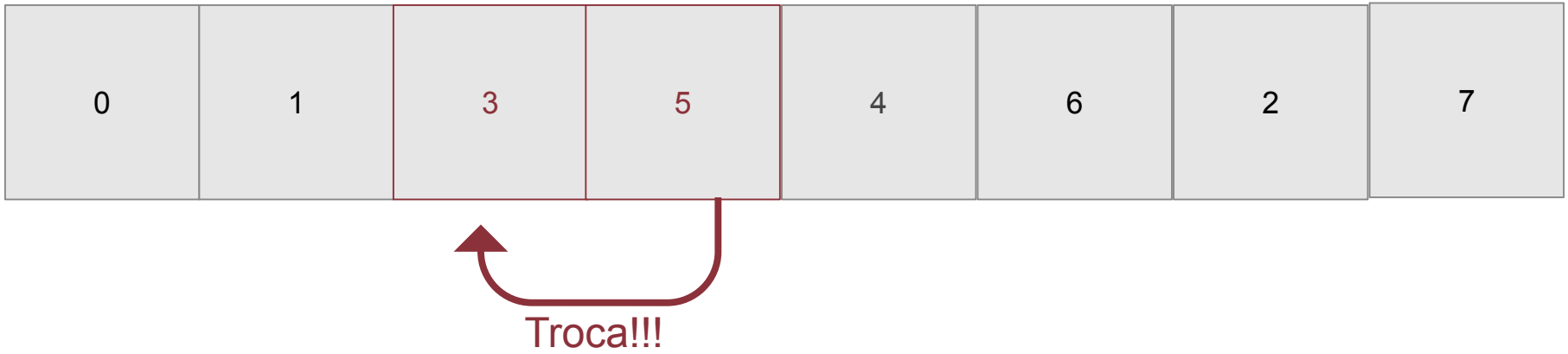
Comparações: 8
Trocas: 4



Demonstração - Shell Sort

Salto: 1

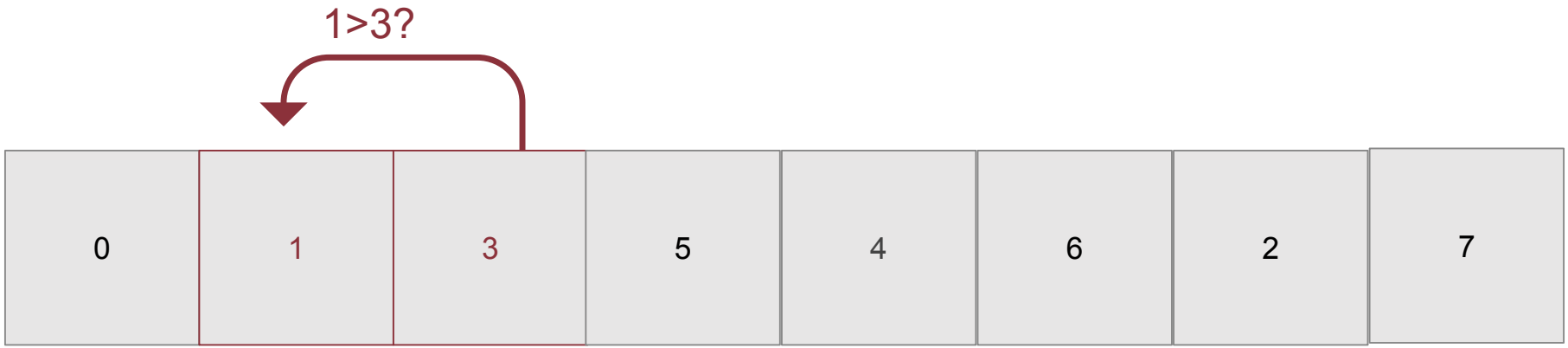
Comparações: 8
Trocas: 5



Demonstração - Shell Sort

Salto: 1

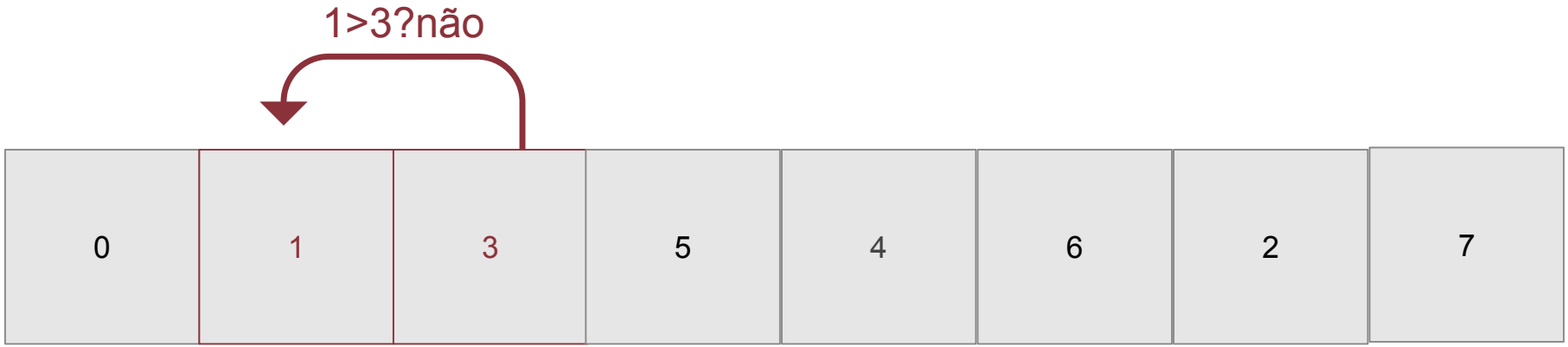
Comparações: 9
Trocas: 5



Demonstração - Shell Sort

Salto: 1

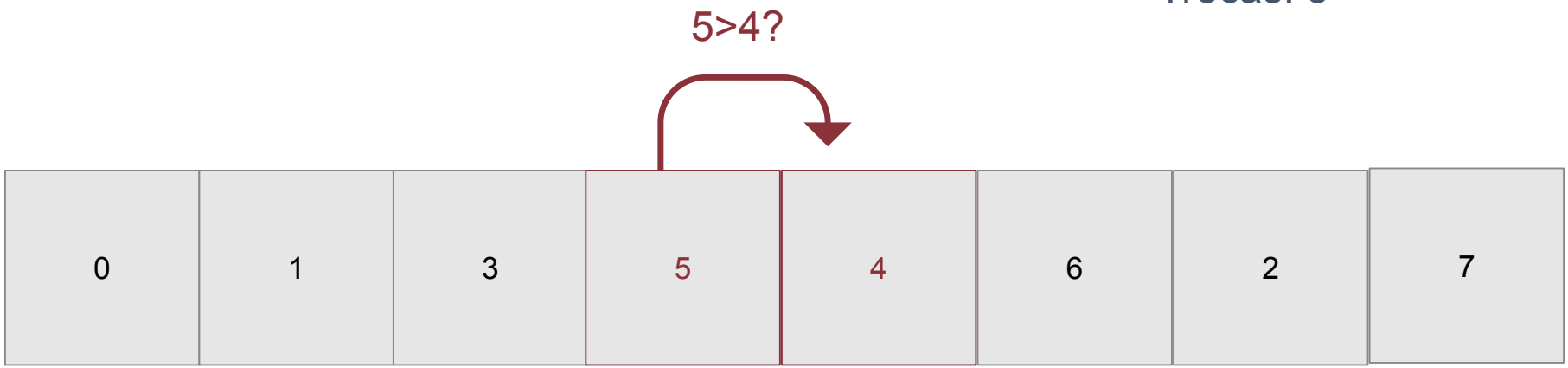
Comparações: 9
Trocas: 5



Demonstração - Shell Sort

Salto: 1

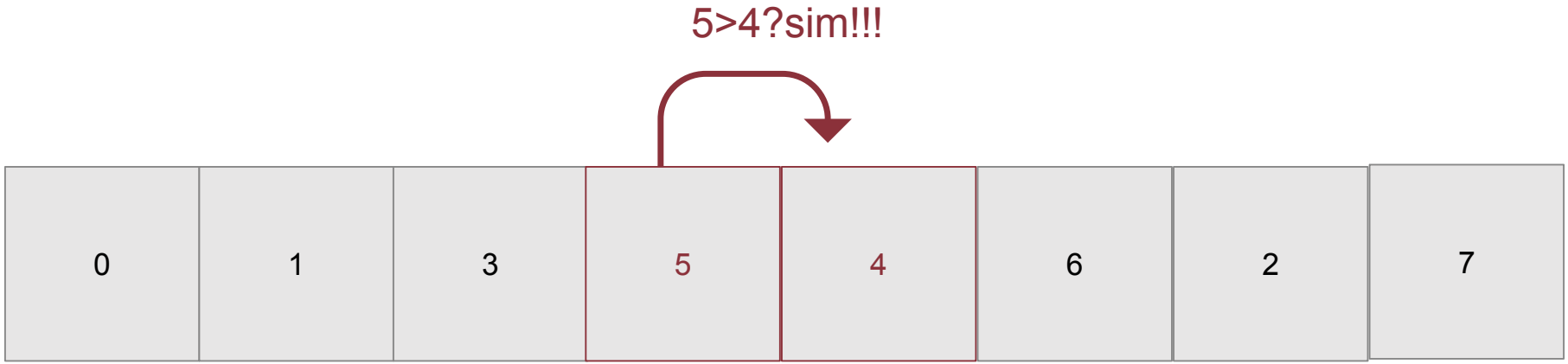
Comparações: 10
Trocas: 5



Demonstração - Shell Sort

Salto: 1

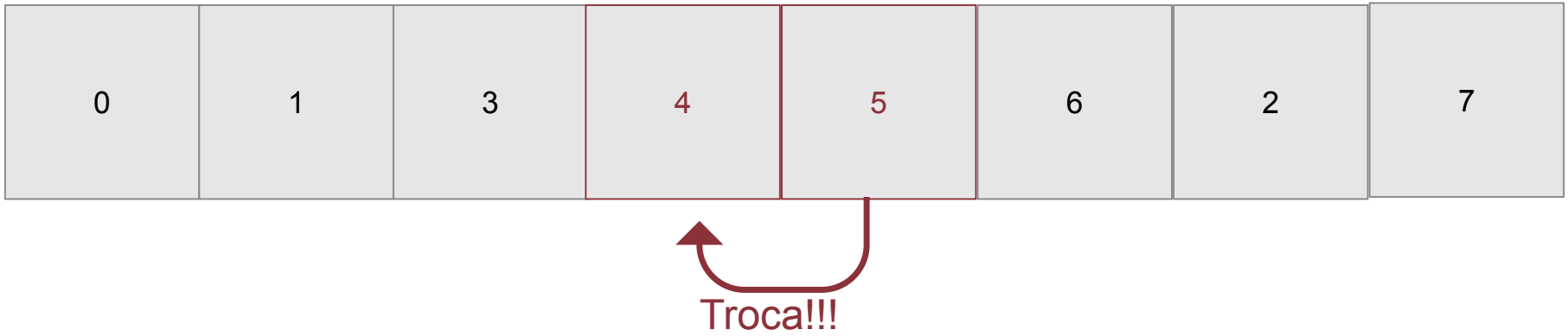
Comparações: 10
Trocas: 5



Demonstração - Shell Sort

Salto: 1

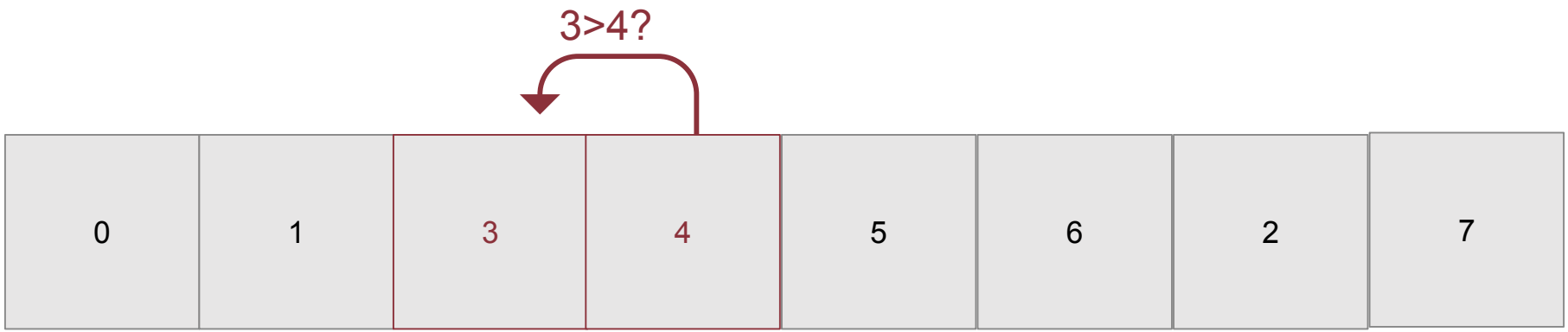
Comparações: 10
Trocas: 6



Demonstração - Shell Sort

Salto: 1

Comparações: 11
Trocas: 6



Demonstração - Shell Sort

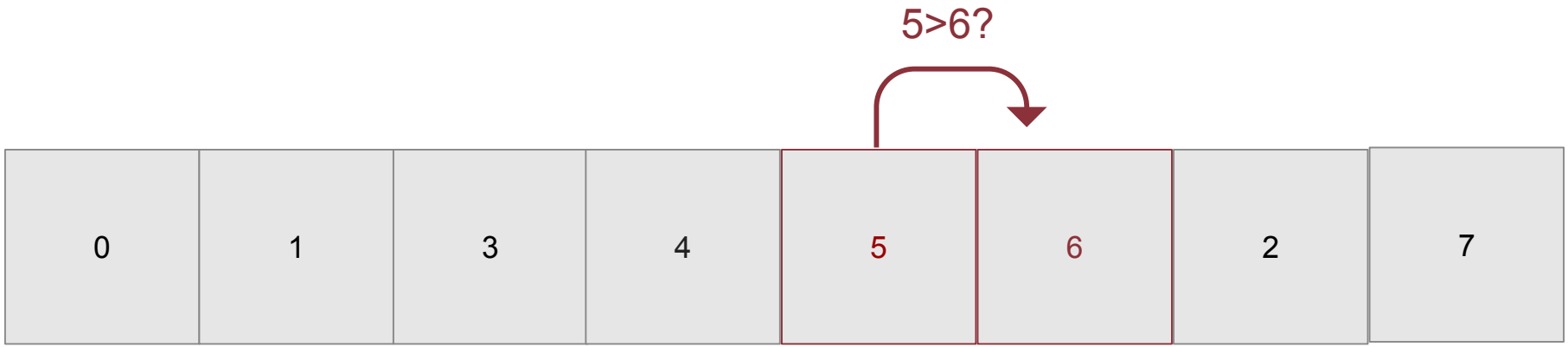
Salto: 1

Comparações: 11
Trocas: 6

Demonstração - Shell Sort

Salto: 1

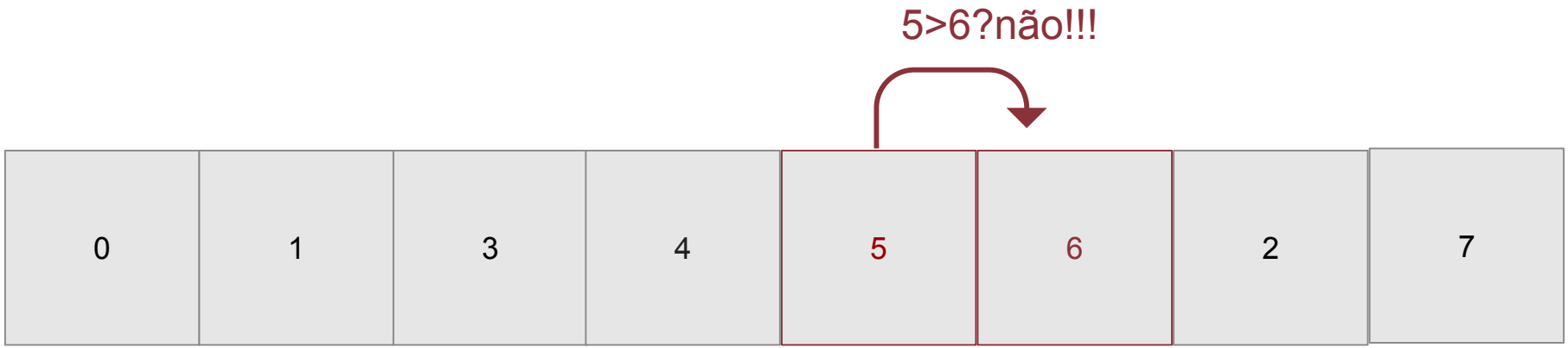
Comparações: 12
Trocas: 6



Demonstração - Shell Sort

Salto: 1

Comparações: 12
Trocas: 6

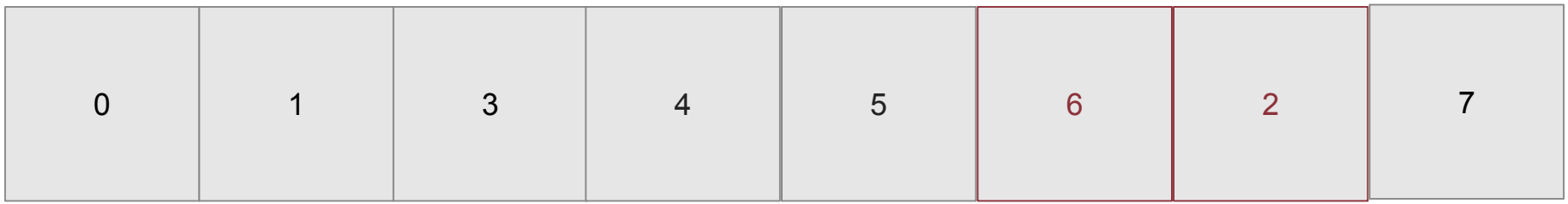


Demonstração - Shell Sort

Salto: 1

Comparações: 13
Trocas: 6

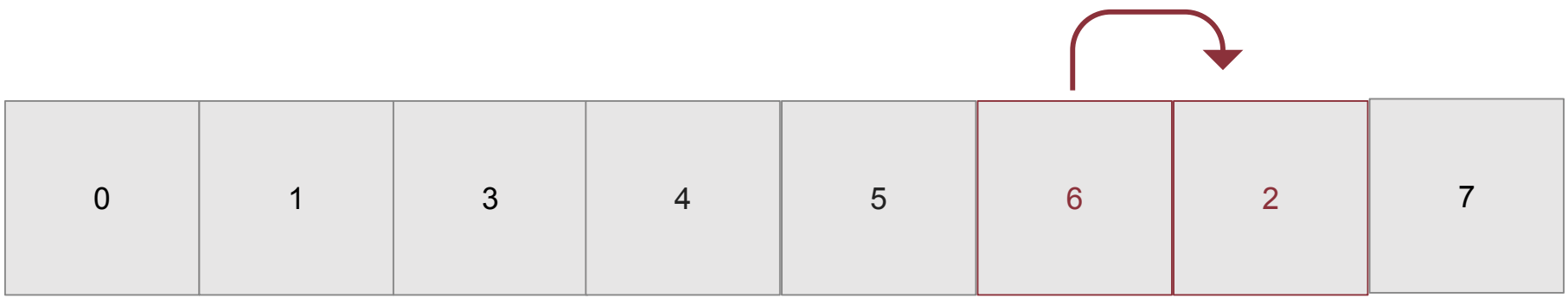
6>2?



Demonstração - Shell Sort

Salto: 1

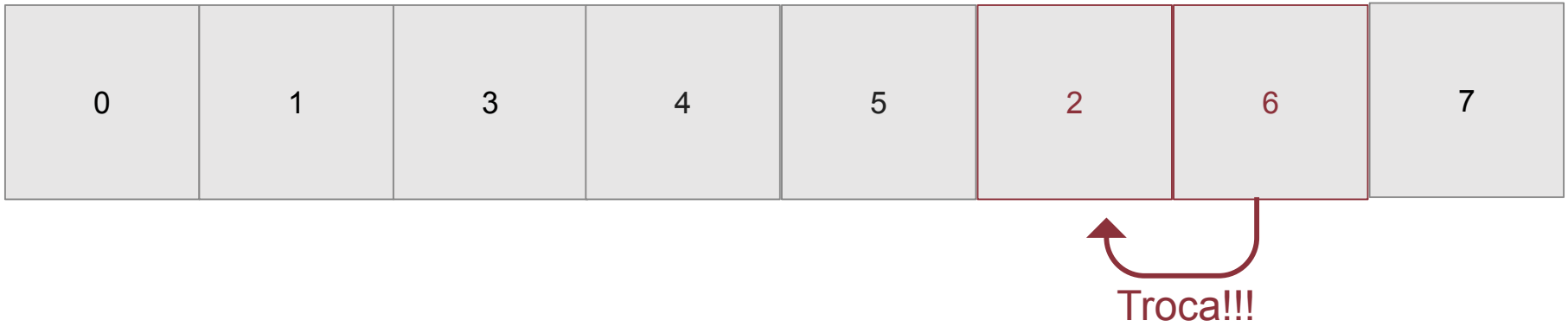
Comparações: 13
Trocas: 6
6>2?sim!!!



Demonstração - Shell Sort

Salto: 1

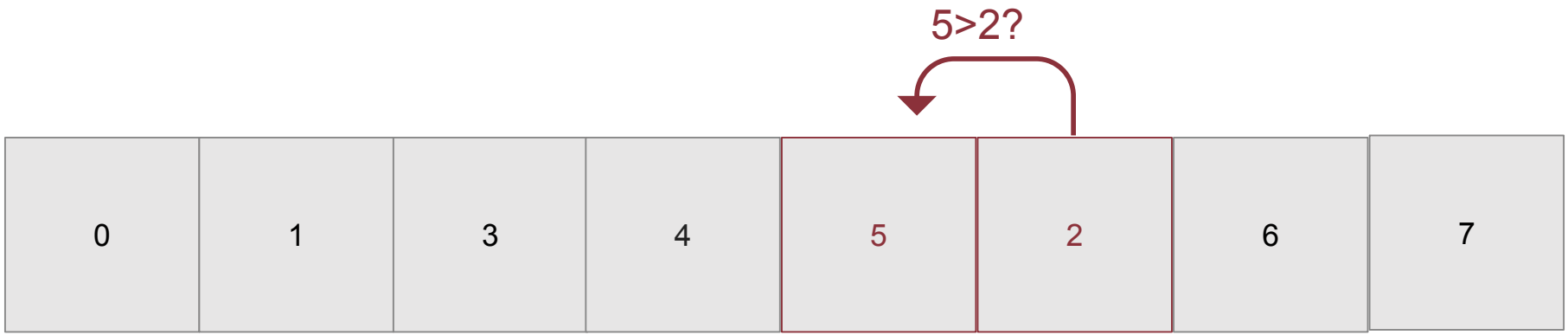
Comparações: 13
Trocas: 7



Demonstração - Shell Sort

Salto: 1

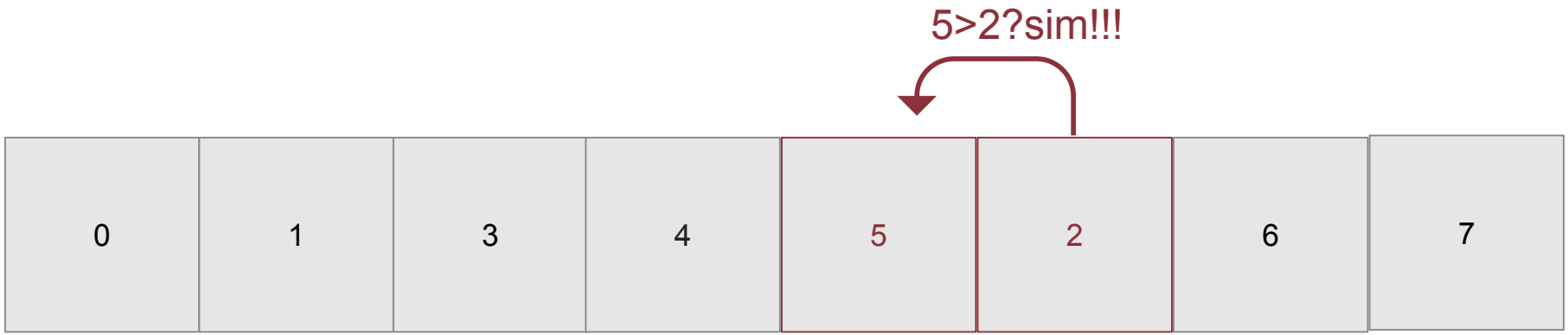
Comparações: 14
Trocas: 7



Demonstração - Shell Sort

Salto: 1

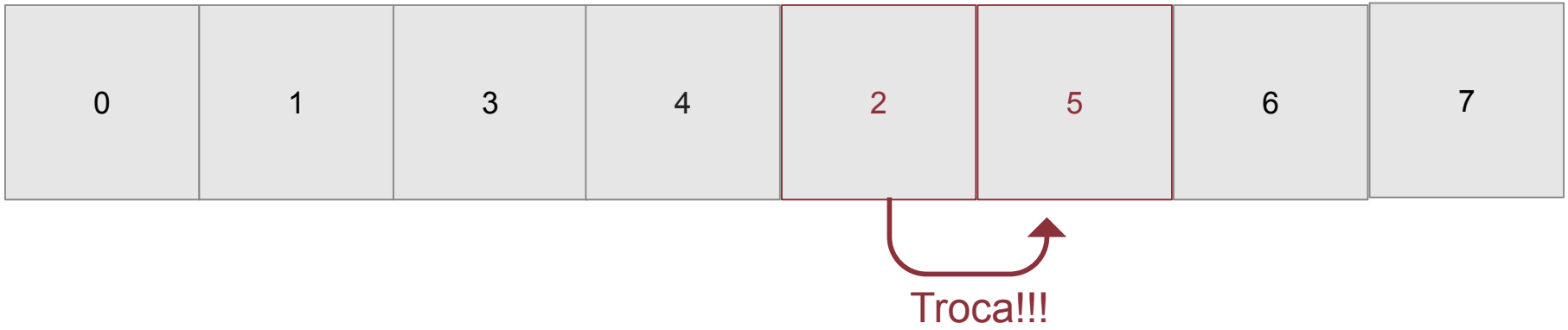
Comparações: 14
Trocas: 7



Demonstração - Shell Sort

Salto: 1

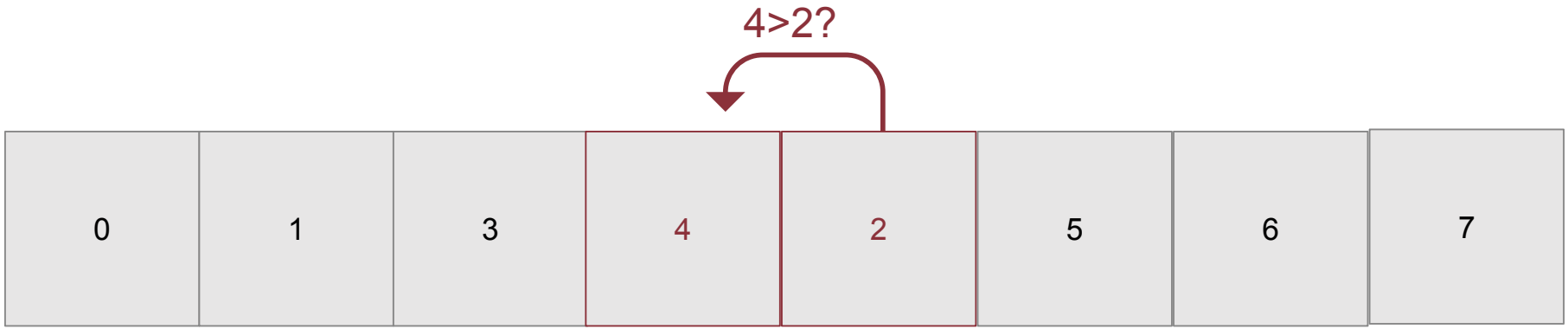
Comparações: 14
Trocas: 8



Demonstração - Shell Sort

Salto: 1

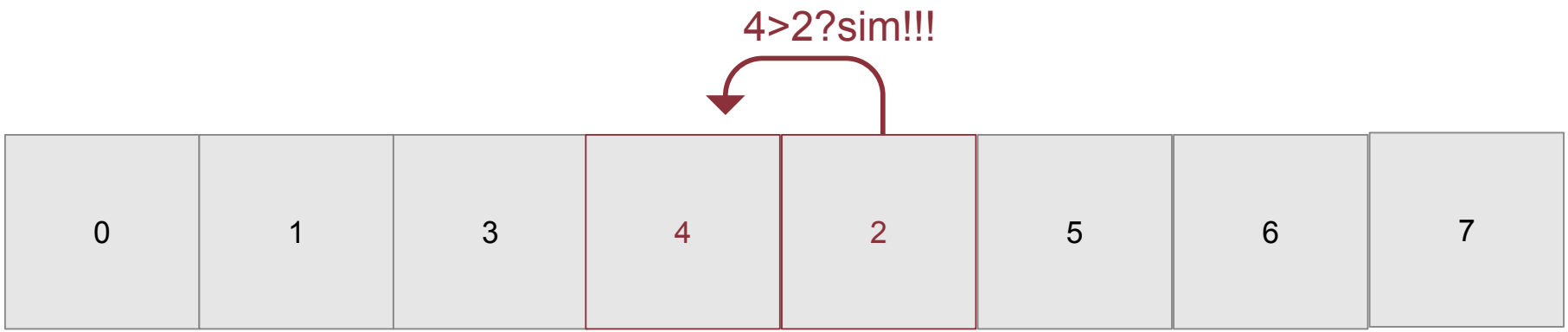
Comparações: 15
Trocas: 8



Demonstração - Shell Sort

Salto: 1

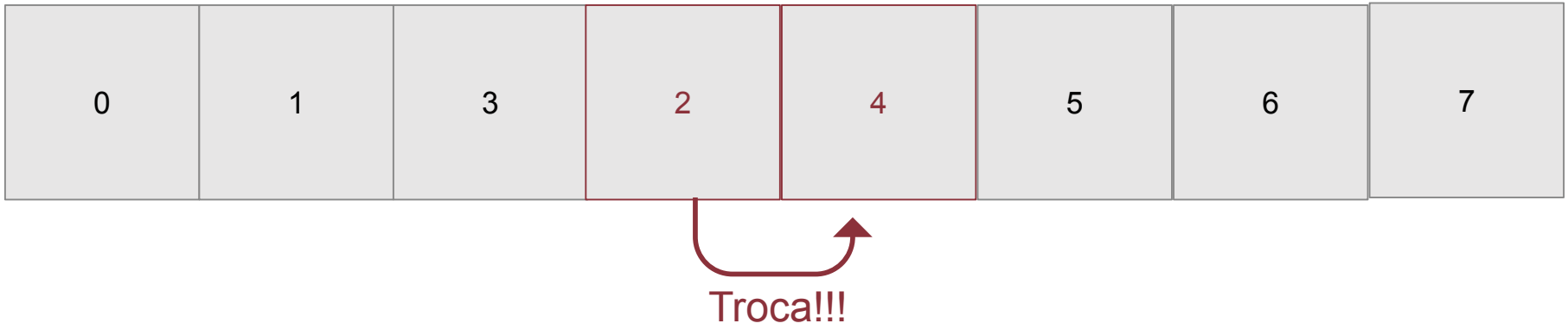
Comparações: 15
Trocas: 8



Demonstração - Shell Sort

Salto: 1

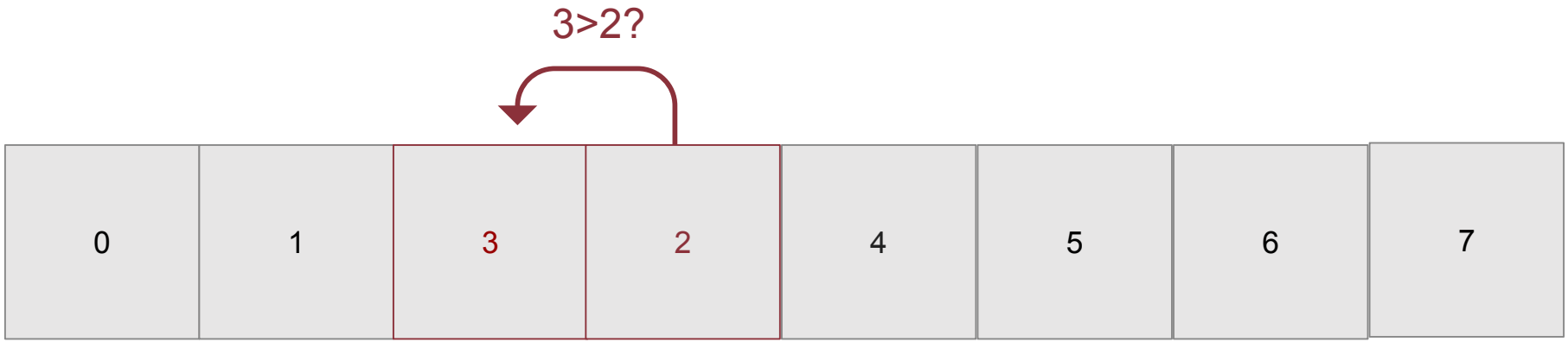
Comparações: 15
Trocas: 9



Demonstração - Shell Sort

Salto: 1

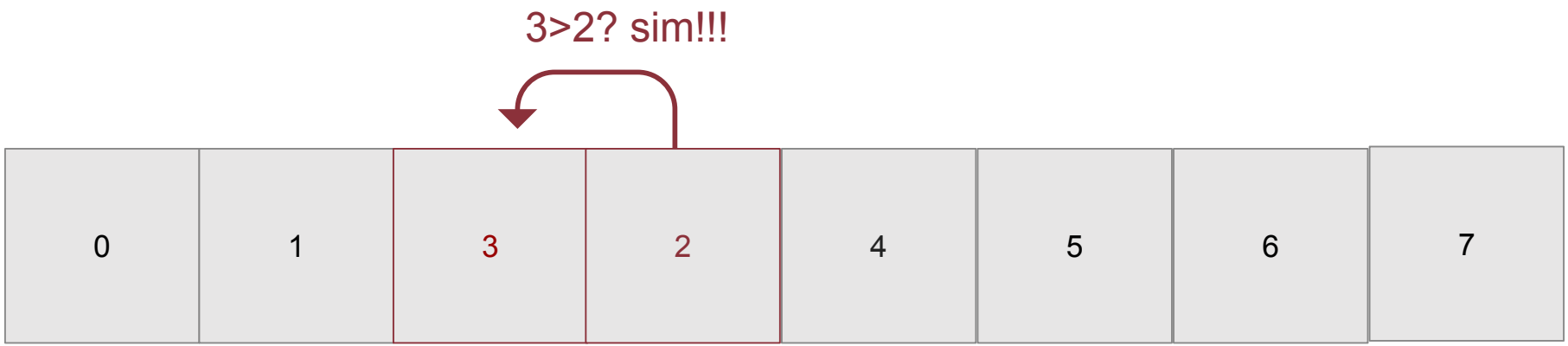
Comparações: 16
Trocas: 9



Demonstração - Shell Sort

Salto: 1

Comparações: 16
Trocas: 9



Demonstração - Shell Sort

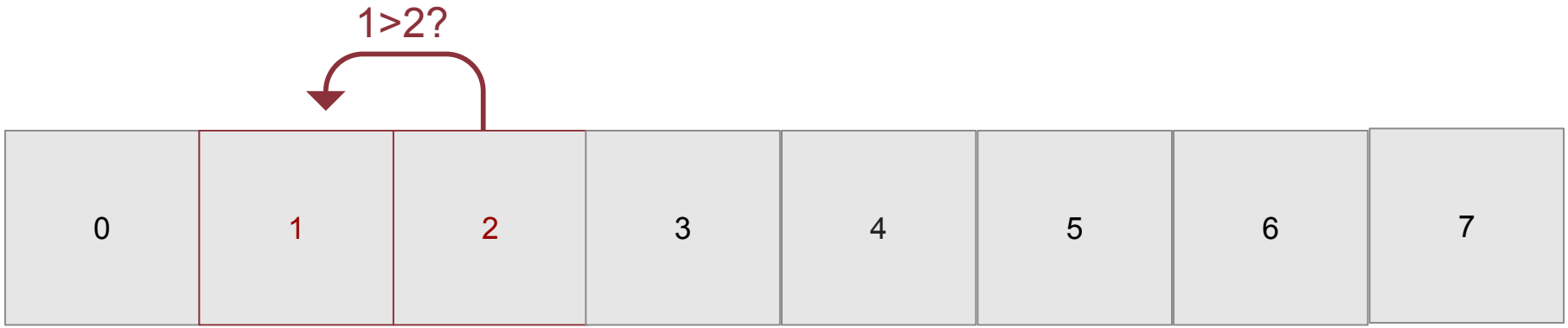
Salto: 1

Comparações: 16
Trocas: 10

Demonstração - Shell Sort

Salto: 1

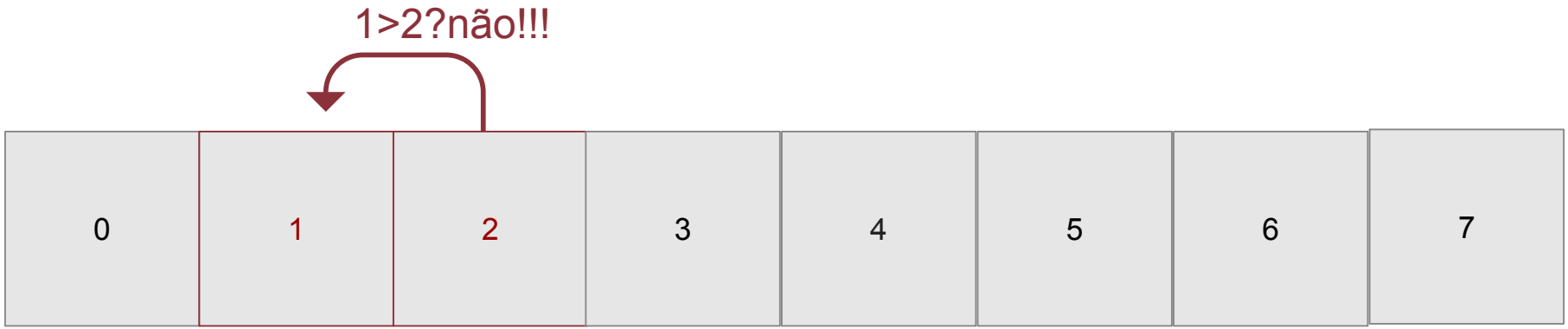
Comparações: 17
Trocas: 10



Demonstração - Shell Sort

Salto: 1

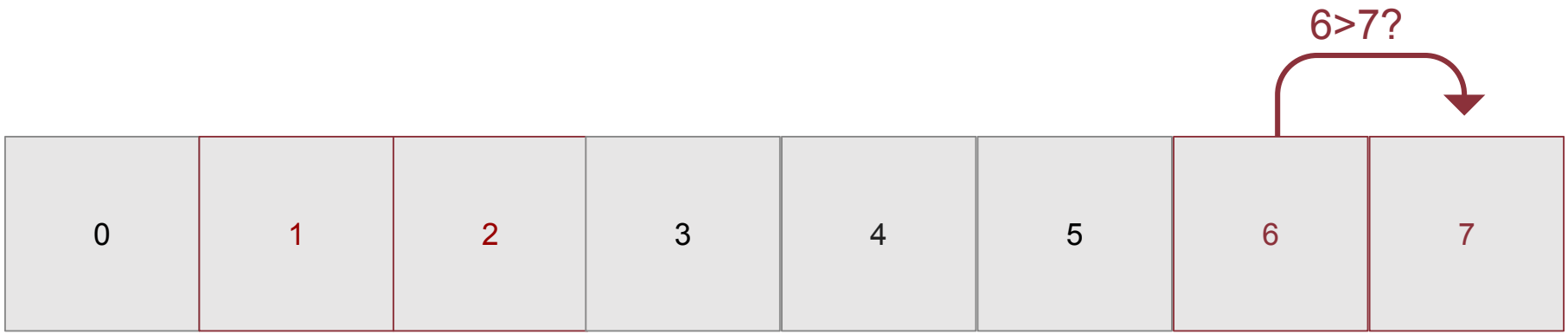
Comparações: 17
Trocas: 10



Demonstração - Shell Sort

Salto: 1

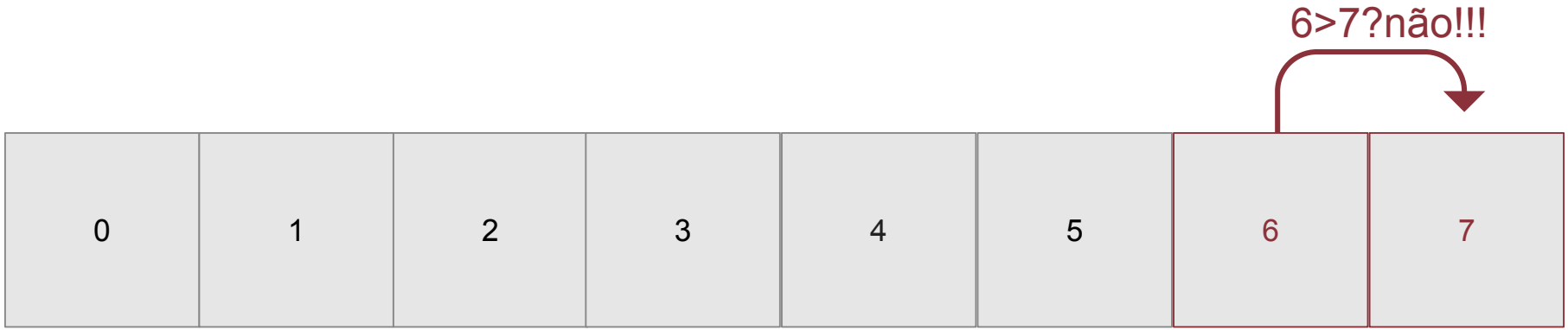
Comparações: 18
Trocas: 10



Demonstração - Shell Sort

Salto: 1

Comparações: 18
Trocas: 10



Demonstração - Shell Sort

Comparações: 18
Trocas: 10

VETOR ORDENADO!!

0	1	2	3	4	5	6	7

Referencial Teórico - Cycle Sort

- Algoritmo baseado em ciclos de permutação;
- Desenvolvido com foco na minimização de escritas em memória;
- Diferencia-se dos algoritmos clássicos de comparação;
- Indicado para cenários onde o custo de escrita é elevado.

Referencial Teórico - Cycle Sort

- Princípio de funcionamento:
  - Cada elemento é colocado diretamente em sua posição final correta;
  - A ordenação ocorre por meio de ciclos de movimentação;
  - Evita trocas desnecessárias;
  - Ao final de cada ciclo, todos os elementos envolvidos estão corretamente posicionados.

Referencial Teórico - Cycle Sort

- Complexidade Assintótica:
  - Complexidade de tempo fixa: $O(n^2)$ em todos os casos;
  - Complexidade de escrita ótima: realiza no máximo n-1 trocas.

Referencial Teórico - Cycle Sort

- Características Operacionais:
  - Algoritmo *in-place*;
  - Não estável;
  - Altamente sensível à presença de elementos duplicados.

Demonstração - Cycle Sort

- **Vetor desordenado:**

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	0	5	2	3

Demonstração - Cycle Sort

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 4

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	0	5	2	3

Demonstração - Cycle Sort

Comparações: 7
Trocas: 0

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 4

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	0	5	2	3

  - Total de elementos menores que **quatro**: 4
  - **pos = 4** (total de elementos menores)

Demonstração - Cycle Sort

Comparações: 7
Trocas: 0

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 4

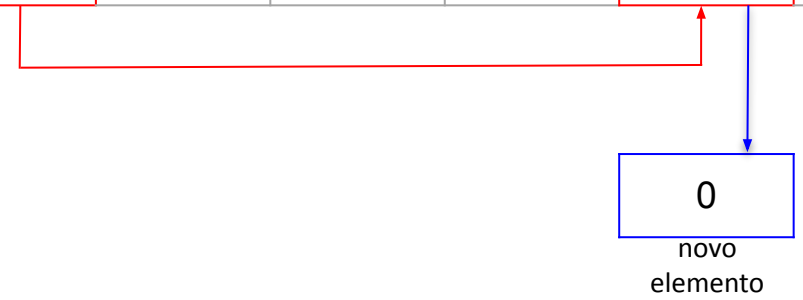
Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	0	5	2	3

Demonstração - Cycle Sort

Comparações: 7
Trocas: 1

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 4

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	4	5	2	3

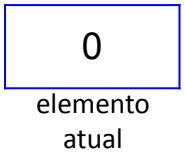


Demonstração - Cycle Sort

Comparações: 7
Trocas: 1

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 0

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	4	5	2	3



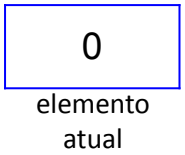
Demonstração - Cycle Sort

Comparações: 14
Trocas: 1

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 0; pos = 0

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	4	5	2	3

  - Total de elementos menores que **zero**: 0

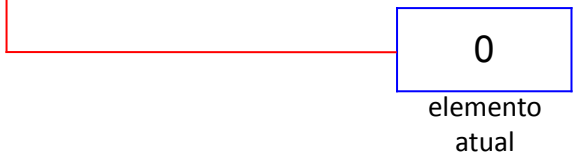


Demonstração - Cycle Sort

Comparações: 14
Trocas: 1

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 0; pos = 0

Índice	0	1	2	3	4	5	6	7
Elemento	4	6	1	7	4	5	2	3



Demonstração - Cycle Sort

Comparações: 14
Trocas: 2

- Ciclo 1:
  - ciclo_inicio = 0; elemento atual = 0; pos = 0

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	2	3

- pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

Comparações: 14
Trocas: 2

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	2	3

Demonstração - Cycle Sort

Comparações: 20
Trocas: 2

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 6; pos = 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	2	3

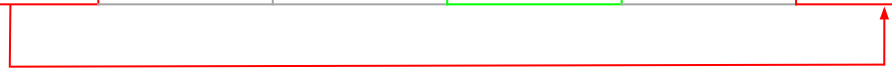
  - Total de elementos menores que **seis**: 6

Demonstração - Cycle Sort

Comparações: 20
Trocas: 2

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 6; pos = 6.

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	2	3

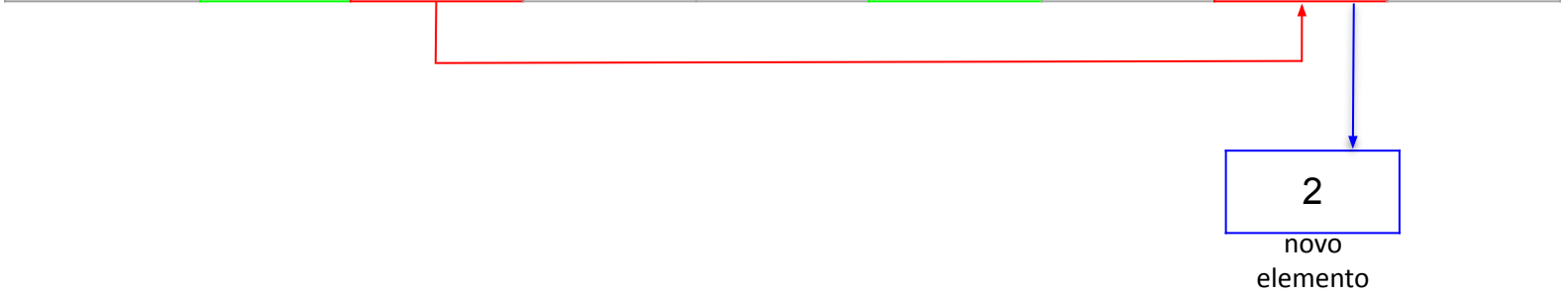


Demonstração - Cycle Sort

Comparações: 20
Trocas: 3

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 6; pos = 6.

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	6	3

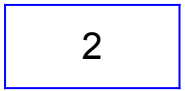


Demonstração - Cycle Sort

Comparações: 20
Trocas: 3

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 2

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	6	3



elemento atual

Demonstração - Cycle Sort

Comparações: 26
Trocas: 3

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 2; pos = 2.

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	6	3

- Total de elementos menores que **dois**: 2

2

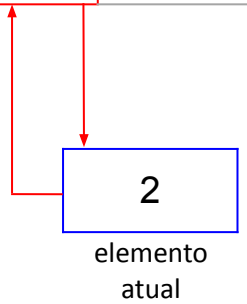
elemento atual

Demonstração - Cycle Sort

Comparações: 26
Trocas: 3

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 2; pos = 2.

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	1	7	4	5	6	3



Demonstração - Cycle Sort

Comparações: 26
Trocas: 4

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 1

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	2	7	4	5	6	3

1

elemento atual

Demonstração - Cycle Sort

Comparações: 32
Trocas: 4

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 1; pos = 1

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	2	7	4	5	6	3

- Total de elementos menores que **um**: 1

1

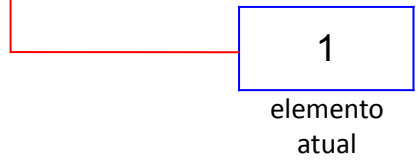
elemento atual

Demonstração - Cycle Sort

Comparações: 32
Trocas: 4

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 1; pos = 1

Índice	0	1	2	3	4	5	6	7
Elemento	0	6	2	7	4	5	6	3



Demonstração - Cycle Sort

Comparações: 32
Trocas: 5

- Ciclo 2:
  - ciclo_inicio = 1; elemento atual = 1; pos = 1

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	3

- pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

- Ciclo 3:
  - ciclo_inicio = 2; elemento atual = 2; pos = 2.

Comparações: 37
Trocas: 5

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	3

  - Total de elementos menores que **dois**: 2
  - pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 7.

Comparações: 37
Trocas: 5

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	3

Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 7; pos = 7.

Comparações: 41
Trocas: 5

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	3

  - Total de elementos menores que **sete:** 7

Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 7; pos = 7.

Comparações: 41
Trocas: 5

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	3

Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 7;

Comparações: 41
Trocas: 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	7

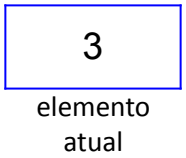
novo elemento

Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 3;

Comparações: 41
Trocas: 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	7



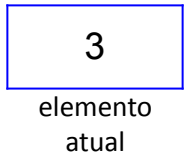
Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 3; pos = 3

Comparações: 45
Trocas: 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	7

- Total de elementos menores que **três:** 3

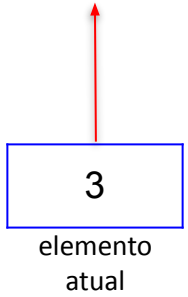


Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 3; pos = 3

Comparações: 45
Trocas: 6

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	7	4	5	6	7



Demonstração - Cycle Sort

- Ciclo 4:
  - ciclo_inicio = 3; elemento atual = 3; pos = 3

Comparações: 45
Trocas: 7

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

  - pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

- Ciclo 5:
  - ciclo_inicio = 4; elemento atual = 4; pos = 4

Comparações: 48
Trocas: 7

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

- Total de elementos menores que **quatro**: 4
- pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

- Ciclo 6:
  - ciclo_inicio = 5; elemento atual = 5.

Comparações: 48
Trocas: 7

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

Demonstração - Cycle Sort

- Ciclo 6:
  - ciclo_inicio = 5; elemento atual = 5; pos = 5.

Comparações: 50
Trocas: 7

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

- Total de elementos menores que **cinco**: 5
- pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

- Ciclo 7:
  - ciclo_inicio = 6; elemento atual = 6; pos = 6.

Comparações: 51
Trocas: 7

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

- Total de elementos menores que **seis**: 6
- pos == ciclo_inicio, logo **ciclo encerrado.**

Demonstração - Cycle Sort

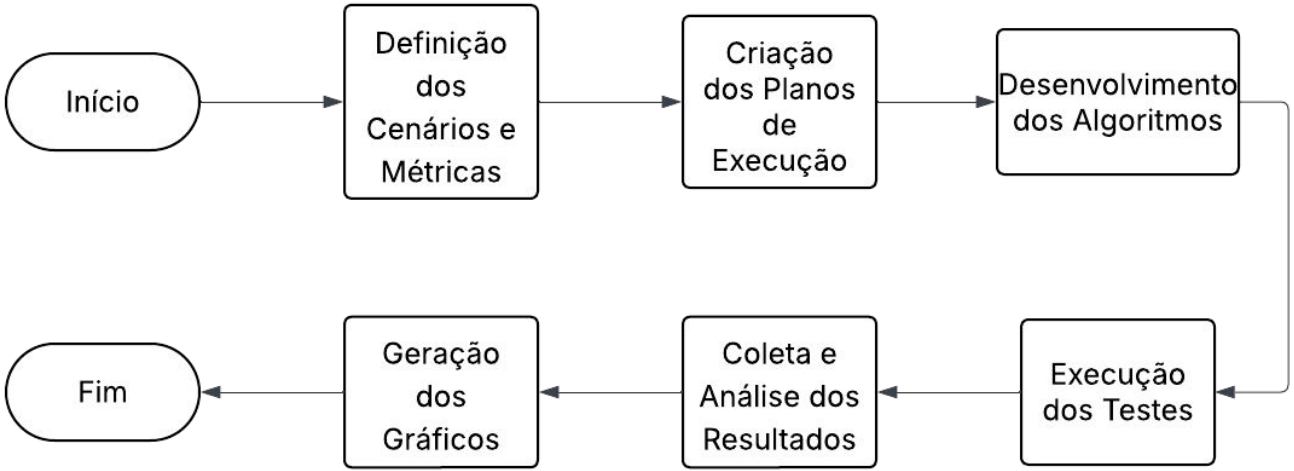
Comparações: 51
Trocas: 7

- **Vetor ordenado:**

Índice	0	1	2	3	4	5	6	7
Elemento	0	1	2	3	4	5	6	7

Metodologia

Figura 1: Fluxograma Metodológico.



Fonte: Autor Próprio, 2026.

Metodologia

Tabela 1: Ambiente de Execução.

Processador:	Intel Core i5-12450H 12º Gen (2.00 GHz) (8 núcleos, 12 threads, 12 MB cache)
RAM:	16,0 GB @ 3200 MHz (utilizável: 15,7 GB)
SO:	Windows 11 Home Single Language (Executado no Ubuntu 24.04.3 LTS via WSL2)
Linguagem:	C

Fonte: Autor Próprio, 2026.

Metodologia

Tabela 2: Critérios de Avaliação e Justificativa.

Critério de Avaliação	Justificativa Para a Escolha
Tempo Médio de Execução (ms)	Avaliação do desempenho temporal dos algoritmos.
Número de Comparações	Reflete o custo lógico do processo de ordenação.
Quantidade de Trocas	Indica o volume de movimentações realizadas na memória.
Complexidade Assintótica	Permite analisar empiricamente a taxa de crescimento do custo computacional.
Desvio Padrão	Avalia a estabilidade do desempenho dos algoritmos.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

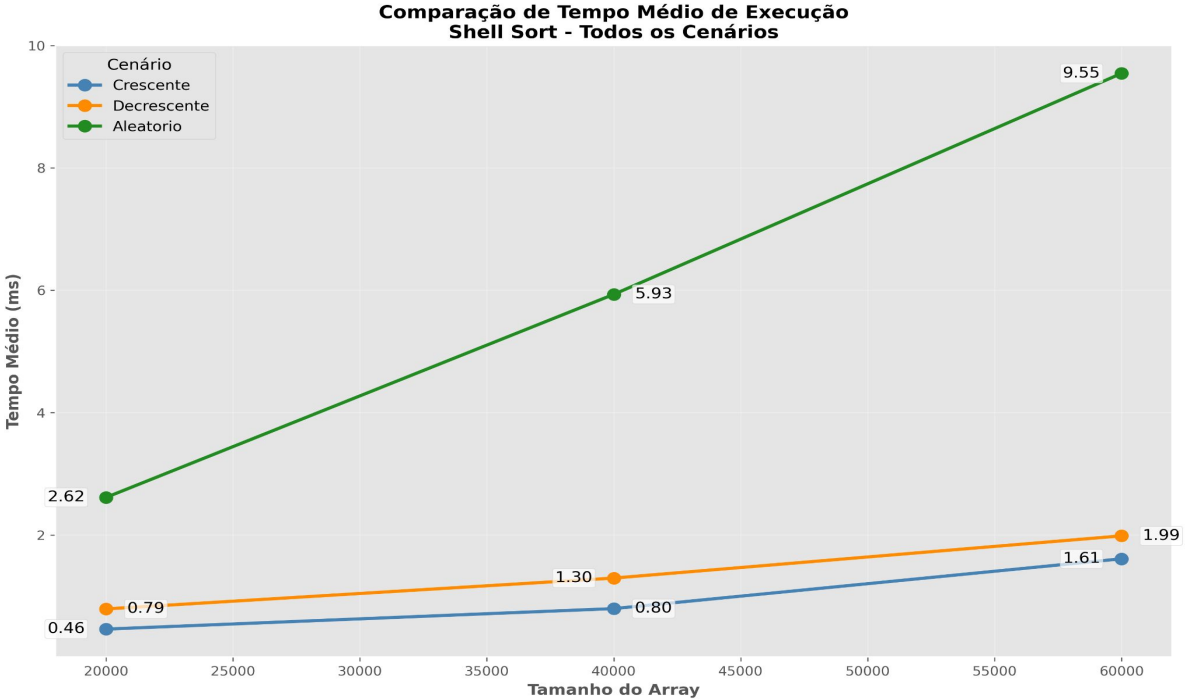


Gráfico 1: Tempo Médio de Execução - Shell Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

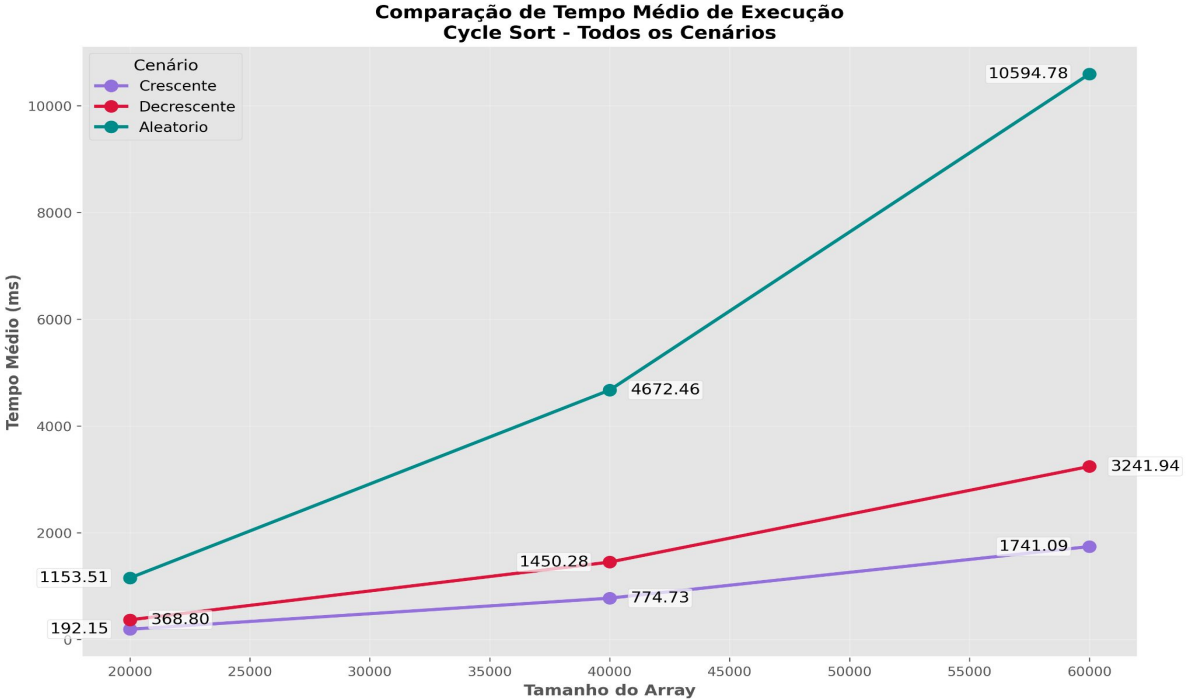


Gráfico 2: Tempo Médio de Execução - Cycle Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

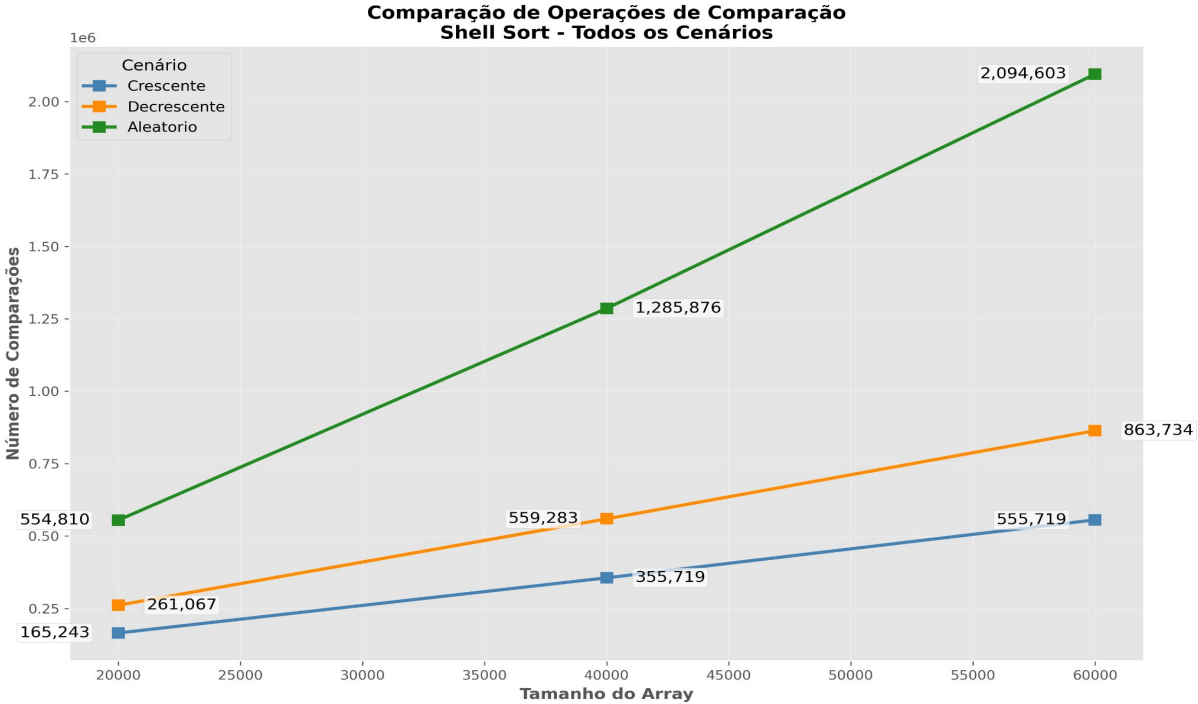


Gráfico 3: Quantidade de Operações de Comparação - Shell Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

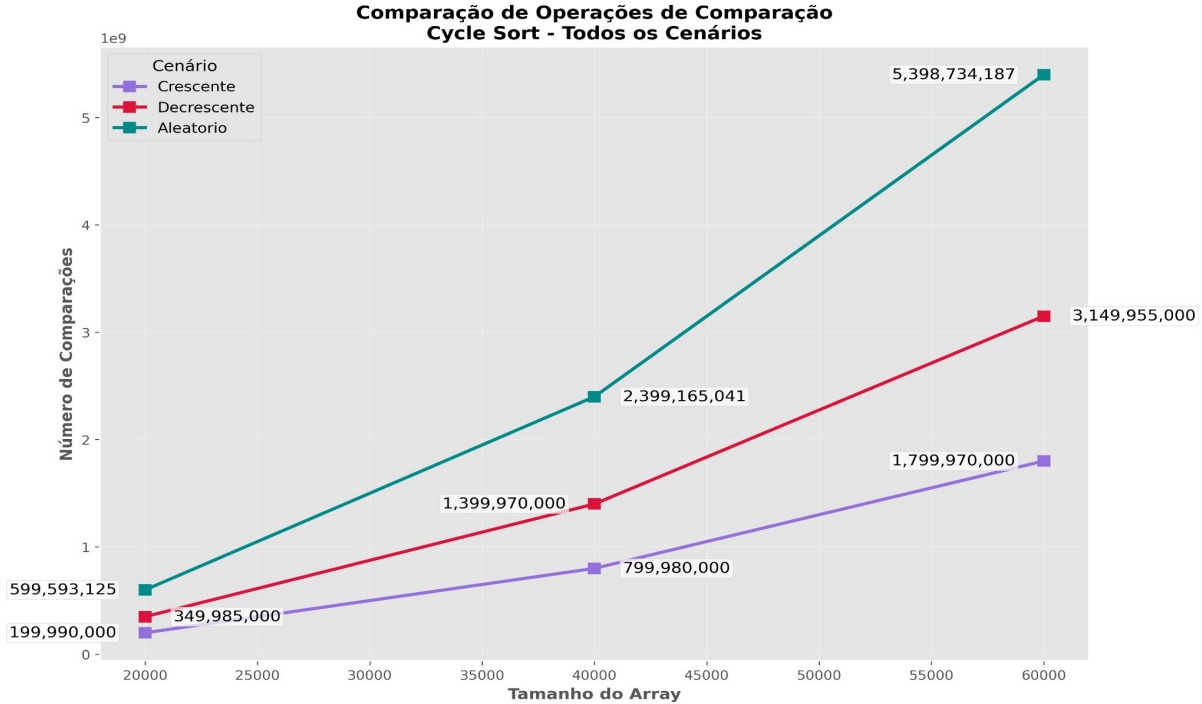


Gráfico 4: Quantidade de Operações de Comparação - Cycle Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

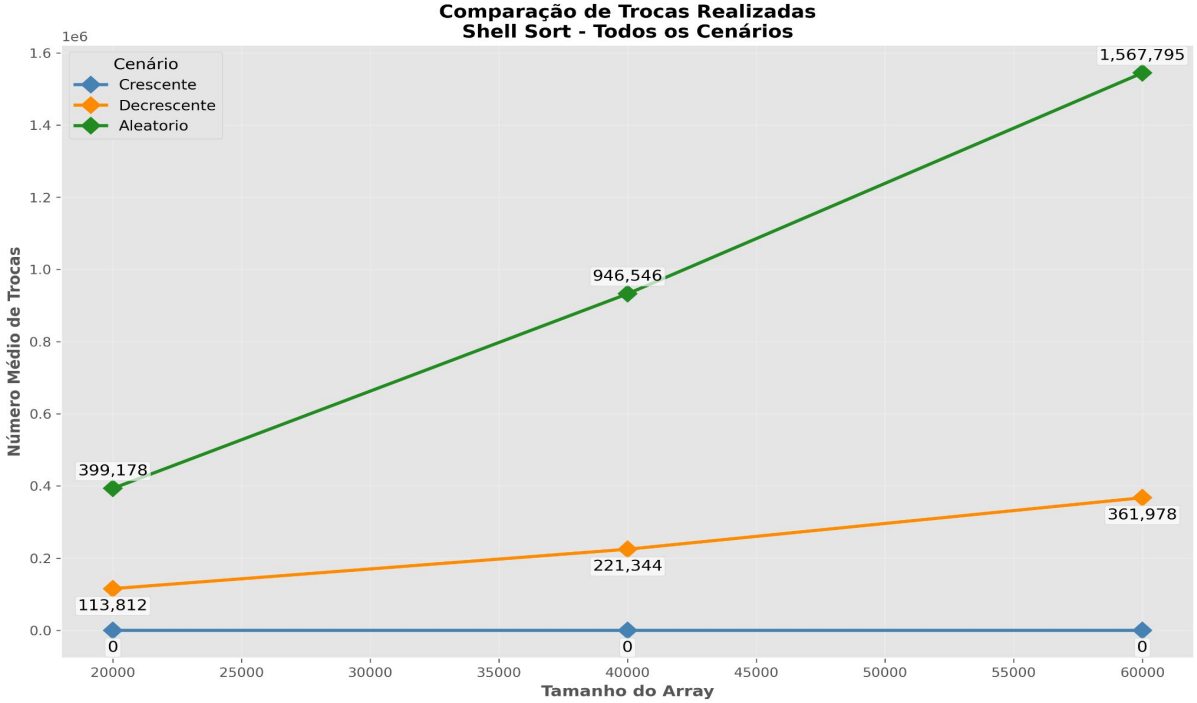


Gráfico 5: Quantidade de Operações de Troca - Shell Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

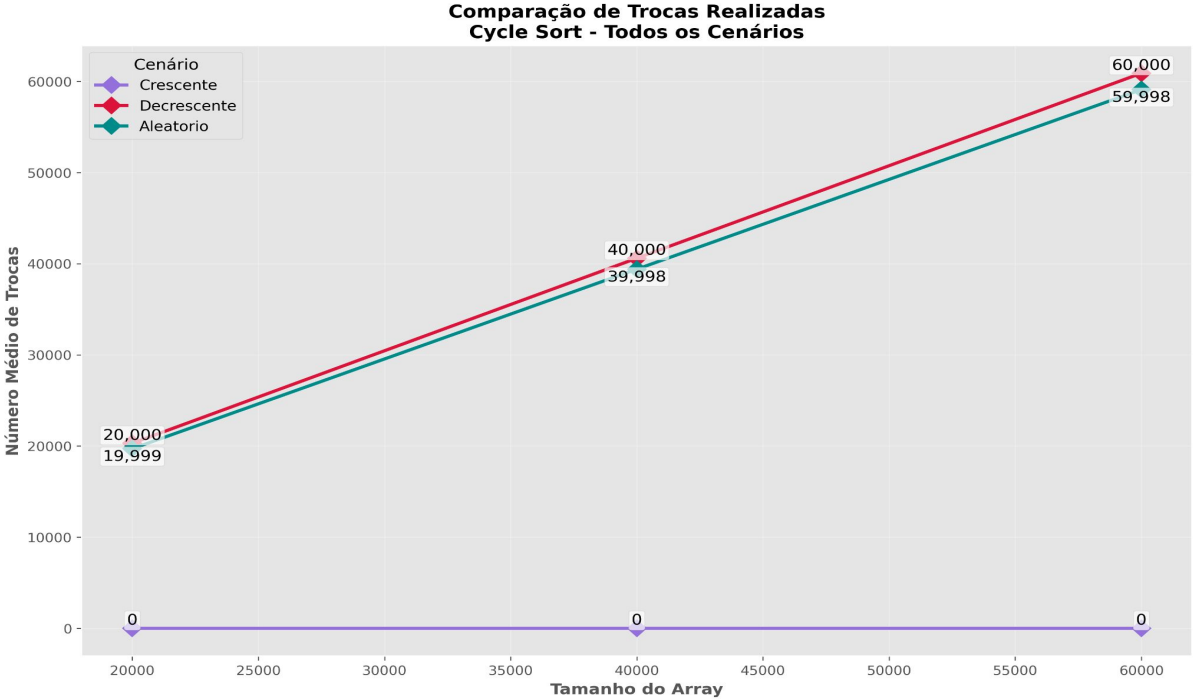


Gráfico 6: Quantidade de Operações de Troca - Cycle Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

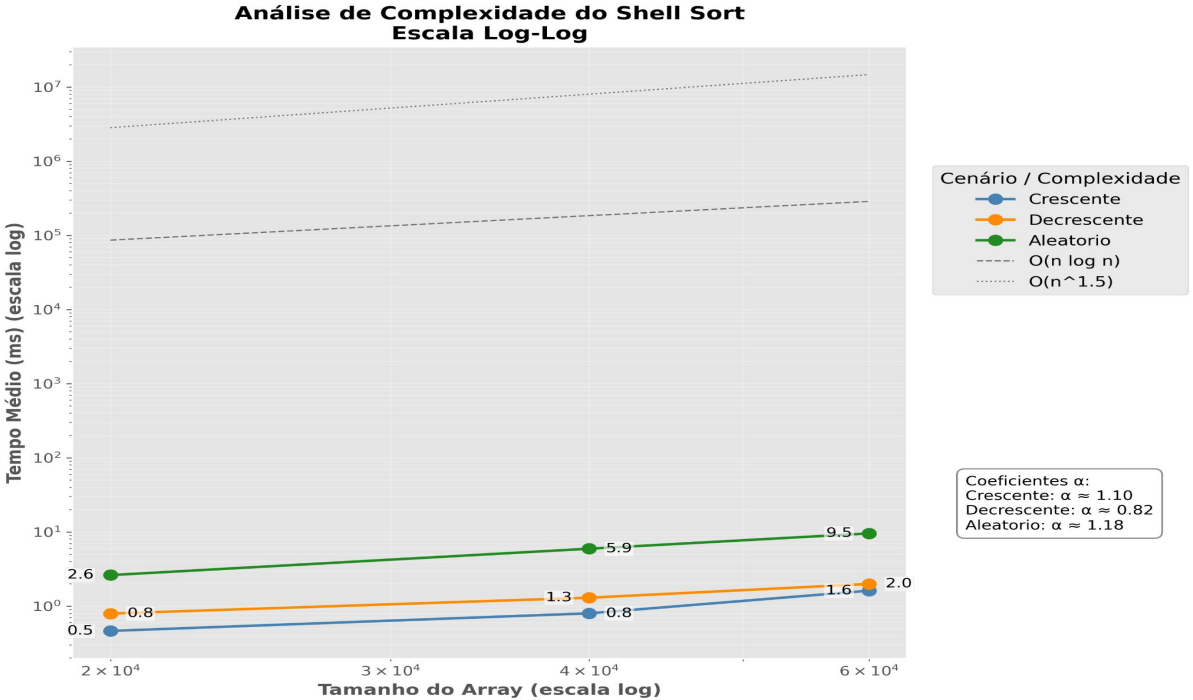


Gráfico 7: Análise de Complexidade - Shell Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

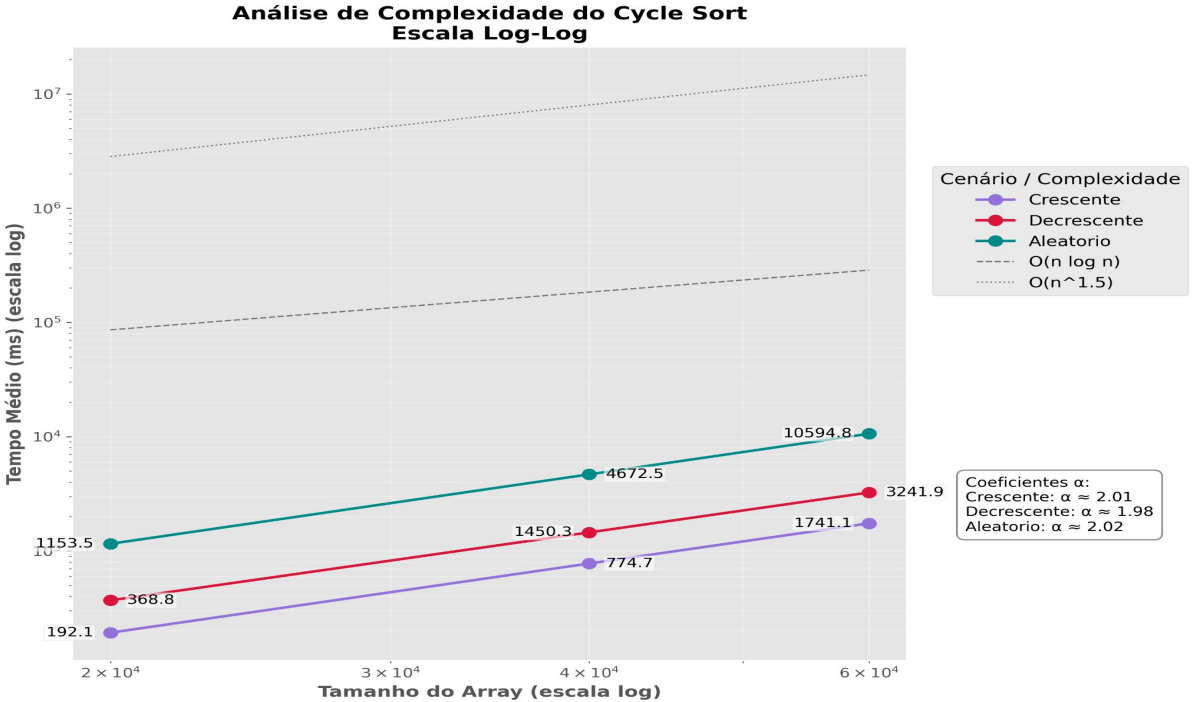


Gráfico 8: Análise de Complexidade - Cycle Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

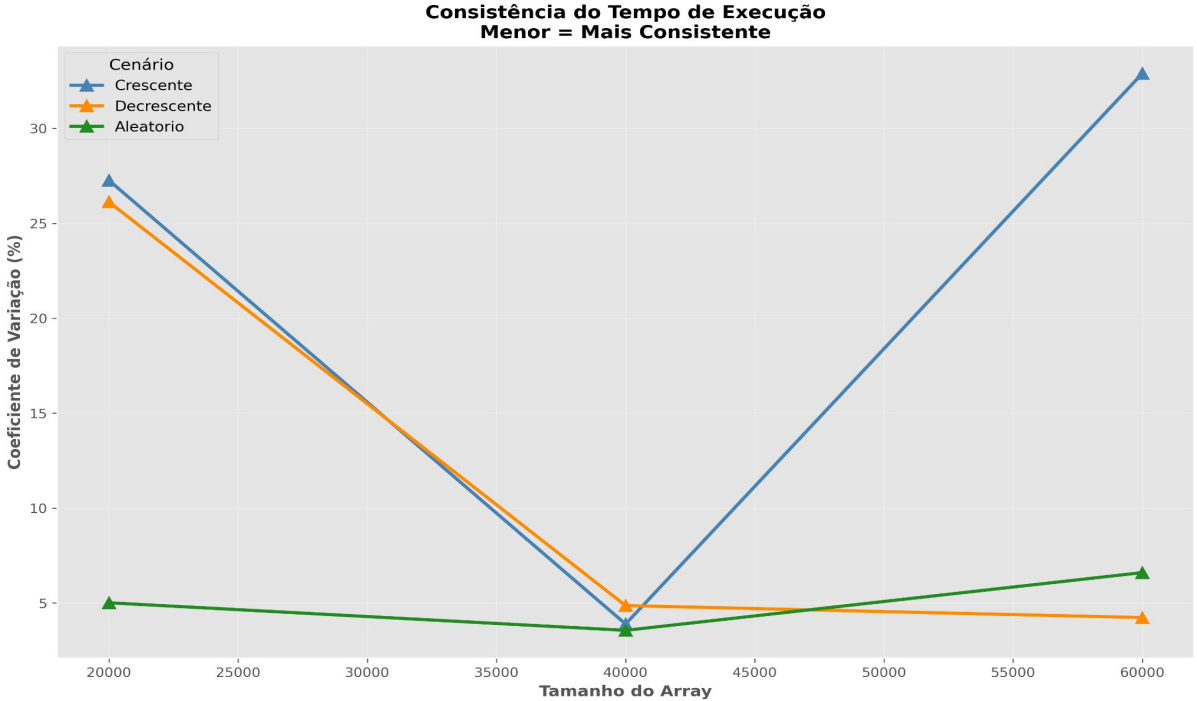


Gráfico 9: Consistência do Tempo de Execução - Shell Sort.

Fonte: Autor Próprio, 2026.

Resultados e Discussão

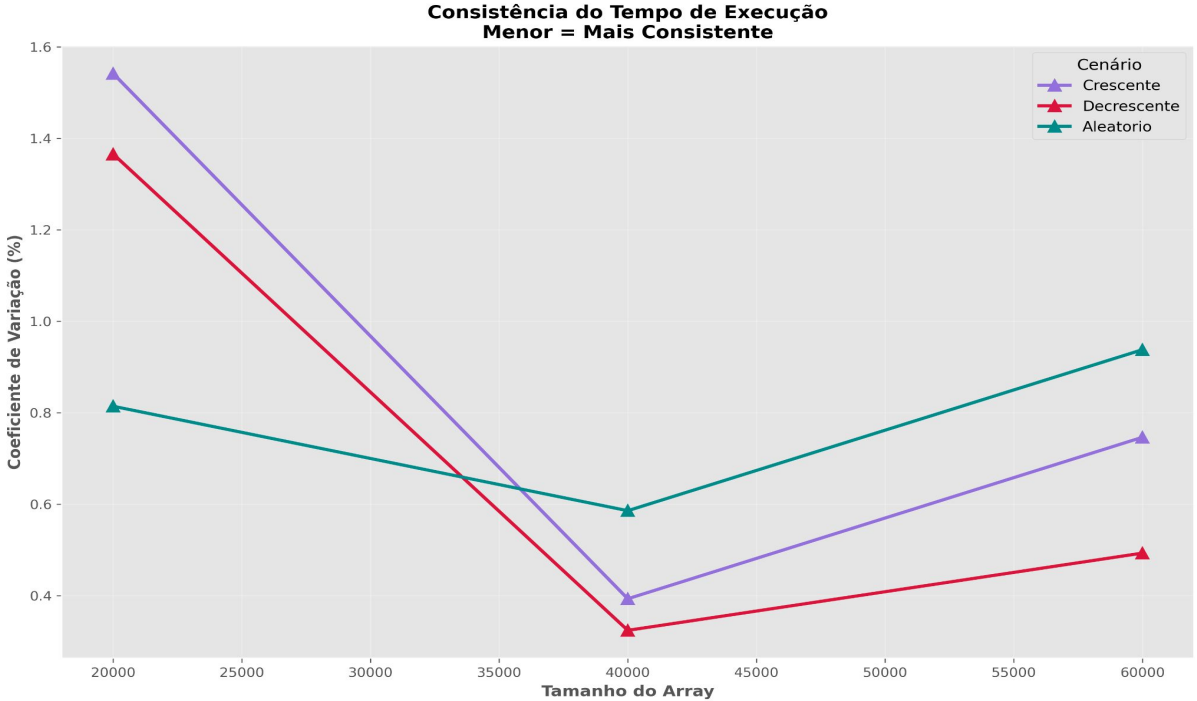


Gráfico 10: Consistência do Tempo de Execução - Cycle Sort.

Fonte: Autor Próprio, 2026.

Conclusão

- *Shell Sort* é eficiente, mas sensível à ordem inicial;
- *Cycle Sort* minimiza trocas, mas tem alto custo em comparações;
- Não existe o algoritmo melhor em todos os casos, mas sim o mais adequado a cada caso.

Referências

Ayazuddin, R. (2025). **A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics**.

Bentley, J. L. and McIlroy, M. D. (1993). **Engineering a sort function. Software: Practice and Experience**, 23(11):1249–1265.

Donald, E. et al. (1997). **The art of computer programming, volume 3: Sorting and searching**.-2nd.

Frank, R. and Lazarus, R. (1960). **A high-speed sorting procedure.** Communications of the ACM, 3(1):20–22.

Gonnet, G. H. and Baeza-Yates, R. (1991). **Handbook of algorithms and data structures: in Pascal and C**. Addison-Wesley Longman Publishing Co., Inc.

Referências

Haddon, B. K. (1990). **Cycle-sort: a linear sorting method**. The Computer Journal, 33(4):365–367.

Hibbard, T. N. (1962). **Some combinatorial properties of certain trees with applications to searching and sorting**. Journal of the ACM (JACM), 9(1):13–28.

Hopcroft, J. E., Ullman, J. D., and Aho, A. V. (1983). **Data structures and algorithms, volume 175**. Addison-wesley Boston, MA, USA:.

Jain, R. (1990). **The art of computer systems performance analysis**. john wiley & sons.

Janson, S. and Knuth, D. E. (1997). **Shellsort with three increments. Random Structures & Algorithms**, 10(1-2):125–142.

Referências

Lipton, L. R. and Naughton, J. Knu d. knuth, **the art of computer programming, vol. 3: Sorting and searching**, addison-wesley, reading, ma, 1973. kri p. krishnan, online prediction algorithms for databases and operating systems,”brown univ.

Mahmoud, H. M. (2011). **Sorting: A distribution theory**. John Wiley & Sons.

Muhammad, M. H. G., Malik, J. A., Akhtar, M., Baloch, M. A., and Rajwana, M. A. (2025). **Sort data faster: Comparing algorithms**. Southern Journal of Computer Science, 1(02):28–37.

Pratt, V. R. (1972). **Shellsort and sorting networks**. Technical report.

Sedgewick, R. (1986). **A new upper bound for shellsort**. Journal of Algorithms, 7(2):159–173.

Referências

Sedgewick, R. and Wayne, K. (2011). **Algorithms**. Addison-wesley professional.

Shell, D. L. (1959). **A high-speed sorting procedure**. Communications of the ACM,

2(7):30–32.

Smythe, R. T. and Wellner, J. (2001). **Stochastic analysis of shell sort. Algorithmica**,

31(3):442–457.

Smythe, R. T. and Wellner, J. A. (2002). **Asymptotic analysis of (3, 2, 1)-shell sort**.

Random Structures & Algorithms, 21(1):59–75.

Souza, R. M., Oliveira, F. S., and Pinto, P. E. (2018). **Um limite superior para a complexidade do shellsort**. In Encontro de Teoria da Computacão (ETC). SBC.

Referências

Sundaramoorthy, S. and Karunanidhi, G. (2025). **A systematic analysis on performance and computational complexity of sorting algorithms**. Discover Computing, 28(1):250.

Thomas H, C., Charles, E., Ronald L, R., Clifford, S., et al. (2009). **Introduction to algorithms third edition**.

Tokuda, N. (1992). **An improved shellsort**. In Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture-Information Processing'92, Volume 1-Volume I, pages 449–457.

Turzo, N. A., Sarker, P., Kumar, B., Ghose, J., and Chakraborty, A. (2020). **Defining a modified cycle sort algorithm and parallel critique with other sorting algorithms**.

Global Research Development (GRD) ISSN: 2455-5703, 5(4):1–8.