Análise Comparativa de Desempenho de Algoritmos de Ordenação

Guilherme Eduardo Gonçalves da Silva
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
guilherme.eduardo1@ufpr.br

***Resumo*—O trabalho consiste em uma análise de desempenho de algoritmos de ordenação. Os resultados destacam a eficiência do Merge Sort e do QuickSort em relação aos demais algoritmos, tornando-os escolhas eficazes para aplicações que envolvem ordenação em grandes conjuntos de dados.**

***Index Terms*—algoritmos, desempenho, ordenação.**

I. Introdução

Este relatório aborda a análise de desempenho entre algoritmos de ordenação, com foco na complexidade ($Big\ \mathcal{O}$) relacionada ao número de comparações e ao tempo de execução em um vetor de inteiros. Os algoritmos estudados incluem HeapSort, Merge Sort, QuickSort, Counting Sort e Shell Sort. As implementações foram feitas em linguagem C, utilizando o método de recursão e iteração. O objetivo deste estudo é quantificar e comparar o desempenho desses algoritmos, visando compreender suas eficiências.

II. Algoritmos de ordenação extra - Shell Sort

O Shell Sort propõe superar a eficiência do Insertion Sort ao buscar uma complexidade inferior a $\mathcal{O}(n^2)$. O algoritmo opera dividindo o vetor não ordenado em subgrupos menores, os quais são ordenados por um algoritmo de inserção. Ao reduzir gradualmente o tamanho dos subgrupos, o vetor como um todo é ordenado. A inovação reside na realização de trocas entre elementos distantes através de um intervalo (*gap*), em contraste com trocas entre elementos adjacentes, agilizando o reposicionamento para as posições corretas.

III. Algoritmo QuickSort

O algoritmo QuickSort foi implementado com duas variantes para a função interna 'particionar', a qual é responsável por selecionar o pivô para a divisão do vetor. O primeiro modelo consiste em escolher o último elemento do vetor como pivô, e o segundo segue uma estratégia recomendada por [1], escolhendo o pivô como $x = \text{vetor}[i]$, onde $n/3 \leq i \leq 2n/3$. Essa abordagem contribuiu para a melhoria da complexidade do algoritmo.

IV. Metodologia

A. Ambiente de Desenvolvimento

Toda a programação e testes foram realizados em um computador com as seguintes configurações:

- Sistema Operacional: Ubuntu 22.04.3 LTS
- Processador: Intel Core i5-9400F 2,9 GHz.
- Kernel: 6.2.0-33-generic x86-64.
- Memória RAM: 2 x 8 Gigabytes-DDR4 Corsair Vengeance LPX - 3200 Mhz.
- Compilador: gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0.

B. Dados de entrada

Para os testes de ordenação, foram realizadas as seguintes implementações: os vetores foram gerados em ordem decrescente (de n elementos até 1) para analisar o pior caso possível e também com valores aleatórios, seguindo o tamanho n do vetor, sendo $\forall i, 1 \leq i \leq n : \text{vetor}[i] \leq n$.

Os tamanhos dos vetores escolhidos foram 1.000.000, 2.500.000, 5.000.000, 7.500.000 e 10.000.000.

V. Resultado

O primeiro teste executado foi entre os algoritmos HeapSort, Merge Sort e QuickSort com o 'particionar' proposto por Cormem. O Merge Sort teve a melhor performance entre os demais, sendo a sua complexidade de $\mathcal{O}(n \log_2 n)$. Por outro lado, o HeapSort, que possui uma complexidade de $n + 2n \log_2 n$, foi o algoritmo que mais realizou comparações entre os elementos dos vetores inicializados de maneira decrescente conforme a figura 1. Os vetores com elementos sorteados aleatoriamente tiverem o resultado bem semelhante a este.

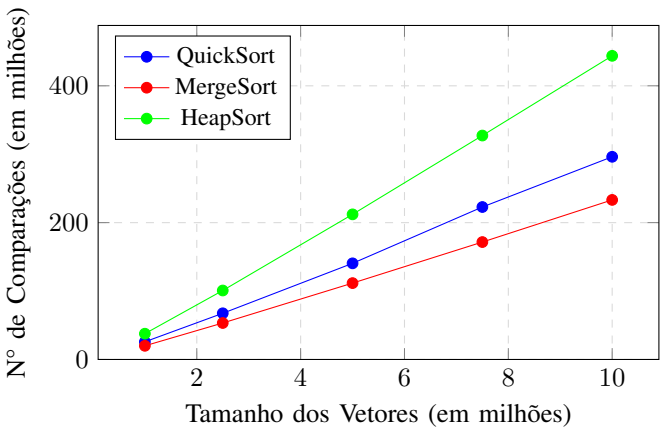

Figura 1. Comparação entre HeapSort, Merge Sort e QuickSort

O teste apresentado na figura 2 trata-se da comparação entre a diferença da função participar convencional e a baseada na explicação do Cormem. Em vetores inicializados de maneira

decrescente, a função aprimorada por Cormem foi superior, tendo a complexidade de $\mathcal{O}(n \log_2 n)$, enquanto a convencional possui a complexidade de $\mathcal{O}(n^2)$.

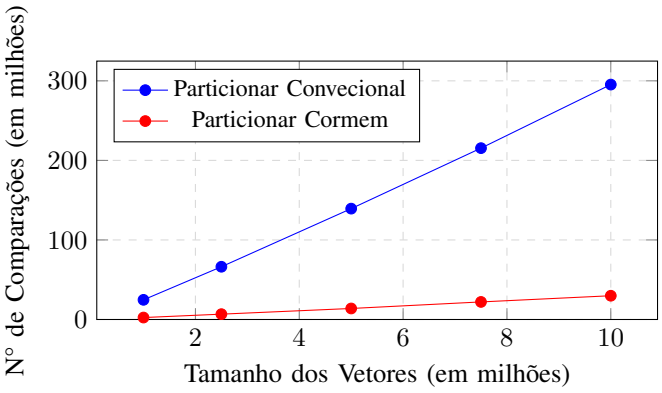

Figura 2. Comparação entre a função particionar do QuickSort

Realizando o mesmo teste, porém com elementos sendo sorteados aleatoriamente, ambas as funções tiverem resultados bem semelhantes conforme apresentado na figura 3.

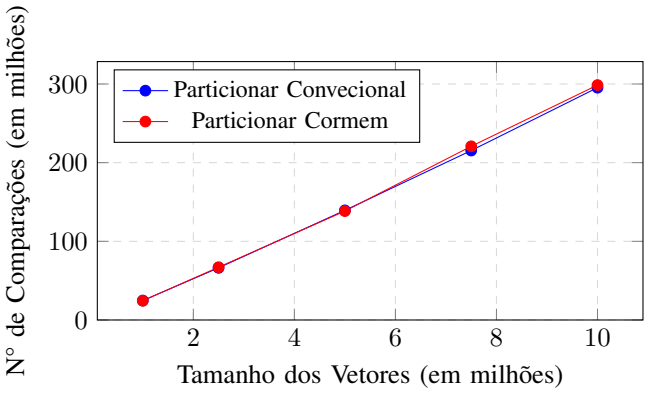

Figura 3. Comparação entre a função particionar com elementos aleatórios no vetor

Tabela I
COUNTING SORT: ANÁLISE DE TEMPO EM RELAÇÃO AO TAMANHO DOS VETORES

Algoritmo	N°Elementos	Tempo (s)
Counting Sort	1,000,000	0.064
	2,500,000	0.307
	5,000,000	0.695
	7,500,000	0.927
	10,000,000	1.503

Além disso, realizamos a inclusão do algoritmo Shell Sort, o qual possui uma eficiência assintótica de tempo $\Theta(n \log n)$. Um caso curioso é que há dificuldades para descobrir o pior caso do algoritmo devido à escolha da sequência (intervalos) e do tamanho do vetor, conforme apresentado por [2]. A sequência/intervalo escolhida para os testes foi a de Donald Knuth, sendo $\frac{3^i-1}{2}$, para $i > 0$. O Shell Sort mostrou realizar menos comparações em relação ao QuickSort em um vetor inicializado de maneira decrescente, conforme mostrado na figura 4.

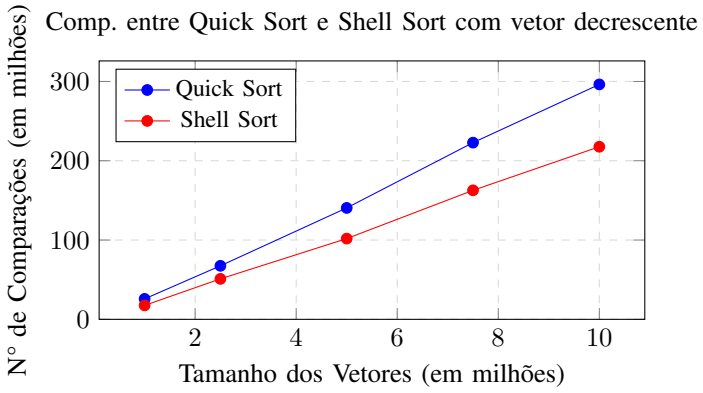

Figura 4. Comparação entre QuickSort e Shell Sort

Inicializando o vetor de maneira aleatória, o QuickSort possui uma performance melhor diante do Shell Sort conforme a figura 5

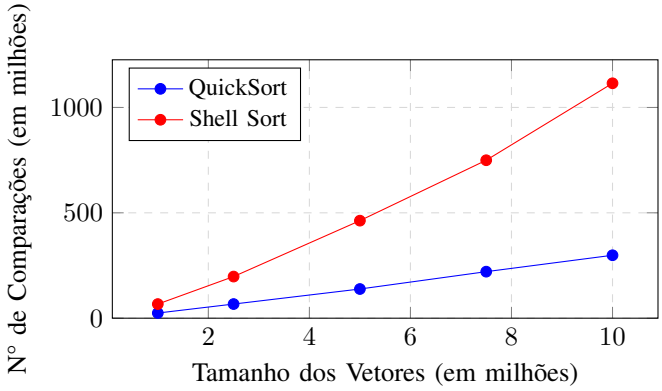

Figura 5. Comparação entre QuickSort e Shell Sort com elementos aleatórios no vetor

A tabela I apresenta o tempo necessário para que o algoritmo Counting Sort realize a ordenação de um vetor inicializado de maneira decrescente conforme o tamanho predefinido e alcança a complexidade de $\mathcal{O}(4n)$ quando o k é igual a n .

VI. CONCLUSÃO

A análise dos resultados dos testes realizados entre os diversos algoritmos de ordenação destacou o desempenho notável do Merge Sort e do QuickSort com particionamento baseado no Cormen. Esses algoritmos demonstraram resultados superiores durante a execução dos testes, consolidando-se como opções mais eficientes.

REFERÊNCIAS

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Algoritmos - Teoria e Prática*, 3rd ed. Rio de Janeiro: Elsevier, 2009.
[2] R. M. de Souza, "Análise de complexidade de pior caso do shellsort por algoritmos," Dissertação de Mestrado, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.