Estudo Sistemático de Algoritmos de Ordenação em Diferentes Contextos

Felipe Duarte Silva
Universidade Federal do Paraná – UFPR
felipeduarte@ufpr.br

***Resumo*—Este relatório aborda e compara diferentes algoritmos de ordenação, considerando sua eficiência, número de comparações e tempo de execução para diferentes tamanhos de entradas.**

***Index Terms*—Algoritmos, Merge Sort, Quick Sort, Heapsort, Counting Sort, Bucket Sort, Análise de eficiência e Comparação de desempenho.**

I. Introdução

Este relatório investiga três versões recursivas e duas versões iterativas de algoritmos de ordenação implementados em C. Para fins de testes, utilizamos vetores ordenados de maneira não crescente e vetores ordenados aleatoriamente. Enquanto analisamos a performance dos algoritmos de ordenação recursivos até quinze milhões de posições, o Counting Sort foi testado até cem milhões e o Bucket Sort foi limitado a cem mil. Antes dos resultados, será apresentado brevemente o algoritmo extra escolhido.

II. Algoritmo Extra

Bucket Sort sobressai pela distribuição uniforme de elementos em baldes, reduzindo comparações. Com complexidade $O(n+k)$ (sendo "k"a diferença máxima entre o valor máximo e mínimo dos elementos) é ideal para grandes conjuntos uniformes, otimizando tempo e permitindo paralelismo. A ordenação individual de baldes, auxiliada pelo Insertion Sort, confere precisão e eficiência. Este algoritmo equilibra uso de memória adicional contra ganhos significativos de desempenho, justificando sua escolha em contextos adequados.

III. Resultados dos Testes

As avaliações dos desempenhos dos algoritmos foram realizadas usando vetores configurados em ordem não crescente e em ordem aleatória testados em uma máquina com processador 11th Gen Intel® Core™ i7-1165G7 com velocidade base de 2.80GHz e um total de 15GiB de RAM. Sistema Operacional: Ubuntu 20.04.6. Os resultados são apresentados abaixo graficamente ilustrados com números de comparações aproximados.

A. Comparações por tamanho

Algoritmos recursivos: Segue representado na figura 1 o gráfico que compara o Merge Sort, Quick Sort e Heap Sort quanto ao número de comparações pelo tamanho do vetor ordenado em ordem não crescente e na figura 2 o gráfico que compara os mesmos algoritmos, com vetores ordenados, desta vez, aleatoriamente.

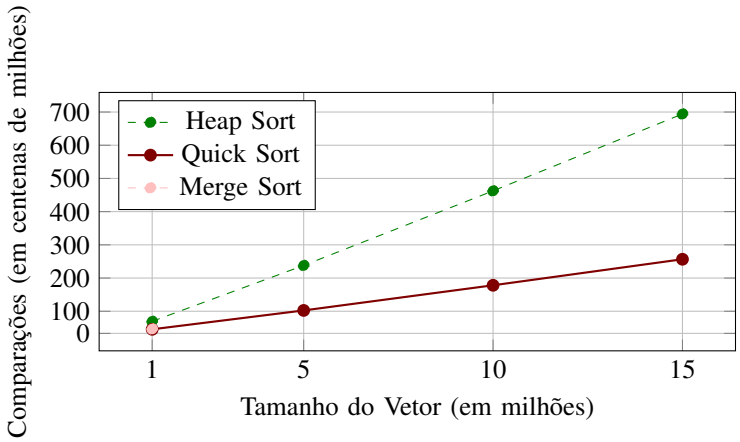

Figura 1: Comparação do número de comparações entre Merge Sort, Quick Sort e Heap Sort

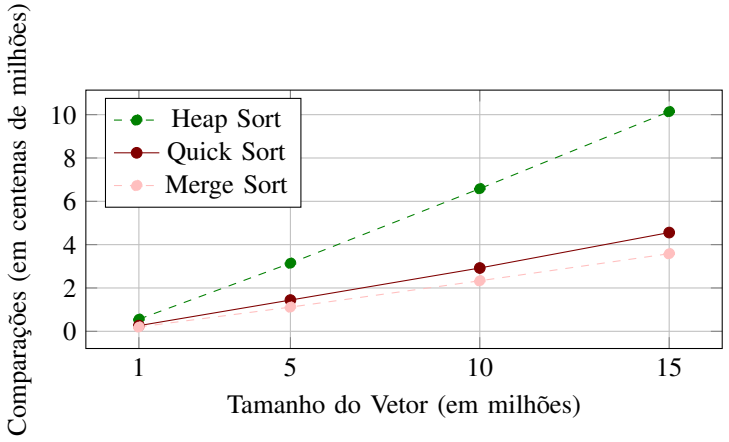

Figura 2: Comparação do número de comparações entre Merge Sort, Quick Sort e Heap Sort em vetores aleatórios

As Figuras 1 e 2 demonstram a quantidade de comparações feitas pelos algoritmos Merge Sort, Quick Sort e Heap Sort, relacionando-as com o tamanho do vetor. O Quick Sort, utilizando um pivô central, realiza menos comparações e se destaca pela eficiência, condizente com sua notação Big O de $O(n \log n)$ no caso médio. Essa eficiência é confirmada em vetores aleatórios, que simulam o comportamento médio esperado. O Merge Sort, também projetado para operar com complexidade $O(n \log n)$, evidencia um crescimento proporcional de comparações conforme o tamanho do vetor aumenta.

Já o Heap Sort, apesar de sua complexidade assintótica teórica equivalente de $O(n \log n)$, demanda um número maior de comparações devido à manutenção da estrutura de heap, o que é particularmente notório em entradas aleatórias, já que no pior caso, possui uma complexidade de $2n \log_2 n + O(n)$. Este padrão sublinha a relevância de selecionar o algoritmo de ordenação mais apropriado com base nas características específicas dos dados a serem ordenados.

Algoritmos iterativos: O Counting Sort organiza elementos sem permutações diretas, operando com contagem e posicionamento que lhe permitem alcançar complexidade $O(4n)$, quando k é igual a n. O Bucket Sort distribui elementos em 'baldes' e, em seguida, aplica um algoritmo de ordenação, como o Insertion Sort, para ordenar os conteúdos destes. O Bucket Sort foi avaliado em termos de número de comparações em vetores aleatórios e os resultados foram representados na tabela abaixo.

Tamanho do Vetor	Comparações Bucket Sort
10,000	3,703
100,000	36,798
1,000,000	368,080

B. Tempo de Execução por tamanho

Algoritmos recursivos: As representações das figuras 3 e 4 são os tempos de execuções dos algoritmos implementados recursivamente que ordenam vetores organizados em ordem não crescente e em ordem aleatória, respectivamente.

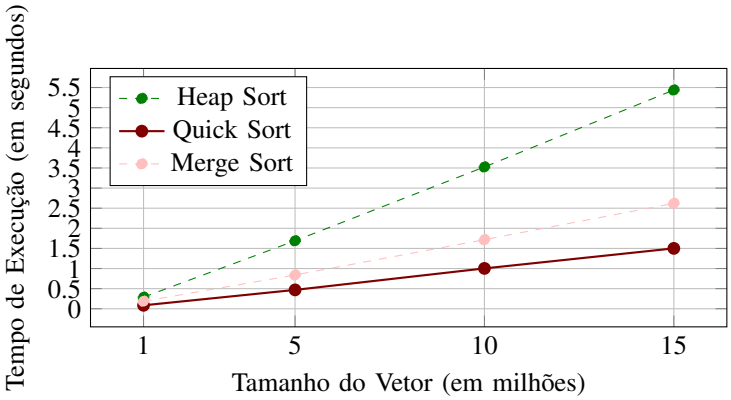

Figura 3: Comparação do tempo de execução entre Merge Sort, Quick Sort e Heap Sort

A Figura 3 mostra o desempenho dos algoritmos Merge Sort, Quick Sort e Heap Sort em vetores decrescentes. O Quick Sort se sobressai em eficiência, alinhando-se à sua complexidade de $O(n \log n)$, enquanto o Merge Sort mantém um aumento de tempo linear-logarítmico conforme o tamanho do vetor aumenta. O Heap Sort, embora também de complexidade $O(n \log n)$, tem o maior tempo de execução, refletindo a sobrecarga inerente à gestão da estrutura de heap.

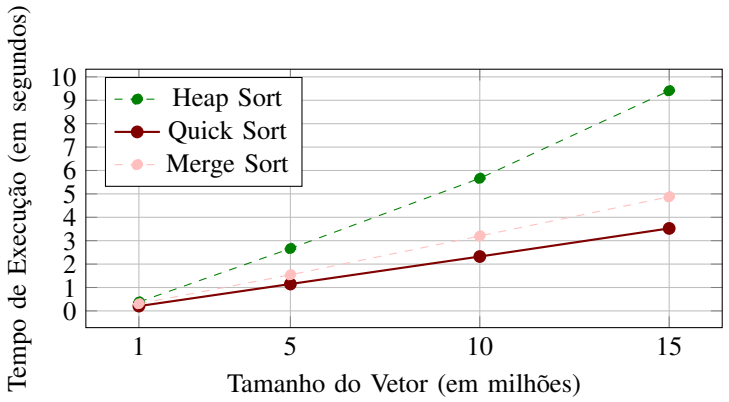

Figura 4: Comparação do tempo de execução entre Merge Sort, Quick Sort e Heap Sort em vetores aleatórios

A Figura 4 apresenta os tempos de execução dos algoritmos Merge Sort, Quick Sort e Heap Sort em vetores aleatórios. O Quick Sort destaca-se pela rapidez, alinhado à sua eficiência teórica de $O(n \log n)$. O Merge Sort segue com um tempo de execução moderado, demonstrando consistência. O Heap Sort, com os tempos mais longos, destaca a carga adicional imposta pela manipulação da estrutura de heap, que afeta mais intensamente conforme os vetores crescem em tamanho.

Algoritmos Iterativos: O Counting Sort exibe eficiência temporal com sua operação linear, refletindo uma complexidade de $O(n)$. Já o Bucket Sort, organizando elementos em 'baldes' para posterior ordenação, demonstra desempenho temporal variável, como mostrado na tabela a seguir.

Tabela I: Tempos de execução do Counting Sort e Bucket Sort para vetores de diferentes tamanhos ordenados aleatoriamente.

Tamanho do Vetor	Counting Sort - Tempo (s)	Bucket Sort - Tempo (s)
10,000	0.000333	0.037220
100,000	0.002383	0.337776
10,000,000	0.151620	N/A
100,000,000	1.537345	N/A

Nota: N/A mostrados na tabela representam a falta de memória na máquina de teste para o algoritmo em questão.

IV. Conclusão

A análise detalhada dos algoritmos de ordenação destaca que não há um algoritmo definitivamente superior; a escolha mais eficiente varia conforme as características dos dados. O Quick Sort se alinha à sua complexidade prevista de $O(n \log n)$, exibindo alta eficiência em diversos cenários com o pivô escolhido no elemento central do vetor. O Merge Sort se mostra consistente, e o Heap Sort, apesar de eficaz, tem seu desempenho afetado pela gestão do heap. O desempenho notável do Counting Sort em conjuntos específicos e o Bucket Sort, que se beneficia de sua estratégia de pré-ordenação, reforçam a importância da seleção adequada do algoritmo.