TDE 03 - Ordenação RA 04

Renan Antonio Hammerschmidt Krefta[1]

[1]Pontifícia Universidade Católica do Paraná (PUCPR)

renankrefta@hotmail.com

Abstract. *This document presents comparison and analysis between the Bubble Sort, Merge Sort and Quick Sort sorting algorithms and presents line graphics and tables containing the comparison data.*

Resumo. *Este documento apresenta comparações e a análise entre os algoritmos de ordenação Bubble Sort, Merge Sort e Quick Sort e apresenta gráficos e tabelas contendo os dados das comparações.*

1. Descrição do Código

Foram implementados no código os algoritmos de ordenação Bubble Sort, Merge Sort e Quick Sort. Os algoritmos foram testados a partir de um mesmo conjunto de elementos aleatórios definidos por uma semente.

Link Para o Código: https://github.com/Ziminny1/EstruturasDeDadosRA04

1.1. Bubble Sort

A função do Bubble Sort recebe como parâmetros um vetor de inteiros e o tamanho do vetor e então realiza comparações dos elementos do vetor a partir do primeiro índice e avançando consecutivamente sobre o vetor, comparando o proximo elemento com o elemento anterior.

1.2. Merge Sort

A função do Merge Sort, implementada de forma recursiva, recebe como parâmetros um vetor de inteiros, o início do vetor e o fim do vetor, que no começo são 0 e o tamanho do vetor respectivamente. A função verifica se o vetor que recebeu possui mais do que 1 elemento e então calcula o meio (fim + início) / 2, então ela chama outras duas funções de Merge Sort para separar o vetor em 2 e recursivamente ir ordenando-o. A função então chama uma função auxiliar chamada Merge para ordenar as duas partes restantes da lista de elementos.

A função de Merge recebe como parâmetros o vetor de inteiros, o início, o meio e o fim da lista. Ela então cria dois novos vetores representando a parte esquerda e parte direita do vetor recebido. Então ela realiza a comparação da parte esquerda com a parte direita, modificando os elementos no vetor original de forma ordenada.

1.3. Quick Sort

A função de Quick Sort recebe como parâmetros o vetor de inteiros, o início e o fim do vetor a ser ordenado. Ela então verifica se o início é menor do que o fim, para evitar casos

com um único elemento. Ela então chama uma função auxiliar particiona, que recebe como parâmetros o vetor, início e fim, com o intuito de receber um pivô, após receber o pivô do retorno da função particiona, ela chama outras duas funções Quick Sort para ordenar a parte esquerda e direita do vetor.

A função Particiona define o último elemento da lista como seu pivô, então define a posição dos elementos inferiores ao pivô como início. A função então entra em um loop onde são realizadas as comparações dos elementos com o pivô, se o elemento for menor ou igual ao pivô, ele é colocado na posição dos elementos inferiores ao pivô, se não, ele permanece no lugar.

2. Comparações

Os algoritmos de ordenação foram comparados das seguintes formas: tempo total para ordenar o vetor em nanosegundos, total de trocas realizadas pelos algoritmos e total de iterações até o fim do algoritmo.

Os dados utilizados são resultados de uma média dada por 15 execuções de cada algoritmo e os tempos são medidos em nanosegundos (ns).

Por questão de visibilidade, foram adicionadas as figuras (Figure 2),(Figure 4) e (Figure 6). Essas figuras excluem a representação do Bubble Sort.

2.1. Tabelas e Gráficos

Table 1. Tabela de Comparação do tempo levado por cada algoritmo

Qtd	BubbleSort	MergeSort	QuickSort
50	48473 ns	23206 ns	8106 ns
500	550813 ns	81893 ns	40306 ns
1000	1328373 ns	184393 ns	75180 ns
5000	30445200 ns	697433 ns	427286 ns
10000	136633993 ns	1498813 ns	851826 ns

Table 2. Tabela de Comparação do total de iterações

Qtd	BubbleSort	MergeSort	QuickSort
50	2450	286	243
500	249500	4488	4604
1000	999000	9976	11152
5000	24995000	61808	77027
10000	99990000	133616	151935

Table 3. Tabela de Comparação do total de trocas

Qtd	BubbleSort	MergeSort	QuickSort
50	651	251	146
500	59482	4149	2643
1000	248942	9300	6725
5000	6186179	57939	47976
10000	24930158	125932	93110

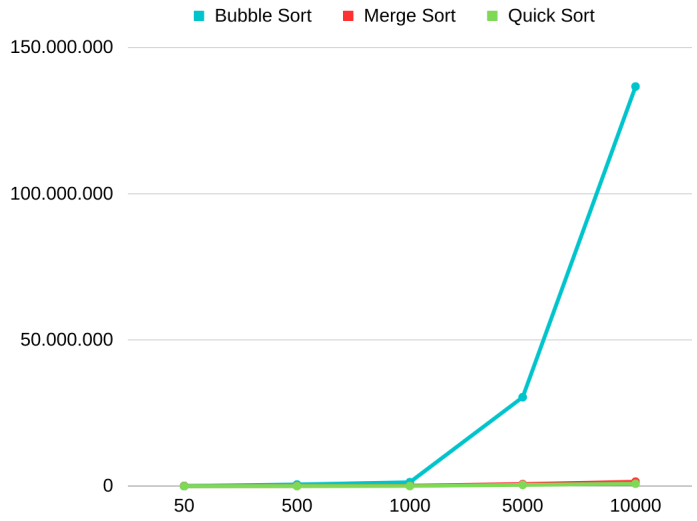

Figure 1. Comparação dos Tempos

Figure 2. Comparação dos Tempos (sem Bubble Sort)

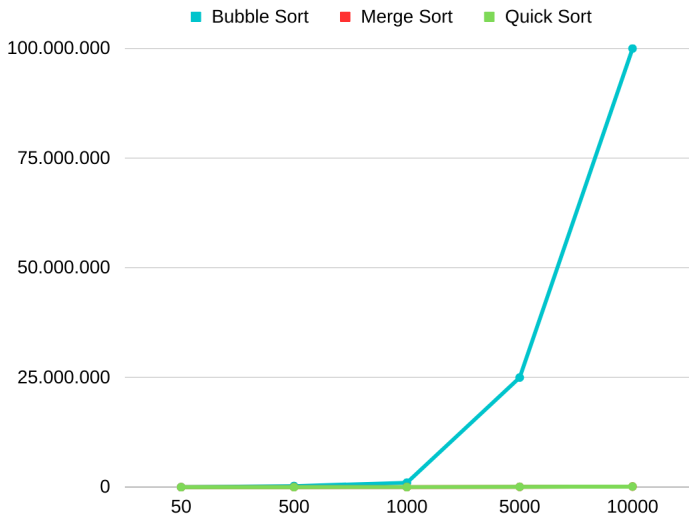

Figure 3. Comparação das iterações

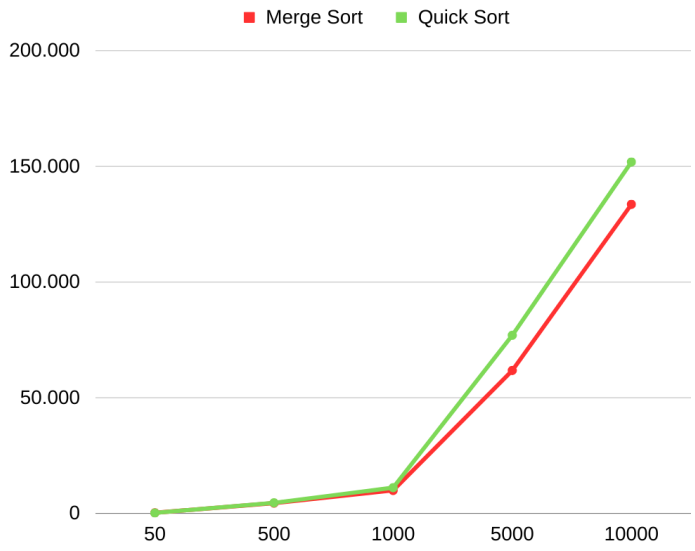

Figure 4. Comparação das iterações (sem Bubble Sort)

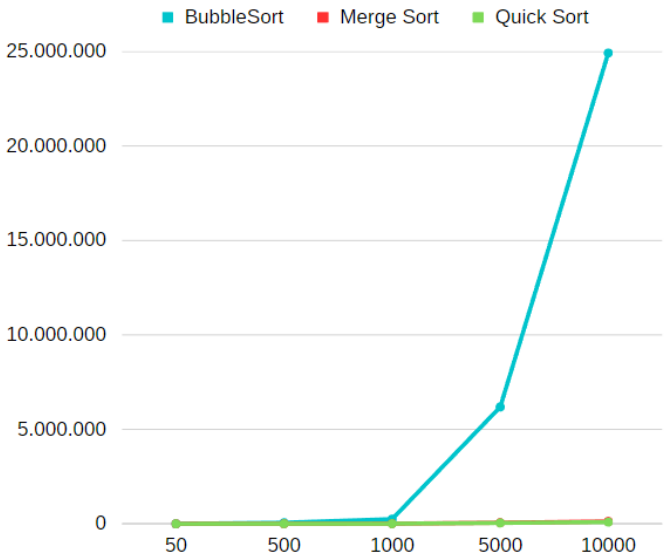

Figure 5. Comparação das trocas

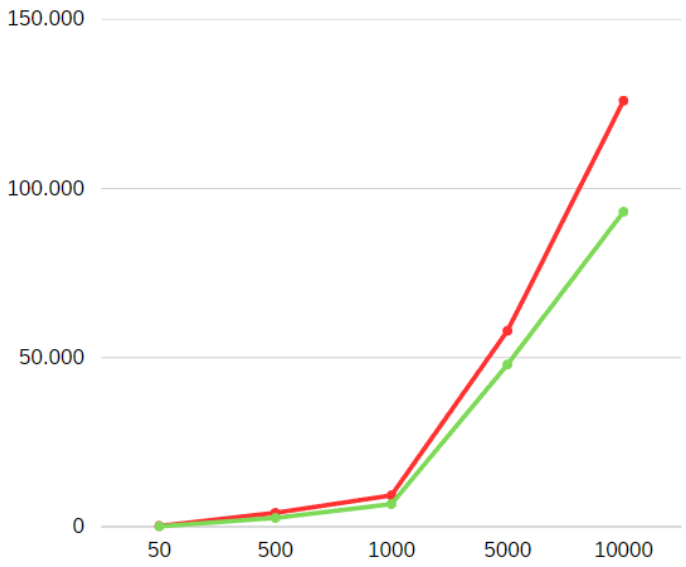

Figure 6. Comparação das trocas (sem Bubble Sort)

3. Conclusao

O Quick Sort é, no geral, mais rápido e finaliza a ordenação realizando um número menor de trocas quando em comparação com o Merge Sort. No quesito número de iterações, o Merge Sort finaliza com um número menor e quanto maior o vetor de elementos em questão, maior a diferença na quantidade total de iterações dos dois algorítmos.