Fúlvio Taroni Monteforte

Thiago Lima Bahia Santos

Trabalho Prático II: Implementação e Análise de Algoritmos de Ordenação por Comparações de Chaves

CEFET-MG

Belo Horizonte

2019

1. INTRODUÇÃO

1.1 Contextualização

Segundo Ziviani(2011), algoritmos de ordenação são algoritmos responsáveis por rearranjar um conjunto de elementos em uma determinada ordem, crescente ou decrescente**,** com objetivo principal de facilitar a recuperação de itens desse conjunto. Existem diversos algoritmos de ordenação e cada técnica pode possuir uma vantagem em particular dependendo da aplicação.

Nesse trabalho foram analisados os seguintes algoritmos de ordenação interna, divididos em eficientes e não eficientes.

- Não eficientes:

1- Bubblesort

2- Inserção

3- Seleção

- Eficientes:

1- Quicksort

2- Heapsort

3- Mergesort

1.2 Análise de Complexidade

Algoritmo	**[1]Complexidade no pior caso**
Bublesort	$O(n^2)$
Inserção	$O(n^2)$
Seleção	$O(n^2)$
Quicksort	$O(n^2)$
Heapsort	O(n log n)
Mergesort	O(n log n)

1 Batista, N. “Notas de aula”

2 IMPLEMENTAÇÃO

2.1 Estruturas de dados utilizadas

As estruturas de dado utilizadas foram vetores de estruturas do tipo item. Essa estrutura “TipoItem” contém os seguintes campos: “TipoChave Chave” e “TipoChave PosOriginal”. TipoChave é uma definição do tipo “long” para o tipo “TipoChave”.

A “Chave” é um valor inteiro que representa o item enquanto a “PosOriginal” é um outro inteiro que guarda a posição original do item dentro do vetor antes dele ser ordenado.

Chave 30 PosOriginal 0	Chave 11 PosOriginal 1	Chave 2 PosrOiginal 2	Chave 151 PosOriginal 3	Chave 9 PosOriginal 4	Chave 1 PosOriginal 5

Exemplo de vetor de TADs do tipo “TipoItem”

2.2 Funcionamento do programa

O programa funciona esperando receber os parâmetros para a técnica de ordenação que deseja ser usada, o tamanho do vetor a ser ordenado, o tipo do vetor (se está ordenado ascendentemente, descendentemente, desordenado ou quase ordenado) e o que deseja ser impresso.

Foram utilizadas as implementações fornecidas no site do Ziviani para os algoritmos de ordenação exceto para o “Mergesort”, para esse foi utilizado o código visto em sala de aula nos slides de apresentação.

Foi implementada uma função adicional no programa que verifica se o algoritmo é estável ou não a partir da segunda chave criada para o item, que guarda a posição original dele no vetor.

Para os testes, nos algoritmos simples (Bubblesort, Inserção e Seleção) foram realizados testes com vetores de tamanho 100, 1000, 10000, 20000 itens. Para os algoritmos eficientes foram usados vetores de tamanho 100, 1000, 10000 e 100000 itens.

Ao adicionar uma segunda chave na estrutura de tipo item o maior teste possível é com um vetor de 200.000 itens, mais que isso ocorre falha de segmentação.

3 EXPERIMENTOS E ANÁLISE DE RESULTADOS

3.1 Configurações da máquina utilizada

Sistema:

• Windows 10 Home

• Processador: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz

• Memória instalada (RAM): 8,00 GB (utilizável 7,86 GB)

Máquina Virtual:

• Ubuntu 18.04 Desktop

• Processador(es) utilizados: 2 de 8 threads

• Memória base: 4,00 GB

A implementação e os testes dos algoritmos foram realizadas no ambiente de uma máquina virtual emulada a partir do software Oracle VM VirtualBox.

3.2 Testes Realizados

Para cada entrada o programa foi rodado 3 vezes e foi feito a média aritmética dos tempos obtidos

3.2.1 Algoritmos eficientes

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Quicksort**	**Heapsort**	**Mergesort**
100	604	583	511
1000	758	926	694
10000	2956	3779	3144
100000	25580	35791	28042

Vetor ordenado em ordem crescente (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Quicksort**	**Heapsort**	**Mergesort**
100	524	520	585
1000	754	790	816
10000	3030	3582	2828
100000	23155	29834	29694

Vetor ordenado em ordem decrescente (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Quicksort**	**Heapsort**	**Mergesort**
100	579	478	541
1000	762	774	739
10000	2451	2924	2615
100000	22369	32517	23938

Vetor aleatório (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Quicksort**	**Heapsort**	**Mergesort**
100	502	628	670
1000	760	779	923
10000	3115	3357	3222
100000	31134	35716	30846

Vetor 20% desordenado(Tempo em microssegundos)

3.2.2 Algoritmos SIMPLES

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Bubblesort**	**Inserção**	**Seleção**
100	742	689	619
1000	4732	829	2748
10000	217389	3638	106322
20000	8314676	5971	401289

Vetor ordenado em ordem crescente (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Bubblesort**	**Inserção**	**Seleção**
100	657	628	670
1000	5554	779	923
10000	304881	3357	3222
20000	1218032	35716	30846

Vetor ordenado em ordem descrescente (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Bubblesort**	**Inserção**	**Seleção**
100	502	734	732
1000	760	2840	2721
10000	3115	118433	154656
20000	31134	436567	655122

Vetor aleatório (Tempo em microssegundos)

	Tempos obtidos com a função getrusage		
Tamanho do Vetor	**Bubblesort**	**Inserção**	**Seleção**
100	690	576	684
1000	7003	1755	2731
10000	385444	58890	103052
20000	1552312	223123	405664

Vetor 20% desordenado (Tempo em microssegundos)

4 CONCLUSÃO

Analisando os testes realizados com os algoritmos de ordenação propostos no trabalho, foi possível concluir que dentre os algoritmos de ordenação eficientes, o que apresentou testes mais rápidos para vetores de mesmo tamanho foi o algoritmo “quicksort” e o que se manteve mais estável independente da forma original do vetor foi o algoritmo “heapsort”. Também foi possível perceber uma queda de desempenho no algoritmo “quicksort” quando o vetor já estava ordenado tanto crescente quanto decrescente

Dentre os algoritmos simples, o algoritmo que apresentou piores casos de teste foi o “bubblesort” enquanto o método de “inserção” se mostrou o mais rápido.

Além disso, usando a função criada para testar a estabilidade do algoritmo, os únicos que se apresentaram estáveis foram os métodos de inserção, bubblesort e mergesort.

5 REFERÊNCIAS BIBLIOGRÁFICAS

Nivio Ziviani. **Projeto de algoritmos: com implementaçõaaes em Java e C++.** 3 ed.

Editora Cengage Learning, 2007.

Natalia Batista. **Notas de aula e slides disponibilizados sobre o tema do trabalho**

CEFET-MG, 2019 – Campus II