RELATORIO DE DESEMPENHO DOS ALGORITMOS DE ORDENAÇÃO

Rafael Rodrigues Monteiro[1]

INTRODUÇÃO

Um tipo de algoritmo muito usado na resolução de problemas computacionais são os algoritmos de ordenação, que servem para ordenar/organizar uma lista de números ou palavras de acordo com a sua necessidade. A ordenação é um dos inúmeros problemas clássicos da computação, com diversas variações, algumas mais simples, outras mais complexas. Existem inúmeros algoritmos de ordenação e eles variam em relação a dificuldade de implementação e desempenho. Este relatório tem como objetivo a implementação de um software que possibilite comparar o desempenho dos métodos de ordenação conhecidos como *SelectSort, InsertSort, QuickSort e MergeSort.*

IMPLEMENTAÇÃO

O software desenvolvido é subdivido em diferentes arquivos, sendo um arquivo para cada método de ordenação testado. Contudo, todos funcionam da mesma forma.

Os testes são realizados em vetores de tamanho 1.000, 10.000, 100.000 e 200.000, para isso uma função de alocação de memória foi desenvolvida para trabalhar com os 4 tamanhos diferentes. O tamanho do vetor a ser criado é passado através da chamada da função.

Uma função para preencher o vetor também foi desenvolvida, a função consegue trabalhar com tamanhos de vetores diferentes e colocar os números de forma aleatório, ordenada e inversamente ordenada. O tamanho e modo de preenchimento também é passado através da chamada da função.

Os testes se iniciam utilizando o método aleatório com os tamanhos variando entre 1.000, 10.000, 100.000 e 200.000. Os testes em vetores aleatórios são repetidos 10 vezes com intuito de obter a uma média entre eles, tornando os dados mais precisos. Após o método aleatório ser finalizado, iniciasse o teste em vetores ordenados e na sequência os inversamente ordenados, esses por vez são testados apenas uma vez nos quatro tamanhos diferentes.

[1] Aluno de Graduação – UEMG. E-mail: rafael.1697550@discente.uemg.br

O software é capaz de calcular o tempo necessário para ordenar o vetor através da função *clock()*, ele também registra o numero de movimentações e comparações necessárias para ordenar o vetor.

Todos os dados obtidos durante os testes são registrados em um arquivo .log que é criado e preenchido enquanto o programa ainda está em execução.

ESTUDO DE COMPLEXIDADE

Abaixo encontrasse o estudo de complexidade de cada método utilizado no software.

Select Sort

O Select Sort é um algoritmo de comparação instável com baixo desempenho. O Select Sort utiliza o método de seleção e apresenta um desempenho de $O(n^2)$.

Insert Sort

Insertion Sort é um algoritmo de comparação estável com baixo desempenho. O Insertion Sort usa o método de inserção e, embora possa ser executado em O(n) na melhor das hipóteses, ele atua em $O(n^2)$, na média e na pior das hipóteses.

Quick Sort

Quick Sort é um algoritmo de comparação instável. Quick Sort usa o método de particionamento e pode executar, na melhor e na média, em O(n log (n)). Ele pode, no entanto, se apresentar em $O(n^2)$, na pior das hipóteses

Merge Sort

Merge Sort é um algoritmo de comparação estável com desempenho excepcional. O Merge Sort usa o método de fusão e executa em O(n log (n)) no melhor, médio e pior caso.

Algoritmo	Comparações			Movimentações			Espaço	Estável	In situ
	Melhor	Médio	Pior	Melhor	Médio	Pior			

Selection	$O(n^2)$			$O(n)$		$O(1)$	Não*	Sim
Insertion	$O(n)$	$O(n^2)$		$O(n)$	$O(n^2)$	$O(1)$	Sim	Sim
Merge	$O(n \log n)$			–		$O(n)$	Sim	Não
Quick	$O(n \log n)$		$O(n^2)$	–		$O(n)$	Não*	Sim

TESTES EXECUTADOS

Tempos de execução em vetores de tamanho 1.000

Método Ordenação	Aleatório	Ordenado	Inversamente Ordenado
Select Sort	0.0011s	0.0010s	0.0010s
Insert Sort	0.0009s	0.0000s	0.0010s
Quick Sort	0.0003s	0.0000s	0.0000s
Merge Sort	0.0002s	0.0000s	0.0000s

Tempos de execução em vetores de tamanho 10.000

Método Ordenação	Aleatório	Ordenado	Inversamente Ordenado
Select Sort	0.1206	0.1360s	0.1030s
Insert Sort	0.0993	0.0000s	0.1430s
Quick Sort	0.0012	0.0010s	0.0000s
Merge Sort	0.0047	0.0010s	0.0030s

Tempos de execução em vetores de tamanho 100.000

Método Ordenação	Aleatório	Ordenado	Inversamente Ordenado
Select Sort	12.7677s	11.1940s	10.5550s
Insert Sort	7.8296s	0.0000s	15.5060s
Quick Sort	0.0115s	0.0030s	0.0040s
Merge Sort	0.0284s	0.0250s	0.0230s

Tempos de execução em vetores de tamanho 200.000

Método Ordenação	Aleatório	Ordenado	Inversamente Ordenado
Select Sort	47.2608s	47.1640s	43.6930s
Insert Sort	33.0932s	0.0010s	64.1500s
Quick Sort	0.0236s	0.0080s	0.0090s
Merge Sort	0.0645s	0.0540s	0.0440s

CONCLUSÃO

Selection Sort

Nos vetores ordenados e aleatórios, o Selection sort foi o segundo pior algoritmo, mas se mostrou mais eficiente do que o Insertion sort em relação ao tempo no vetor inversamente ordenado

Insertion Sort

No vetor ordenado, o Insertion sort se mostrou mais eficiente que todos os outros algoritmos em relação ao tempo. No vetor inversamente ordenada foi menos eficiente do que o Selection Sort e na desordenada aleatória a diferença de tempo entre o Insertion e o Selection foi pequena.

Quick Sort

O Quick Sort certamente é o algoritmo mais eficiente em vetores totalmente desordenados, ele se torna muito eficiente em relação aos outros no quesito de tempo. Em lista inversamente

ordenadas e aleatórias a diferença de tempo do Quick Sort em comparação aos outros foi absurdamente grande. Seu ponto fraco se da pelo fato de ser $O(n^2)$ em seu pior caso.

Merge Sort

O Merge Sort também é um algoritmo eficiente em vetores totalmente desordenados, perdendo apenas para o Quick Sort no quesito tempo. Sua prinpal diferença se da pelo fato de ser O(n log n) para todos os casos.

Quick Sort x Merge Sort

Em vetores de tamanho máximo de 10 mil, algoritmos mais simples como Selection Sort e Insertion Sort podem ser empregados sem que tenhamos uma grande perda de desempenho.

Quando entramos no quesito de vetores maiores que 10 mil, algoritmos como Merge Sort e Quick Sort

S.No.	Quick Sort	Merge Sort
1.	Segue o método de divisão e conquista.	Também segue o método de dividir e conquistar.
2.	Aqui, classificamos os componentes comparando cada componente com o pivô.	Ele divide a matriz em dois segmentos ou subarrays na repetição até que um componente seja deixado.
3.	É ineficiente para grandes matrizes em comparação com o tipo de fusão.	É mais eficiente em comparação com o tipo rápido.
4.	É uma técnica interna de triagem.	É uma técnica de triagem externa.
5.	A complexidade do tempo para o pior caso é O (n2).	A complexidade do tempo para o pior caso é O (n log n).
6.	O tipo rápido é principalmente preferido para grandes matrizes não sortidas.	O tipo de fusão é aplicável principalmente para as listas vinculadas.
7.	É um algoritmo de tipo instável. Temos que torná-lo estável mudando o código.	É um algoritmo de tipo estável.
8.	Aqui não precisamos de espaço adicional para executar a tarefa.	Aqui precisamos de mais espaço para realizar operações.

Outro fato importante que devemos levar em conta é que o Merge Sort precisa de memória adicional para realizar a ordenação. Cabe ao desenvolvedor escolher o melhor algoritmos para o seu código, tendo em vista que todos os métodos apresentam vantagens e desvantagens.

REFERENCIAS

Difference between Quick Sort and Merge Sort. **Byjus.** Disponível em: https://byjus.com/gate/difference-between-quick-sort-and-merge-sort/. Acesso em: 19 mar. 2022.

Algoritmos de ordenação: análise e comparação. **Dev Media.** Disponível em: https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261. Acesso em: 19 mar. 2022.