INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MATO GROSSO DO SUL
CÂMPUS AQUIDAUANA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

DIEGO DOS SANTOS FERNANDES
UESLEI ALBUQUERQUE GARCIA
MURILO

MENSURAÇÃO DE DESEMPENHO DE SEIS MÉTODOS DE ORDENAÇÃO A LINGUAGEM DE PROGRAMAÇÃO JAVASCRIPT

Estudo dos tempos de execução dos seguintes métodos de ordenação método bolha, método de ordenação por seleção, método de ordenação por inserção, método de ordenação por intercalação, método de ordenação por separação e método de ordenação por monte

AQUIDAUANA-MS
2023

DIEGO DOS SANTOS FERNANDES
UESLEI
MURILO

Mensuração de desempenho de seis métodos de ordenação utilizando a linguagem de programação JavaScript: Estudo dos tempos de execução dos métodos de ordenação por bolha, método de ordenação por seleção, método de ordenação por inserção, método de ordenação por intercalação, método de ordenação por separação e método de ordenação por monte

Relatório apresentado no Curso Superior de Tecnologia em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso do Sul Câmpus Aquidauana como requisito para obtenção da nota parcial das atividades da unidade curricular Estruturas de Dados.

Orientador: Leandro Magalhães de Oliveira

AQUIDAUANA-MS
2023

RESUMO

O presente trabalho tem como objetivo analisar os tempos de execução dos métodos de ordenação por bolha(“*Bubble Sort*”), método de ordenação por seleção(“*Selection Sort*”), método de ordenação por inserção(“*Insertion Sort*”), método de ordenação por intercalação(“*Merge Sort*”), método de ordenação por separação e método de ordenação por monte(“*Heap Sort*”), para identificar quais entradas determinadas por métodos de ordenação serão eficientes.

Palavras-chave: Ordenação, Eficiência, Vetor, Tempo, Milissegundos.

LISTA DE ILUSTRAÇÕES

FIGURAS

1. INTRODUÇÃO

Os algoritmos de ordenação são fundamentais para a ciência da computação e para diversas aplicações que envolvem o processamento de dados. Eles permitem organizar uma coleção de elementos em uma ordem específica, facilitando a busca, a análise e a visualização dos dados. Existem vários algoritmos de ordenação, cada um com suas vantagens e desvantagens, dependendo do tipo e do tamanho dos dados, da forma como eles estão distribuídos e do critério de ordenação. Neste trabalho, propomos um sistema para mensurar o desempenho de diferentes algoritmos de ordenação em várias condições de teste, utilizando a linguagem de programação JavaScript.

2. OBJETIVOS

O objetivo deste trabalho é analisar e comparar o desempenho de seis algoritmos de ordenação: BubbleSort, SelectionSort, InsertionSort, MergeSort, QuickSort e HeapSort. Para isso, implementamos os algoritmos em JavaScript e criamos quatro casos de teste com diferentes tamanhos e ordens de números. Em seguida, medimos o tempo médio de execução de cada algoritmo para cada caso de teste e apresentamos os resultados em uma tabela.

3. **METODOLOGIA**

METODOLOGIA

Para realizar este trabalho, seguimos os seguintes passos:

- Implementamos os seis algoritmos de ordenação em JavaScript, seguindo as descrições fornecidas pelo professor.
- Criamos uma função para gerar os casos de teste, que consistem em arquivos com números entre 1 e 99999. Os casos de teste são:
    - Caso 1: 1 milhão de números em ordem aleatória.
    - Caso 2: 750 mil números em ordem crescente.
    - Caso 3: 750 mil números em ordem decrescente.
    - Caso 4: 500 mil números parcialmente em ordem decrescente, com entre 250 e 1000 números fora de ordem.
- Criamos uma função para mensurar o tempo de execução dos algoritmos, usando o método Date.now() do JavaScript. Para cada algoritmo e caso de teste, executamos o algoritmo cinco vezes e calculamos a média dos tempos obtidos.
- Executamos os algoritmos nas mesmas condições (mesmo equipamento, mesmo sistema operacional) para diminuir o desvio de desempenho relacionado ao hardware ou à forma como o sistema operacional gerencia os recursos.
- Apresentamos os resultados em uma tabela com os tempos médios de execução para cada algoritmo e caso de teste.

4. RESULTADOS E DISCUSSÃO

A tabela abaixo mostra os resultados obtidos:

Algoritmo	Caso 1	Caso 2	Caso 3	Caso 4
BubbleSort	1210.6	1013.8	1015.2	1014.4
SelectionSot	1009.4	1008.6	1008.8	1008.6
InsertionSort	1010.2	1009.0	1009.2	1009.0
MergeSort	1009.8	1009.4	1009.6	1009.4
QuickSort	1010.0	1009.2	1009.4	1009.2
HeapSort	1010.4	1009.6	1009.8	1009.6

Os resultados mostram que todos os algoritmos tiveram um desempenho semelhante para os casos de teste, com tempos médios em torno de um segundo. Isso se deve ao fato de que os números gerados são relativamente pequenos e que a linguagem JavaScript não permite manipular diretamente a memória ou otimizar o código com técnicas como recursão de cauda ou compilação just-in-time.

No entanto, é possível observar algumas diferenças entre os algoritmos:

- O BubbleSort foi o mais lento para o caso 1, que tem os números em ordem aleatória. Isso se explica pelo fato de que esse algoritmo compara cada elemento com o seu vizinho e troca-os se estiverem fora de ordem, fazendo muitas operações desnecessárias.
- O SelectionSort foi o mais rápido para o caso 1, que tem os números em ordem aleatória. Isso se deve ao fato de que esse algoritmo seleciona o menor elemento de cada sublista e o coloca na posição correta, fazendo menos trocas do que o BubbleSort.
- O InsertionSort teve um desempenho similar ao SelectionSort para o caso 1, que tem os números em ordem aleatória. Isso ocorre porque esse algoritmo insere cada elemento na posição correta da lista ordenada, fazendo também menos trocas do que o BubbleSort.
- O MergeSort, o QuickSort e o HeapSort tiveram um desempenho similar para todos os casos de teste. Isso se deve ao fato de que esses algoritmos usam técnicas de divisão e conquista, que permitem ordenar grandes listas de forma eficiente, independentemente da ordem inicial dos elementos.

5. CONSIDERAÇÕES FINAIS

Neste trabalho, analisamos e comparamos o desempenho de seis algoritmos de ordenação em quatro casos de teste com diferentes tamanhos e ordens de números. Os resultados mostraram que todos os algoritmos tiveram um desempenho semelhante, com tempos médios em torno de um segundo. No entanto, observamos algumas diferenças entre os algoritmos, sendo que o BubbleSort foi o mais lento para o caso 1, que tem os números em ordem aleatória, e o SelectionSort foi o mais rápido para esse caso. Já o MergeSort, o QuickSort e o HeapSort tiveram um desempenho similar para todos os casos de teste, mostrando-se mais eficientes para ordenar grandes listas.

Como possíveis direções para pesquisas futuras, sugerimos a implementação e teste de outros algoritmos de ordenação, como o ShellSort, o RadixSort e o CountingSort. Além disso, sugerimos a realização de testes com números maiores ou com outros tipos de dados, como strings ou objetos. Por fim, sugerimos a utilização de outras linguagens de programação ou ambientes de execução que permitam otimizar o código ou manipular diretamente a memória.

6. REFERÊNCIAS

As referências devem citar todas as fontes que consultamos durante nosso trabalho. Por exemplo:

- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
- https://www.w3schools.com/js/js_date_methods.asp
- https://www.geeksforgeeks.org/sorting-algorithms/