Hérick Vitor Vieira Bittencourt
UNIVALI - Universidade do Vale de Itajaí
Ciências da Computação – Estrutura de dados
Atividade de Implementação 3 – Ordenação
Professor - Marcos Cesar Cardoso Carrard
Data - 22/06/2023

Herick@edu.univali.br

1 Introdução

O programa desenvolvido para a análise de performance dos algoritmos de ordenação utiliza Insertion, Bubble, Selection e Merge sort para a realização do procedimento, é solicitado ao usuário a quantidade de arrays aleatórios, o tamanho dos arrays e quantas vezes cada um dos arrays devem ser testados por cada algoritmo de ordenação.

Após a entrada de dados, o programa gera todos os arrays, incluindo um array extra no início e no final da lista de arrays, representando o melhor caso (ordenado de forma crescente) e o pior caso (ordenado de forma decrescente) e então cria um vetor tridimensional[X][Y][Z] cujo irá armazenar todos os resultados obtidos, com X representando o algoritmo de ordenação, Y o índice do array na lista de arrays e X o tempo obtido durante a X+1° tentativa, um for loop é executado na qual atravessa o vetor tridimensional chamando pela função ordenar (escolhe um algoritmo de ordenação conforme o valor de X do array tridimensional) em todos os casos e retorna o tempo delta entre o começo e o fim da ordenação (a função também retorna o array ao estado original após acabar para poder repetir mais vezes), ao finalizar um array, um valor flutuante é atualizado, mostrando a porcentagem de completação do programa e a média de tempo é comparada com o pior/melhor resultado já registrado no algoritmo utilizado, se houver um novo recorde, os valores do recorde são substituídos pelo novo array mais rápido/lento do algoritmo utilizado.

Ao final do programa, o vetor tridimensional tem seus resultados comprimidos em um vetor bidimensional[X][Y] com a média aritmética aplicada, X ainda representa o algoritmo, enquanto Y agora representa o melhor caso (crescente), caso médio (todos os resultados entre o index 1 e N-2) e pior caso (decrescente), os resultados são mostrados ao usuário, incluindo os recordes de cada algoritmo e o programa é finalizado.

2. Análise dos resultados [1]

```
ESTATISTICAS DE ORDENACAO GERADAS APOS 42 MINUTOS
--------------------
Dados fornecidos pelo usuario:
Quantidade de arrays aleatorios: 100
Tamanho dos arrays: 10000
Vezes que cada array deve ser submetido aos algoritmos: 50
--------------------
RESULTADOS FINAIS:
--------------------
Insertion Sort:
Melhor caso: 0ms
Caso medio: 77.4548ms
Pior caso: 155.02ms
Tempo do melhor vetor aleatorio: 0ms
Tempo do pior vetor aleatorio: 155.02ms
--------------------
Bubble Sort:
Melhor caso: 0ms
Caso medio: 293.268ms
Pior caso: 271.86ms
Tempo do melhor vetor aleatorio: 0ms
Tempo do pior vetor aleatorio: 295.38ms
```

```
--------------------
Selection Sort:
Melhor caso: 131.3ms
Caso medio: 130.242ms
Pior caso: 122ms
Tempo do melhor vetor aleatorio: 122ms
Tempo do pior vetor aleatorio: 131.3ms
--------------------
Merge Sort:
Melhor caso: 1.02ms
Caso medio: 2.0026ms
Pior caso: 1.02ms
Tempo do melhor vetor aleatorio: 1.02ms
Tempo do pior vetor aleatorio: 2.26ms

Process finished with exit code 0
```

Figura 1 e 2: Resultados de execução

2-A. Insertion Sort

O algoritmo de ordenação por inserção no melhor caso possui uma complexidade de O(n), enquanto em seu pior caso e caso médio possui uma complexidade de $O(n^2)$, isto significa que o melhor caso de um insertion sort apenas ocorre caso o array já esteja ordenado como demonstrado pela figura 3, no entanto, a maioria dos casos ocorrem em complexidade $O(n^2)$, elevando ao quadrado a quantidade de operações necessárias a cada elemento novo no array.

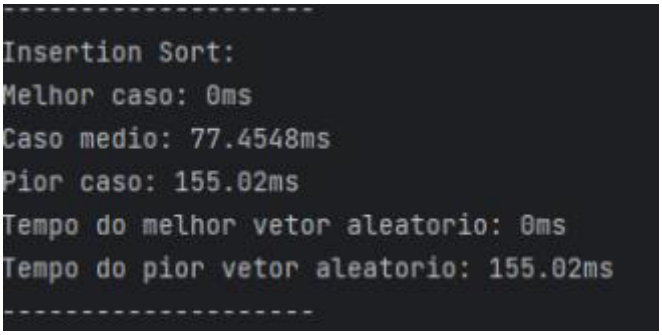

Figura 3: Resultados da ordenação por inserção

2-B. Bubble Sort

Apesar de sua simplicidade, a ordenação bolha é extremamente ineficiente, em todos os casos ele possui uma complexidade de $O(n^2)$, isto significa que a não ser que o array já esteja ordenado, a operação possui uma média extremamente próxima de seu pior caso e pode levar muito tempo para a operação ser concluída em vetores de larga escala, com a única exceção sendo a busca por um array crescente.

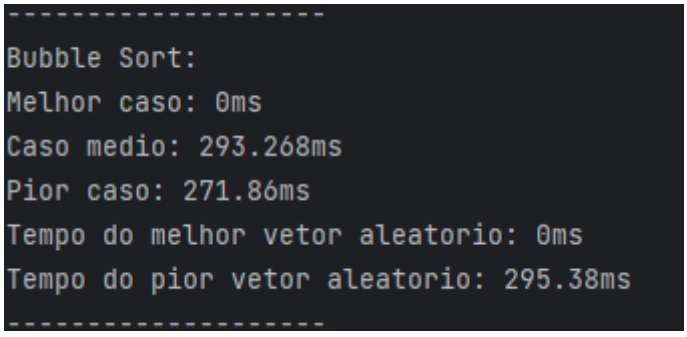

Figura 4: Resultados da ordenação bolha

2-C. Selection Sort

A ordenação por seleção possui uma média de tempo melhor do que o bubble sort devido a sua lógica se comportar melhor diante do caso médio, no entanto ainda possui a complexidade geral de $O(n^2)$, a diferença notável entre os dois é que a ordenação por seleção performa com mais eficiência o pior caso do bubble sort, no entanto, se o array já está ordenado, o algoritmo realizará muito mais operações para chegar à conclusão de que a ordenação está pronta.

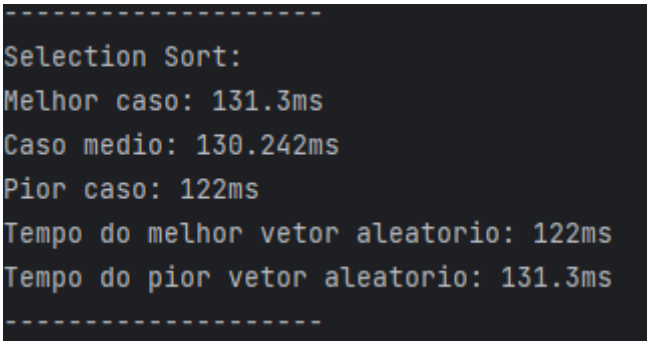

Figura 5: Resultados da ordenação por seleção

2-D. Merge Sort

O merge sort possui um comportamento especial comparado aos outros algoritmos testados, na qual devido ao seu comportamento recursivo, sua complexidade sempre será de O (n log n), sendo independente da quantidade de elementos no array, se tornando cada vez mais eficaz de usar conforme maior o array, diferentemente dos outros algoritmos testados, na qual perdem performance conforme o array aumenta.

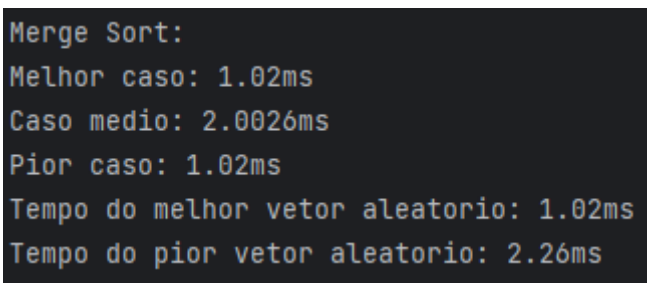

Figura 6: Resultados do Merge Sort

Pós-Escrito: o display dos recordes de cada algoritmo INCLUI o melhor/pior caso e isto é intencional, código fonte final atualizado para não aparentar o contrário.

Figura 7: Correção Pós-Escrito

Bibliográfia

[1] - VIANA, Daniel. Conheça os principais algoritmos de ordenação. **Treinaweb,** 2017. Disponível em: http://web.archive.org/web/20230619031906/https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao. Acesso em: 19 jun. 2023.