Trabalho Prático de Práticas de Programação

Métodos de Ordenação

Nome: Mariana Cristina de Almeida Alipio

Arquivo: TrabalhoPratico1
https://github.com/MariAlipio/ProjetoPratica

Introdução

Os algoritmos de ordenação, tem por objetivo colocar elementos de uma dada sequência em uma certa ordem, crescente ou decrescente. O objetivo de uma ordenação é, por exemplo, facilitar a recuperação dos dados de uma lista.

Os métodos simples são adequados para pequenos vetores, são programas pequenos e fáceis de entender. Possuem complexidade $C(n) = O(n^2)$, ou seja, requerem $O(n^2)$ comparações.

No presente estudo, foram escolhidos alguns algoritmos de ordenação para serem estudado, são eles: Selection Sort, Insertion Sort, Bubble Sort, Merge Sort e Quick Sort.

Selection Sort, Insertion Sort, Bubble Sort são métodos simples, ou seja, são métodos adequados para pequenos vetores, são programas pequenos e fáceis de entender.

O método *Selection sort* seleciona o menor item no vetor e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição e segue estes passos até que reste um único elemento, que por sua vez estará na última posição. Para todos os casos possui complexidade $C(n) = O(n^2)$ e não é um algoritmo estável.

Insertion Sort, funciona percorrendo um vetor de elementos da esquerda para a direita e à medida que avança ordena os elementos à esquerda. Sua complexidade é $C(n) = O(n)$ no melhor caso e é $C(n) = O(n^2)$ no caso médio e pior caso. É considerado um método de ordenação estável, ou seja, a ordem relativa dos itens iguais não se altera durante a ordenação.

Bubble sort é o método mais simples e o menos eficiente. Nele, cada elemento da posição i será comparado com o elemento da posição posterior. Por causa

dessa formatação, o vetor terá que ser percorrido quantas vezes que for necessária, tornando o algoritmo ineficiente para listas muito grandes.

Já os métodos Merge Sort e Quick Sort são conhecidos como Métodos eficientes, portanto, são mais complexos nos detalhes e requerem menos comparações. São projetados para trabalhar com uma quantidade maior de dados.

O *Quicksort*, é o método de ordenação interna mais rápido que se conhece para uma ampla variedade de situações. Provavelmente é o mais utilizado. Possui complexidade $C(n) = O(n^2)$ no pior caso e $C(n) = O(n \log n)$ no melhor e médio caso e não é um algoritmo estável.

É um algoritmo de comparação que emprega a estratégia de *"divisão e conquista"*. A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores. Os vetores menores são ordenados de forma independente e combina-se os resultados para produzir a solução final.

Criado em 1945 o *Mergesort* é um método de ordenação que, assim como o QuickSort, faz uso da estratégia "dividir para conquistar" para resolver problemas. É um método estável e possui complexidade $C(n) = O(n \log n)$ para todos os casos.

Ele divide o problema em pedaços menores, resolve cada um separadamente e depois junta os resultados, fazendo um merge, daí o nome.

Implementação

Inicialmente foram criados os métodos para criar os vetores. Para os vetores A e B foi utilizado as estruturas de repetição for para preenchimento dos vetores.

Já o vetor C que é preenchido de forma randômica através da classe random.

Após criar os métodos, os vetores são preenchidos e declarados nos métodos de ordenação para que sejam reordenados.

Foram criadas as classes que retornam os métodos de ordenação.

O merge sort é do tipo list<int> e recebe parâmetros também desse tipo, portanto ao chamar o método os vetores A, B e C deveriam ser também do tipo list<>.

Para calcular o tempo de cada método foi utilizado a classe StopWatch e seus métodos para permitir comparar o tempo de execução de cada método para cada vetor.

Análise

O programa retorna o tempo gasto para ordenar os 3 vetores criados por 5 métodos diferentes (Selection, Insertion, Bubble, Merge e Quick Sort).

Abaixo na Tabela 1, estão os tempos de cada método para ordenar um vetor de 10.000 posições. E em seguida um gráfico (Gráfico 1) para posteriores análises.

Tabela 1 – Dados extraídos do programa

Método	Vetor	Tempo (ms)
Selection	Vetor A	1593642
	Vetor B	1813689
	Vetor C	1813689
Insertion	Vetor A	3397
	Vetor B	688
	Vetor C	683
Bubble	Vetor A	3424855
	Vetor B	3321669
	Vetor C	3144001
Merge	Vetor A	251209
	Vetor B	189979
	Vetor C	185245
Quick	Vetor A	915827
	Vetor B	1048459
	Vetor C	977586

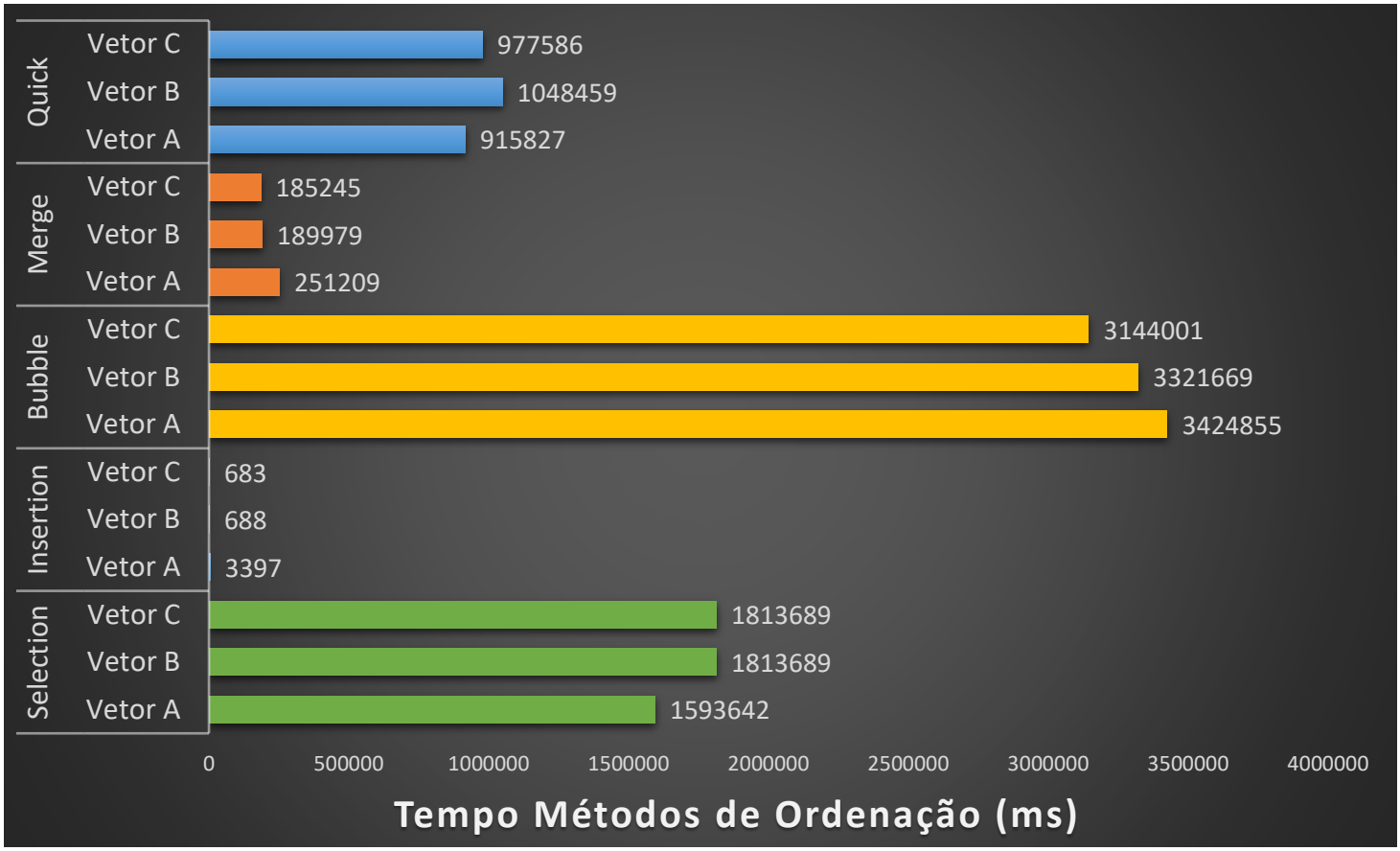

Gráfico 1 – Comparativo entre os métodos

Comparando o Vetor A(vetor crescente) nos 5 métodos comparados, o Insertion Sort foi o que se apresentou mais eficiente

Como observado o método mais eficiente no contexto demonstrado é o Insertion Sort. . Esse resultado se deve ao fato do Insertion Sort comparar os elementos de um vetor de forma global, identificando rapidamente que o vetor já está ordenado como deveria, não realizando permutações. Após o Insertion os métodos mais eficientes para ordenar os vetores propostos foram os métodos eficientes, Merge e Quick Sort, respectivamente. Como abordado na introdução são bons métodos em cenários de ordenação de valores altos.

O Merge Sort, como podemos observar, devido a recursividade ser sua principal ferramenta, seu melhor resultado vem ao lidar com estruturas lineares aleatórias.

O algoritmo Quick Sort, ao subdividir o vetor e fazer inserções diretas utilizando um valor de referência (pivô), reduz seu tempo de execução, mas, as quantidades de comparações (leitura) e, principalmente, trocas (escrita) ainda são muito altas.

O método Bubble Sort apresenta o pior desempenho, resultando em altos valores de comparações e trocas.

Conclusão

O presente estudo permite refletir que apesar da fácil implementação o método Bubble Sort não apresenta resultados satisfatórios, realizando um número alto de comparações, podendo em maquinas com poucos recursos computacionais, causar lentidão e esperas longas.

O Insertion Sort apresenta baixo número de comparações, e, o excessivo número de trocas, é a sua desvantagem. O Selection Sort realiza um número de trocas inferior ao número de comparações, consumindo, assim, mais tempo de leitura e menos de escrita. Os métodos Insertion e Selection, apesar de suas diferentes características, são muito usados em associação com outros algoritmos de ordenação, como os Merge Sort e Quick Sort possuem a característica de subdividir as listas a serem organizadas em listas menores, sendo mais eficientemente utilizados.

O Merge Sort é um algoritmo de ordenação mediano. Em listas pequenas ou já pré-ordenadas, a recursividade pode ser uma desvantagem consumindo tempo de processamento e realizando trocas desnecessárias.