Relatório

Trabalho Pratico (Ordenação)

Nome: Eric Almeida do Espirito Santo

RA: 318135918

Introdução

Neste trabalho foi abordado sobre como fazer os métodos de ordenação para que haja entendimento dos processos e de qual método é o mais viável para devido projeto.

Implementação

Na implementação foi utilizada a estrutura de vetores para a criação das ordens crescente, decrescente e randômica.

Para estabelecer o randômico foi utilizado o “random” do net framework (que após utilizar ele gera um número aleatório entre o número mínimo e máximo que o usuário colocar).

Para verificar o tempo de ordenação dos vetores foi utilizado o comando Stopwatch (que inicia uma contagem de tempo que o comando leva para ser executado) e o TimeSpan (que anota o tempo que o stopwatch contou).

Também criei um menu para que cada processo de ordenação rodasse individualmente (assim levando um tempo menor para gerar o tempo de cada vetor).

Analise

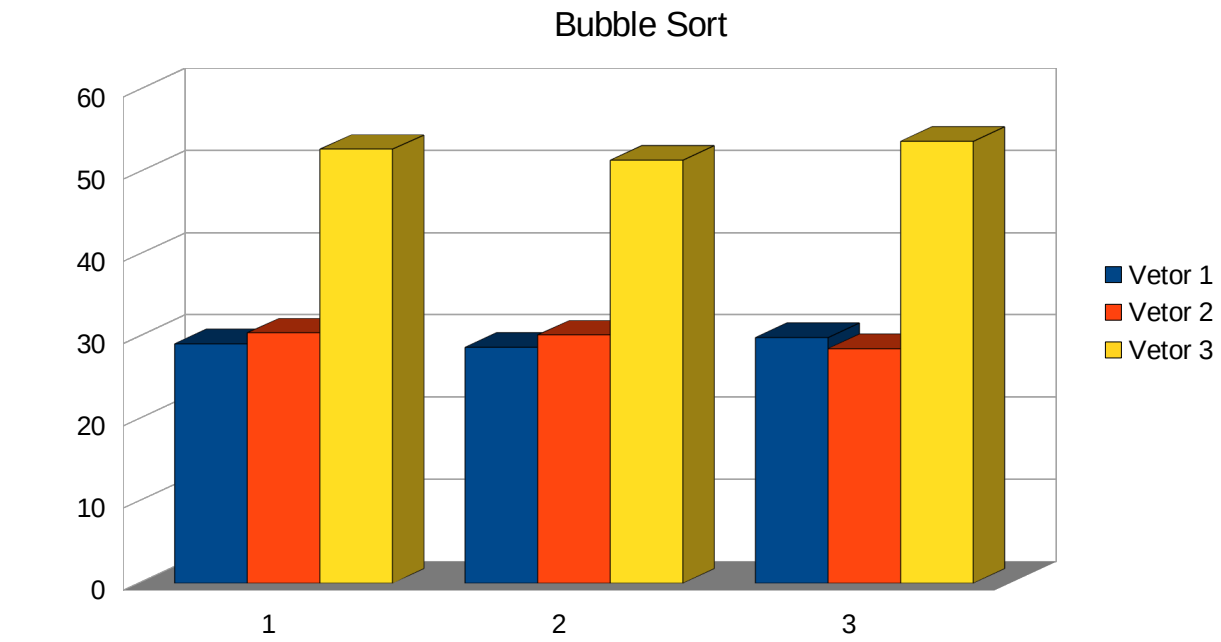
Bubble sort

O processo bubble sort é um algoritmo simples e não muito viável pois leva mais tempo para fazer a ordenação.

A ideia dele é percorrer o vetor várias vezes e alterar posição por posição com o próximo até o vetor ficar ordenado.

Bubble Sort	Vetor 1(crescente)	Vetor 2(decrescente)	Vetor 3(randômico)
Primeira execução	29,118	30,431	52,748
Segunda execução	28,688	30,158	51,415
Terceira execução	29,890	28,532	53,751

Tempo estabelecido em segundos.

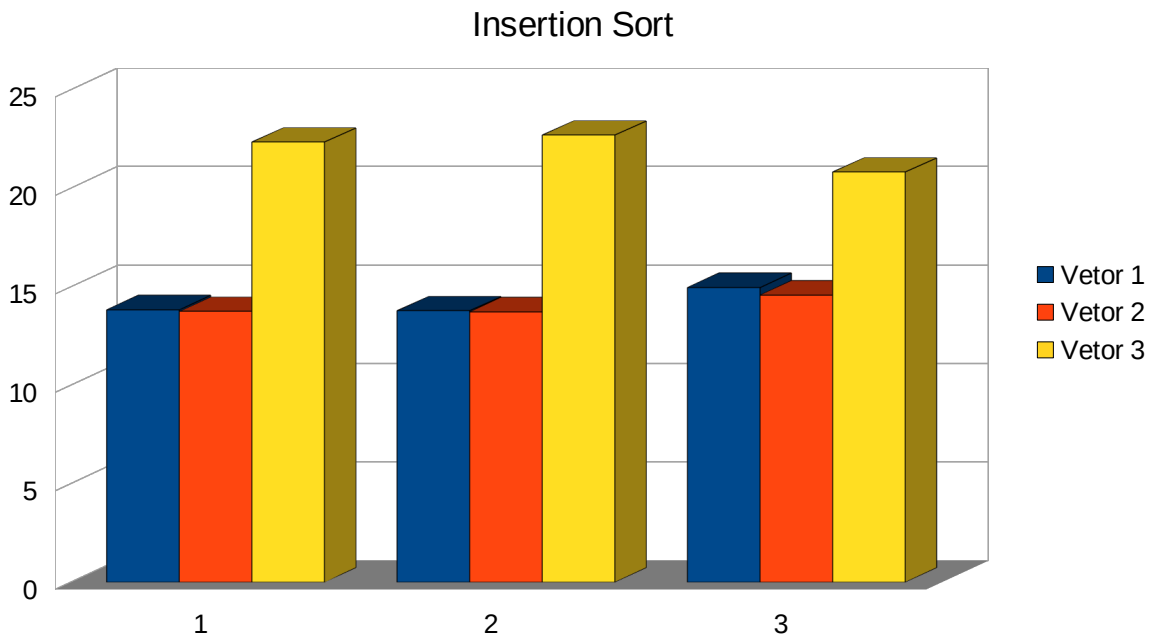


Insertion Sort

O insertion ele já é mais ágil que o bubble pois ele pega o a posição do vetor e confere com seus anteriores já colocando ele na sua posição correta em relação aos outros. Assim ele apenas precisa rodar pelo vetor uma vez.

Insertion Sort	Vetor 1(crescente)	Vetor 2(decrescente)	Vetor 3(randômico)
Primeira execução	13,832	13,740	22,335
Segunda execução	13,772	13,720	22,693
Terceira execução	14,969	14,571	20,811

Tempo estabelecido em segundos.



Selection Sort

o Selection Sort roda um pouco diferente pois ele acha direto o menor valor do vetor e coloca ele na primeira posição e vai fazendo isso seguidamente com os próximos números até o vetor esta ordenado. Mas pra isso ele tem que rodas várias vezes no vetor para achar o menor valor assim fazendo quando que ele ainda seja mais rápido que o bubble mas seja mais lento que o insertion.

Selection Sort	Vetor 1(crescente)	Vetor 2(decrescente)	Vetor 3(randômico)
Primeira execução	28,852	30,026	44,423
Segunda execução	30,194	29,056	45,656
Terceira execução	30,592	29,207	45,715

Tempo estabelecido em segundos.

Merge Sort

o merge sort separa o vetor em vários outros vetores até que ele tenha um vetor de apenas uma unidade ai ele chama outro método para ordenar e junta novamente esses vetores

(não consegui implementar um código funcional do merge sort)

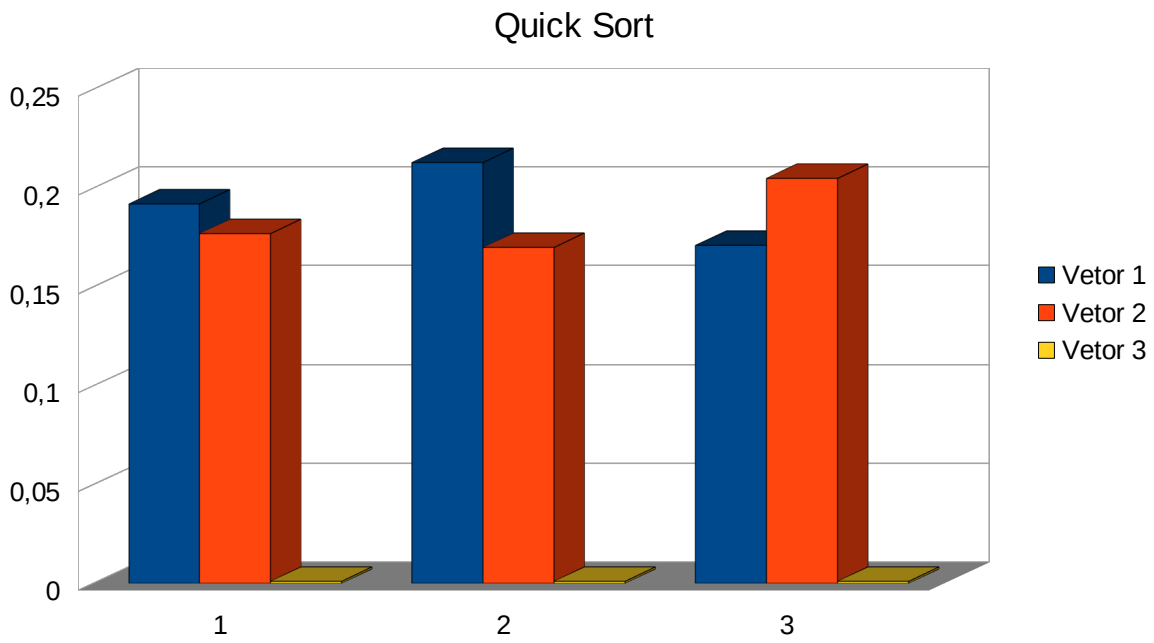
Quick Sort

O quick sort já tem usa ideia do próprio nome (ou seja ser rápido) para isso ele pega divide o vetor no meio e estabelece uma chave(pivô) e usa essa chave pra ordena a área. Assim fazendo com que o processo bem mais ágil para refazer. Esse método foi o mais ágil para fazer os vetores randômicos.

Quick Sort	Vetor 1(crescente)	Vetor 2(decrescente)	Vetor 3(randômico)
Primeira execução	0,192	0,177	0,001
Segunda execução	0,213	0,170	0,001
Terceira execução	0,171	0,205	0,001

Tempo estabelecido em segundos.

Obs: apesar do tempo pequeno do vetor 3 ao imprimir ele voltou totalmente ordenado



Conclusão

O trabalho foi interessante para aprender diferentes maneiras de ordenação existentes.

A maior dificuldade foi no merge sort pois mesmo implementando diferentes tipos de métodos dele eu não consegui entender como o código funcionava e o motivo qual ele não estava ordenando meus vetores.

Como resultado final foi visto que o quick sort é bem mais rápido do que os outros métodos de ordenação.