Trabalho Pesquisa E Ordenação De Dados
Métodos de Ordenação de Dados
Professora Andressa Sebben
Fabrício Romani

Bubble Sort (com flag)

O que pude notar é que o bubble sort é muito simples, mas é um método que custa muito. Seria adequado usar ele quando se tem registros pequenos, listas quase ordenadas, coisas básicas.

O número de operações não se altera se o vetor já estiver parcialmente ordenado, porém, isso pode ser resolvido com uma flag. No melhor caso, ele executa apenas N vezes (lista já ordenada).

Como funciona: o que o algoritmo de ordenação por bolha faz é comparar seus elementos em sucessão, troca os elementos de dois em dois, casos [0] for menor que [1] (sendo estes, posições da lista desordenada). A lista ordenada fica sempre a direita, porque os elementos maiores vão “flutuando” para a frente. O número de interações depende do tamanho da lista e da ordem, não são fixos como no insertion sort.

Ordem da lista	**10k – Comparações**	**10k – Tempo (ms)**	**10k - Trocas**
Aleatório	49.992.722,00	203.000000 ms	25.341.218,00
Crescente	9.999,00	000 ms	0,00
Decrescente	49.995.000,00	208.000000 ms	49.995.000,00

Ordem da lista	**50k – Comparações**	**50k – Tempo (ms)**	**50k - Trocas**
Aleatório	1.249.849.249,00	6404.000000 ms	624.460.895,00
Crescente	49.999,00	1.000000 ms	0,00
Decrescente	1.249.975.000,00	4109.000000 ms	1.249.975.000,00

Ordem da lista	**100k – Comparações**	**100k – Tempo (ms)**	**100k - Trocas**
Aleatório	704.982.473,00	27681.000000 ms	1.792.668.901,00
Crescente	99.999,00	1.000000 ms	0,00
Decrescente	704.982.704,00	27130.000000 ms	704.982.704,00

Insertion Sort

Funciona inserindo um elemento de cada vez, como o exemplo que a professora deu em aula no mês de março, ele pode ser visto como um jogo de baralho. Você pega um elemento por vez e compara com os que você já tem, assim você o “insere” já na posição certa.

Quando a lista é ordenada, ele não faz nenhuma troca, apenas insere os elementos. No pior caso, serão comparados todos os elementos entre si. É um método bom para ser utilizado quando o arquivo está quase ordenado. Ele é instável.

Como funciona: o método de ordenação insertion pode ser imaginado como um jogo de cartas, aonde se pega os elementos um por vez, compara com as outras cartas que já estão na sua mão (lista) e os “insere” no lugar já ordenado. É essa a ideia do insertion sort, você vai pegar os elementos um por vez e os comparar com cada um da sua lista. Nessa lógica, pode-se imaginar porque não ha trocar quando esta lista já vem ordenada, já que você não precisa fazer nenhuma mudança nas suas cartas que já estão na ordem certa (crescente).

Ordem da lista	**10k – Comparações**	**10k – Tempo (ms)**	**10k - Trocas**
Aleatório	9.999,00	136.000000 ms	25.341.218,00
Crescente	9.999,00	0.000000 ms	0,00
Decrescente	9.999,00	209.000000 ms	49.995.000,00

Ordem da lista	**50k – Comparações**	**50k – Tempo (ms)**	**50k - Trocas**
Aleatório	49.999,00	1785.000000 ms	624.460.895,00
Crescente	49.999,00	0 ms	0,00
Decrescente	49.999,00	3390.000000 ms	1.249.975.000,00

Ordem da lista	**100k – Comparações**	**100k – Tempo (ms)**	**100k - Trocas**
Aleatório	99.999,00	13698.000000 ms	1.792.668.901,00
Crescente	99.999,00	13699.000000 ms	0,00
Decrescente	99.999,00	18868.000000 ms	704.982.704,00

Selection Sort

Ele se baseia em passar o menor elemento desta lista para a [0] do vetor. Ele passa comparando e descobre qual é o menor elemento que a parte desordenada possui, quando faz isso, o troca de posição com o elemento [0] da lista desordenada, fazendo esse elemento se “transferir” para a parte ordenada. Compara a cada interação um elemento com outro, por isso não existe um melhor caso mesmo que o vetor já esteja em ordem, ou em qualquer situação. A complexidade sempre será n^2. Ocupa menos memória por não precisar de um vetor auxiliar. Não é recomendado se o vetor já estiver ordenado ou quase ordenado, já que, de qualquer forma fara sempre $(n^2-n)/2$ comparações.

Como funciona: No método de ordenação selection sort, é selecionado na lista desordenada o menor elemento, e ele é trocado com a posição [0] desta mesma lista, esse elemento então começa a fazer parte da lista ordenada que fica sempre a esquerda (como no bubble sort). Ele não é sensível a ordenação inicial, ou seja, não muda se a lista que você quer ordenar esta decrescente, aleatória ou se ela já veio ordenada. Na minha opinião, é por causa disso que para listar quase ordenadas ou ordenadas, este é o pior método. Pois ele vai realizar sempre comparações fixas. Não sei se teria um jeito, como é feito no bubble, de burlar essa característica.

Ordem da lista	**10k – Comparações**	**10k – Tempo (ms)**	**10k – Trocas**
Aleatório	50.005.000,00	156.000000 ms	74.431,00
Crescente	50.005.000,00	143.000000 ms	0,00
Decrescente	50.005.000,00	187.000000 ms	25.000.000,00

Ordem da lista	**50k – Comparações**	**50k – Tempo (ms)**	**50k - Trocas**
Aleatório	1.250.025.000,00	3726.000000 ms	423.611,00
Crescente	1.250.025.000,00	3841.000000 ms	0,00
Decrescente	1.250.025.000,00	3604.000000 ms	625.000.000,00

Ordem da lista	**100k – Comparações**	**100k – Tempo (ms)**	**100k - Trocas**
Aleatório	705.082.704,00	13690.000000	858.558,00
Crescente	705.082.704,00	13656.000000 ms	0,00
Decrescente	705.082.704,00	18645.000000 ms	1.794.967.296,00