Revisão P1 Pesquisa e Ordenação de Dados Giancarlo

Erickson G. Müller

September 25, 2024

1 Conteúdos

1. Complexidade de Algoritmos
2. Bubble Sort
3. Selection Sort
4. Insertion Sort
5. Merge Sort
6. Quick Sort
7. Heap Sort

2 Métodos de Ordenação

2.1 Ordenação Estável

Preserva a ordem relativa dos elementos que possuem o mesmo valor para a chave de ordenação. Composto por mais de uma chave.

2.2 Ordenação Não Estável

2.3 In place/In situ

Os valores são permutados dentro da própria estrutura do vetor, não havendo necessidade de duplicar a memória.

2.4 Ordenação Interna

O arquivo a ser ordenado cabe dentro da memória principal (RAM), qualquer registro pode ser acessado imediatamente.

2.5 Ordenação Externa

O arquivo a ser ordenado não cabe na memória principal(RAM), os registros são acessados sequencialmente ou em grandes blocos.

3 Bubble Sort

Compara pares de elementos adjacentes, dois de cada vez, colocando os dois em ordem. Se o elemento da esquerda é maior que o da direita, troca-os de posição. São realizadas até $n-1$ iterações, em que cada uma delas:

1. Percorre a lista a partir do início
2. Compara cada elemento com o seu sucessor
3. Troca os elementos caso o sucessor seja menor que o elemento comparado

O objetivo de cada iteração é levar o maior elemento para o final do vetor. Portanto, a parte ordenada fica à direita.
Complexidade: $O(n^2)$
Memória: *in place*(constante)
Estável.
Número de comparações: $\frac{n^2-n}{2}$

4 Selection Sort

Consiste em identificar o menor valor e trocar com o elemento de menor índice, e assim sucessivamente até que reste apenas o elemento de último índice. Durante o processo, dividir a lista em parte ordenada (à esquerda) e parte não ordenada (à direita).
São realizadas $n-1$ iterações, em que cada uma delas:

1. Seleciona o menor elemento da parte não ordenada.
2. Troca-o com o elemento de menor posição da parte não ordenada.
3. Incorpora esse elemento na parte ordenada.

Complexidade: $O(n^2)$ em **todos** os casos.
Memória: *in place*(constante).
Não é estável.
Recomendado para registros muito grandes, onde o custo da movimentação supera o custo das comparações (nessa ordenação, são feitas no máximo n trocas).

5 Insertion Sort

Divide-se o vetor em parte ordenada (à esquerda) e parte não ordenada (à direita). A parte não ordenada funciona como uma fila para entrar na parte ordenada, o primeiro elemento da parte ordenada é removido e posicionado no local certo da parte ordenada, movendo todos os elementos que forem maiores que ele uma casa para a direita.
São realizadas até $n-1$ iterações, em que cada uma delas:

1. Pega o primeiro elemento da parte não ordenada.
2. Verifica qual seria sua posição de inserção na parte ordenada.
3. Desloca os elementos maiores para a direita até encontrar a posição.
4. P é inserido à esquerda do último elemento movido.

Complexidade: $O(n^2)$ quando a lista for inversamente ordenada, pois cada iteração requer que todos os elementos sejam deslocados.
Memória: *in place*.
Estável.
Número de comparações: $\frac{n^2-n}{2}$ (pior caso) ou $n-1$ (melhor caso).
Eficiente para listas quase ordenadas, ou quando for necessário inserir um elemento numa lista já ordenada.
Com a complexidade $O(n^2)$, o algoritmo é muito mais lento que o selection sort, pois executa o mesmo número de comparações, mas com mais trocas. Com a complexidade $O(n)$, o algoritmo é muito mais rápido que o selection sort.

6 Merge Sort

Dividir em elementos ordenados e depois intercalar na ordem correta

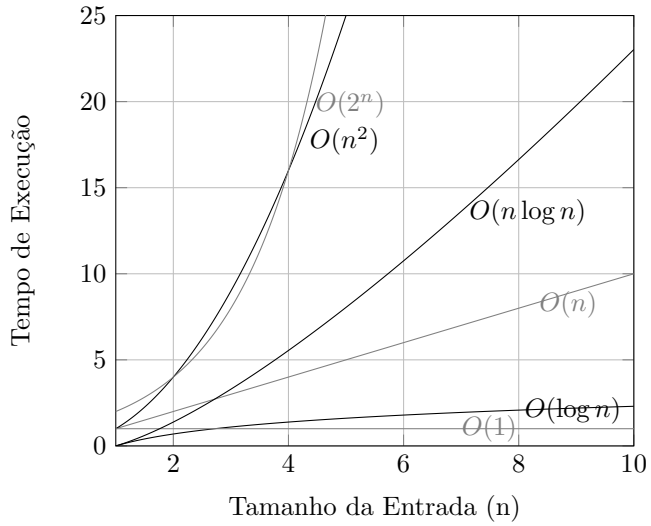
7 Quick Sort

Não precisa de memória extra(in-place).
Em tese $n \log n$
Pior caso = n^2(Quando já está ordenado).
Algoritmo de **Ordenação Instável**
Divisão e consquista.
i = posição que estou fazendo a comparação
k = se $i > pivo \rightarrow$ k fica parado e i vai para o próximo elemento
se $pivo > i \rightarrow$ elemento k troca com i, k e i vão para o próximo elemento
Quando i chega na posição do pivô $\rightarrow$ trocar o k pelo pivô

7.1 Regras

Se o Elemento for maior que o pivô : i anda, k fica parado
Se o Elemento for menor que o pivô: i anda, k anda
Se o Elemento for menor que a posição do k: troca elemento com o k, k anda
Passos:

1. Escolher o pivô (tradicionalmente o último elemento);
2. Particionamento (posicionar em relação ao pivô)



8 Counting Sort

Só serve para ordenação de inteiros positivos, Complexidade N. Ruim para economizar memória (questão do Count = K+1). Ver counting.md

9 Radix Sort

O counting do radix é de apenas 10 posições. Conta-se os dígitos do numeral.