Algoritmos de Ordenação e Busca e sua Análise de Complexidade

Alneu de Andrade Lopes
Rosane Minghim

Conteúdo

- Introdução
- Busca seqüencial
- Busca binária
- Notação O($f(n)$)
- Hierarquia de funções
- Selection sort
- Insertion sort
- Quick sort

Busca Seqüencial

- Chaves → Campos usados para identificação de um registro no arquivo.
- Busca através de exames das chaves. Se um arquivo tem n registro, com *Ki* como o valor da chave do registro *Ri*, a recuperação é feita via exame das chaves *Kn*,...*k1*.

```
Procedure Seqsearch (F,n,i,K)
Ki = K
K0 ← K
i ← n
while Ki ≠K do
        i ← i-1
end
end Seqseach
```

- Pior caso → n + 1 comparações
- Busca com êxito
- Número de comparações em média

$$\sum_{1 \le i \le n} (n-1+1)/n = (n+1)/2$$

N ↑ eficiência ↓

Busca Binária

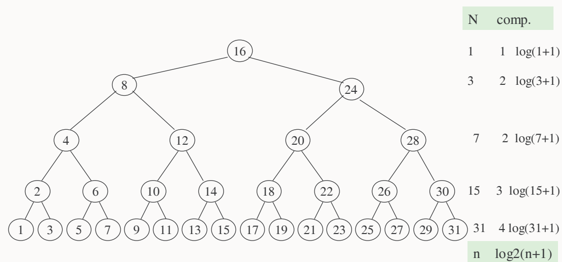
- Arquivo ordenado seqüencialmente.
- Busca começa examinando o registro do meio do arquivo, ao invés de um por um, iniciando em uma das extremidades, como na busca seqüencial.
- Supondo que o arquivo está ordenada por valores de chave crescente, dependendo do resultado da comparação com a chave do meio:
  - K < Km, registro procurado encontra-se na metade do arquivo com chaves de valores mais baixos;
  - K = Km, registro encontrado;
  - K > Km, registro procurado encontra-se na metade do arquivo com chaves de valores mais altos.

A cada comparação ou a busca termina ou o tamanho do arquivo a ser pesquisado é reduzido a metade do tamanho original

Comp.	1	2	3	...	n
Tam.	n/2	n/4	n/8	...	n/2^k

Estratégia: Dividir para conquistar!!

Busca Binária



Considere que o valor de um módulo é o índice da chave a ser testada.
N=31 registros; 1a chave a ser testada é K16, pois ⌊(1+31)/2⌋=16.

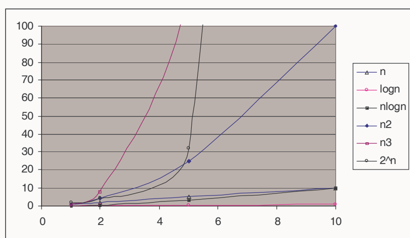
Notação

- Para capturar o conceito de uma função ser proporcional a outra quando ela cresce introduzimos uma nova terminologia e notação.
- Ex. a função 0.01n² + 10n é dita ser ***da ordem*** da função n², pois quando ela cresce torna-se mais proximamente proporcional a n².
- De forma precisa
  - $f(n)$ é da ordem $g(n)$ ou $f(n)$ é **O(*g*(*n*))** se existem inteiros positivos a e b tal que $f(n) \leq a * g(n)$, para todo $n \geq b$
  - Ex. $f(n) = n^2 + 100n$, $f(n)$ é $O(n^2)$ => a = 2, b = 100

Busca Seqüencial => O(n)

Busca Binária => O(log2(n))

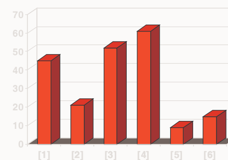
Hierarquia de funções

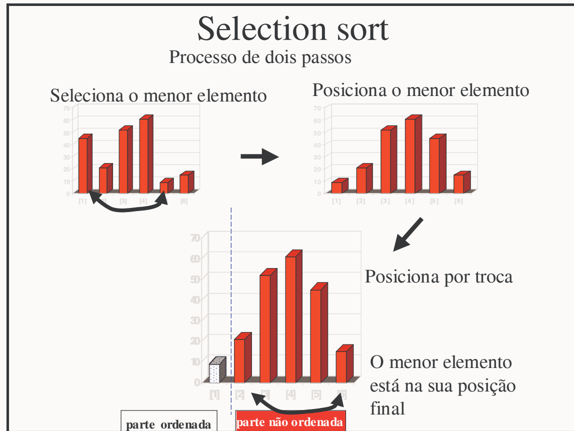


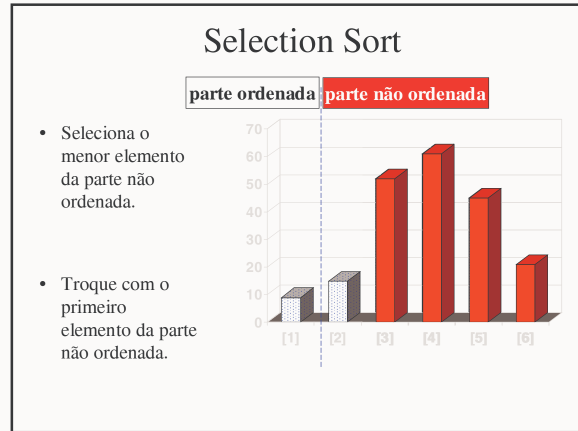
Classificação

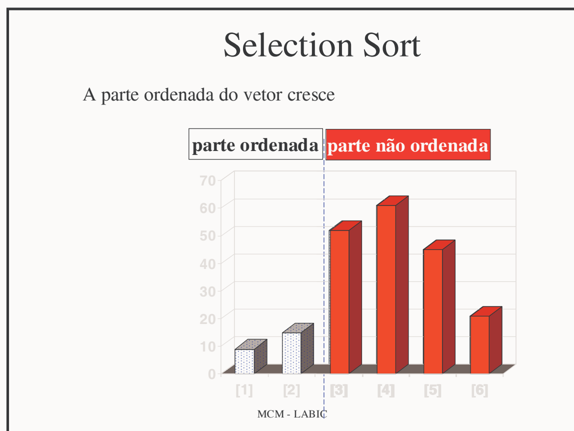
- Dois algoritmos lentos, porém simples, são **Selectionsort** e **Insertionsort**.

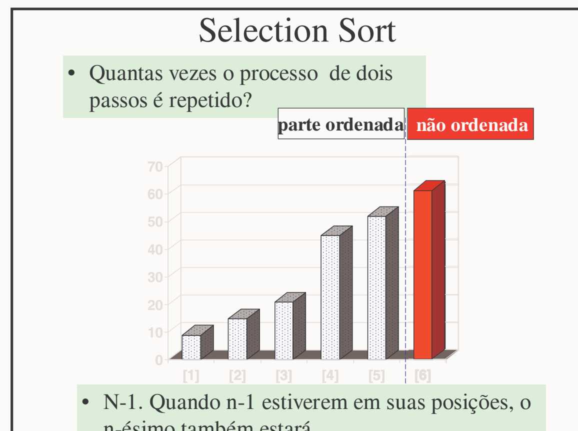
- Esta figura mostra um vetor (array) de seis números inteiros que nós queremos ordenar em ordem crescente.




Selection sort
Processo de dois passos
Seleciona o menor elemento
Posiciona o menor elemento
Posiciona por troca
O menor elemento
está na sua posição
final
parte ordenada
parte não ordenada


Selection Sort
parte ordenada
parte não ordenada
• Seleciona o menor elemento da parte não ordenada.
• Troque com o primeiro elemento da parte não ordenada.
70
60
50
40
30
20
10
0
[1]
[2]
[3]
[4]
[5]
[6]


Selection Sort
A parte ordenada do vetor cresce
parte ordenada
parte não ordenada
70
60
50
40
30
20
10
0
[1]
[2]
[3]
[4]
[5]
[6]
MCM - LABIC


Selection Sort
• Quantas vezes o processo de dois passos é repetido?
parte ordenada
não ordenada
70
60
50
40
30
20
10
0
[1]
[2]
[3]
[4]
[5]
[6]
• N-1. Quando n-1 estiverem em suas posições, o n-ésimo também estará

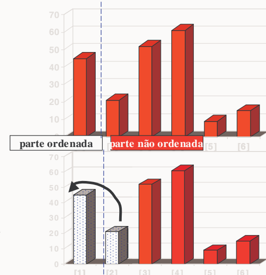
Insertion Sort

- Também enxerga o vetor como tendo uma parte ordenada e outra não ordenada.
- A parte ordenada inicia com o primeiro elemento do vetor desordenado.

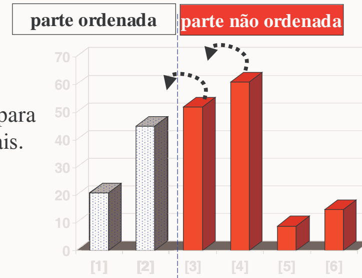
Processo de dois passos:

•Escolhe sempre o primeiro elemento da parte não ordenada

•Insere este elemento na parte ordenada



Insertion sort



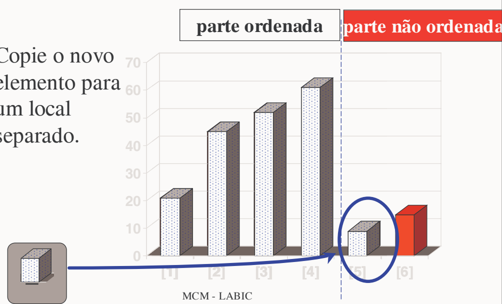
- Repete para os demais.

- Quantas vezes o processo de dois passos é repetido?

Inserindo um elemento

Inserção por deslocamento:

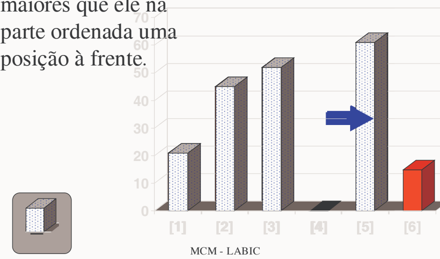
1. Copie o novo elemento para um local separado.



MCM - LABIC

Como inserir um elemento

2. Desloque os maiores que ele na parte ordenada uma posição à frente.

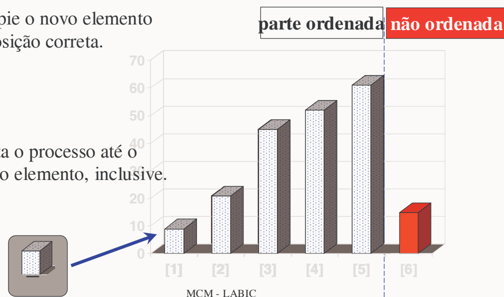


MCM - LABIC

Como inserir um elemento

3.Copie o novo elemento na posição correta.

Repita o processo até o último elemento, inclusive.



Complexidade

- Desenvolver os algoritmos e contar o número de comparações e deslocamentos.
- Complexidade $O(n^2)$. Tanto o Selectionsort quanto o Insertionsort tem um tempo para o pior caso na ordem de n^2 (vetor ordenado decrescentemente), sendo impraticáveis para grandes vetores.
- São fáceis de programar e corrigir.
- Insertionsort tem uma boa performance quando o vetor está parcialmente ordenado ou n pequeno.
- Porém, Algoritmos mais sofisticados são necessários para obtermos boa performance em qualquer caso, para grandes vetores.

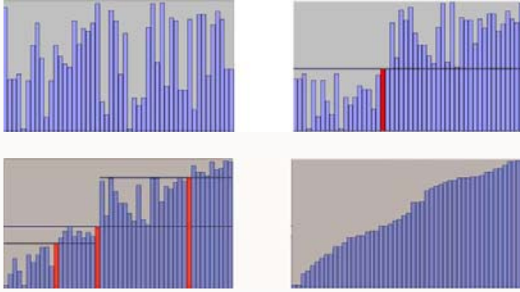
Quick Sort

- **Quicksort** é um algoritmo de ordenação recursivo criado por Tony Hoare.
- Bom tempo médio de execução.
- Pouco requisitos de espaço.
- Baseia-se no particionamento do vetor em dois, através da adoção de um *pivot* entre os dois lado, e em duas chamadas recursivas para ordenar cada uma das partições.

Passos

1. Partição
   - Escolha um elemento em v[l..r], chamado de elemento **pivot**.
   - Redistribua os elementos de forma que, a partir de duas posições p1 e p2, tq. v[i]>= v[p2], qq. i >= p2 E v[i]<= v[p1], qq. i <= p1.
2. Repita o processo recursivamente para v[l..p1] e para v[p2..r].

Ilustração



Fonte das figuras: wikipedia – “quicksort”

Código

```
void quicksort (int[] a, int lo, int hi)
{
    int i=lo, j=hi, h;
    int x=a[(lo+hi)/2];

    //  partition
    do
    {
        while (a[i]<x) i++;
        while (a[j]>x) j--;
        if (i<=j)
        {
            h=a[i]; a[i]=a[j]; a[j]=h;
            i++; j--;
        }
    } while (i<=j);

    //  recursion
    if (lo<j) quicksort(a, lo, j);
    if (i<hi) quicksort(a, i, hi);
}
```

Fonte:

http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/quick/quicken.htm

Análise da complexidade

No caso geral, cada vez que o pivot for posicionado, o sub-arquivo à esquerda é aproximadamente igual ao da direita. Isso nos deixa com 2 arquivos de tamanho aproximado de n/2. O tempo para posicionar o pivot é O(n); se T(n) for o tempo necessário para classificar um arquivo de n registros, então quando o arquivo se divide em dois, cada vez que o pivot for posicionado, temos:

$$T(n) \le cn + 2T(n/2), \text{ para alguma constante } c$$
$$\le cn + 2(cn/2 + 2T(n/4))$$
$$\le 2cn + 4T(n/4)$$
$$\ldots$$
$$\le cn\log_2 n + nT(1) = O(n\log_2 n)$$

Complexidade

- O pior caso do Quicksort é $O(n^2)$. Quando?
- Mas seu tempo médio é muito melhor: O(n log n).
- Para evitar o pior caso procure selecionar um bom pivot, desta forma utilize sempre o elemento médio entre três candidatos.

Resumo

- Busca seqüencial => $O(n)$
- Busca binária => $O(log(n))$
- Selection sort => $O(n^2)$
- Insertion sort => $O(n^2)$
- Quick sort (médio) => $O(nlog(n))$
- Quick sort (pior caso) => $O(n^2)$

Questão: Suponha que você desenvolveu alguns algoritmos que tenham as complexidades acima, considere que você está usando um PC típico, até quantos registros você se arriscaria usar em testes de cada um de seus algoritmos?