ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Algoritmos clássicos de ordenação I

Ordenação

Ordenação é largamente utilizada

- **Listas telefônicas e dicionários**
- **Grandes sistemas de BD e processamento de dados**
- **Etc.**

Algoritmos de ordenação são ilustrativos

- **Como resolver problemas computacionais**
- **Como lidar com estruturas de dados**
- **Como desenvolver algoritmos elegantes e como analisar e comparar seus desempenhos**

Ordenação

Definição: organizar uma sequência de elementos de modo que os mesmos estabeleçam alguma relação de ordem.

Diz-se que os elementos k_1, ..., k_n estarão dispostos de modo que $k_1 \leq k_2 \leq ... \leq k_n$.

Ordenação

Ocasionalmente, dá menos trabalho buscar um elemento em um conjunto desordenado do que ordenar primeiro e depois buscar.

Por outro lado, se a busca for uma operação frequente, vale a pena ordenar (pode ser feita apenas uma vez).

Depende das circunstâncias!

Terminologia

Ordenação de registros, em que cada registro é ordenado por sua chave.

Ordenação interna vs. externa:

- **Interna: se todos os registros cabem na memória principal.**
- **Externa: se os dados não cabem na memória principal, precisando ser armazenados em disco.**

Ordenação estável: ordem original de registros com chaves iguais é preservada após a ordenação.

Bubble sort

Um dos métodos mais conhecidos e intuitivos.

Ideia básica:

- **Percorrer o conjunto várias vezes.**
- **A cada iteração, comparar cada elemento com seu sucessor (v[i] com v[i+1]) e trocá-los de lugar caso estejam na ordem incorreta.**

Bubble sort: passo 1

- Vetor inicial
  - X = (25 , 57 , 48 , 37 , 12 , 92 , 86 , 33)
- X[0] com X[1] (25 com 57) não ocorre permutação
  - X = (25 , 57 , 48 , 37 , 12 , 92 , 86 , 33)
- X[1] com X[2] (57 com 48) ocorre permutação
  - X = (25 , 48 , 57 , 37 , 12 , 92 , 86 , 33)
- X[2] com X[3] (57 com 37) ocorre permutação
  - X = (25 , 48 , 37 , 57 , 12 , 92 , 86 , 33)
- X[3] com X[4] (57 com 12) ocorre permutação
  - X = (25 , 48 , 37 , 12 , 57 , 92 , 86 , 33)
- X[4] com X[5] (57 com 92) não ocorre permutação
  - X = (25 , 48 , 37 , 12 , 57 , 92 , 86 , 33)
- X[5] com X[6] (92 com 86) ocorre permutação
  - X = (25 , 48 , 37 , 12 , 57 , 86 , 92 , 33)
- X[6] com X[7] (92 com 33) ocorre permutação
  - X = (25 , 48 , 37 , 12 , 57 , 86 , 33 , 92)

Bubble sort: passo 2

- Vetor após passo 1
  - X = (25 , 48 , 37 , 12 , 57 , 86 , 33 , 92)
- X[0] com X[1] (25 com 48) não ocorre permutação
  - X = (25 , 48 , 37 , 12 , 57 , 86 , 33 , 92)
- X[1] com X[2] (48 com 37) ocorre permutação
  - X = (25 , 37 , 48 , 12 , 57 , 86 , 33 , 92)
- X[2] com X[3] (48 com 12) ocorre permutação
  - X = (25 , 37 , 12 , 48 , 57 , 86 , 33 , 92)
- X[3] com X[4] (48 com 57) não ocorre permutação
  - X = (25 , 37 , 12 , 48 , 57 , 86 , 33 , 92)
- X[4] com X[5] (57 com 86) não ocorre permutação
  - X = (25 , 37 , 12 , 48 , 57 , 86 , 33 , 92)
- X[5] com X[6] (86 com 33) ocorre permutação
  - X = (25 , 37 , 12 , 48 , 57 , 33 , 86 , 92)
- X[6] com X[7] (86 com 92) não ocorre permutação
  - X = (25 , 37 , 12 , 48 , 57 , 33 , 86 , 92)

Bubble sort

Depois do primeiro passo

- Vetor: (24, 48, 37, 12, 57, 86, 33, 92)

- O maior elemento (92) está na posição correta

Para um vetor de n elementos, são necessárias n-1 iterações.

A cada iteração, os elementos vão assumindo suas posições corretas.

Bubble sort

Implementação em Python

```
def bubble_sort(v):
    for i in range(len(v)-1):
        for j in range(len(v)-i-1):
            if(v[j] > v[j+1]):
                v[j], v[j+1] = v[j+1], v[j]
```

Insertion sort

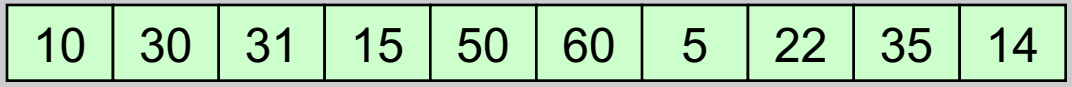
Ideia básica:

- **Ordenar o conjunto inserindo os elementos em um subconjunto já ordenado.**
- **No i-ésimo passo, inserir o i-ésimo elemento na posição correta entre x[0], ..., x[i-1], que já estão em ordem.**
  - **Realocar elementos**

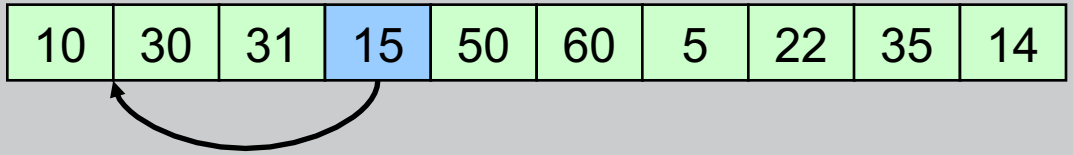
Insertion sort

Exemplo

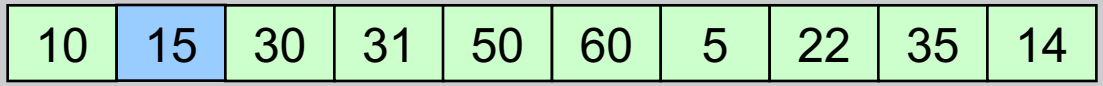
Vetor original



Realocando o elemento 15



30 e 31 são realocados e 15 é inserido



Insertion sort: exemplo

- X = (44 , 55 , 12 , 42 , 94 , 18 , 06 , 67)
- passo 1 (55) 44 55 12 42 94 18 06 67
- passo 2 (12) 12 44 55 42 94 18 06 67
- passo 3 (42) 12 42 44 55 94 18 06 67
- passo 4 (94) 12 42 44 55 94 18 06 67
- passo 5 (18) 12 18 42 44 55 94 06 67
- passo 6 (06) 06 12 18 42 44 55 94 67
- passo 7 (67) 06 12 18 42 44 55 67 94

Insertion sort

Implementação em Python

```
def insertion_sort(v):
    for i in range(1, len(v)):
        x = v[i]
        j = i-1
        while j >= 0 and x < v[j]:
            v[j+1] = v[j]
            j -= 1
        v[j+1] = x
```

ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Algoritmos clássicos de ordenação I