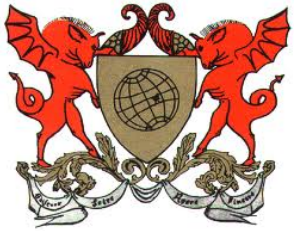
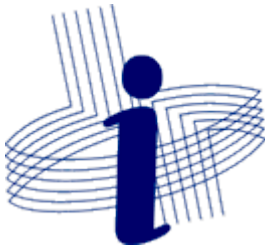


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF 112
Programação 2
Aula “7”
Ordenação

Ordenação

- Ordenação: dado um conjunto de elementos, ordenar tais elementos significa reorganizá-los de forma ascendente ou descendente de acordo com algum critério.

- Em geral, deseja-se ordenar números, caracteres, strings e, mais importante, registros contendo vários campos. No caso dos registros, um campo deve ser escolhido como “chave” na ordenação.

- Formalmente, dada uma sequencia s_1, s_2, .., s_k de chaves, o objetivo é encontrar uma permutação i_1, i_2, ... , i_k dos índices 1,2 .. k, de modo que $s_{i1} \leq s_{i2} \leq ... \leq s_{ik}$

- Em geral, o objetivo da ordenação é facilitar a recuperação destes elementos. Além disso, a ordenação é muitas vezes utilizadas como método auxiliar em vários algoritmos importantes.

Ordenação

- Existem vários algoritmos de ordenação.

- Um algoritmo pode ser melhor do que outro mas não existe um algoritmo de ordenação que seja “o melhor de todos” em todas as situações: alguns são simples de implementar, mas lentos; outros são rápidos, mas complexos; outros são rápidos, mas usam muita memória, etc...

- Assim, o conhecimento sobre os vários algoritmos existentes é importante para um profissional da área de computação.

Principais características dos métodos

- Eficiência de CPU: normalmente se mede a eficiência do método com base no número de operações de comparação realizadas (porque?).

- Outra medida muito utilizada é o número de movimentos realizados. Em geral, avalia-se o número de operações de leitura e escrita dos dados (exemplo de utilidade disso: SSD, flash).

- Uso de memória: algoritmos que fazem ordenação “in-place” são os que não precisam de espaço extra na memória para serem executados.

Principais características dos métodos

- **Ordenação estável:** se dois elementos possuem chaves “iguais”, a ordem relativa entre eles após a ordenação será mantida. Isso pode ser útil, por exemplo, ao ordenar “estudantes”: podemos ordená-los por ordem alfabética e, então, ordená-los por CRA. Após as duas ordenações, os estudantes estarão ordenados por CRA mas os que tiverem CRAs iguais serão apresentados em ordem alfabética.

Principais características dos métodos

- **Ordenação interna:** quando os elementos estão armazenados na memória principal do computador. A maior parte dos algoritmos é para memória interna.

Principais características dos métodos

- **Ordenação externa:** quando os elementos estão na memória secundária (disco, fita , etc).

Ordenação direta/indireta

- Algumas vezes, a ordenação é feita de forma “indireta”. Isso é útil, principalmente, quando os dados a serem ordenados são registros muito grandes que possuem chaves de comparação “rápida”.
- A ordenação indireta é feita ordenando-se um conjunto de índices/ponteiros para os elementos. Assim, a posição dos registros a serem ordenados não é alterada (o que se altera é a posição dos ponteiros).
- Ex:

1	2	3	4	5	6	7
G	**P**	**D**	**A**	**V**	**R**	**F**

Esses são os dados

Vetor original

1	2	3	4	5	6	7
A	**D**	**F**	**G**	**P**	**R**	**V**

Vetor ordenado (de forma direta)

1	2	3	4	5	6	7
4	**3**	**7**	**1**	**2**	**6**	**5**

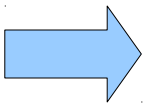
Vetor ordenado (de forma indireta)

Ordenação direta/indireta

- Por simplicidade, vamos supor que temos um arranjo de inteiros e desejamos ordenar tal arranjo de forma direta.

1	2	3	4	5	6	7
4	**5**	**2**	**1**	**7**	**5**	**3**

Vetor original



1	2	3	4	5	6	7
1	**2**	**3**	**4**	**5**	**6**	**7**

Vetor ordenado (de forma direta)

Estamos trocando os inteiros
de lugar dentro do vetor

Selection Sort

- Ordenação por seleção

```
void SelectionSort(int *v, int n) {
  for (int i=0; i<n-1; i++) {
     // acha a posicao do menor elemento
     // entre as posições (i) e (n-1)
     int posMenor = i;
     for (int j=i+1; j<n; j++)
       if (v[j] < v[posMenor])
         posMenor = j;

     // troca o menor elemento (que está na
     // posicao posMenor) com o elemento (i)
     int aux = v[i];
     v[i] = v[posMenor];
     v[posMenor] = aux;
   }
}
```

Selection Sort

- Ordenação por seleção

  1) Exercícios: rastreie a ordenação do arranjo: [5,7,4,2,1]

  2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

  3) O algoritmo *Selection Sort* é estável? justifique.

  4) Ao ordernar um arranjo de tamanho *n*, quantos movimentos de dados o algoritmo de seleção realiza?

  5) O algoritmo da seleção realiza ordenação "*in-place*"?

Selection Sort

- Ordenação por seleção
- Exercícios: rastreie a ordenação do arranjo: [5,7,4,2,1]

  vetor = 5 - 7 - 4 - 2 - 1
- Para o primeiro for o índice inicial é 0. O for mais interno começa do índice 1 (índice_inicial_externo + 1) e percorre o vetor até encontrar o menor elemento, que, no caso, é o 1. O 1 passa para a posição inicial do vetor que na primeira iteração do for é 0.

  vetor = 1 - 7 - 4 - 2 - 5
- Ao fim do for interno o for externo incrementa uma unidade, agora a posição inicial a ser comparada é a posição 1. Agora repetimos o processo, buscando o segundo menor elemento, neste caso o 2.

  vetor = 1 - 2 - 4 - 7 – 5

  vetor = 1 - 2 - 4 - 7 – 5

  vetor = 1 - 2 - 4 - 7 – 5

  vetor = 1 - 2 - 4 - 5 – 7

Selection Sort

- Ordenação por seleção

  2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

Selection Sort

- Ordenação por seleção

  2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

  n(n-1)/2

Selection Sort

- Ordenação por seleção

  3) O algoritmo Selection Sort é estável?

  - Estável: se dois elementos possuem chaves “iguais”, a ordem relativa entre eles após a ordenação será mantida.

Selection Sort

- Ordenação por seleção

  4) Ao ordernar um arranjo de tamanho *n*, quantos movimentos de dados o algoritmo de seleção realiza?

Selection Sort

- Ordenação por seleção

  4) Ao ordernar um arranjo de tamanho *n*, quantos movimentos de dados o algoritmo de seleção realiza?

  Aproximadamente n

Selection Sort

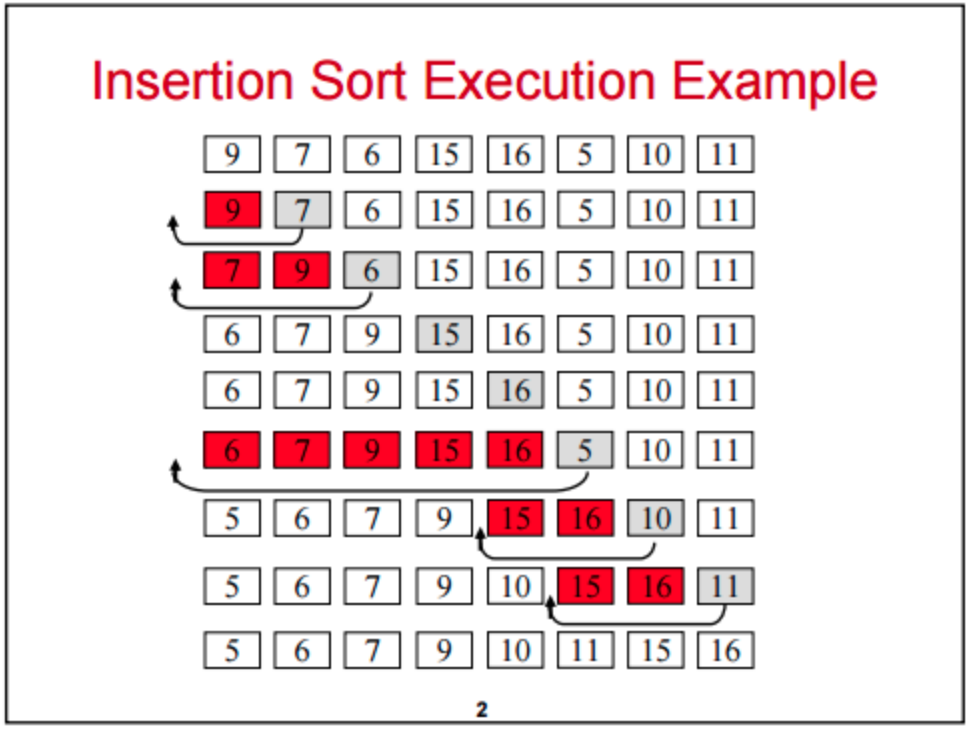
- Ordenação por seleção

  5) O algoritmo da seleção realiza ordenação "*in-place*"?

  Sim!

Ordenação por inserção

Ordenação por inserção



Adaptado de: http://www.geeksforgeeks.org/insertion-sort/

Ordenação por inserção

- Imagine que temos um arranjo de tamanho *n*, onde os *k* primeiros elementos já estão ordenados.
- A ideia do algoritmo de inserção é aumentar o número de elementos já ordenados inserindo (de 1 em 1) elementos na parte já ordenada do arranjo.
- Inicialmente, o conjunto de elementos ordenados é vazio.
- Muito utilizado por pessoas na “vida real”. Por exemplo, para ordenar cartas de baralho, é comum manter um conjunto de cartas ordenado e, então, ir “inserindo” cartas repetidamente nesse conjunto (mantendo-o ordenado).

Ordenação por inserção

```
void insertionSort(int *v, int n) {
  for (int i=1; i<n; i++) {
     // o arranjo entre as posicoes [0,i) já está ordenado
     int elemInserir = v[i];
     int j = i-1;
     while( j>=0 && v[j] > elemInserir) {
                    v[j+1] = v[j];
                    j--;
     }
     v[j+1] = elemInserir;
    }
}
```

Ordenação por inserção

Exercícios

1) Rastreie a ordenação do arranjo: [5,7,4,2,1]

vetor = 5 - 7 - 4 - 2 – 1

vetor = 5 - 7 - 4 - 2 – 1

vetor = 4 - 5 - 7 - 2 – 1

vetor = 2 - 4 - 5 - 7 – 1

vetor = 1 - 2 - 4 - 5 - 7

Elemento a ser inserido.

Elemento que será deslocado.

Ordenação por inserção

2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

n*n

Ordenação por inserção

3) Em qual situação o algoritmo se comportaria (em termos de tempo) de forma pior?

Ordenação por inserção

3) Em qual situação o algoritmo se comportaria (em termos de tempo) de forma pior?

Vetor ordenado de maneira decrescente.

Ordenação por inserção

4) O algoritmo *Insertion Sort* é estável?

Ordenação por inserção

4) O algoritmo *Insertion Sort* é estável? Sim!

Ordenação por inserção

5) Cite vantagens/desvantagens em relação ao algoritmo de seleção.

Ordenação por inserção

5) Cite vantagens/desvantagens em relação ao algoritmo de seleção.

- Vantagem: Compara o elemento que deseja inserir apenas com os elementos já ordenados que são necessários.

  Se os elementos já estiverem ordenados, o selection sort ignora isso.

- Desvantagem: mover dados no vetor.

Ordenação por inserção

6) Ao ordernar um arranjo de tamanho *n*, quantos movimentos de dados o algoritmo de inserção realiza?

Ordenação por inserção

6) Ao ordernar um arranjo de tamanho *n*, quantos movimentos de dados o algoritmo de inserção realiza?

O(n*n)

Ordenação por inserção

7) O algoritmo da inserção realiza ordenação "*in-place*"?

Ordenação por inserção

7) O algoritmo da inserção realiza ordenação "*in-place*"?

Sim!

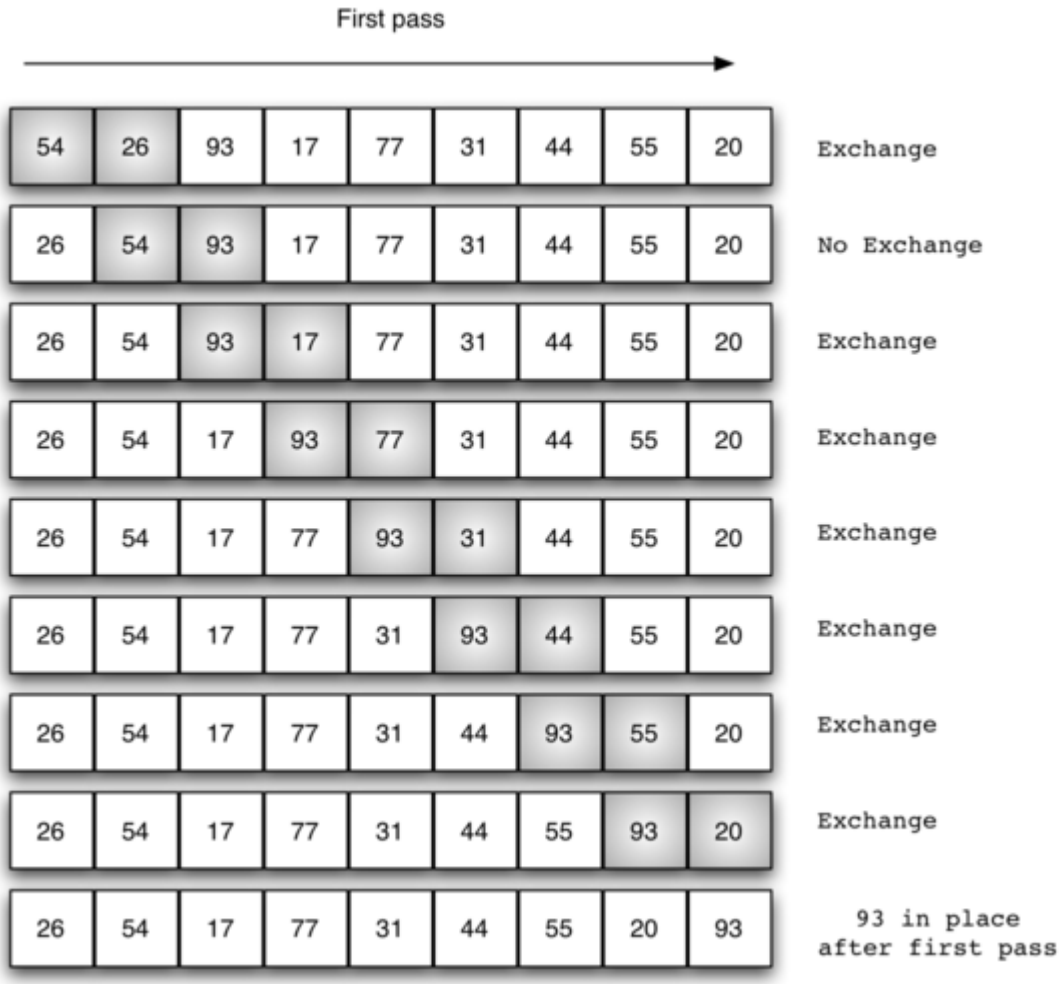
Ordenação por inserção

- Ordenação por inserção
- Considerações:
  - O algoritmo é bastante eficiente para pequenas quantidades de dados.
  - Dos métodos mais simples, em geral, é o mais eficiente na prática.
  - É eficiente caso o arranjo já esteja ordenado (ou parcialmente ordenado).
  - Pode realizar bem mais movimentos de dados do que o método da seleção.

Bolha

- Algoritmo da “bolha”: algoritmo de implementação bastante simples.
- Ideia: o algoritmo varre o vetor repetidas vezes verificando os pares de elementos adjacentes. Se um par estiver fora de ordem, ele é trocado.
- Na primeira varredura do array, o maior elemento do array “sobe até a última posição”. Na segunda varredura, o segundo maior elemento sobe até a penúltima posição....
- Esse processo lembra “bolhas” subindo em um líquido até a superfície.

Bolha



Adaptado de: http://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html

Bolha

```
void bubbleSort(int *v, int n) {
  for(int i=0;i<n-1;i++)
        for(int j=0;j<n-1-i;j++)
              if (v[j] > v[j+1]) {
                    swap(v[j],v[j+1]);
                    /*int aux = v[j];
                    v[j] = v[j+1];
                    v[j+1] = aux;*/
              }
}
```

Bolha

- Exercícios:

  1) Rastreie a ordenação do arranjo: [5,7,4,2,1]

  2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

  3) Em qual situação o algoritmo se comportaria (em termos de tempo) de forma pior?

  4) O algoritmo *Bubble Sort* é estável? justifique.

  5) O algoritmo da bolha realiza ordenação "*in-place*"?

  6) Prove que se após varrer o arranjo o algoritmo não realizar nenhuma troca, então o arranjo já está ordenado. Use isso para melhorar (ou melhor, deixar menos pior) a eficiência do algoritmo da bolha.

Bolha

- Exercícios:

  1) Rastreie a ordenação do arranjo: [5,7,4,2,1]

  vetor = 5 - 7 - 4 - 2 – 1

  vetor = 5 - 7 - 4 - 2 – 1

  vetor = 5 - 4 - 7 - 2 – 1

  vetor = 5 - 4 - 2 - 7 – 1

  vetor = 5 - 4 - 2 - 1 – 7

  vetor = 5 - 4 - 2 - 1 – 7

  vetor = 4 - 5 - 2 - 1 – 7

  vetor = 4 - 2 - 5 - 1 – 7

  vetor = 4 - 2 - 1 - 5 – 7

  ...

Bolha

2) Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?

Bolha

3) Em qual situação o algoritmo se comportaria (em termos de tempo) de forma pior?

Bolha

4) O algoritmo *Bubble Sort* é estável?

Bolha

5) O algoritmo da bolha realiza ordenação "*in-place*"?

Bolha

- Foram vistos vários métodos elementares.
- A maioria deles não é eficiente na prática (principalmente para entradas grandes).