Estrutura de Dados 1

Algoritmos Elementares de Ordenação

Prof. Lucas Boaventura
lucas.boaventura@unb.br

Algoritmos de Ordenação

- **Por que estudar ordenação?**
  - Ordenação como problema fundamental em computação
  - Busca eficiente
  - Organização de dados
  - Etapa base para outros algoritmos

Algoritmos de Ordenação

- Algoritmos de ordenação elementares são os algoritmos mais simples para ordenar um vetor.

- Ordenação é você rearranjar um vetor de modo que todos os elementos fiquem em ordem crescente.

0	1	2	3	4	5	6	7	8	9	10
111	999	222	999	333	888	444	777	555	666	555

111	222	333	444	555	555	666	777	888	999	999

Insertion Sort

- **Ideia básica do algoritmo**
  - Analogia: ordenar cartas na mão
  - Construção incremental da parte ordenada
  - A cada passo, um novo elemento é inserido na posição correta

Insertion Sort

- **Algoritmo:**

vetor = [

]

Insertion Sort

- **Algoritmo:** Pega a primeira não ordenada

vetor = [

]

Insertion Sort

- **Algoritmo:** Pega a segunda carta e comparamos se ela é menor que a anterior

vetor = [

]

Insertion Sort

- **Algoritmo:** Se for menor, você troca as cartas d
  eposição

vetor = []

Insertion Sort

- **Algoritmo:** Pega a terceira carta e compara com a anterior

vetor = []

Insertion Sort

- **Algoritmo:** Como 1 é menor que 13, a gente troca as cartas de posição entre si

vetor = [

]

Insertion Sort

- **Algoritmo:** Compara novamente, como 1 também é menor que 2 a gente também troca as posições das cartas

vetor = [

]

Insertion Sort

- **Algoritmo:** Pega a próxima carta, número 3, e compara ela com o 13.

vetor = []

Insertion Sort

- **Algoritmo:** Como 3 é menor que 13, troca de posição e compara o 3 com o 2.

vetor = [

]

Insertion Sort

- **Algoritmo:** Como 3 NÃO é menor que o 2, significa que ele está na posição correta.

vetor = [

Insertion Sort

- **Algoritmo:** Pega a próxima carta, número 8, e compara ela com o 13 e troca de posição.

vetor = []

Insertion Sort

- **Algoritmo:** Agora, compara o o 8 com o 3, e nesse caso não precisa mais trocar nada.

vetor = []

Insertion Sort

- **Algoritmo:** Por fim, pegamos a carta do 5 e comparamos com o 13, em seguida efetuamos a troca entre elas.

```
vetor = [                                                    ]
```

Insertion Sort

- **Algoritmo:** Como o 5 também é menor que 8, seguimos trocando a posição.

```
vetor = [                                                    ]
```

Insertion Sort

- **Algoritmo:** Agora sim, nosso vetor está ordenado seguindo a regra do algoritmo insertion sort.

```
vetor = [                                                    ]
```

Insertion Sort

- **Visão geral do algoritmo:**
  - Divide o vetor em duas partes: a parte a ordenada e a parte não ordenada.
  - Insere o próximo elemento vetor não ordenado na posição correta do vetor ordenado.

Insertion Sort - Código

```
1  void insertionsort (int v[], int n) {
2     for (int j = 1; j < n; ++j) {
3        int x = v[j];
4        int i;
5
6        for (i = j-1; i >= 0 && v[i] > x; --i)
7           v[i+1] = v[i];
8
9        v[i+1] = x;
10    }
11 }
```

Insertion Sort - Código

- **for (j = 1; j < n; j++)**
  - Controla o crescimento da parte ordenada do vetor

- **for (i = j-1; i >= 0 && v[i] > x; i--)**
  - Desloca os elementos maiores para a direita

```
1  void insertionsort (int v[], int n) {
2     for (int j = 1; j < n; ++j) {
3        int x = v[j];
4        int i;
5
6        for (i = j-1; i >= 0 && v[i] > x; --i)
7           v[i+1] = v[i];
8
9        v[i+1] = x;
10    }
11 }
```

Insertion Sort - Análise Pior Caso

- **Quantas comparações são feitas?**
  - v[i] > x

- **Soma das comparações:**
  - Total no pior caso:
    - $1 + 2 + 3 + \cdots + (n-1) = \mathbf{n * (n-1) / 2}$

- **Ordem de crescimento: $O(n^2)$**

Insertion Sort - Tempo

- **Crescimento quadrático**
  - Se tamanho = N, tempo = T
  - Se tamanho = 2N, tempo ≈ 4T
  - Se tamanho = 10N, tempo ≈ 100T

- Conclusão:
  - Algoritmo inviável para vetores grandes

Insertion Sort

- **Vantagens**
  - Simples de implementar
  - Bom desempenho para vetores pequenos
  - Excelente para dados quase ordenados

- **Desvantagens**:
  - Lento para grandes volumes de dados

Selection Sort

- **Ideia básica do algoritmo**
  - Encontrar o menor elemento do vetor
  - Colocá-lo na primeira posição
  - Repetir o processo para o restante do vetor

Selection Sort

- **Algoritmo:** A partir do seguinte vetor desordenado

vetor = [

]

Selection Sort

- **Algoritmo:** Na primeira iteração pega o **menor** valor

Selection Sort

- **Algoritmo:** Em seguida pega o **segundo menor**

Selection Sort

- **Algoritmo:** Pega o **terceiro menor**

Selection Sort

- **Algoritmo:** Pega o **quarto menor**

vetor = []

Selection Sort

- **Algoritmo:** Pega o **quinto menor**

vetor = []

Selection Sort

- **Algoritmo:** Pega o **quinto menor**

vetor = []

Selection Sort

- **Algoritmo:** Até finalizar todos elementos e você ter o vetor ordenado.

vetor = []

Selection Sort

- **Visão geral do algoritmo:**
  - Divide o vetor em duas partes: parte ordenada à esquerda e a parte não ordenada à direita.
  - A cada iteração:
    - Busca-se o menor elemento da parte não ordenada
    - Realiza-se uma troca

Selection Sort - Código

```
 1 void selectionsort(int v[], int n) {
 2     for (int i = 0; i < n-1; i++) {
 3         int min = i;
 4         for (int j = i+1; j < n; j++) {
 5             if (v[j] < v[min])
 6                 min = j;
 7         }
 8         int aux = v[i];
 9         v[i] = v[min];
10         v[min] = aux;
11     }
12 }
```

Selection Sort - Código

- **i:** delimita a parte ordenada do vetor
- **j:** percorre a parte não ordenada
- **min:** guarda o índice do menor elemento

```
1  void selectionsort(int v[], int n) {
2      for (int i = 0; i < n-1; i++) {
3          int min = i;
4          for (int j = i+1; j < n; j++) {
5              if (v[j] < v[min])
6                  min = j;
7          }
8          int aux = v[i];
9          v[i] = v[min];
10         v[min] = aux;
11     }
12 }
```

Selection Sort - Análise de Comparações

- **Quantas comparações são feitas?**
  - Para cada posição i:
    - O laço interno percorre n - i - 1 elementos

- **Soma das comparações:**
  - Total no pior caso:
    - $(n-1) + (n-2) + \cdots + 1 = n*(n-1) / 2$

- **Ordem de crescimento: $O(n^2)$**

Selection Sort - Tempo

- **Custo do Selection Sort:**
  - Melhor caso: **O(n²)**
  - Pior caso: **O(n²)**
  - Caso médio: **O(n²)**

- Diferente do Insertion Sort, não melhora com vetor quase ordenado

Selection Sort

- **Vantagens**
  - Simples de entender e implementar
  - Poucas trocas
  - Fácil análise

- **Desvantagens**:
  - Lento para vetores grandes
  - Não se adapta a vetores quase ordenados

Insertion Sort vs Selection Sort

CRITÉRIO	Insertion Sort	Selection Sort
Melhor Caso	O(n)	O(n2)
Trocas	Muitas	Poucas
Vetor quase ordenado	Muito bom	Ruim
Simplicidade	Alta	Alta

Dúvidas?

- lucas.boaventura@unb.br