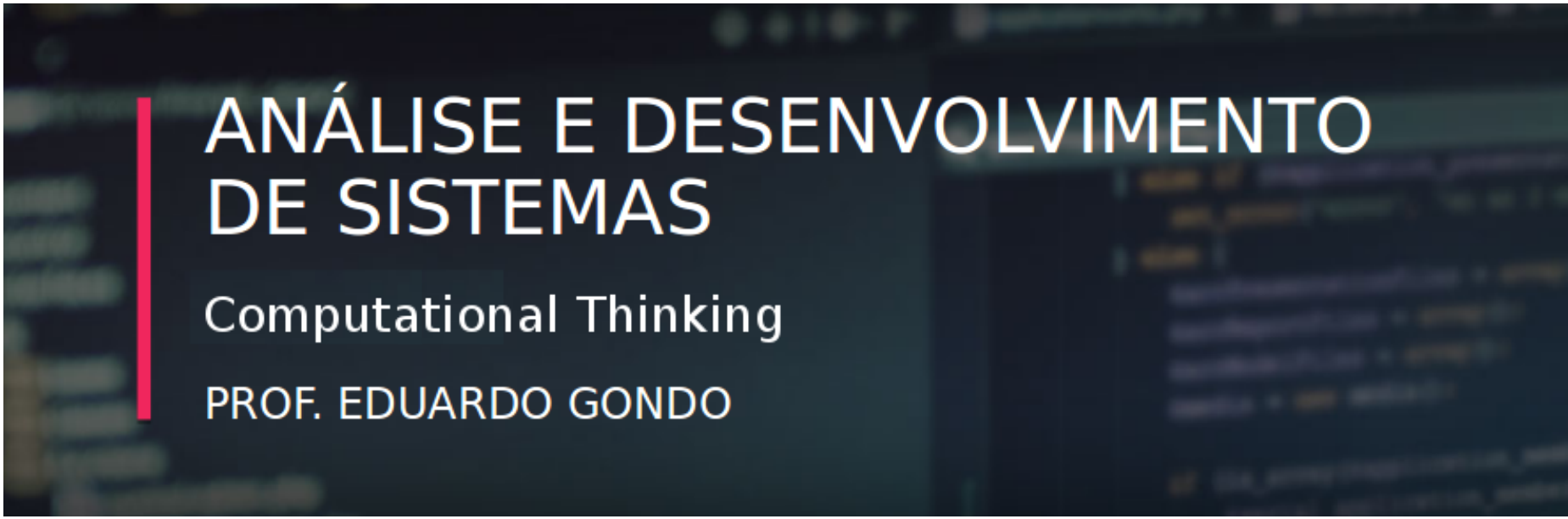

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

Ordenação

PROBLEMA: Encontrar uma permutação sobre um vetor `v[0..n-1]` tal que $v[0] \leq v[1] \leq \ldots \leq v[n-1]$.

O interesse em encontrar algoritmos para ordenar elementos está na eficiência em buscas sobre o vetor.

Já vimos que o busca binária é muito eficiente em comparação ao busca simples.

Ordenação — Algoritmo 1

- é fácil ver que podemos ordenar o vetor gerando todas as permutações dos elementos de v e escolher uma permutação onde os elementos de v estão em ordem crescente
- o problema dessa solução é que ela demora muito quando o número de elementos de v é grande
- desenvolva um método em Java que recebe um vetor contendo uma permutação e verifica se ele está ordenado

Ordenação — Algoritmo 1

- é fácil ver que podemos ordenar o vetor gerando todas as permutações dos elementos de v e escolher uma permutação onde os elementos de v estão em ordem crescente
- o problema dessa solução é que ela demora muito quando o número de elementos de v é grande
- desenvolva um método em Java que recebe um vetor contendo uma permutação e verifica se ele está ordenado

```
1  public boolean estaOrdenado(int[] v) {
2      int i = 1;
3      while (i < v.length && v[i-1] <= v[i])
4          i++;
5      if (i == v.length)
6          return true;
7      else
8          return false;
9  }
```

Ordenação — Algoritmo 2

- considere o seguinte subproblema: dado um vetor v encontre a posição do menor elemento do vetor
- será que este problema nos ajuda a ordenar um vetor?

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos procurar o índice do menor elemento do vetor.

Ordenação — Algoritmo 2

- considere o seguinte subproblema: dado um vetor v encontre a posição do menor elemento do vetor
- será que este problema nos ajuda a ordenar um vetor?

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos procurar o índice do menor elemento do vetor.

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

A posição do menor índice está selecionado em vermelho, em qual posição devemos colocar este elemento se queremos ordenar o vetor?

Ordenação — Simulação

Na primeira posição do vetor!

Ordenação — Simulação

Na primeira posição do vetor!

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos trocar de posição os elementos selecionados em azul.

Ordenação — Simulação

Na primeira posição do vetor!

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

agora, vamos procurar o índice do menor elemento do vetor desconsiderando a primeira posição (em verde).

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

segue agora o vetor com as duas primeiras posições em ordem

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento do vetor desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

segue agora o vetor com as três primeiras posições em ordem

0	1	2	3	4	5	6	7	8	9
5	8	10	19	30	12	84	15	10	17

Ordenacão — Selection Sort

- vamos voltar ao problema de encontrar a posição do menor elemento do vetor
- dado um vetor v e um natural i, encontre a posição de um menor elemento do vetor a partir de i
- segue a assinatura do método `public int menor(int[] v, int i)`
- chamando `j = menor(v, 0)` e trocando o elemento $v[0]$ com $v[j]$ posicionamos o elemento de menor valor na primeira posicao do vetor v
- agora, basta repetir para `j = menor(v, 1)` e trocar o elemento $v[1]$ com $v[j]$ posicionamos o segundo elemento de menor valor na segunda posição do vetor v
- repetimos o processo até para `j = menor(v, v.length-1)` onde teremos nosso vetor ordenado

Selection Sort — Separado em 2 métodos

Segue a implementação da ordenação

```
1  void selectionSort(int[] v) {
2     for(int i = 0; i < v.length - 1; i++) {
3          int j = menor(v, i);
4          int aux = v[i];
5          v[i] = v[j];
6          v[j] = aux;
7     }
8  }
```

a implementação do método `menor`:

```
1  public int menor(int[] v, int i) {
2     int pos = i;
3     i++;
4     while (i < v.length) {
5        if (v[i] < v[pos])
6            pos = i;
7        i++;
8     }
9     return pos;
10 }
```

Selection Sort — Método menor

Segue a implementação da ordenação com os comandos de repetição encadeados:

```
1  void selectionSort(int[] v) {
2     for(int i = 0; i < v.length - 1; i++) {
3          int pos = i;
4          int j = i + 1;
5          while (j < v.length) {
6             if (v[j] < v[pos])
7                 pos = j;
8             j++;
9          }
10         int aux = v[i];
11         v[i] = v[pos];
12         v[pos] = aux;
13    }
14 }
```

Insertion Sort

Antes de mostrar o algoritmo de ordenação Insertion Sort, vamos analisar a situação: dados um vetor de números inteiros onde, apenas a última posição, não está em ordem crescente. Veja um exemplo na figura abaixo:

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44	55	67	27

Você consegue elaborar um algoritmo para colocar na ordem esse último elemento?

Insertion Sort

A ideia é abrir espaço no vetor para colocar o 27 na posição correta. Armazenando o 27 em uma variável movimentamos os elementos:

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44	55		67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44		55	67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32		44	55	67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30		32	44	55	67

Insertion Sort

0	1	2	3	4	5	6	7	8	9
10	15	18	19		30	32	44	55	67

Neste momento, encontramos a posição que o 27 deve ser inserido:

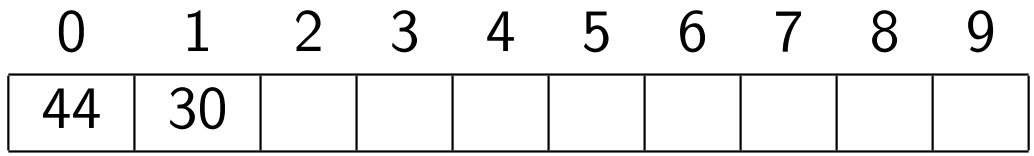
0	1	2	3	4	5	6	7	8	9
10	15	18	19	27	30	32	44	55	67

Segue o algoritmo que coloca o último elemento na posição correta:

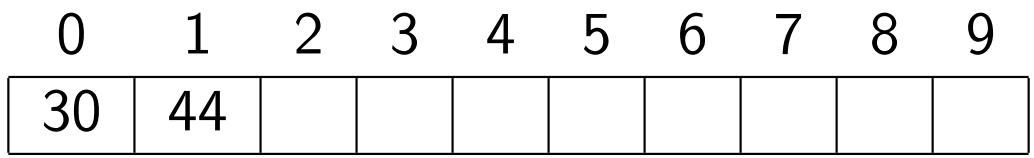
```
1  public void organiza(int[] vetor) {
2      int i = vetor.length - 1;
3      int aux = vetor[i];
4      while (i > 0 && vetor[i-1] > aux) {
5          vetor[i] = vetor[i-1];
6          i--;
7      }
8      vetor[i] = aux;
9  }
```

Insertion Sort

Considere a situação onde queremos ordenar apenas os dois primeiros elementos do vetor:



Alterando o método `organiza(int[] vetor)` para `organiza(int[] vetor, int pos)` onde pos indica a posição do vetor que está fora de ordem. Chamando o método para o vetor acima com pos=1, obtemos:



Insertion Sort

Suponha que no índice 2 temos o 18:

0	1	2	3	4	5	6	7	8	9
30	44	18							

Repetindo a chamada do método organiza para pos=2, obtemos:

0	1	2	3	4	5	6	7	8	9
18	30	44							

Continuando as chamadas até a última posição do vetor, obtemos o vetor ordenado!

Insertion Sort - Implementação

Veja abaixo, a implementação completa do insertion sort:

```
1  public void insertionSort(int[] vetor) {
2     for (int i = 1; i < vetor.length; i++) {
3           int j = i;
4           int aux = vetor[j];
5           while (j > 0 && vetor[j-1] > aux) {
6              vetor[j] = vetor[j-1];
7              j--;
8           }
9           vetor[j] = aux;
10    }
11 }
```

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Deitel e Deitel - Java como Programar
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados

Copyleft