- Componente Curricular: Estrutura de Dados
- Métodos de Ordenação
- Prof. José Luiz de Freitas Júnior

Algoritmos de Ordenação de Vetores

– Ordenação por Seleção

- Ordenação por Inserção

– BubbleSort

– ShellSort

– Outros

Critérios de Avaliação

- Seja n o número de registros em um vetor, considera-se duas medidas de complexidade:

  - ✓ Número de comparações C(n) entre as chaves;

  - ✓ Número de trocas ou movimentações M(n) de itens.

Ordenação por Seleção

```
for (int i = 0; i < dados.length - 1; i++) {
        int indexDoMenor = i;
        for (int j = i + 1; j < dados.length; j++) {
                if (dados[indexDoMenor] > dados[j]) {
                    indexDoMenor = j;
                }
        }
         int temp = dados[i];
         dados[i] = dados[indexDoMenor];
         dados[indexDoMenor] = temp;
}
```

FUNCIONAMENTO

A ideia é sempre procurar o menor elemento do vetor e inseri-lo no início do vetor. Procuramos o menor valor do vetor e colocamos ele em vetor[0]. Procuramos o menor valor do vetor excluindo o já colocado e colocamos ele em vetor[1]. E assim vamos indo até termos todo o vetor ordenado.

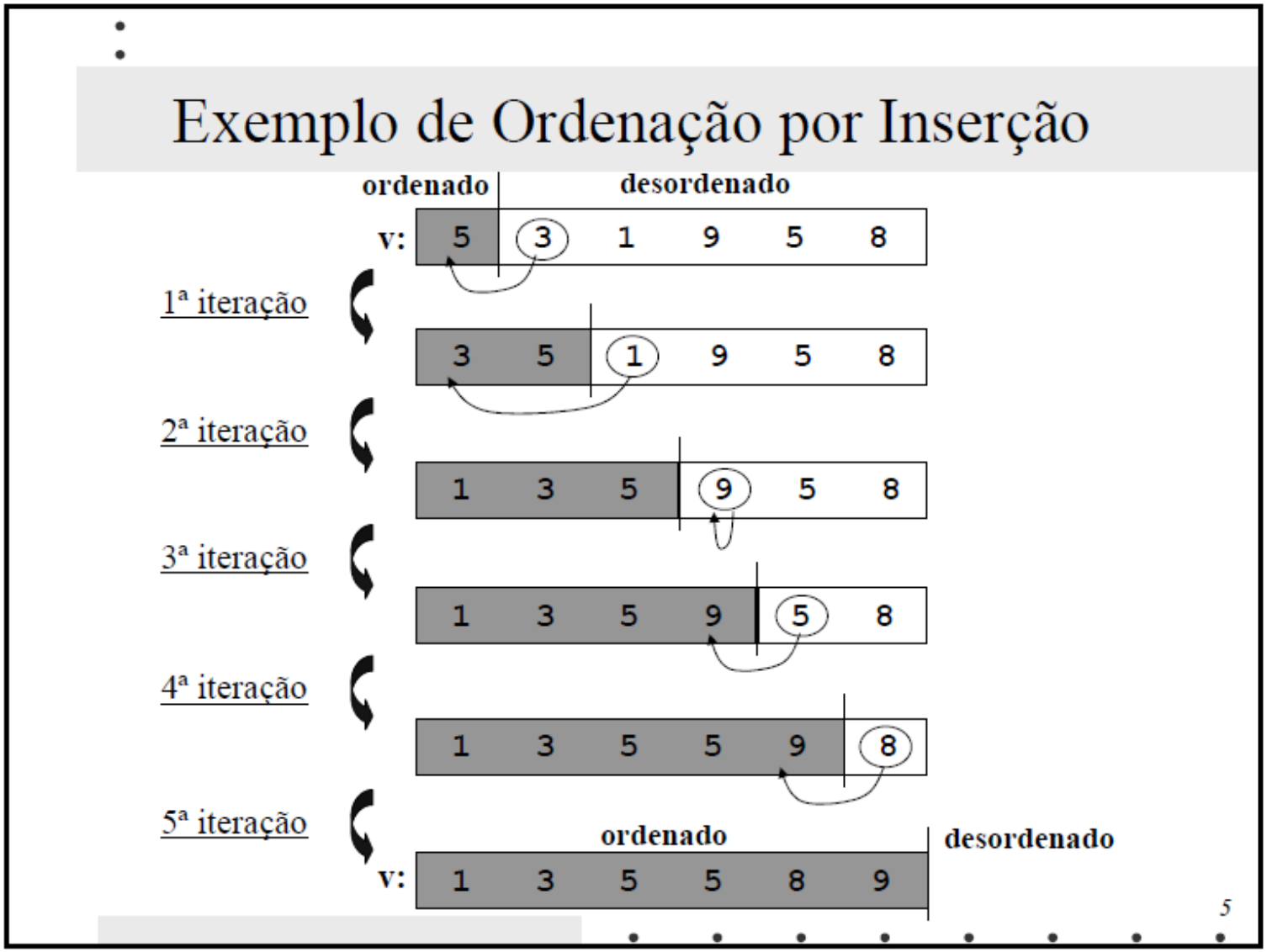
Partindo sempre a partir do último elemento reordenado (a partir do i), o programa procura o menor elemento no vetor e o substitui pelo elemento i atual.

EXEMPLO DE FUNCIONAMENTO

E	X	E	M	P	L	O
E	X	E	M	P	L	O
E	E	X	M	P	L	O
E	E	L	M	P	X	O
E	E	L	M	P	X	O
E	E	L	M	O	X	P
E	E	L	M	O	P	X
E	E	L	M	O	P	X

CUSTO: Este algoritmo não tem um melhor/pior caso, porque todos os elementos são varridos, sempre. Medir seu custo é simples. Custo de linha por linha... Custo= $(n^2+n)/2$

Ordenação por Inserção



Ordenação por Inserção

```
1  class insercao{
2  public static void insert (int vet [])
3  {
4
5       for (int j = 1; j < vet.length; j++){
6
7       int i;
8       int aux;
9       aux = vet[j];
10      i = j - 1 ;
11
12        while (i >= 0 && vet[i] > aux){
13
14          vet[i + 1] = vet[i];
15          i = i - 1;
16          }
17            vet[i + 1] = aux;
18 }
19 }
20 }
```

Ordenação por Inserção

Análise da Ordenação por Inserção

- 2 ciclos encaixados, cada um pode ter N iterações :
  - **$O(N^2)$**
- Caso mais desfavorável: vetor em ordem inversa
  - **$O(N^2)$**
- Caso mais favorável: vetor já ordenado
  - **$O(N)$**
- **Conclusão:**
  - **Só deve ser utilizado para vetores pequenos…**

BubbleSort (Bolha)

– Compara elementos adjacentes. Se o segundo for menor do que o primeiro, troca-os;

– Fazer isto desde o primeiro até ao último par

– Repetir para todos os elementos exceto o último (que já está correto)

– Repetir, usando menos um par em cada iteração até não haver mais pares (ou não haver trocas)

• 2 ciclos encaixados, cada um pode ter N iterações:

– **Complexidade $O(N^2)$**

BubbleSort (Bolha)

- O método ilustrado:

	1	2	3	4	5	6
Chaves iniciais:	O	R	D	E	N	A
i = 6	O	D	E	N	A	**R**
i = 5	D	E	N	A	**O**	R
i = 4	D	E	A	**N**	O	R
i = 3	D	A	**E**	N	O	R
i = 2	A	**D**	E	N	O	R

BubbleSort (Bolha)

```
public static void bolha(int vet[]) {
        int i, j, aux;
        for (i = vet.length - 1; i > 0; i--) {
            // o maior valor entre vet[0] e vet[i] vai para a posição vet[i]
            for (j = 0; j < i; j++) {
                if (vet[j] > vet[j + 1]) {
                    aux = vet[j];
                    vet[j] = vet[j + 1];
                    vet[j + 1] = aux;
                }
            }
        }
    }
```

BubbleSort (Bolha _melhorado)

```
public static void bolhaMelhor(int vet[]) {
      int i, j, aux;
      boolean troca = false;
      for (i = vet.length - 1; i > 0; i--) {
         // o maior valor entre vet[0] e vet[i] vai para a posição vet[i]
         troca = false;
         for (j = 0; j < i; j++) {
            if (vet[j] > vet[j + 1]) {
               aux = vet[j]; vet[j] = vet[j + 1];   vet[j + 1] = aux;
               troca = true;
            }
         }
         if (!troca)
            return;
       }
  }
```

InsertionSort X BubbleSort

	Melhor caso	Pior caso
InsertionSort	$O(n)$	$O(n^2)$
BubbleSort	$O(n^2)$	$O(n^2)$

ShellSort

- Criado por Donald Shell em 1959, publicado pela Universidade de Cincinnati, Shell sort é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática.
- O algoritmo difere do método de inserção direta pelo fato de no lugar de considerar o array a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles.

ShellSort

- Compara elementos distantes;

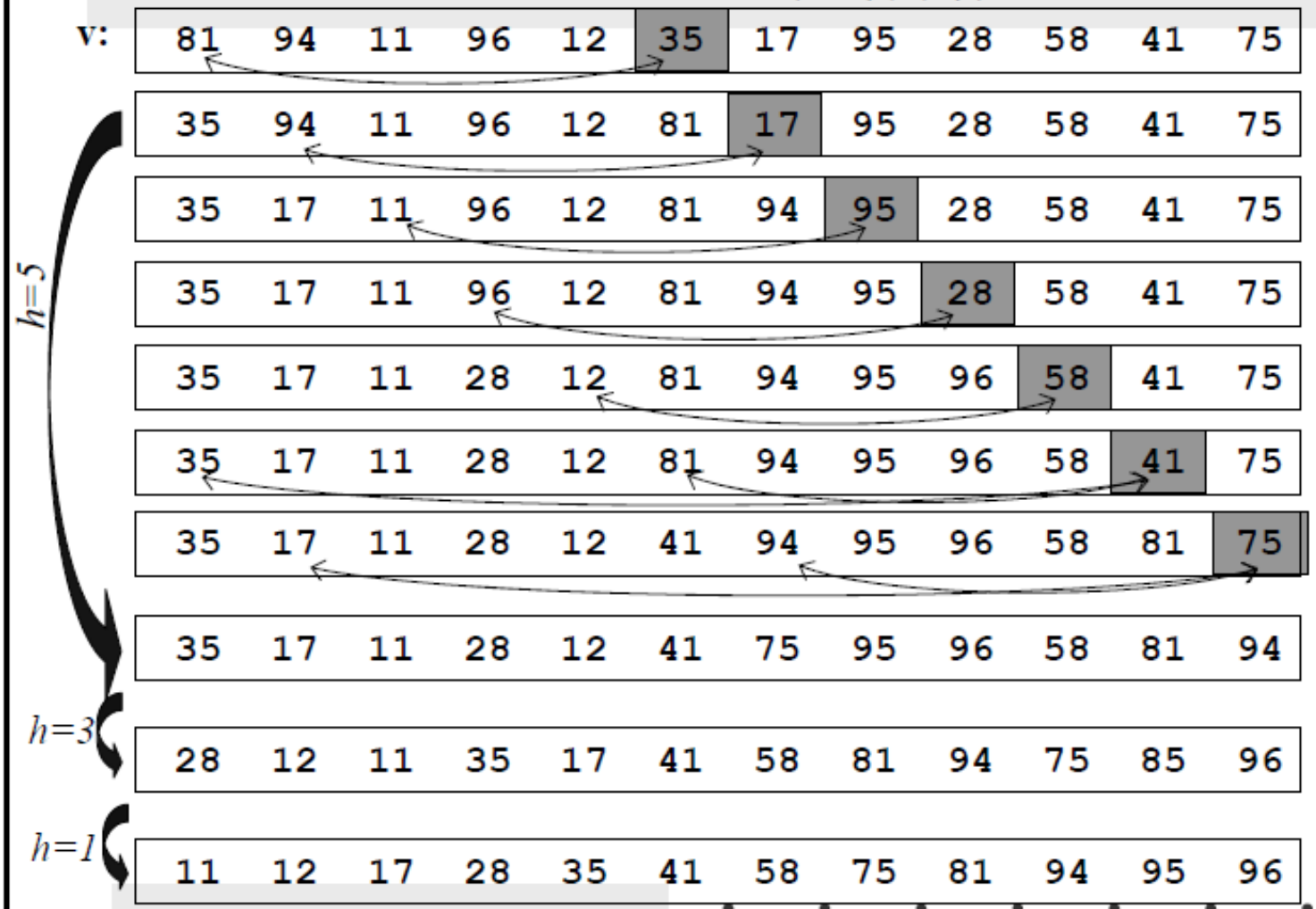
- Distância entre elementos comparados vai diminuindo, até que a comparação seja sobre elementos adjacentes

  – Usa a sequência *h1, h2, …, ht (h1=1)*

  – Em determinado passo, usando incremento *hk, todos os* elementos separados da distância *hk estão ordenados, vet[i] ≤ vet[i+hk].*

- https://www.youtube.com/watch?v=RVbgifh0HEA

ShellSort (h = {5, 3, 1})



ShellSort

```
public static void ordena_shellSort(Aluno[] a, int qde) {
        Aluno tmp;
        int j;
        for (int gap = qde / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < qde; i++) {
                tmp = a[i];
                for (j = i; j >= gap && (tmp.getNome().compareToIgnoreCase(a[j -
gap].getNome())< 0); j -= gap) {
                    a[j] = a[j - gap];
                }
                a[j] = tmp;
            }
        }
    }
```