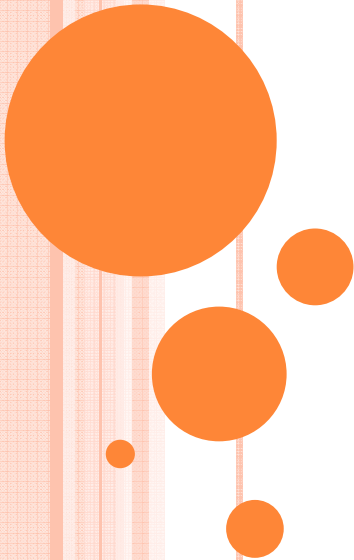




ALGORITMOS E ESTRUTURAS DE DADOS CES-11

Prof. Paulo André Castro

pauloac@ita.br

Sala 110 – Prédio da Computação

www.comp.ita.br/~pauloac

IECE - ITA

CAP. 5. TÉCNICAS DE ORDENAÇÃO

5.1. Introdução

5.2. Métodos simples de ordenação

5.3. O método Shell-Sort

5.4. O método Quick-Sort

5.1 – INTRODUÇÃO

- **Ordenação** é uma atividade importantíssima em processamento automático de informações.
 - Que tal **procurar**
 - Um nome numa lista telefônica
 - Um nome numa lista de contas bancárias
 - Uma palavra em um dicionário lingüístico
- se as **relações** não estiverem em **ordem alfabética**?

- O processo de ordenação se aplica a um conjunto de **objetos** em que se possa verificar entre eles a relação

$\leq$ (**menor ou igual**), ou $\geq$ (**maior ou igual**).

- Os conjuntos mais comuns são vetores de

- Números

- Nomes ou cadeias de caracteres em geral

- **Vetores de estruturas** também podem ser ordenados
- **Exemplo:** relação com diversas informações sobre os funcionários de uma empresa:

[illegible]

Nome	Data de Nascimento			Setor	Salário
	Dia	Mês	Ano		

- Este vetor pode ser ordenado por **qualquer campo** dessas estruturas:
 - Ordem alfabética dos **nomes** ou dos **setores**
 - Ordem crescente ou decrescente de **salários** ou de **idade**
- O **campo** de um vetor de estruturas usado para **ordená-lo** é denominado **chave** da ordenação.

- Existem vários algoritmos para fazer ordenação de vetores:
 - Alguns mais **simples**, porém **ineficientes**
 - Outros mais **eficientes**, porém mais **complexos**
- Alguns **critérios** para julgar a **eficiência** dos algoritmos de ordenação:
 - Número de **comparações** de chaves
 - Número de **movimentações** de elementos

- O **tempo de comparação** depende da **chave** usada:
 - Para números, basta um dos operadores $\leq$ ou $\geq$;
 - Para cadeias de caracteres, usa-se funções para compará-las; exemplo: **strcmp** da linguagem C
- Há casos de **chaves múltiplas**, como a ordenação por **idade**, sendo dada a **data de nascimento**:
 - Compara-se primeiramente o **ano**, depois o **mês** e finalmente o **dia**
 - Nestes casos, a comparação é mais **complexa** e pode ser conveniente um **subprograma** específico

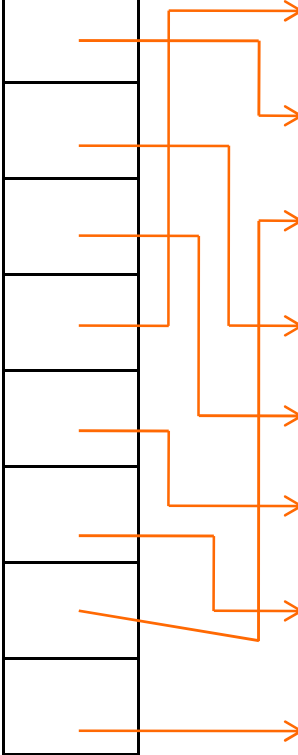
- O **tempo de movimentação** depende do **tipo** e **tamanho** dos elementos do vetor a ser ordenado:
 - Para **números**, a movimentação é **pequena**
 - Para **grandes estruturas**, há **grande** movimentação de dados

- No caso de **grandes estruturas**, usa-se um **vetor de ponteiros**, cada um para uma estrutura do vetor

	Pont	Nome	Data de Nascimento			Setor	Salário
			Dia	Mês	Ano		
1	→	Gilberto	03	05	1980	Almox	850
2	→	Antonio	05	12	1970	Admin	2300
3	→	Pedro	23	07	1975	Fabric	1050
4	→	Augusto	05	10	1960	Diret	8000
5	→	Claudio	07	03	1984	Ferr.	1850
6	→	Marcos	17	11	1976	Recep	850
7	→	Nilton	20	01	1982	Proj	2300
8	→	Vicente	15	08	1979	Contr	2000

- Para a ordenação basta **alterar os apontamentos**, evitando-se a movimentação dessas estruturas

	Pont	Nome	Data de Nascimento			Setor	Salário
			Dia	Mês	Ano		
1		Gilberto	03	05	1980	Almox	850
2		Antonio	05	12	1970	Admin	2300
3		Pedro	23	07	1975	Fabric	1050
4		Augusto	05	10	1960	Diret	8000
5		Claudio	07	03	1984	Ferram.	1850
6		Marcos	17	11	1976	Recep	850
7		Nilton	20	01	1982	Proj	2300
8		Vicente	15	08	1979	Contr	2000



- No restante deste capítulo, para maior **simplicidade**, os **algoritmos** irão atuar sobre **vetores de inteiros**.
- A seguinte declaração será usada por esses algoritmos:

```
typedef int vetor [nmax]
```

(**nmax** é o número máximo de elementos dos vetores)

- Para vetores de **outros tipos**, basta alterar as **comparações** e os **movimentos** de elementos.
- São apresentados a seguir **três métodos simples** de ordenação e, depois, **quatro** outros mais **eficientes**.

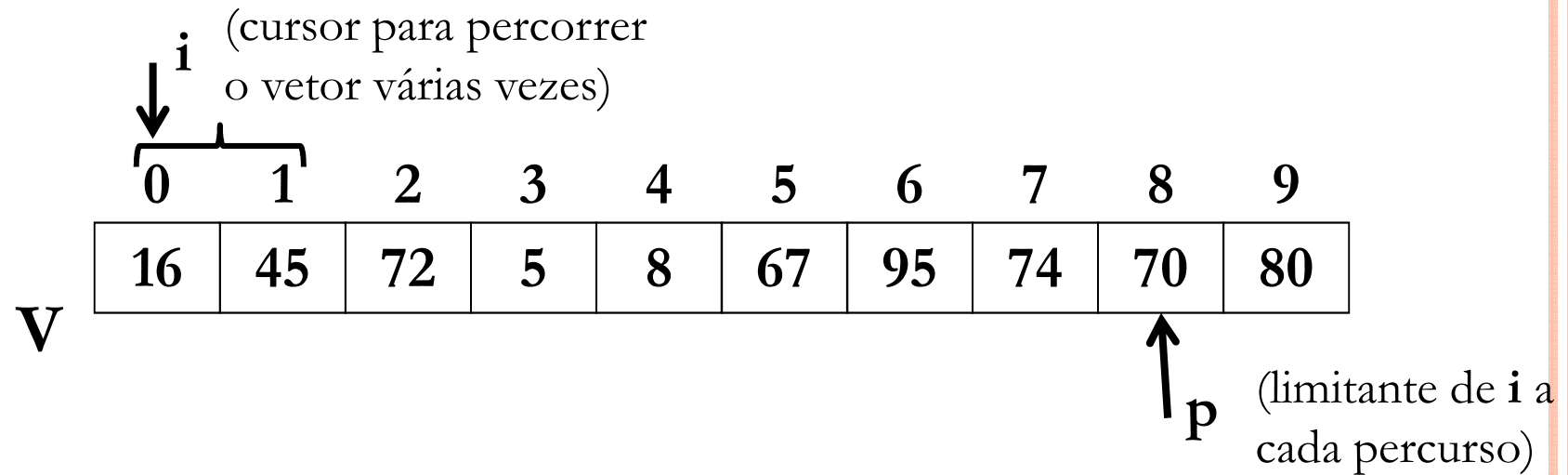
5.2 – MÉTODOS SIMPLES DE ORDENAÇÃO

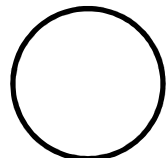
- Tais métodos são considerados lentos **$O(n^2)$** no pior caso e no caso médio)
- São usados para ordenar vetores pequenos
- São apresentados aqui três desses métodos:
 - Bubble-Sort
 - Insertion-Sort
 - Selection-Sort

5.2.1 – Bubble-Sort

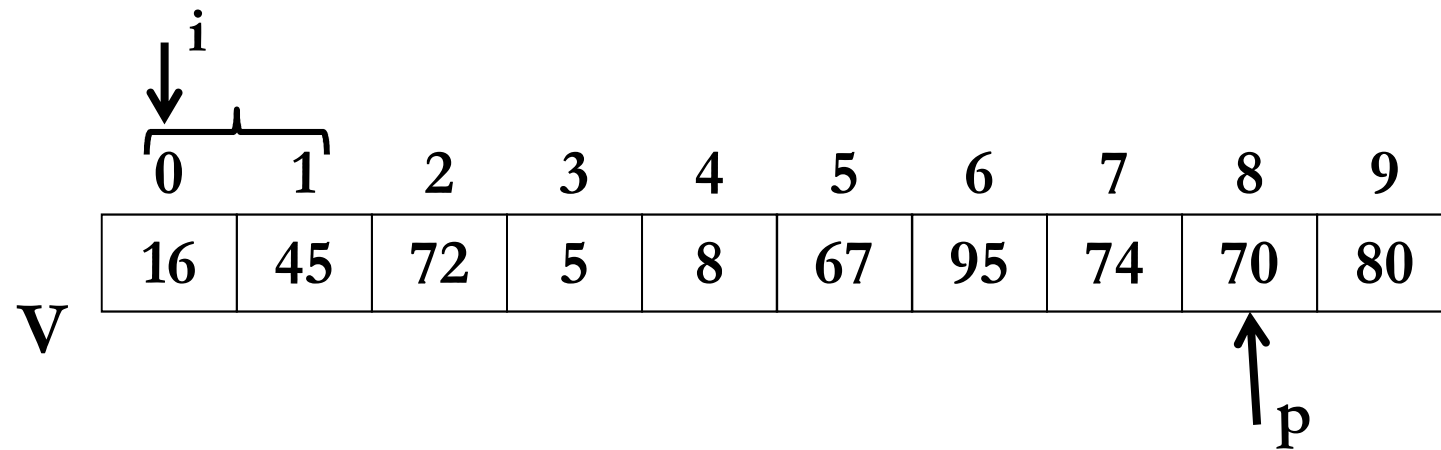
- O método consiste em:
 - **Percorrer o vetor** várias vezes
 - Durante cada percurso, efetuar troca de posição de **elementos adjacentes**, caso o elemento da esquerda seja maior que o da direita
 - Se, durante um dos percursos, não houver trocas, considera-se que o vetor está ordenado
 - Em cada percurso, um elemento atinge sua posição definitiva

Exemplo: seja o vetor abaixo e o método em seu início



 **trocou**
(semáforo que acende quando há troca)

Não trocar



○ trocou

Não trocar

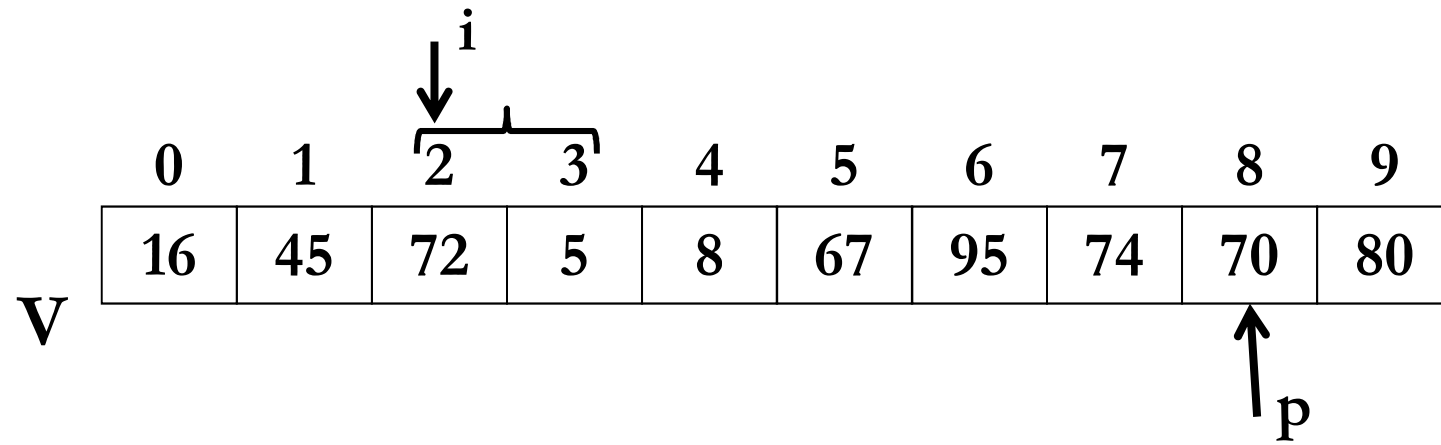
V

0	1	2	3	4	5	6	7	8	9
16	45	72	5	8	67	95	74	70	80

Diagram illustrating a selection sort step. The array **V** contains the values [16, 45, 72, 5, 8, 67, 95, 74, 70, 80]. The current element being compared is at index 1 (value 45), indicated by a downward arrow labeled **i**. The element at index 8 (value 70) is the current minimum, indicated by an upward arrow labeled **p**. A bracket connects index 1 and index 2.

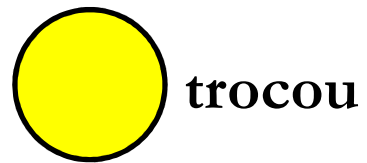
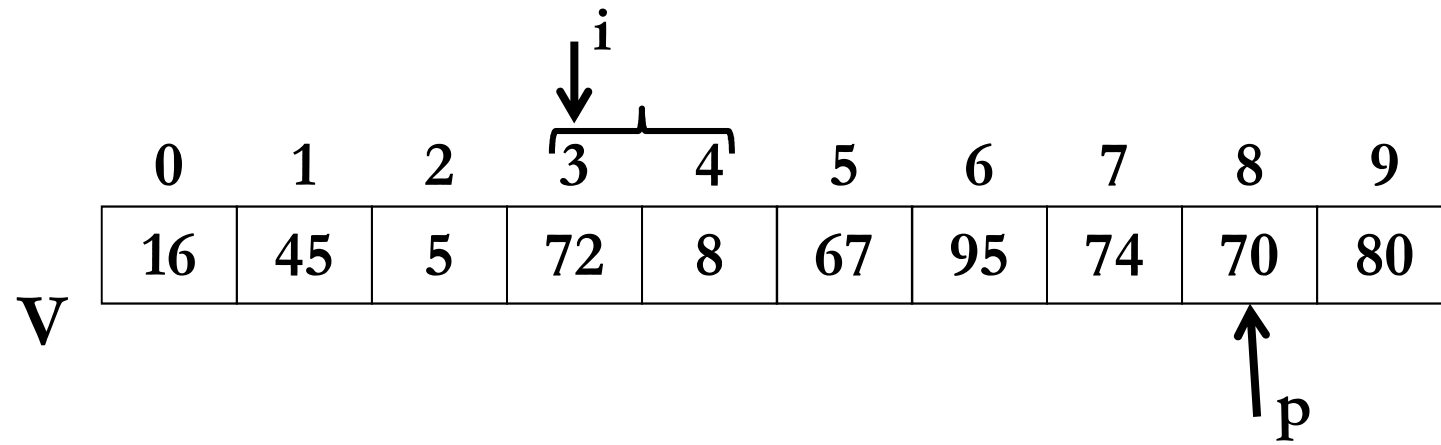
○ trocou

Trocar

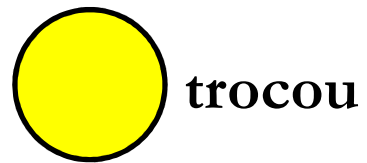
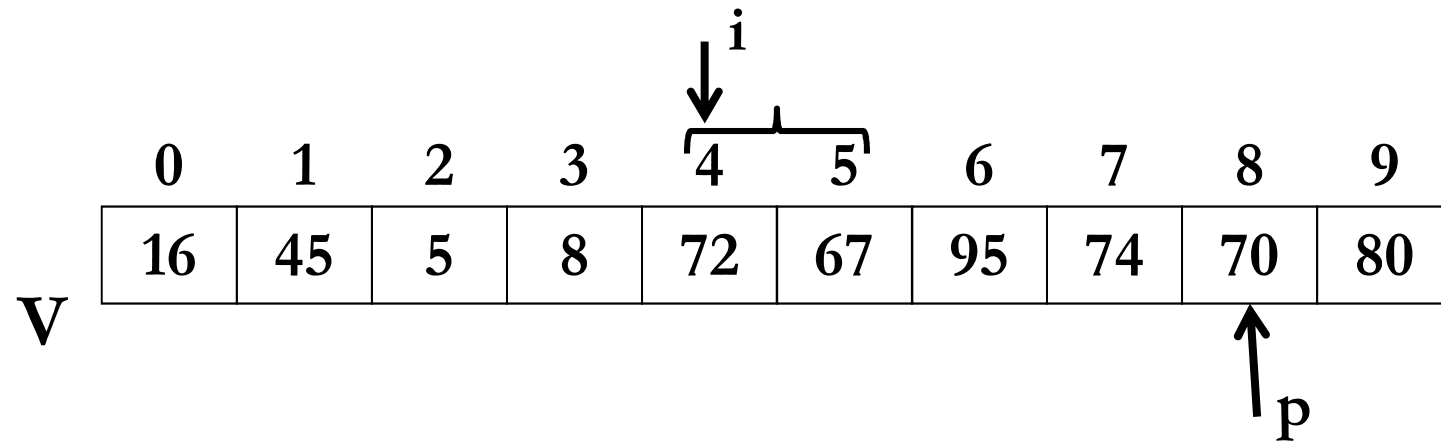


trocou

Trocar



Trocar

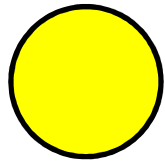


Não trocar

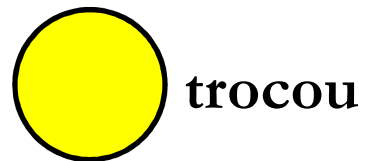
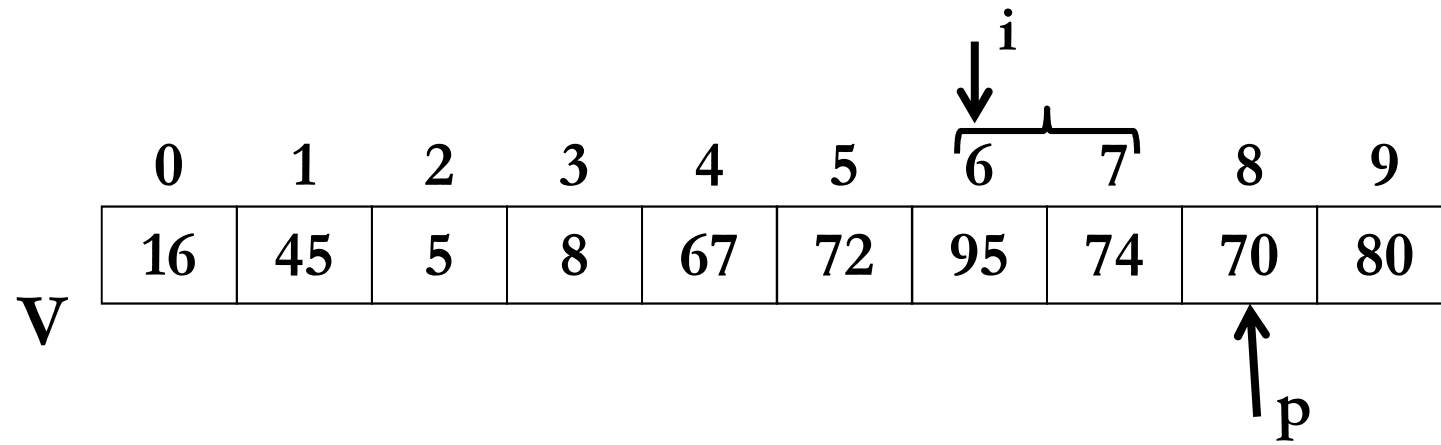
V

0	1	2	3	4	$\overset{i}{\downarrow}$ 5	6	7	8	9
16	45	5	8	67	72	95	74	70	80

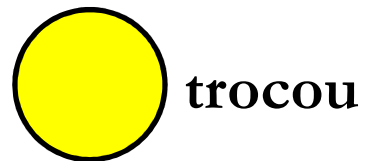
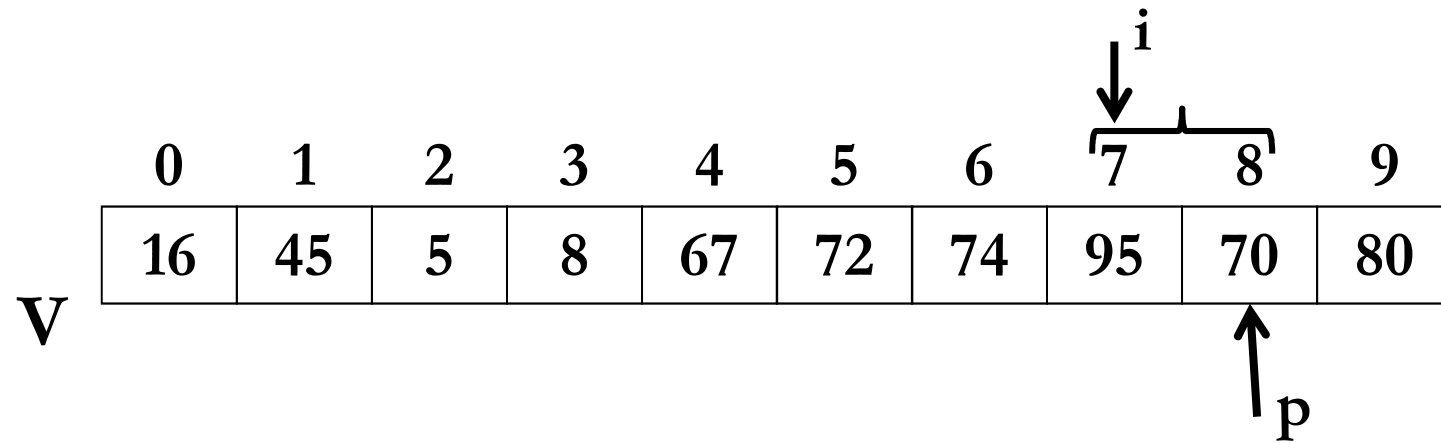
$\uparrow$
p

 trocou

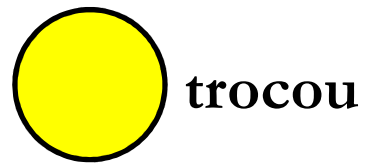
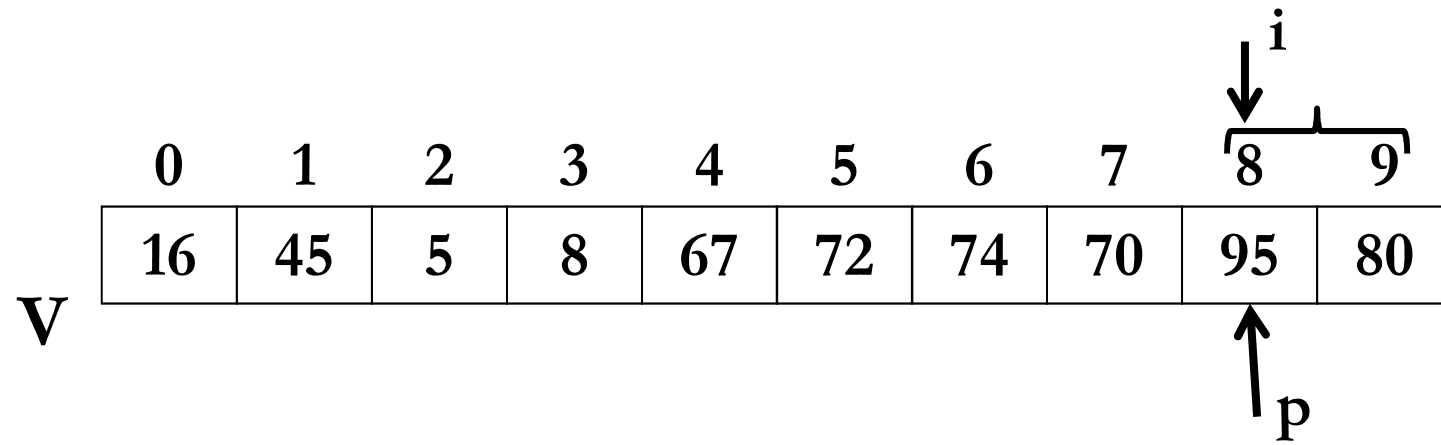
Trocar



Trocar



Trocar

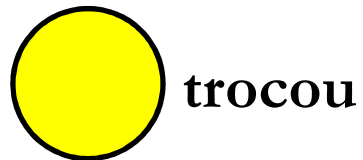


95 em sua posição definitiva

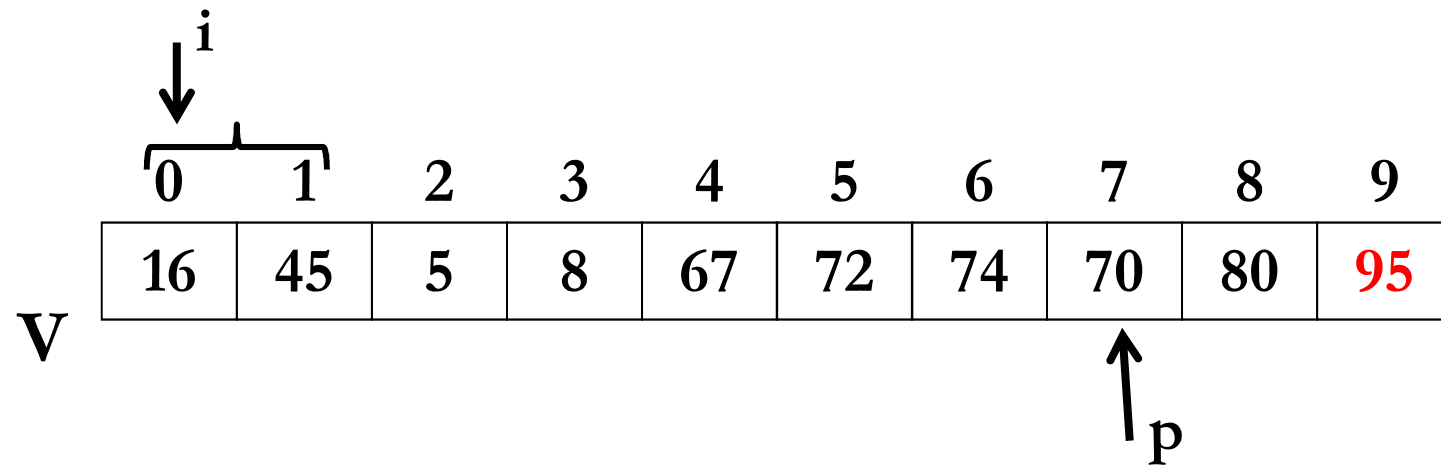
trocou acesa: começar novo percurso

retroceder p

	0	1	2	3	4	5	6	7	8	9
V	16	45	5	8	67	72	74	70	80	95
									↑ p	

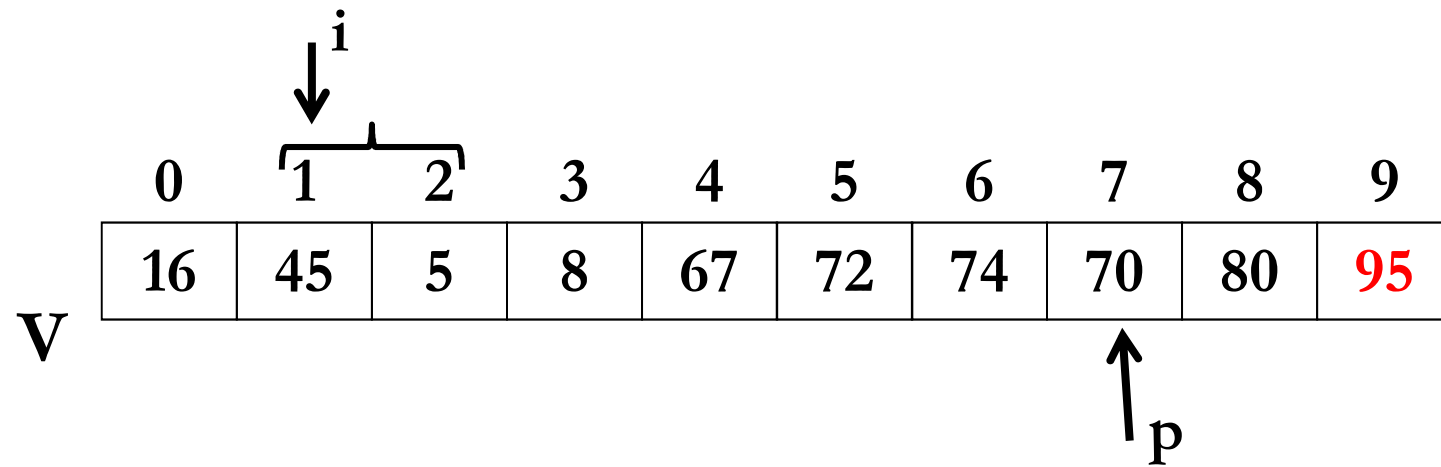


Não trocar



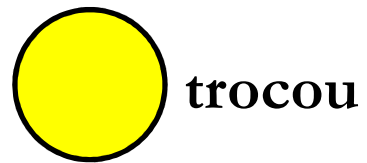
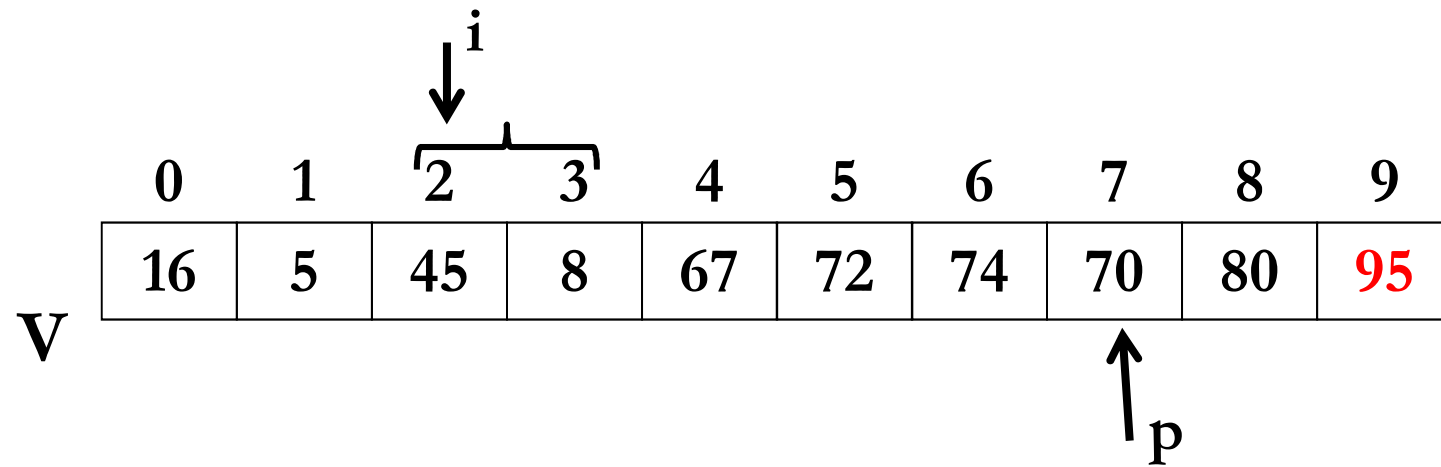
○ trocou

Trocar



○ trocou

Trocar

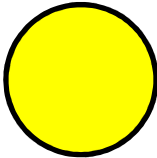


Não trocar

V

0	1	2	3	4	5	6	7	8	9
16	5	8	45	67	72	74	70	80	95

Diagram illustrating a selection sort step. The array **V** contains the values [16, 5, 8, 45, 67, 72, 74, 70, 80, 95]. The current element being compared is at index 3 (value 45), indicated by a downward arrow labeled **i**. The element at index 7 (value 70) is indicated by an upward arrow labeled **p**. A bracket connects indices 3 and 4, suggesting a comparison or swap operation. The value 95 at index 9 is highlighted in red.

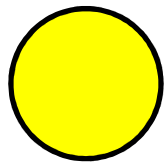
 trocou

Não trocar

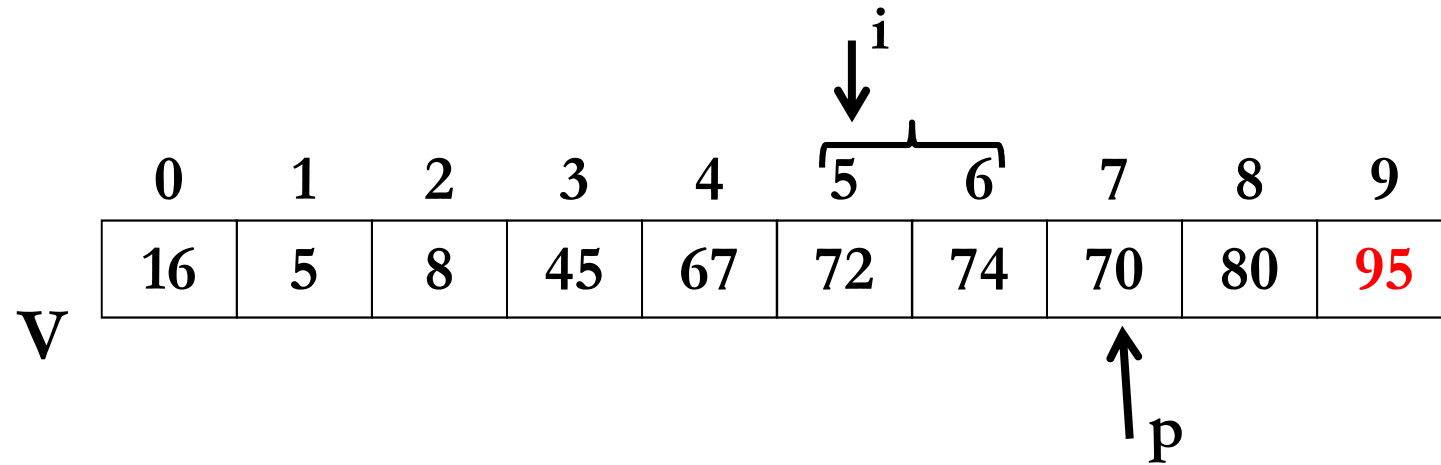
V

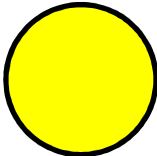
0	1	2	3	$\downarrow$ 4	5	6	7	8	9
16	5	8	45	67	72	74	70	80	95

$\uparrow$
p

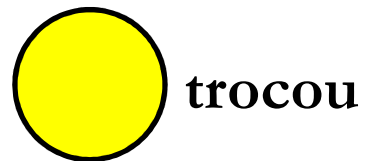
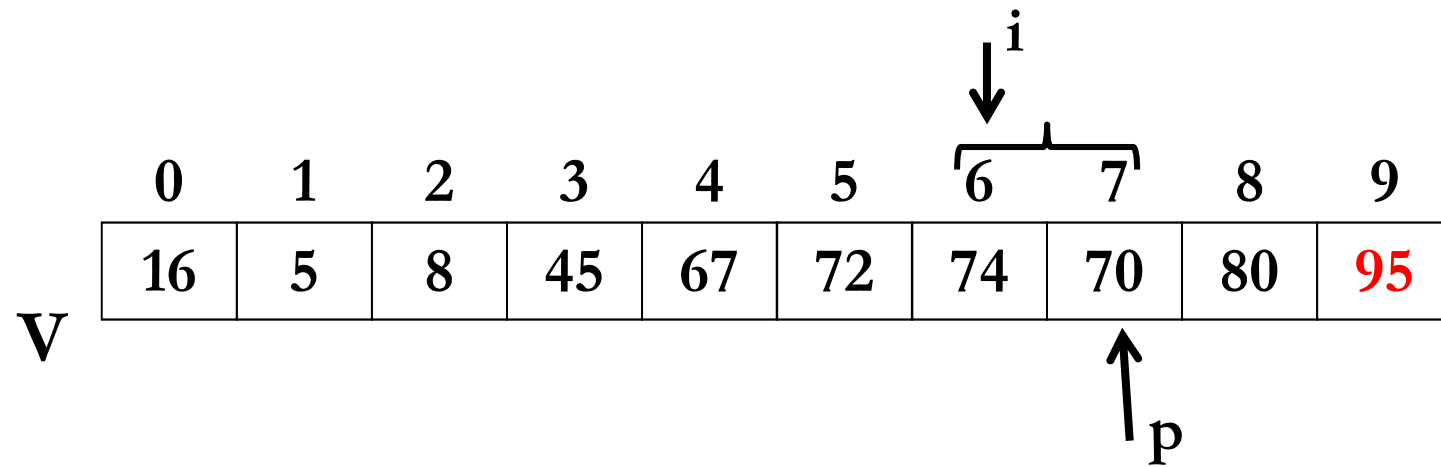
 trocou

Não trocar



 trocou

Trocar



Não trocar

V

0	1	2	3	4	5	6	i ↓ 7	8	9
16	5	8	45	67	72	70	74	80	95

↑
p

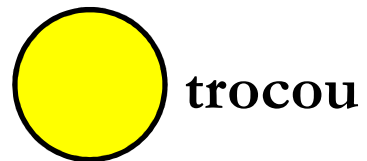
trocou

trocou acesa: iniciar novo percurso

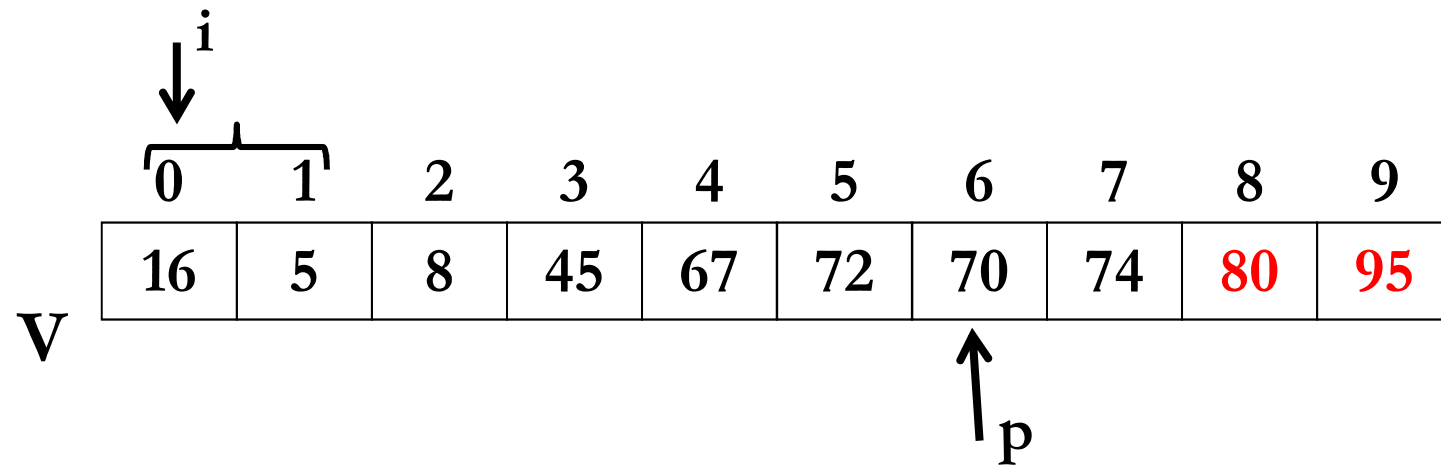
V

0	1	2	3	4	5	6	7	8	9
16	5	8	45	67	72	70	74	80	95

↑
p

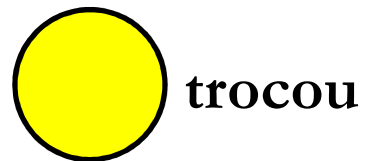
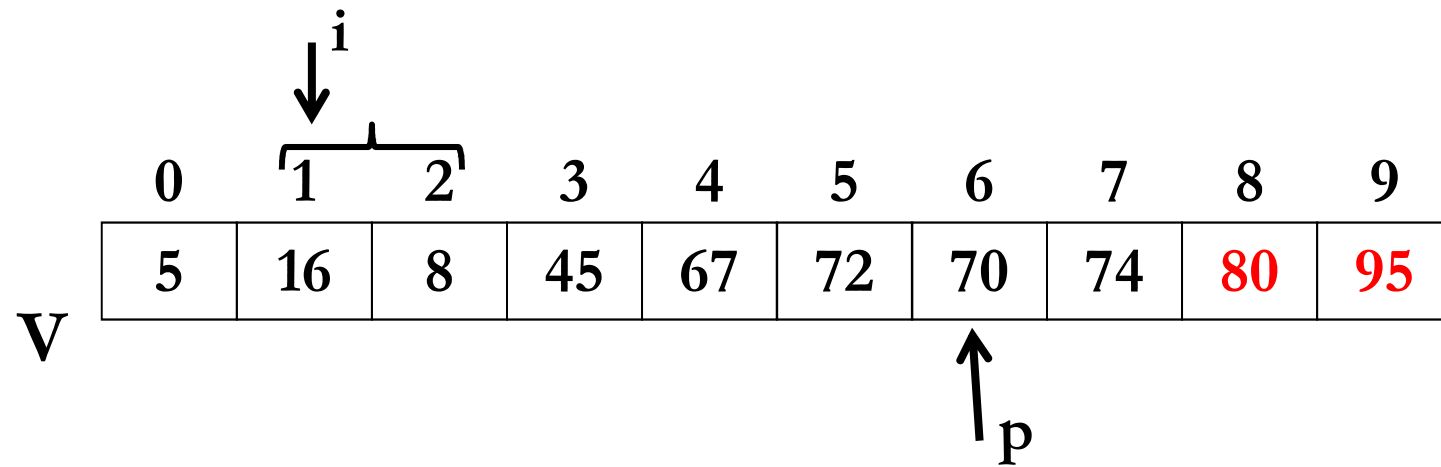


Trocar

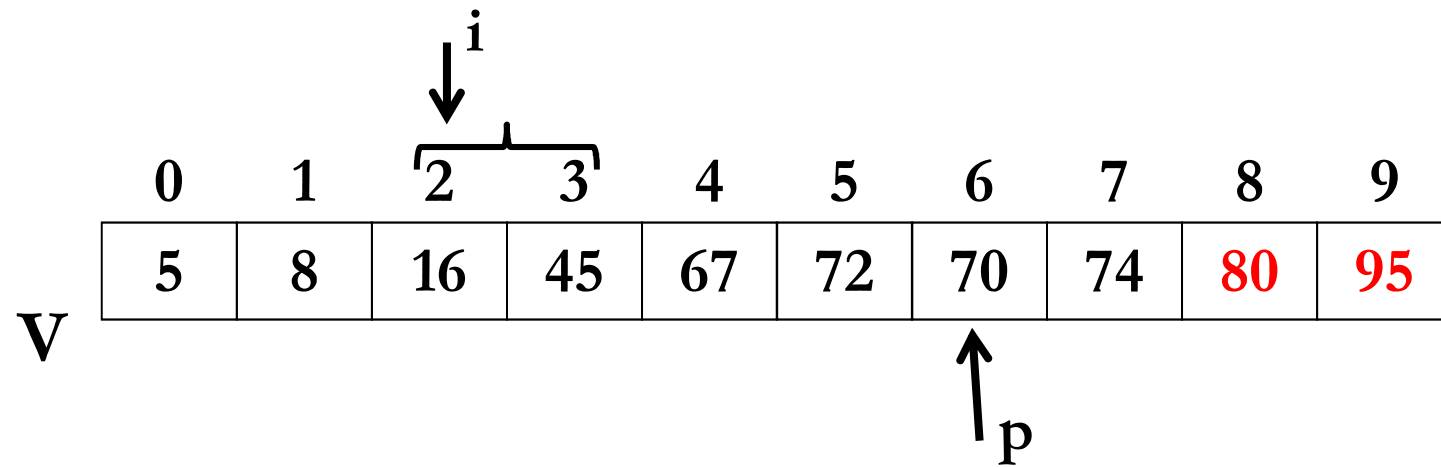


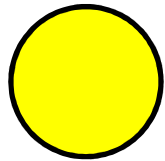
○ trocou

Trocar



Não trocar



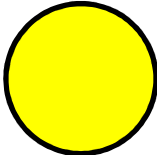
 trocou

Não trocar

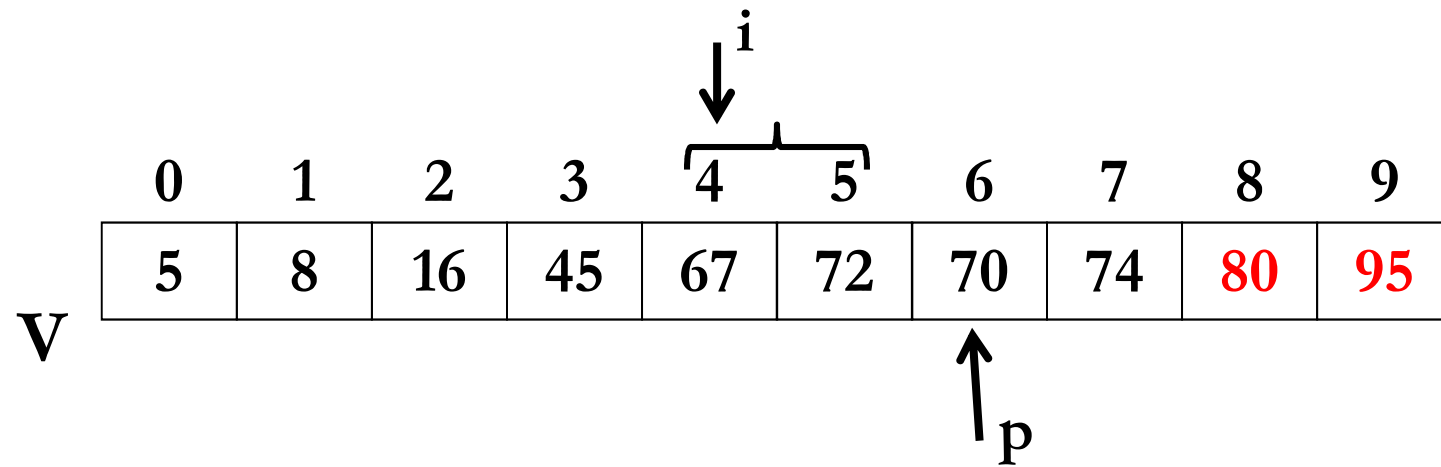
V

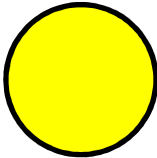
0	1	2	$\downarrow^i$ 3	4	5	6	7	8	9
5	8	16	45	67	72	70	74	80	95

$\uparrow_p$

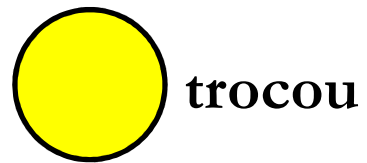
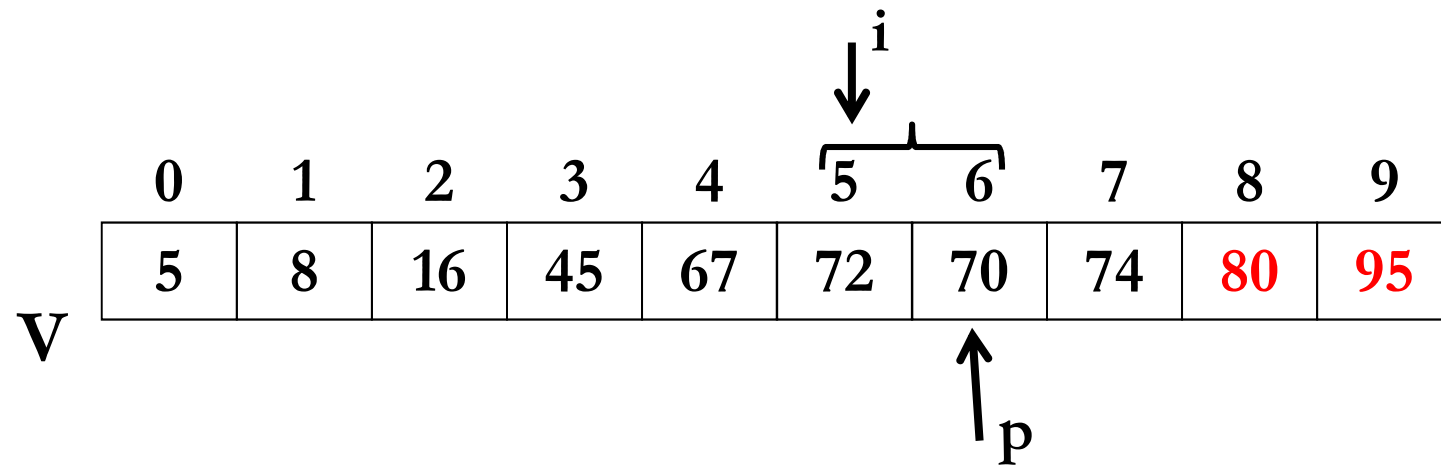
 trocou

Não trocar

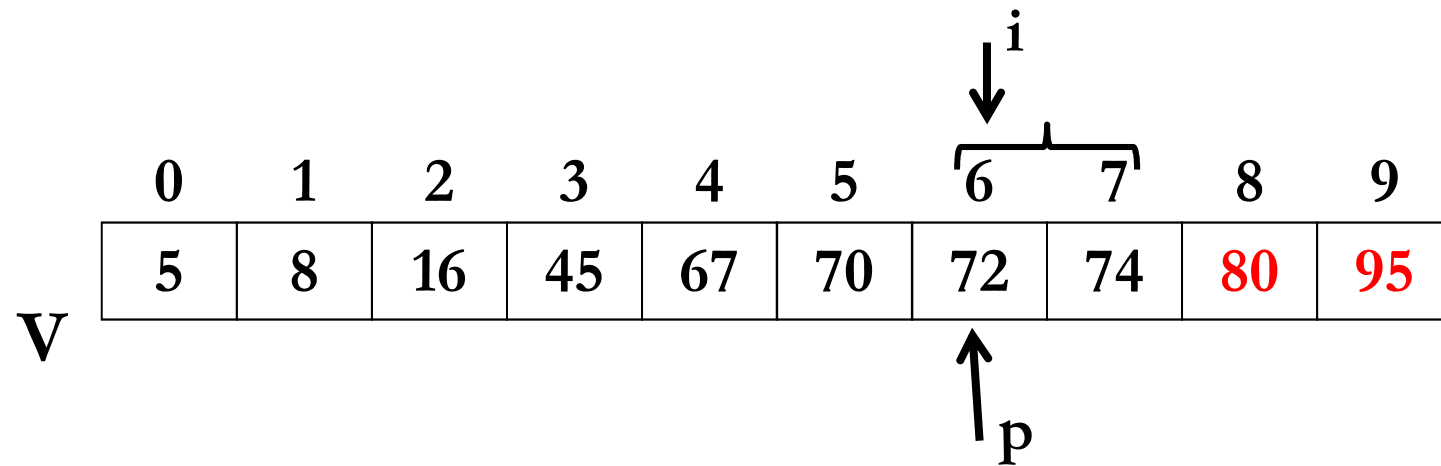


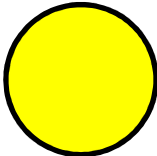
 trocou

Trocar



Não trocar



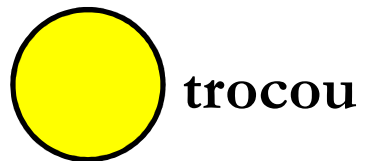
 **trocou**

trocou acesa: novo percurso

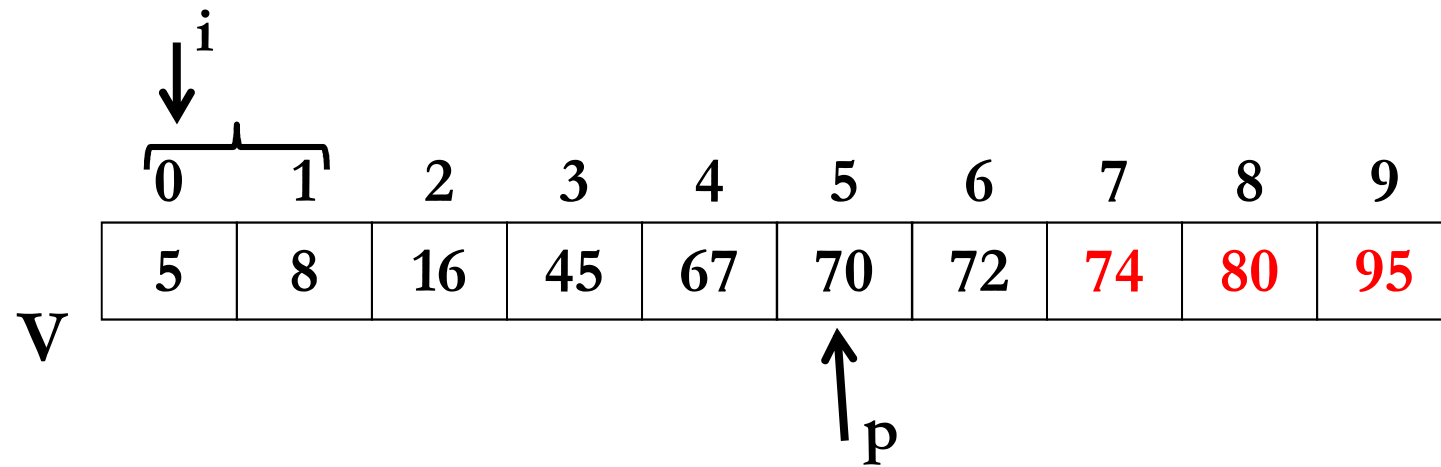
V

0	1	2	3	4	5	6	7	8	9
5	8	16	45	67	70	72	74	80	95

↑
p

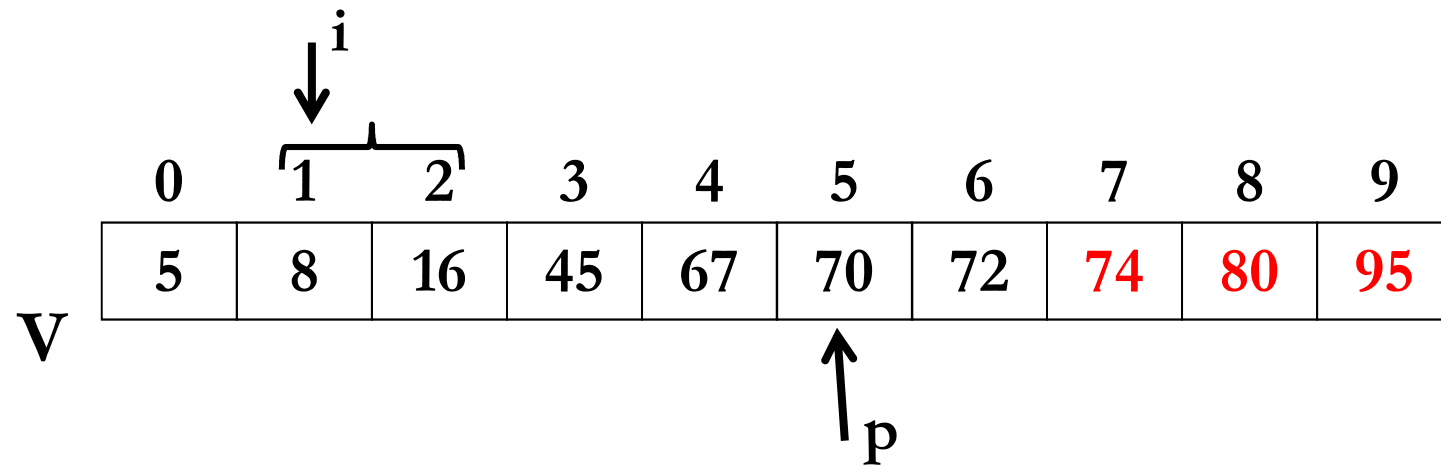


Não trocar



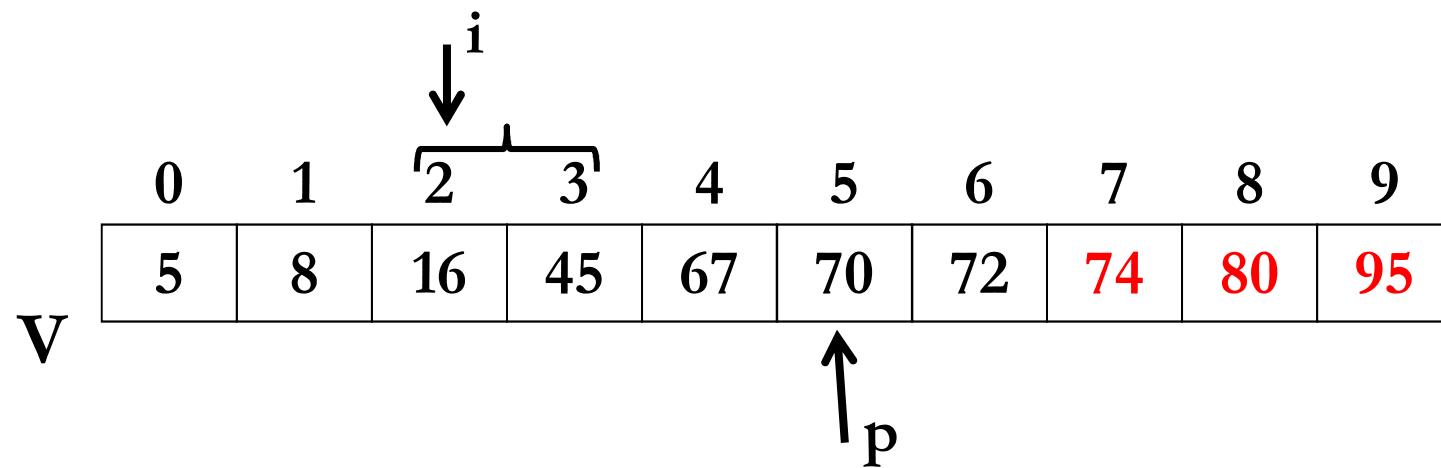
○ trocou

Não trocar



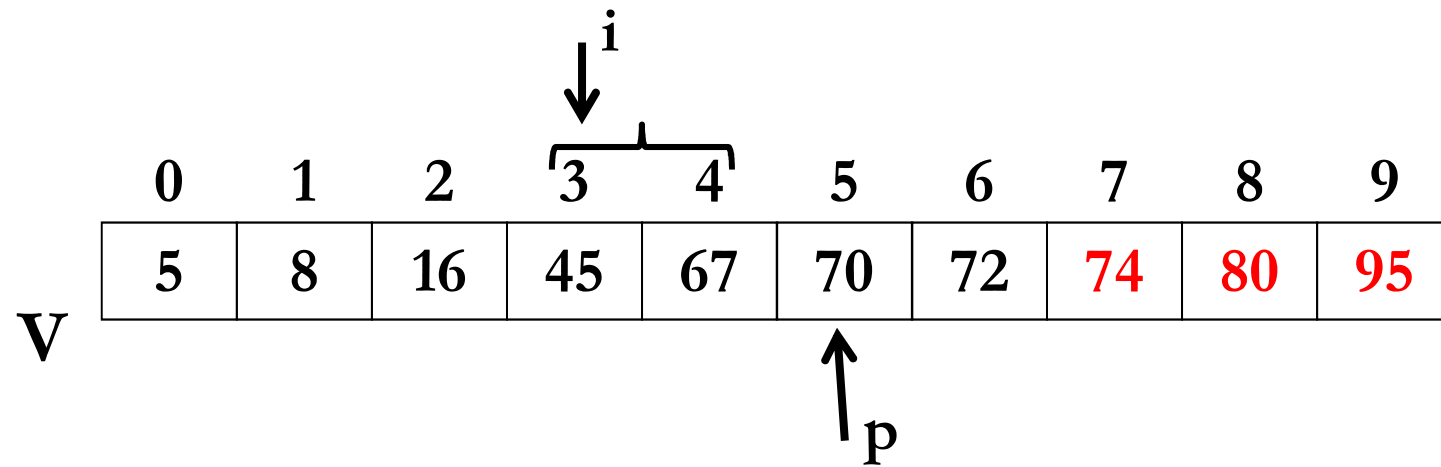
○ trocou

Não trocar



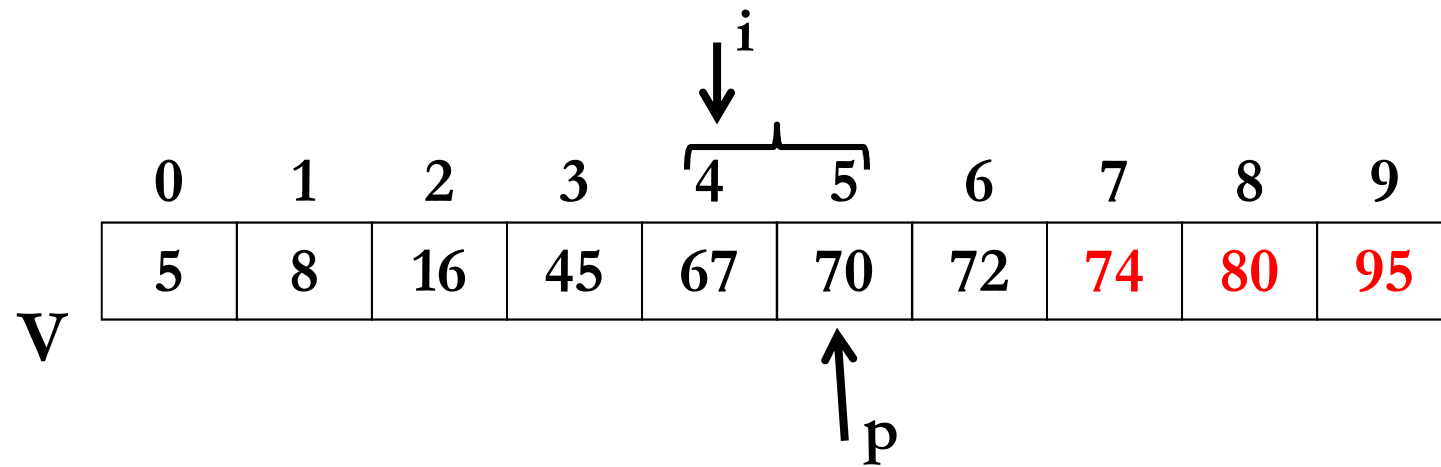
○ trocou

Não trocar



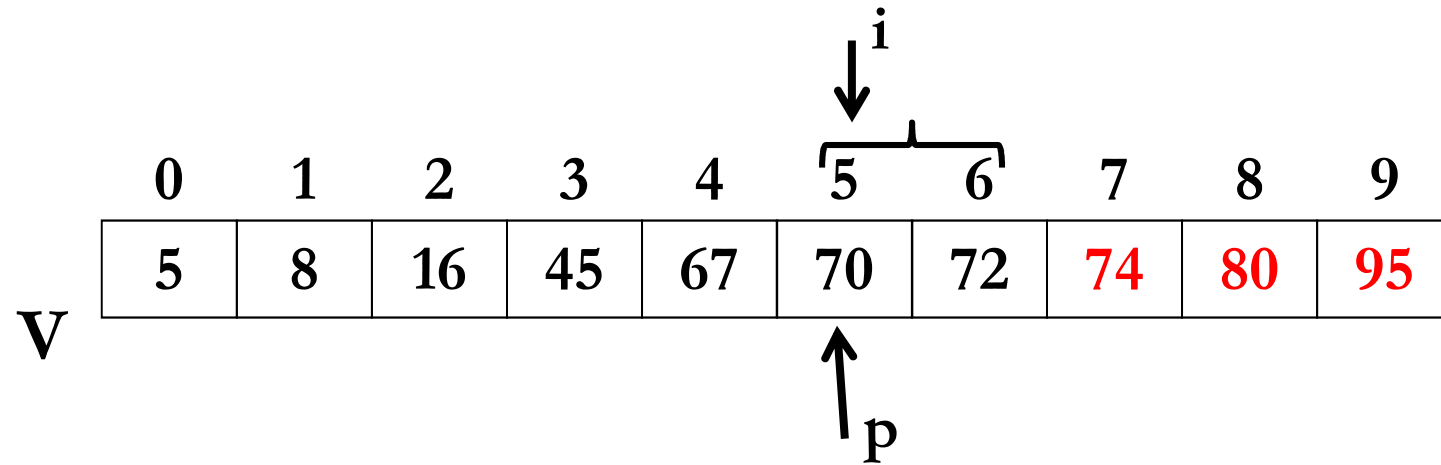
○ trocou

Não trocar



○ trocou

Não trocar

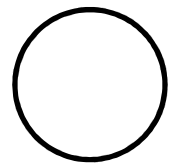


○ trocou

trocou apagada: vetor ordenado

V

0	1	2	3	4	5	6	7	8	9
5	8	16	45	67	70	72	74	80	95

 trocou

A função **BubbleSort**:

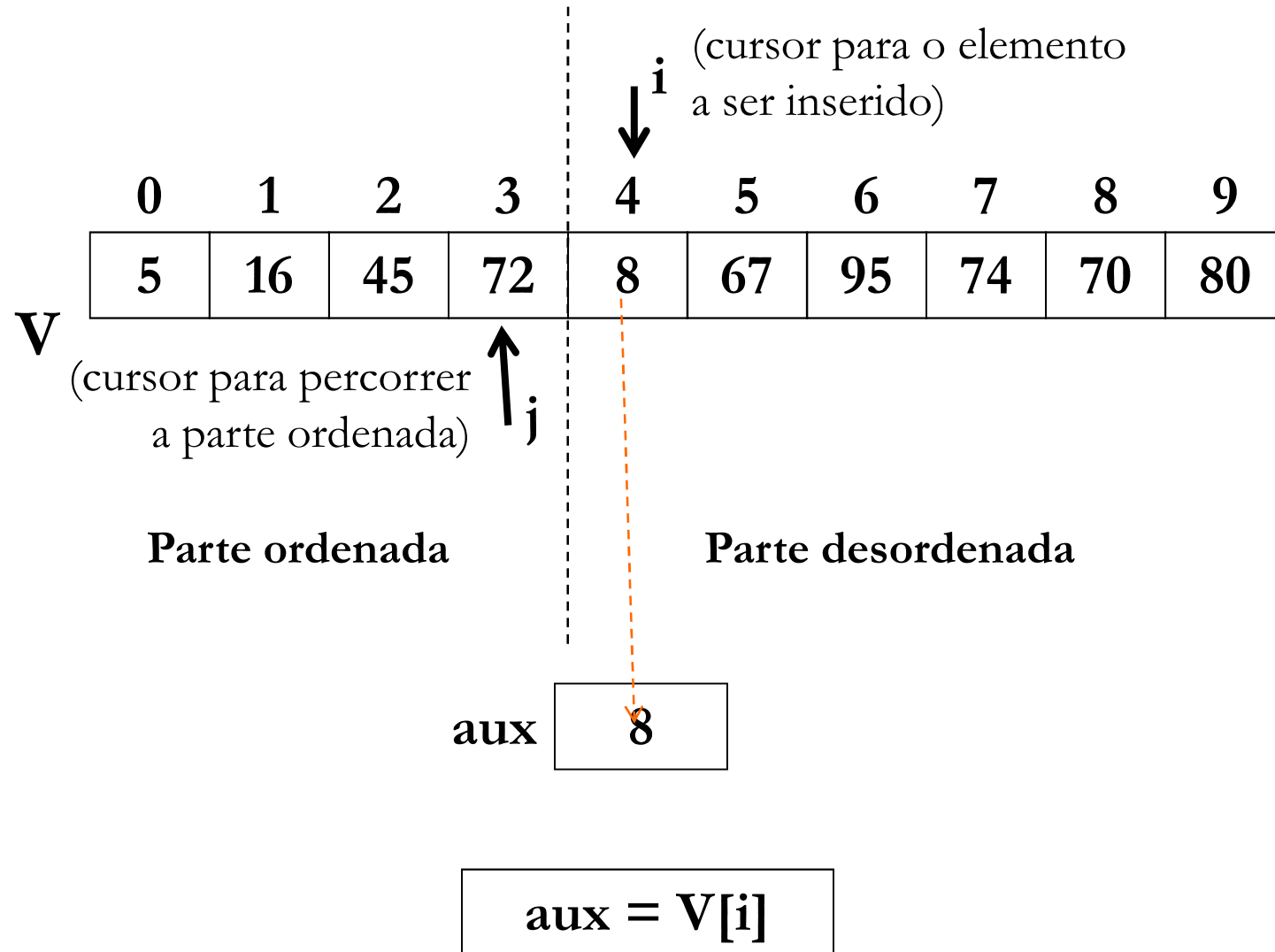
```
void BubbleSort (int n, vetor V) {  
    int i, p;  
    logic trocou;  
    for (p = n-2, trocou = TRUE; p >= 0 && trocou; p--)  
        for (trocou = FALSE, i = 0; i <= p; i++)  
            if (V[i] > V[i+1]) {  
                Trocar (&V[i], &V[i+1]);  
                trocou = TRUE;  
            }  
}  
  
void Trocar (int *x, int *y) {  
    int temp;  
    temp = *x; *x = *y; *y = temp;  
}
```

A cada vez que a condição do if se verificar, haverá movimentação de 3 elementos

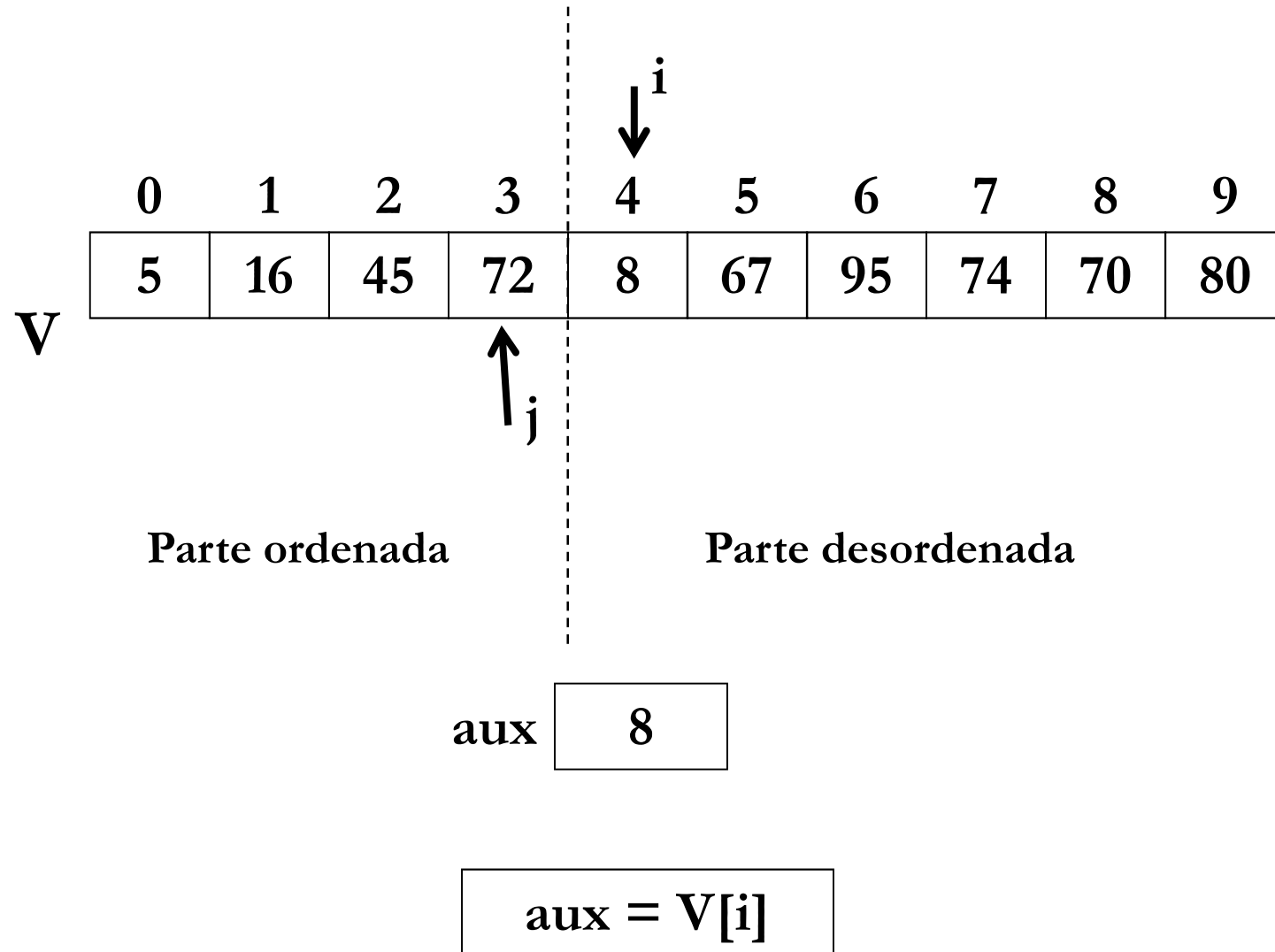
5.2.2 – Insertion-Sort

- O método considera o vetor sempre **dividido** em duas partes: a parte **ordenada**, do lado esquerdo, e a parte **desordenada**, do lado direito.
- Inicialmente, a parte **ordenada** contém apenas **V[0]**; depois, cada elemento da parte desordenada é inserido na parte ordenada, mantendo a ordenação
- A cada elemento a ser inserido, percorre-se a parte ordenada até encontrar a **posição correta** para inserção.

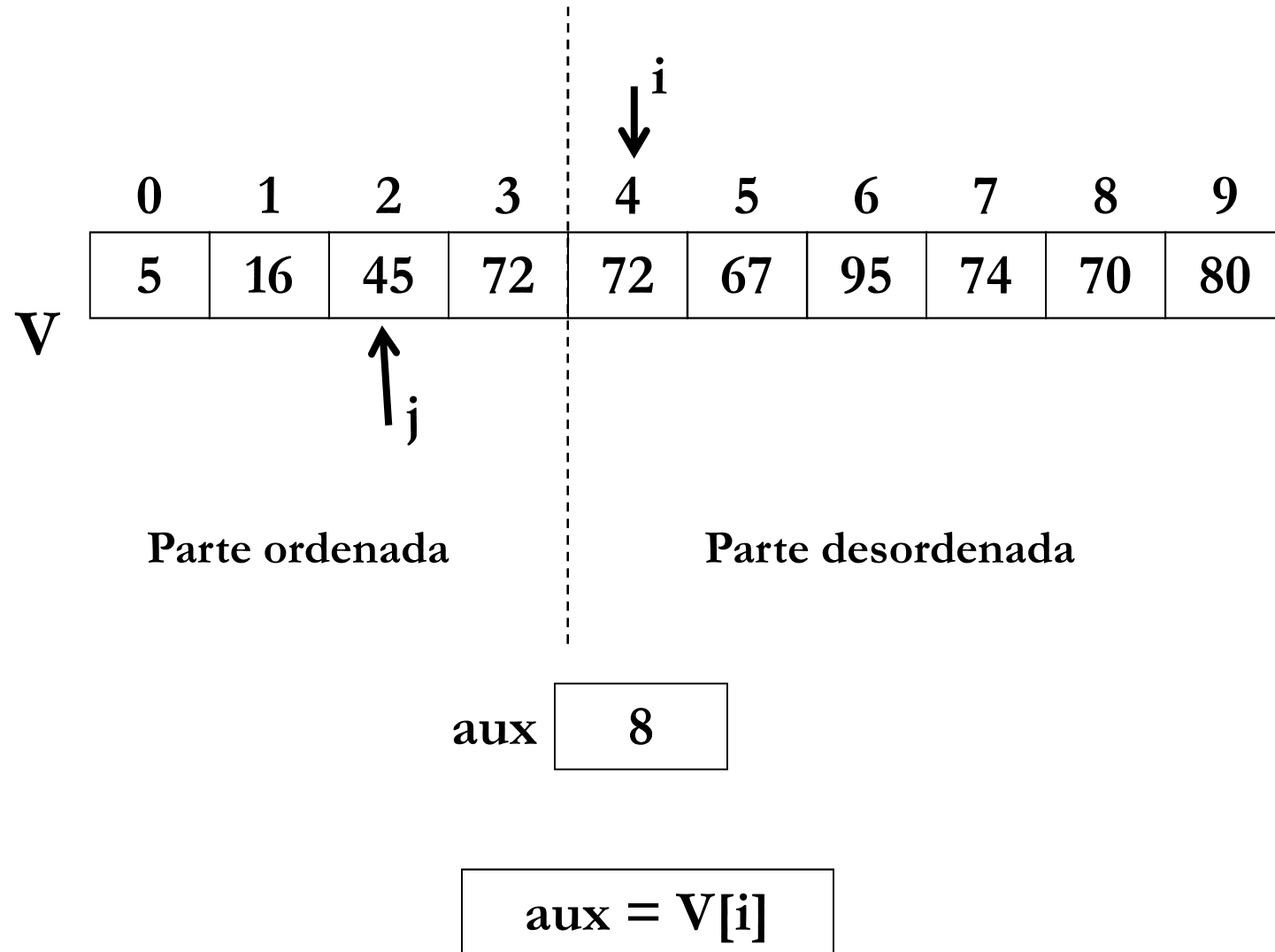
Exemplo: seja o vetor abaixo e o método no início da inserção de um elemento genérico



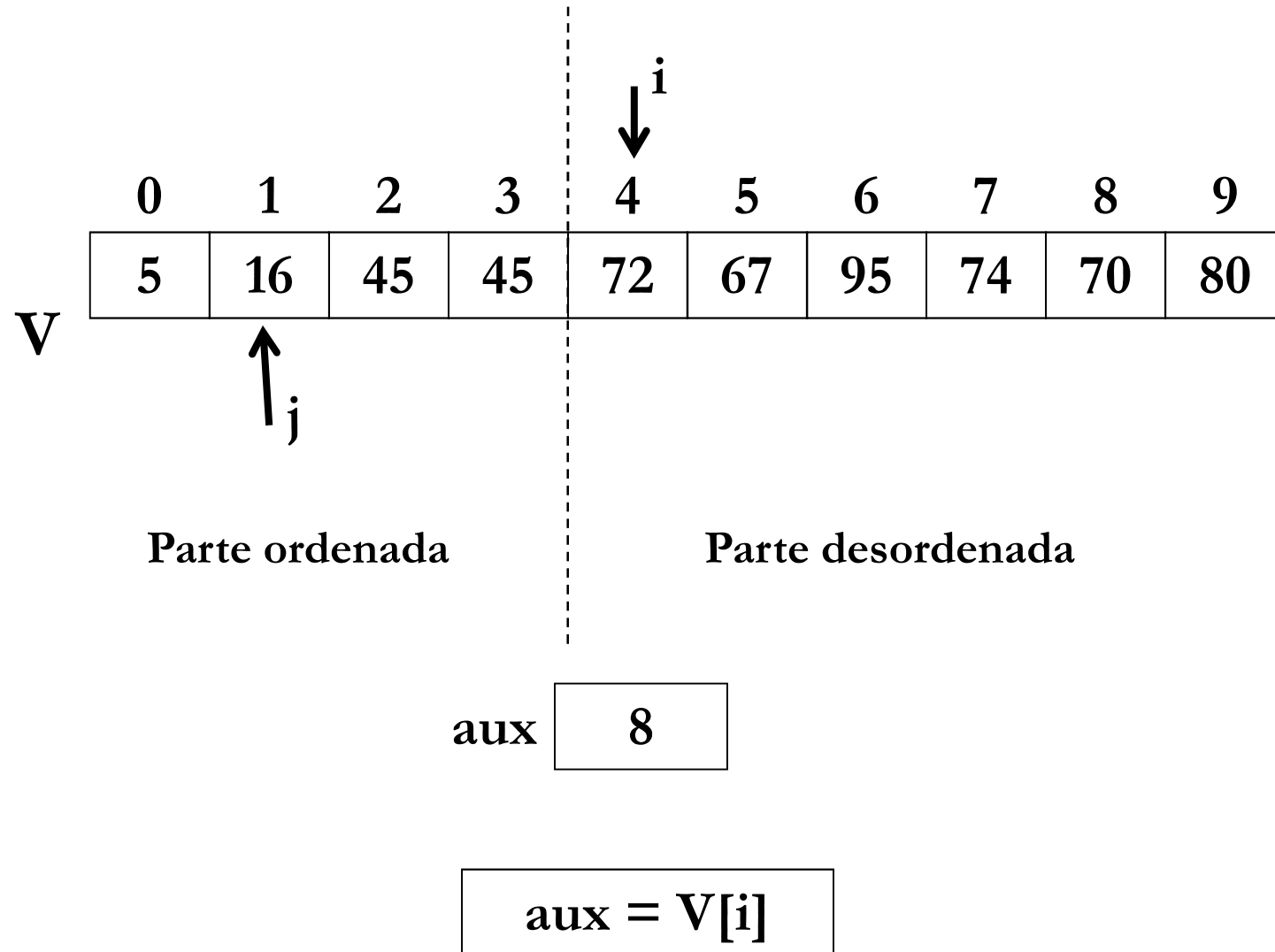
$\text{aux} < V[j]$: $V[j+1]$ recibe $V[j]$; retroceder j



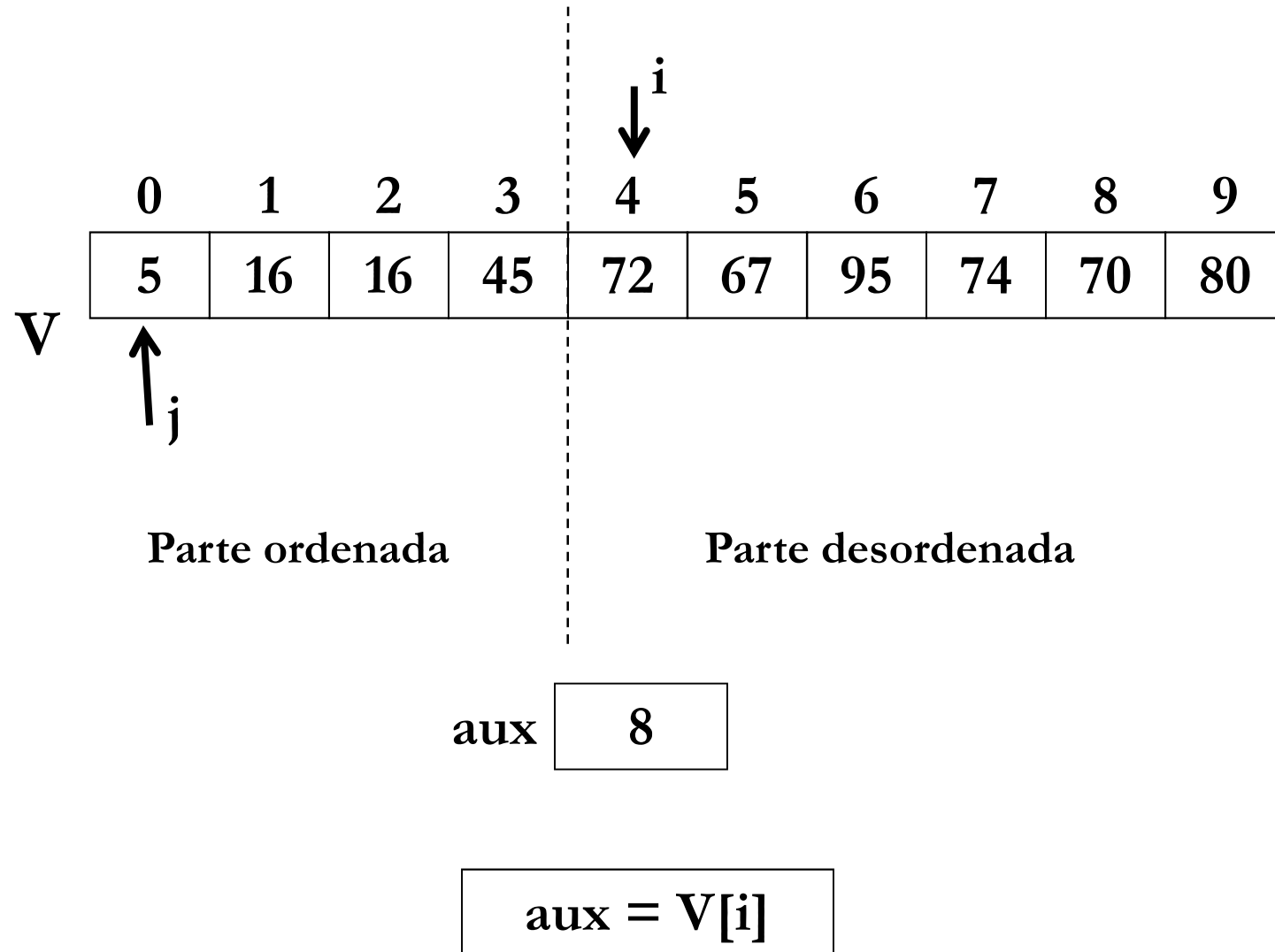
$\text{aux} < V[j]$: $V[j+1]$ recibe $V[j]$; retroceder j

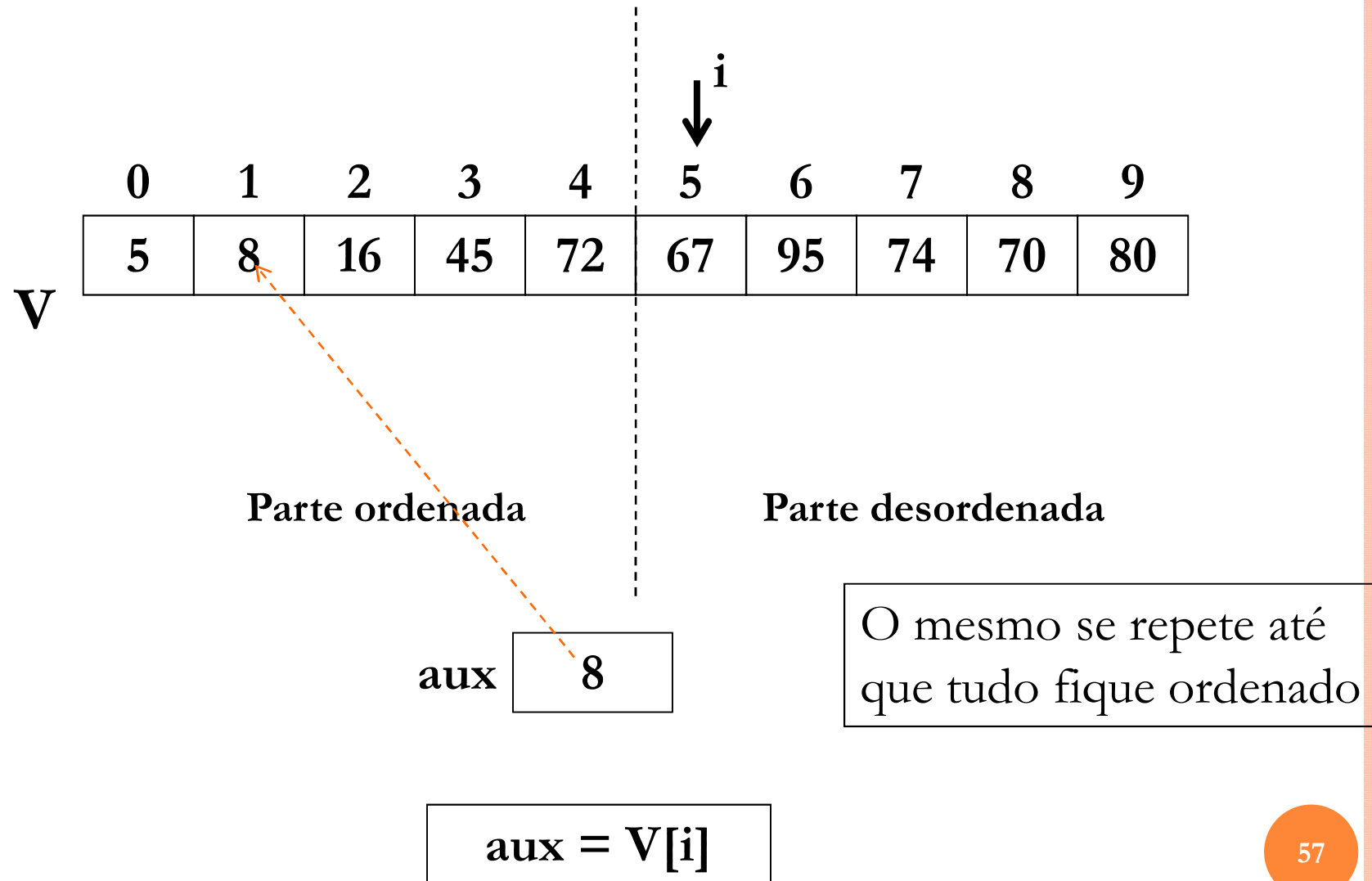


$\text{aux} < V[j]$: $V[j+1]$ recibe $V[j]$; retroceder j



aux >= V[j]: V[j+1] recebe aux; avançar a parte ordenada





A função InsertionSort:

```
void InsertionSort (int n, vetor V) {  
    int i, j, aux;  
    int achou;  
    for (i = 1; i < n; i++) {  
        aux = V[i];  
        achou = FALSE; j = i-1;  
        while (j >= 0 && !achou)  
            if (aux < V[j]) {  
                V[j+1] = V[j];  
                j--;  
            }  
        else achou = TRUE;  
        if (j+1 != i)  
            V[j+1] = aux;  
    }  
}
```

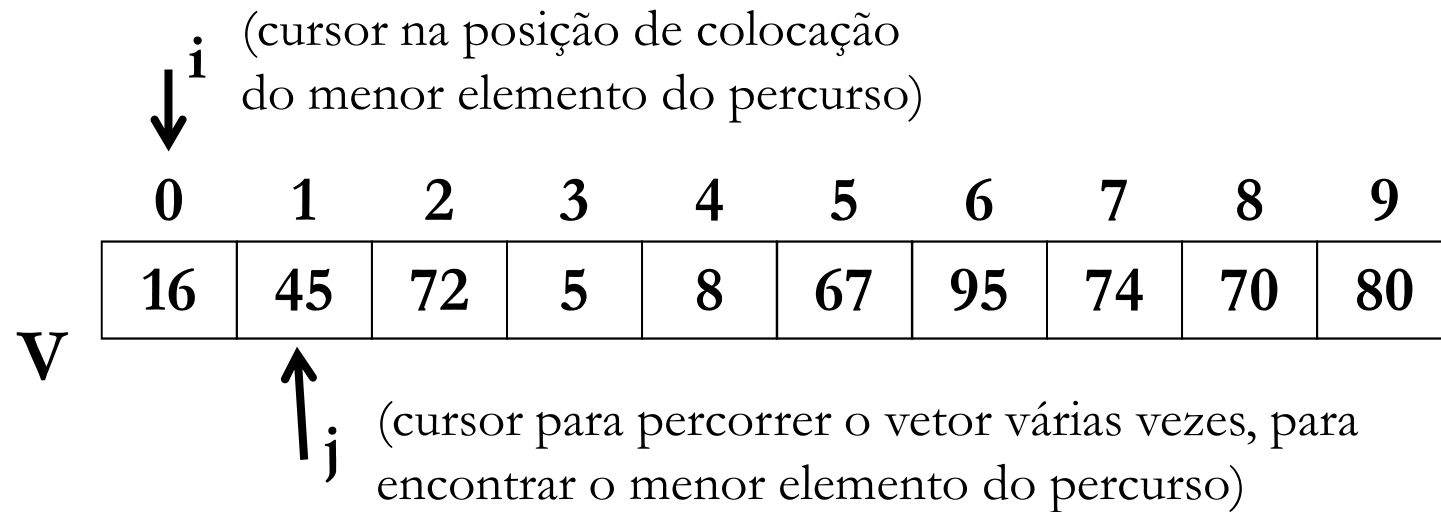
A cada vez que a condição do if se verificar, haverá movimentação de 1 elemento

Antes e depois do while
mais uma

5.2.3 – Selection-Sort

- O método consiste em **percorrer o vetor** várias vezes
- Durante cada percurso, **seleciona o menor elemento**, colocando-o em sua **posição definitiva**.

Exemplo: seja o vetor abaixo e o método em seu início



indmin

(destinado a guardar o índice do menor elemento de um percurso)

Após o 1º percurso:

indmin $\neq$ i: trocar V[i] com V[indmin]

V

i ↓ 0	1	2	3	4	5	6	7	8	9
16	45	72	5	8	67	95	74	70	80

indmin

3

Avançar **i**

Após o 2º percurso:

$\text{indmin} \neq i$: trocar $V[i]$ com $V[\text{indmin}]$

V

0	1	2	3	4	5	6	7	8	9
5	45	72	16	8	67	95	74	70	80

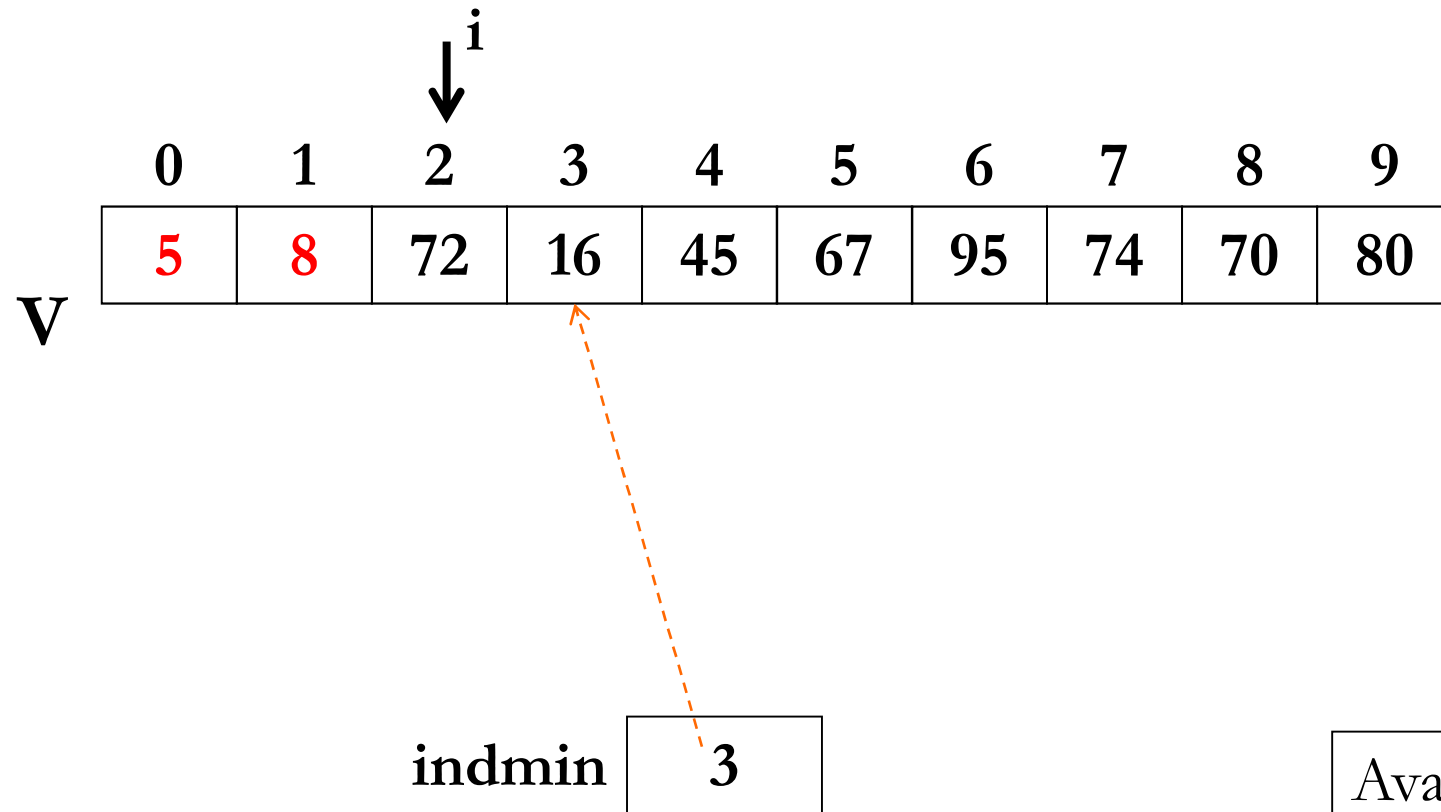
indmin

4

Avançar **i**

Após o 3º percurso:

$\text{indmin} \neq i$: trocar $V[i]$ com $V[\text{indmin}]$



V

0	1	2	3	4	5	6	7	8	9
5	8	16	72	45	67	95	74	70	80

↓ⁱ

indmin

E assim por diante,
até que o vetor fique
ordenado

A função **SelectionSort**:

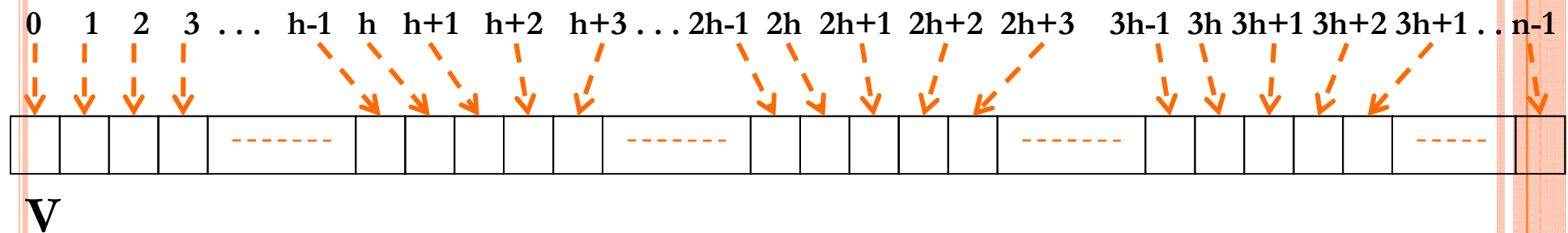
```
void SelectionSort (int n, vetor V) {  
    int i, j, indmin;  
    for (i = 0; i < n-1; i++) {  
        indmin = i;  
        for (j = i+1; j <= n-1; j++)  
            if (V[j] < V[indmin])  
                indmin = j;  
        if (indmin != i)  
            Trocar (&V[i], &V[indmin]);  
    }  
}
```

Só ocorre movimentação de elementos quando o índice do menor for encontrado

5.3 – O MÉTODO SHELL-SORT

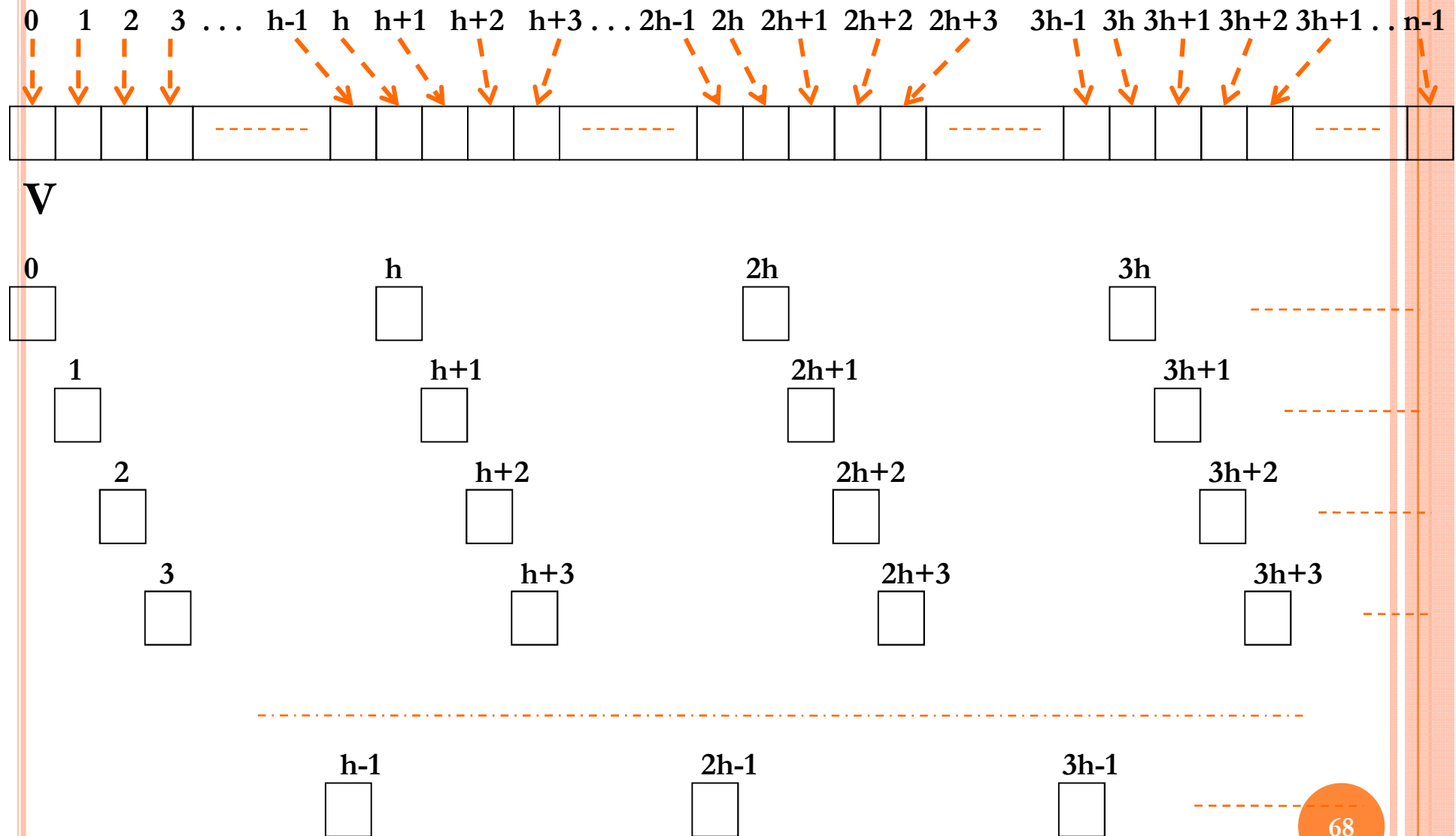
- O método Shell-Sort é uma **generalização** do **Insertion-Sort**.
- No Insertion-Sort, a maioria dos **deslocamentos** de elementos é de apenas **uma casa**.
- Se o **menor** elemento estiver no **final** do vetor, são necessários **n deslocamentos** para colocá-lo ordenadamente.
- **Shell-Sort** faz movimentos de **passos maiores**

- O vetor $\mathbf{V}$ pode ser assim visualizado:

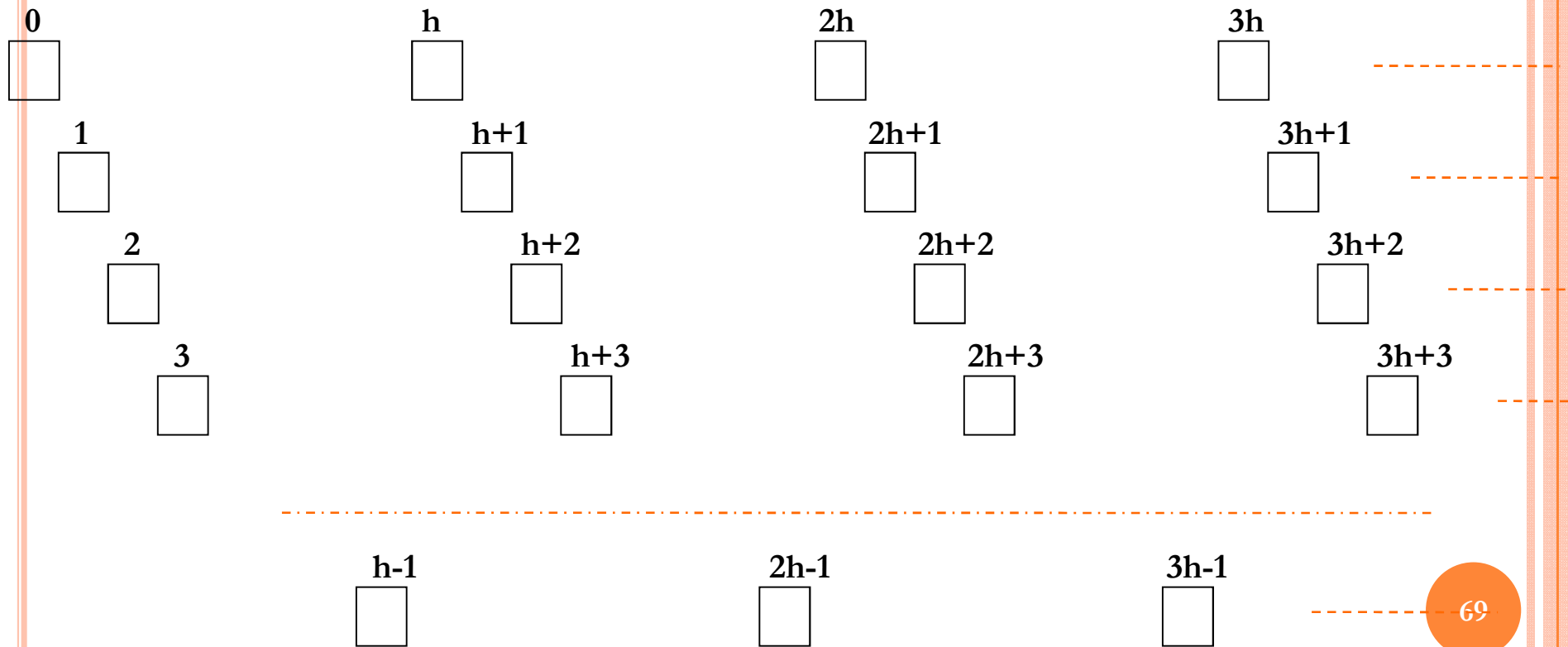


- O valor de $h < n$ é empírico

- Pode-se dividir V em h sub-vetores intercalados:

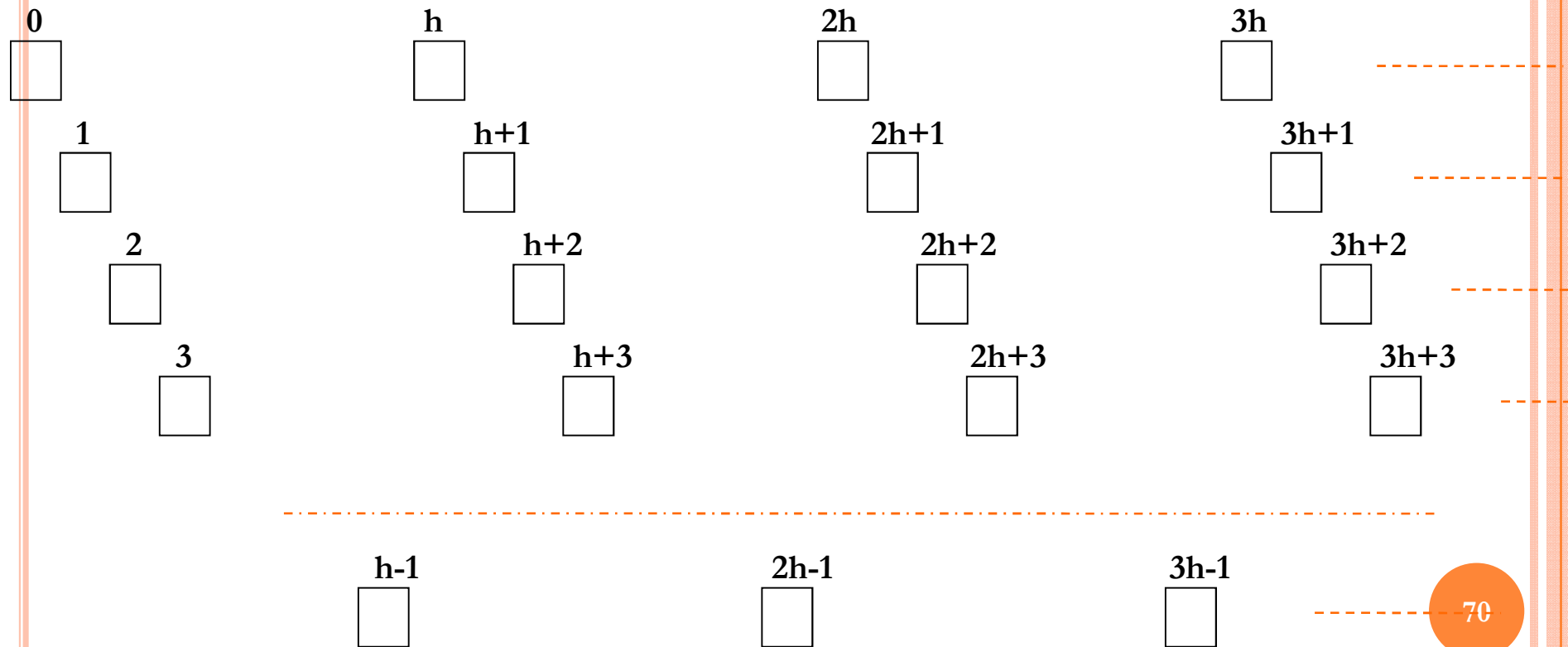


- O valor de **h** é inicialmente grande, mas vai caindo até 1
- Para cada valor de **h** , ordena-se cada um desses subvetores pelo Insertion-Sort

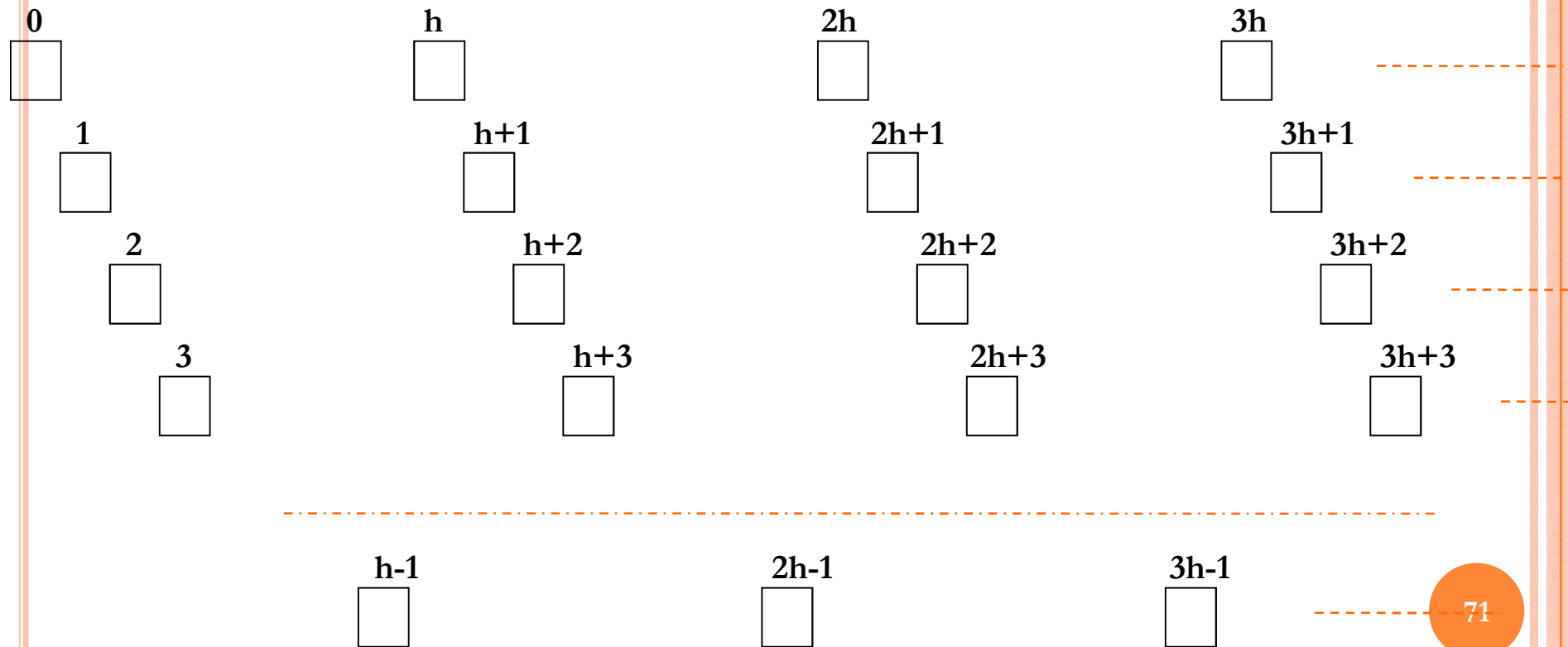


○ Quando $h = 1$, tem-se o **Insertion-Sort original**:

- Então **boa porção** do vetor já estará **ordenada** pelos passos anteriores (maiores valores de h)



- O valor inicial de h e a razão de seu **decrécimo** é escolhida **empiricamente**; para cada escolha, um **comportamento** diferente.



- O algoritmo a seguir escolhe a seguinte seqüência de valores de h , que tem dado bons valores de desempenho:

$\dots, 1093, 364, 121, 40, 13, 4, 1$

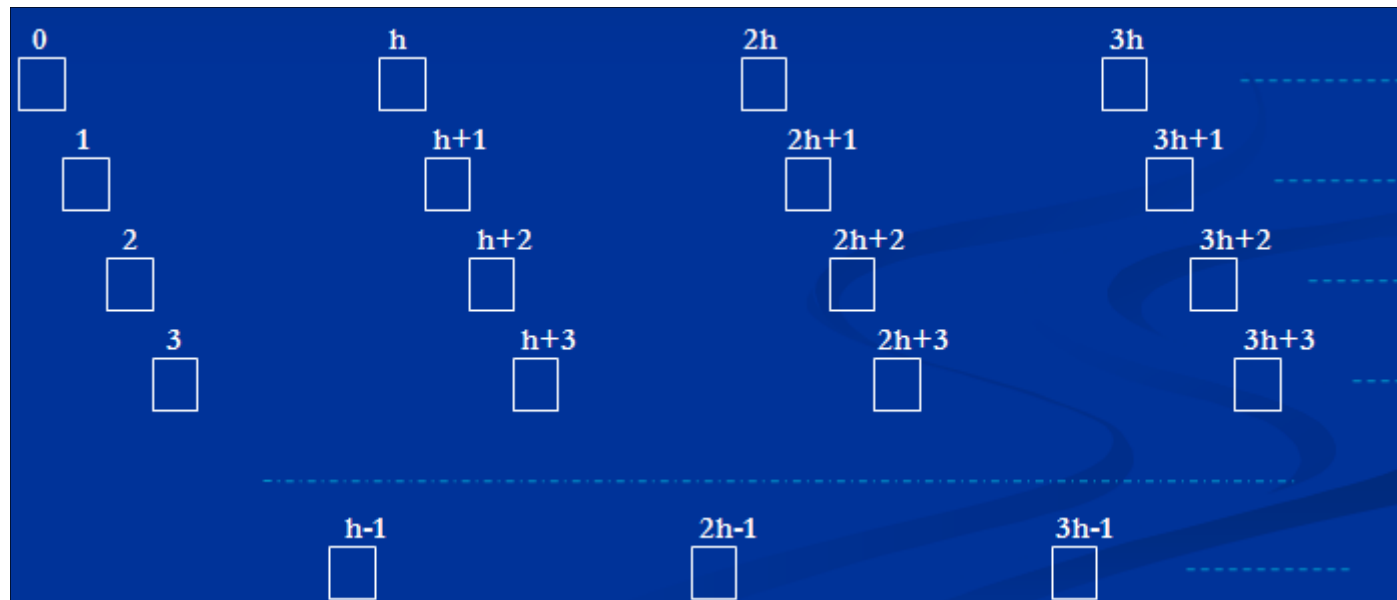
O **valor inicial** é obtido pela fórmula

$$h(\text{novo}) = 3 * h(\text{antigo}) + 1$$

até encontrar o maior valor de $h < n$

Os valores seguintes de h são obtidos pela fórmula

$$h(\text{novo}) = h(\text{antigo}) / 3$$



```
void ShellSort (int n, vetor V) {  
    int i, j, h, aux; int achou;  
  
    h = 1;  
    do h = 3*h + 1; while (h < n);  
    do {  
        h = h/3;  
        for (i = h; i <= n-1; i++) {  
            aux = V[i]; j = i; achou = FALSE;  
            while (!achou && j >= h)  
                if (aux < V[j-h]) {  
                    V[j] = V[j-h]; j = j - h;  
                }  
            else achou = TRUE;  
            if (j != i) V[j] = aux;  
        }  
    } while (h > 1);  
}
```

Insertion-Sort de
passo **h**, intercalado

- **Comparação** entre o número de movimentos de elementos no Insertion-Sort e Shell-Sort, usando vetor de caracteres:

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

1

nº acumulado
movtos

1

Sempre há um
movimento para aux

S -> aux

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

3

nº acumulado
movtos

4

```
O -> aux;  
S -> 2  
aux -> 1
```

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	O	S	R	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

3

nº acumulado
movtos

7

```
R -> aux;  
S -> 3  
aux -> 2
```

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	O	R	S	T	I	N	G	E	X	A	M	P	L	E

aux

T -> aux;

nº movtos

1

nº acumulado
movtos

8

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	O	R	S	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

6

nº acumulado
movtos

14

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	I	O	R	S	T	N	G	E	X	A	M	P	L	E

aux

nº movtos

6

nº acumulado
movtos

20

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	I	N	O	R	S	T	G	E	X	A	M	P	L	E

aux

nº movtos

8

nº acumulado
movtos

28

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	G	I	N	O	R	S	T	E	X	A	M	P	L	E

aux

nº movtos

9

nº acumulado
movtos

37

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	G	I	N	O	R	S	T	X	A	M	P	L	E

aux

nº movtos

1

nº acumulado
movtos

38

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	G	I	N	O	R	S	T	X	A	M	P	L	E

aux

nº movtos

11

nº acumulado
movtos

49

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	I	N	O	R	S	T	X	M	P	L	E

aux

nº movtos

8

nº acumulado
movtos

57

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	I	M	N	O	R	S	T	X	P	L	E

aux

nº movtos

6

nº acumulado
movtos

63

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	I	M	N	O	P	R	S	T	X	L	E

aux

nº movtos

10

nº acumulado
movtos

73

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	I	L	M	N	O	P	R	S	T	X	E

aux

nº movtos

13

nº acumulado
movtos

86

a) Insertion-Sort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

aux

nº movtos

nº acumulado
movtos

86

b) Shell-Sort:

Ordenação para $h = 13$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
1			1

b) Shell-Sort:

Ordenação para $h = 13$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E

aux

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4			4

b) Shell-Sort:

Fim da ordenação para $h = 13$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S

aux

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4			4

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	1		5

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	2		6

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	5		9

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	N	R	T	I	O	G	E	X	A	M	P	L	S

aux

--

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	8		12

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	N	G	T	I	O	R	E	X	A	M	P	L	S

aux

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	11		15

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	N	G	E	I	O	R	T	X	A	M	P	L	S

aux

nº movtos

1

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	12		16

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	N	G	E	I	O	R	T	X	A	M	P	L	S

aux

--

nº movtos

4

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	16		20

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	R	T	X	O	M	P	L	S

aux

--

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	19		23

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	T	X	O	R	P	L	S

aux

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	22		26

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	P	X	O	R	T	L	S

aux

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	25		29

b) Shell-Sort:

Ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S

aux

--

nº movtos

1

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26		30

b) Shell-Sort:

Fim da ordenação para $h = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26		30

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	1	31

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S

aux

--

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	4	34

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	E	I	N	M	P	L	O	R	T	X	S

aux

--

nº movtos

1

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	5	35

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	G	E	I	N	M	P	L	O	R	T	X	S

aux

--

nº movtos

3

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	8	38

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	N	M	P	L	O	R	T	X	S

aux

nº movtos

5

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	13	43

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	M	N	P	L	O	R	T	X	S

aux

nº movtos

6

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	19	49

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	L	M	N	P	O	R	T	X	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	22	52

b) Shell-Sort:

Ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	L	M	N	O	P	R	T	X	S

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	29	59

b) Shell-Sort:

Fim da ordenação para $h = 1$ (Insertion-Sort)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

aux

nº movtos

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	29	59

Insertion-Sort:

nº acumulado
de movtos

86

Shell-Sort:

nº acumulado de movtos			
$h = 13$	$h = 4$	$h = 1$	total
4	26	29	59

- No Shell-Sort, o número de movimentos para $h = 1$ é 29
- **(86-29=57)** 57 movimentos do Insertion-Sort foram compensados com **(59-29=30)** 30 movimentos do Shell-Sort com $h > 1$.

Observações:

- Para vetor inicialmente **já ordenado**, Shell-Sort é pior que Insertion-Sort, devido ao número de **comparações**
- **Outros métodos** simples de ordenação poderiam ser usados para cada valor de **h** , mas o mais usado é o **Insertion-Sort**.
- Há quem use um método para **$h > 1$** e outro para **$h = 1$**

Observações:

- Referências bibliográficas indicam que o algoritmo anterior é $O(n (\log n)^2)$ e também $O(n^{1.5})$.
- Donald Knuth prova que, mesmo usando apenas **2 valores de h**, a saber,

$$(16 n / \pi)^{1/3} \text{ e } 1$$

O método resultante é $O(n^{5/3})$, o que é melhor que o pior caso do Insertion-Sort, que é $O(n^2)$.

- O tempo de execução do algoritmo apresentado **não** é muito **sensível ao estado inicial** do vetor, ao contrário do **Insertion-Sort** que é $O(n)$ para vetor já ordenado e $O(n^2)$ para vetor ordenado inversamente.