

MCTA001-13-Algoritmos e Estruturas de Dados I

---

## Aula 08

# Algoritmos de Ordenação de Vetores

Marcio K. Oikawa  
([marcio.oikawa@ufabc.edu.br](mailto:marcio.oikawa@ufabc.edu.br))

1o. Quadrimestre de 2020



## Introdução

---

- Ordenação é uma operação básica de conjuntos de dados.
- A chave de ordenação é um valor (ou conjunto de valores) usado como referência para comparação.
- A chave de ordenação deve pertencer a um conjunto com a propriedade de ordem parcial.

## Problema de Ordenação (*Sorting*)

---

- Considere um vetor  $v[0..n-1]$  de  $n$  elementos,  $n > 1$ , dispostos de forma aleatória. Um algoritmo de ordenação deverá realizar permutações entre esses elementos de modo que, ao final, tenhamos válida a seguinte propriedade:  
$$v[0] \leq v[1] \leq v[2] \leq \dots \leq v[n-1]$$
- Os algoritmos clássicos são iterativos, buscando ordenar parcialmente o vetor a cada iteração.

## Exemplo

Vetor original:

|   |    |   |   |   |    |   |    |   |    |   |
|---|----|---|---|---|----|---|----|---|----|---|
| V | 15 | 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
|   | 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

Vetor ordenado:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 1 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

## *Bubble Sort*

---

- O *Bubble Sort* se baseia na troca de elementos vizinhos do vetor, deslocando os menores para a esquerda e os maiores para a direita.
- O nome *Bubble Sort* é dado porque a dinâmica do algoritmo leva elementos maiores a assumir suas posições mais rápido que elementos menores.

## Bubble Sort

Vetor original:

|   |    |   |   |   |    |   |    |   |    |   |
|---|----|---|---|---|----|---|----|---|----|---|
| V | 15 | 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
|   | 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

Primeira iteração:

|   |   |    |   |   |    |   |    |   |    |   |
|---|---|----|---|---|----|---|----|---|----|---|
| V | 8 | 15 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
|   | 0 | 1  | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

## Bubble Sort


V

|   |    |   |   |    |   |    |   |    |   |
|---|----|---|---|----|---|----|---|----|---|
| 8 | 15 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1  | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |    |   |    |   |    |   |    |   |
|---|---|----|---|----|---|----|---|----|---|
| 8 | 9 | 15 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2  | 3 | 4  | 5 | 6  | 7 | 8  | 9 |



*Bubble Sort*


V

|   |   |    |   |    |   |    |   |    |   |
|---|---|----|---|----|---|----|---|----|---|
| 8 | 9 | 15 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2  | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
| 8 | 9 | 5 | 15 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |





## Bubble Sort


V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
| 8 | 9 | 5 | 15 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
| 8 | 9 | 5 | -3 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |



## Bubble Sort


V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
| 8 | 9 | 5 | -3 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |   |    |    |   |    |   |
|---|---|---|----|---|----|----|---|----|---|
| 8 | 9 | 5 | -3 | 4 | 15 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6  | 7 | 8  | 9 |



## Bubble Sort


V

|   |   |   |    |   |    |    |   |    |   |
|---|---|---|----|---|----|----|---|----|---|
| 8 | 9 | 5 | -3 | 4 | 15 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |   |    |    |   |    |   |
|---|---|---|----|---|----|----|---|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 15 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6  | 7 | 8  | 9 |



## Bubble Sort


V

|   |   |   |    |   |    |    |   |    |   |
|---|---|---|----|---|----|----|---|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 15 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6  | 7 | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |   |    |   |    |    |   |
|---|---|---|----|---|----|---|----|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 7 | 15 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6 | 7  | 8  | 9 |



## Bubble Sort


V

|   |   |   |    |   |    |   |    |    |   |
|---|---|---|----|---|----|---|----|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 7 | 15 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6 | 7  | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |   |    |   |    |    |   |
|---|---|---|----|---|----|---|----|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 15 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6 | 7  | 8  | 9 |



## Bubble Sort


V

|   |   |   |    |   |    |   |    |    |   |
|---|---|---|----|---|----|---|----|----|---|
| 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 15 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6 | 7  | 8  | 9 |

Primeira iteração:

V

|   |   |   |    |   |    |   |    |   |    |
|---|---|---|----|---|----|---|----|---|----|
| 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 | 15 |
| 0 | 1 | 2 | 3  | 4 | 5  | 6 | 7  | 8 | 9  |



## *Bubble Sort*

---

- O vetor ainda não está ordenado, porém é possível perceber alguma variação em relação ao original:
  - O maior valor sempre atinge sua posição definitiva na primeira iteração.
  - Se o processo for repetido um número suficiente de vezes, é possível garantir que o vetor ficará ordenado.

## Bubble Sort

Segunda iteração:

|   |   |   |    |   |   |   |    |   |    |    |
|---|---|---|----|---|---|---|----|---|----|----|
| V | 8 | 5 | -3 | 4 | 9 | 7 | 10 | 1 | 12 | 15 |
|   | 0 | 1 | 2  | 3 | 4 | 5 | 6  | 7 | 8  | 9  |

Terceira iteração:

|   |   |    |   |   |   |   |   |    |    |    |
|---|---|----|---|---|---|---|---|----|----|----|
| V | 5 | -3 | 4 | 8 | 7 | 9 | 1 | 10 | 12 | 15 |
|   | 0 | 1  | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |



## Bubble Sort

Quarta iteração:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 4 | 5 | 7 | 8 | 1 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

Quinta iteração:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 4 | 5 | 7 | 1 | 8 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

## Bubble Sort

Sexta iteração:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 4 | 5 | 1 | 7 | 8 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

Sétima iteração:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 4 | 1 | 5 | 6 | 8 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

## Bubble Sort

---

Oitava iteração:

|   |    |   |   |   |   |   |   |    |    |    |
|---|----|---|---|---|---|---|---|----|----|----|
| V | -3 | 1 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 |
|   | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

## Bubble Sort

Função auxiliar para trocar valores de posição dentro do vetor:

```
void troca (int* v, int i, int j) {  
    int tmp;  
  
    tmp = v[i];  
    v[i] = v[j];  
    v[j] = tmp;  
  
    return;  
}
```

## Bubble Sort

Primeira versão do *bubble sort*:

```
void bubblesort (int* v, int tam){  
    int i, j;  
    for (i=0; i<tam; i++){  
        for (j=0; j<tam-1; j++){  
            if (v[j] > v[j+1]){  
                troca (v, j, j+1);  
            }  
        }  
    }  
}
```

## *Bubble Sort*

---

- Esta versão não é muito eficiente, pois faz várias comparações desnecessárias.
- Como vimos, a cada iteração (interna), os maiores elementos assumem suas posições definitivas. Não temos necessidade de testá-los novamente.

## Bubble Sort

Nova versão do *bubble sort*:

```
void bubblesort (int* v, int tam){  
    int i, j;  
    for (i=0; i<tam; i++){  
        for (j=0; j<tam-i-1; j++){  
            if (v[j] > v[j+1]){  
                troca (v, j, j+1);  
            }  
        }  
    }  
}
```

## *Selection Sort*

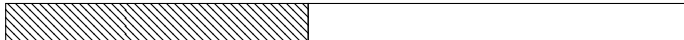
---

- O algoritmo de ordenação por seleção também ordena o vetor de forma iterativa, buscando os menores elementos e os colocando em suas posições definitivas.



## *Selection Sort*

- A ordenação é realizada progressivamente a cada iteração:



← subvetor ordenado —×— subvetor não-ordenado →

## Selection Sort

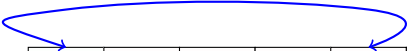
Menor elemento não-ordenado: -3

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| 15 | 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 8 | 9 | 5 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |



## Selection Sort


Menor elemento não-ordenado: 1

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 8 | 9 | 5 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 9 | 5 | 15 | 4 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |



## Selection Sort

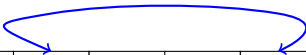
Menor elemento não-ordenado: 4

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 9 | 5 | 15 | 4 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 4 | 5 | 15 | 9 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |



## Selection Sort


Menor elemento não-ordenado: 5

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 4 | 5 | 15 | 9 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 4 | 5 | 15 | 9 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |



## Selection Sort

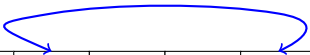
Menor elemento não-ordenado: 7

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 1 | 4 | 5 | 15 | 9 | 12 | 7 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

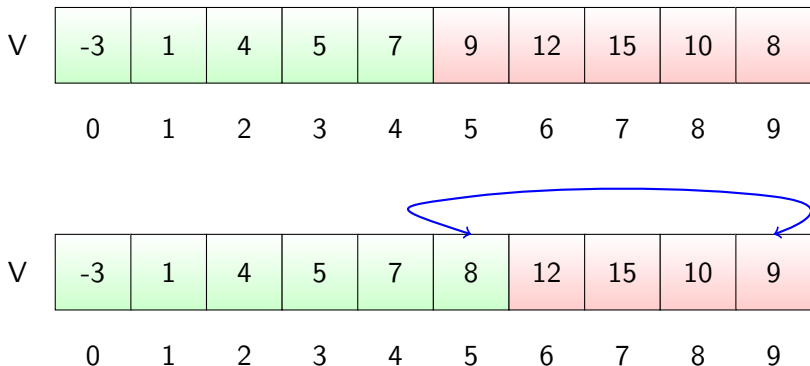
V

|    |   |   |   |   |   |    |    |    |   |
|----|---|---|---|---|---|----|----|----|---|
| -3 | 1 | 4 | 5 | 7 | 9 | 12 | 15 | 10 | 8 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9 |



## Selection Sort

Menor elemento não-ordenado: 8



## Selection Sort


Menor elemento não-ordenado: 9

V

|    |   |   |   |   |   |    |    |    |   |
|----|---|---|---|---|---|----|----|----|---|
| -3 | 1 | 4 | 5 | 7 | 8 | 12 | 15 | 10 | 9 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9 |

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 | 1 | 4 | 5 | 7 | 8 | 9 | 15 | 10 | 12 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |





## Selection Sort


Menor elemento não-ordenado: 10

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 | 1 | 4 | 5 | 7 | 8 | 9 | 15 | 10 | 12 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 | 1 | 4 | 5 | 7 | 8 | 9 | 10 | 15 | 12 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |



## Selection Sort


Menor elemento não-ordenado: 12

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 | 1 | 4 | 5 | 7 | 8 | 9 | 10 | 15 | 12 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 | 1 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |



## Selection Sort

Implementação do *selection sort*:

```
/* Devolve o indice do menor elemento do
   sub-vetor v[a..b] */
int menor (int* v, int a, int b){
    int i,min = a;
    for (i=a+1; i<=b; i++){
        if (v[i] < v[min]){
            min = i;
        }
    }
    return min;
}
```

## Selection Sort

Implementação do *selection sort*:

```
void selectionsort (int* v, int tam){  
    int i,min;  
    for (i=0; i<tam; i++){  
        min = menor (v,i,tam-1);  
        troca (v,min,i);  
    }  
    return;  
}
```

## *Insertion Sort*

---

- O algoritmo de ordenação por inserção é similar à forma como organizamos as cartas em um jogo de baralho.
- O algoritmo também é iterativo e, a cada iteração, busca-se a posição mais adequada a um elemento, deslocando os seus antecessores no vetor.

## Insertion Sort

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| 15 | 8 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

8

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| 15 |   | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

8

V

|   |    |   |   |    |   |    |   |    |   |
|---|----|---|---|----|---|----|---|----|---|
|   | 15 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1  | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

## Insertion Sort

V

|   |    |   |   |    |   |    |   |    |   |
|---|----|---|---|----|---|----|---|----|---|
| 8 | 15 | 9 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1  | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

9

V

|   |    |   |   |    |   |    |   |    |   |
|---|----|---|---|----|---|----|---|----|---|
| 8 | 15 |   | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1  | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

9

V

|   |   |    |   |    |   |    |   |    |   |
|---|---|----|---|----|---|----|---|----|---|
| 8 |   | 15 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2  | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

## Insertion Sort

V

|   |   |    |   |    |   |    |   |    |   |
|---|---|----|---|----|---|----|---|----|---|
| 8 | 9 | 15 | 5 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2  | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

5

V

|   |   |    |   |    |   |    |   |    |   |
|---|---|----|---|----|---|----|---|----|---|
| 8 | 9 | 15 |   | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2  | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

5

V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
|   | 8 | 9 | 15 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |



## Insertion Sort

V

|   |   |   |    |    |   |    |   |    |   |
|---|---|---|----|----|---|----|---|----|---|
| 5 | 8 | 9 | 15 | -3 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8  | 9 |

-3

V

|   |   |   |    |   |   |    |   |    |   |
|---|---|---|----|---|---|----|---|----|---|
| 5 | 8 | 9 | 15 |   | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3  | 4 | 5 | 6  | 7 | 8  | 9 |

-3

V

|   |   |   |   |    |   |    |   |    |   |
|---|---|---|---|----|---|----|---|----|---|
|   | 5 | 8 | 9 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0 | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

## Insertion Sort

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 5 | 8 | 9 | 15 | 4 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

4

V

|    |   |   |   |    |   |    |   |    |   |
|----|---|---|---|----|---|----|---|----|---|
| -3 | 5 | 8 | 9 | 15 |   | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8  | 9 |

4

V

|    |   |   |   |   |    |    |   |    |   |
|----|---|---|---|---|----|----|---|----|---|
| -3 |   | 5 | 8 | 9 | 15 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6  | 7 | 8  | 9 |

## Insertion Sort

V

|    |   |   |   |   |    |    |   |    |   |
|----|---|---|---|---|----|----|---|----|---|
| -3 | 4 | 5 | 8 | 9 | 15 | 12 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6  | 7 | 8  | 9 |

12

V

|    |   |   |   |   |    |   |   |    |   |
|----|---|---|---|---|----|---|---|----|---|
| -3 | 4 | 5 | 8 | 9 | 15 |   | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8  | 9 |

12

V

|    |   |   |   |   |   |    |   |    |   |
|----|---|---|---|---|---|----|---|----|---|
| -3 | 4 | 5 | 8 | 9 |   | 15 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7 | 8  | 9 |

## Insertion Sort

V

|    |   |   |   |   |    |    |   |    |   |
|----|---|---|---|---|----|----|---|----|---|
| -3 | 4 | 5 | 8 | 9 | 12 | 15 | 7 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6  | 7 | 8  | 9 |

7

V

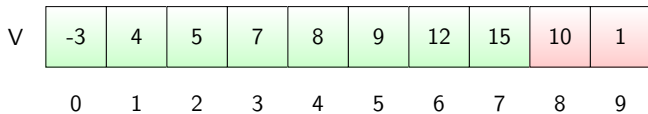
|    |   |   |   |   |    |    |   |    |   |
|----|---|---|---|---|----|----|---|----|---|
| -3 | 4 | 5 | 8 | 9 | 12 | 15 |   | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6  | 7 | 8  | 9 |

7

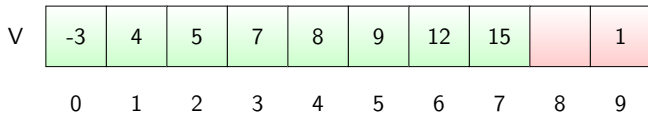
V

|    |   |   |   |   |   |    |    |    |   |
|----|---|---|---|---|---|----|----|----|---|
| -3 | 4 | 5 |   | 8 | 9 | 12 | 15 | 10 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9 |

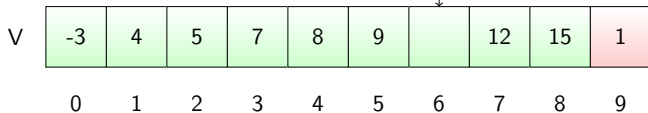
## Insertion Sort



10



10



## Insertion Sort

V

|    |   |   |   |   |   |    |    |    |   |
|----|---|---|---|---|---|----|----|----|---|
| -3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 | 1 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9 |

1

V

|    |   |   |   |   |   |    |    |    |   |
|----|---|---|---|---|---|----|----|----|---|
| -3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 |   |
| 0  | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9 |

1

V

|    |   |   |   |   |   |   |    |    |    |
|----|---|---|---|---|---|---|----|----|----|
| -3 |   | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 15 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7  | 8  | 9  |

## Insertion Sort

Implementação do *insertion sort*:

```
void insertionsort (int* v, int tam){  
    int i,j,tmp;  
    for (i=1; i<tam; i++){  
        tmp = v[i];  
        for (j=i-1; j>=0 && v[j] > tmp; j--){  
            v[j+1] = v[j];  
        }  
        v[j+1] = tmp;  
    }  
    return;  
}
```