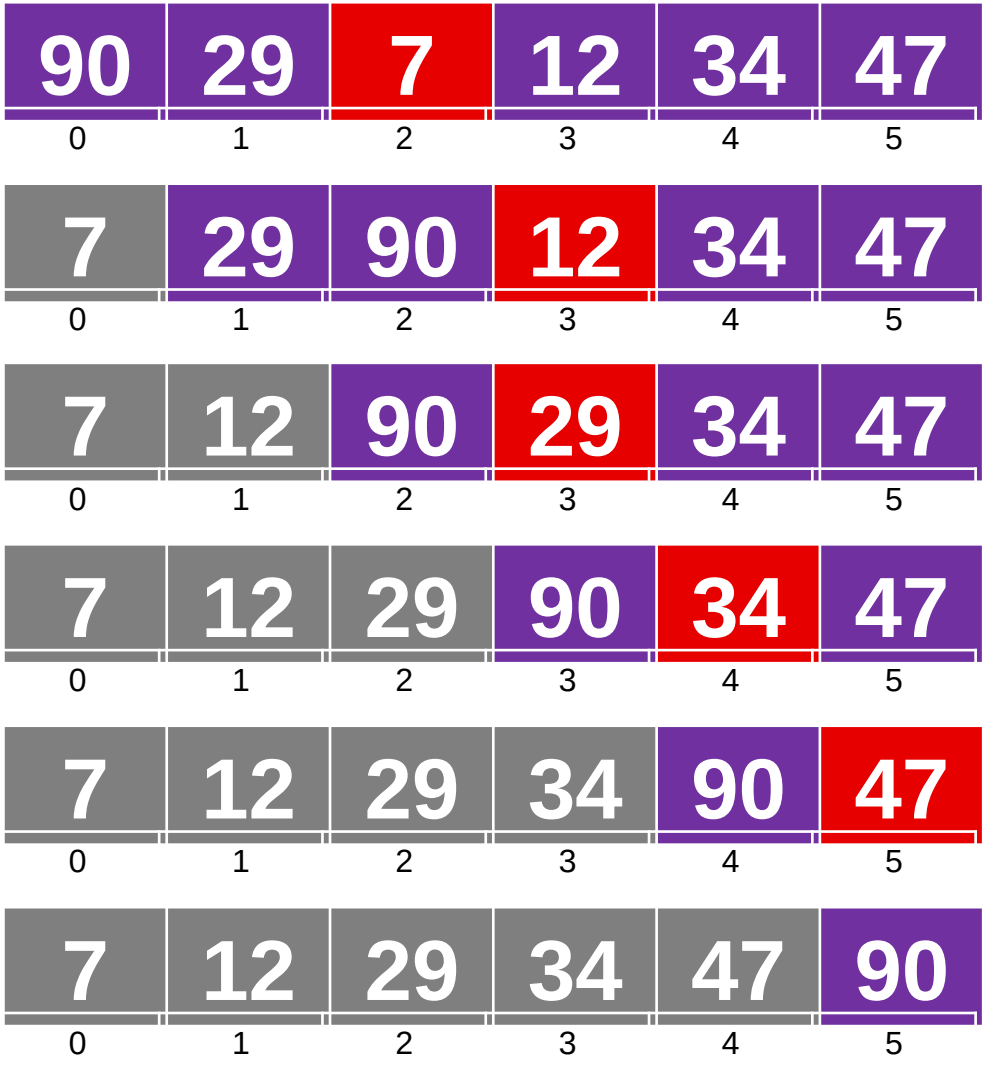
# Algoritmos e Estruturas de Dados 1

## Algoritmos de Ordenação

(visão inicial – seleção, inserção, troca)

# Ordenação por Seleção

A cada rodada, **seleciona** o menor e coloca na posição.



# Ordenação por Seleção

```
void SelectionSort (int v[], int n) {
    int i, j, min, aux;
    for (i = 0; i < n - 1; i++) { // v[min] vai para v[i]
            min = i;
            for (j = i + 1; j < n; j++)
                    if (v[j] < v[min])
                            min = j;
            aux = v[i]; // troca v[min] com v[i]
            v[i] = v[min];
            v[min] = aux;
    } // for (i = 0; i < n - 1; i++)
} // SelectionSort(int v[], int n)
```

# Ordenação por Seleção

**Desempenho de tempo:**

**O($n^2$) em qualquer caso –** ex.: vetor ordenado ou invertido

- em todas as iterações do laço externo, o laço interno realiza o número máximo de iterações
- 1 + 2 + 3+... n-3 + n-2 + n-1
- n(1+ (n – 1)) / 2 // Soma-da-PA = ( n* (a1 + an) ) / 2
- = $n^2$ / 2
- = O($n^2$)

# Ordenação por Seleção

**Análise de corretude:**

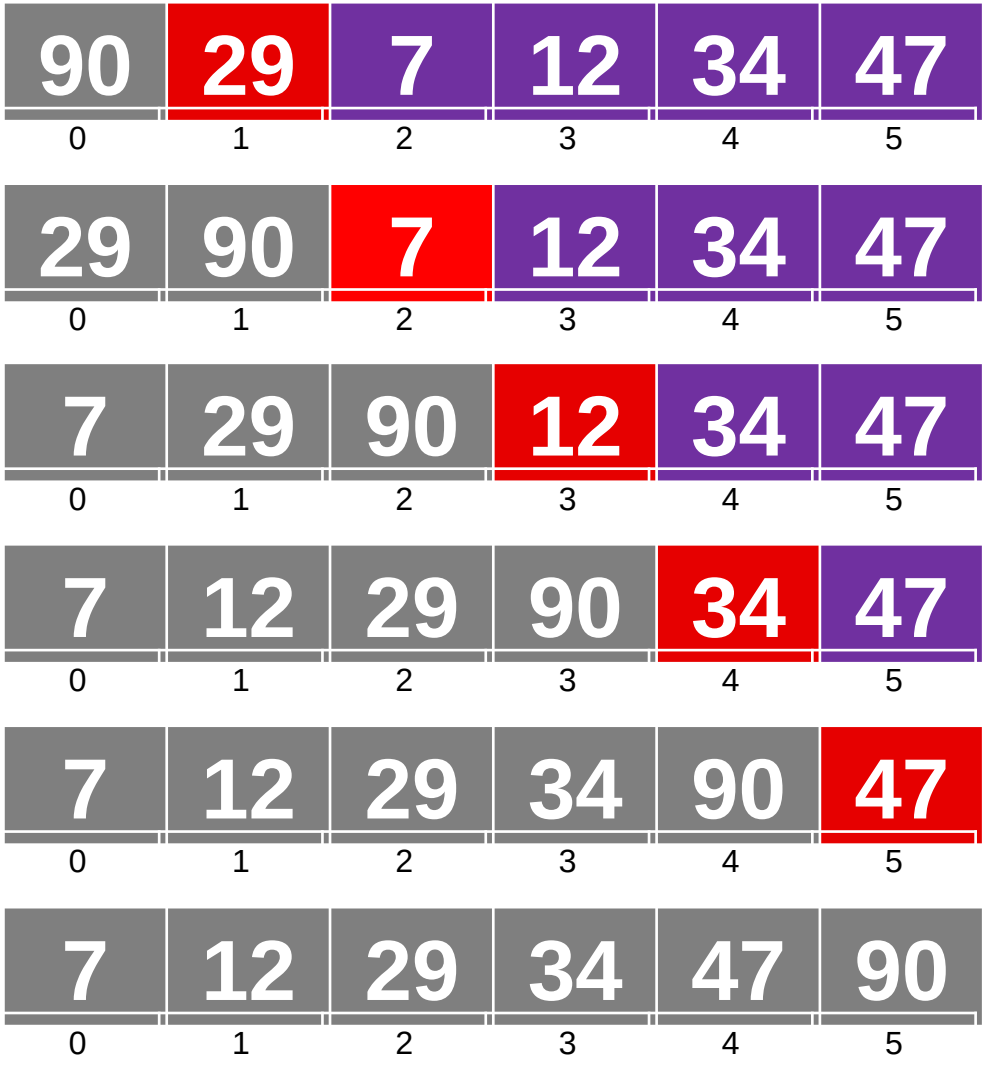
No início de cada iteração do “for” externo:

- o vetor  v[0..n-1]  é uma permutação do vetor original,
- v[0..i-1] está em ordem crescente;
- v[i-1] <= v[i..n-1].

- Assim, após a execução do “for” externo, o vetor estará ordenado.

# Ordenação por Inserção

Percorre o vetor e **insere** o elemento na posição correta.



# Ordenação por Inserção

```
void InsertionSort (int v[], int n) {
    int i, j, aux;

   // inicialmente, v[0] é o vetor ordenado
   for (j = 1; j < n; j++) { // insere os v[ j ] 1 a 1 no vetor ordenado
          aux = v[ j ]; // v[ j ] será inserido na posição correta do vet ordenado

          // desloca para direita todos os elementos maiores que aux
          for (i = j;  (i > 0) && (aux < v[ i-1 ]); i--)
                 v[ i ] = v[ i - 1];

          v[ i ] = aux; // insere aux, ou v [ j ] na posição correta
} // for (j = 1; j < n - 1; j++)
} // InsertionSort(int v[], int n)
```

# Ordenação por Inserção

**Desempenho de tempo:**

**O(n) no melhor caso –** vetor previamente ordenado

- O “for” interno não será executado nenhuma vez

**O($n^2$) no pior caso –** ex.: vetor em orden invertida

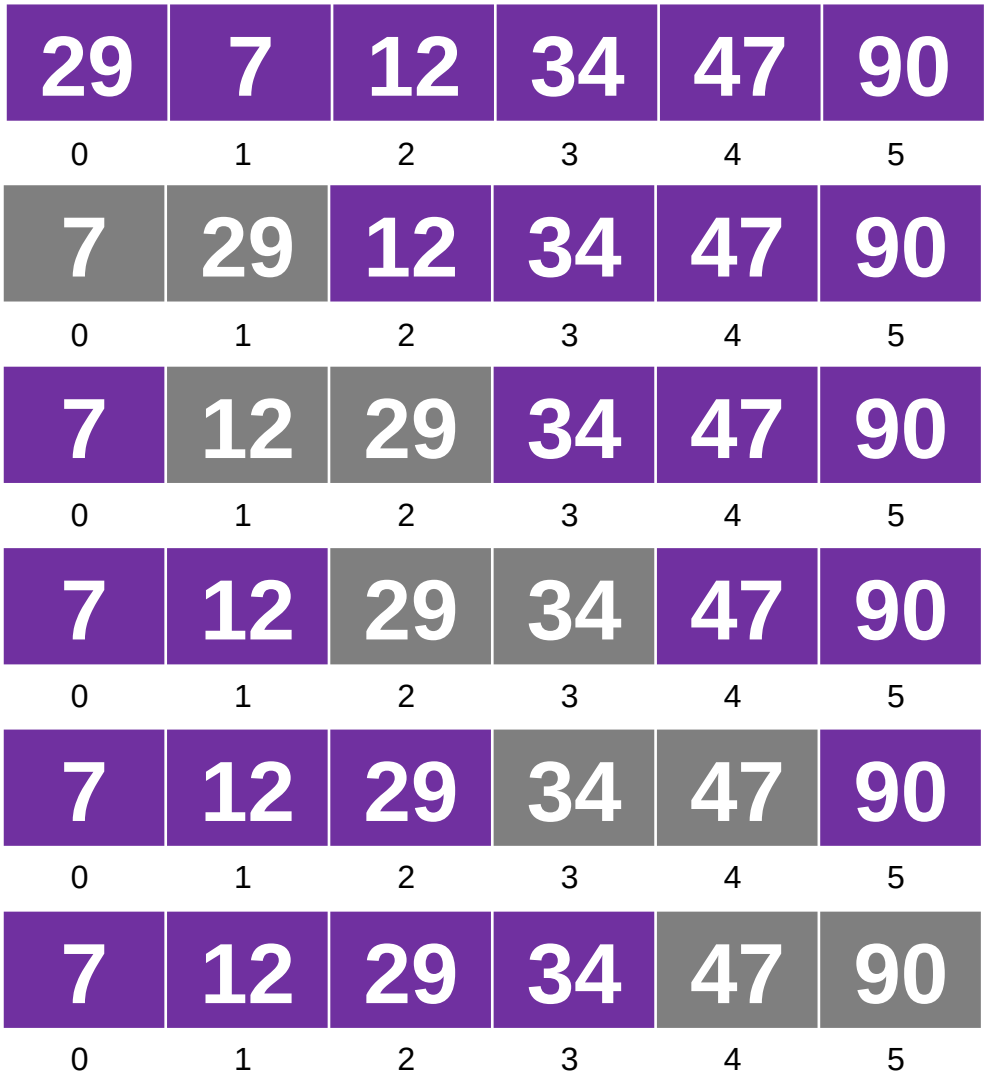
- em todas as n - 1 iterações do laço externo, o laço interno realiza o número máximo de iterações
- 1 + 2 + 3+... N-3 + n-2 + n-1
- n(1+ (n – 1)) / 2 // Soma-da-PA = ( n* (a1 + an) ) / 2
- = $n^2$ / 2
- = O($n^2$)

# Ordenação por Troca (bubble)

Percorre várias vezes o vetor **invertendo** pares adjascentes fora de ordem.

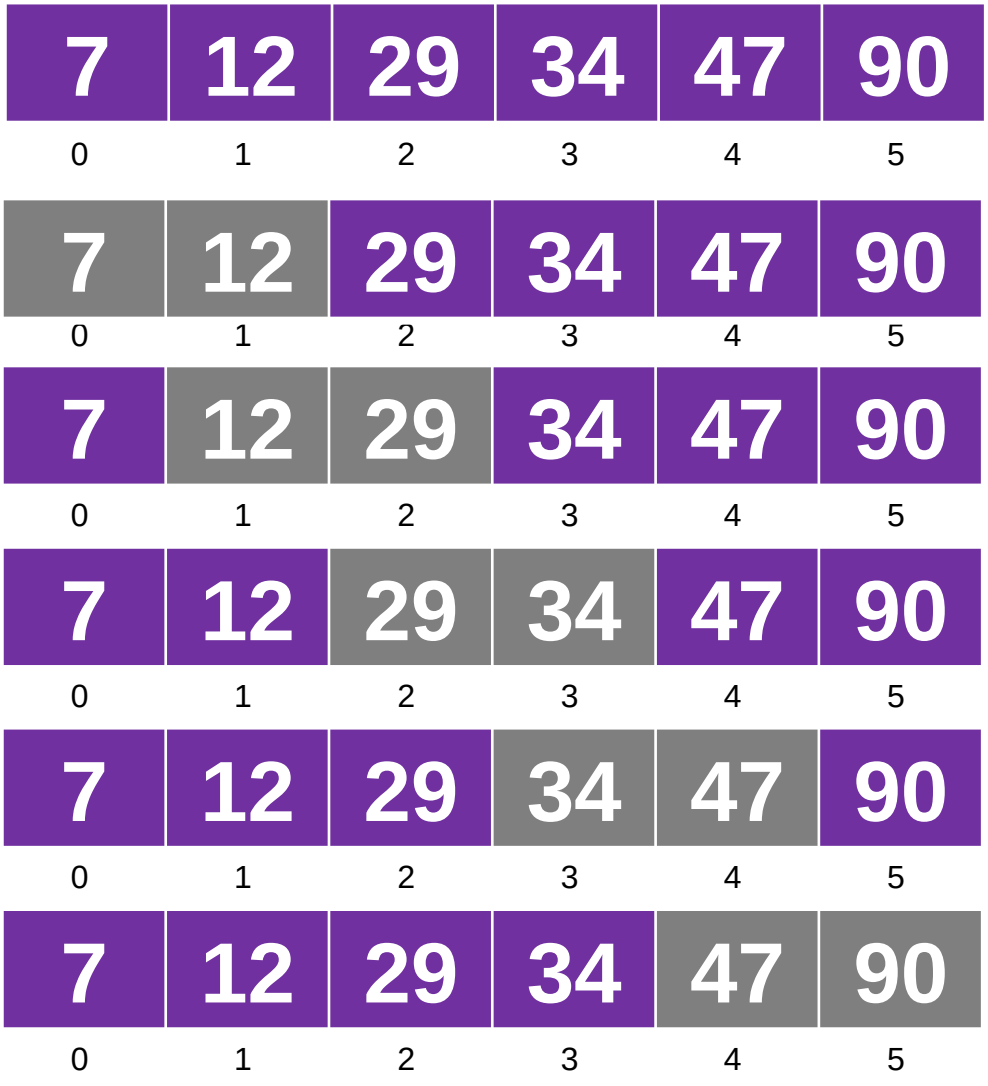
# Ordenação por Troca (bubble)

Percorre a segunda vez...



# Ordenação por Troca (bubble)

Percorre a terceira vez...




Não teve nenhuma troca

# Ordenação por Troca

```
void BubbleSort (int v[], int n) {
    int i, j, min, aux;
    for ( i = 0; i < n; i++) {
          for ( j = 1; j < n; j++)
                if (v[ j - 1] > v[ j ]) {
                      aux = v[ j ]; // troca v[ j ] com v[ j-1 ]
                      v[ j ] = v[ j - 1 ];
                      v[ j - 1 ] = aux;
                      } // troca
    } // for (i = 0; i < n; i++)
} // BubbleSort(int v[], int n)
```

# Ordenação por Troca

**Desempenho de tempo:**

**$O(n^2)$ em qualquer caso –** ex.: vetor ordenado ou invertido

- em todas as iterações do laço externo, o laço interno realiza o número máximo de iterações
- n – 1 passadas, e em cada passada, n – 1 comparações
- (n – 1) * (n-1)
- $N^2 - 2n + 1$
- = $O(n^2)$

**Exemplo de otimização:**

- Parar de passar se não houver troca;
- Exercício: algoritmo; análise de desempenho;
- O que mudaria na análise de desempenho?

# Exercícios:

Exercício 5: Implemente e teste o **algoritmo SelectionSort**. Faça a **análise de desempenho - tempo de execução (use a notação O**).

Exercício 6: Implemente e teste o **algoritmo InsertionSort**. Faça a **análise de desempenho - tempo de execução (use a notação O**).

Exercício 7: Implemente e teste o **algoritmo BubbleSort** otimizado. Faça a **análise de desempenho - tempo de execução (use a notação O**).