Ordenação iterativa

```
void selection_sort (int v[], int n) {
    int i = 0, max;
    while (i < n-1) {
        max = 0;
        int j = 1;
        while (j < n-i) {
            if (v[j] > v[max]) max = j;
            j += 1;
        }
        swap(v[max], v[n - i - 1]);
        i += 1;
    }
}
```

Algoritmo 1 selectionSort (vetor v, tamanho n)

i $\leftarrow$ 0

Algoritmo 1 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do

    i ← i + 1
end while
```

Algoritmo 1 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1




    i ← i + 1
end while
```

Algoritmo 1 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do



        j ← j + 1
    end while

    i ← i + 1
end while
```

Algoritmo 1 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do
        if v[j] < v[max] then
            max ← j
        end if
        j ← j + 1
    end while

    i ← i + 1
end while
```

Algoritmo 1 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do
        if v[j] < v[max] then
            max ← j
        end if
        j ← j + 1
    end while
    swap(v[max], v[n - i -1])
    i ← i + 1
end while
```

Selection sort

- Melhor caso:
  - Laço interno: $3(\frac{n(n-1)}{2}) = \frac{3}{2}n^2 - \frac{3}{2}n$

Algoritmo 3 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do
        if v[j] < v[max] then
            max ← j
        end if
        j ← j + 1
    end while
    swap(v[max], v[n - i -1])
    i ← i + 1
end while
```

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7(n-1) = 7n - 7$

Algoritmo 3 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do
        if v[j] < v[max] then
            max ← j
        end if
        j ← j + 1
    end while
    swap(v[max], v[n - i -1])
    i ← i + 1
end while
```

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7n - 7$
  - Demais operações: 2

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = \frac{3n^2}{2} + \frac{11n}{2} - 5$

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = \frac{3n^2}{2} + \frac{11n}{2} - 5$
  - $t_{\text{melhor}} \in O(n^2)$

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = \frac{3n^2}{2} + \frac{11n}{2} - 5$
  - $t_{\text{melhor}} \in O(n^2)$
  - $t_{\text{melhor}} \in \Omega(n^2)$

Selection sort

- Melhor caso:
  - Laço interno: $\frac{3}{2}n^2 - \frac{3}{2}n$
  - Laço externo: $7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = \frac{3n^2}{2} + \frac{11n}{2} - 5$
  - $t_{\text{melhor}} \in O(n^2)$
  - $t_{\text{melhor}} \in \Omega(n^2)$

  $\Rightarrow t_{\text{melhor}} \in \Theta(n^2) \rightarrow$ Complexidade quadrática

Algoritmo 3 selectionSort (vetor v, tamanho n)

```
i ← 0
while i < n - 1 do
    max ← 0; j ← 1
    while j < n − i do
        if v[j] < v[max] then
            max ← j
        end if
        j ← j + 1
    end while
    swap(v[max], v[n - i -1])
    i ← i + 1
end while
```

Selection sort

- Pior caso:
    - Laço interno: $4(\frac{n(n-1)}{2}) = 2n^2 - 2n$

Selection sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2

Selection sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 5n - 5$

Selection sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 5n - 5$
  - $t_{\text{pior}} \in O(n^2)$

Selection sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 5n - 5$
  - $t_{\text{pior}} \in O(n^2)$
  - $t_{\text{pior}} \in \Omega(n^2)$

Selection sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 5n - 5$
  - $t_{\text{pior}} \in O(n^2)$
  - $t_{\text{pior}} \in \Omega(n^2)$

  $\Rightarrow t_{\text{pior}} \in \Theta(n^2) \rightarrow$ Complexidade quadrática

```
void insertion_sort (int v[], int n) {
    int i = 1, atual;
    while (i < n) {
        atual = v[i];
        int j = i - 1;
        while (j >= 0 && atual < v[j]) {
            v[j+1] = v[j];
            j -= 1;
        }
        v[j+1] = atual;
        i += 1;
    }
}
```

Algoritmo 2 insertionSort (vetor v, tamanho n)

i $\leftarrow$ 1

Algoritmo 2 insertionSort (vetor v, tamanho n)

```
i ← 1
while i < n do

end while
```

Algoritmo 2 insertionSort (vetor v, tamanho n)

i $\leftarrow$ 1

while i $<$ n **do**

  atual $\leftarrow$ v[i]

end while

Algoritmo 2 insertionSort (vetor v, tamanho n)

i $\leftarrow$ 1

while i $<$ n **do**

  atual $\leftarrow$ v[i]

  j $\leftarrow$ i-1

end while

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do

    end while

end while
```

Algoritmo 2 insertionSort (vetor v, tamanho n)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while


end while
```

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual

end while
```

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Melhor caso:
  - Laço interno: 0

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = 7n - 5$

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = 7n - 5$
  - $t_{\text{melhor}} \in O(n)$

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = 7n - 5$
  - $t_{\text{melhor}} \in O(n)$
  - $t_{\text{melhor}} \in \Omega(n)$

Insertion sort

- Melhor caso:
  - Laço interno: 0
  - Laço externo: $7(n-1) = 7n - 7$
  - Demais operações: 2
- $t_{\text{melhor}}(n) = 7n - 5$
  - $t_{\text{melhor}} \in O(n)$
  - $t_{\text{melhor}} \in \Omega(n)$

  $\Rightarrow t_{\text{melhor}} \in \Theta(n) \rightarrow$ Complexidade linear

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Pior caso:
  - Laço interno: $4(\frac{n(n-1)}{2}) = 2(n^2 - n) = 2n^2 - 2n$

Algoritmo 2 insertionSort (vetor v, tamanho n)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6(n-1) = 6n - 6$

Algoritmo 2 insertionSort (vetor *v*, tamanho *n*)

```
i ← 1
while i < n do
    atual ← v[i]
    j ← i-1
    while (j ≥ 0) e (atual < v[j]) do
        v[j+1] = v[j]
        j ← j - 1
    end while
    v[j+1] ← atual
    i ← i + 1
end while
```

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6n - 6$
  - Demais operações: 2

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6n - 6$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 4n - 4$

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6n - 6$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 4n - 4$
  - $t_{\text{pior}} \in O(n^2)$

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6n - 6$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 4n - 4$
  - $t_{\text{pior}} \in O(n^2)$
  - $t_{\text{pior}} \in \Omega(n^2)$

Insertion sort

- Pior caso:
  - Laço interno: $2n^2 - 2n$
  - Laço externo: $6n - 6$
  - Demais operações: 2
- $t_{\text{pior}}(n) = 2n^2 + 4n - 4$
  - $t_{\text{pior}} \in O(n^2)$
  - $t_{\text{pior}} \in \Omega(n^2)$

  $\Rightarrow t_{\text{pior}} \in \Theta(n^2) \rightarrow$ Complexidade quadrática