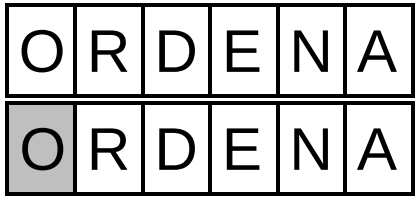
Ordenação Quadrática

Letícia Rodrigues Bueno

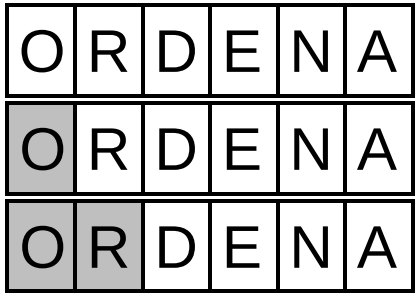
UFABC

Inserção: insere chave na posição certa

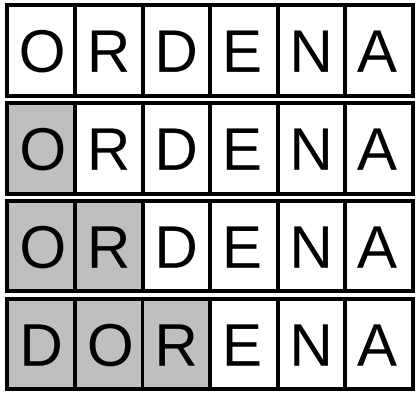
Inserção: insere chave na posição certa



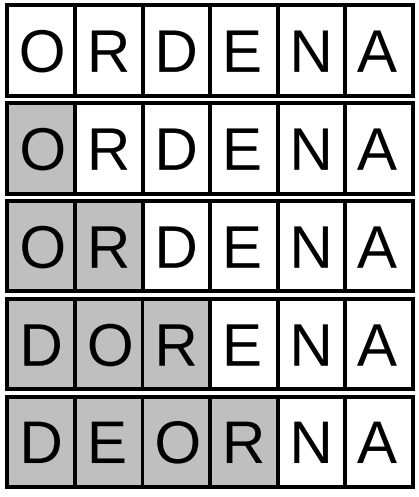
Inserção: insere chave na posição certa



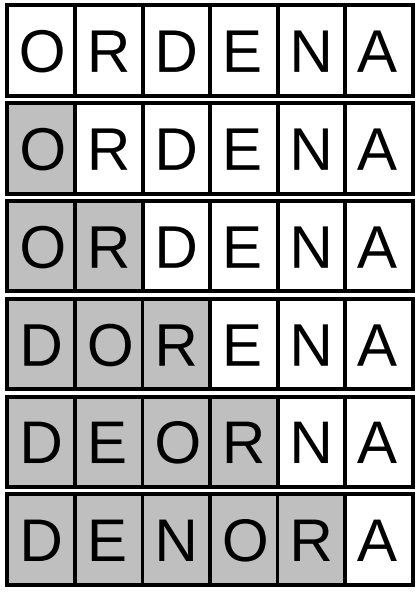
Inserção: insere chave na posição certa



Inserção: insere chave na posição certa



Inserção: insere chave na posição certa



Inserção: insere chave na posição certa

O	R	D	E	N	A
O	R	D	E	N	A
O	R	D	E	N	A
D	O	R	E	N	A
D	E	O	R	N	A
D	E	N	O	R	A
A	D	E	N	O	R

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Ordenação por Inserção

Análise da complexidade:

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Ordenação por Inserção

Análise da complexidade:

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

- Comparação de chaves (linha 6) será executada:

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- Pior caso: $\Theta(n^2)$;

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- Pior caso: $\Theta(n^2)$;
- Melhor caso: $\Theta(n^2)$;

Ordenação por Inserção

```
1  insertion(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          k = j;
5          for (i=j-1; i ≥ 1; i–)
6              if (S[i]>key)
7                  S[i+1] = S[i];
8                  k=i;
9          S[k] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- Pior caso: $\Theta(n^2)$;
- Melhor caso: $\Theta(n^2)$;

Ordenação por Inserção Melhorado

```
1  insertionbest(n, S, key):
2       for (j=2; j ≤ n; j++)
3            key = S[j];
4            i = j-1;
5            while ((i>0) && (S[i]>key))
6                 S[i+1] = S[i];
7                 i=i-1;
8            S[i+1] = key;
```

Ordenação por Inserção Melhorado

Análise da complexidade:

```
1  insertionbest(n, S, key):
2       for (j=2; j ≤ n; j++)
3            key = S[j];
4            i = j-1;
5            while ((i>0) && (S[i]>key))
6                 S[i+1] = S[i];
7                 i=i-1;
8            S[i+1] = key;
```

Ordenação por Inserção Melhorado

```
1  insertionbest(n, S, key):
2       for (j=2; j ≤ n; j++)
3            key = S[j];
4            i = j-1;
5            while ((i>0) && (S[i]>key))
6                 S[i+1] = S[i];
7                 i=i-1;
8            S[i+1] = key;
```

Análise da complexidade:

- **Pior caso:** comparação de chaves (linha 5) será executada:

Ordenação por Inserção Melhorado

```
1  insertionbest(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          i = j-1;
5          while ((i>0) && (S[i]>key))
6              S[i+1] = S[i];
7              i=i-1;
8          S[i+1] = key;
```

Análise da complexidade:

- **Pior caso:** comparação de chaves (linha 5) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

Ordenação por Inserção Melhorado

```
1  insertionbest(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          i = j-1;
5          while ((i>0) && (S[i]>key))
6              S[i+1] = S[i];
7              i=i-1;
8          S[i+1] = key;
```

Análise da complexidade:

- **Pior caso:** comparação de chaves (linha 5) será executada:

$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

Ordenação por Inserção Melhorado

```
1  insertionbest(n, S, key):
2      for (j=2; j ≤ n; j++)
3          key = S[j];
4          i = j-1;
5          while ((i>0) && (S[i]>key))
6              S[i+1] = S[i];
7              i=i-1;
8          S[i+1] = key;
```

Análise da complexidade:

- **Pior caso:** comparação de chaves (linha 5) será executada:

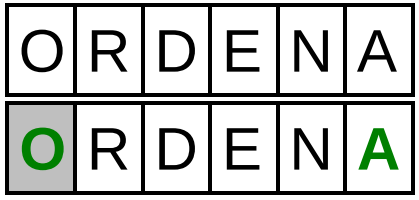
$$1+2+\ldots+n-2+n-1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- **Melhor caso:** linha 5 executa 1 vez para cada j. Total: $n-1$ vezes $= \Theta(n)$;

Seleção: seleciona menor elemento da subsequência

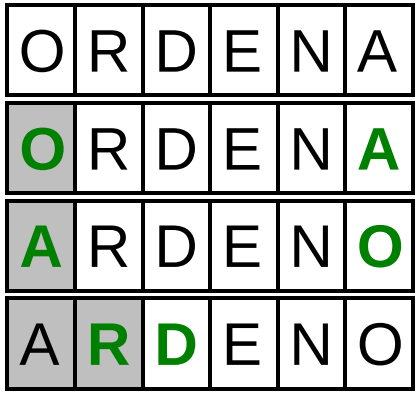
Seleção: seleciona menor elemento da subsequência



Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**

Seleção: seleciona menor elemento da subsequência



Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O
A	D	**E**	**R**	N	O

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O
A	D	**E**	**R**	N	O
A	D	E	**R**	**N**	O

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O
A	D	**E**	**R**	N	O
A	D	E	**R**	**N**	O
A	D	E	**N**	**R**	O

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O
A	D	**E**	**R**	N	O
A	D	E	**R**	**N**	O
A	D	E	**N**	**R**	O
A	D	E	N	**R**	**O**

Seleção: seleciona menor elemento da subsequência

O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	D	E	N	**O**
A	**R**	**D**	E	N	O
A	**D**	**R**	E	N	O
A	D	**R**	**E**	N	O
A	D	**E**	**R**	N	O
A	D	E	**R**	**N**	O
A	D	E	**N**	**R**	O
A	D	E	N	**R**	**O**
A	D	E	N	**O**	**R**

Ordenação por Seleção

```
 1  selection(n, S, key):
 2      for (i=1; i ≤ n-1; i++)
 3          key = S[i];
 4          index = i;
 5          for (j = i+1; j ≤ n; j++)
 6              if (S[j]<key)
 7                  key = S[j];
 8                  index = j;
 9          S[index]= S[i];
10          S[i] = key;
```

Ordenação por Seleção

Análise da complexidade:

```
1  selection(n, S, key):
2      for (i=1; i ≤ n-1; i++)
3          key = S[i];
4          index = i;
5          for (j = i+1; j ≤ n; j++)
6              if (S[j]<key)
7                  key = S[j];
8                  index = j;
9          S[index]= S[i];
10         S[i] = key;
```

Ordenação por Seleção

```
 1  selection(n, S, key):
 2      for (i=1; i ≤ n-1; i++)
 3          key = S[i];
 4          index = i;
 5          for (j = i+1; j ≤ n; j++)
 6              if (S[j]<key)
 7                  key = S[j];
 8                  index = j;
 9          S[index]= S[i];
10          S[i] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

Ordenação por Seleção

```
1  selection(n, S, key):
2      for (i=1; i ≤ n-1; i++)
3          key = S[i];
4          index = i;
5          for (j = i+1; j ≤ n; j++)
6              if (S[j]<key)
7                  key = S[j];
8                  index = j;
9          S[index]= S[i];
10         S[i] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

Ordenação por Seleção

```
 1  selection(n, S, key):
 2      for (i=1; i ≤ n-1; i++)
 3          key = S[i];
 4          index = i;
 5          for (j = i+1; j ≤ n; j++)
 6              if (S[j]<key)
 7                  key = S[j];
 8                  index = j;
 9          S[index]= S[i];
10          S[i] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

Ordenação por Seleção

```
1  selection(n, S, key):
2      for (i=1; i ≤ n-1; i++)
3          key = S[i];
4          index = i;
5          for (j = i+1; j ≤ n; j++)
6              if (S[j]<key)
7                  key = S[j];
8                  index = j;
9          S[index]= S[i];
10         S[i] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- Pior caso: $\Theta(n^2)$;

Ordenação por Seleção

```
1  selection(n, S, key):
2      for (i=1; i ≤ n-1; i++)
3          key = S[i];
4          index = i;
5          for (j = i+1; j ≤ n; j++)
6              if (S[j]<key)
7                  key = S[j];
8                  index = j;
9          S[index]= S[i];
10         S[i] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 6) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- Pior caso: $\Theta(n^2)$;
- Melhor caso: $\Theta(n^2)$;

Bolha: chave “flutua” (bolha) até posição certa

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**
D	E	N	**O**	A	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**
D	E	N	**O**	A	**R**
D	E	**N**	A	**O**	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**
D	E	N	**O**	A	**R**
D	E	**N**	A	**O**	**R**
D	**E**	A	**N**	**O**	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**
D	E	N	**O**	A	**R**
D	E	**N**	A	**O**	**R**
D	**E**	A	**N**	**O**	**R**
D	A	**E**	**N**	**O**	**R**

Bolha: chave “flutua” (bolha) até posição certa

O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	**R**
D	**O**	E	N	A	**R**
D	E	**O**	N	A	**R**
D	E	N	**O**	A	**R**
D	E	**N**	A	**O**	**R**
D	**E**	A	**N**	**O**	**R**
D	A	**E**	**N**	**O**	**R**
A	**D**	**E**	**N**	**O**	**R**

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Ordenação por Bolha

Análise da complexidade:

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

  $n(n-1)$

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

$$n(n-1) = n^2 - n$$

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

$$n(n-1) = n^2 - n = \Theta(n^2)$$

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

  $n(n-1) = n^2 - n = \Theta(n^2)$

- Pior caso = $\Theta(n^2)$

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

  $n(n-1) = n^2 - n = \Theta(n^2)$

- Pior caso = $\Theta(n^2)$
- Melhor caso = $\Theta(n^2)$

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

  $n(n-1) = n^2 - n = \Theta(n^2)$

- Pior caso = $\Theta(n^2)$
- Melhor caso = $\Theta(n^2)$

Podemos melhorar?

Ordenação por Bolha

```
1  bubble(n, S, key):
2      for (i = 1; i ≤ n; i++)
3          for (j = 2; j ≤ n; j++)
4              if (S[j-1]>S[j])
5                  key = S[j-1];
6                  S[j-1] = S[j];
7                  S[j] = key;
```

Análise da complexidade:

- Comparação de chaves (linha 4) será executada:

  $n(n-1) = n^2 - n = \Theta(n^2)$

- Pior caso = $\Theta(n^2)$
- Melhor caso = $\Theta(n^2)$

Podemos melhorar? **Yes, we can!**

Ordenação por Bolha Melhorado

```
 1  bubblebest(n, S, key):
 2      pas=1;
 3      ok=false;
 4      while (pas<n && not ok)
 5          ok=true;
 6          for (i=1; i ≤ n-pas; i++)
 7              if (S[i]>S[i+1])
 8                  key = S[i];
 9                  S[i] = S[i+1];
10                  S[i+1] = key;
11                  ok=false;
12          pas=pas+1;
```

Ordenação por Bolha Melhorado

Análise da complexidade:

```
 1  bubblebest(n, S, key):
 2      pas=1;
 3      ok=false;
 4      while (pas<n && not ok)
 5          ok=true;
 6          for (i=1; i ≤ n-pas; i++)
 7              if (S[i]>S[i+1])
 8                  key = S[i];
 9                  S[i] = S[i+1];
10                  S[i+1] = key;
11                  ok=false;
12          pas=pas+1;
```

Ordenação por Bolha Melhorado

```
1  bubblebest(n, S, key):
2      pas=1;
3      ok=false;
4      while (pas<n && not ok)
5          ok=true;
6          for (i=1; i ≤ n-pas; i++)
7              if (S[i]>S[i+1])
8                  key = S[i];
9                  S[i] = S[i+1];
10                 S[i+1] = key;
11                 ok=false;
12         pas=pas+1;
```

Análise da complexidade:

- Pior caso: comparação de chaves (linha 7) será executada:

Ordenação por Bolha Melhorado

```
1  bubblebest(n, S, key):
2      pas=1;
3      ok=false;
4      while (pas<n && not ok)
5          ok=true;
6          for (i=1; i ≤ n-pas; i++)
7              if (S[i]>S[i+1])
8                  key = S[i];
9                  S[i] = S[i+1];
10                 S[i+1] = key;
11                 ok=false;
12         pas=pas+1;
```

Análise da complexidade:

- Pior caso: comparação de chaves (linha 7) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

Ordenação por Bolha Melhorado

```
1  bubblebest(n, S, key):
2      pas=1;
3      ok=false;
4      while (pas<n && not ok)
5          ok=true;
6          for (i=1; i ≤ n-pas; i++)
7              if (S[i]>S[i+1])
8                  key = S[i];
9                  S[i] = S[i+1];
10                 S[i+1] = key;
11                 ok=false;
12         pas=pas+1;
```

Análise da complexidade:

- Pior caso: comparação de chaves (linha 7) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

Ordenação por Bolha Melhorado

```
1  bubblebest(n, S, key):
2      pas=1;
3      ok=false;
4      while (pas<n && not ok)
5          ok=true;
6          for (i=1; i ≤ n-pas; i++)
7              if (S[i]>S[i+1])
8                  key = S[i];
9                  S[i] = S[i+1];
10                 S[i+1] = key;
11                 ok=false;
12         pas=pas+1;
```

Análise da complexidade:

- Pior caso: comparação de chaves (linha 7) será executada:

$$n-1+n-2+\ldots+2+1 = \sum_{i=1}^{n-1} i =$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

- **Melhor caso:** linha 7 executa $n-1$ vezes para ver sequência já ordenada. Logo: $\Theta(n)$

Exercícios

1. Um algoritmo de ordenação é **estável** se preserva ordem relativa de itens com valores idênticos. Responda: ordenação por inserção, seleção e bolha são estáveis? Exemplifique. Para cada um deles que não for estável, é possível modificá-lo para se tornar estável?

Exercícios

1. Um algoritmo de ordenação é **estável** se preserva ordem relativa de itens com valores idênticos. Responda: ordenação por inserção, seleção e bolha são estáveis? Exemplifique. Para cada um deles que não for estável, é possível modificá-lo para se tornar estável?
2. Analise os algoritmos de ordenação por inserção, seleção e bolha em relação aos números de trocas realizadas no pior e melhor caso. Compare-os.

Exercícios

1. Um algoritmo de ordenação é **estável** se preserva ordem relativa de itens com valores idênticos. Responda: ordenação por inserção, seleção e bolha são estáveis? Exemplifique. Para cada um deles que não for estável, é possível modificá-lo para se tornar estável?
2. Analise os algoritmos de ordenação por inserção, seleção e bolha em relação aos números de trocas realizadas no pior e melhor caso. Compare-os.
3. Escreva uma função que verifique se um vetor $V[1..n]$ está em ordem crescente. Seu programa deve ser o mais econômico possível.

Exercícios

1. Um algoritmo de ordenação é **estável** se preserva ordem relativa de itens com valores idênticos. Responda: ordenação por inserção, seleção e bolha são estáveis? Exemplifique. Para cada um deles que não for estável, é possível modificá-lo para se tornar estável?
2. Analise os algoritmos de ordenação por inserção, seleção e bolha em relação aos números de trocas realizadas no pior e melhor caso. Compare-os.
3. Escreva uma função que verifique se um vetor $V[1..n]$ está em ordem crescente. Seu programa deve ser o mais econômico possível.
4. Modifique os três algoritmos de ordenação (inserção, seleção e bolha) para fazer ordenação decrescente.

Exercícios

1. Um algoritmo de ordenação é **estável** se preserva ordem relativa de itens com valores idênticos. Responda: ordenação por inserção, seleção e bolha são estáveis? Exemplifique. Para cada um deles que não for estável, é possível modificá-lo para se tornar estável?
2. Analise os algoritmos de ordenação por inserção, seleção e bolha em relação aos números de trocas realizadas no pior e melhor caso. Compare-os.
3. Escreva uma função que verifique se um vetor $V[1..n]$ está em ordem crescente. Seu programa deve ser o mais econômico possível.
4. Modifique os três algoritmos de ordenação (inserção, seleção e bolha) para fazer ordenação decrescente.
5. Escreva uma função que coloque em ordem lexicográfica um vetor de strings. Use o algoritmo de inserção.

Bibliografia Utilizada

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3ª edição, MIT Press, 2009.

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Pascal e C, 2ª edição, Cengage Learning, 2009.