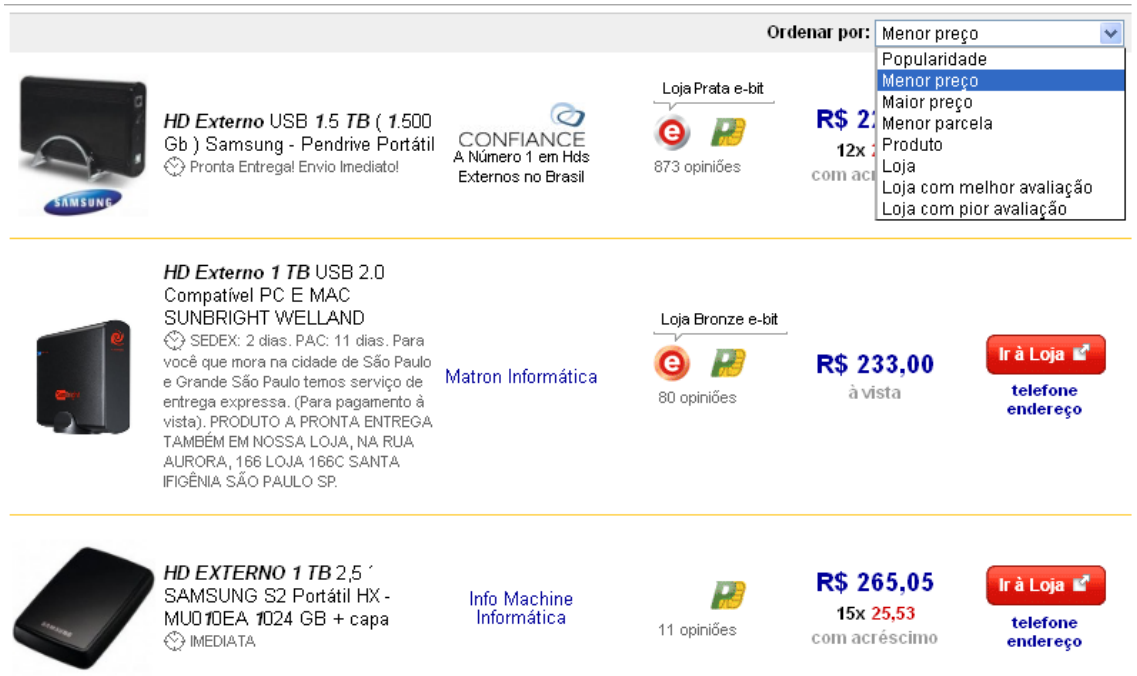
SCC-601– Introdução à Ciência da Computação II

Ordenação e Complexidade – Parte 1

Lucas Antiqueira

Por que ordenar?

- Aplicações:



- Ordenação de e-mail por data
- Ordenação de lista de arquivos por tamanho
- Ordenação de produtos por preço
- Ordenação dos resultados exibidos pelo Google

- Em geral, ordenar qualquer tipo de consulta feita a uma base de dados

Por que ordenar?

- Algoritmos de busca: são eficientes se o vetor estiver ordenado

-1.2	0.6	3.7	.....	13.1	52.0

- Encontrar a mediana
- Excluir itens duplicados
- Compressão de dados (algoritmo de Huffman)
- Representação de conjuntos
- Etc...

O Problema da Ordenação

- Algoritmos de ordenação são ilustrativos
  - Como resolver problemas computacionais
  - Como lidar com estruturas de dados
  - Como desenvolver algoritmos elegantes e como analisar e comparar seus desempenhos

O Problema da Ordenação

- Ordenar (ou classificar)
  - *Definição: organizar uma seqüência de elementos de modo que os mesmos estabeleçam alguma relação de ordem*
    - *Diz-se que os elementos $k_1, \ldots, k_n$ estarão dispostos de modo que $k_1 \leq k_2 \leq \ldots \leq k_n$*

O Problema da Ordenação

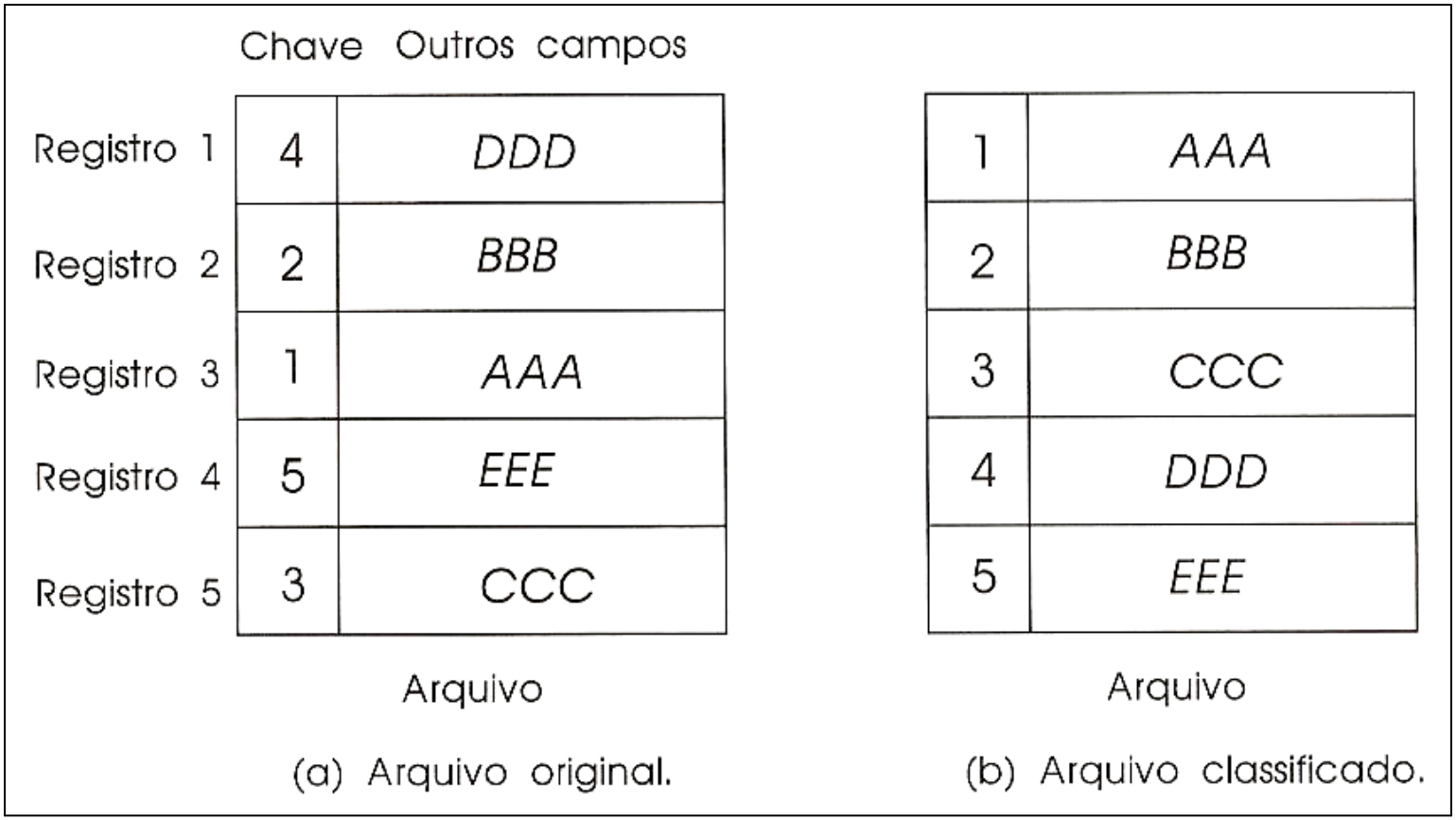
- Terminologia/conceitos
  - Ordenação de **registros**, em que cada registro é ordenado por sua **chave**
  - Ordenação **interna** vs. **externa**
  - Ordenação **estável**: ordenação original de registros com mesma chave é preservada após a ordenação dos registros

O Problema da Ordenação

- Terminologia/conceitos
  - Ordenação sobre os próprios registros
    - Os registros são trocados de posição
  - Ordenação por endereços
    - Mantém-se uma tabela de ponteiros para os registros e alteram-se somente os ponteiros durante a ordenação

O Problema da Ordenação

- Exemplo: ordenação sobre os próprios registros



	Chave	Outros campos
Registro 1	4	*DDD*
Registro 2	2	*BBB*
Registro 3	1	*AAA*
Registro 4	5	*EEE*
Registro 5	3	*CCC*

Arquivo

(a) Arquivo original.

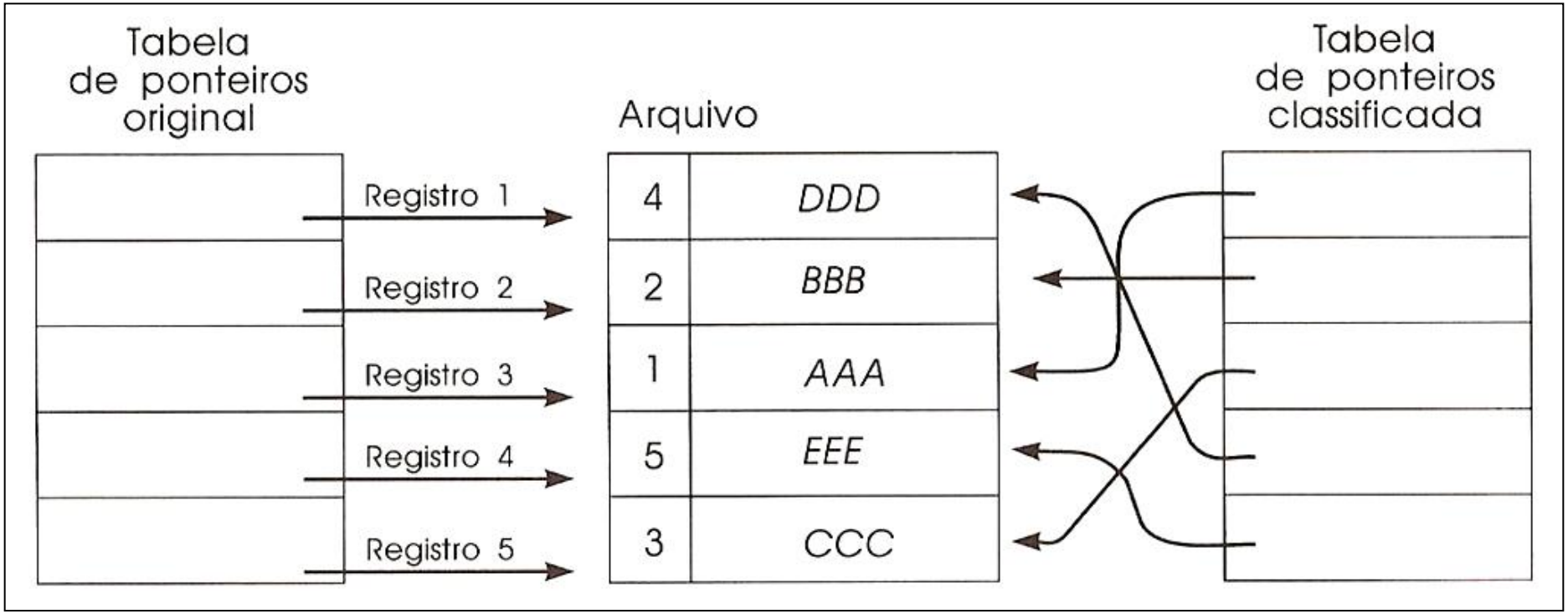
1	*AAA*
2	*BBB*
3	*CCC*
4	*DDD*
5	*EEE*

Arquivo

(b) Arquivo classificado.

O Problema da Ordenação

- Exemplo: ordenação por endereços



O Problema da Ordenação

- Existem vários maneiras de se implementar ordenação
- Dependendo do problema, um algoritmo apresenta vantagens e desvantagens sobre outro
- Como comparar?

O Problema da Ordenação

- Existem vários maneiras de se implementar ordenação
- Dependendo do problema, um algoritmo apresenta vantagens e desvantagens sobre outro
- Como comparar?
  - Devemos comparar as complexidades dos algoritmos

O Problema da Ordenação

- Comparação das complexidades dos algoritmos de ordenação

- Qual a operação dominante?
  - Número de comparações entre elementos, na maioria dos casos
  - *Detalhe*: somente as comparações que podem resultar em trocas de elementos

Algoritmos de Ordenação

- Tradicionalmente, nos estudos dos métodos de ordenação, assume-se que a entrada dos algoritmos é um vetor de números inteiros

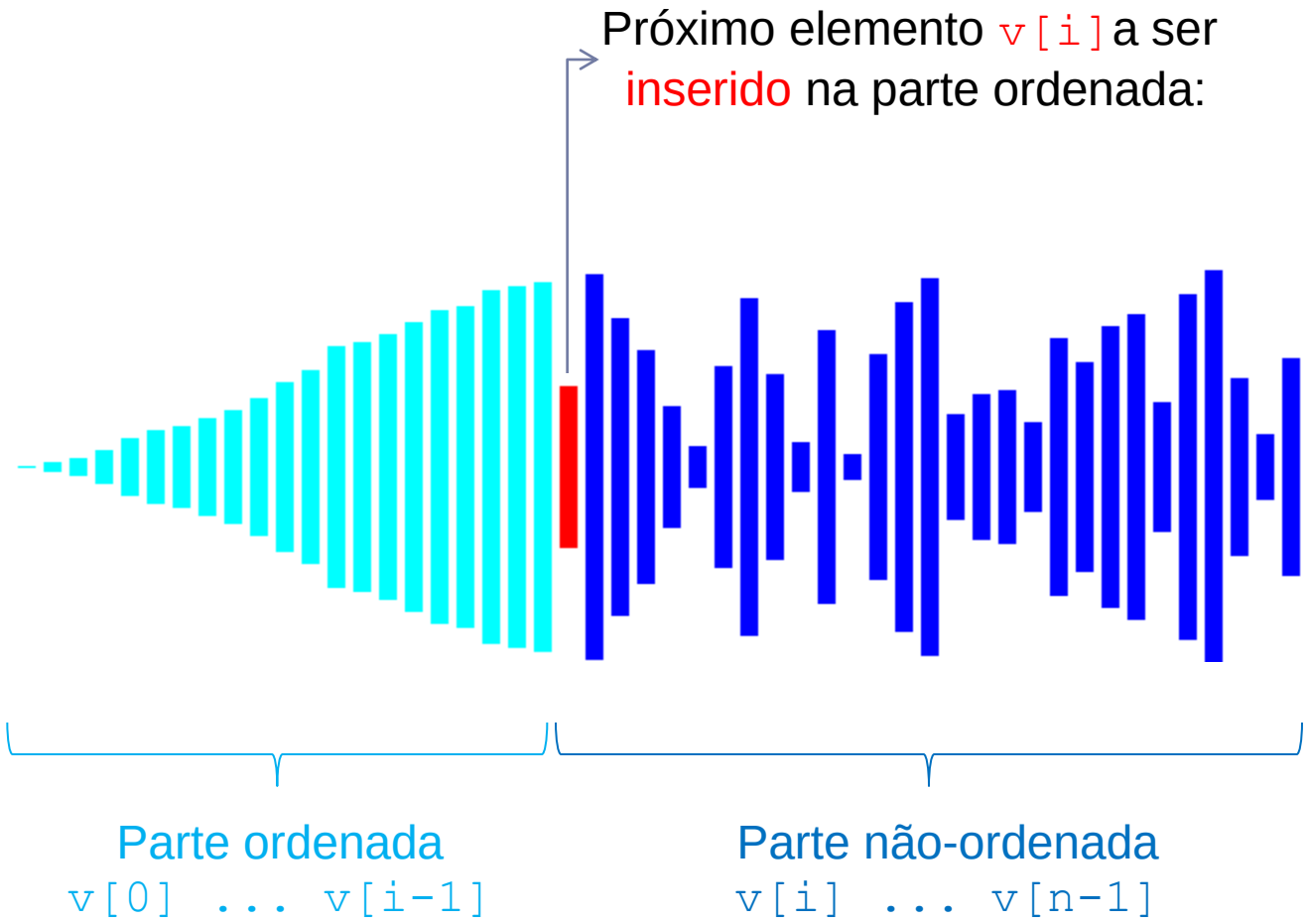
- Procura-se deixar o vetor em ordem crescente

Inserção Simples

Inserção Simples

- Idéia básica
  - Ordenar o conjunto inserindo os elementos não-ordenados em um subconjunto já ordenado
    - No i-ésimo passo, inserir o i-ésimo elemento na posição correta entre v[0], ... , v[i-1], que já estão em ordem
      - Conseqüentemente, alguns elementos são realocados

Inserção Simples



Inserção Simples: exemplo

v = (44 , 55 , 12 , 42 , 94 , 18 , 06 , 67)

44 | 55 12 42 94 18 06 67
44 55 | 12 42 94 18 06 67
12 44 55 | 42 94 18 06 67
12 42 44 55 | 94 18 06 67
12 42 44 55 94 | 18 06 67
12 18 42 44 55 94 | 06 67
06 12 18 42 44 55 94 | 67
06 12 18 42 44 55 67 94 |

Inserção Simples

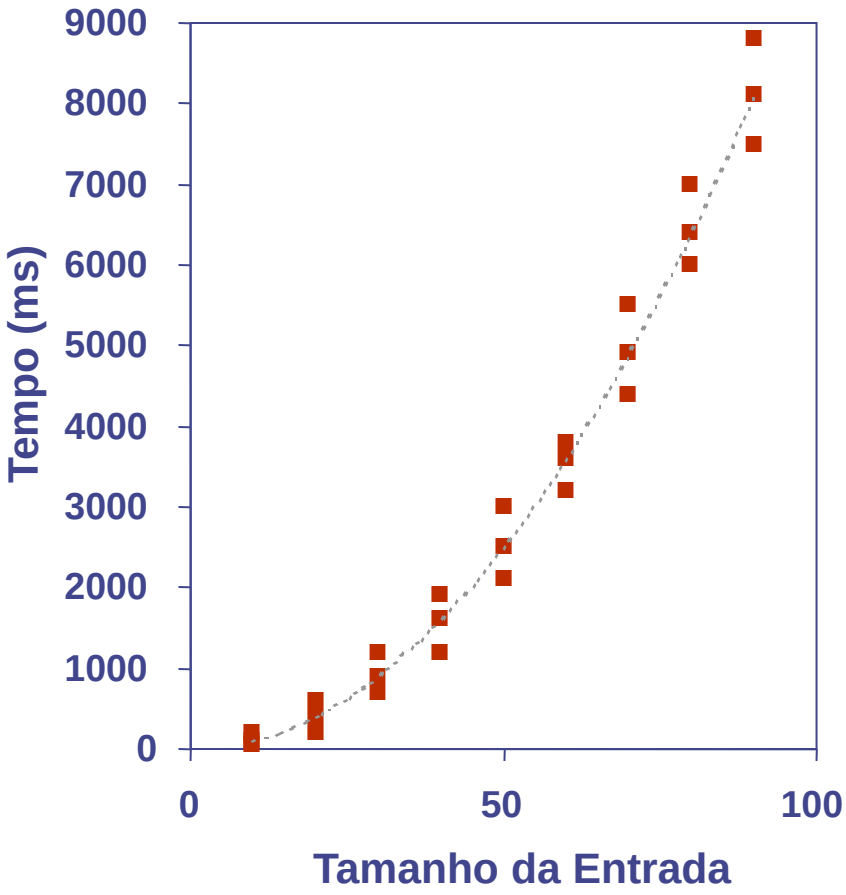
```
void insertion_sort(int v[], int n) {
  int i, j, x;
  for (i = 1; i < n; i++) {
       x = v[i];
       j = i - 1;
       while ((j >= 0) && (x < v[j])) {
              v[j+1] = v[j];
              j--;
       }
       v[j+1] = x;
  }
}
```

Inserção Simples

- Complexidade de tempo?

Complexidade: método experimental

- Execute o programa variando a composição e o tamanho das entradas.
- Use algum método para obter medidas do tempo.
- Esboce um gráfico de resultados.



Limitações do método experimental

- É preciso implementar o algoritmo.
- Os resultados podem não servir como indicativo do tempo de execução para outras entradas que não foram consideradas nos testes.
- Para comparar dois algoritmos, devem ser utilizadas exatamente as mesmas condições, configurações e ambientes de hardware e software.

Complexidade: análise teórica

- Pode se basear em uma descrição de alto nível do algoritmo, ao invés de uma dada implementação.
- Caracteriza o tempo de execução como uma função do tamanho da entrada, ***n***.
  - No caso da ordenação, ***n*** é o tamanho do vetor a ser ordenado.
- Leva em consideração todas as possíveis entradas.
- Permite avaliar a rapidez de um algoritmo de forma independente de qualquer ambiente de hardware e/ou software.

Contagem de operações primitivas

- São ações básicas executadas pelos algoritmos.
- Não dependem da linguagem de programação.
- Considera-se que tenham tempo de execução constante.

- Exemplos:
  - Atribuição de valor a uma variável.
  - Operação aritmética com dois números.
  - Comparação de dois números.
  - Indexação em um vetor.
  - Chamar ou retornar de uma rotina.

Contagem de operações

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

Quantas operações primitivas são executadas no algoritmo de ordenação por inserção?

Contagem de operações

```
void insertion_sort(int v[], int n) {
  int i, j, x;
  for (i = 1; i < n; i++) {
      x = v[i];                    2 operações (indexação + atribuição)
      j = i - 1;                   2 operações (subtração + atribuição)
      while ((j >= 0) && (x < v[j])) {
          v[j+1] = v[j];           4 operações (2 * indexação + soma + atribuição)
          j--;                     2 operações (subtração + atribuição)
      }
      v[j+1] = x;                  2 operações (indexação + atribuição)
  }
}
```

Contagem de operações

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

Loop com n-1 iterações:
→ 1 inicialização
→ n comparações
→ n-1 incrementos (2 operações cada)

Contagem de operações

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

Loop com n-1 iterações:
1 + n + 2(n-1) =
= 1 + n + 2n - 2 =
= 3n - 1 operações no cabeçalho do loop

Contagem de operações

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

3n - 1 operações

2 operações

2 operações

? operações

4 operações

2 operações

2 operações

Contagem de operações

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

4 operações (indexação + 3*comparações) realizadas i+1 vezes

Contagem de operações

```
void insertion_sort(int v[], int n) {
  int i, j, x;
  for (i = 1; i < n; i++) {                    3n - 1 operações
      x = v[i];                                2 operações
      j = i - 1;                               2 operações
      while ((j >= 0) && (x < v[j])) {         4(i+1) operações
            v[j+1] = v[j];                     4 operações
            j--;                               2 operações
      }
      v[j+1] = x;                              2 operações
  }
}
```

Contagem de operações

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

4 operações
2 operações

Dentro de um loop.
Quantas iterações?

Contagem de operações

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

4 operações
2 operações
} i iterações

Consideramos o pior caso
(vetor ordenado inversamente)
O loop termina somente quando j<0

Contagem de operações

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

4(i+1) operações

6i operações

Contagem de operações

```
void insertion_sort(int v[], int n) {
  int i, j, x;
  for (i = 1; i < n; i++) {                    3n - 1 operações
      x = v[i];                                2 operações
      j = i - 1;                               2 operações
      while ((j >= 0) && (x < v[j])) {         4(i+1) operações
            v[j+1] = v[j];                     6i operações
            j--;
      }
      v[j+1] = x;                              2 operações
  }
}
```

Contagem de operações

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

3n - 1 operações

2 operações

2 operações

4(i+1) operações

6i operações

2 operações

Quantas iterações?

Contagem de operações

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

3n - 1 operações

2 operações

2 operações

4(i+1) operações

6i operações

2 operações

n-1 iterações

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

(n-1)* 4(i+1) oper.

(n-1)* 6i operações

(n-1)* 2 operações

n-1 iterações

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

~~ERRADO~~

n-1 iterações

(n-1)* 2 operações

Devemos deixar o número de operações SOMENTE em função de n

Contagem de operações

O loop mais interno (while) é executado da seguinte maneira (índice j):

```
Para i = 1  :       j varia de 0 a 0   (1 iteração)
Para i = 2  :       j varia de 1 a 0   (2 iterações)
Para i = 3  :       j varia de 2 a 0   (3 iterações)
Para i = 4  :       j varia de 3 a 0   (4 iterações)
...
Para i = n-1:       j varia de n-2 a 0 (n-1 iterações)
```

Contagem de operações

O loop mais interno (while) é executado da seguinte maneira (índice j):

```
Para i = 1  :       j varia de 0 a 0    (1 iteração)
Para i = 2  :       j varia de 1 a 0    (2 iterações)
Para i = 3  :       j varia de 2 a 0    (3 iterações)
Para i = 4  :       j varia de 3 a 0    (4 iterações)
...
Para i = n-1:       j varia de n-2 a 0 (n-1 iterações)
```

Quantas vezes o loop mais interno é executado?

Contagem de operações

O loop mais interno (while) é executado da seguinte maneira (índice j):

```
Para i = 1  :       j varia de 0 a 0    (1 iteração)
Para i = 2  :       j varia de 1 a 0    (2 iterações)
Para i = 3  :       j varia de 2 a 0    (3 iterações)
Para i = 4  :       j varia de 3 a 0    (4 iterações)
...
Para i = n-1:       j varia de n-2 a 0 (n-1 iterações)
```

Quantas vezes o loop mais interno é executado? No pior caso:
$1 + 2 + ... + n-1 = [(1 + n-1)*n-1] / 2 = (n^2 – n)/2$

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

$(n^2 - n)/2$ vezes 6 operações internas

(n-1)* 2 operações

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

??? 6*(n^2 – n)/2 operações

(n-1)* 2 operações

Contagem de operações

O **cabeçalho** do loop mais interno (while) é executado da seguinte maneira (índice j):

```
Para i = 1  :       j varia de 0 a -1    (2 iterações)
Para i = 2  :       j varia de 1 a -1    (3 iterações)
Para i = 3  :       j varia de 2 a -1    (4 iterações)
Para i = 4  :       j varia de 3 a -1    (5 iterações)
...
Para i = n-1:       j varia de n-2 a -1 (n iterações)
```

$2 + 3 + ... + n = [(2 + n)*n-1] / 2 = (n^2 + n - 2)/2$

Contagem de operações

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

4 operações (indexação + 3*comparações) realizadas $(n^2 + n - 2)/2$ vezes

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

$4*(n^2 + n - 2)/2$ operações

$6*(n^2 – n)/2$ operações

(n-1)* 2 operações

Contagem de operações

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

$4*(n^2 + n - 2)/2$ operações

$6*(n^2 – n)/2$ operações

(n-1)* 2 operações

Tudo está agora em função de n, basta somar para obter o total de operações da função

Contagem de operações

Número de operações primitivas executadas no algoritmo de ordenação por inserção:

$3n - 1 + 2(n-1) + 2(n-1) + 4(n^2 + n - 2)/2 + 6(n^2 - n)/2 + 2(n-1) =$

$= 3n - 1 + 6(n-1) + [4(n^2 + n - 2) + 6(n^2 - n)]/2 =$

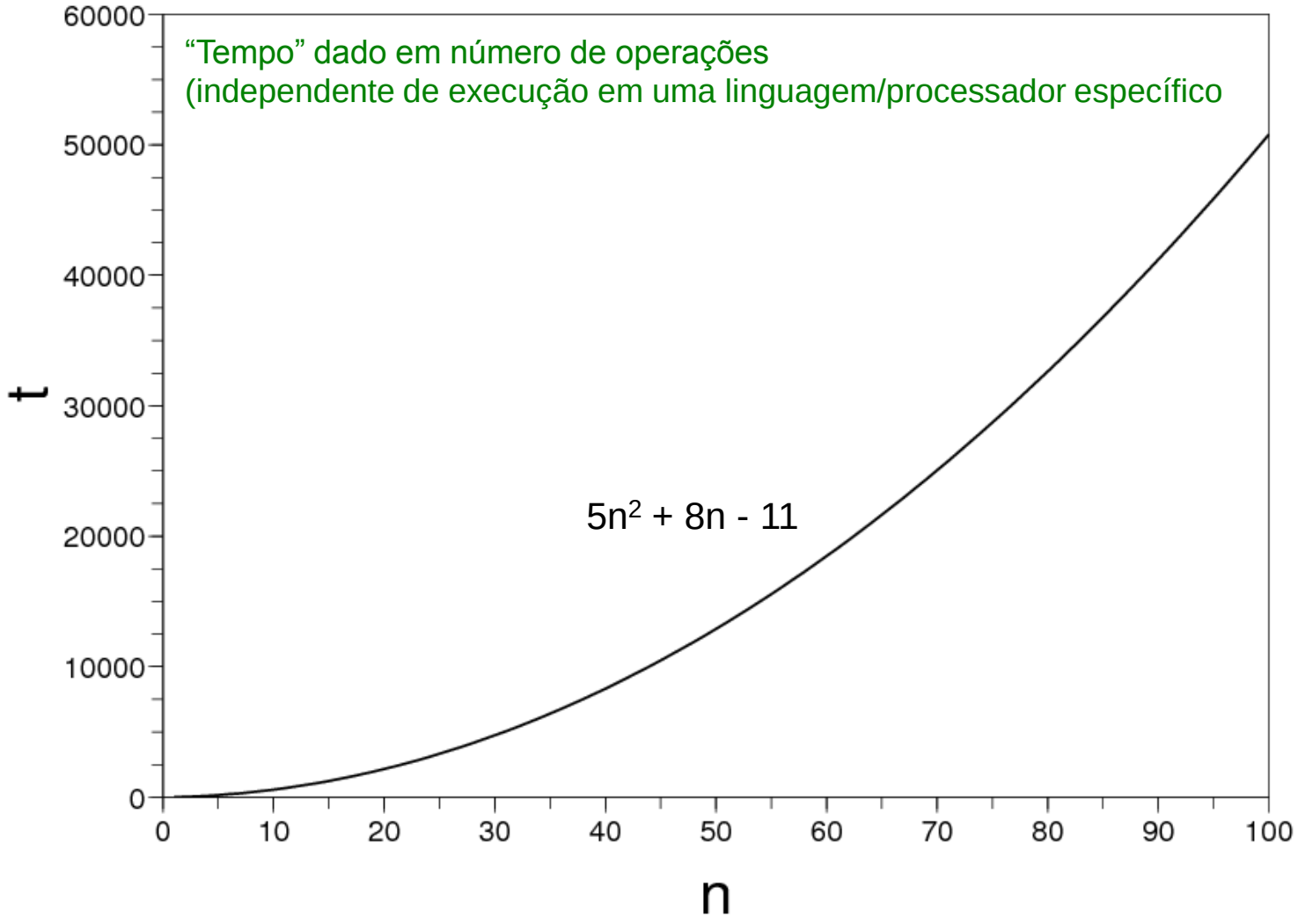
$= 9n - 7 + (10n^2 - 2n - 8)/2 =$

$= 9n - 7 + 5n^2 - n - 4 =$

$= 5n^2 + 8n - 11$

Portanto, o tempo de execução é uma função de n que varia de acordo com um polinômio de grau 2

Contagem de operações



“Despoluindo” a notação

- Termos constantes não são significativos.
- Termos de menor ordem também não são.

Por que?

“Despoluindo” a notação

- Termos constantes não são significativos.
- Termos de menor ordem também não são.

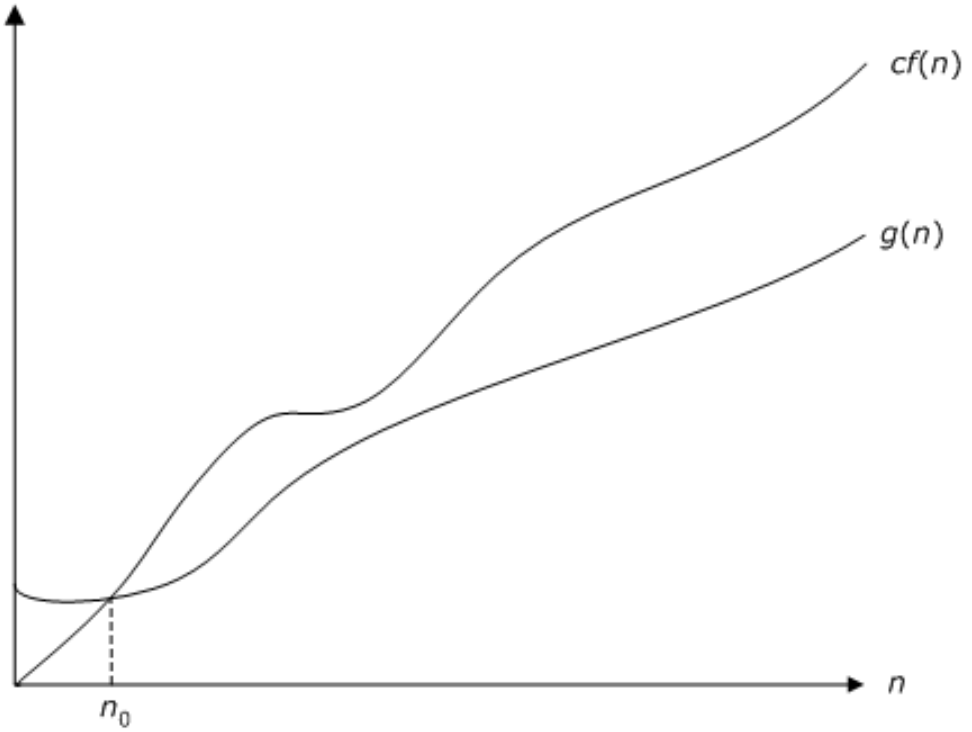
Por que?

Porque considera-se o comportamento assintótico .

Notação Big-O

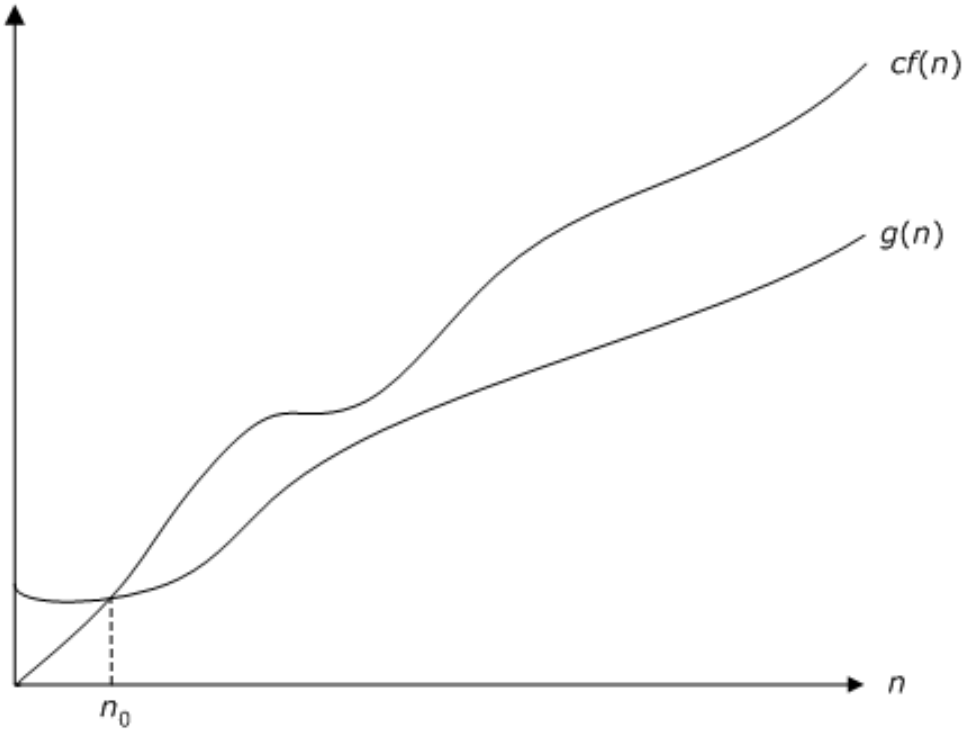
g(n) = O(f(n)) (g é da ordem de f) se existirem constantes c e n_0 tal que $g(n) \leq c*f(n)$ quando $n \geq n_0$

Ou seja, a taxa de crescimento de g(n) é menor ou igual à taxa de f(n) a partir de n_0



Notação Big-O

Ao dizer que g(n) = O(f(n)), mostra-se que g(n) cresce numa taxa não maior do que f(n), ou seja, f(n) é seu limitante superior



Notação Big-O (inserção simples)

Tome $g(n) = 5n^2 + 8n - 11$ (*algoritmo de inserção*)

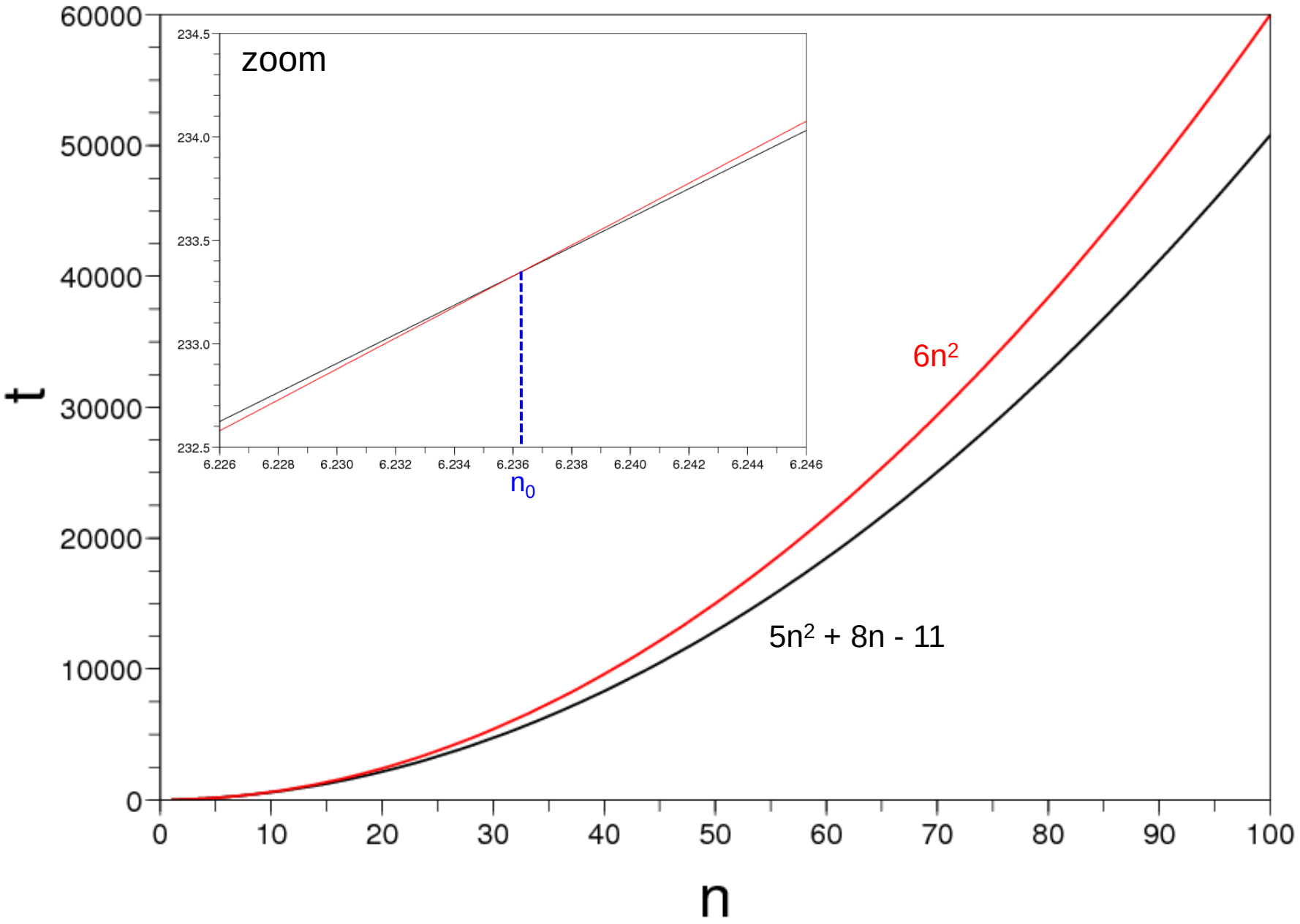
É correto dizer que $g(n) = O(n^2)$?

Notação Big-O (inserção simples)

Tome $g(n) = 5n^2 + 8n - 11$ (*algoritmo de inserção*)

É correto dizer que $g(n) = O(n^2)$?

Sim, pois se tomarmos, por exemplo
$c=6$, então $g(n) \leq c*n^2$ para $n \geq 6,2$ (aprox.)


zoom
t
n
$6n^2$
$5n^2 + 8n - 11$
n_0
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
70
80
90
100
234.5
234.0
233.5
233.0
232.5
6.226
6.228
6.230
6.232
6.234
6.236
6.238
6.240
6.242
6.244
6.246

Inserção simples

- Portanto, a complexidade de tempo do algoritmo de ordenação por inserção é: $O(n^2)$

- Lembre-se que estudamos até aqui o pior caso (vetor inicialmente disposto em ordem inversa)

Análise assintótica de algoritmos

- A análise assintótica de um algoritmo determina o seu tempo de execução em notação Big-O.

- Para fazer a análise assintótica:
  - Encontra-se o número de operações primitivas executado pelo algoritmo (sempre em função do tamanho da entrada).
  - Escreve-se essa função com notação Big-O.

- Como os fatores constantes e termos de menor ordem são descartados, podemos "esquecê-los".

Cuidados

- Deve-se ter sempre em mente que a análise assintótica "esconde" fatores irrelevantes, mas que em alguns casos podem ser relevantes, particularmente se o problema de interesse se limitar a entradas (relativamente) pequenas na prática.

Tamanho da entrada (n)	**1**	**10**	**100**	**1.000**	**10.000**
n^2+n	*2*	*110*	*10.100*	*1.001.000*	*100.010.000*
1.000n	*1.000*	*10.000*	*100.000*	*1.000.000*	*10.000.000*

Análise assintótica de espaço

- Podemos também utilizar a análise assintótica para analisar a ocupação de memória de um algoritmo (complexidade de espaço).

- No caso da ordenação por inserção: n + 4
  (aproximação sem contar o número de bytes de cada variável)

- Complexidade de espaço: O(n) → linear

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

Inserção Simples

- Complexidade de tempo no pior caso
  - Vetor ordenado inversamente: $O(n^2)$
- E no melhor caso ?
  - Vetor já ordenado: O(???)

Inserção Simples (melhor caso)

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

4 operações (testa somente uma vez)

0 operações (não entra no loop)

(n-1)* 2 operações

Inserção Simples (melhor caso)

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

4 operações

0 operações

(n-1)* 2 operações

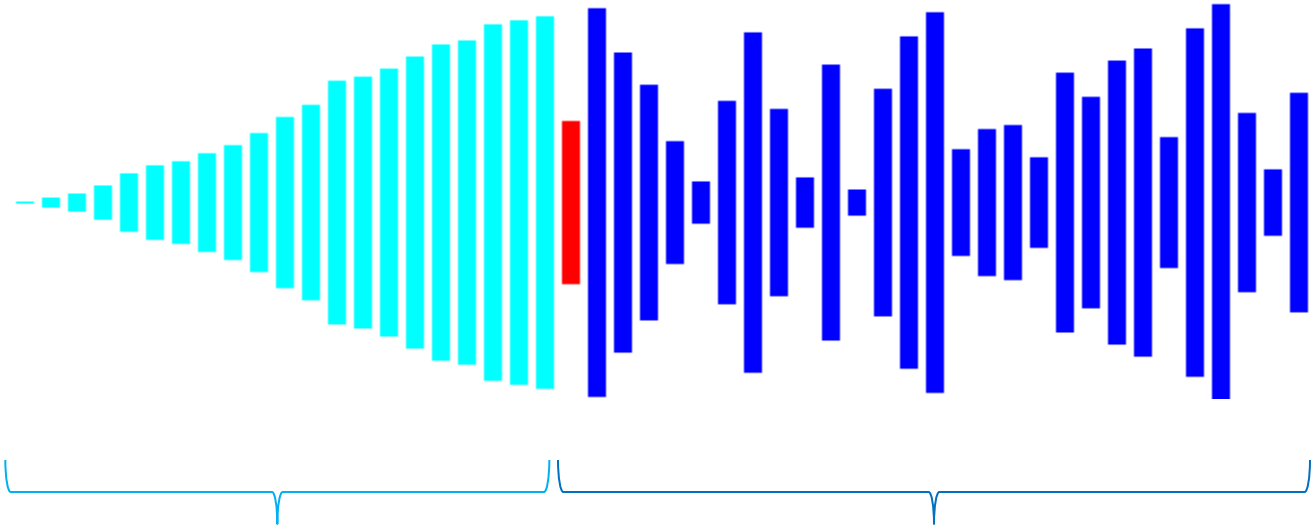
$$3n - 1 + 6*(n-1) + 4 = 9n - 3 \rightarrow O(n)$$

Inserção Simples

- Complexidade de tempo no pior caso
  - Vetor ordenado inversamente: $O(n^2)$
- Complexidade de tempo no melhor caso
  - Vetor já ordenado: $O(n)$
- E no caso médio ?

Inserção Simples (caso médio)

Assume-se que, na média, metade da parte ordenada deve ser percorrida para que a inserção de v[i] seja realizada



Parte ordenada
`v[0] ... v[i-1]`

Parte não-ordenada
`v[i] ... v[n-1]`

Inserção Simples (caso médio)

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

3n - 1 operações

(n-1)*2 operações

(n-1)* 2 operações

$4*(n^2 + n - 2)/2$ operações

$6*(n^2 – n)/2$ operações

(n-1)* 2 operações

Na média, metade dessas operações serão executadas

Inserção Simples (caso médio)

Número de operações primitivas executadas no algoritmo de ordenação por inserção:

$3n - 1 + 2(n-1) + 2(n-1) + [4(n^2 + n - 2)/2]/2 + [6(n^2 - n)/2]/2 + 2(n-1) =$

$= 3n - 1 + 6(n-1) + [4(n^2 + n - 2) + 6(n^2 - n)]/4 =$

$= 9n - 7 + (10n^2 - 2n - 8)/4 =$

$= 9n - 7 + 2{,}5n^2 - 0{,}5n - 2 =$

$= 2{,}5n^2 + 8{,}5n - 9$

Portanto, $O(n^2)$

Inserção Simples

- Complexidade de tempo no pior caso
  - $O(n^2)$
- Complexidade de tempo no melhor caso
  - $O(n)$
- Complexidade de tempo no caso médio
  - $O(n^2)$

Inserção Simples

- Complexidade de tempo no pior caso
  - $O(n^2)$
- Complexidade de tempo no melhor caso
  - $O(n)$
- Complexidade de tempo no caso médio
  - $O(n^2)$

Dependendo do caso, pode ser necessária uma análise probabilistica do caso médio

Inserção Simples

- Complexidade de tempo no pior caso
  - $O(n^2)$

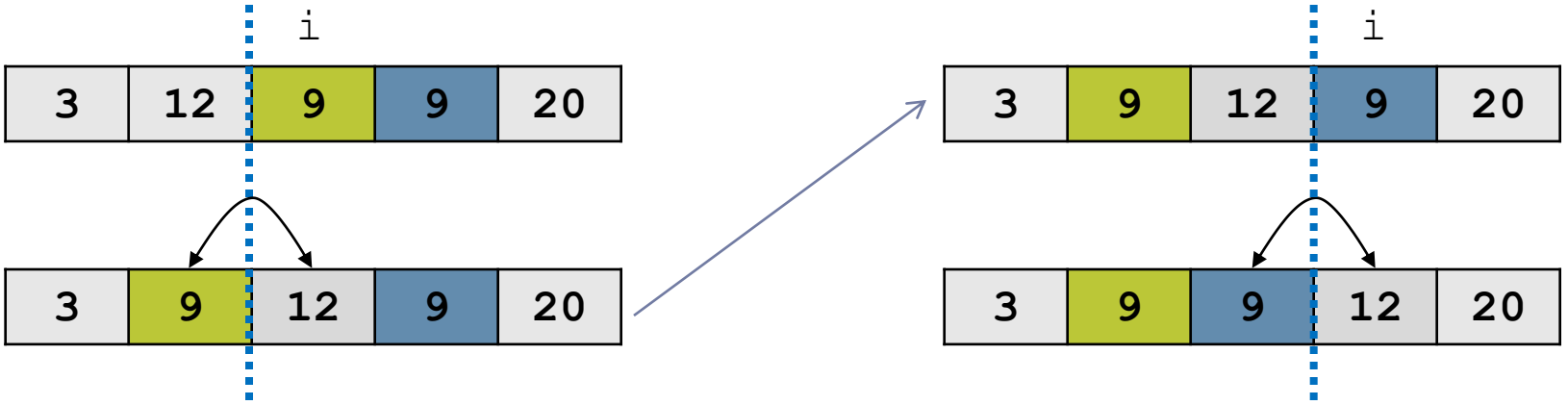
Pior caso costuma ser mais utilizado, por indica que o algoritmo não terá um desempenho menor que esse

- Complexidade de tempo no melhor caso
  - $O(n)$

- Complexidade de tempo no caso médio
  - $O(n^2)$

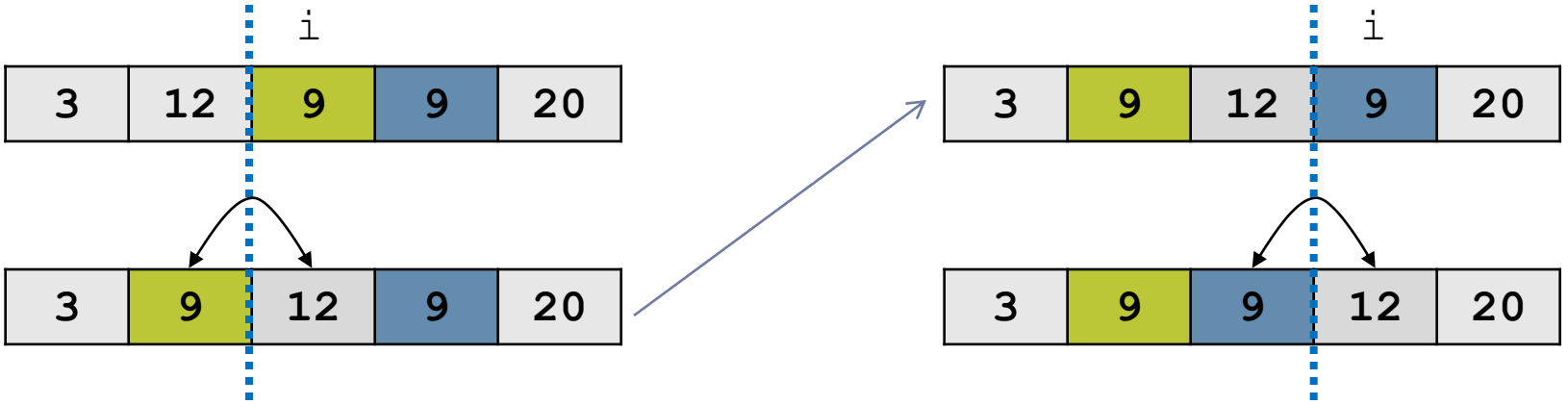
Algoritmo de inserção

- O algoritmo de inserção é estável?



Algoritmo de inserção

- O algoritmo de inserção é estável?



É estável