CI165 — Análise de Algoritmos Iterativos

André Vignatti

Ordenação

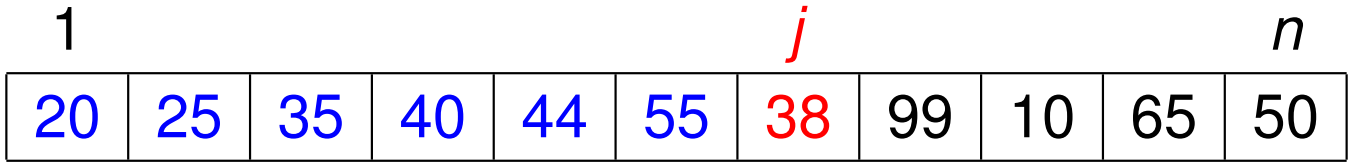
Problema: ordenar um vetor em ordem crescente

Entrada: um vetor $A[1 \ldots n]$

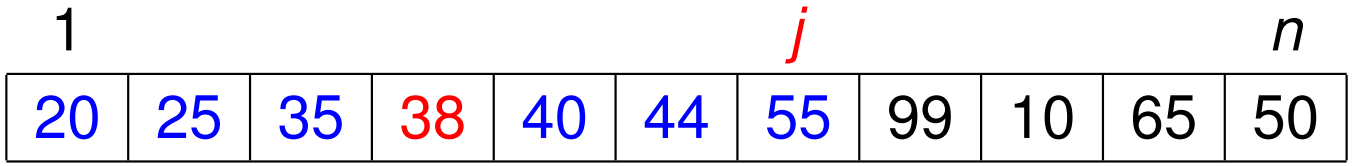
Saída: vetor $A[1 \ldots n]$ rearranjado em ordem crescente

Vamos começar estudando o algoritmo de ordenação baseado no método de inserção.

Inserção em um vetor ordenado

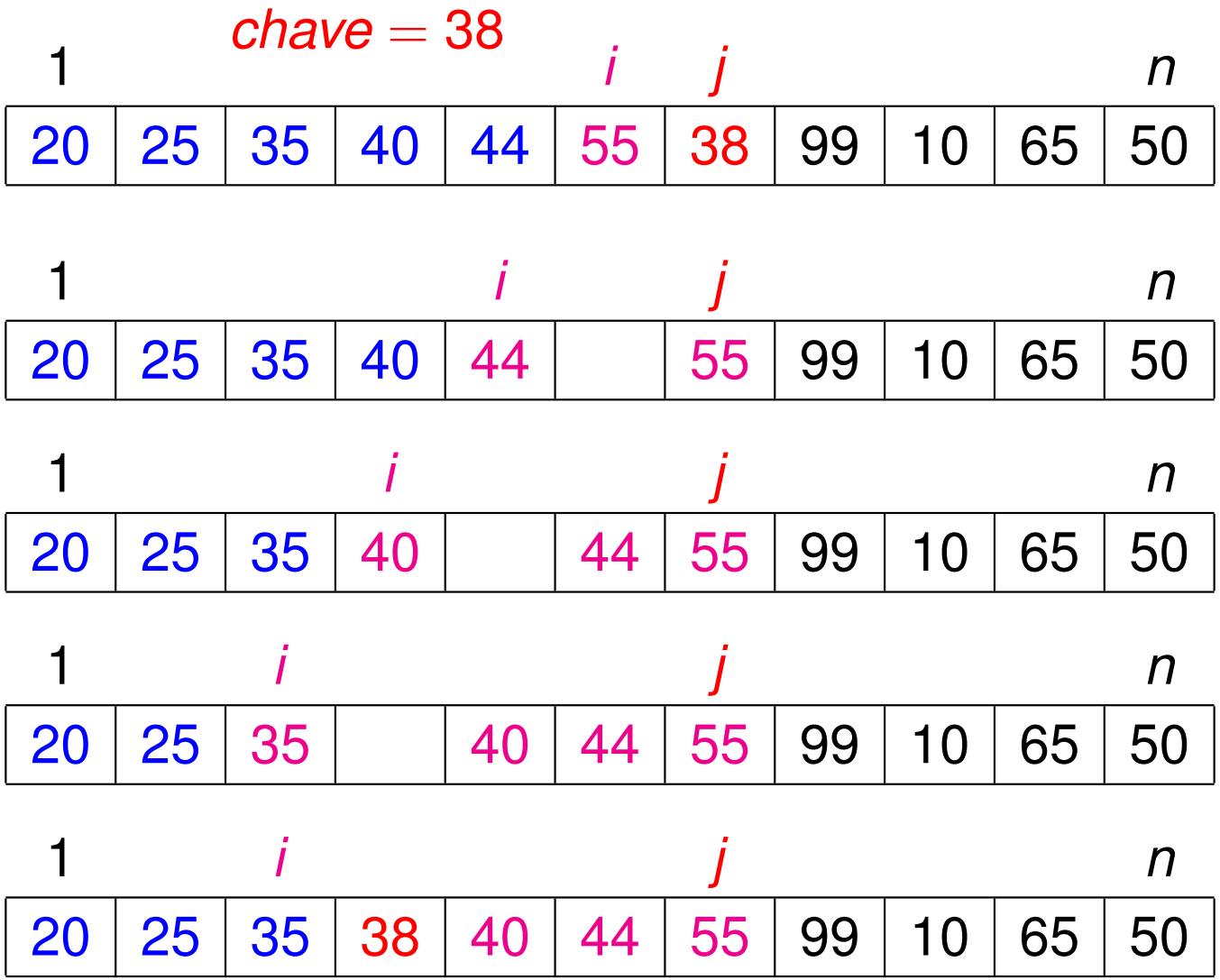


- O subvetor $A[1 \ldots j-1]$ está ordenado.
- Queremos inserir a $chave = 38 = A[j]$ em $A[1 \ldots j-1]$ de modo que no final tenhamos:



- Agora $A[1 \ldots j]$ está ordenado.

Como fazer a inserção



Ordenação por inserção

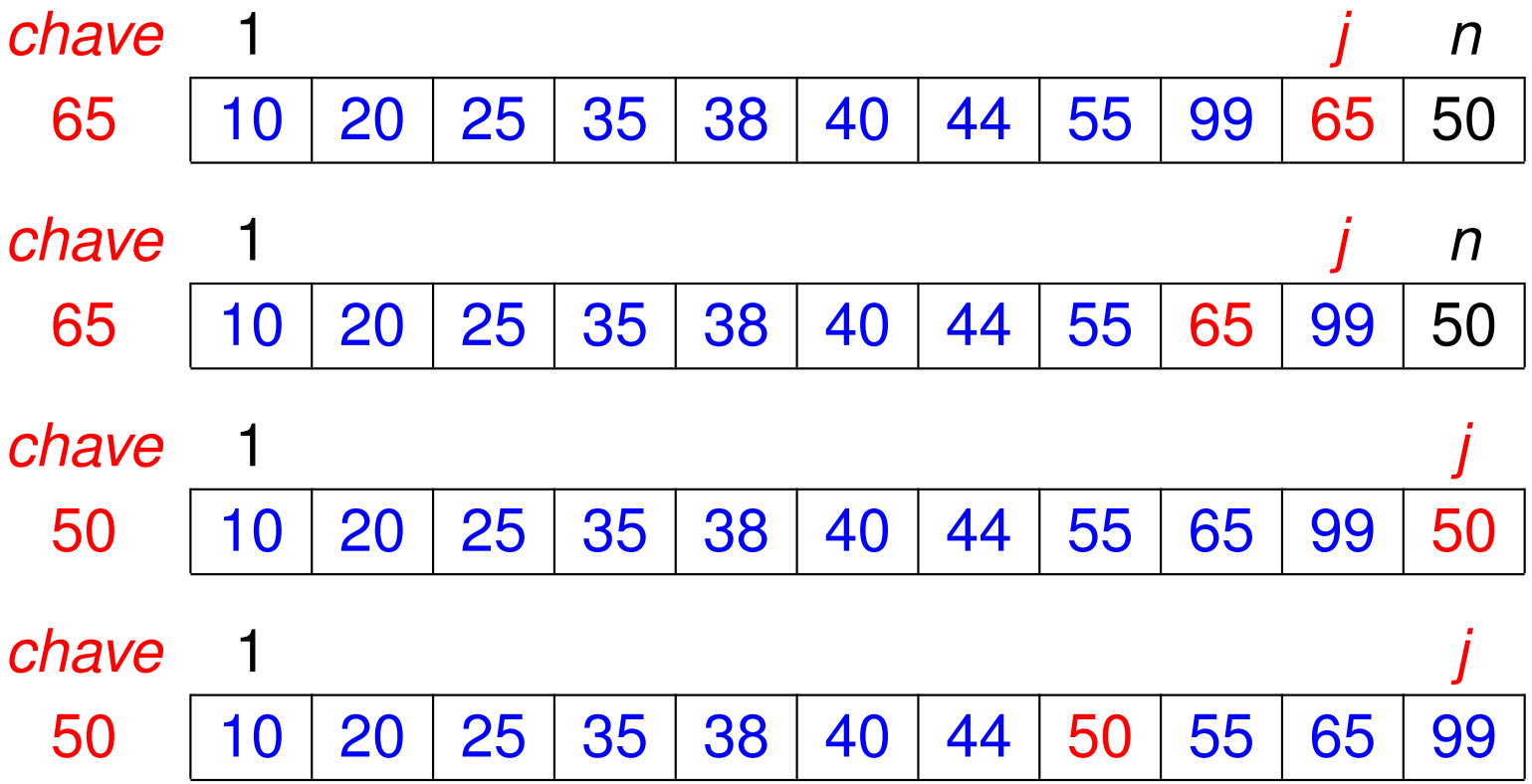
	1							*j*		*n*
chave 99	20	25	35	38	40	44	55	99	10	65	50

	1							*j*		*n*
chave 99	20	25	35	38	40	44	55	99	10	65	50

	1								*j*	*n*
chave 10	20	25	35	38	40	44	55	99	10	65	50

	1								*j*	*n*
chave 10	10	20	25	35	38	40	44	55	99	65	50

Ordenação por inserção



Pseudo-código

```
ORDENA-POR-INSERÇÃO(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       ▷ Insere A[j] no subvetor ordenado A[1..j − 1]
4       i ← j − 1
5       enquanto i ≥ 1  e A[i] > chave faça
6           A[i + 1] ← A[i]
7           i ← i − 1
8       A[i + 1] ← chave
```

Análise do algoritmo

O que é importante analisar ?

- Finitude: o algoritmo pára?
- Corretude: o algoritmo faz o que promete?
- Complexidade de tempo: quantas intruções são necessárias no **pior caso** para ordenar os n elementos?

O algoritmo pára

```
ORDENA-POR-INSERÇÃO(A, n)
1   para j ← 2 até n faça
    ...
4       i ← j − 1
5       enquanto i ≥ 1  e A[i] > chave faça
6           ...
7           i ← i − 1
8       ...
```

No **enquanto** na linha 5 o valor de i diminui a cada iteração e o valor inicial é $i = j - 1 \geq 1$. Logo, a sua execução pára devido ao teste condicional $i \geq 1$.

O laço na linha 1 evidentemente pára (o contador j atingirá o valor $n + 1$ após $n - 1$ iterações).

Portanto, o algoritmo **pára**.

Ordena-Por-Inserção

```
ORDENA-POR-INSERÇÃO(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       ▷ Insere A[j] no subvetor ordenado A[1..j − 1]
4       i ← j − 1
5       enquanto i ≥ 1  e A[i] > chave faça
6           A[i + 1] ← A[i]
7           i ← i − 1
8       A[i + 1] ← chave
```

O que falta fazer ?

- Verificar se ele produz uma resposta correta.
- Analisar sua complexidade de tempo.

Invariante Principal

```
ORDENA-POR-INSERÇÃO(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       ▷ Insere A[j] no subvetor ordenado A[1..j − 1]
4       i ← j − 1
5       enquanto i ≥ 1  e A[i] > chave faça
6           A[i + 1] ← A[i]
7           i ← i − 1
8       A[i + 1] ← chave
```

Invariante principal de ORDENA-POR-INSERÇÃO: (i1)

No começo de cada iteração j do laço **para** das linha 1–8, o subvetor $A[1 \ldots j-1]$ está ordenado.

Esboço da demonstração de corretude

(Isso é um **esboço**, não é a demonstração formal!)

1. **Base**: com $j = 2$ o invariante simplesmente afirma que $A[1 \ldots 1]$ está ordenado, o que é evidente.
2. **Hipótese**: Os elementos "anteriores" já estão ordenados.
3. **Passo**: O algoritmo empurra os elementos maiores que a chave para seus lugares corretos e ela é colocada no espaço vazio.
4. Uma demonstração formal exige invariantes auxiliares para o laço interno **enquanto**.
5. **Corretude do algoritmo**: na última iteração, temos $j = n + 1$ e logo $A[1 \ldots n]$ está ordenado com os elementos originais do vetor. Portanto, o algoritmo é correto.

Complexidade do algoritmo

Vamos determinar o tempo de execução (ou complexidade de tempo) de ORDENA-POR-INSERÇÃO em função do tamanho de entrada.

- Na ordenação: usar como tamanho de entrada a dimensão do vetor e ignorar os valores dos seus elementos (modelo RAM).

Complexidade de Tempo:

Complexidade de tempo é o número de instruções básicas (operações elementares ou primitivas) que executa a partir de uma entrada.

Exemplo: comparação e atribuição entre números ou variáveis numéricas, operações aritméticas, etc.

Vamos contar ?

ORDENA-POR-INSERÇÃO(A, n)	Custo	# execuções
1 **para** $j \leftarrow 2$ **até** n **faça**	c_1	?
2 $\quad$ *chave* $\leftarrow A[j]$	c_2	?
3 $\quad \triangleright$ Insere $A[j]$ em $A[1 \ldots j-1]$	0	?
4 $\quad i \leftarrow j-1$	c_4	?
5 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] >$ *chave* **faça**	c_5	?
6 $\quad\quad A[i+1] \leftarrow A[i]$	c_6	?
7 $\quad\quad i \leftarrow i-1$	c_7	?
8 $\quad A[i+1] \leftarrow$ *chave*	c_8	?

A constante c_k representa o custo (tempo) de cada execução da linha k.

Denote por t_j o número de vezes que o teste no laço **enquanto** na linha 5 é feito para aquele valor de j.

Vamos contar ?

ORDENA-POR-INSERÇÃO(A, n)	Custo	Vezes
1 **para** $j \leftarrow 2$ **até** n **faça**	c_1	n
2 $\quad$ *chave* $\leftarrow A[j]$	c_2	$n-1$
3 $\quad \triangleright$ Insere $A[j]$ em $A[1 \ldots j-1]$	0	$n-1$
4 $\quad i \leftarrow j-1$	c_4	$n-1$
5 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] >$ *chave* **faça**	c_5	$\sum_{j=2}^{n} t_j$
6 $\quad\quad A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $\quad\quad i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $\quad A[i+1] \leftarrow$ *chave*	c_8	$n-1$

A constante c_k representa o custo (tempo) de cada execução da linha k.

Denote por t_j o número de vezes que o teste no laço **enquanto** na linha 5 é feito para aquele valor de j.

Tempo de execução total

Logo, o tempo total de execução $T(n)$ é a soma dos tempos de cada uma das linhas do algoritmo, ou seja:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j \\ &\quad + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7 \sum_{j=2}^{n}(t_j - 1) \\ &\quad + c_8(n-1) \end{aligned}$$

Entradas de tamanho igual (i.e., mesmo valor de n), podem apresentar tempos de execução diferentes já que o valor de $T(n)$ depende dos valores dos t_j.

Melhor caso

O melhor caso de Ordena-Por-Inserção ocorre quando o vetor A já está ordenado. Para $j = 2, \ldots, n$ temos $A[i] \leq chave$ na linha 5 quando $i = j - 1$. Assim, $t_j = 1$ para $j = 2, \ldots, n$.

Logo,

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

Este tempo de execução é da forma $an + b$ para constantes a e b que dependem apenas dos c_i.

Portanto, **no melhor caso**, o tempo de execução é uma **função linear** no **tamanho da entrada**.

Pior Caso

Quando o vetor A está em ordem decrescente, ocorre o pior caso para Ordena-Por-Inserção. Para inserir a *chave* em $A[1 \ldots j-1]$, temos que compará-la com todos os elementos neste subvetor. Assim, $t_j = j$ para $j = 2, \ldots, n$.

Lembre-se que:

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

e

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}.$$

Pior caso – continuação

Temos então que

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

O tempo de execução no pior caso é da forma $an^2 + bn + c$ onde a, b, c são constantes que dependem apenas dos c_i.

Portanto, **no pior caso**, o tempo de execução é uma **função quadrática** no **tamanho da entrada**.

Complexidade assintótica de algoritmos

Como já dito, na maior parte desta disciplina, nos concentraremos na **análise de pior caso** e no **comportamento assintótico** dos algoritmos (instâncias de tamanho grande).

- O algoritmo tem como complexidade (de pior caso) uma função quadrática $an^2 + bn + c$, onde a, b, c são constantes que dependem apenas dos custos c_i.

O estudo assintótico "joga para debaixo do tapete" as **constantes** e **termos de baixa ordem**.

PERGUNTA:

Por que podemos fazer isso ?

Análise assintótica de funções quadráticas

Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

Como se vê, $3n^2$ é o termo dominante quando n é grande.

De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.

Notação assintótica

- Usando notação assintótica, dizemos que o algoritmo Ordena-Por-Inserção tem complexidade de tempo de pior caso $\Theta(n^2)$.
- Isto quer dizer duas coisas:
  - a complexidade de tempo é limitada (superiormente) por algum polinômio da forma an^2
  - existe alguma instância de tamanho n que consome tempo pelo menos dn^2, para alguma constante positiva d.
- Mais adiante discutiremos em detalhes o uso da notação assintótica em análise de algoritmos.