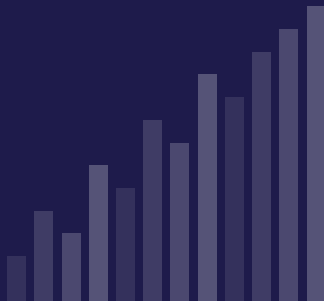

AULA 2 · BLOCO 1: FUNDAMENTOS, ORDENAÇÃO E CUSTO

Ordenação por inserção e invariantes de laço

Seção 2.1

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Executar a ordenação por inserção à mão, passo a passo
- ▶ Provar a corretude com um invariante de laço: inicialização, manutenção e término
- ▶ Traduzir o pseudocódigo (índices $1..n$) para C (índices $0..n - 1$)
- ▶ Medir comparações e confrontar com $n - 1$, $n^2/4$ e $n(n - 1)/2$



Teoria

O problema da ordenação, o algoritmo e a prova por invariante



Prática em C

Implementação em C e testes com a entrada do livro



Experimento

Contar comparações em entradas ordenadas, aleatórias e invertidas



Conexões

Timsort, sistemas embarcados, fluxos de dados e verificação formal

O problema da ordenação

- ▶ **Entrada:** uma sequência de n números $\langle a_1, a_2, \dots, a_n \rangle$.
- ▶ **Saída:** uma permutação $\langle a'_1, a'_2, \dots, a'_n \rangle$ com $a'_1 \leq a'_2 \leq \dots \leq a'_n$.
- ▶ Os números são as **chaves**; na prática, cada chave carrega *dados satélite* (um registro inteiro).
- ▶ Ordenação por inserção: eficiente para n pequeno, **no lugar** (*in place*) e **estável**.
- ▶ Ideia: como ordenamos cartas na mão. Cada carta nova é comparada da direita para a esquerda até achar seu lugar.

Um algoritmo é **correto** se, para toda entrada, ele para com a saída certa. Testes mostram erros; só a prova mostra corretude.

O algoritmo

- ▶ j aponta a “carta” que será inserida.
- ▶ $A[1..j - 1]$ já está ordenado: é a mão.
- ▶ O **enquanto** desloca para a direita todos os maiores que a chave.
- ▶ A chave cai no buraco aberto em $A[i + 1]$.

ORDENA-INserção(A)

```
1 para  $j \leftarrow 2$  até  $A.comprimento$  faça  
2    $chave \leftarrow A[j]$   
3   // insere  $A[j]$  na sequência ordenada  
    $A[1..j - 1]$   
4    $i \leftarrow j - 1$   
5   enquanto  $i > 0$  e  $A[i] > chave$  faça  
6      $A[i + 1] \leftarrow A[i]$   
7      $i \leftarrow i - 1$   
8    $A[i + 1] \leftarrow chave$ 
```

Execução em $A = \langle 5, 2, 4, 6, 1, 3 \rangle$

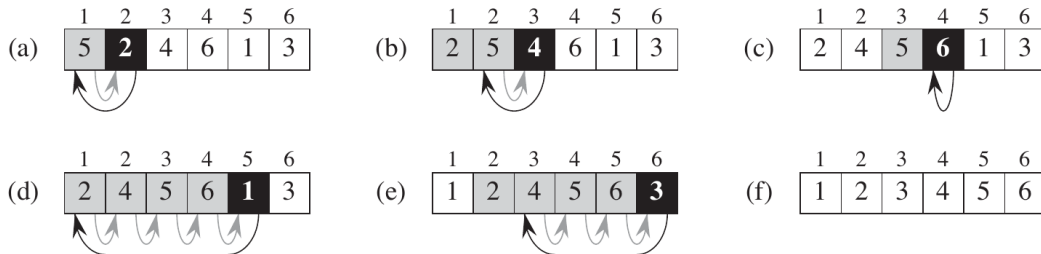


Figura 2.2 de Cormen et al. (2009) — (a)–(e) cada volta do **para**: o retângulo preto é $A[j]$; as setas mostram os deslocamentos. (f) vetor ordenado.

Em (d), a chave 1 é a menor de todas: o **enquanto** percorre a mão inteira. Esse é o caso que custa caro.

Invariante de laço

No início de cada iteração do **para** (linhas 1–8), o subvetor $A[1..j - 1]$ contém os elementos que estavam originalmente em $A[1..j - 1]$, **em ordem**.

Para usar um invariante, mostramos três coisas:

1. **Inicialização:** vale antes da primeira iteração.
2. **Manutenção:** se vale antes de uma iteração, vale antes da próxima.
3. **Término:** quando o laço acaba, o invariante dá uma propriedade útil.

É a **indução matemática**: inicialização é a base, manutenção é o passo. A diferença: paramos quando o laço termina.

A prova de corretude

- ▶ **Inicialização.** Com $j = 2$, $A[1..1]$ tem um só elemento, o original, e um elemento sozinho está ordenado.
- ▶ **Manutenção.** O corpo do laço move $A[j - 1], A[j - 2], \dots$ uma posição à direita até achar o lugar de $A[j]$ e o insere ali. Então $A[1..j]$ tem os elementos originais, em ordem. Incrementar j restaura o invariante.
- ▶ **Término.** O laço para quando $j = n + 1$. Substituindo no invariante: $A[1..n]$ tem os elementos originais, em ordem. Ou seja, **o vetor inteiro está ordenado.**

Rigor completo exigiria um invariante também para o **enquanto** interno (exercício 2.1-3 usa a mesma técnica na busca linear).



Prática em C

- ▶ Do pseudocódigo ($1..n$) para C ($0..n - 1$)
- ▶ Leitura com malloc, ordenação e impressão
- ▶ Teste com a entrada da Figura 2.2

Do livro para o C

Cormen	C
$A[1..n]$	$A[0]$ a $A[n-1]$
para $j \leftarrow 2$ até n	for ($j = 1$; $j < n$; $j++$)
$i > 0$	$i \geq 0$
$A.comprimento$	parâmetro n

O erro mais comum: copiar o $i > 0$ do livro. Em C o primeiro elemento é $A[0]$; com $i > 0$ ele nunca é comparado.

$A[i] > \text{chave}$ (estrito) mantém a ordem dos iguais: o algoritmo é **estável**.

Ordenação por inserção em C (1/2)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* INSERTION-SORT(A): o livro indexa de 1 a n; em C, de 0 a n-1.
5     Invariante: no início de cada volta, A[0..j-1] contém os elementos
6     originais de A[0..j-1], em ordem. */
7  void ordena_insercao(int A[], int n) {
8      for (int j = 1; j < n; j++) {          // linha 1 do livro: j = 2 até n
9          int chave = A[j];                 // a "carta" que será inserida
10         int i = j - 1;
11         while (i >= 0 && A[i] > chave) {    // desloca os maiores uma posição à direita
12             A[i + 1] = A[i];
13             i--;
14         }
```

Ordenação por inserção em C (2/2)

```
15         A[i + 1] = chave;                // a chave cai no buraco aberto
16     }
17 }
18
19 int main(void) {
20     int n;
21     if (scanf("%d", &n) != 1 || n <= 0) return 1;
22     int *A = malloc(n * sizeof(int));      // vetor do tamanho pedido
23     for (int k = 0; k < n; k++)
24         if (scanf("%d", &A[k]) != 1) return 1;
25     ordena_insercao(A, n);
26     for (int k = 0; k < n; k++) printf("%d%c", A[k], k < n - 1 ? ' ' : '\n');
27     free(A);
28     return 0;
29 }
```



Experimento

- ▶ Hipótese: $n - 1$, $\approx n^2/4$ e $n(n - 1)/2$ comparações
- ▶ Contador instrumentado no teste $A[i] > chave$
- ▶ De $n = 500$ a 8000, semente fixa

Quantas comparações a inserção faz?

Hipótese. Para n elementos, o teste $A[i] > chave$ é feito

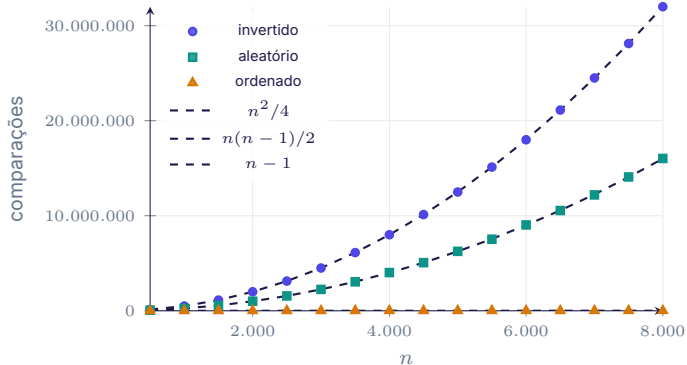
- ▶ $n - 1$ vezes se o vetor já está **ordenado** (melhor caso);
- ▶ $n(n - 1)/2$ vezes se está **invertido** (pior caso);
- ▶ cerca de $n^2/4$ vezes numa ordem **aleatória**: cada par fora de ordem (inversão) custa uma comparação, e em média metade dos $\binom{n}{2}$ pares está invertida.

Método

1. Variável global `comparacoes` incrementada dentro da condição do `while`
2. n de 500 a 8000, passo 500
3. Aleatório: média de 5 permutações (`srand(2026)`, reproduzível)
4. Saída em CSV:
`n,ordenado,aleatorio,invertido`

EXPERIMENTO

Comparações medidas e previstas



n	medido	razão
500,0	64.358	1,030
2.000	1.005.586	1,006
3.500	3.048.098	0,995
5.000	6.256.426	1,001
6.500	10.548.037	0,999
8.000	16.030.699	1,002

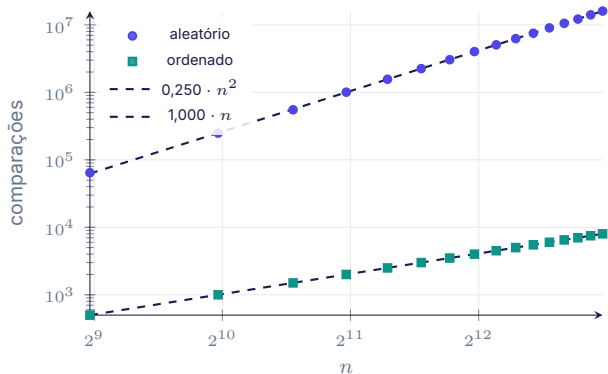
razão = medido $\div$ $n^2/4$

✓ desvio máx. 0,5%

A razão medido/ $(n^2/4)$ fica colada em 1: a média de inversões de uma permutação aleatória é $n(n-1)/4$.

EXPERIMENTO

Em escala log-log, o expoente vira inclinação



<i>n</i>	medido	razão
500,0	64.358	0,257
2.000	1.005.586	0,251
3.500	3.048.098	0,249
5.000	6.256.426	0,250
6.500	10.548.037	0,250
8.000	16.030.699	0,250

razão = medido $\div n^2$

✓ desvio máx. 0,5%

Se $T(n) = c n^k$, então
 $\log T = \log c + k \log n$: uma reta de
inclinação k . Aleatório: inclinação 2.
Ordenado: inclinação 1.

EXPERIMENTO

O que o experimento mostrou

As três previsões se confirmam: **exatamente** $n - 1$ e $n(n - 1)/2$ nos casos extremos, e $n^2/4$ com desvio abaixo de 3% no caso aleatório. A ordenação por inserção é **linear** em dados ordenados e **quadrática** em dados aleatórios: o tipo de entrada importa tanto quanto o tamanho.

Desafio: conte também os deslocamentos ($A[i + 1] \leftarrow A[i]$) e compare com o número de inversões da entrada (problema 2-4).

>_ Reproduza: `gcc -O2 aula02_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Bibliotecas de linguagens
- ▶ Sistemas embarcados
- ▶ Fluxos de dados
- ▶ Verificação formal

Onde isso aparece na computação

Bibliotecas de linguagens

O Timsort (sorted do Python, Arrays.sort de objetos no Java) ordena trechos curtos com inserção binária: para n pequeno, ela vence algoritmos $n \lg n$.

Sistemas embarcados

No lugar, sem recursão e com código minúsculo: cabe em microcontroladores com poucos KB de RAM, onde `malloc` e pilha profunda são luxo.

Fluxos de dados

É um algoritmo *online*: ordena à medida que os dados chegam. Manter um placar ao vivo ou as k maiores leituras de um sensor é uma inserção por evento.

Verificação formal

Invariantes de laço são a base de ferramentas como Dafny, Frama-C (ACSL) e Why3, que provam código real. Sem o invariante, o provador não consegue.

Para casa

Exercícios

1. (2.1-1) Ilustre a execução em $A = \langle 31, 41, 59, 26, 41, 58 \rangle$, como na Figura 2.2.
2. (2.1-2) Reescreva o pseudocódigo para ordenar em ordem **decrecente**.
3. (2.1-3) Escreva a busca linear e prove sua corretude com um invariante de laço.
4. Altere o experimento para contar deslocamentos e verifique: deslocamentos = número de inversões.
5. Meça o **tempo** (com `clock()`) além das comparações. A forma da curva muda?

Leitura

Cormen et al., seção 2.1 e problema 2-4 (inversões).

Código

josedihago.net/estruturas