

Departamento de Engenharia de Sistemas e Computação

Ordenando Coisas

Método da bolha, inserção, intercalação e mergesort

João Araujo

Algoritmos Computacionais I

Aula C2 (Eng. de Computação)

Aula extra para Engenharia de Computação

Onde se encaixa

Capítulo **8** do livro; continua a aula C1 (Recursão). Não faz parte do curso genérico para as demais engenharias.

Roteiro

- Por que ordenar? (Cap. 8)
- Ordenando uma lista: método da bolha (8.1) e sentinela
- Ordenação por inserção (8.2)
- Intercalação de listas e *mergesort* (Exercícios 8.5 e 8.6)
- Comparando algoritmos; a ordenação do Python

Por que ordenar?

- Dados ordenados são **muito mais fáceis de usar**: lista telefônica, dicionário, classificação de notas
- Buscar em dados ordenados é rápido: busca binária (Aula 4): $\log_2 n$ passos
- Muitos algoritmos assumem entrada ordenada: intercalar, remover duplicatas, achar medianas

Problema

Dada uma lista de n números (ou nomes, ou registros), reorganizá-la em ordem crescente.

Veremos **três** algoritmos: dois simples ($\approx n^2$ operações) e um eficiente ($\approx n \log n$).

O método da bolha (*bubble sort*): a ideia

1. Compare cada par de elementos **consecutivos**
2. Se estão fora de ordem, **troque-os**
3. Ao fim de uma passada, o maior elemento está na última posição ("a bolha sobe")
4. Repita: o segundo maior ocupa a penúltima posição, etc.
5. Após $n - 1$ passadas, a lista está ordenada

Por que $n - 1$ passadas?

Cada passada coloca *pelo menos* um elemento na posição final. Com $n - 1$ elementos no lugar, o último também está.

Acompanhando a bolha: $[5, 2, 4, 1, 3]$

						Observação
início	5	2	4	1	3	
1ª passada	2	4	1	3	**5**	o maior foi para o fim
2ª passada	2	1	3	**4**	**5**	
3ª passada	1	2	**3**	**4**	**5**	já ordenada
4ª passada	1	2	3	4	5	passada "inútil"

Na 1ª passada: `(5,2)` troca; `(5,4)` troca; `(5,1)` troca; `(5,3)` troca.

Reparou? A lista ficou ordenada na 3ª passada, mas o algoritmo faria a 4ª de qualquer jeito.

Bolha em Python

```python
def bubblesort(lista):
    """Ordena uma lista pelo método da bolha."""
    n = len(lista)
    for i in range(n - 1):                # n-1 passadas
        for j in range(n - 1):            # compara pares consecutivos
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]   # troca
        print(lista)                      # mostra cada passada

l = [503, 87, 512, 61, 908, 170, 897, 275]
bubblesort(l)
```

- A **troca** `a, b = b, a` usa o desempacotamento de tuplas (Aula 20)
- Ordena a lista *no lugar* (alias! Aula 18) e devolve `None`

Fonte: `code/Cap8/prog.8.02_bubblesort.py`

Melhoria: a *sentinela*

```
def bubblesort(lista):
    """Ordena uma lista pelo método da bolha, com sentinela."""
    n = len(lista)
    fora_de_ordem = True
    while fora_de_ordem:
        fora_de_ordem = False            # sentinela: nenhuma troca ainda
        for j in range(n - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                fora_de_ordem = True     # houve troca: outra passada
```

- Se uma passada não fez nenhuma troca, a lista já está ordenada: pare
- Para uma lista já ordenada, faz *uma* passada: é o caso em que a bolha é imbatível
- Exercício 8.2: aproveite que o maior elemento já está no lugar para *encurtar* a passada

Fonte: `code/Cap8/prog.8.03_bubblesort_com_sentinela.py`

Ordenação por inserção: a ideia do baralho

Imagine ordenar cartas na mão:

1. Pegue as duas primeiras e ordene-as
2. Pegue a próxima carta e **insira-a no lugar certo** entre as já ordenadas (comparando a partir da última)
3. Repita até acabarem as cartas

passo	lista (parte ordenada em negrito)				
início	5	2	4	1	3
insere 2	**2**	**5**	4	1	3
insere 4	**2**	**4**	**5**	1	3
insere 1	**1**	**2**	**4**	**5**	3
insere 3	**1**	**2**	**3**	**4**	**5**

Inserção em Python

```
def insert_sort(lista):
    """Ordena uma lista pelo método da inserção."""
    for j in range(1, len(lista)):
        elemento = lista[j]                  # a "carta" a inserir
        i = j - 1
        while i >= 0 and lista[i] > elemento:
            lista[i + 1] = lista[i]          # desloca para a direita
            i -= 1
        lista[i + 1] = elemento              # coloca no lugar
        print(lista)

l = [503, 87, 512, 61, 908, 170, 897, 275]
insert_sort(l)
```

- Em vez de trocar, *desloca* elementos e insere uma única vez
- Ótima para listas *quase ordenadas* e pequenas
- Exercício 8.3: construa sua versão passo a passo; Exercício 8.4: versão recursiva (não ganha desempenho)

Ordenando nomes e registros (Exercício 8.1)

```
nomes = ['Maria', 'Ana', 'João', 'Beatriz']
print('Ana' < 'Maria')            # True: strings comparam em ordem alfabética
bubblesort(nomes)                 # mesmo código serve para strings!

alunos = [('Ana', 8.5), ('João', 7.0), ('Beatriz', 9.2)]
por_nota = sorted(alunos, key=lambda a: a[1], reverse=True)
```

- O mesmo algoritmo funciona para qualquer tipo que possa ser comparado com >
- Cuidado com maiúsculas/acentos: `'Zé' < 'ana'` é `True` (código ASCII/Unicode, Aula 2); use `key=str.lower`
- Para registros, compare por um **campo** (nota, nome, data)

Intercalação: juntando duas listas ordenadas (Exercício 8.5)

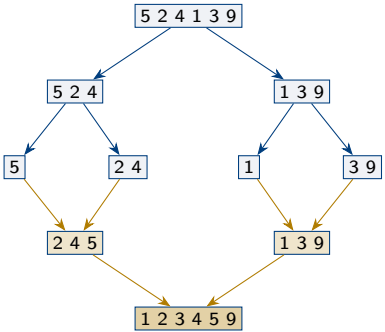
```python
def intercala(lista1, lista2):
    """Intercala duas listas já ordenadas numa terceira, também ordenada."""
    resultado = []
    i = j = 0
    while i < len(lista1) and j < len(lista2):
        if lista1[i] <= lista2[j]:
            resultado.append(lista1[i]); i += 1
        else:
            resultado.append(lista2[j]); j += 1
    resultado.extend(lista1[i:])        # sobras (só uma das listas tem)
    resultado.extend(lista2[j:])
    return resultado

print(intercala([1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))
```

Percorre cada lista *uma única vez*: $n_1 + n_2$ passos. Como duas pilhas de cartas já ordenadas.

Mergesort: dividir para conquistar

1. Se a lista tem 0 ou 1 elemento, já está ordenada (caso base)
2. Divida-a em duas metades
3. Ordene cada metade **recursivamente**
4. **Intercale** as metades ordenadas



Mergesort em Python (Exercício 8.6)

```python
def mergesort(lista):
    """Devolve uma NOVA lista ordenada (não altera a original)."""
    if len(lista) <= 1:                  # caso base
        return lista
    meio = len(lista) // 2
    esquerda = mergesort(lista[:meio])   # ordena a 1a metade
    direita = mergesort(lista[meio:])    # ordena a 2a metade
    return intercala(esquerda, direita)  # junta

print(mergesort([5, 2, 4, 1, 3, 9, 0]))  # [0, 1, 2, 3, 4, 5, 9]
```

- Combina **recursão** (aula C1) e **intercalação**
- Cada nível da recursão processa n elementos e há $\log_2 n$ níveis: $\approx n \log_2 n$ operações

Comparando os algoritmos

	Bolha	Inserção	Mergesort
Comparações (pior caso)	$\approx n^2$	$\approx n^2/2$	$\approx n \log_2 n$
Melhor caso (já ordenada)	n (com sentinela)	n	$n \log_2 n$
Memória extra	nenhuma	nenhuma	$\approx n$
Simplicidade	alta	alta	média (recursão)

n	n^2	$n \log_2 n$
1 000	1 000 000	$\approx$ 10 000
1 000 000	10^{12}	$\approx$ 20 000 000

Medindo o tempo

```
import random, time

for n in (500, 1000, 2000, 4000):
    dados = [random.randint(0, 10**6) for _ in range(n)]
    t0 = time.perf_counter(); bubblesort(dados[:]);        t1 = time.perf_counter()
    mergesort(dados);                                      t2 = time.perf_counter()
    sorted(dados);                                         t3 = time.perf_counter()
    print(f"{n:5d}  bolha {t1-t0:.3f}s  merge {t2-t1:.4f}s  sorted {t3-t2:.5f}s")
```

- Dobrar n: a bolha leva 4 vezes mais; o mergesort pouco mais que 2 vezes
- Remova os `print`s dos algoritmos antes de medir!
- `sorted` e `list.sort` (Aula 18) usam o *Timsort*: mistura de inserção e mergesort, em C

Na prática: use `sorted` e `sort`

- Em programas reais, use `sorted(lista)` ou `lista.sort()`: rápidos, testados e *estáveis*
- `key=` escolhe o critério; `reverse=True` inverte

Então por que aprender os algoritmos?

- Para entender **o que acontece por baixo** e seu custo
- Para desenvolver o pensamento algorítmico (invariantes, casos-limite, eficiência)
- Para reconhecer padrões: *dividir e conquistar*, *inserção incremental*
- Em entrevistas e concursos

Curiosidade: existem outros algoritmos clássicos: seleção, *quicksort*, *heapsort*, ordenação por contagem... (disciplina de Estruturas de Dados).

Atividades

1. Faça o **teste de mesa** da bolha e da inserção para $[6, 3, 8, 2, 9, 1]$.
2. Conte as **trocas** e as **comparações** feitas pela bolha: acrescente contadores ao código. Compare com e sem sentinela.
3. Escreva `selection_sort`: ache o menor elemento e coloque-o na 1ª posição; depois o 2º menor...
4. Ordene uma lista de alunos (dicionários) por nota decrescente e, em caso de empate, por nome.
5. Implemente **busca binária** (Aula 4) em uma lista ordenada e compare com `in`.

Para praticar: exercícios do livro

Site do livro – Capítulo 8

- Bolha: **8.1** (com nomes), **8.2** (com sentinela melhorada)
- Inserção: **8.3** (desenvolva o algoritmo), **8.4** (versão recursiva)
- Intercalação e mergesort: **8.5**, **8.6**

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap8/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Ordenar facilita buscar e processar dados
- Bolha e inserção: simples, $\approx n^2$; sentinela evita passadas inúteis
- Mergesort: dividir, ordenar recursivamente e intercalar: $\approx n \log n$
- Em programas reais, use `sorted` e `list.sort`
- Recursão e ordenação encerram o conteúdo extra de Engenharia de Computação