Complexidade de algoritmos

Um algoritmo é qualquer procedimento computacional bem-definido que toma algum(ns) valor(es) como entrada e produz algum(ns) valor(es) como saída. Um algoritmo é, portanto, uma seqüência de passos computacionais que transforma a entrada na saída [Cormen et al., 1990].

O termo algoritmo é usado para descrever um método para resolver um dado problema, adequado para ser implementado como um programa computacional [Sedgewick, 1992].

Exemplo: Problema de ordenação.
Entrada: Uma seqüência de n números $\langle a_1,\ldots,a_n\rangle$.
Saída: Uma permutação (reordenação) $\langle a'_1,\ldots,a'_n\rangle$ da seqüência de entrada tal que $a'_1 \le \ldots \le a'_n$.
<5,2,4,6,1,3> $\Rightarrow$ <1,2,3,4,5,6>

Insertion-sort

```
5 | 2 4 6 1 3
2 5 | 4 6 1 3
2 4 5 | 6 1 3
2 4 5 6 | 1 3
1 2 4 5 6 | 3
1 2 3 4 5 6 |
```

Melhor caso - entrada ordenada:
O(n): (an+b).

Pior caso - entrada ordenada inversamente:
O(n^2): (an^2+bn+c).

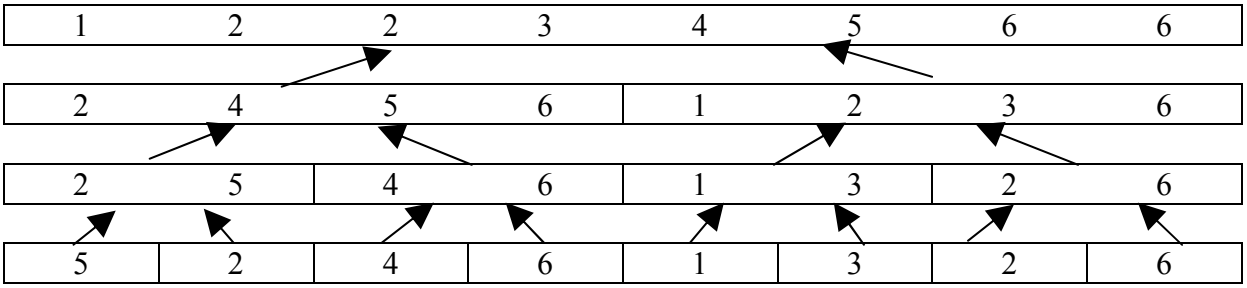
Normalmente, utiliza-se análise do pior caso ou do caso médio. Normalmente, consideramos que um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem uma ordem de crescimento menor. Esta avaliação pode ser errada para as entradas pequenas, mas para entradas suficientemente grandes, vale. Exemplo: $\theta(n)$ melhor que $\theta(n \log n)$, melhor que $\theta(n^2)$, melhor que $\theta(n^3)$, etc.

Problema: Para cada função $f(n)$ e tempo t, determine o maior tamanho n de um problema que pode ser resolvido em tempo t, assumindo que algoritmo que resolve o problema leva $f(n)$ microssegundos.

	10^6 μs	6×10^7 μs	4×10^9 μs	10^{11} μs	2×10^{12} μs	3×10^{13} μs	3×10^{15} μs
	1 s	1 min	1 h	1 dia	1 mês	1 ano	1 século
$\log_2(n)$	$2^{(10^6)}$						
$\sqrt{n}$	10^{12}						
n	**10^6**						
$n \log_2(n)$	8000	**3×10^6**	10^8	3×10^9			
n^2	1000			3×10^5	**10^6**		6×10^7
n^3	100			5000			10^5
2^n	20			36			51
$n!$							18

Merge-sort:

- Divida a seqüência de n elementos a ser ordenada em duas subseqüências de $n/2$ elementos cada.
- Conquiste: ordene duas subseqüências recursivamente usando merge-sort.
- Combine: as duas subseqüências ordenadas para produzir uma resposta ordenada.



O algoritmo funciona mesmo que o tamanho da seqüência não seja 2^d (d inteiro). Mas, para facilitar a análise, consideraremos $n = 2^d$. Isso não afeta a ordem de crescimento.

Merge_sort demora O($n \log_2(n)$).

Para que fique clara a superioridade do merge_sort sobre insertion_sort, considere que um computador 486 utiliza merge_sort e um super-computador usa insertion_sort. Vamos supor que 486 executa 1 milhão de instruções por segundo e supercomputador 100 milhões. Para piorar, vamos supor que o super-computador possui bom compilador (necessita de $2n^2$ instruções para ordenar n números) e 486 utiliza uma linguagem interpretada, como Basic (necessita de $50 \times n \log_2(n)$ instruções para ordenar n números). Os dois têm que ordenar 1 milhão de números.

Super-computador: $\frac{2 \times (10^6)^2 \text{ instruções}}{10^8 \text{ instruções/seg.}} = 20000 \text{ s} \approx 5.5 \text{ horas}$.

486: $\frac{50 \times 10^6 \log_2(10^6) \text{ instruções}}{10^6 \text{ instruções/seg.}} = 1000 \text{ s} \approx 16.7 \text{ minutos}$

Dados obtidos na prática (usando Pentium-100):

	n=100000	n=1000000
Merge_sort	5 s	56 s
Insertion_sort	5 min	?

Exemplo de uso de análise de complexidade no processamento de imagens (separabilidade):

Fazer dilatação por $B = \begin{bmatrix} \boldsymbol{1} & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ é o mesmo que fazer dilatação por $B_1 = \begin{bmatrix} \boldsymbol{1} & 1 & 1 \end{bmatrix}$ seguido por $B_2 = \begin{bmatrix} 1 \\ 1 \\ \boldsymbol{1} \end{bmatrix}$.

Fazer dilatação de uma imagem A com n pixels por um elemento estruturante quadrada B com m pixels ($\sqrt{m} \times \sqrt{m}$) leva normalmente tempo $O(nm)$. Usando a separabilidade, chegamos ao tempo $O(n\sqrt{m})$.

Exemplo: Fazer dilatação de uma imagem 1000×1000 por um elemento estruturante 20×20. Sem separabilidade, preciso efetuar "aproximadamente": 400 milhões de operações. Com separabilidade, o número de operações cai para "aproximadamente": 20 milhões de operações.

Outras operações separáveis de processamento de imagens:

- Média móvel
- Convolução com gaussiana
- Erosão
- FFT
- DCT

Operações não separáveis:

- Filtro mediana
- Convolução com as derivadas da gaussiana