ORDENAÇÃO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Selection sort

Informalmente:

1. Procure o menor elemento da lista e troque com o primeiro elemento. Siga para o Passo 2.
2. Se não terminou de ordenar, repita o Passo anterior, procurando o próximo menor elemento e trocando este pela próxima posição.

$$\underbrace{A_0 \le A_1 \le \cdots \le A_{i-1}}_{\text{in their final positions}} \;\Big|\; \underbrace{A_i, \ldots, A_{min}, \ldots, A_{n-1}}_{\text{the last } n-i \text{ elements}}$$ [1]

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Selection sort

Algoritmo: SelectionSort(A[0..n - 1])

for $i \leftarrow 0$ **to** $n-2$ **do**
  $min \leftarrow i$;
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[j] < A[min]$ **then** $min \leftarrow j$;
  swap $A[i]$ and $A[min]$;

```
| 89    45    68    90    29    34    17
  17  | 45    68    90    29    34    89
  17    29  | 68    90    45    34    89
  17    29    34  | 90    45    68    89
  17    29    34    45  | 90    68    89
  17    29    34    45    68  | 90    89
  17    29    34    45    68    89  | 90
```
[2]

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Selection sort: complexidade

Considerando como operação básica $A[j] < A[min]$:

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2}((n-1)-(i+1)+1) \\
&= \sum_{i=0}^{n-2}(n-1-i) \\
&= \sum_{i=0}^{n-2}(n-1) - \sum_{i=0}^{n-2} i \\
&= (n-1)\sum_{i=0}^{n-2} 1 - \frac{(0+(n-2))((n-2)-(0)+1)}{2} \\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} \\
&= \frac{(n-1)n}{2} \in \Theta(n^2)
\end{aligned}
$$

Selection sort: complexidade

Considerando como operação básica $A[j] < A[min]$:

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) \\
&= \sum_{i=0}^{n-2} (n-1-i) \\
&= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \\
&= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(0+(n-2))((n-2)-(0)+1)}{2} \\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} \\
&= \frac{(n-1)n}{2} \in \Theta(n^2)
\end{aligned}
$$

Contudo, a quantidade de *swaps* é $S_{worst}(n) = n - 1 \in \Theta(n)$.

- Bubble sort: $C(n) \in \Theta(n^2)$, $S_{worst}(n) \in \Theta(n^2)$.

Agenda

Decrease-and-conquer

Explorar a relação entre a solução de uma instância e a solução de uma instância menor (de forma *top-down* ou *bottom-up*).

- Decrease by a constant
  - Para calcular a^n em $\Theta(n)$ (similar à força bruta):

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{se } n > 0 \\ 1 & \text{se } n = 0 \end{cases}$$

- Decrease by a constant factor
  - Para calcular a^n em $\Theta(log\ n)$:

$$a^n = \begin{cases} (a^{n/2})^2 & \text{se } n \text{ for par e positivo} \\ (a^{(n-1)/2})^2 \cdot a & \text{se } n \text{ for ímpar} \\ 1 & \text{se } n = 0 \end{cases}$$

  - Outro exemplo: busca binária
- Variable size decrease
  - $gcd(m, n) = gcd(n, m \bmod n)$

Insertion sort

Ideia: dado um array $A[0..n-1]$, assumindo que sabemos como ordenar $A[0..n-2]$, para ordenar o array completo, basta inserir o último elemento na posição apropriada (*decrease by a constant*).

Algoritmo: InsertionSort(A[0..n - 1])

for $i \leftarrow 1$ **to** $n-1$ **do**
- $v \leftarrow A[i]$;
- $j \leftarrow i-1$;
- **while** $j \geq 0 \wedge A[j] > v$ **do**
  - $A[j+1] \leftarrow A[j]$;
  - $j \leftarrow j-1$;
- $A[j+1] \leftarrow v$;

<table>
<tr><td>89 |</td><td>45</td><td>68</td><td>90</td><td>29</td><td>34</td><td>17</td></tr>
<tr><td>45</td><td>89 |</td><td>68</td><td>90</td><td>29</td><td>34</td><td>17</td></tr>
<tr><td>45</td><td>68</td><td>89 |</td><td>90</td><td>29</td><td>34</td><td>17</td></tr>
<tr><td>45</td><td>68</td><td>89</td><td>90 |</td><td>29</td><td>34</td><td>17</td></tr>
<tr><td>29</td><td>45</td><td>68</td><td>89</td><td>90 |</td><td>34</td><td>17</td></tr>
<tr><td>29</td><td>34</td><td>45</td><td>68</td><td>89</td><td>90 |</td><td>17</td></tr>
<tr><td>17</td><td>29</td><td>34</td><td>45</td><td>68</td><td>89</td><td>90 [a]</td></tr>
</table>

[a] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Insertion sort: complexidade

Considerando como operação básica $A[j] > 0$:

$$
\begin{aligned}
C_{worst}(n) &= \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 \\
&= \sum_{i=1}^{n-1} ((i-1) - 0 + 1) \\
&= \sum_{i=1}^{n-1} i \\
&= \frac{(1+(n-1))((n-1)-1+1)}{2} \\
&= \frac{(n-1)n}{2} \in \Theta(n^2)
\end{aligned}
$$

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

$$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

Agenda

Divide-and-conquer

Ideia: divida o problema em vários problemas do mesmo tipo, resolva os subproblemas, por fim, combine as soluções dos subproblemas para obter uma solução para o problema inicial.

Exemplo: somar n números $a_0, \cdots, a_{n-1}$

- Dividir em dois sub-problemas
  - $a_0 + \cdots + a_{\lfloor n/2 \rfloor - 1}$
  - $a_{\lfloor n/2 \rfloor + \cdots + a_{n-1}}$
- Resolver os sub-problemas de forma recursiva
- Combinar (somar) as soluções

Complexidade (para $n = 2^k$): $A(n) = 2A(n/2) + 1$

Divide-and-conquer

Complexidade (em termos gerais):

- $T(n) = aT(n/b) + f(n)$, onde $a \geq 1$ e $b \geq 1$
- n = tamanho da instância do problema
- b = quantidade de sub-instâncias
- a = quantidade de sub-instâncias que precisam ser resolvidas
- $f(n)$ = tempo de divisão e composição

Teorema Mestre: se $f(n) \in \Theta(n^d)$, onde $d \geq 0$, então:

$$T(n) = \begin{cases} \Theta(n^d) & \text{se } a < b^d \\ \Theta(n^d \log n) & \text{se } a = b^d \\ \Theta(n^{\log_b a}) & \text{se } a > b^d \end{cases}$$

Versões análogas para O e Ω.

Divide-and-conquer

Considerando o exemplo da soma:

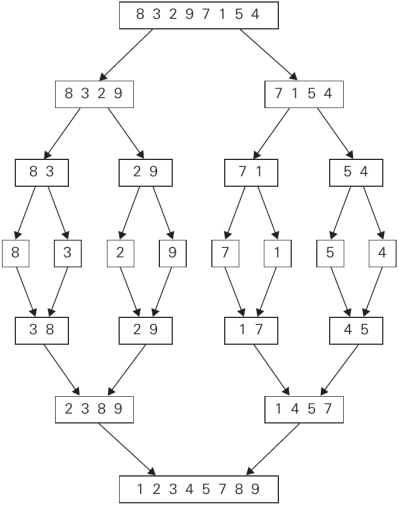
- $A(n) = 2A(n/2) + 1$, logo $a = b = 2$.
- $f(n) = 1 \in \Theta(1) = \Theta(n^d)$, logo $d = 0$.
- Como $a > b^d$ (i.e., $2 > 2^0$), $A(n) \in \Theta(n^{log_b a}) = \Theta(n^{log_2 2}) = \Theta(n)$.

Nem todo “divide-and-conquer” é melhor que força bruta!

Se $a = 1$, $T(n)$ representa algoritmos *decrease-by-a-constant factor*

Agenda

Mergesort



[3]

[3] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Mergesort

Algoritmo: Mergesort(A[0..n - 1])

if $n > 1$ **then**
  copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$;
  copy $A[\lfloor n/2 \rfloor..n - 1]$ to $C[0..\lceil n/2 \rceil - 1]$;
  $Mergesort(B[0..\lfloor n/2 \rfloor - 1])$;
  $Mergesort(C[0..\lceil n/2 \rceil - 1])$;
  $Merge(B, C, A)$;

Mergesort

Algoritmo: Merge(B[0..p - 1], C[0..q - 1], A[0..p + q - 1])

```
// Premissa: B e C estão ordenados
i ← j ← k ← 0;
while i < p ∧ j < q do
    if B[i] ≤ C[j] then
        A[k] ← B[i];
        i ← i + 1;
    else
        A[k] ← C[j];
        j ← j + 1;
    k ← k + 1;
if i = p then
    copy C[j..q − 1] to A[k..p + q − 1];
else
    copy B[i..p − 1] to A[k..p + q − 1];
```

Mergesort: complexidade

Assumindo $n = 2^k$ (operação básica = comparação chaves):

$C(n) = 2C(n/2) + C_{merge}(n)$ para $n > 1$
$C(1) = 0$

Pior caso do C_{merge} quando é preciso entrelaçar B e C:

$C_{merge}(n) = n - 1$.

Portanto, temos:

$C_{worst}(n) = 2C_{worst}(n/2) + n - 1$ para $n > 1$
$C_{worst}(1) = 0$

Mergesort: complexidade

Pelo Teorema Mestre:

- $C_{worst}(n) = 2C_{worst}(n/2) + n - 1$ para $n > 1$, logo $a = b = 2$.
- $f(n) = n - 1 \in \Theta(n) = \Theta(n^d)$, logo $d = 1$.
- Como $a = b^d$ (i.e., $2 = 2^1$), $C(n) \in \Theta(n^d log\ n) = \Theta(n\ log\ n)$.

Outros algoritmos de ordenação: quicksort e heapsort

- Vantagem do mergesort: **estabilidade**
- Desvantagem: espaço extra (i.e., eficiência espacial)

Agenda

Quicksort

Divide o array pelos seus valores (e não posição):

$$\underbrace{A[0]\ldots A[s-1]}_{\text{all are } \leq A[s]} \quad A[s] \quad \underbrace{A[s+1]\ldots A[n-1]}_{\text{all are } \geq A[s]}$$ [4]

Chamada: $\mathit{Quicksort}(A[0..n-1])$

Algoritmo: Quicksort(A[l..r])

if $l < r$ **then**
    // s é uma "split position"
    $s \leftarrow \mathit{Partition}(A[l..r])$;
    $\mathit{Quicksort}(A[l..s-1])$;
    $\mathit{Quicksort}(A[s+1..r])$

Quicksort:

- Dividir: trabalhoso
- Conquistar: imediato

Mergesort:

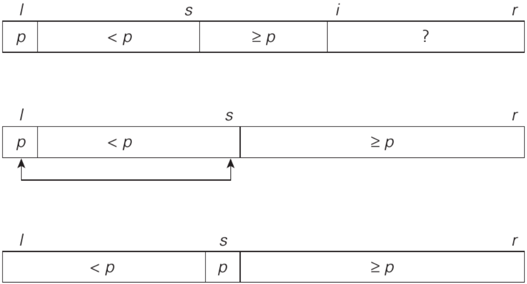
- Dividir: imediato
- Conquistar: trabalhoso

[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Quicksort: particionamento por Lomuto

Três segmentos contíguos (após o pivô): $< p \mid \geq p \mid ?\ p$.

- Iniciando de $i = l + 1$ (i.e., após o pivô), compara $A[i]$ com p. Se $A[i] \geq p$, incrementa i. Se $A[i] < p$, incrementa s, troca $A[i]$ com $A[s]$ e incrementa i. Ao terminar, troca o pivô com $A[s]$.



[5]

[5] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Quicksort: particionamento por Lomuto

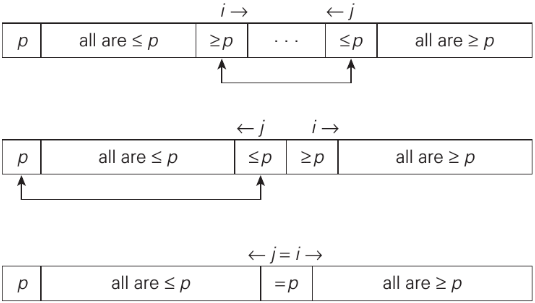
Algoritmo: LomutoPartition(A[l..r])

```
p ← A[l];
s ← l;
for i ← l + 1 to r do
    if A[i] < p then
        s ← s + 1;
        swap A[s] and A[i];
swap A[l] and A[s];
return s;                                   // nova posição do pivô
```

Quicksort: particionamento por C.A.R. Hoare

Ideia central: varrer o array a partir do início (i) e do final (j)

- Se $A[i] < p$, incrementa i. Se $A[j] > p$, decrementa j.
- Quando i e j param, há três situações possíveis.



[6]

[6] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

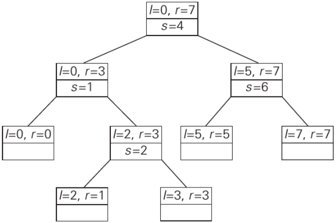
Quicksort: particionamento por C.A.R. Hoare

Algoritmo: HoarePartition(A[l..r])

$p \leftarrow A[l]$;
$i \leftarrow l$;
$j \leftarrow r + 1$;
repeat
  repeat
    $i \leftarrow i + 1$;
  until $A[i] \geq p \vee i \geq r$;
  repeat
    $j \leftarrow j - 1$;
  until $A[j] \leq p$;
  swap $A[i]$ and $A[j]$;
until $i \geq j$;
swap $A[i]$ and $A[j]$; `// desfaz último swap no caso de` $i \geq j$
swap $A[l]$ and $A[j]$;
return j;

Quicksort: particionamento por C.A.R. Hoare

0	1	2	3	4	5	6	7
5	$\overset{i}{3}$	1	9	8	2	4	$\overset{j}{7}$
5	3	1	$\overset{i}{9}$	8	2	$\overset{j}{4}$	7
5	3	1	$\overset{i}{4}$	8	2	$\overset{j}{9}$	7
5	3	1	4	$\overset{i}{8}$	$\overset{j}{2}$	9	7
5	3	1	4	$\overset{i}{2}$	$\overset{j}{8}$	9	7
5	3	1	4	$\overset{j}{2}$	$\overset{i}{8}$	9	7
2	3	1	4	**5**	8	9	7
2	$\overset{i}{3}$	1	$\overset{j}{4}$				
2	$\overset{i}{3}$	$\overset{j}{1}$	4				
2	$\overset{i}{1}$	$\overset{j}{3}$	4				
2	$\overset{j}{1}$	$\overset{i}{3}$	4				
1	**2**	3	4				
1							
		3	$\overset{ij}{4}$				
		$\overset{j}{\mathbf{3}}$	$\overset{i}{4}$				
			4				
					8	$\overset{i}{9}$	$\overset{j}{7}$
					8	$\overset{i}{7}$	$\overset{j}{9}$
					8	$\overset{j}{7}$	$\overset{i}{9}$
					7	**8**	9
					7		
							9



[7]

[7] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Quicksort: complexidade

Quantidade de operações básicas: $n + 1$ se $i > j$, n se $i = j$

Melhor caso: após particionar, pivô no meio

Assumindo $n = 2^k$:

$C_{best}(n) = 2C_{best}(n/2) + n$ para $n > 1$

$C(1) = 0$

Pelo Teorema Mestre:

- $C_{best}(n) = 2C_{best}(n/2) + n$ para $n > 1$, logo $a = b = 2$.
- $f(n) = n \in \Theta(n) = \Theta(n^d)$, logo $d = 1$.
- Como $a = b^d$ (i.e., $2 = 2^1$), $C_{best}(n) \in \Theta(n^d log\ n) = \Theta(n\ log\ n)$.

Quicksort: complexidade

Pior caso: um dos subarrays é vazio, o outro diminui -1

Assumindo $n = 2^k$:

$$C_{worst}(n) = (n+1) + n + \cdots + 3 = \frac{((n+1)+1)((n+1)-1+1)}{2} - 3$$

$$C_{worst}(n) = \frac{(n+2)(n+1)}{2} - 3 \in \Theta(n^2)$$

Caso médio (39% mais comparações que o melhor caso):

$$C_{avg}(n) \approx 2n \ln n \approx 1.39 n \log_2\, n$$

Normalmente, mais rápido que mergesort e heapsort

- Contudo, não é estável...

Algoritmos de ordenação: complexidade

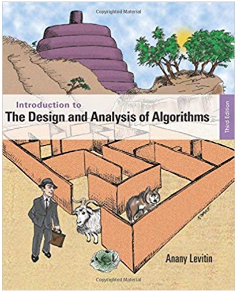
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

[8]

[8] http://bigocheatsheet.com/

Agenda

Bibliografia + leitura recomendada



Capítulo 3 (pp. 98–102)
Capítulo 4 (pp. 131–136)
Capítulo 5 (pp. 169–181)

Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

ORDENAÇÃO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil