AULA 13

Ordenação em tempo linear

CLRS 8.2–8.3

Ordenação por contagem

Recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ os elementos de $A[1\,..\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

Entra:

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

Ordenação por contagem

Recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ os elementos de $A[1\,..\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

Entra:

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

Sai:

	1	2	3	4	5	6	7	8	9	10
B	0	0	0	2	2	3	3	3	5	5

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C						

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	0	0	0	0

Ordenação por contagem

	j									
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	1	0	0	0

Ordenação por contagem

j (above position 2)

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0
B										

	0	1	2	3	4	5
C	0	0	1	0	0	1

Ordenação por contagem

$$
\begin{array}{r|c|c|c|c|c|c|c|c|c|c|}
 & & & j & & & & & & & \\
\hline
A & 2 & 5 & {\color{blue}3} & 0 & 2 & 3 & 0 & 5 & 3 & 0 \\
\hline
\end{array}
$$

$$
\begin{array}{r|c|c|c|c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
B & & & & & & & & & & \\
\hline
\end{array}
$$

$$
\begin{array}{r|c|c|c|c|c|c|}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
C & 0 & 0 & 1 & {\color{blue}1} & 0 & 1 \\
\hline
\end{array}
$$

Ordenação por contagem

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & & & & j & & & & & & \\
\cline{2-11}
A & 2 & 5 & 3 & 0 & 2 & 3 & 0 & 5 & 3 & 0 \\
\cline{2-11}
\end{array}
$$

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\cline{2-11}
B & & & & & & & & & & \\
\cline{2-11}
\end{array}
$$

$$
\begin{array}{c|c|c|c|c|c|c|}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\cline{2-7}
C & 1 & 0 & 1 & 1 & 0 & 1 \\
\cline{2-7}
\end{array}
$$

Ordenação por contagem

					j					
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	1	0	2	1	0	1

Ordenação por contagem

						j				
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	1	0	2	2	0	1

Ordenação por contagem

							j			
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	2	0	1

Ordenação por contagem

								j		
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	2	0	2

Ordenação por contagem

									j	
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	3	0	2

Ordenação por contagem

										j
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	0	2	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	0	2	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	2	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	10

Ordenação por contagem

										j
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	10

Ordenação por contagem

									j	
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B			0							

	0	1	2	3	4	5
C	2	3	5	8	8	10

Ordenação por contagem

j (acima da posição 8 de A)

A

2	5	3	0	2	3	0	5	3	0

B

1	2	3	4	5	6	7	8	9	10
		0					3		

C

0	1	2	3	4	5
2	3	5	7	8	10

Ordenação por contagem

							j			
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B			0					3		5

	0	1	2	3	4	5
C	2	3	5	7	8	9

Ordenação por contagem

						j				
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0					3		5

	0	1	2	3	4	5
C	1	3	5	7	8	9

Ordenação por contagem

						j					
A		2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0				3	3		5

	0	1	2	3	4	5
C	1	3	5	6	8	9

Ordenação por contagem

				j						
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0		2		3	3		5

	0	1	2	3	4	5
C	1	3	4	6	8	9

Ordenação por contagem

			j							
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2		3	3		5

	0	1	2	3	4	5
C	0	3	4	6	8	9

Ordenação por contagem

		j								
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2	3	3	3		5

	0	1	2	3	4	5
C	0	3	4	5	8	9

Ordenação por contagem

j (above $A[1]$)

A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2	3	3	3	5	5

	0	1	2	3	4	5
C	0	3	4	5	8	8

Ordenação por contagem

j

A									
2	5	3	0	2	3	0	5	3	0

B

1	2	3	4	5	6	7	8	9	10
0	0	0	2	2	3	3	3	5	5

C

0	1	2	3	4	5
0	3	3	5	8	8

Ordenação por contagem

COUNTING-SORT (A, B, n, k)

para $i \leftarrow 0$ **até** k **faça**
    $C[i] \leftarrow 0$

para $j \leftarrow 1$ **até** n **faça**
    $C[A[j]] \leftarrow C[A[j]] + 1$
  $\triangleright$ $C[i]$ é o número de js tais que $A[j] = i$

para $i \leftarrow 1$ **até** k **faça**
    $C[i] \leftarrow C[i] + C[i-1]$
  $\triangleright$ $C[i]$ é o número de js tais que $A[j] \leq i$

para $j \leftarrow n$ **decrescendo até** 1 **faça**
    $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Consumo de tempo

linha	consumo na linha
$1-2$	$\Theta(k)$
$3-4$	$\Theta(n)$
$5-6$	$\Theta(k)$
$7-9$	$\Theta(n)$

Consumo total: $\Theta(n+k)$

Conclusões

O consumo de tempo do COUNTING-SORT é
$\Theta(n+k)$.

- se $k \leq n$ então consumo é $\Theta(n)$
- se $k \leq 10n$ então consumo é $\Theta(n)$
- se $k = \mathrm{O}(n)$ então consumo é $\Theta(n)$
- se $k \geq n^2$ então consumo é $\Theta(k)$

A propósito: COUNTING-SORT é estável

Ordenação digital (=radix sort)

Exemplo:

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Cada $A[j]$ têm d dígitos decimais:

$A[j] = a_d\, 10^{d-1} + \cdots + a_2\, 10^1 + a_1\, 10^0$

Exemplo com $d = 3$: $\quad 3 \cdot 10^2 + 2 \cdot 10 + 9$

Ordenação digital

RADIX-SORT (A, n, d)
1 para $i \leftarrow 1$ até d faça
2 $\quad\triangleright$ 1 até d e não o contrário!
3 $\quad$ ordene $A[1\,.\,.\,n]$ pelo dígito i

Linha 3:

- faz ordenação $A[j_1\,.\,.\,j_n]$ de $A[1\,.\,.\,n]$ tal que

$$A[j_1]_i \leq \cdots \leq A[j_n]_i;$$

- ordenação deve ser estável; e
- use COUNTING-SORT.

Conclusões

- dígitos decimais: $\Theta(dn)$
- dígitos em $0\,..\,k$: $\Theta(d\,(n+k))$.

Exemplo com $d=5$ e $k=128$:

$a_5 128^4 + a_4 128^3 + a_3 128^2 + a_2 128 + a_1$

sendo $0 \leq a_i \leq 127$

Dados n números com b bits e um inteiro $r \leq b$, RADIX-SORT ordena esses números em tempo

$$\Theta(\frac{b}{r}(n+2^r)).$$

Mais experimentação

Os programas foram executados na

SunOS rebutosa 5.7 Generic_106541-04 sun4d sparc.

Os códigos foram compilados com o

gcc version 2.95.2 19991024 (release)

e opção de compilação

-Wall -ansi -pedantic -O2.

Algoritmos implementados:

`merger`	MERGE-SORT recursivo
`mergei`	MERGE-SORT iterativo
`heap`	HEAPSORT
`quick`	QUICKSORT recursivo.
`qCLR`	QUICKSORT do CLR.
`radix`	RADIX-SORT

Resultados

n	merger	mergei	heap	quick	qCLR	radix
4096	0.02	0.03	0.01	0.01	0.01	0.01
8192	0.05	0.06	0.03	0.02	0.02	0.03
16384	0.12	0.13	0.07	0.06	0.06	0.06
32768	0.24	0.27	0.15	0.11	0.11	0.11
65536	0.52	0.58	0.33	0.25	0.23	0.22
131072	1.10	1.25	0.75	0.58	0.48	0.49
262144	2.37	2.64	1.71	1.39	0.98	0.98
524288	5.10	5.57	5.07	3.91	1.93	2.06
1048576	10.86	11.77	14.07	10.12	4.32	4.44
2097152	22.71	24.45	37.48	26.61	9.05	10.61
4194304	47.63	52.23	90.99	75.76	19.19	23.98
8388608	99.90	108.73	214.78	231.30	39.91	44.70

Código

```
#define BITSWORD    32
#define BITSBYTE    8
#define K           256
#define BYTESWORD   4
#define R           (1 << BITSBYTE)
#define digit(k,d)
     (((k)>>(BITSWORD-(d)*BITSBYTE))&(R-1))
void
radix_sort(int *v, int l, int r)
{
  int i; /* digito */

  for (i = BYTEWORD-1; i >= 0; i--)
    {
      count_sort(v, l, r+1, i);
    }
}
```

Exercícios

Exercício 17.A

O seguinte algoritmo promete rearranjar o vetor $A[1\,..\,n]$ em ordem crescente supondo que cada $A[i]$ está em $\{0, \ldots, k\}$. O algoritmo está correto?

C-SORT (A, n, k)

para $i \leftarrow 0$ **até** k **faça**
  $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $C[A[j]] \leftarrow C[A[j]] + 1$
$j \leftarrow 1$
para $i \leftarrow 0$ **até** k **faça**
  enquanto $C[i] > 0$ **faça**
    $A[j] \leftarrow i$
    $j \leftarrow j + 1$
    $C[i] \leftarrow C[i] - 1$

Qual o consumo de tempo do algoritmo?

Mais exercícios

Exercício 17.B

O seguinte algoritmo promete rearranjar o vetor $A[1\,..\,n]$ em ordem crescente supondo que cada $A[j]$ está em $\{1, \ldots, k\}$. O algoritmo está correto? Estime, em notação O, o consumo de tempo do algoritmo.

VITO-SORT (A, n, k)

$i \leftarrow 1$
para $a \leftarrow 1$ **até** $k - 1$ **faça**
  para $j \leftarrow i$ **até** n **faça**
    se $A[j] = a$
      então $A[j] \leftrightarrow A[i]$ $\quad \triangleright$ troca
        $i \leftarrow i + 1$

Exercício 17.C

Suponha que os components do vetor $A[1\,..\,n]$ estão todos em $\{0, 1\}$. Prove que $n - 1$ comparações são suficientes para rearranjar o vetor em ordem crescente.

Exercício 17.D

Qual a principal invariante do algoritmo RADIX-SORT?

i-ésimo menor elemento

CLRS 9

i-ésimo menor

Problema: Encontrar o i-ésimo menor elemento de $A[1..n]$

Suponha $A[1..n]$ sem elementos repetidos.

Exemplo: 33 é o 4o. menor elemento de:

1									10	
22	99	32	88	34	**33**	11	97	55	66	A

1			4						10	
11	22	32	**33**	34	55	66	88	97	99	ordenado

Mediana

Mediana é o $\lfloor \frac{n+1}{2} \rfloor$-ésimo menor ou o $\lceil \frac{n+1}{2} \rceil$-ésimo menor elemento

Exemplo: a mediana é 34 ou 55:

1									10	
22	99	32	88	**34**	33	11	97	**55**	66	A

1				5	6				10	
11	22	32	33	**34**	**55**	66	88	97	99	ordenado

Menor

Recebe um vetor $A[1\,..\,n]$ e devolve o valor do menor elemento.

MENOR (A, n)
$\text{menor} \leftarrow A[1]$
para $k \leftarrow 2$ **até** n **faça**
    se $A[k] < \text{menor}$
        então $\text{menor} \leftarrow A[k]$
devolva menor

O consumo de tempo do algoritmo MENOR é $\Theta(n)$.

Segundo menor

Recebe um vetor $A[1\,..\,n]$ e devolve o valor do segundo menor elemento, supondo $n \geq 2$.

SEG-MENOR (A, n)

$\mathrm{menor} \leftarrow \min\{A[1], A[2]\}$ $\quad$ $\mathrm{segmenor} \leftarrow \max\{A[1], A[2]\}$
para $k \leftarrow 3$ **até** n **faça**
  se $A[k] < \mathrm{menor}$
    então $\mathrm{segmenor} \leftarrow \mathrm{menor}$
      $\mathrm{menor} \leftarrow A[k]$
  senão se $A[k] < \mathrm{segmenor}$
    então $\mathrm{segmenor} \leftarrow A[k]$
devolva $\mathrm{segmenor}$

O consumo de tempo do algoritmo SEG-MENOR é $\Theta(n)$.

i-ésimo menor

Recebe $A[1\,..\,n]$ e i tal que $1 \leq i \leq n$
e devolve valor do i-ésimo menor elemento de $A[1\,..\,n]$

SELECT-ORD (A, n, i)
1 ORDENE (A, n)
2 **devolva** $A[i]$

O consumo de tempo do algoritmo SELECT-ORD é $\Theta(n \lg n)$.

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \leq q \leq r$ e
$A[p\,..\,q{-}1] \leq A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)

$x \leftarrow A[r]$ $\quad \triangleright$ x é o “pivô”
$i \leftarrow p{-}1$
para $j \leftarrow p$ **até** $r - 1$ **faça**
 se $A[j] \leq x$
  então $i \leftarrow i + 1$
   $A[i] \leftrightarrow A[j]$
$A[i{+}1] \leftrightarrow A[r]$
devolva $i + 1$

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

```
PARTICIONE (A, p, r)
    x ← A[r]          ▷ x é o "pivô"
    i ← p−1
    para j ← p até r − 1 faça
          se A[j] ≤ x
                então i ← i + 1
                      A[i] ↔ A[j]
    A[i+1] ↔ A[r]
    devolva i + 1
```

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \leq q \leq r$ e
$A[p\,..\,q{-}1] \leq A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)

$x \leftarrow A[r]$ $\qquad \triangleright$ x é o "pivô"
$i \leftarrow p{-}1$
para $j \leftarrow p$ **até** $r-1$ **faça**
    se $A[j] \leq x$
        então $i \leftarrow i+1$
            $A[i] \leftrightarrow A[j]$
$A[i{+}1] \leftrightarrow A[r]$
devolva $i+1$

O algoritmo PARTICIONE consome tempo $\Theta(n)$.

Algoritmo SELECT

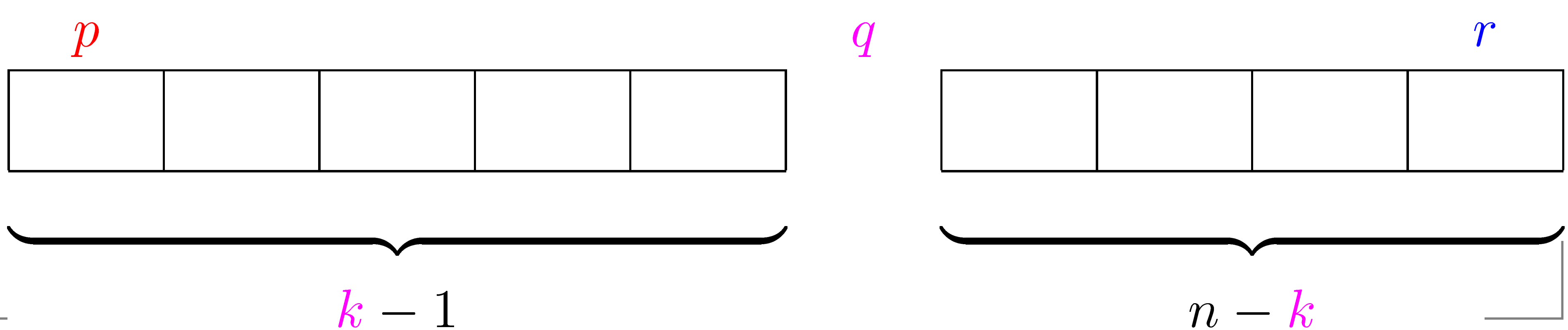
Recebe $A[p\,..\,r]$ e i tal que $1 \leq i \leq r{-}p{+}1$
e devolve valor do i-ésimo menor elemento de $A[p\,..\,r]$

```
SELECT(A, p, r, i)
    se p = r
        então devolva A[p]
    q ← PARTICIONE (p, r)
    k ← q − p + 1
    se k = i
        então devolva A[q]
    se k > i
        então devolva SELECT (A, p, q − 1, i)
        senão devolva SELECT (A, q + 1, r, i − k)
```

Algoritmo SELECT

SELECT(A, p, r, i)

se $p = r$
    então devolva $A[p]$
$q \leftarrow$ PARTICIONE (A, p, r)
$k \leftarrow q - p + 1$
se $k = i$
    então devolva $A[q]$
se $k > i$
    então devolva SELECT $(A, p, q - 1, i)$
    senão devolva SELECT $(A, q + 1, r, i - k)$



Consumo de tempo

$T(n)$ = consumo de tempo **máximo** quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 2\,\Theta(1)$
3	$= \Theta(n)$
4-7	$= 4\,\Theta(1)$
8	$= T(k-1)$
9	$= T(n-k)$

T(n) $= \Theta(n+6) + \max\{T(k-1), T(n-k)\}$

$= \Theta(n) + \max\{T(k-1), T(n-k)\}$

Consumo de tempo

$T(n)$ pertence a mesma classe Θ que:

$$T'(1) = 1$$
$$T'(n) = T'(n-1) + n \quad \text{para } n = 2, 3, \ldots$$

Solução assintótica: $T'(n)$ é $\Theta(n^2)$

Solução exata:

$$T'(n) = \frac{n^2}{2} + \frac{n}{2}.$$

Algumas conclusões

No melhor caso o consumo de tempo do algoritmo SELECT é $\Theta(n)$.

No pior caso o consumo de tempo do algoritmo SELECT é $\Theta(n^2)$.

Consumo médio?
$\mathrm{E}[T(n)] =$???

Exemplos

Número médio de comparações sobre todas as permutações de $A[p \mathinner{..} r]$ (supondo que nas linhas 8 e 9 o algoritmo sempre escolhe o lado maior):

$A[p \mathinner{..} r]$	comps
1,2	1+0
2,1	1+0
média	2/2

$A[p \mathinner{..} r]$	comps
1,2,3	2+1
2,1,3	2+1
1,3,2	2+0
3,1,2	2+0
2,3,1	2+1
3,2,1	2+1
média	16/6

Mais exemplos

$A[p\,..\,r]$	comps	$A[p\,..\,r]$	comps
1,2,3,4	3+3	1,3,4,2	3+1
2,1,3,4	3+3	3,1,4,2	3+1
1,3,2,4	3+2	1,4,3,2	3+1
3,1,2,4	3+2	4,1,3,2	3+1
2,3,1,4	3+3	3,4,1,2	3+1
3,2,1,4	3+3	4,3,1,2	3+1
1,2,4,3	3+1	2,3,4,1	3+3
2,1,4,3	3+1	3,2,4,1	3+3
1,4,2,3	3+1	2,4,3,1	3+2
4,1,2,3	3+1	4,2,3,1	3+2
2,4,1,3	3+1	3,4,2,1	3+3
4,2,1,3	3+1	4,3,2,1	3+3

média 116/24

Ainda exemplos

No caso $r - p + 1 = 5$, a média é $864/120$.

n	$\mathrm{E}[T(n)]$	$\cong$
1	0	0
2	$2/2$	1
3	$16/6$	2.7
4	$116/24$	4.8
5	$864/120$	7.2

Número de comparações

O consumo de tempo assintótico é proporcional a

$C(n) =$ número de comparações entre elementos de A quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 0$
3	$= n - 1$
4-7	$= 0$
8	$= C(k-1)$
9	$= C(n-k)$

total $\leq \max\{C(k-1), C(n-k)\} + n - 1$

Número de comparações

No pior caso $C(n)$ pertence a mesma classe Θ que:

$$C'(1) = 0$$
$$C'(n) = C'(n-1) + n - 1 \quad \text{para } n = 3, 4, \ldots$$

Solução assintótica: $C'(n)$ é $\Theta(n^2)$

Solução exata:

$$C'(n) = \frac{n^2}{2} - \frac{n}{2}.$$

Particione aleatorizado

Rearranja $A[p \mathinner{.\,.} r]$ de modo que $p \leq q \leq r$ e $A[p \mathinner{.\,.} q{-}1] \leq A[q] < A[q{+}1 \mathinner{.\,.} r]$

```
PARTICIONE-ALEA(A, p, r)
    i ← RANDOM(p, r)
    A[i] ↔ A[r]
    devolva PARTICIONE(A, p, r)
```

O algoritmo PARTICIONE-ALEA consome tempo $\Theta(n)$.

SELECT-ALEATORIZADO (= randomized select)

Recebe $A[p\,..\,r]$ e i tal que $1 \leq i \leq r-p+1$
e devolve valor do i-ésimo menor elemento de $A[p\,..\,r]$

SELECT-ALEA(A, p, r, i)
    se $p = r$
        então devolva $A[p]$
    $q \leftarrow$ PARTICIONE-ALEA (A, p, r)
    $k \leftarrow q - p + 1$
    se $k = i$
        então devolva $A[q]$
    se $k > i$
        então devolva SELECT-ALEA $(A, p, q-1, i)$
        senão devolva SELECT-ALEA $(A, q+1, r, i-k)$

Cosumo de tempo

O consumo de tempo é proporcional a

$T(n) =$ número de comparações entre elementos de A quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 0$
3	$= n - 1$
4-7	$= 0$
8	$= T(k-1)$
9	$= T(n-k)$
total	$\leq \max\{T(k-1), T(n-k)\} + n - 1$

$T(n)$ é uma variável aleatória.

Consumo de tempo

$$T(1) = 0$$

$$T(n) \leq \sum_{h=1}^{n-1} X_h T(h) + n - 1 \quad \text{para } n = 2, 3, \ldots$$

onde

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

$\Pr\{X_h = 1\} = \mathrm{E}[X_h]$

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_h valer 1?

$\Pr\{X_h = 1\} = \mathrm{E}[X_h]$

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_h valer 1?

Para $h = 1, \ldots, \lfloor n/2 \rfloor - 1$, $\Pr\{X_h = 1\} = 0 = \mathrm{E}[X_h]$.

Para $h = \lceil n/2 \rceil, \ldots, n$,

$$\text{Pr}\,\{X_h{=}1\} \;=\; \frac{1}{n} + \frac{1}{n} \;=\; \frac{2}{n} \;=\; \mathrm{E}[X_h]$$

Se n é impar e $h = \lfloor n/2 \rfloor$, então

$$\Pr\{X_h = 1\} = \frac{1}{n} = \mathrm{E}[X_h]$$

Consumo de tempo esperado

$$\mathrm{E}[T(1)] = 0$$

$$\begin{aligned}
\mathrm{E}[T(n)] &\leq \sum_{h=1}^{n-1} \mathrm{E}[X_h T(h)] + n - 1 \\
&\leq \sum_{h=1}^{n-1} \mathrm{E}[X_h]\,\mathrm{E}[T(h)] + n - 1 \quad (\text{CLRS 9.2-2}) \\
&\leq \frac{2}{n}\sum_{h=a}^{n-1} \mathrm{E}[T(h)] + n - 1 \quad \text{para } n = 2, 3, \ldots
\end{aligned}$$

onde $a = \lfloor n/2 \rfloor$.

Solução: $\mathrm{E}[T(n)] = \mathrm{O}(n)$.

Consumo de tempo esperado

$\mathrm{E}[T(n)]$ pertence a mesma classe O que:

$$S(1) = 0$$

$$S(n) \le \frac{2}{n}\sum_{h=a}^{n-1} S(h) + n - 1 \quad \text{para } n = 2, 3, \ldots$$

onde $a = \lfloor n/2 \rfloor$.

n	1	2	3	4	5	6	7	8	9	10
$S(n)$	0.0	1.0	2.7	4.8	7.4	10.0	13.1	15.8	19.4	22.1
$4n$	4	8	12	16	20	24	28	32	36	40

Vamos verificar que $S(n) < 4\,n$ para $n = 1, 2, 3, 4, \ldots$

Recorrência

Prova: Se $n = 1$, então $S(n) = 0 < 4 = 4 \cdot 1 = 4n$. Se $n \geq 2$,

$$
\begin{aligned}
S(n) &\leq \frac{2}{n} \sum_{h=a}^{n-1} S(h) + n - 1 \\
&\overset{\text{hi}}{<} \frac{2}{n} \sum_{h=a}^{n-1} 4\,h + n - 1 \\
&= \frac{8}{n} \Big(\sum_{h=1}^{n-1} h - \sum_{h=1}^{a-1} h \Big) + n - 1 \\
&\leq \frac{4}{n} \Big(n^2 - n - \frac{(n-1)(n-3)}{4} \Big) + n - 1 \\
&= \frac{4}{n} \Big(\frac{3n^2}{4} - \frac{3}{4} \Big) + n - 1 \\
&= 3n - \frac{3}{n} + n - 1 = 4n - \frac{3}{n} - 1 < 4n.
\end{aligned}
$$

Conclusão

O consumo de tempo esperado do algoritmo SELECT-ALEA é $\mathrm{O}(n)$.

Exercícios

Exercício 18.A [CLRS 9.1-1] [muito bom!]
Mostre que o segundo menor elemento de um vetor $A[1\,..\,n]$ pode ser encontrado com não mais que $n + \lceil \lg n \rceil - 2$ comparações.

Exercício 18.B
Prove que o algoritmo Select Aleatorizado (= Randomized Select) funciona corretamente.

Exercício 18.C [CLRS 9.2-3]
Escreva uma versão iterativa do algoritmo Select Aleatorizado (= Randomized Select).