# MAC0338 Análise de Algoritmos

*DCC–IME–USP, 18 de maio de 2007*

## Instruções

(*i*) Esta prova contém quatro questões, cada uma vale dois pontos e meio.

(*ii*) **Enuncie** os teoremas e propriedades usados para justificar suas afirmações.

(*iii*) Você **pode** utilizar como subrotina qualquer algoritmo visto em sala de aula sem reescrevê-lo. No entanto, você **deve** descrever clara e sucintamente o que o algoritmo recebe, devolve ou faz e o seu consumo de tempo. Exemplo

> *"O algoritmo* Blá-Blá-Blá *usa como subrotina o algoritmo* Ordenação-Lerda $(A, n)$ *que recebe e rearranja um vetor* $A[1\,..\,n]$ *de modo que ele fique em ordem crescente. O consumo de tempo do algoritmo* Ordenação-Lerda *é* $O(n^n)$*."*

(*iv*) Não é permitida a consulta a livros, anotações, colegas, calculadoras, Internet, computadores ...

### Solução da prova

**Questão 1** [CLRS 8.3-4]
Descreva um algoritmo ORDENE$(A, n)$ que recebe e ordena um vetor $A[1\,..\,n]$ em que todos os elementos pertencem a $\{0, 1, \ldots, n^2 - 1\}$. O consumo de tempo do algoritmo deve ser $O(n)$, onde $n = p - r + 1$. Justifique a sua resposta.

A existência desse algoritmo linear para ordenação contraria em algo o limite inferior para o consumo de tempo de algoritmos de ordenação? Enuncie a asserção desse "limite inferior para ordenação" para justificar a sua resposta.

Estamos habituados a representar números naturais na base decimal que utiliza os algarismos $0, 1, \ldots, 9$. Na solução consideraremos a representação de cada elemento em $A[1\,..\,n]$ na base $n$, que utiliza os algarismos $0, 1, \ldots, n - 1$. Desta forma, para $j = 1, \ldots, n$, temos que

$$A[j] = a_1 \times n + a_0,$$

onde $a_0$ e $a_1$ são algarismos em $\{0, 1, \ldots, n - 1\}$. Diremos que $a_0$ é o algarismo **menos significativo** de $A[j]$ e que $a_1$ é o algarismo **mais significativo** de $A[j]$.

O algoritmo COUNTING-SORT recebe e ordena um vetor $X[1\,..\,n]$ em que todos os elementos estão em $\{0, 1, \ldots, k\}$, para um dado $k$. O consumo de tempo do algoritmo é $\Theta(n + k)$. Em particular, se $k$ é $\Theta(n)$, então o consumo de tempo do COUNTING-SORT é $\Theta(n)$.

O algoritmo ORDENE $(A, n)$ é uma mera adaptação do RADIX-SORT.

ORDENE $(A, n)$
1 ordene $A[1\,..\,n]$ usando como chaves seus algarismos **menos significativos**.
2 ordene $A[1\,..\,n]$ usando como chaves seus algarismos **mais significativos**.

Utilizando uma adaptação do COUNTING-SORT para $k = n - 1$ nas ordenações das linhas 1 e 2 obtemos um algoritmo de consumo de tempo $\Theta(n + n) = \Theta(n)$. A estabilidade do COUNTING-SORT é fundamental para a correção do algoritmo ORDENE.

Sabe-se que todo algoritmo de ordenação **baseado em comparações** faz

$$\Omega(n \lg n)$$

comparações no **pior caso**. O limite inferior da ordenação, em termos da notação-o, se traduz em

> **Não existe** algoritmo de ordenação **baseado em comparações** de consumo de tempo $o(n \lg n)$.

A existência do algoritmo linear ORDENE$(A, n)$ não contraria em nada o limite inferior para ordenação já que esse algoritmo não é baseado em comparações, ou seja, para determinar a ordem relativa entre os elemento do vetor $A[1\,..\,n]$ o algoritmo não utiliza apenas testes como

$$A[i] < A[j], A[i] \leq A[j], A[i] = A[j], A[i] \geq A[j] \text{ e } A[i] > A[j].$$

O algoritmo ORDENE examina o valor de cada elemento e também utiliza operações de "mascara" para extrair um determinado dígito de cada elemento de $A$.

**Questão 2** [CLRS 9.3-5]
Suponha que MEDIANA$(A, p, r)$ é um algoritmo que rearranja os elementos de um dado vetor $A[p\,..\,r]$ de números inteiros e devolve um índice $q$, $p \le q \le r$, tal que

$$A[p\,..\,q-1] \le A[q] \le A[q+1\,..\,r]$$

e $A[q]$ é a mediana de $A[p\,..\,r]$. Suponha ainda que o consumo de tempo do algoritmo MEDIANA é linear.

Escreva um algoritmo SELECT$(A, p, r, i)$ que recebe um vetor $A[p\,..\,r]$ de números inteiros e um número inteiro $i$, $1 \le i \le r - p + 1$, e devolve o valor do $i$-ésimo menor elemento de $A[p\,..\,r]$. O algoritmo deve utilizar o algoritmo MEDIANA como subrotina e deve consumir tempo linear. Explique sucintamente porque seu algoritmo está correto e tem o consumo de tempo pedido.

**Solução:** Eis uma versão de algoritmo SELECT que faz o serviço:

```
SELECT (A, p, r, i)
1  se p = r
2      então devolva A[p]
3  q ← MEDIANA (A, p, r)
4  k ← q − p + 1
5  se k = i
6      então devolva A[q]
7  se k > i
8      então devolva SELECT (A, p, q − 1, i)
9      senão devolva SELECT (A, q + 1, r, i − k)
```

Como $1 \le i \le r - p + 1$, se $p = q$, então $i = 1$ e o algoritmo devolve na linha 2 o único elemento do vetor $A[p\,..\,q]$. O algoritmo determina na linha 4 o número de elementos no vetor $A[p\,..\,q]$. Devido a especificação do algoritmo MEDIANA vemos que:

- se $k = i$, então o $i$-ésimo menor elemento do vetor $A[p\,..\,q]$ é $A[q]$;
- se $k > i$, então o elemento desejado é o $i$-ésimo menor elemento do vetor $A[p\,..\,q-1]$; e
- se $k < i$, então o elemento procurado é o $(i-k)$-ésimo menor elemento do vetor $A[q+1\,..\,r]$.

A correção do algoritmo SELECT é devida a este ser uma mera implementação dos fatos acima.

Seja $T(n)$ o consumo de tempo máximo quando $n = r - p + 1$. O consumo de tempo pela execução de cada linha do algoritmo está na tabela a seguir.

| linha | consumo de tempo da linha |
|---|---|
| 1-2 | $= 2\,\Theta(1)$ |
| 3 | $= \Theta(n)$ |
| 4-7 | $= 4\,\Theta(1)$ |
| 8-9 | $= T(\lfloor n/2 \rfloor)$ |

$$\begin{aligned} T(n) &= \Theta(n+6) + T(\lfloor n/2 \rfloor) \\ &= \Theta(n) + T(\lfloor n/2 \rfloor) \end{aligned}$$

Obtemos desta forma a recorrência que descreve o consumo de tempo do algoritmo SELECT no pior caso:

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= \Theta(n) + T(\lfloor n/2 \rfloor) \quad \text{para } n = 2, 3, \ldots \end{aligned}$$

Seja $c$ uma constante tal que o consumo de tempo $\Theta(n)$ correspondente às linhas 1–7 é não superior a $c\,n$ para $n = 1, 2, \ldots$ Desta forma, para $n \geq 2$,

$$\begin{aligned} T(n) &= \Theta(n) + T\big(\left\lfloor \frac{n}{2} \right\rfloor\big) \\ &\leq c\,n + T\big(\left\lfloor \frac{n}{2} \right\rfloor\big) \\ &\leq c\,n + c\left\lfloor \frac{n}{2} \right\rfloor + T\big(\left\lfloor \frac{n}{2^2} \right\rfloor\big) \\ &\leq c\,n + c\left\lfloor \frac{n}{2} \right\rfloor + c\left\lfloor \frac{n}{2^2} \right\rfloor + T\big(\left\lfloor \frac{n}{2^3} \right\rfloor\big) \\ &\leq \ldots \\ &\leq c\,n + c\left\lfloor \frac{n}{2} \right\rfloor + c\left\lfloor \frac{n}{2^2} \right\rfloor + \cdots + c\left\lfloor \frac{n}{2^k} \right\rfloor \\ &\leq c\,n + c\,\frac{n}{2} + c\,\frac{n}{2^2} + \cdots + c\,\frac{n}{2^k} \\ &= c\,n\,\big(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k}\big) \\ &< c\,n\,\big(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k} + \cdots\big) \\ &= 2c\,n \end{aligned}$$

onde $k = \lfloor \lg n \rfloor$. Como $T(n) \leq 2c\,n$ para $n = 2, 3, \ldots$ concluímos que $T(n)$ é $\mathrm{O}(n)$ (e portanto $\Theta(n)$).

**Questão 3** [CLRS 5.3-4]
Considere o algoritmo a seguir que recebe um vetor $A[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ uma permutação dos elementos de $A$.

```
PERMUTE-BY-CYCLIC(A, B, n)
1  k ← RANDOM(1, n)
2  para i ← 1 até n faça
3       j ← i + k
4       se j > n
5            então j ← j − n
6       B[j] ← A[i]
```

Suponha que o vetor $A[1\,..\,n]$ contém uma permutação dos inteiros $1, 2, \ldots, n$. Se $i$ e $j$ são números inteiros em $\{1, \ldots, n\}$, então quanto vale $\Pr\{B[j] = A[i]\}$? O professor McSperto afirma que o vetor $B[1\,..\,n]$ produzido pelo algoritmo é uma permutação aleatória uniforme de $A[1\,..\,n]$. O professor tem razão? Justifique as suas respostas.

**Solução:** Se $j > i$, então

$$\begin{aligned} \Pr\{B[j] = A[i]\} &= \Pr\{j = i + k\} && (1)\\ &= \Pr\{k = j - i\} \\ &= 1/n\ , && (2) \end{aligned}$$

onde a igualdade (1) segue da linha 3 e do fato de $A[1\,..\,n]$ conter elementos distintos e a igualdade (2) é devida à especificação do algoritmo RANDOM. Analogamente, se $j \leq i$

$$\begin{aligned} \Pr\{B[j] = A[i]\} &= \Pr\{j = i + k - n\} && (3)\\ &= \Pr\{k = j - i + n\} \\ &= 1/n\ , \end{aligned}$$

onde a igualdade (3) é conseqüência das linhas 3, 4 e 5 e do fato de $A[1\,..\,n]$ conter elementos distintos.

O algoritmo não produz uma permutação aleatória uniforme para $n > 2$. Em virtude da linha 1 vê-se que o algoritmo é capaz de produzir $n$ permutação de $A[1\,..\,n]$ enquanto o número de permutações é $n!$ ($n! > n$ para $n > 2$).

**Questão 4** [CLRS 15.4-5]
Uma subseqüência $Z[1\,..\,k]$ de um vetor de números inteiros $A[1\,..\,n]$ é **crescente** se

$$Z[1] \leq \cdots \leq Z[k].$$

Uma subseqüência crescente de $A[1\,..\,n]$ é **máxima** se não existe outra subseqüência crescente mais longa. Por exemplo, $\langle 5, 6, 9\rangle$ é uma subseqüência crescente de $\langle 9, \mathbf{5}, 2, \mathbf{6}, 3, 5, \mathbf{9}, 6\rangle$ e $\langle 2, 3, 5, 6\rangle$ é uma subseqüência crescente máxima de $\langle 9, 5, \mathbf{2}, 6, \mathbf{3}, \mathbf{5}, 9, \mathbf{6}\rangle$.

Considere o problema de encontrar uma subseqüência crescente máxima de um vetor $A[1\,..\,n]$ dado.

a) Qual a subestrutura ótima para este problema?

b) Escreva um algoritmo Compr-Subseq-Máxima$(A, n)$ que recebe um vetor $A[1\,..\,n]$ e devolve o **comprimento** de uma subseqüência crescente máxima. O consumo de tempo do algoritmo deve ser $O(n^2)$. Justifique a correção e o consumo de tempo do algoritmo.

**Solução** de a). Suponha que $Z[1\,..\,k]$ é uma subseqüência crescente máxima de de $A[1\,..\,n]$.

Se $A[n] = Z[k]$, então $Z[1\,..\,k-1]$ é uma subseqüência crescente máxima de $A[1\,..\,n-1]$, senão $Z[1\,..\,k]$ é uma subseqüência crescente máxima de $A[1\,..\,n-1]$.

**Solução** de b). Seja $t[i]$ o comprimento de uma subseqüêcia crescente máxima de $A[1\,..\,i]$ que tem $A[i]$ como último elemento. Temos que,

$$\begin{aligned} t[0] &= 0 \\ t[i] &= 1 + \max\{t[k] : 0 \leq k \leq i-1 \ \text{ e } \ A[k] \leq A[i]\} \end{aligned}$$

para $i = 1, 2, \ldots, n$. Suponha aqui que $A[0] = -\infty$.

O algoritmo a seguir baseia-se na recorrência acima.

```
Compr-Subseq-Máxima(A, n)
0  compr ← 0
1  para i ← 1 até n faça
2       t[i] ← 1
3       para k ← 1 até i − 1 faça
4            se A[k] ≤ A[i] e t[k] ≥ t[i]
5                 então t[i] ← 1 + t[k]
6       se t[i] > compr
7            então compr ← t[i]
8  devolva compr
```

**Consumo de tempo**

| linha | consumo de **todas** as execuções da linha |
|---|---|
| 0 | $\Theta(1)$ |
| 1 | $\Theta(n)$ |
| 2 | $\Theta(n)$ |
| 3 | $\Theta(1 + 2 + \cdots + n) = \Theta(n^2)$ |
| 4 | $\Theta(1 + 2 + \cdots + (n-1)) = \Theta(n^2)$ |
| 5 | $\mathrm{O}(1 + 2 + \cdots + (n-1)) = \mathrm{O}(n^2)$ |
| 6 | $\Theta(n)$ |
| 7 | $\mathrm{O}(n)$ |
| 8 | $\Theta(1)$ |
| **total** | $2\Theta(n^2) + \mathrm{O}(n^2) + 3\Theta(n) + \mathrm{O}(n) + 2\Theta(1) = \Theta(n^2)$ |

Não faz parte da questão, mas é legal ...

Abaixo está um algoritmo SUBSEQ-MÁXIMA$(A, n, t, compr)$ que recebe um vetor $A[1\,..\,n]$ e o vetor $t[0\,..\,n]$ e *compr* calculados pelo algoritmo COMPR-SUBSEQ-MÁXIMA$(A, n)$ do item anterior e devolve uma **subseqüência** crescente máxima $Z[1\,..\,k]$ de $A[1\,..\,n]$. O consumo de tempo do trecho da modificação ou do algoritmo deve ser $O(n)$.

```
SUBSEQ-MÁXIMA (A, n, t, compr)
1  k ← compr
2  Z[k + 1] ← ∞
3  i ← n
4  enquanto k > 0 faça    ▷ aqui vale que i > 0
5      se t[i] = k e A[i] ≤ Z[k + 1]
6          então Z[k] ← A[i]
7                k ← k − 1
8      i ← i − 1
9  devolva Z[1 .. compr]
```

**Consumo de tempo**

| linha | consumo de **todas** as execuções da linha |
|---|---|
| 1 | $\Theta(1)$ |
| 2 | $\Theta(1)$ |
| 3 | $\Theta(1)$ |
| 4 | $O(n)$ |
| 5 | $O(n)$ |
| 6 | $\Theta(k)$ |
| 7 | $\Theta(k)$ |
| 8 | $O(n)$ |
| 9 | $\Theta(k)$ |
| **total** | $O(3n) + \Theta(3k + 3) = O(n) + \Theta(k) = O(n + k) = O(n)$ |

É possível fazermos uma versão do algoritmo acima que consome tempo $\Theta(k)$. Para isto é necessário que, além do vetor $t[1\,..\,n]$, o algoritmo COMPR-SUBSEQ-MÁXIMA construa um vetor $b[1\,..\,n]$ em que $b[i] = k$ se $A[i] < A[k]$ e $t[i] = t[k] + 1$ para $i = 1, \ldots, n$. Suponha aqui que $A[0] = -\infty$.