Análise de Algoritmos

Primeira Lista de Exercícios

Fundamentos

1.

$f_{\mathcal{A}}(n)$	1 seg.	1 min.	1 hora	1 dia	1 mês	1 ano	1 século
$\ln(n)$	e^{10^6}	$e^{6\,10^7}$	$e^{3.6\,10^9}$	$e^{8.64\,10^{10}}$	$e^{2.592\,10^{12}}$	$e^{3.1104\,10^{13}}$	$e^{3.1104\,10^{15}}$
$\sqrt{n}$	10^{12}	$3.6\,10^{15}$	$3.6^2\,10^{19}$	$8.64^2\,10^{20}$	$2.592^2\,10^{24}$	$3.1104^2\,10^{26}$	$3.1104^2\,10^{30}$
$n\ln(n)$	87847	$3.9501\,10^6$	$1.8890\,10^8$	$3.9117\,10^9$	$1.0217\,10^{11}$	$1.1210\,10^{12}$	$9.6591\,10^{13}$
n^3	10^2	$60^{1/3}10^2$	$3.6^{1/3}10^3$	$86.4^{1/3}10^3$	$2.592^{1/3}10^4$	$31.104^{1/3}10^4$	$3.1104^{1/3}10^5$

$f_{\mathcal{A}}(n)$	1 seg.	1 min.	1 hora	1 dia	1 mês	1 ano	1 século
n^2	$10^{9/2}$	$\sqrt{6}\,10^5$	$\sqrt{3.6}\,10^6$	$\sqrt{86.4}\,10^6$	$\sqrt{25.92}\,10^7$	$\sqrt{3.1104}\,10^8$	$\sqrt{3.1104}\,10^9$
n^3	10^3	$60^{1/3}10^3$	$3.6^{1/3}10^4$	$86.4^{1/3}10^4$	$2.592^{1/3}10^5$	$31.104^{1/3}10^5$	$3.1104^{1/3}10^6$
2^n	30	36	42	46	51	55	61
$n!$	12	13	15	16	17	18	20

2. (a) Para $a = 3$, $b = 2$ e $f(n) = n^2$ tem-se que $n^2 = f(n) \in \Omega(n^{\log_2 3+\epsilon})$, onde $\epsilon \approx 0.4$. Observe que $3f(n/2) = 3n^2/4 \leq cn^2$, onde $c = 7/8$. Portanto, pelo item 3 do Teorema Master, $T(n) \in \Theta(n^2)$.

   (b) Observe que em cada chamada recursiva temos um custo adicional de 1. Analisando a árvore recursiva de $T(n)$ nota-se uma árvore binária de altura $n-1$ onde em cada nível i existem 2^i nós, cada um rotulado com o valor 1. Portanto,

$$T(n) = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \in \Theta(2^n)$$

   Obs: a prova por indução de que $T(n) \in \Theta(2^n)$ não é necessária neste caso porque a solução do somatório representando $T(n)$ foi feita diretamente. Ela pode ser feita mostrando que $c_1 2^n \leq T(n) \leq c_2 2^n - 1$, para constantes c_1 e c_2.

   (c) Para $a = 2$, $b = 2$ e $f(n) = n$ tem-se que $n \in \Theta(n)$ (reflexividade). Portanto, pelo item 2 do TM, $T(n) \in \Theta(n \lg(n))$.

   (d) Para $a = 3$, $b = 2$ e $f(n) = n\ln(n)$ tem-se que $n^{\lg 3} = n^{1.5+\epsilon_0}$, onde $0 < \epsilon_0 < 0.085$. Observe que $n^{1.5} = n^{1/2}n$ e que $\lim_{n\to\infty} \frac{n^{1/2}}{\ln(n)} = \infty$. Logo, $f(n) \in O(n^{\lg 3 - \epsilon_0})$. Portanto, pelo item 1 do TM, $T(n) \in \Theta(n^{\lg 3})$.

   (e) A adição de 5 não deve ser relevante para n suficientemente grande. Pode-se então usar a árvore recursiva para a resolução de $T'(n) = 3T'(n/3) + n/2$ e provar por indução se a solucão é um limite apropriado para $T(n)$. Logo

$$\begin{aligned} T'(n) &= \sum_{i=0}^{\log_3(n)-1} 3^i \frac{n}{3^i 2} + 3^{\log_3(n)}\Theta(1) \\ &= \sum_{i=0}^{\log_3(n)-1} \frac{n}{2} + n\Theta(1) \\ &= \frac{1}{2} n \log_3(n) + \Theta(n) \end{aligned}$$

Por indução tem-se que

$$\begin{aligned}
T(n) &= 3T\Big(\frac{n}{3}+5\Big)+\frac{n}{2} \\
&\geq 3c_1\Big(\frac{n}{3}+5\Big)\log_3\Big(\frac{n}{3}+5\Big)+\frac{n}{2} \quad (HI) \\
&> 3c_1\frac{n}{3}\log_3\Big(\frac{n}{3}\Big)+\frac{n}{2} \quad (monot.) \\
&= c_1 n(\log_3(n)-1)+\frac{n}{2} \\
&= c_1 n\log_3(n)+n\Big(\frac{1}{2}-c_1\Big) \\
&\geq c_1 n\log_3(n), \quad \text{para } c_1 \leq \frac{1}{2}
\end{aligned}$$

Portanto, $T(n) \in \Omega(n\log_3(n))$.

(f) Analisando a árvore recursiva de $T(n)$ e supondo que $n = 2^m$ tem-se que

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg(n)-1} 2^i\frac{n}{2^i}\frac{1}{\ln(n/2^i)} + 2^{\lg(n)}\Theta(1) \\
&= \sum_{i=0}^{\lg(n)-1} \frac{n}{\ln(n/2^i)} + n\Theta(1) \\
&= n\sum_{i=0}^{\lg(n)-1} \frac{1}{\ln(n/2^i)} + \Theta(n) \\
&= n\sum_{i=1}^{\lg(n)} \frac{1}{\ln(2^i)} + \Theta(n) \\
&= \frac{n}{\ln(2)}\sum_{i=1}^{\lg(n)} \frac{1}{i} + \Theta(n) \\
&\leq \frac{n}{\ln(2)}\big(\lg(\lg(n))+1\big) + \Theta(n) \\
&\in \Theta(n\lg(\lg(n)))
\end{aligned}$$

(g) Pela árvore recursiva tem-se que

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-2} \frac{1}{n-i} + \Theta(1) \\
&= \sum_{i=2}^{n} \frac{1}{i} + \Theta(1) \\
&\leq \lg(n) + \Theta(1) * \\
&\in \Theta(\lg(n))
\end{aligned}$$

* Observe que de $\sum_{i=1}^{n}\frac{1}{i} \leq \lg(n)+1$ obtem-se a desigualdade acima.

(h) Pela árvore recursiva de $T(n)$ tem-se que

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-2} \ln(n-i) + \Theta(1) \\ &= \sum_{i=2}^{n} \ln(i) + \Theta(1) \\ &\geq \int_1^n \ln(i)\mathrm{d}i + \Theta(1) \\ &= n\ln(n) - n + 1 + \Theta(1) \\ &= (\ln(n) - 1)n + \Theta(1) \\ &\in \Theta(n\ln(n)) \end{aligned}$$

(i) Seja $S(m) = T(2^m) = 2^{m/2}T(2^{m/2}) + 2^m$. Então $S(m) = 2^{m/2}S(m/2) + 2^m$. Analisando árvore recursiva de $S(m)$ tem-se que

$$\begin{aligned} S(m) &= \prod_{i=1}^{\lg(m)} 2^{m/2^i} S(m/2^{\lg(m)}) + \lg(m)2^m \\ &= \prod_{i=1}^{\lg(m)} 2^{m/2^i} S(1) + \lg(m)2^m \\ &= 2^{\sum_{i=1}^{\lg m} m/2^i} S(1) + \lg(m)2^m \\ &= 2^{m\sum_{i=1}^{\lg m}(1/2)^i} S(1) + \lg(m)2^m \\ &\leq 2^m S(1) + \lg(m)2^m \end{aligned}$$

Portanto

$$T(n) \leq nT(2) + n\lg(\lg(n)) \in \Theta(n\lg(\lg(n)))$$

3. (a) Para $\phi = (1+\sqrt{5})/2$ e $\tilde{\phi} = (1-\sqrt{5})/2$ tem-se que

$$\begin{aligned} \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\tilde{\phi}z}\right) &= \frac{1}{\sqrt{5}}\frac{\phi z - \tilde{\phi}z}{(1-\phi z)(1-\tilde{\phi}z)} \\ &= \frac{1}{\sqrt{5}}\frac{(\phi - \tilde{\phi})z}{(1-\phi z)(1-\tilde{\phi}z)} \\ &= \frac{1}{\sqrt{5}}\frac{\sqrt{5}z}{(1-\phi z)(1-\tilde{\phi}z)} \\ &= \frac{z}{(1-\phi z)(1-\tilde{\phi}z)} \\ &= \frac{z}{1-\tilde{\phi}z-\phi z+\phi\tilde{\phi}z^2} \\ &= \frac{z}{1-z-z^2} \end{aligned}$$

Por outro lado tem-se que

$$\begin{aligned}
\mathcal{F}(z) &= \sum_{i=0}^{\infty} F(i)z^i \\
&= \sum_{i=1}^{\infty} F(i)z^i \\
&= z + \sum_{i=2}^{\infty} F(i)z^i \\
&= z + \sum_{i=2}^{\infty} (F(i-1) + F(i-2))z^i \\
&= z + \sum_{i=2}^{\infty} F(i-1)z^i + \sum_{i=2}^{\infty} F(i-2)z^i \\
&= z + z\sum_{i=2}^{\infty} F(i-1)z^{i-1} + z^2\sum_{i=2}^{\infty} F(i-2)z^{i-2} \\
&= z + z\sum_{i=1}^{\infty} F(i)z^i + z^2\sum_{i=0}^{\infty} F(i)z^i \\
&= z + z\mathcal{F}(z) + z^2\mathcal{F}(z)
\end{aligned}$$

Logo,

$$z = \mathcal{F}(z) - z\mathcal{F}(z) - z^2\mathcal{F}(z) = \mathcal{F}(z)(1 - z - z^2)$$

Portanto,

$$\mathcal{F}(z) = \frac{z}{1 - z - z^2}$$

Consequentemente,

$$\mathcal{F}(z) = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\tilde{\phi} z}\right)$$

(b) Para $\tilde{\phi} < z < 1 - \phi$,

$$\begin{aligned}
\sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \tilde{\phi}^i)z^i &= \frac{1}{\sqrt{5}}\left(\sum_{i=0}^{\infty}(\phi z)^i - \sum_{i=0}^{\infty}(\tilde{\phi} z)^i\right) \\
&= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\tilde{\phi} z}\right) \\
&= \mathcal{F}(z)
\end{aligned}$$

(c) De $\sum_{i=0}^{\infty} F(i)z^i = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \tilde{\phi}^i)z^i$ tem-se que $F(i) = \frac{1}{\sqrt{5}}(\phi^i - \tilde{\phi}^i)$, $\forall i \geq 0$. Observe que $\frac{|\tilde{\phi}^i|}{\sqrt{5}} < \frac{1}{\sqrt{5}} < \frac{1}{2}$. Portanto $F(i) = \frac{\phi^i}{\sqrt{5}}$, aproximado ao inteiro mais próximo.

(d) Para $i \geq 0$,

$$\begin{aligned} F(i+2) &= F(i) + F(i+1) \\ &= \frac{1}{\sqrt{5}}(\phi^i - \tilde{\phi}^i + \phi^{i+1} - \tilde{\phi}^{i+1}) \\ &= \phi^i \frac{(\phi+1)}{\sqrt{5}} - \tilde{\phi}^i \frac{(\tilde{\phi}+1)}{\sqrt{5}} \\ &\geq \phi^i \frac{(\phi+1)}{\sqrt{5}} - |\tilde{\phi}|^i \frac{(\tilde{\phi}+1)}{\sqrt{5}} \end{aligned}$$

Note que $(1+\tilde{\phi})/\sqrt{5} > 0$, $|\tilde{\phi}|^i \leq 0$ e $(1+\phi)/\sqrt{5} > 1$. Logo, se $|\tilde{\phi}|^i \frac{(\tilde{\phi}+1)}{\sqrt{5}} \leq \phi^i\big(\frac{(\phi+1)}{\sqrt{5}} - 1\big)$, então $F(i+2) \geq \phi^i$.

Observe que $\frac{(\phi+1)}{\sqrt{5}} - 1 = \frac{(\tilde{\phi}+1)}{\sqrt{5}}$ e que $|\tilde{\phi}|^i \leq \phi^i$ para todo $i \geq 0$ ($|\tilde{\phi}|^i \to 0$, $\phi^i \to \infty$ quando $i \to \infty$). Assim,

$$\phi^i \frac{(\phi+1)}{\sqrt{5}} - |\tilde{\phi}|^i \frac{(\tilde{\phi}+1)}{\sqrt{5}} \geq \phi^i \frac{(\phi+1)}{\sqrt{5}} - \phi^i\Big(\frac{(\phi+1)}{\sqrt{5}} - 1\Big) = \phi^i$$

(e) Por (d) tem-se que, para $n \geq 2$

$$F(n) \geq \phi^{n-2} = \frac{1}{\phi^2}\phi^n \in \Omega(\phi^n)$$

Por (c) tem-se que

$$F(n) = \frac{1}{\sqrt{5}}(\phi^n - \tilde{\phi}^n) \leq \frac{1}{\sqrt{5}}(\phi^n + |\tilde{\phi}^n|) \in O(\phi^n)$$

Portanto, $F(n) \in \Theta(\phi^n)$.

4. (a) Para $n = 1$, basta mover o disco de A para C.

Para $n > 1$, observe que são feitos $M(n-1)$ movimentos para transportar $n-1$ discos de A para B, usando C como auxiliar. Um movimento do maior disco é feito de A para C e depois mais $M(n-1)$ movimentos dos $n-1$ discos em B para C, usando A como torre auxiliar.

(b) $M(n) = \sum_{i=0}^{n-1} 2^i = \frac{2^n-1}{2-1} = 2^n - 1 \in \Theta(2^n)$.

Algoritmos de Ordenação

5.

$$\begin{aligned} W(n) = \sum_{i=1}^{n-1} \lfloor \lg(i) \rfloor + 1 &\leq \sum_{i=1}^{n-1} \lg(i) + 1 \\ &\leq \int_1^n \lg(i) + 1 \, \mathrm{d}i \\ &= \frac{1}{\ln(2)} \int_1^n \ln(i) \mathrm{d}i + \int_1^n 1 \, \mathrm{d}i \\ &= \frac{1}{\ln(2)} \big(n \ln(n) - n + 1\big) + (n-1) \\ &= n \lg(n) + n \left(1 - \frac{1}{\ln(2)}\right) + \frac{1}{\ln(2)} - 1 \\ &\in \Theta(n \lg(n)) \end{aligned}$$

6. A seguir uma análise do número de possíveis soluções, pensando nas árvores de decisão associadas aos possíveis algoritmos.

   Assim, para a situação inicial $a_1 > a_2 > a_3 > a_4$ e $b_1 > b_2$, as possíveis respostas são 15. As árvores de decisão com 15 folhas terão no mínimo a altura $\lceil \lg 15 \rceil = 4$. Existe então algum algoritmo associado a uma árvore de decisão que precise no pior caso de 4 comparações? Deve-se então analisar os passos de um algoritmo hipotético.

   Suponha que o primeiro passo de um algoritmo qualquer seja comparar b_1 com uma das quatro chaves a_1, a_2, a_3 ou a_4 (caso para b_2 é análogo). Logo, se a primeira comparação do algoritmo é $a_1 : b_1$ tem-se duas possibilidades: $a_1 < b_1$ e tem-se uma lista ordenada de 5 elementos e uma pendência (o elemento b_2) ou $a_1 > b_1$ com uma lista ordenada de 3 elementos e 3 pendências. No primeiro sub-caso, as possíveis respostas serão 5 e no segundo sub-caso serão 10. Portanto, no pior caso tem-se uma sub-árvore de decisão com altura $\lceil \lg 10 \rceil = 4$. Consequentemente, tem-se uma total para o pior caso de 5 comparações (no mínimo).

   Os casos de algoritmos que realizem como primeira comparação $a_2 : b_1$, $a_3 : b_1$ e $a_4 : b_1$ são analisados similarmente e em todos eles tem-se pelo menos mais 4 comparações no pior caso.

   Novamente, algoritmos que escolhem b_2 para primeira comparação são análogos.

7. Para a otimização do algoritmo de busca binária para listas de comprimento 5 (Figura 1), a, b, c, d e e por exemplo, elementos isolados são inserido em listas unitárias obtendo $a < b, c < d, e$. Observe que desse ponto a inserção do elemento isolado e em uma das duas listas de comprimento 2 não é conveniente porque seria melhor fazer uma inserção binária em uma lista de tamanho 3, pois ambos tem 2 comparações no pior caso. Assim, as duas maiores chaves das duas listas de tamanho 2 são comparadas, obtendo a estrutura quádrupla ilustrada no terceiro

passo da Figura 1. Note que esse passo resume-se a uma inserção binária em uma lista unitária. A chave isolada e pode então ser inserida em uma lista de tamanho 3, obtendo-se o resultado ilustrado no quarto passo da figura ou um resultado mais simples, onde a chave e é maior que a maior chave do quarteto. Finalmente, no quinto passo, a chave pendente é inserida binariamente em uma lista de tamanho 2 ou 3. Observe que nos últimos dois passo a busca binária é usada de maneira ótima.

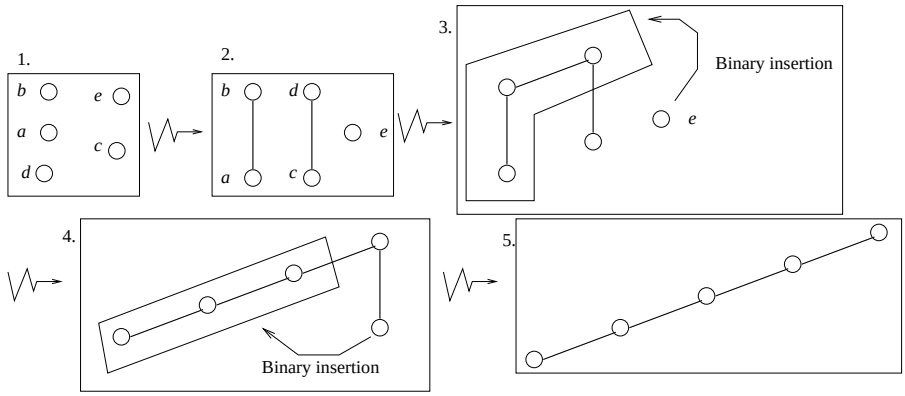

Figure 1: Ordenação com merge sort otimizado para listas de comprimento 5

8. Como descrito no enunciado, a idéia básica por trás dos métodos de otimização dos algoritmos de ordenação é utilizar-se de um mecanismo de escape na recursividade de tal forma que o algortimo ótimo para uma entrada de tamanho específico seja chamado, evitando comparações desnecessárias.

   Para o exemplo do merge otimizado tem-se

$$T(5\,2^k) = \sum_{i=0}^{k-1} 2^i c \frac{n}{2^i} + 7\,2^k = (5ck + 7)2^k$$

   enquanto o algoritmo sem otimização teria uma custo de $(5ck + 8)2^k$. Portanto, o algoritmo otimizado faria 2^k comparações a menos que o algoritmo usual.

9. (a) **Insertion Sort**

   Para o insertion sort, o pior caso acontece quando tem-se zeros no fim da lista, equivalendo a uma lista ordenada na ordem inversa (que é o pior caso do algoritimo). Portanto, para uma lista de comprimento n com $n/2$ zeros no

fim da lista tem-se que

$$\begin{aligned} W(n) &= (\frac{n}{2}-1)+\frac{n}{2}+\sum_{i=n/2+1}^{n}\frac{n}{2}+1 \\ &= n-1+\sum_{i=1}^{n/2-1}\frac{n}{2}+1 \\ &= n-1+\frac{n}{2}-1+\Big(\frac{n}{2}-1\Big)\frac{n}{2} \\ &= \frac{n^2}{4}+n-2 \in \Theta(n^2) \end{aligned}$$

Merge Sort

A comparação feita pelo merge sort usando $<$ não aproveita a informação de quando duas chaves são iguais. O pior caso seria quando a lista tem 0's e 1's intercalados por toda lista. Assim, para cada iteração com uma lista de comprimento n teríam-se $3n/4$ comparações e, para $n = 2$, $W(2) = T(2) = 1$. Logo,

$$\begin{aligned} W(n) &= \sum_{i=0}^{\lg(n)-1} 2^i\frac{3}{4}\frac{n}{2^i}+2^{\lg(n)-1} \\ &= \sum_{i=0}^{\lg(n)-1}\frac{3}{4}n+\frac{n}{2} \\ &= \frac{3}{4}n\lg(n)+\frac{n}{2} \in \Theta(n\lg(n)) \end{aligned}$$

Quick Sort

O pior caso, como para uma lista sem entradas repetidas, seria tal que o problema original de tamanho n seja dividido em um subproblema de tamanho $n-1$ e outro de tamanho 0. Uma lista com apenas 0's ou 1's teria esse comportamento. Assim, $W(n) = \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{1}{2}(n+1)n \in \Theta(n^2)$.

(b) ENTRADA: lista L com 0's e 1's de tamanho n
SAÍDA: L com todos os 0's, caso haja algum, nas primeiras posições.

```
BEGIN

 zero=1;

  FOR (i=1,i<=n,i++)

  {
    IF (L[i] == 0)

     {
        EXCHANGE L[i] <-> L[zero];
```

```
        zero++;
     }

  }

END.
```

O pior caso será o de uma lista com apenas 0's. Assim, para um custo constante c de troca de chaves,

$$W(n) = \sum_{i=1}^{n} 1 + 1 + c = (2 + c)n = \Theta(n)$$

(c) ENTRADA: lista L com chaves "vermelha", "branca" e "azul" de tamanho n
SAÍDA: L ordenada por "vermelha"'s $<$ "branca"'s $<$ "azul"'s.

```
BEGIN

 vermelho=1, azul=n ;

  FOR (i=1,i<=azul,i++)

  {
    IF (L[i] == "vermelho")

     {
        EXCHANGE L[i] <-> L[vermelho];
        vermelho++;
     }

    ELSE IF (L[i] == "azul")

        {
            EXCHANGE L[i] <-> L[azul];
            azul--;
            i--;
        }

  }

END.
```

Observe que o pior caso será com uma lista com apenas "azul", pois teríamos 3 comparações e uma troca de chaves desnecessária a cada iteração. Com um

custo constante c para troca de chaves tem-se

$$W(n) = \sum_{i=1}^{n} 3 + c = (3 + c)n = \Theta(n)$$

10. Exercício de leitura (Verifique na Bibliografia o capítulo de [Baa88] sobre Sorting).