

1) Podemos testar se a entrada é o caso especial e diretamente retornar a resposta certa.

2) Não, pois existem entradas que fazem o insertion sort executar em tempo xn , para alguma constante x , mas $xn \neq \Theta(n^2)$ pois, em particular, $xn \neq \Omega(n^2)$.

3) --

4) Primeiro vamos verificar que $\max\{f(n), g(n)\} = O(f(n) + g(n))$.

Note que independente de termos $f(n) = \max\{f(n), g(n)\}$ ou $g(n) = \max\{f(n), g(n)\}$, vale que

$$\max\{f(n), g(n)\} \leq f(n) + g(n)$$

(pois $f(n) \leq f(n) + g(n)$ e $g(n) \leq f(n) + g(n)$).

Então tomando $c=1$ e $n_0=1$ temos o resultado.

Agora vamos verificar que $\max\{f(n), g(n)\} = \Omega(f(n) + g(n))$.

Suponha, s.p.g, que $f(n) = \max\{f(n), g(n)\}$. Então temos que $f(n) \geq g(n)$. Assim

$$f(n) = \frac{f(n)}{2} + \frac{f(n)}{2} \geq \frac{f(n)}{2} + \frac{g(n)}{2} = \frac{f(n) + g(n)}{2}$$

Note que o mesmo valeria se $g(n) = \max\{f(n), g(n)\}$. Logo,

$$\max\{f(n), g(n)\} \geq \frac{1}{2}(f(n) + g(n))$$

Então tomando $c = \frac{1}{2}$ e $n_0 = 1$ temos o resultado.

5) (a) $f(n) = \sqrt{n}$ e $g(n) = n^{\frac{2}{3}}$

Claramente, $\sqrt{n} = n^{\frac{1}{2}} \leq n^{\frac{2}{3}}$, sempre.

Logo, tomando $c=1$ e $n_0=1$ vale que $\sqrt{n} = O(n^{\frac{2}{3}})$.

Suponha $\sqrt{n} = \Omega(n^{\frac{2}{3}})$, então devem haver constantes c e n_0 tais que $\sqrt{n} \geq c \cdot n^{\frac{2}{3}} \quad \forall n \geq n_0$.
 Isolando o c , temos $c \leq \frac{n^{\frac{1}{2}}}{n^{\frac{2}{3}}} = \frac{1}{n^{\frac{1}{6}}}$.
 mas $\frac{1}{n^{\frac{1}{6}}}$ tende a 0 conforme n cresce, e portanto
 não é possível achar c constante menor.
 Logo, $\sqrt{n} \neq \Omega(n^{\frac{2}{3}})$.

(b) $f(n) = n \log n$ e $g(n) = 10n \log(10n)$

note que $10n \log(10n) = 10n \log 10 + 10n \log n$.

Claramente, $n \log n \leq 10n \log n \leq 10n \log n + 10n \log 10 = g(n)$.

Então tomando $c=1$ e $n_0=1$, $n \log n = O(10n \log(10n))$.

Também vale que

$$\begin{aligned} n \log n &= \frac{20}{20} n \log n = \frac{1}{20} (10n \log n + 10n \log 10) \\ &\geq \frac{1}{20} (10n \log n + 10n \log 10) = \frac{1}{20} 10n \log(10n) \end{aligned}$$

onde a desigualdade vale se $n \geq 10$.

Logo, tomando $c=\frac{1}{20}$ e $n_0=10$, $n \log n = \Omega(10n \log(10n))$.

Claramente, $n \log n = \Theta(10n \log(10n))$.

$$(c) f(n) = \frac{n}{1000} \text{ e } g(n) = 50^{100}$$

Suponha para fins de contradicção que $\frac{n}{1000} = O(50^{100})$.

Então existem constantes c e n_0 tais que

$$\frac{n}{1000} \leq c \cdot 50^{100} \quad \forall n \geq n_0$$

Isolando c , temos que $c \geq \frac{n}{1000 \cdot 50^{100}}$.

mas $\frac{n}{1000 \cdot 50^{100}}$ ~~essa~~ tende a ∞ conforme n tende a ∞ .

Logo, não é possível que c seja menor que essa expressão, contradicção. Então $\frac{n}{1000} \neq O(50^{100})$.

Note que $\frac{n}{1000} \geq 50^{100}$ sempre que $n \geq 1000 \cdot 50^{100}$

Logo, tomando $c=1$ e $n_0 = 1000 \cdot 50^{100}$, $\frac{n}{1000} = \Omega(50^{100})$.

$$(d) f(n) = \log_a n \text{ e } g(n) = \log_b n$$

Por propriedades logarítmicas, vale que

$$\log_a n = \frac{\log_b n}{\log_b a}$$

então tomando $c = \frac{\log_b a}{\log_b n}$ e $n_0 = 1$, vale que

$\log_a n = O(\log_b n)$, $\log_b n = \Omega(\log_b n)$ e, portanto,
 $\log_a n = \Theta(\log_b n)$.

$$(e) f(n) = 1000 \log n \text{ e } g(n) = \log(n^2)$$

Note que $\log(n^2) = 2 \log n$.

Como $1000 \log n = 500 \cdot 2 \log n$, podemos tomar

$c = 500$ e $n_0 = 0$ para mostrar que $1000 \log n = O(\log(n^2))$,
 $1000 \log n = \Omega(\log(n^2))$ e, portanto, $1000 \log n = \Theta(\log(n^2))$.

$$(f) f(n) = 100^{n+a} \quad e \quad g(n) = 100^n$$

note que $100^{n+a} = 100^n \cdot 100^a$ ~~$\leq 100^n$~~ $\forall n \geq 1$.

Claramente, tomando $c = 100^a$ e $n_0 = 1$ temos que $100^{n+a} = O(100^n)$, $100^{n+a} = \Omega(100^n)$ e, logo, $100^{n+a} = \Theta(100^n)$.

$$(g) f(n) = 99^{n+a} \quad e \quad g(n) = 100^n$$

note que $99^{n+a} = 99^n \cdot 99^a$ e, claramente, $99^n \leq 100^n$.

Assim, $99^{n+a} = 99^a \cdot 99^n \leq 99^a \cdot 100^n$.

Logo, tomando $c = 99^a$ e $n_0 = 1$, $99^{n+a} = O(100^n)$.

Suponha que $99^{n+a} = \Omega(100^n)$. Então existem constantes c e n_0 tais que $99^{n+a} \geq c \cdot 100^n \quad \forall n \geq n_0$.

Isolando o c , temos que $c \leq \frac{99^n \cdot 99^a}{100^n} = 99^a \left(\frac{99}{100}\right)^n$.

Mas $\left(\frac{99}{100}\right)^n$ tende a 0 conforme n cresce, então é um absurdo que exista c menor que $99^a \left(\frac{99}{100}\right)^n$. Logo, $99^{n+a} \neq \Omega(100^n)$.

$$(h) f(n) = 100^{an} \quad e \quad g(n) = 99^n \quad (\text{suponha } a > 1)$$

note que se $100^{an} = O(99^n)$, então valeria que $100^{an} \leq c \cdot 99^n$ para alguma constante c . Mas então valeria $c \geq \left(\frac{100}{99}\right)^n$, o que é impossível pois $\left(\frac{100}{99}\right)^n$ cresce indefinidamente. Então $100^{an} \neq O(99^n)$.

Por outro lado, claramente $100^a \geq 99$, de modo que $100^{an} \geq 99^n$. Então tomando $c = 1$ e $n_0 = 1$, temos que $100^{an} = \Omega(99^n)$.

$$(i) f(n) = n^{1.01} \quad e \quad g(n) = n \log^2 n$$

Esse exercício não é tão direto quanto os outros.

Primeiro note que estamos comparando $n^{1.01} = n \cdot n^{0.01} = n^{\frac{101}{100}}$ com $n \cdot \log^2 n$ (ou seja, basicamente $n^{\frac{101}{100}}$ com $\log^2 n$).

A intenção desse exercício é mostrar que qualquer polinômio em n (mesmo $n^{\frac{100}{100}}$) é maior do que o \log .

Veja a tabela a seguir:

n	$n^{\frac{101}{100}}$	$\log^2 n$
1	1	①
$2^{\frac{100}{100}}$	2	100^2
$4^{\frac{100}{100}}$	4	200^2

Ela pode nos levar à conclusão que $\log^2 n$ é maior que $n^{\frac{100}{100}}$, mas isso está errado! Para um valor bem grande de n (aproximadamente $2,6 \times 10^{669}$), $n^{\frac{101}{100}}$ passa a ser maior sempre.

Em particular,

$$\lim_{n \rightarrow \infty} \frac{n^{0.01}}{\log^2 n} = \lim_{n \rightarrow \infty} \frac{0.005 \cdot n^{0.01}}{\log n} = \lim_{n \rightarrow \infty} 0.00005 \cdot n^{0.01} = \infty$$

ou seja, $n^{0.01}$ é realmente maior que $\log^2 n$ conforme n cresce.

Assim, $n^{1.01} \geq n \log^2 n$ para $n \geq 10^{700}$, ou seja, tomando $c=1$ e $n_0 = 10^{700}$ temos que $n^{1.01} = \Omega(n \log^2 n)$.

□□ Observe que o cálculo do limite acima já prova que existe algum n_0 para o qual $n^{1.01} \geq n \log^2 n$ sempre que $n \geq n_0$, de forma que especificá-lo não era necessário aqui. Inclusive, só consegui encontrá-lo pois tinha acesso a outros ferramentas. ▷

Por outro lado, se valesse que $n^{1.01} \leq c \cdot n \log^2 n$ para alguma constante c e $n \geq n_0$, então seria o caso de $c \geq \frac{n^{0.01}}{\log^2 n}$, o que é um absurdo pois vimos que $\log^2 n$ cresce indefinidamente.

Então $n^{1.01} \neq O(n \log^2 n)$.

$$(j) f(n) = 3^n \text{ e } g(n) = 2^n$$

Se fosse o caso de $3^n = O(2^n)$, deveria haver constantes c e n_0 tais que $3^n \leq c \cdot 2^n$ sempre que $n \geq n_0$. mas então $c \geq \left(\frac{3}{2}\right)^n$, o que é impossível pois $\left(\frac{3}{2}\right)^n$ cresce indefinidamente. Então $3^n \neq O(2^n)$.

Por outro lado, claramente $3^n \geq 2^n$, de forma que tomado $c=1$ e $n_0=1$ temos que $3^n = \Omega(2^n)$.

$$(k) f(n) = \log \sqrt{n} \text{ e } g(n) = \log n.$$

Como $\log \sqrt{n} = \log n^{\frac{1}{2}} = \frac{1}{2} \log n$, claramente tomando $c = \frac{1}{2}$ e $n_0 = 1$ temos que $\log \sqrt{n} = O(\log n)$, $\log \sqrt{n} = \Omega(\log n)$ e, portanto, $\log \sqrt{n} = \Theta(\log n)$.

$$(l) f(n) = \log n! \text{ e } g(n) = n \log n$$

Por definição, $n! = 1 \cdot 2 \cdot 3 \cdots \underbrace{m}_{n \text{ termos}}$.

Todos os termos são $\leq m$, logo, $n! \leq \overbrace{m \cdot m \cdots m}^{n \text{ termos}} = m^n$. (*)

Mínimo dos termos são $\geq \frac{m}{2}$, logo, $n! \geq \underbrace{\frac{m}{2} \cdot \frac{m}{2} \cdots \frac{m}{2}}_{\frac{m}{2} \text{ termos}} = \left(\frac{m}{2}\right)^{\frac{m}{2}}$. (**)

Tomando log na expressão (*), temos que

$$\log n! \leq \log m^n = n \cdot \log m$$

assim, tomando $c=1$ e $n_0=1$, $\log n! = O(n \log n)$.

Tomando log na expressão (**), temos que

$$\log n! \geq \log \left(\frac{m}{2}\right)^{\frac{m}{2}} = \frac{m}{2} \log \frac{m}{2} = \frac{m}{2} (\log m - \log 2)$$

$$= \frac{m}{2} \log m - \frac{m}{2} \log 2 \geq \frac{m}{2} \log m - \frac{m}{4} \log m \quad (***)$$

$$= \left(\frac{1}{2} - \frac{1}{4}\right) m \log m = \frac{1}{4} m \log m$$

onde a desigualdade em (***) vale sempre que $\log m \geq 2$.

Assim, tomando $c = \frac{1}{4}$ e $n_0 = 4$, temos $\log n! = \Omega(n \log n)$.

Logo, $\log n! = \Theta(n \log n)$.

6) BASE: Se $m=6$, $F_6=8$ por definição e $2^{0,5 \cdot 6} = 2^3 = 8$.

Se $m=7$, $F_7=13$ por def. e $2^{0,5 \cdot 7} \approx 11,314$.

Logo $F_m \geq 2^{0,5m}$ no caso base.

HIPÓTESE: $F_k \geq 2^{\frac{k}{2}}$ para $k < m$.

PASSO: seja $m > 7$. Temos

$$\begin{aligned} F_m &= F_{m-1} + F_{m-2} && (\text{por def.}) \\ &\geq 2^{\frac{m-1}{2}} + 2^{\frac{m-2}{2}} && (\text{por hip.}) \\ &= 2^{\frac{m}{2}-\frac{1}{2}} + 2^{\frac{m}{2}-1} \\ &= \frac{1}{\sqrt{2}} 2^{\frac{m}{2}} + \frac{1}{2} 2^{\frac{m}{2}} \\ &= \left(\frac{1}{\sqrt{2}} + \frac{1}{2}\right) 2^{\frac{m}{2}} \geq 2^{\frac{m}{2}} && (\text{pois } \frac{1}{\sqrt{2}} + \frac{1}{2} \geq 1) \end{aligned}$$

7) --

8) Primeiro considere o alg. PARTITION.

A frase "no começo de cada iteração do laço "para", o subvetor $A[i_{\text{ini}}+1..i-1]$ tem elementos menores que o pivô" e o subvetor $A[i..j-1]$ tem elementos maiores que o pivô" é uma invariante:

(1) antes do laço começar, quando $j = i_{\text{ini}}+1$ e $i = i_{\text{ini}}+1$, os subvetores $A[i_{\text{ini}}+1..i-1] = A[i_{\text{ini}}+1..i_{\text{ini}}]$ e $A[i..j-1] = A[i_{\text{ini}}+1..i_{\text{ini}}]$ estão vazios e, portanto, a frase vale trivialmente.

(2) considere uma iteração qualquer e suponha que temos $A[i_{\text{ini}}+1..i-1] < \text{pivô}$ e $A[i..j-1] > \text{pivô}$ (abuso de notação aqui).

Nessa iteração podemos ter $A[j] < \text{pivô}$ ou o contrário.

Se $A[j] < \text{pivô}$, então o algoritmo troca $A[j]$ com $A[i]$ e incrementa i . Assim, $A[i_{\text{ini}}+1..i]$ tem elementos menores que o pivô e $A[i+1..j]$ tem elementos maiores (pois antes do incremento $A[i]$ era, por suposição, maior que o pivô). Ou seja, a frase vale antes da próxima iteração (quando temos $j+1$).

Se $A[j] > \text{pivô}$, o algoritmo não faz nada, de forma que $A[i_{\text{ini}}+1..i-1]$ continua com elementos menores que o pivô

e $A[i..j]$ tem elementos maiores. Ou seja, a frase vale para a próxima iteração.

Ela é uma invariante útil, pois ao fim temos $j = \text{fim} + 1$, e a invariante nos diz que $A[\text{ini}+1..i-1] < \text{pivô}$ e $A[i..j] > \text{pivô}$. Como o PARTITION ainda troca $A[\text{ini}]$ com $A[i-1]$, temos que ele termina com $A[\text{ini}..i-2] < \text{pivô}$, $A[i-1] = \text{pivô}$ e $A[i..j] > \text{pivô}$, isto é, o vetor de fato está partitionado ao redor do pivô.

Agora considere o algoritmo Quicksort. Vamos usar indução no tamanho n do vetor recebido para provar que ele ordena qualquer vetor.

BASE: $n=1$. Nesse caso o algoritmo não faz nada. De fato, um vetor com um elemento já está ordenado. Então no caso base o algoritmo está correto.

HIPÓTESE: o Quicksort ordena corretamente um vetor com K elementos, onde $K \leq n$.

PASSO: seja $n > 1$. A primeira coisa que o algoritmo faz é chamar o PARTITION, garantindo que o pivô escolhido está na posição ini . Como visto acima, o PARTITION corretamente partitiona o vetor dodo, deixando o pivô em uma posição q .

O algoritmo então recursivamente ordena $A[\text{ini}..q-1]$ e $A[q+1..fim]$ corretamente, por hipótese de indução (note que ambos subvetores têm tamanho menor que n pois pelo menos o elemento pivô foi removido). Como o elemento pivô está em sua posição final, todos à sua esquerda são menores e estão ordenados e todos à sua direita são maiores e estão ordenados, temos que o vetor todo está ordenado.

9) Para montar o vetor dividido em três partes, precisaremos de uma variável extra para montar essa nova "fronteira". Especificamente, queremos que a qualquer momento o vetor esteja da seguinte forma:

ini	i	j	k	fim
p	$< p$	$= p$	$> p$??

NovoPARTITION (A , ini , fim)

$$p = A[ini]$$

$$i = j = ini + 1$$

para $K = ini + 1$ até fim

: se $A[K] < p$

: troque $A[K]$ com $A[i]$

: $i = i + 1$

: se $A[K] == p$

: troque $A[K]$ com $A[j]$

: $j = j + 1$

troque $A[ini]$ com $A[i - 1]$

→ claramente

ainda está em
tempo linear

NovoQUICKSORT (A , ini , fim)

se $ini \geq fim$

retorne

escolha um pivô e o coloque em ini

NovoPARTITION (A , ini , fim)

sejam p e q as posições tais que $A[p] = A[p+1] = \dots = A[q] = \text{pivô}$

// com $A[p-1] < \text{pivô}$ e $A[q+1] > \text{pivô}$

NovoQUICKSORT (A , ini , $p-1$)

NovoQUICKSORT (A , $q+1$, fim)

10) (a) $T(n) = T\left(\frac{n}{3}\right) + n$ (iteração)

$$\begin{aligned} T(n) &= T\left(\frac{n}{3}\right) + n \\ &= T\left(\frac{n}{9}\right) + \frac{n}{3} + n = T\left(\frac{n}{3^2}\right) + \frac{n}{3} + n \\ &= T\left(\frac{n}{27}\right) + \frac{n}{9} + \frac{n}{3} + n = T\left(\frac{n}{3^3}\right) + \frac{n}{3^2} + \frac{n}{3} + n \\ &= \dots = T\left(\frac{n}{3^i}\right) + \frac{n}{3^{i-1}} + \dots + \frac{n}{3} + n \\ &= T\left(\frac{n}{3^i}\right) + \sum_{k=0}^{i-1} \frac{n}{3^k} = T\left(\frac{n}{3^i}\right) + n \cdot \sum_{k=0}^{i-1} \left(\frac{1}{3}\right)^k \end{aligned}$$

Como $\frac{n}{3^i} = 1$ quando $i = \log_3 n$, temos

$$\begin{aligned} T(n) &= T(1) + n \left(\sum_{k=0}^{\log_3 n - 1} \left(\frac{1}{3}\right)^k \right) = 1 + n \left(\frac{1 - \left(\frac{1}{3}\right)^{\log_3 n}}{1 - \frac{1}{3}} \right) \\ &= 1 + \frac{3}{2}n \left(1 - \frac{1}{n}\right) = \frac{3}{2}n - \frac{1}{2} \end{aligned}$$

Assim, $T(n) = \Theta(n)$ (pois só usamos igualdades acima e a expressão $\frac{3}{2}n - \frac{1}{2}$ é claramente $O(n)$ e $\Omega(n)$).

□□ Podermos ter simplificado os contos acima, caso fosse necessário, já que o exercício só pede para resolver com O :

$$\begin{aligned} T(n) &= T\left(\frac{n}{3}\right) + n \\ &= T\left(\frac{n}{9}\right) + \frac{n}{3} + n \leq T\left(\frac{n}{9}\right) + n + n = T\left(\frac{n}{3^2}\right) + 2n \\ &= T\left(\frac{n}{27}\right) + \frac{n}{9} + n + n \leq T\left(\frac{n}{27}\right) + n + n + n = T\left(\frac{n}{3^3}\right) + 3n \\ &= \dots = T\left(\frac{n}{3^i}\right) + in \\ &= T(1) + (\log_3 n)n = 1 + (\log_3 n)n \end{aligned}$$

Como usamos \leq acima, só podemos afirmar que $T(n) = O(n \log n)$.

Perceba como os contos ficaram mais fáceis. No entanto, abrimos mão de uma análise apertada.

De fato, $T(n) = O(n^{\log_3 2})$, mas saber que $T(n) = \Theta(n)$ é muito mais útil. ▷

$$(b) T(n) = aT\left(\frac{n}{a}\right) + m \quad (\text{iteração})$$

$$T(n) = aT\left(\frac{n}{a}\right) + m$$

$$= a\left[aT\left(\frac{n}{a^2}\right) + \frac{m}{a}\right] + m = a^2T\left(\frac{n}{a^2}\right) + m + m$$

$$= a^2\left[aT\left(\frac{n}{a^3}\right) + \frac{m}{a^2}\right] + 2m = a^3T\left(\frac{n}{a^3}\right) + m + 2m$$

$$= \dots = a^i T\left(\frac{n}{a^i}\right) + im$$

Como $\frac{n}{a^i} = 1$ quando $i = \log_a n$, temos

$$T(n) = a^{\log_a n} T(1) + m \cdot \log_a n$$

$$= m + m \log_a n$$

$$\text{Assim, } T(n) = \Theta(m \log n).$$

$$(c) T(n) = 2T(n-1) + m \quad (\text{iteração})$$

$$T(n) = 2T(n-1) + m$$

$$= 2[2T(n-2) + m - 1] + m = 4T(n-2) + 2m - 2 + m$$

$$= 4[2T(n-3) + m - 2] + 3m - 2 = 8T(n-3) + 4m - 8 + 3m - 2$$

$$= 8[2T(n-4) + m - 3] + 7m - 10 = 16T(n-4) + 8m - 24 + 7m - 10$$

$$= \dots = 2^i T(n-i) + \sum_{k=0}^{i-1} 2^k (m-k)$$

Como $n-i=1$ quando $i=n+1$, temos

$$T(n) = 2^{n+1} T(1) + \sum_{k=0}^n 2^k (m-k) = 2^{n+1} + \left(\sum_{k=0}^n 2^k m \right) - \left(\sum_{k=0}^n 2^k k \right) \quad (*)$$

$$= 2^{n+1} + (m2^{n+1} - m) + (m2^{n+1} + 2 - 2^{n+1})$$

$$= m2^{n+2} - m + 2$$

$$\text{Assim, } T(n) = \Theta(m2^n)$$

□ Poderíamos ter simplificado em (*):

$$T(n) = 2^{n+1} + \sum_{k=0}^n 2^k m - \sum_{k=0}^n 2^k k \leq 2^{n+1} + \sum_{k=0}^n 2^k m = 2^{n+1} + m2^{n+1} - m$$

de onde temos $T(n) = O(m2^n)$.

Ou então durante a iteração:

$$T(n) = 2T(n-1) + m$$

$$= 2[2T(n-2) + m - 1] + m = 4T(n-2) + 2m - 2 + m \leq 4T(n-2) + 4m$$

$$= 4[2T(n-3) + m - 2] + 4m = 8T(n-3) + 4m - 8 + 4m \leq 8T(n-2) + 8m$$

$$= \dots = 2^i T(n-i) + 2^i m = 2^{n+1} T(1) + 2^{n+1} m$$

de onde também temos $T(n) = O(m2^n)$. ▷

$$(d) T(n) = 4T\left(\frac{n}{2}\right) + n \quad (\text{substituição, suponha } T(n) = \Theta(n^2))$$

Vamos mostrar que $T(n) \leq c \cdot n^2$ para alguma const. c .
 BASE: $n=1$. Aqui $T(1)=1$ por definição. Teremos $c \cdot 1^2 \geq 1$
 se escolhermos $c \geq 1$.

HIPÓTESE: $T(k) \leq c \cdot k^2$ para $k < n$.

PASSO: seja $T(n) = 4T\left(\frac{n}{2}\right) + n$. Por hipótese, $T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2$.

$$\text{Então } T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$\leq 4c\left(\frac{n}{2}\right)^2 + n = cm^2 + m$$

~~folhou!~~

Não conseguimos mostrar que $T(n) \leq cm^2$.

Vamos mostrar que $T(n) \leq cm^2 + dm$ p/ constantes c e d .

BASE: $n=1$. Teremos $c+d \geq 1$ se $c \geq 1-d$.

HIPÓTESE: $T(k) \leq ck^2 + dk$ para $k < n$.

PASSO: seja $T(n) = 4T\left(\frac{n}{2}\right) + n$. Por HI, $T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 + d\left(\frac{n}{2}\right)$.

$$\text{Então } T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$\leq 4\left(c\left(\frac{n}{2}\right)^2 + d\left(\frac{n}{2}\right)\right) + n$$

$$= cm^2 + 2dm + n \leq cm^2 + dm$$

onde a última desigualdade vale se $d \geq 2d+1$, ou seja,
 se $d \leq -1$.

Vamos tomar $d=-1$, o que nos faz ter $c \geq 2$.

mostremos então que $T(n) \leq 2n^2 - n$. Como $2n^2 - n$
 é $O(n^2)$, mostremos que $T(n) = O(n^2)$.

$$(e) T(n) = T(n-1) + T(n-2) + 3 \quad (\text{substituição, suponha } T(n) = O(2^n))$$

Vamos mostrar que $T(n) \leq c \cdot 2^n$ para alguma constante c .

BASE: $n=1$. Aqui, $T(1)=1$ por definição. Teremos $1 \leq c \cdot 2^1$ se escolhermos $c \geq \frac{1}{2}$.

HIPÓTESE: $T(k) \leq c \cdot 2^k$ para $k < n$.

PASSO: Seja $T(n) = T(n-1) + T(n-2) + 3$. Por hipótese, temos $T(n-1) \leq c \cdot 2^{n-1}$ e $T(n-2) \leq c \cdot 2^{n-2}$. Então

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 3 \\ &\leq c \cdot 2^{n-1} + c \cdot 2^{n-2} + 3 \\ &= c \cdot \frac{1}{2} \cdot 2^n + c \cdot \frac{1}{4} \cdot 2^n + 3 \\ &= \left(\frac{c}{2} + \frac{c}{4}\right)2^n + 3 \\ &\leq c \cdot 2^n + 3 \quad (\text{pois } \frac{c}{2} + \frac{c}{4} \leq c) \end{aligned}$$

follow!

Vamos mostrar que $T(n) \leq c \cdot 2^n + d$ para constantes c e d .

BASE: $n=1$. Teremos $1 \leq c \cdot 2^1 + d$ se escolhermos $c \geq \frac{1-d}{2}$.

HIPÓTESE: $T(k) \leq c \cdot 2^k + d$ para $k < n$.

PASSO: Seja $T(n) = T(n-1) + T(n-2) + 3$. Sómos, por hipótese, que $T(n-1) \leq c \cdot 2^{n-1} + d$ e que $T(n-2) \leq c \cdot 2^{n-2} + d$. Então,

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 3 \\ &\leq c \cdot 2^{n-1} + d + c \cdot 2^{n-2} + d + 3 \\ &= \left(\frac{c}{2} + \frac{c}{4}\right)2^n + 2d + 3 \\ &\leq c \cdot 2^n + 2d + 3 \leq c \cdot 2^n + d \end{aligned}$$

onde a última desigualdade vale se $d \geq 2d + 3$, ou seja, se $d \leq -3$. Vamos tomar $d = -3$. Assim, para a base valer basta escolher $c \geq 2$.

mostremos então que $T(n) \leq 2 \cdot 2^n - 3$. Como $2 \cdot 2^n - 3$ é $O(2^n)$, mostremos que $T(n) = O(2^n)$.

$$(f) T(n) = 4T\left(\frac{n}{2}\right) + \sqrt{n} \quad (\text{árvore de recursão e substituição})$$

Vamos usar a árvore para fazer um palpite e depois usar substituição para comprovar-lo.

NÍVEL	TAMANHO PROBLEMAS	QUANTIDADE PROBLEMAS	TRABALHO FEITO
0	n	1	\sqrt{n}
1	$\frac{n}{2}$	4	$4 \times \sqrt{\frac{n}{2}}$
2	$\frac{n}{4}$	16	$16 \times \sqrt{\frac{n}{4}}$
...
i	$\frac{n}{2^i}$	4^i	$4^i \times \sqrt{\frac{n}{2^i}}$
...
l	$\frac{n}{2^l} = 1$	4^l	$4^l \times 1$

Como $\frac{n}{2^l} = 1$, temos que $l = \log n$. Somando o trabalho feito em todos os níveis temos o tempo total gasto:

$$\sum_{i=0}^{\log n} 4^i \sqrt{\frac{n}{2^i}} = \sqrt{n} \sum_{i=0}^{\log n} \left(\frac{4}{\sqrt{2}}\right)^i$$

$$(\text{pois } \sqrt{2^i} = (2^i)^{\frac{1}{2}} = (2^{\frac{1}{2}})^i)$$

$$\begin{aligned} \sqrt{n} \sum_{i=0}^{\log n} \left(\frac{4}{\sqrt{2}}\right)^i &= \sqrt{n} \left(\frac{\left(\frac{4}{\sqrt{2}}\right)^{\log n + 1} - 1}{\frac{4}{\sqrt{2}} - 1} \right) = \sqrt{n} \left(\frac{\left(\frac{4}{\sqrt{2}}\right) \left(\frac{4}{\sqrt{2}}\right)^{\log n} - 1}{\frac{4}{\sqrt{2}} - 1} \right) \\ &= \sqrt{n} \left(\frac{\frac{4}{\sqrt{2}} \cdot n^{\log \frac{4}{\sqrt{2}}} - 1}{\frac{4}{\sqrt{2}} - 1} \right) = \sqrt{n} \left(\frac{\frac{4}{\sqrt{2}} \cdot n^{\log 4 - \log \sqrt{2}} - 1}{\frac{4}{\sqrt{2}} - 1} \right) \\ &= n^{\frac{1}{2}} \left(\frac{\frac{4}{\sqrt{2}} m^{2 - \frac{1}{2}} - 1}{\frac{4}{\sqrt{2}} - 1} \right) = \frac{\frac{4}{\sqrt{2}} m^2}{\frac{4}{\sqrt{2}} - 1} - \frac{m^{\frac{1}{2}}}{\frac{4}{\sqrt{2}} - 1} \leq m^2 \end{aligned}$$

nosso palpite então é que $T(n) = O(n^2)$.

Vamos então provar por substituição que $T(n) \leq cn^2$.

BASE: $n=1$. Como $T(1)=1$, $1 \leq c \cdot 1^2$ se $c \geq 1$.

HIPÓTESE: $T(k) \leq ck^2$ para $k \leq n$.

PASSO: Temos $T(n) = 4T\left(\frac{n}{2}\right) + \sqrt{n}$ e sabemos, por hipótese, que $T\left(\frac{n}{2}\right) \leq c \cdot \left(\frac{n}{2}\right)^2$. Então

$$\begin{aligned}
 T(n) &= 4T\left(\frac{n}{2}\right) + \sqrt{n} \\
 &\leq 4c\left(\frac{n}{2}\right)^2 + \sqrt{n} \\
 &= cn^2 + \sqrt{n} \quad \text{folhou.}
 \end{aligned}$$

Vamos mostrar que $T(n) \leq cn^2 + d\sqrt{n}$ para constantes c e d .

BASE: $n=1$. $1 \leq c \cdot 1^2 + d\sqrt{1}$ se $c \geq 1-d$.

HIPÓTESE: $T(k) \leq ck^2 + d\sqrt{k}$ para $k < n$.

PASSO: Temos $T(n) = 4T\left(\frac{n}{2}\right) + \sqrt{n}$ e sabemos, por hipótese, que $T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 + d\sqrt{\frac{n}{2}}$. Então

$$\begin{aligned}
 T(n) &= 4T\left(\frac{n}{2}\right) + \sqrt{n} \\
 &\leq 4\left(c\left(\frac{n}{2}\right)^2 + d\sqrt{\frac{n}{2}}\right) + \sqrt{n} \\
 &= cn^2 + \frac{4}{\sqrt{2}}d\sqrt{n} + \sqrt{n} \\
 &= cn^2 + \left(\frac{4}{\sqrt{2}}d+1\right)\sqrt{n} \leq cn^2 + d\sqrt{n}
 \end{aligned}$$

onde a última inequação vale se $d \geq \frac{4}{\sqrt{2}}d+1$, ou seja, se $d \leq -\frac{\sqrt{2}}{4-\sqrt{2}} \approx -0,546918$.

Vamos tomar $d=-2$, o que nos faz ter $c \geq 3$.

mostraremos então que $T(n) \leq 3n^2 - 2\sqrt{n}$. Como $3n^2 - 2\sqrt{n}$ é $O(n^2)$, mostraremos que $T(n) = O(n^2)$.

(g) $T(n) = 7(T(\frac{m}{3})) + m^2$ (árvore de recursão e Teorema mestre)

NÍVEL	TAMANHO PROBLEMA	QUANTIDADE PROBLEMAS	TEMPO NO NÍVEL
0	m	1	m^2
1	$\frac{m}{3}$	7	$7 \times (\frac{m}{3})^2$
2	$\frac{m}{9}$	49	$49 \times (\frac{m}{9})^2$
:	:	:	:
i	$\frac{m}{3^i}$	7^i	$7^i \times (\frac{m}{3^i})^2$

Somando o tempo gasto em todos os níveis (temos $\log_3 n + 1$ níveis ao todo):

$$\sum_{i=0}^{\log_3 m} 7^i \left(\frac{m}{3^i}\right)^2 = m^2 \sum_{i=0}^{\log_3 m} \left(\frac{7}{9}\right)^i = m^2 \left(\frac{1 - \left(\frac{7}{9}\right)^{\log_3 m}}{1 - \frac{7}{9}} \right)$$

$$= \frac{9}{2} m^2 \left(1 - m^{\log_3 \frac{7}{9}}\right) = \frac{9}{2} m^2 - \frac{9}{2} m^{2 + \log_3 \frac{7}{9}} \leq 5m^2$$

Pelo método da substituição temos que $T(n) = O(n^2)$.

Pelo método mestre, temos $a=7$, $b=3$ e $f(n)=n^2$.

Como $n^2 = m^{\log_3 9}$ e $m^{\log_3 9} = m^{\log_3 7 + \varepsilon}$, estamos no caso (3) do teorema. Logo, $T(n) = \Theta(f(n)) = \Theta(n^2)$.

(h) $T(n) = T\left(\frac{m}{2}\right) + T\left(\frac{m}{4}\right) + m$ (árvore de recursão e substituição)

NÍVEL	TAMANHO PROBLEMAS	QUANT. PROB.	TEMPO GASTO
0	m	1	m
1	$\frac{m}{2}$ a $\frac{m}{4}$	2	$\frac{m}{2} + \frac{m}{4} = \frac{3}{4}m$
2	$\frac{m}{4}$ a $\frac{m}{16}$	4	$\frac{m}{4} + 2 \cdot \frac{m}{8} + \frac{m}{16} = \frac{9}{16}m$
3	$\frac{m}{8}$ a $\frac{m}{64}$	8	$\frac{m}{8} + 3 \cdot \frac{m}{16} + 3 \cdot \frac{m}{32} + \frac{m}{64} = \frac{27}{64}m$
:	:	:	:
i	$\frac{m}{2^i}$ a $\frac{m}{4^i}$	2^i	$(\frac{3}{4})^i m$
$\log n$			
$\log_4 n$			

Veja que o tempo gasto por nível é $(\frac{3}{4})^i m$ em geral, exceto a partir de um certo ponto, quando a árvore fica desbalanceada. O que podemos fazer é superar que a árvore estaria balanceada para poder fazer os cálculos limitando o tempo de execução:

$$\sum_{i=0}^{\log n} (\frac{3}{4})^i m \leq T(n) \leq \sum_{i=0}^{\log_4 n} (\frac{3}{4})^i m$$

que, resolvendo o somatório, nos diz que

$$4m - 4m^{1+\log_{3/4} 1} \leq T(n) \leq 4m - 4m^{1+\log_{3/4} 3/4} \leq 4m$$

(Note que $\log_{3/4}$ e $\log_{3/4} 3/4$ são números negativos)

Então pela árvore temos que $T(n) = O(n)$. como palpite.

Vamos provar por indução que $T(n) \leq cn$ para uma const. c.

BASE: $n=1$. $T(1)=1 \leq c \cdot 1$ se $c \geq 1$.

HIPÓTESE: $T(k) \leq ck$ para $k < n$.

PASSO: como $T\left(\frac{m}{2}\right) \leq c \frac{m}{2}$ e $T\left(\frac{m}{4}\right) \leq c \frac{m}{4}$ pela hipótese, temos

$$\begin{aligned} T(n) &= T\left(\frac{m}{2}\right) + T\left(\frac{m}{4}\right) + m \\ &\leq c \frac{m}{2} + c \frac{m}{4} + m = \left(\frac{c}{2} + \frac{c}{4} + 1\right)m \leq cm \end{aligned}$$

Onde a última inequação vale se $c \geq \frac{c}{2} + \frac{c}{4} + 1$, o que é verdade quando $c \geq 4$. Escolhemos então $c=10$ (assim a base vale).

Logo, $T(n) = O(n)$ de fato.

$$(i) T(n) = 64T\left(\frac{n}{8}\right) + 7n^3 \quad (\text{teorema mestre})$$

Temos $a=64$, $b=8$ e $f(n)=7n^3$.

Como $\log_8 64 = 2$ e $7n^3 = \Omega(n^{2+\varepsilon})$ com $\varepsilon=1$, estarmos no caso (3) do Teorema e, assim, $T(n)=\Theta(n^3)$.

$$(j) T(n) = 4T\left(\frac{n}{8}\right) + \sqrt{n} \quad (\text{teorema mestre})$$

Temos $a=4$ e $b=8$ e $f(n)=n^{1/2}$

Como $\log_8 4 \approx 0.667$ e $n^{1/2} = O(n^{0.667 - 0.167})$, estarmos no caso (1) do Teorema e, assim, $T(n)=\Theta(n^{\log_8 4})$.

$$(k) T(n) = T(\sqrt{n}) + 1$$

Seja $m = \log n$, o que implica em $n = 2^m$. Assim,

$$T(2^m) = T(2^{m/2}) + 1.$$

Defina $S(m) = T(2^m)$. Assim,

$$S(m) = S\left(\frac{m}{2}\right) + 1.$$

Pelo Teorema mestre, como $a=1$, $b=2$ e $f(m)=1=m^0$, e
como $m^0 = \Theta(m^{\log_2 1})$, vale o caso (2) e $S(m) = \Theta(\log m)$.

$$\text{mos ent\~ao } T(n) = \Theta(\log \log n).$$

$$(l) T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

$$\text{Como } T(n) \leq 2T\left(\frac{n}{2}\right) + n^2 \text{ e } 2T\left(\frac{n}{2}\right) + n^2 \text{ \'e } \Theta(n^2)$$

pelo Teorema mestre, temos que $T(n) = O(n^2)$.

PS: esse resultado n\~ao \'e "aperto". \'E poss\'ivel mostrar que
 $T(n) = \Theta(n \log^2 n)$.

11) BUSCA BINARIA (A, ini, fim, K)

se fim \leq ini

 retorne fim

$$\text{meio} = \lfloor (\text{fim} + \text{ini})/2 \rfloor$$

se $A[\text{meio}] == K$

 retorne meio

senão se $K < A[\text{meio}]$

 BUSCA BINARIA (A, ini, meio-1, K)

senão

 BUSCA BINARIA (A, meio+1, fim, K)

Seja $T(n)$ o tempo de execução do algoritmo acima sobre um vetor com n elementos. Claramente, $T(n) = T(\frac{n}{2}) + 1$.

Usando o método de iteração:

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$= T\left(\frac{n}{4}\right) + 1 + 1 = T\left(\frac{n}{4}\right) + 2$$

$$= T\left(\frac{n}{8}\right) + 1 + 2 = T\left(\frac{n}{8}\right) + 3$$

$$= \dots = T\left(\frac{n}{2^i}\right) + i$$

Como $\frac{n}{2^i} = 1$ quando $i = \log n$, temos que

$$T(n) = T(1) + \log n = \log n + 1.$$

Claramente, $T(n) = \Theta(\log n)$.

12) Solução no livro dos professores.

13) PROCURA_SOMA (A, m, k)

1 ORDENA (A, n)

2 para $i = 1$ até n

3 $j = \text{BUSCABINARIA} (A, 1, m, k - A[i])$

4 se $j > 0$

5 retorne (i, j)

6 retorne $(-1, -1)$

O algoritmo usa o fato de que se existirem posições i e j no vetor tais que $A[i] + A[j] = k$, então $A[j] = k - A[i]$, de modo que o vetor contém os elementos $A[i]$ e $k - A[i]$. Assim, a ideia é que para cada elemento $A[i]$ procuremos no vetor por $k - A[i]$ e, se este existir, encontramos os índices.

Para garantir bom tempo de execução podemos utilizar um bom algoritmo de busca (já que a operação de busca pode ser repetida uma vez para cada elemento). Por isso optamos por utilizar a busca binária, que pelo exercício 11 executa em tempo $\Theta(\log n)$. Para isso, o algoritmo precisa ordenar o vetor inicialmente (na linha 1 chamamos algum algoritmo de ordenação que garante tempo $\Theta(n \log n)$ no pior caso, como por exemplo o mergeSort).

Assim, o tempo total de execução é dado pelo tempo de ordenar o vetor somado à n execuções da busca binária, isto é, $\Theta(n \log n)$.

O algoritmo dado, portanto, tem tempo $\Theta(n \log n)$.

14) —