Universidade Federal da Paraíba
Centro de Informática
Disciplina: Análise e Projeto de Algoritmos
Professor: Bruno Bruck

Lista de exercícios (Unidade 1) - Respostas

Exercício 1

a) Não. O fato de A_2 ter uma complexidade assintótica melhor somente nos indica que tal algoritmo tende a ser mais eficiente para valores de entrada grandes o suficiente. O próprio conceito de complexidade assintótica não fornece garantias de que A_2 será mais eficiente para todos os valores de entrada. No caso dos algoritmos em questão, se assumirmos, por exemplo, $n = 2$, temos que $T_{A_1}(2) = 3$ e $T_{A_2}(2) = 16$, o que na verdade confirma que nem sempre A_2 é mais eficiente que A_1.

b) Sim. Se tivéssemos como plotar $T_{A_1}(n)$ e $T_{A_2}(n)$ veríamos claramente que, para valores de entrada menores ou iguais a 33, o algoritmo A_1 executa um número menor de operações básicas em comparação ao algoritmo A_2, sendo, portanto, mais eficiente nesses casos. Entretanto, mesmo sem plotar as duas equações, bastaria demonstrar, por exemplo, que para os valores $n = 1$ e $n = 2$ o mesmo comportamento pode ser observado.

c) O argumento aqui é basicamente o mesmo dos itens anteriores. O fato de A_3 ser assintoticamente melhor não garante que será sempre mais eficiente. Além disso, quando $n = 2$, $T_{A_3}(2) = 2000$, o que já demonstra que nem sempre A_3 é mais eficiente.

Exercício 2

a) Primeiramente, é importante notar que o laço da linha 1 ("repita") é executado sempre que o laço mais interno, da linha 4, fizer ao menos uma troca. Sendo assim, um primeiro ponto seria considerar como maximizar o número de execuções desse laço, o que é garantido caso o menor elemento esteja na última posição do vetor. Nesse cenário, a cada execução do laço da linha 1, o menor elemento seria movido uma posição para a esquerda, necessitando de $n - 1$ iterações para finalmente chegar à posição 0 do vetor. Entretanto, não podemos nos esquecer do teste executado na linha 5, que também depende da organização dos elementos no vetor. Devemos então, considerar também como maximizar o número de vezes em que o teste é bem sucedido e as linhas 6 e 7 também são executadas. Para isso, não basta garantir somente que o menor elemento esteja na última posição do vetor, mas também que o vetor esteja ordenado em ordem não-crescente. Sendo assim, o pior caso do algoritmo é encontrado quando o vetor de entrada está ordenado ao contrário.

Antes de determinar a função $T(n)$, temos antes que fazer algumas observações. A primeira delas é que a linha 6 do algoritmo é na verdade uma chamada de função. Portanto, primeiramente precisamos saber quantas operações básicas são executadas a cada chamada dessa função. Uma implementação bastante simples, e que será utilizada em nossa análise, é a seguinte:

```
Função Troca(ref_elem1, ref_elem2)
    temp = ref_elem1;
    ref_elem1 = ref_elem2;
    ref_elem2 = temp;
```

Uma simples análise por contagem de passos nos informa que essa função consome um total de 3 operações básicas, dentre acessos a elementos do vetor e atribuições.

Outra observação é que, no pior caso, a cada iteração do laço da linha 1, exatamente 1 elemento será movido para sua posição final. Sendo assim, a cada iteração desse laço, o teste da linha 5 falhará uma vez a mais que na iteração anterior. Nesse sentido, é possível notar que o número de execuções das linhas 6 e 7 segue uma progressão aritmética com $n-1$ elementos, razão -1 e $a_1 = n-1$. Sendo assim, podemos inferir que as linhas 6 e 7 serão executadas um total de $\frac{(n-1)(n-1+1)}{2} = \frac{n^2-n}{2}$ vezes.

A análise do algoritmo por contagem de passos no pior caso fica é, então, como a seguir:

	# op.	# exec.
repita	0	$n-1$
houveTroca = Falso;	1	$n-1$
para $i = 0$ até Tamanho(entrada) - 1 **faça**	2	$n-1$
se entrada[i] > entrada[i+1] **então**	4	$n-1$
Troca(entrada[i], entrada[i+1]);	6	$\frac{n^2-n}{2}$
houveTroca = Verdadeiro;	1	$\frac{n^2-n}{2}$
fim;		
fim;		
até houveTroca;	1	$n-1$

Sendo assim, temos que:

$$T(n) = (n-1) + 2(n-1) + 4(n-1) + 6(\frac{n^2-n}{2}) + \frac{n^2-n}{2} + (n-1)$$

$$T(n) = \frac{7n^2+9n}{2} - 8$$

b) Considerando o que já foi analisado no item anterior, no melhor caso, queremos minimizar tanto o número de repetições do laço da linha 1 quanto o número de vezes em que o teste da linha 6 é bem sucedido. Note que, caso o vetor já esteja ordenado, o teste da linha 6 sempre irá falhar e, portanto, nenhuma troca será feita. Como consequência, o laço da linha 1 será executado uma única vez. Essa situação configura o melhor caso do algoritmo.

Com relação à análise de contagem de passos, esta pode ser feita como a seguir:

	# op.	# exec.
repita	0	1
houveTroca = Falso;	1	1
para $i = 0$ até Tamanho(entrada) - 1 **faça**	2	$n-1$
se entrada[i] > entrada[i+1] **então**	4	$n-1$
Troca(entrada[i], entrada[i+1]);	6	0
houveTroca = Verdadeiro;	1	0
fim;		
fim;		
até houveTroca;	1	1

Sendo assim, temos que:

$$T(n) = 1 + 2(n-1) + 4(n-1) + 1$$
$$T(n) = 6n - 4$$

c) No pior caso temos $T(n) = \frac{7n^2+9n}{2} - 8$. Sendo assim, um possível limite assintótico superior é $O(n^2)$. Para demonstrar que tal limite é válido, podemos analisar o resultado de $\lim_{n\to\infty} \frac{T(n)}{g(n)}$). De acordo com nossa hipótese de limite assintótico, sabemos que $g(n) = n^2$, portanto:

$$\lim_{n\to\infty} \frac{\frac{7n^2+9n}{2} - 8}{n^2} = d$$

, onde d é uma constante positiva. Com esse resultado, podemos afirmar que $T(n)$ e $g(n)$ possuem a mesma taxa de crescimento. Consequentemente é possível afirmar que $T(n) = O(n^2)$.

d) No melhor caso temos $T(n) = 6n - 4$, sugerindo que $T(n) = \Theta(n)$. Para provar esse limite podemos analisar o resultado de $\lim_{n\to\infty} \frac{T(n)}{g(n)}$). Ao fazer isso temos que

$$\lim_{n\to\infty} \frac{6n-4}{n} = d$$

, onde d é uma constante positiva. Esse resultado indica que a função $g(n) = n$ tem a mesma taxa de crescimento que $T(n)$. Consequentemente, podemos concluir que de fato $T(n) = \Theta(n)$ é um limite válido para o algoritmo no melhor caso.

e) Sim. Podemos demonstrar isso observando que $\lim_{n\to\infty} \frac{T(n)}{1} = \infty$ tanto no pior quanto no melhor caso. Além disso, sabemos que $\Omega(1)$ é o limite inferior mais baixo que podemos fornecer e não existem curvas de complexidade mais eficientes que esta. Portanto, o limite $\Omega(1)$ não só é válido para o Bubble Sort para qualquer tipo de entrada, como também é válido para qualquer algoritmo.

Exercício 3

a) A afirmação $f(n) = \Omega(g(n))$ e $g(n) = O(f(n))$ está correta. É possível demonstrar que $f(n) = \Omega(g(n))$, pois sabemos que $\lim_{n\to\infty} \frac{n^3}{n \log n} = \infty$. Isso indica que a função $g(n)$, de fato, cresce assintoticamente mais lentamente do que $f(n)$ e, portanto, define um limite assintótico inferior para $f(n)$. De modo similar, podemos afirmar que $g(n) = O(f(n))$, pois $\lim_{x\to\infty} \frac{n \log n}{n^3} = 0$, o que indica que a função $f(n)$ cresce assintoticamente mais rapidamente que $g(n)$.

b) Para que o limite $g(n) = \Omega(f(n))$, fosse válido, o resultado de $\lim_{n\to\infty} \frac{g(n)}{f(n)}$ deveria ser igual a ∞ ou a uma constante $d > 0$, indicando, respectivamente, que a função $g(n)$ cresce assintoticamente mais devagar que $f(n)$, ou que elas tem a mesma taxa de crescimento. Entretanto, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n \log n}{n^3} = 0$, o que inviabiliza o limite proposto. Já o segundo limite da afirmação, isto é, $h(n) = \Omega(g(n))$, é válido, pois

$\lim_{n\to\infty} \frac{n^2}{n \log n} = \infty$. De qualquer forma, a afirmação como um todo está incorreta por conta do primeiro limite, que é inválido.

c) O limite $g(n) = O(p(n))$ é válido pois $\lim_{n\to\infty} \frac{g(n)}{p(n)} = \lim_{n\to\infty} \frac{n \log n}{n!} = 0$, indicando que $p(n)$ cresce mais rapidamente que $g(n)$. Entretanto, o limite $f(n) = O(k(n))$ está incorreto, pois $\lim_{n\to\infty} \frac{f(n)}{k(n)} = \lim_{n\to\infty} \frac{n^3}{3n^2+2n} = \infty$, de onde concluímos que a função $k(n)$ não cresce mais rapidamente que a função $f(n)$, inviabilizando tal limite. Portanto, a afirmação está incorreta.

d) A afirmação está incorreta, pois o limite $k(n) = \Omega(f(n))$ é inválido. Isso pode ser comprovado observando que $\lim_{n\to\infty} \frac{k(n)}{f(n)} = \lim_{n\to\infty} \frac{3n^2+2n}{n^3} = 0$, indicando que a função $f(n)$ na verdade possui taxa de crescimento superior a $k(n)$. O segundo limite, $p(n) = O(g(n))$, também é inválido, visto que $\lim_{n\to\infty} \frac{p(n)}{g(n)} = \lim_{n\to\infty} \frac{n!}{n \log n} = \infty$. Tal resultado especifica que a função $p(n)$ possui taxa de crescimento superior a $g(n)$, inviabilizando o limite proposto.

e) O limite $h(n) = O(f(n))$ está correto, pois $\lim_{n\to\infty} \frac{h(n)}{f(n)} = \lim_{n\to\infty} \frac{n^2}{n^3} = 0$, indicando que a função $f(n)$ de fato cresce assintoticamente mais rápido do que $h(n)$. Entretanto, o limite $k(n) = O(g(n))$ é inválido, pois $\lim_{n\to\infty} \frac{k(n)}{g(n)} = \lim_{n\to\infty} \frac{3n^2+2n}{n \log n} = \infty$, o que demonstra que $g(n)$ não é um limite assintótico superior para $k(n)$.

Exercício 4

a) Sabendo que o pior caso do algoritmo é $O(n^2)$ podemos inferir que tal limite é válido para qualquer entrada. Uma conclusão parecida poderia ser feita para seu limite $\Omega(n)$ no melhor caso. Entretanto, a única forma de poder especificar que o algoritmo é $\Theta(n^2)$ para qualquer tipo de entrada seria caso também tivéssemos provado que o algoritmo é $\Omega(n^2)$ no melhor caso, o que não sabemos se é verdade. Sendo assim, a afirmativa está incorreta. Não podemos comprovar o limite $\Theta(n^2)$ somente com base nas informações fornecidas.

b) A afirmação está correta. Podemos demonstrar sua veracidade observando que $\lim_{n\to\infty} \frac{2n^2+n}{n^3} = 0$, o que indica que a função $g(n) = n^3$ possui taxa de crescimento assintoticamente superior a $T(n)$, o que é consistente com a definição do limite O.

c) A afirmação está correta. No pior caso estamos interessados em analisar o cenário em que o algoritmo consome a maior quantidade possível do recurso computacional que está sendo considerado. Sendo assim, por definição, não deve existir um cenário pior que esse. Portanto, ao definir um limite superior para o pior caso de um algoritmo, no caso $O(n^2)$, este limite automaticamente passa a ser válido para qualquer entrada do algoritmo. **Observação:** Note que o mesmo não poderia ser dito com certeza caso o limite estabelecido para o pior caso utilizasse a notação Ω.

d) A afirmação está incorreta. Na verdade, existem dois problemas com tal afirmação. O primeiro deles é quando especificamos que $f(n)$ e $g(n)$ podem ser quaisquer funções. As notações assintóticas exigem que tanto a função $f(n)$ quanto a $g(n)$ sejam assintoticamente positivas. Outro problema com a afirmação é quando especificamos que o limite é válido para **algum** valor de $n \geq n_0$. Pela definição de qualquer um dos três tipos de limites assintóticos, temos que garantir que tal limite seja válido para **todo** valor de $n \geq n_0$.

Exercício 5

Observando o comportamento da função:

$$\begin{aligned} T(n) &= T(n-1) + n \\ T(n-1) &= T(n-2) + n - 1 \\ T(n-2) &= T(n-3) + n - 2 \\ T(n-3) &= T(n-4) + n - 3 \end{aligned}$$

Expandindo a recorrência:

$$\begin{aligned} T(n) &= T(n-1) + n && \textit{passo } 1 \\ &= (T(n-2) + n - 1) + n \\ &= T(n-2) + (n-1) + n && \textit{passo } 2 \\ &= (T(n-3) + n - 2) + (n-1) + n \\ &= T(n-3) + (n-2) + (n-1) + n && \textit{passo } 3 \\ &= (T(n-4) + n - 3) + (n-2) + (n-1) + n \\ &= T(n-4) + (n-3) + (n-2) + (n-1) + n && \textit{passo } 4 \end{aligned}$$

Após k passos, temos:

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

No caso base, temos que:

$$\begin{aligned} n - k &= 0 \\ k &= n \end{aligned}$$

Portanto:

$$T(n) = T(n-n) + \sum_{i=0}^{n-1} (n-i) \qquad \text{(obs: note que esse somatório é uma PA)}$$

$$T(n) = \cancelto{1}{T(0)} + \frac{n(n+1)}{2}$$

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

Exercício 6

a) O limite $O(n^2)$ é válido para a função $T(n)$ do exercício anterior. Isso pode ser demonstrado pelo fato de que

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{2} + \frac{n}{2} + 1}{n^2} = d,$$

onde d é uma constante positiva maior que 0 e diferente de ∞. Com o resultado desse limite podemos concluir que ambas as funções, $T(n)$ e $g(n)$, possuem a mesma taxa de crescimento. Visto que pela definição da notação assintótica O (Big-O) é permitido que a função $g(n)$ tenha a mesma taxa de crescimento de $T(n)$, o limite proposto é válido.

b) Podemos afirmar que $T(n) = \Omega(n^2)$ é um limite válido, pois

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{2} + \frac{n}{2} + 1}{n^2} = d,$$

onde $d > 0$ e $d \neq \infty$. Tal resultado implica que $T(n)$ e $g(n)$ possuem exatamente a mesma taxa de crescimento. Visto que pela definição da notação assintótica Ω é permitido que a função $g(n)$ tenha a mesma taxa de crescimento de $T(n)$, o limite proposto é válido.

c) Sabemos que uma das formas de demonstrar um limite assintoticamente restrito (i.e., um limite Θ) é escolhendo uma função assintoticamente positiva $g(n)$ e demonstrando separadamente que $T(n) = O(g(n))$ e que $T(n) = \Omega(g(n))$. Nos itens (a) e (b) deste exercício demonstramos que $T(n) = O(n^2)$ e $T(n) = \Omega(n^2)$. Sendo assim, utilizando $g(n) = n^2$ podemos concluir que $T(n) = \Theta(n^2)$.

Observação: Uma forma alternativa de responder a este exercício seria demonstrar que $\lim_{n\to\infty} \frac{T(n)}{g(n)} = d$, onde $d > 0$ e $d \neq \infty$, e utilizar tal resultado como base do argumento para demonstrar que n^2 possui a mesma taxa de crescimento que $T(n)$.

d) Podemos afirmar que $T(n) = O(n^4)$ é um limite válido, pois

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{2} + \frac{n}{2} + 1}{n^4} = 0.$$

Tal resultado indica que a função $g(n)$ cresce mais rapidamente que $T(n)$ e, portanto, pode ser utilizada como um limite assintótico superior.

e) Podemos afirmar que $T(n) = \Omega(n)$ é um limite válido, pois

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{2} + \frac{n}{2} + 1}{n} = \infty.$$

Isso indica que a função $T(n)$ possui taxa de crescimento superior à função $g(n)$. Portanto, $g(n)$ pode ser utilizada para definir um limite assintótico inferior. Entretanto, é importante notar que esse limite, apesar de válido, não é justo, visto que $T(n) = \Omega(n^2)$ também é um limite válido, como demonstrado no item (b).

Exercício 7

a) Durante sua execução, o algoritmo CountingSort faz uso de um vetor auxiliar contendo k elementos. A escolha do valor de k deve ser feita de forma a garantir que nenhum elemento do vetor a ser ordenado seja maior que k. Visto que o algoritmo itera sobre cada um dos elementos do vetor auxiliar duas vezes, o valor dessa constante acaba sendo importante na avaliação da complexidade do algoritmo. Esse é um dos motivos pelo qual a complexidade do CountingSort é $\Theta(n + k)$ e explica porque o algoritmo é pseudolinear. Em casos onde k não é assintoticamente maior que n, o algoritmo terá tempo de execução linear em função do tamanho da entrada. Entretanto, é possível que k seja muito maior que n (por exemplo, $k = n^2$), o que acaba impactando significativamente o tempo de execução do algoritmo a ponto de se comportar como um algoritmo de complexidade maior.

b) Um algoritmo de ordenação é estável se após a ordenação elementos de mesmo valor permanecem na mesma ordem que no vetor original.

O CountingSort é um algoritmo estável e o último passo do algoritmo é crucial para entendermos o porquê. Esse passo é responsável por iterar sobre o vetor A utilizando-se o vetor auxiliar C para descobrir a posição correta de cada elemento no vetor ordenado B. Caso existam elementos de mesmo valor em A, o que determina se o algoritmo será estável ou não é a ordem em que os elementos de A são analisados. No CoutingSort, essa análise é feita da direita para a esquerda justamente para garantir que elementos de mesmo valor mantenham a mesma ordem no vetor B. Isso é garantido pois, todas as vezes que adicionamos um elemento de valor x no vetor B, decrementamos o valor de $C[x]$, de forma que o próximo elemento de valor x será certamente adicionado em uma posição de índice inferior.

c)

Exercício 9

a) Para utilizarmos o Teorema Mestre, primeiramente é necessário que a recorrência possua o seguinte formato:

$$T(n) = aT(\frac{n}{b}) + f(n),$$

e que a função $f(n)$ possa ser reescrita como $\Theta(n^d)$. No caso da recorrência deste exercício, temos que $a = 2$, $b = 3$ e $d = 1$. Temos que $\log_b a = \log_3 2 < 1$, portanto $d > \log_b a$, o que se encaixa no caso 1 do Teorema Mestre, que trata do cenário onde o custo dominante aparece no primeiro nível da árvore de subproblemas. Neste cenário a complexidade do algoritmo é dada por $\Theta(n^d)$, ou seja $\Theta(n)$.

Como utilizamos o caso 1, é desejável demonstrarmos também que $f(n)$ é polinomialmente maior que $n^{log_b a}$ a fim de demonstrarmos que o limite encontrado é de fato correto. Para isso basta demonstrar que $\frac{f(n)}{n^{log_b a}} = n^{\epsilon}$, onde $\epsilon > 0$. No caso temos que:

$$\frac{f(n)}{n^{log_b a}} = \frac{n}{n^{log_3 2}} = n^{1-log_3 2}$$

Portanto, visto que $\epsilon = 1 - log_3 2 > 0$, sabemos que o limite encontrado é válido.

b)