MAC5711 ANÁLISE DE ALGORITMOS: LISTA 1

Entregar: Exercícios 1.2 (apenas itens (a) e (b)), 1.3 (apenas item (e)), 2.4 e 3.7.
Prazo: 16/maio/2021 às 23h59
Instruções: ATENÇÃO às instruções postadas no e-disciplinas para o formato de entrega.

Os sobrescritos ao lado de cada rótulo de exercício indicam *aproximadamente* a partir de qual aula é recomendável que se trabalhe naquele exercício.

1. Notação assintótica e Recorrências

Exercício 1.1.$^{\mathrm{A01}}$ Lembre-se que $\lg n$ denota o logaritmo na base 2 de n. Usando a definição de notação O, prove que

(a) 3^n não é $O(2^n)$;

(b) $\log_{10} n$ é $O(\lg n)$;

(c) $\lg n$ é $O(\log_{10} n)$.

Exercício 1.2.$^{\mathrm{A01}}$ Usando a definição de notação O, prove que

(a) $n^7 - 7n^5 + 10^{\pi/e} n^2 + 5000 = O(n^7)$;

(b) $2\lceil n/5 \rceil = O(n)$;

(c) $n = O(2^n)$;

(d) $n/1000$ não é $O(1)$;

(e) $n^2/2$ não é $O(n)$.

Exercício 1.3.$^{\mathrm{A02}}$ Prove ou dê um contraexemplo para cada uma das afirmações abaixo:

(a) $\lg \sqrt{n} = O(\lg n)$.

(b) Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$.

(c) Se $f(n) = O(g(n))$ e $g(n) = \Theta(h(n))$ então $f(n) = \Theta(h(n))$.

(d) Suponha que $\lg(g(n)) > 0$ e que $f(n) \geq 1$ para todo n suficientemente grande. Nesse caso, se $f(n) = O(g(n))$ então $\lg(f(n)) = O(\lg(g(n)))$.

(e) Se $f(n) = O(g(n))$ então $2^{f(n)} = O(2^{g(n)})$.

Exercício 1.4.$^{\mathrm{A07}}$ Prove que: (a) $\sum_{j=1}^{n} j^k \in \Theta(n^{k+1})$; (b) $\sum_{j=1}^{n} j/2^j \leq 2$.

Exercício 1.5.$^{\mathrm{A04}}$ Seja C uma constante real positiva. Para cada uma das recorrências $T(n)$ abaixo, encontre uma fórmula fechada (não recursiva) — mesmo que não seja válida para todos valores de n — e deduza um limitante em notação $O(\cdot)$ para a função T. Depois disso, *prove* que, pelo menos nos valores de n que você escolheu, $T(n)$ é igual à fórmula que você encontrou. Considere que $T(0) = 1$.

(a) $T(n) = 2T(\lfloor n/2 \rfloor) + Cn^2$;

(b) $T(n) = 8T(\lfloor n/2 \rfloor) + Cn^2$;

Data: 26 de março de 2021 às 20h34 -0300.

(c) $T(n) = 2T(\lfloor n/2 \rfloor) + Cn^3$;

(d) $T(n) = T(\lfloor 9n/10 \rfloor) + Cn$.

Exercício 1.6.[A04] Seja $M(n)$ definida pela recorrência

$$M(n) := \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1, \\ \min\limits_{0 \le k \le n-1} \{M(k) + M(n-1-k)\} + n & \text{se } n \ge 2. \end{cases}$$

Mostre que $M(n) \ge n \log_{10} n$ para todo $n \ge 1$.

2. Ordenação e Seleção por Comparação

Exercício 2.1.[A03] Escreva um algoritmo que ordena uma lista de n itens dividindo-a em três sublistas de aproximadamente $n/3$ itens, ordenando cada sublista recursivamente e intercalando as três sublistas ordenadas. Analise seu algoritmo concluindo qual é o seu consumo de tempo.

Exercício 2.2.[A03] Seja $A[1\,..\,n]$ um vetor de inteiros e i e j dois índices distintos de A, ou seja, i e j são inteiros entre 1 e n. Dizemos que o par (i, j) é uma *inversão* de A se $i < j$ e $A[i] > A[j]$. Escreva um algoritmo $O(n \lg n)$ que devolva o número de inversões em um vetor A, onde n é o número de elementos em A. Você pode assumir que não existem itens repetidos no vetor.

Exercício 2.3.[A07] Descreva um algoritmo que, dados inteiros n e k, juntamente com k listas ordenadas que em conjunto tenham n registros, produza uma única lista ordenada contendo todos os registros dessas listas (isto é, faça uma *intercalação*). O seu algoritmo deve ter complexidade $O(n \lg k)$. Note que isto se transforma em $O(n \lg n)$ no caso de n listas de 1 elemento, e em $O(n)$ se só houver duas listas (no total com n elementos).

Exercício 2.4.[A03] Considere a sequência de vetores $A_k[1\,..\,2^k], A_{k-1}[1\,..\,2^{k-1}], \ldots, A_1[1\,..\,2^1]$, e $A_0[1\,..\,2^0]$. Suponha que cada um dos vetores é crescente. Queremos reunir, por meio de sucessivas operações de intercalação (= *merge*), o conteúdo dos vetores $A_0, \ldots, A_k$ em um único vetor crescente $B[1\,..\,n]$, onde $n = 2^{k+1} - 1$. Escreva um algoritmo que faça isso em $O(n)$ unidades de tempo. Use como subrotina o Intercala visto em aula.

Exercício 2.5.[A07] Suponha que $A[1\,..\,m]$ é um heap. Suponha que $i < j$ e $A[i] < A[j]$. Se os valores de $A[i]$ e $A[j]$ forem trocados, $A[1\,..\,m]$ continuará sendo um heap? Repita o exercício sob a hipótese $A[i] > A[j]$.

Exercício 2.6 (CLRS 6.5-3).[A07] Escreva em pseudocódigo cada um dos seguintes procedimentos, que implementam uma fila de prioridade com um heap de mínimo: Heap-Mínimo, Heap-Extrai-Mínimo, Heap-Diminui-Chave e Heap-Insere.

Exercício 2.7.[A06] Sejam $X[1\,..\,n]$ e $Y[1\,..\,n]$ dois vetores, cada um contendo n números ordenados. Escreva um algoritmo $O(\lg n)$ para encontrar uma das medianas de todos os $2n$ elementos nos vetores X e Y.

Exercício 2.8.[A06] Para esta questão, vamos dizer que a mediana de um vetor $A[p\,..\,r]$ com números inteiros é o valor que ficaria na posição $A[\lfloor (p+r)/2 \rfloor]$ depois que o vetor $A[p\,..\,r]$ fosse ordenado.

Dado um algoritmo linear "caixa-preta" que devolve a mediana de um vetor, descreva um algoritmo simples, linear, que, dado um vetor $A[p\,..\,r]$ de inteiros distintos e um inteiro k, devolve o k-ésimo mínimo do vetor. (O k-ésimo mínimo de um vetor de inteiros distintos é o elemento que estaria na k-ésima posição do vetor se ele fosse ordenado.)

3. Ordenação em tempo linear

Exercício 3.1.A08 Desenhe a árvore de decisão para o SelectionSort aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 3.2 (CLRS 8.1-1)**.**A08 Qual a menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

Exercício 3.3 (CLRS 8.2-1)**.**A08 Simule a execução do CountingSort usando como entrada o vetor

$$A[1\,..\,11] = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\rangle.$$

Exercício 3.4.A08 Escreva uma função que recebe um vetor com n letras A's e B's e, por meio de trocas, move todos os A's para o início do vetor. Sua função deve consumir tempo $O(n)$.

Exercício 3.5 (CLRS 8.2-2)**.**A08 Mostre que o CountingSort é estável.

Exercício 3.6 (CLRS 8.2-3)**.**A08 Suponha que o **para** da linha 7 do CountingSort é substituído por

para $j \leftarrow 1$ **até** n **faça**

Mostre que o CountingSort ainda funciona. O algoritmo resultante continua estável?

Exercício 3.7 (CLRS 8.2-4)**.**A08 Descreva um algoritmo que, dados n inteiros no intervalo de 1 a k, preprocesse sua entrada e então responda em $O(1)$ qualquer consulta sobre quantos dos n inteiros dados caem em um intervalo $[a\,..\,b]$. O preprocessamento efetuado pelo seu algoritmo deve consumir tempo $O(n + k)$.

Exercício 3.8 (CLRS 8.3-2)**.**A08 Quais dos seguintes algoritmos de ordenação são estáveis: insertionsort, mergesort, heapsort e quicksort. Descreva uma maneira simples de deixar qualquer algoritmo de ordenação estável. Quanto tempo e/ou espaço adicional a sua estratégia usa?

Exercício 3.9.A01 Qual a diferença de consumo de tempo entre uma busca binária em um vetor com n elementos e uma busca binária em um vetor com n^2 elementos?

Exercício 3.10 (CLRS 8.3-4)**.**A08 Mostre como ordenar n inteiros no intervalo de 0 até $n^2 - 1$ em tempo $O(n)$.

Exercício 3.11 (CLRS 8.4-1)**.**A08 Simule a execução do BucketSort com o vetor

$$A[1\,..\,10] = \langle 0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42\rangle.$$

Exercício 3.12 (CLRS 8.4-2)**.**A08 Explique por que o consumo de tempo de pior caso para o BucketSort é $\Theta(n^2)$. Que simples ajuste do algoritmo melhora o seu pior caso para $O(n \lg n)$ e mantém o seu consumo esperado de tempo linear.

Exercício 3.13 (CLRS 8.4-3)**.**A05 Seja X uma variável aleatória que é igual ao número de caras em duas jogadas de uma moeda justa. Quanto vale $E[X^2]$? Quanto vale $E[X]^2$?

Apêndice A. Exemplos de soluções

Seguem alguns exercícios resolvidos, para que você tenha uma ideia do nível de detalhe/formalismo adequado para suas soluções.

Note, em particular, que toda **expressão/fórmula matemática aparece dentro de uma frase com sujeito e predicado**. Note ainda que expressões matemáticas aparecem sempre ligadas por conectivos lógicos, e nunca "soltas" no meio do texto (fora de frases ou sem conectivos lógicos).

Solução 1 (Exercício 1.2, item (e))**.** Suponha que $n^2/2$ é $O(n)$ e vamos derivar uma contradição. Por definição, existem constantes positivas c e n_0 tais que,

$$\text{para qualquer inteiro } n \geq n_0, \text{ vale que } n^2/2 \leq cn. \tag{1}$$

Defina[1]

$$m_0 \coloneqq \max\{n_0, 1\} \qquad \text{e} \qquad d \coloneqq \max\{c, 1\},$$

e

de modo que valem

$$m_0 \geq n_0, \tag{2}$$

$$m_0 \geq 1, \tag{3}$$

$$d \geq c, \tag{4}$$

e

$$d \geq 1. \tag{5}$$

Defina

$$n \coloneqq 4dm_0. \tag{6}$$

Temos

$$n \overset{(6)}{=} 4dm_0 \overset{(5)}{\geq} m_0 \overset{(2)}{\geq} n_0.$$

Logo, por (1), vale que

$$8d^2m_0^2 = (4dm_0)^2/2 \overset{(6)}{=} n^2/2 \overset{(1)}{\leq} cn \overset{(4)}{\leq} dn \overset{(6)}{=} 4d^2m_0. \tag{7}$$

Dividindo ambas as extremidades de (7) inequação por $8d^2m_0$, obtemos $m_0 \leq 1/2$, uma contradição, pois $m_0 \geq 1$ por (3).

Concluímos que $n^2/2$ não é $O(n)$.

Solução 2 (Exercício 1.3, item (b))**.** Vamos provar a afirmação. Suponha que $f(n)$ é $O(g(n))$ e que $g(n) = O(h(n))$. Da primeira suposição, obtemos que existem constantes positivas c e n_0 tais que, para todo inteiro $n \geq n_0$, vale que $f(n) \leq c \cdot g(n)$. Da segunda suposição, obtemos que existem constantes positivas d e m_0 tais que, para todo inteiro $n \geq m_0$, vale que $g(n) \leq d \cdot h(n)$.

Defina $N_0 \coloneqq \max\{n_0, m_0\}$ e $C \coloneqq c \cdot d$. Note que N_0 e C são constantes positivas. Seja n um inteiro tal que $n \geq N_0$. Então $n \geq N_0 \geq n_0$ implica que $f(n) \leq c \cdot g(n)$. Similarmente, $n \geq N_0 \geq m_0$ implica que $g(n) \leq d \cdot h(n)$; multiplicando (ambos os lados d)esta última inequação por $c > 0$, obtemos $c \cdot g(n) \leq c \cdot d \cdot h(n) = C \cdot h(n)$. Concluímos que $f(n) \leq c \cdot g(n) \leq C \cdot h(n)$. Como n foi escolhido arbitrariamente tal que $n \geq N_0$, provamos assim que, para todo inteiro $n \geq N_0$, vale que $f(n) \leq C \cdot h(n)$. Em outras palavras, provamos que $f(n)$ é $O(h(n))$.

Solução 3 (Exercício 2.8)**.** Suponha a existência de um algoritmo Mediana(A, p, r) que recebe um vetor $A[1\,..\,n]$ de inteiros e devolve a mediana de $A[p\,..\,r]$ em tempo $O(m)$, onde $m \coloneqq r - p + 1$. O algoritmo A.1 abaixo modifica o algoritmo de seleção aleatorizado, fazendo com que o pivô da rotina Particione sempre seja a mediana do subvetor analisado.

Note que a corretude do algoritmo A.1 segue da corretude do algoritmo aleatorizado visto na aula 5 para seleção do k-ésimo menor elemento.

O consumo de tempo de uma chamada ao algoritmo A.1 com um vetor $A[p\,..\,r]$ de tamanho $n \coloneqq r - p + 1$ é $T(n) \leq T(n/2) + \Theta(n)$. Aqui estamos ignorando detalhes como o piso/teto de $n/2$ na *potencial* chamada recursiva, e levamos em conta que as chamadas às funções Mediana e

[1] Note a distinção entre '=' e ':='; o último indica que o símbolo da esquerda é definido pela expressão à direita, como o operador de atribuição de linguagens de programação.

Algorithm A.1 Algoritmo de seleção via medianas

```
1: function Seleção-por-mediana(A, p, r, k)
2:     if p = r then
3:         return A[p]
4:     x ← Mediana(A, p, r)
5:     Encontre um índice i em p . . r tal que A[i] = x
6:     A[i] ↔ A[r]
7:     q ← Particione(A, p, r)
8:     if k = q − p + 1 then
9:         return A[q]
10:    if k < q − p + 1 then
11:        return Seleção-por-mediana(A, p, q − 1, k)
12:    else
13:        return Seleção-por-mediana(A, q + 1, r, k − (q − p + 1))
```

Particione, bem como a busca pela mediana na linha 5, consomem tempo total $\Theta(n)$. (Como a chamada recursiva é apenas potencial, a recorrência é descrita como '$\leq$' no lugar de '$=$'.)

Finalmente, verificamos que a recorrência $T(n) \leq T(n/2) + n$, com base $T(1) = 1$, satisfaz $T(n) \leq 2n$ sempre que n for uma potência de 2, ou seja, sempre que $n = 2^k$ para algum inteiro $k \geq 0$. Vamos provar por indução em k que $T(2^k) \leq 2^{k+1}$. Para a base da indução, temos $k = 0$ e $T(2^0) = T(1) = 1 \leq 2 = 2^{0+1}$. Seja $k > 0$ um inteiro. Então

$$\begin{aligned} T(2^k) &\leq T(2^k/2) + 2^k && \text{pela recorrência} \\ &= T(2^{k-1}) + 2^k \\ &\leq 2^{(k-1)+1} + 2^k && \text{pela hipótese de indução} \\ &= 2^k + 2^k = 2^{k+1}. \end{aligned}$$

Isso conclui a prova de que $T(n)$ é $O(n)$ (provamos apenas para n potência de 2, mas isso é o suficiente para nossos propósitos nesta disciplina).