MO417A — Complexidade de Algoritmos I

Nome: Nathana Facion **RA:**191079 **Exercícios:** Lista 1

Questão 1. Disserte brevemente (200/400 palavras) sobre algoritmos. O texto deve conter definições de: problema, algoritmo, modelo computacional RAM e complexidade de tempo.

Um **algoritmo** é qualquer procedimento computacional bem definido (que seja claro no que deve-se ser realizado), que recebe um valor como entrada e retorna algum valor ou conjunto de valores como saída.

Um problema é uma determinada questão ou assunto que requer uma solução. Outra forma de ver algoritmo é considerá-lo como uma forma de resolver **problemas. Algoritmos resolvem problemas computacionais** que sejam bem definidos. O algoritmo então descreve uma sequência de passos computacionais para que conseguimos por meio da entrada gerar uma saída.

Para analisar se um algoritmo é bom precisamos considerar um modelo computacional, assim os algoritmos podem ser analisados de forma a considerar o mesmo modelo. O modelo usado é a **RAM(Random-Acess-Machine)**, neste modelo consideramos que as instruções serão implementadas e executadas uma após outra, sem operações simultâneas. As instruções que temos nesse modelo são as mesmas que existem em computadores da nossa realidade: instruções aritméticas, de movimento de dados, e de controle.

O modelo também fornece uma noção de passo elementar(instruções) de computação da qual derivamos medidas de complexidade de tempo. O tempo de execução do modelo permite termos o número de passos elementares(instruções) necessários para concluir uma computação.

Com a complexidade de tempo podemos estabelecer por meio de comparações qual é o algoritmo mais eficiente. É possível calcular o melhor caso, caso médio e pior caso. Entretanto o mais importante é o pior caso, pois sabemos que não será executado em um tempo acima deste.

Questão 2. (Adaptado de Horowitz et al.) Uma maneira de analisar a complexidade de um algoritmo é instrumentando o (pseudo) código, ou seja, adicionando uma variável global conta inicializada com zero e, a cada declaração do algoritmo original, inserir um incremento à variável conta que corresponde ao número de instruções executadas por aquela declaração.

```
Algoritmo D(x,n)
1  i := 1;
2  repita
3        x[i] := x[i] + 2;  i := i + 2;
4  até  (i > n);
5  i := 1;
6  enquanto  (i ≤ ⌊n/2⌋)
7        x[i] := x[i] + x[i + 1];  i := i + 1;
```

a) Insira declarações para instrumentar o algoritmo acima introduzindo incrementos na variável conta. Escolha um número de instruções adequado para cada declaração.

Algoritmo D(x,n)

i:=1;
Conta++;
Repita
Conta++;
x[i]:=x[i] + 2;
Conta++;
i:= i + 2;
Conta++;
Até (i > n) ;
i:=1;
Conta++;
Conta++;
Enquanto (i<= |n/2|)
Conta++;
x[i]:=x[i] + x[i + 1] ;
Conta++;
i:= i + 1;
Conta++

(b) Qual é o valor exato da variável conta quando o algoritmo termina? Presuma que ela foi inicializada com zero antes da execução.
R: T(n) = 1 + 3*n/2 + 1 + 1 +3*(n/2)
T(n) = 1 + 3n/2 + 2 + 3n/2
T(n) = 3+ 3n

(c) Obtenha uma contagem de instruções do algoritmo acima usando uma tabela de frequência. Mostre a tabela de contagem.

Constante/ Numero linha	Valor
c1	1
c2	1
c3	2* n/2
c4	n/2
c5	1
c6	n/2
c7	2*n/2

$T(n) = c1 + c2 + 2c3(n/2) + c4(n/2) + c5 + c6(n/2) + 2c7(n/2)$
$T(n) = c1 + c2 + nc3 + (n/2)c4 + c5 + (n/2)c6 + nc7$

Considere todas as constantes com valor igual a 1 :

$T(n) = 1 + 1 + n + n/2 + 1 + n/2 + n = 3n + 3$

Questão 3. (Adaptado de Horowitz et al.) Um quadrado mágico é uma matriz n×n de inteiros de 1 até n 2 tais que a soma de cada linha, coluna, ou diagonal é a mesma. H. Coxter deu a seguinte regra para para criar um quadrado mágico quando n é ímpar. Comece com 1 no centro da última linha; então vá para baixo e para a direita, atribuindo números em ordem crescente nos quadrados vazios; se você sair do quadrado, imagine que o mesmo quadrado está ladrilhando o plano e continue; se o próximo quadrado já estiver ocupado, suba uma linha (ao invés de se mover para baixo e para a direita) e continue.

(a) Escreva um algoritmo no formato de pseudocódigo que implemente a regra de Coxter acima.

Algorithm 1 Algoritmo Coxter

```
1:  procedure ALGORITMOCOXTER(n)
2:      for (i=0; i< n; i++) do                          ▷ inicializa a matriz
3:          for (j=0; j< n; j++) do matriz[i][j] = nil
4:      numero ← 0
5:      lin ← n − 1
6:      col ← ⌊n/2⌋
7:      anteriorCol ← 0
8:      anteriorLin ← 0
9:      tamMatriz ← n ∗ n
10:     while numero <= tamMatriz do
11:         matriz[lin][col] ← ++numero
12:         anteriorCol ← col          ▷ salva valor antigo para caso já exista valor na posição
13:         col ← (col + 1) mod n                        ▷ volta para o comeco das colunas
14:         anteriorLin ← lin          ▷ salva valor antigo para caso jah exista valor na posição
15:         lin ← (lin + 1)
16:         if lin > (n-1) then
17:             lin ← 0                ▷ se for negativo volta para a ultima linha da matriz
18:         if matriz[lin][col] != nil then      ▷ se a matriz ja tiver preenchida na posicao...
19:             col ← anteriorCol                        ▷ preencher outra posição
20:             lin ← anteriorLin − 1
21:             if lin < 0 then
22:                 lin ← n − 1
23:     return matriz
```

(b) Analise a complexidade desse algoritmo. Justifique a resposta.

(1) A matriz para ser inicializada nas linhas 2 e 3 tem a realização de c1 x n x n operações. Consideremos c1*n² para a soma.
(2) Nas linhas seguintes de 4-9 teremos inicializações que gastaram 1 operação vezes uma constante, estas por serem um valor muito pequeno não influenciaram no cálculo da complexidade. Consideremos valor c2 a soma de todos esses valores constantes.
(3) A expressão do While na linha 10 a 22 tem n x n operações vezes uma constante. Consideremos c3*n² para a soma.
(4) A somatória do número dos passos 1,2 e 3 acima resultará em polinomial igual a 2 portanto, uma complexidade O(n²). A soma será : (c1 + c3)*n² + c2

Questão 4. (Dasgupta et al.) Em cada uma das situações abaixo, indique se f = O(g), ou se f = Ω(g), ou ambos (quando f = Θ(g)). [Indique a resposta de todos e justifique formalmente somente as respostas dos itens (g), (n) e (q). Dica: para (q), suponha que k é constante e compare o quanto cada função cresce para de n para n+1.]

	$f(n)$	$g(n)$
(a)	$n-100$	$n-200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n+\log n$	$n+(\log n)^2$
(d)	$n\log n$	$10n\log 10n$
(e)	$\log 2n$	$\log 3n$
(f)	$10\log n$	$\log(n^2)$
(g)	$n^{1.01}$	$n\log^2 n$
(h)	$n^2/\log n$	$n(\log n)^2$
(i)	$n^{0.1}$	$(\log n)^{10}$
(j)	$(\log n)^{\log n}$	$n/\log n$
(k)	$\sqrt{n}$	$(\log n)^3$
(l)	$n^{1/2}$	$5^{\log_2 n}$
(m)	$n2^n$	3^n
(n)	2^n	2^{n+1}
(o)	$n!$	2^n
(p)	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$
(q)	$\sum_{i=1}^{n} i^k$	n^{k+1}

a) f = O(g) e f = Ω(g) então f = Θ(g)
b) f = O(g)
c) f = O(g) e f = Ω(g) então f = Θ(g)
d) f = O(g) e f = Ω(g) então f = Θ(g)
e) f = O(g) e f = Ω(g) então f = Θ(g)
f) f = O(g) e f = Ω(g) então f = Θ(g)
g)

Para resolver o problema podemos aplicar L'Hospistal e derivar. Derivando várias vezes temos uma função com n com potência próxima de 2. Como infinito vai ser levado ao quadrado o limite será infinito. Com o limite infinito temos f = ω(g) e como o limite > 0 então f = Ω(g).

$$\lim_{n\to\infty} \frac{n^{1,01}}{nlog^2 n} = \lim_{n\to\infty} \frac{n^{0,01}}{log^2 n} = \lim_{n\to\infty} \frac{0,01n^{-0,99}}{2n^{-1}logn} = \lim_{n\to\infty} \frac{0,01n^{0,01}}{2logn} = \lim_{n\to\infty} \frac{0,00001n^{-0,99}}{2n^{-1}}$$

$$= \lim_{n\to\infty} 0.00001n^{0,01} = \infty$$

h) f = Ω(g)
i) f = Ω(g)
j) f = Ω(g)
k) f = Ω(g)
l) f = O(g)
m) f = O(g)
n) 2^n > C* (2^n) * 2 ,
Considere: C = 1/4

Então f = Ω(g)

2^n < C * (2^n) * 2 ,

Considere: C = 1

Então f = O(g)

Portanto f = O(g) e f = Ω(g) então f = Θ(g)

o) f = Ω(g)

p) f = O(g)

q)

Para resolver esse exercício foi usado inicialmente a intuição:

Se k = 1 a somatória será 1 + 2 +3 … n e isso é uma P.A, ou seja : n(n + 1)/2 = $n^2 + \frac{n}{2}$.

Se k = 2 a somatória será $1^2 + 2^2 + 3^2 \dots n^2$, temos n(n+1)(2n+1)/6 = $\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$

Se k = 3 a somatória será $1^3 + 2^3 + 3^3 \dots n^3$, temos $n^2 (n+1)^2/4$ = $\frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$

Então podemos perceber que a somatória tem como maior polinômio $\frac{n^{k+1}}{k+1}$, com isso

$\frac{n^{k+1}}{k+1}$ > c1 n^{k+1} se a constante c1 = $\frac{1}{k+2}$, assim Ω(g) e

$\frac{n^{k+1}}{k+1}$ < c1 n^{k+1} se a constante c1 = 1 , assim O(g).

Com isso podemos concluir que a f = Θ(g).

.

Questão 5. (Kleinberg e Tardos) Considere o seguinte problema básico. Você recebe um vetor A consistindo de n inteiros A[1],A[2],...,A[n]. O seu objetivo é exibir um vetor bidimensional B, de dimensão n-por-n, em que B[i, j] (para i < j) contenha a soma das posições A[i] até [j], isso é, a soma A[i]+A[i+1]+...+A[j] (o valor de B[i, j] é deixado indefinido sempre que i ≥ j). Aqui está um algoritmo simples para resolver esse problema:

1 Para i = 1,2,...,n

2 Para j = i+1,i+2,...,n

3 Some todos os valores de A[i] até A[j] Guarde o resultado em B[i, j]

(a) Para alguma função f que você escolha, dê um limitando da forma O(f(n)) no tempo de execução desse algoritmo sob uma entrada de tamanho n (i.e., limite o número de operações realizadas pelo algoritmo).

R: O for da linha 1 realizará n operações , o for da linha 2 realizará (n-1) operações, mas será executado n vezes devido ao for da minha 1 então terá n(n-1)/2 operações . Para realizar a soma será necessário um outro for que constará na linha 3, este realizando j - i operações sendo que j pode variar até n-1 pois vai de 2 até n, como existe n(n-1) operações devido aos outros 2 for temos n(n-1)(n-1) . Então O(n³).

(b) Para essa mesma função f , mostre que o tempo de execução do algoritmo par uma entrada de tamanho n também é Ω(f(n)). (Isso mostra um limitante assintótico justo de Θ(f(n)) no tempo de execução.)
R:
O primeiro for executa n vezes. O número de iterações do segundo for é:
(n-1)+(n-2)+...+1+0 = (1/2) (n-1) ((n+1) - 1) = (n^2 - n)/2

Para o terceiro for vamos usar i = $\frac{n}{2}$, por considerar o número médio de operações. Portanto, para i=n/2, o número de operações de soma é:
1+2+...+(n - $\frac{n}{2}$) = (n - ($\frac{n}{2}$))(n - ($\frac{n}{2}$) + 1)/2 = ($\frac{n}{2}$ * $\frac{n}{2}$)/2 = $\frac{n^2}{8}$

Todos os valores de i <= $\frac{n}{2}$ farão mais adições do que para i = $\frac{n}{2}$, então eles farão pelo menos $\frac{n^2}{8}$ operações de adição. Quanto menor o i maior o número de operações, logo teremos $\frac{n}{2}$ multiplicado por $\frac{n^2}{8}$ será o total de operações considerando os números de i <= $\frac{n}{2}$. Com isso temos $\frac{n^3}{16}$. Assim a constante encontrada é 1/16. Como temos O(n³) e Ω(n³) então Θ(f(n)) = Θ(n³)

(c) Embora o algoritmo que você analisou nos itens (a) e (b) é o meio mais natural de resolver o problema — afinal, ele só itera através das entradas relevantes do vetor B, preenchendo o valor de cada uma — ele contém fontes de ineficiência altamente desnecessárias. Dê um algoritmo diferente que resolve esse problema com um tempo de execução assintoticamente melhor. Em outras palavras, você deve projetar um algoritmo com tempo de execução O(g(n)), onde $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ = 0.

```
B[1,1] = A[1]
Para i = 2.... n
        B[1,i] = A[i-1] + A[i]

Para i = 2,...,n
        Para j = 2,...,n
```

B[i,j] = B[i-1,j] - B[i-1,i-1]

Esse algoritmo tem $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ .Porque $\lim_{n\to\infty} \frac{n^2}{n^3} = \frac{1}{n} = \frac{1}{\infty} = 0$

Questão 6. Leia o texto das notas de aula de Jeff Erickson, em http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/00-intro.pdf, da seção 0.5 A Longer Example: Stable Matching até subseção Running Time (inclusive) e faça os exercícios 2 e 3 do mesmo documento.

2. Describe and analyze the Boston Pool stable matching algorithm in more detail, so that the worst-case running time is O(n²), as claimed earlier in the notes.
R: Segue uma descrição do algoritmo:

Marque que todos os Hospitais e Médicos estão livres;
Enquanto algum hospital H estiver livre **faça**
    M = primeiro médico na lista de H em que H ainda não tenha se oferecido:
    se M está livre **então**
        marque que M aceitou proposta de H
    senão
        se M prefere H a seu hospital anterior H' **então**
            marque que M aceitou a proposta de H e que H' está livre
        senão
            M rejeita a proposta de H (e H continua livre)
        fimSe
    fimSe
fimEnquanto

No pior caso, um dos hospitais ficará com sua última opção de médico, enquanto que os outros ficarão com a segunda pior opção, isto é, 1 dos hospitais fará no máximo n propostas por médicos, e os outros (n - 1) hospitais farão no máximo (n - 1) propostas, até que se encontre uma combinação estável.

Portanto, o algoritmo realizará no máximo (n-1)(n-1) + n = n² -2n + 1 + n = n² - n +1
Assim chegamos a O(n²).

3. Prove that it is possible for the Boston Pool algorithm to execute Ω(n²) rounds. (You need to describe both a suitable input and a sequence of Ω(n²) valid proposals.)

q	r	s	t
D	A	B	A
C	B	A	D
B	D	D	C
A	C	C	B

A	B	C	D
q	t	s	s
s	q	t	t
t	s	q	q
r	r	r	r

O algoritmo de Gale tem $T(n) = n^2 - n + 1 = 4^2 - 4 + 1 = 13$. Seguem os passos para obter a solução das tabelas acima:

1) O hospital A se oferece ao médico q.
2) O hospital B se oferece ao médico t.
3) O hospital C se oferece ao médico s.
4) O hospital D se oferece ao médico s, que rejeita a oferta anterior do hospital C.
5) O hospital C se oferece ao médico t, que rejeita a oferta anterior do hospital B.
6) O hospital B se oferece ao médico q, que rejeita a oferta anterior do hospital A.
7) O hospital A se oferece ao médico s,que rejeita a oferta anterior do hospital D.
8) O hospital D se oferece ao médico t,que rejeita a oferta anterior do hospital C.
9) O hospital C se oferece ao médico q,que rejeita a oferta anterior do hospital B.
10) O hospital B se oferece ao médico s,que rejeita a oferta anterior do hospital A.
11) O hospital A se oferece ao médico t,que rejeita a oferta anterior do hospital D.
12) O hospital D se oferece ao médico q,que rejeita a oferta anterior do hospital C.
13) O hospital C se oferece ao médico r.

Para $\Omega(n^2)$:

$n^2 - n + 1 > c1*n^2$

$4 - 2 + 1 > ½ *(4)$

$3 > 2$

Considere: $c1 = ½$ $n = 2$