Instituto de Computação – UNICAMP
Complexidade de Algoritmos I – Turma A
Exercícios: **Introdução**

- Os exercícios devem ser submetidos como um arquivo em formato PDF (digitado ou manuscrito digitalizado), no prazo estipulado, na página https://www.ic.unicamp.br/~lehilton/mo417a/submit/.
- Só serão aceitas listas com todas questões ser respondidas, mas serão corrigidos **apenas** os itens sorteados em https://www.randomresult.com/ticket.php?t=224478R3SF2.

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

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)
    i := 1;
    repita
        x[i] := x[i]+2;  i := i+2;
    até  (i > n);
    i := 1;
    enquanto  (i ≤ ⌊n/2⌋)
        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.

(b) Qual é o valor exato da variável *conta* quando o algoritmo termina? Presuma que ela foi inicializada com zero antes da execução.

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

Questão 3. (Adaptado de Horowitz et al.) Um quadrado mágico é uma matriz $n \times n$ 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.

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

Questão 4. (Dasgupta et al.) Em cada uma das situações abaixo, indique se $f = O(g)$, ou se $f = \Omega(g)$, ou ambos (quando $f = \Theta(g)$).

[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}

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 $i \geq j$). Aqui está um algoritmo simples para resolver esse problema:

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

(b) Para essa mesma função f, mostre que o tempo de execução do algoritmo par uma entrada de tamanho n também é $\Omega(f(n))$. (Isso mostra um limitante assintótico justo de $\Theta(f(n))$ no tempo de execução.)

(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 $\lim_{n\to\infty} g(n)/f(n) = 0$.

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.