1ª Lista de Exercícios de Introdução à Análise de Algoritmos
Prof. Glauber Cintra – Entrega: 23/jan/2012

Esta lista deve ser feita por grupos de no **mínimo** 3 e no **máximo** 4 alunos.

Nomes: _____
_____
_____
_____

1) **(1,5 pontos)** Numere as funções abaixo em ordem estritamente crescente de dominação assintótica. Se duas funções f e g são tais que $f \in o(g)$ então f deve ter um número menor do que g. Se $f \in \Theta(g)$ então f e g devem ter o mesmo número.

(5) $2n + n^2$	(2) $3\log n^2$	(4) $n\log n^2$
(3) $6n$	(7) 2^{n+1}	(7) 2^n
(8) $n!$	(6) n^3	(2) $\log n$
(1) 5		

2) **(1,5 pontos)** Explique o significado dos termos *algoritmo*, *algoritmo computacional, algoritmo correto, algoritmo eficiente* e *tamanho da entrada de um algoritmo*.

Algoritmo: Sequência de instruções que visa resolver instâncias de um problema.
Algoritmo Computacional: Algoritmo constituído apenas de instruções não ambíguas.
Algoritmo Correto: Algoritmo que resolve corretamente e em tempo finito todas as instâncias do problema para o qual foi desenvolvido.
Algoritmo eficiente: Algoritmo que executa uma quantidade de instruções elementares limitada por um polinômio no tamanho da entrada. Onde instruções elementares são instruções que são executadas em tempo constante no agente utilizado.
Tamanho da entrada: quantidade de símbolos utilizados para codificar os dados de entradas.

3) **(2 pontos)** Indique quais são o melhor caso e o pior caso do algoritmo abaixo e sua região crítica. Qual a complexidade temporal desse algoritmo no melhor e no pior caso? Qual a complexidade espacial desse algoritmo? O algoritmo é eficiente? Justifique suas respostas.

```
Algoritmo Maior
   Entrada: um vetor v com n posições
   Saída: o maior elemento de v
   maior = v[0]
   para i = 1 até n - 1
        se v[i] > maior
             maior = v[i]
   devolva maior
```

Melhor caso: quando o maior valor está na primeira posição do vetor.
Pior Caso: quando o vetor está ordenado de forma crescente.
Região critica: `se v[i] > maior`
Complexidade Temporal: tanto no melhor caso como no pior caso temos como uma região crítica a linha "`se v[i] > maior`" que é executada $n-1$ vezes logo a complexidade temporal do melhor caso e do pior caso pertence a $\Theta(n)$.

Complexidade espacial: O algoritmo utiliza duas variáveis escalares, ou seja, a complexidade espacial pertence $O(1)$.

O algoritmo é eficiente pois a complexidade temporal é limitada por um polinômio no tamanho da entrada.

4) **(3 pontos)** Indique quais são o melhor caso e o pior caso do algoritmo abaixo e sua região crítica. Qual a complexidade temporal desse algoritmo no melhor e no pior caso? Qual a complexidade espacial desse algoritmo? O algoritmo é eficiente? Justifique suas respostas. Finalmente, prove que esse algoritmo é correto.

```
Algoritmo Seleção
  Entrada: um vetor v com n posições (indexado a partir do zero)
  Saída: o vetor v em ordem crescente
  para i = 0 to n - 2 (I)
    min = i
    para j = i + 1 to n - 1 (II)
      se v[min] > v[j]
        min = j
    aux = v[i]
    v[i] = v[min]
    v[min] = aux
```

Melhor caso: quando o maior valor está em ordem crescente.
Pior Caso: quando o vetor está ordenado de forma decrescente.
Região critica: `se v[min] > v[j]`
Complexidade Temporal: tanto no melhor caso como no pior caso temos como região crítica a linha "`se v[min] > v[j]`" que é executada $\sum_{i=1}^{n-1} n-i=(n^2-n)/2$ vezes logo a complexidade temporal do melhor caso e do pior caso pertence a $\Theta(n^2)$.

Complexidade espacial: O algoritmo utiliza quatro variáveis escalares, ou seja, a complexidade espacial pertence $O(1)$.

O algoritmo é eficiente pois a complexidade temporal é limitada por um polinômio no tamanho da entrada.

O algoritmo Seleção é correto.

Prova: O algoritmo Seleção intera no laço `(I)` até o elemento $V[n-2]$, com isso o invariante 1 garante que os elementos do sub-arranjo $V(0..n-2)$ serão os $n-1$ menores elementos de V e que eles estarão ordenados de forma crescente. Também pelo invariante 1 é fácil perceber que o elemento $V[n-1]$ possuirá no final do laço um valor maior ou igual ao maior elemento do sub-arranjo $V(0..n-2)$, o que garante que todo o vetor V estará ordenado de forma crescente no final da execução do algoritmo.

Invariante 1: A cada interação do laço `(I)` teremos dois sub-arranjos de $V(0..n-1)$, $Vo(0..i)$ e $Vd(i+1..n-1)$, onde n é o tamanho do vetor V e i é o contador do

laço. O sub-arranjo Vo contém os $i+1$ menores elementos de V ordenados de forma crescente e o sub-arranjo Vd contém os outros elementos de V.

Prova: Façamos indução em i com base 0.

Na primeira interação do laço `(I)` Vo terá um elemento e Vd terá $n-1$. O algoritmo entra no laço `(II)` e o invariante 2 garante que a variável min contém o índice do menor elemento em V. A troca de valores entre $V[0]$ e $V[min]$ garante que Vo terá o menor valor de V e Vd os outros valores. Como Vo só tem um elemento ele está ordenado de forma crescente.

Em uma interação com $i=t+1$ o laço `(II)` irá intera sobre os elementos de Vd da interação anterior $(i=t)$. O invariante 2 garante o índice do menor elemento em $Vd(i=t)$, a troca entre o primeiro elemento de $Vd(i=t)$ e $V[min]$ faz com que o menor elemento de $Vd(i=t)$, que é maior ou igual a qualquer elemento de $Vo(i=t)$, esteja na primeira posição de $Vd(i=t)$, que no final da interação passa a ser o ultimo elemento de $Vo(i=t+1)$. Com isso no final da interação $i=t+1$ teremos Vo com os $t+2$ menores elementos de V ordenados de forma crescente e Vd com os outros elementos.

Invariante 2: No final de cada interação do laço `(II)` o valor da variável min será o índice do menor elemento no sub-arranjo $V(i..i+x)$, onde i é o contador do laço `(I)` e x é o número da interação no laço `(II)`.

Prova: Façamos indução em x com base 0.

Na primeira interação ($x=1$) temos um sub-arranjo $V(i..i+1)$ com a variável min com o valor de i. O laço `(II)` executará uma vez, fazendo a comparação entre $V[i]$ e $V[i+1]$, se $V[i+1]$ for menor do que $V[i]$ a variável min recebe o valor $i+i$, o que é correto, caso contrário continua com valor igual a i, o que é correto.

Em uma interação x terremos um sub-arranjo $V(i..i+x)$, que nada mais é do que o sub-arranjo da interação anterior ($V(i..i+x-1)$), que contém um elemento cujo índice está na variável min e esse é o menor elemento desse sub-arranjo, adicionado de um elemento ($V[i+x]$). Para garanti que o índice do menor elemento esteja na variável min é feita a comparação entre $V[min]$ e $V[i+x]$, se $V[i+x]$ for menor do que $V[min]$ então a variável min recebe o valor $i+x$, o que é correto, caso contrário continua com o mesmo valor, o que é correto.

5) **(3 pontos)** Escreva um algoritmo que receba uma lista de números (e, naturalmente, seu tamanho) e devolva a soma dos elementos dessa lista. Prove que seu algoritmo é correto e determine a complexidade temporal e espacial do algoritmo. O seu algoritmo é eficiente?

```
#Algoritmo Somatório
Entrada: uma lista L de números e de tamanho t
Saída: a soma dos elementos da lista L
Inicio:
        r = 0
        para i = 1 até t
                r = r + L[i]
        devolva r
Fim.
```

O algoritmo Somatório é correto.

Prova: É fácil perceber que quando a lista é vazia ($t=0$) o algoritmo não entra no laço e retorna 0, o que é correto. Quando a lista possui pelo menos um elemento ($t>0$), o

algoritmo entra no laço e o invariante 1 nos garante que o valor de r após o laço, que será devolvido, será o somatório dos elementos da lista ($r=\sum_{j=1}^{t} L[j]$), o que é correto.

Invariante 1: No final de cada interação do laço temos que $r=\sum_{j=1}^{i} L[j]$,onde i é o contador do laço.

Prova: Façamos indução em i com base 1.

Observe que no final da primeira interação teremos $r=0+L[1]=L[1]=\sum_{j=1}^{1} L[j]$.

Resta-nos provar que em uma interação com $i=t+1$ teremos $r=\sum_{j=1}^{t+1} L[j]$.

Note que na interação com $i=t+1$ o que faremos é somar o valor de r da interação anterior ($i=t$) com o elemento que está em $L[t+1]$.

Logo temos que $r=\left(\sum_{j=1}^{t} L[j]\right)+L[t+1]=\sum_{j=1}^{t+1} L[j]$.

Complexidade Espacial: o algoritmo somatório usará 2 variáveis escaláveis (r e i). Logo a complexidade espacial do algoritmo somatório pertence a O(1).

Complexidade Temporal: uma região crítica é “r = r+L[i]” que será executada t vezes, logo a complexidade temporal pertence a Θ(t).

O Algoritmo Somatório é eficiente.