Tópicos Avançados em Algoritmos - resolução sumária da folha de exercícios

1. **Verdadeiro ou falso?** F F F F

2. **LCS: a mais longa sub-sequência comum**

   (a) Ver apontamentos.

   (b) Descrição em linguagem python...

```
def sm(x,y):
    return seqmax(x,y,len(x)-1,len(y)-1)

def seqmax(x,y,i,j):
    # sequencia vazia?
    if i<0 or j<0:
        return 0
    if x[i]==y[j]:
        return 1+seqmax(x,y,i-1,j-1)
    return max(seqmax(x,y,i-1,j), seqmax(x,y,i,j-1))
```

   (c) Descrição em linguagem python...

```
def seqmax(x,y):
    m=len(x)
    n=len(y)
    maxi=10
    v = [[0]*maxi for i in range(maxi)]  # definir v[0..maxi-1][0..maxi-1]
    # 1a linha (i=0) e 1a coluna (j=0) ja' preenchidas com 0's
    i=1
    j=1
    comp=0
    for i in range(1,m):      # strings a começar no índice 1
        for j in range(1,n):
            if x[i]==y[j]:
                comp = v[i-1][j-1] + 1
            else:
                comp = max(v[i-1][j],v[i][j-1])
            v[i][j] = comp
    return comp
```

3. **Procura de x**
   Considere o problema de procurar um valor x num vector `v[1..n]` (não necessariamente ordenado) com n elementos distintos. A resposta é o índice `i` tal que `v[i]`$= x$, ou `-1` se x não ocorre em `v`. Utiliza-se o modelo externo dos dados com a análise no pior caso; todos os acessos aos dados são comparações entre x e um elemento `v[i]`. Seja $c(n)$ o número de acessos aos dados (comparações) no pior caso.

   (a) Usando o Princípio da Informação Necessária, determine um minorante de $c(n)$.
   O número de respostas possíveis é $n + 1$; temos então, pelo Princípio da Informação Necessária, $3^c \geq n + 1$, ou $c \geq \lceil \log_3(n + 1)\rceil$.

   (b) Determine o majorante seguinte: $c(n) \leq n$.
   Basta considerar o algoritmo clássico da pesquisa sequencial, o qual, quando x não está no vector, efectua exactamente n comparações.

   (c) Mostre que se verifica o minorante $c(n) \geq n$.
   Suponhamos que não existe qualquer comparação com um determinado elemento do vector `v`, digamos `v[i]`. Então, há situações em que o algoritmo não responde correctamente: basta pensar no caso em que $\texttt{v[j]} \neq x$ para todos os $\texttt{j} \neq \texttt{i}$.

   - Se o algoritmo responde `-1`, pode ser $\texttt{v[i]} = \texttt{x}$ e a resposta deveria ser i
   - Se responde `j` com $\texttt{j} \neq \texttt{i}$, está certamente errado.
   - Se responde `i`, pode ser $\texttt{v[i]} \neq x$.

   Conclui-se que são necessárias pelo menos n comparações, no pior caso.

   (d) Mostre que se usarmos a pesquisa binária (supondo que o vector está ordenado), temos um número de comparações $c(n)$ que satisfaz $c(n) \leq \log(n + 1)$, onde se supõe que n é da forma $n = 2^p - 1$. Que conclui neste caso (pesquisa binária) do minorante obtido na alínea 3a?
   Em termos de p, um majorante do número de comparações satisfaz a recorrência $f(1) = 1$, $f(p) = f(p - 1) + 1$ para $p \geq 2$, cuja solução é $f(p) = p$, ou seja, $c(n) = \log(n + 1)$.
   Este majorante difere essencialmente do minorante determinado na alínea 3a de uma constante multiplicativa, MAJORANTE $= \log 3 \times$ MINORANTE.

4. **Comparações no "merge"**
Considere o algoritmo de `merge(u[1..m],v[1..n])` onde todos os $m+n$ elementos dos vectores são distintos. Usa-se o modelo externo dos dados, com a análise no pior caso; todos os acessos aos dados da forma "$x =$`u[i]<v[j]`?".

   (a) Mostre que cada resultado possível pode ser representado por uma sequência de $m+n$ caracteres `u` ou `v`, com m caracteres `u` e n caracteres v e vice-versa.
   Um exemplo (com $m=4$ e $n=5$): `merge([1,2,8,10],[3,4,9])` $\Rightarrow$ `uuvvuvu`
   A partir de um vector resultado podemos formar uma sequência da forma indicada da forma seguinte:

   > Seja $x=\varepsilon$ no início. Percorremos o vector resultado, e acrescentamos à direita de x a letra `u` ou `v` conforme o valor seja originário do vector `u[]` ou do do vector `v[]`.

   Reciprocamente,

   > Uma sequência da forma indicada determina univocamente o resultado; basta percorrer essa sequência, preenchendo o vector resultado com o elemento seguinte de `u[]` ou de `v[]`, conforme o caracter seja `'u'` ou `'v'`.

   (b) Usando o Princípio da Informação Necessária determine um minorante do número de comparações efectuadas entre elementos de `u` e elementos de `v`.
   (ver apontamentos) Existem $\binom{m+n}{m}$ sequências de comprimento $m+n$ com m caracteres `'u'` e n caracteres `'v'`. Este é o número de resultados possível. Por outro lado, com c comparações, cada uma delas podendo ter 2 resultados, o número de resultados possível não pode exceder 2^c; concluimos que $2^c \geq \binom{m+n}{m}$.

   (c) Supondo que $m=n$ e usando a fórmula de Stirling, $n! \approx \sqrt{2\pi n}(n/e)^n$, determine uma forma fechada (sem usar factoriais nem combinações) para o minorante obtido na alínea anterior.
   (ver apontamentos)

$$c(n) \geq \log \binom{2n}{n} \approx \log \frac{\sqrt{4\pi n}(2n/e)^{2n}}{2\pi n(n/e)^{2n}} = \log\left(\frac{1}{\sqrt{\pi n}} \times 2^{2n}\right) = 2n - \frac{1}{2}\log(\pi n)$$

   Trata-se de um minorante próximo do majorante obtido do algoritmo de merge, $c(n) = 2n-1$.

5. **Ordenações...**
Seja `v[1..n]` um vector com n elementos distintos que se pretende ordenar. Mostre que qualquer algoritmo de ordenação baseado em comparações pode, para cada n, ter que efectuar (isto é, efectua no pior caso) $c(n) = \lceil \log(n!) \rceil$ comparações. Tabele esse minorante $c(n)$ do número de comparações para $n = 2, 3, 5, 6$.
(ver apontamentos) Essencialmente, temos

   – Supondo que cada comparação pode ter 2 resultados e que há c comparações, o número de configurações possíveis do algoritmo no final é, no máximo, 2^n configurações.
   – o número de resultados possíveis é $n!$

donde, $2^c \geq n!$, o que implica, sendo c inteiro, $c(n) = \lceil \log(n!) \rceil$.