1. (2.0pt) Fomos contratados para definir um planejamento de tarefas para uma equipe para as próximas n semanas. Para cada semana existem 3 tarefas possíveis: A , B e C. A realização da tarefa A na semana i gera um lucro de a_i; a realização da tarefa B na semana i gera um lucro de b_i e a realização da tarefa C na semana i gera um lucro de c_i. Sabe-se também que, para $i = 2, ..., n$, a tarefa C só pode ser realizada na semana i se a tarefa B for realizada na semana $i - 1$.

a) Seja $OPT(i)$ o rendimento máximo que pode ser obtido nas i primeiras semanas. Encontre uma equação de recorrência para $OPT(i)$.

b) Escreva um algoritmo polinomial e não recursivo para computar as tarefas que devem ser realizadas a cada semana de modo a maximizar o rendimento total.

Questão 2 (3.0pt) Considere a equação de recorrência que define o número de Lupinutesky: Se $i > 1$ ou $j > 1$ então

$$Lup(i, j) = 3Lup(i - 2, j) + \max_{k=1,...,j-1}\{k + Lup(i, k)\}.$$

Se $i \leq 1$ ou $j \leq 1$, então

$$Lup(i, j) = 1$$

.

a)(2.0pt) Desenvolva o PSEUDO-CÓDIGO de um algoritmo polinomial e recursivo para calcular $Lup(n, n)$.

b)(1.0pt) Analise a complexidade do algoritmo proposto no item (a) em função de n .

You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties.

Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

5.(3.0pt) Dada duas cadeias $X = x_1x_2...x_n$ e $Y = y_1y_2...y_m$, queremos encontrar o tamanho da maior subcadeia comum entre X e Y, ou seja, o maior valor de k tal que existem índices i e j satisfazendo $x_ix_{i+1}...x_{i+k-1} = y_jy_{j+1}...y_{j+k-1}$. Como exemplo, a maior subcadeia comum entre X=ABABD e Y=AABABBA é ABAB. Neste caso, $k = 4, i = 1$ e $j = 2$. Seja X_r a subcadeia de X com os r primeiros símbolos de X e Y_s é a subcadeia de Y com os s primeiros símbolos de Y. Por exemplo, se X=ABABD, então $X_3 = ABA$ e $X_2 = AB$.

Definimos $OPT(r, s)$ como o **tamanho** da maior subcadeia comum entre X_r e Y_s que **inclui** x_r e y_s. Sabemos que se $x_r = y_s$ e $r, s \geq 1$, então $OPT(r, s) = 1 + OPT(r - 1, s - 1)$. Se $x_r \neq y_s$ ou $r = 0$ ou $s = 0$, então $OPT(r, s) = 0$,

a) Descreva uma algoritmo **recursivo e eficiente** para calcular $OPT(n, m)$. Qual a complexidade do algoritmo?

b) Assuma que recebemos uma matriz $M[0..n, 0..m]$ preenchida com os valores de $OPT(,)$, ou seja, $M[i, j] = OPT(i, j)$ para todo i, j. Como podemos obter a **maior subcadeia comum** entre X e Y que inclui x_r e y_s a partir da matriz M? Note que não está sendo pedido o tamanho da maior subcadeia comum.

c) Assuma que recebemos uma matriz $M[0..n, 0..m]$ preenchida com os valores de OPT, ou seja, $M[i, j] = OPT(i, j)$ para todo i, j. Como podemos obter o **tamanho** da maior subcadeia comum entre X e Y? Qual a complexidade do procedimento?

2.19. ***A k-way merge operation.*** **Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.**

(a) Here's one strategy: Using the `merge` procedure from Section 2.3, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?

(b) Give a more efficient solution to this problem, using divide-and-conquer.

5) O número $B(n)$ de árvores binárias com n nós satisfaz a seguinte equação de recorrência $B(n) = \sum_{i=o}^{n} B(i)B(n-i)$ e $B(1) = 1$ para $n > 2$ e $B(1) = 1$.

a) Projete um algoritmo eficiente que recebe n e devolve o número de árvores binárias de tamanho n.

b) Analise a complexidade de tempo do algoritmo proposto. (Assuma que toda multiplicação e adição tem custo constante).

3. Seja $A = \{a_1, \ldots, a_n\}$ um conjunto de n números reais positivos. O objetivo desta questão é projetar um algoritmo para encontrar índices i e j, com $1 \leq i \leq j \leq n$, tal que $\prod_{k=i}^{j} a_k$ tem o maior valor possível.

a) Como seria um algoritmo de complexidade $O(n^2)$ para resolver este problema?

b) Como seria um algoritmo de complexidade $T(n)$ que satisfaz $\lim_{n\to\infty} \frac{T(n)}{n^2} = 0$?