There are 5 questions for a total of 100 points.

1. (20 points) (Gizmos)

Consider the following problem. You wish to purchase (at least) \( n \) identical gizmos. Gizmos come in packages of different sizes (i.e. the number of gizmos in a package) and different prices. You can buy any number of packages of each size, as long as the total number of gizmos is at least \( n \). You wish to find the minimum total price of such a set of packages.

The input is given as \( n \) and an array \( \text{Packages}[1..m] \), where each \( \text{Package}[i] \) has a positive integer field \( \text{Package}[i].size \) and a positive real field \( \text{Package}[i].price \) giving the number of gizmos in the package and the price of the package.

A recursive algorithm to solve this problem is:

\[
\text{BestPrice}(n : \text{positive integer}, \text{Packages}[1..m] : \text{array of pairs } (\text{size: integer, price: real}))
\]

1. \( \text{MinPrice}[1..m] \leftarrow [\infty, \infty, \ldots, \infty] \);
2. For \( d = 1 \) to \( m \) do:
3. \hspace{1em} IF \( \text{Packages}[d].size \geq n \) THEN
4. \hspace{2em} \( \text{MinPrice}[d] \leftarrow \text{Packages}[d].price \);
5. \hspace{1em} ELSE
6. \hspace{2em} \( \text{MinPrice}[d] \leftarrow \text{Packages}[d].price + \text{BestPrice}(n - \text{Packages}[d].size, \text{Packages}) \);
7. Return the minimum price in \( \text{MinPrice} \);

a. Explain (informally) why this algorithm is correct. (5 points)
b. Give a worst-case bound on the number of recursive calls this algorithm makes in terms of \( m \) and \( n \), assuming each package size is distinct. (15 points) (Hint: you can get a tighter bound than the most obvious one.)

2. (20 points) Consider two binary strings \( x_1 \cdots x_n \) and \( y_1 \cdots y_m \). A common subsequence of \( x \) and \( y \) is a pair of collections of indices \( i_1 < i_2 < \ldots < i_k \) and \( j_1 < j_2 < \ldots < j_k \) such that \( x_{i_1}x_{i_2}\cdots x_{i_k} = y_{j_1}y_{j_2}\cdots y_{j_k} \). For example, given \( x = \text{BANANA} \) and \( y = \text{ATANA} \), \( \langle [2,4],[1,3] \rangle \) is a common subsequence because \( x_2x_4 = y_1y_3 = \text{AA} \). The problem is: given \( x \) and \( y \), find the length of the longest common subsequence (LCS) of the two.

Here is a simple back-tracking recursive algorithm for this problem, based on the two cases: if the last characters are the same, then the LCS of \( x \) and \( y \) is the LCS of \( x[1 : n-1] \) and \( y[1 : m-1] \) plus the last character. If they are different, then the LCS of \( x \) and \( y \) is the LCS of \( x[1 : n-1] \) and \( y[1 : m-1] \) or the LCS of \( x[1 : n-1] \) and \( y[1 : m-1] \).

\[
\text{BTLongestCommonSubsequence} (x_1 \cdots x_n, y_1 \cdots y_m):
\]

1. IF \( n = 0 \) OR \( m = 0 \) THEN return 0;
2. IF \( x_n = y_m \) THEN return \( \text{BTLongestCommonSubsequence}(x_1 \cdots x_{n-1}, y_1 \cdots y_{m-1}) + 1 \);
3. ELSE return max(\( \text{BTLongestCommonSubsequence}(x_1 \cdots x_n, y_1 \cdots y_{m-1}) \), \( \text{BTLongestCommonSubsequence}(x_1 \cdots x_{n-1}, y_1 \cdots y_m) \));

a. (2 points): Show the tree of recursions the above algorithm would make on the above example.
b. (4 points). Give an upper bound on the total number of recursive calls this algorithm might make in terms of \( n \) and \( m \).
c. (2 points). Which distinct sub-problems can arise in the recursive calls for this problem?
d. (6 points): Translate the recursive algorithm into an equivalent DP algorithm, using your answer to part c.
e. (3 points) Give a time analysis for your DP algorithm
f. (3 points) Give the matrix your DP algorithm would produce on the above example.
NOTE: For questions 3, 4, and 5, structure your answer in the following format. You should explicitly give:

1. Description of sub-problems (2 points)
2. Base Case(s) (2 points)
3. Recursion (with justification) (A complete proof by induction is NOT required. However, you should explain why the recursion makes sense and how it covers all possibilities) (6 points)
4. Order in which sub-problems are solved (2 point)
5. Form of output (how do we get the final answer?) (2 point)
6. Pseudocode (3 points)
7. Runtime analysis (3 points)
8. A small example explained using an array or matrix as in the previous questions (Optional)

3. (20 points) Consider an $n \times n$ matrix $M$, where each entry is either 0 or 1. We want to find the side length of the largest square of all-1s in $M$.

$$
\begin{pmatrix}
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0
\end{pmatrix}
$$

In the above example, the algorithm should return 2 since the largest square of 1s has dimensions 2 $\times$ 2. Give an efficient algorithm that takes an $n \times n$ matrix $M$ and determines the side length of the largest all-1s square. Its running time should be $O(n^2)$.

4. (20 points) A bipartite graph is a graph in which the vertices can be partitioned into disjoint sets $L$ and $R$ such that all edges in the graph are between a vertex in $L$ and a vertex in $R$. Given a bipartite graph $G = (L, R, E)$, where $|L| = |R| = n$ and $L$ and $R$ are ordered (i.e. $L = \{v_1, \ldots, v_n\}$ and $R = \{u_1, \ldots, u_n\}$), a monotone matching is a way of pairing vertices in $L$ and $R$ up such that:

- each vertex appears in at most one pair (possibly in no pairs);
- each pair of vertices has an edge between them in $G$, and;
- for any two pairs $(v_i, u_j)$ and $(v_{i'}, u_{j'})$, if $i > i'$ then $j > j'$.

The problem is, given such a graph $G$, find the size (the number of pairs) of the largest monotone matching. Give an efficient algorithm solving the problem.

5. (20 points) The Hamming distance between two strings $x_1 \cdots x_n$ and $y_1 \cdots y_n$ is the number of positions $i$ where $x_i \neq y_i$. The Hamming distance between a string $x$ and a language $L \subseteq \{0, 1\}^*$ is the minimum Hamming distance between $x$ and some $n$-bit string $y \in L$.

Suppose $L$ is the language ‘all strings that don’t contain 010 as a substring.’ The problem now is: given a string $x$, find the Hamming distance between $x$ and $L$. 

3 of 3