Note: This exam has four questions for a total of 100 points. (The questions are on pages 1 and 2 of this exam.) Answer all questions. Write all your answers on your blue book(s).

Question I (30 points total)

(a) An algorithm to determine the satisfiability of Horn clauses was discussed in class. (See page 3 of this exam for an outline of the algorithm.) Use that algorithm to determine whether or not the following collection of five clauses defined on the set $X = \{x_1, x_2, x_3, x_4\}$ of Boolean variables is satisfiable:

$$
C_1 = (x_1 \lor x_2 \lor x_4), \quad C_2 = (x_3 \lor x_4), \quad C_3 = (x_3), \quad C_4 = (x_2 \lor x_3 \lor x_4), \quad C_5 = (x_3 \lor x_4)
$$

In each step, you should indicate which variable is being set to True. If you conclude that there is a satisfying assignment, you must show such an assignment. If you conclude that there is no satisfying assignment, you must indicate why you reached that conclusion. (15 points)

(b) We have six symbols $a$, $b$, $c$, $d$, $e$ and $f$ with respective frequencies 10, 20, 4, 4, 38 and 24. Construct an optimal prefix code for these symbols using Huffman’s Algorithm. Your answer needs to show only the final binary tree and the code for each symbol. You need not show the intermediate trees. (Pseudocode for Huffman’s Algorithm appears on page 3 of this examination.) (15 points)

Question II (25 points total)

(a) In the knapsack problem, we are given a collection of $n$ items, denoted by $I_1, I_2, \ldots, I_n$. Each item $I_j$ has a weight $w_j$ and a profit $p_j$, $1 \leq j \leq n$. We are also given a knapsack which can hold a total weight of $B$. Assume that all weights and profits are integers $\geq 1$. The goal is to pick a subset of the items so that the total weight of the items is at most $B$ and the total profit for the chosen items is a maximum over all subsets of items whose total weight is at most $B$. One greedy approach to solve this problem is as follows.

1. For each item $I_j$, compute the ratio $r_j = p_j/w_j$ (profit per unit weight), $1 \leq j \leq n$.
2. Sort the items into non-increasing order of their ratio values. Assume (by renumbering the items if necessary) that the sorted order of items is $(I_1, I_2, \ldots, I_n)$.
3. Initialize: remaining_capacity $= B$ and total_profit $= 0$.
4. for $j = 1$ to $n$ do
   if (remaining_capacity $\geq w_j$) then
       (i) Add item $I_j$ (with weight $w_j$ and profit $p_j$) to the knapsack.
       (ii) Decrement remaining_capacity by $w_j$ and increment total_profit by $p_j$.
5. Print total_profit.

Give an example to show that the above algorithm will not always produce the optimal profit.

Your answer must explicitly specify the weights of the items, their profits and the knapsack capacity. You must also indicate the sorted order by the ratio, the profit produced by the above heuristic and the optimal profit. (17 points)
(b) Let $C$ be a set of symbols. For each symbol $x \in C$, let $f(x)$ denote the frequency of $x$. **Prove or disprove:** If $C$ has two symbols $a$ and $b$ such that $f(a) = f(b)$, then in any optimal prefix code, the number of bits used to encode $a$ must be equal to the number of bits used to encode $b$.

**Question III** (25 points total)

Let $X[1..n]$ be an array where each element contains a lower case letter (i.e., one of the letters a, b, ..., z). Assume that $n \geq 2$. Recall that the five letters a, e, i, o and u are the **vowels**. A subarray $X[i..j]$, where $i \leq j$, consists of the elements $X[i], X[i+1], \ldots, X[j]$, and its length is $j - i + 1$. Note that a single element $X[i]$ is a subarray of length 1. A subarray $X[i..j]$ is **special** if all the characters in that subarray are vowels. This problem asks you think about a **divide-and-conquer** algorithm to find the length of a longest special subarray of $X$. This question has three parts.

(a) Indicate clearly the actions to be performed in the Divide, Conquer and the Combine steps of your algorithm. As part of the Conquer step, be sure to indicate when recursion ends and the value returned when recursion ends. To receive any credit, the total time taken by your Divide and Combine steps must be $O(n)$. (14 points)

(b) Based on your answer to Part (a), write pseudocode for the function

```c
int max_length_special(X, p, q)
```

which returns the length of a longest special subarray within the subarray $X[p..q]$. Assume that $p \leq q$. (6 points)

(c) Let $T(n)$ denote the running time of your algorithm when $X$ has $n$ elements. Specify a recurrence for $T(n)$ and indicate briefly how you arrived at the recurrence. There is no need to solve the recurrence. (5 points)

**Question IV** (20 points total)

We are given an unlimited number of stamps of $n$ different denominations $d_1, d_2, \ldots, d_n$, which are all positive integers. We are also given another positive integer $Y$. This problem explores a dynamic programming approach to decide whether the amount $Y$ can be realized using given denominations of stamps. Note that we may use any number (including zero) of stamps of a given denomination.

**Example:** Suppose we have a collection of 5-cent and 7-cent stamps. We can realize the amount of 17 cents (using two 5-cent stamps and one 7-cent stamp) but we cannot realize the amount of 13 cents.

The dynamic programming table $B[0..Y]$ (to be computed) is a one-dimensional Boolean array with the following significance: for $0 \leq j \leq Y$, $B[j]$ is **True** if stamps for $j$ cents can be realized using the given denominations (i.e., $d_1, d_2, \ldots, d_n$) and **False** otherwise.

(a) What is the value of $B[0]$? Why? (2 points)

(b) Suppose that for some $j \geq 1$, we have computed all the entries of the subarray $B[0..j-1]$. Specify an equation for computing the value of $B[j]$ using previously computed values. Explain how you arrived at the equation. Also indicate the time (as a big-O estimate) for computing $B[j]$ using your equation. To receive any credit, the time used to compute $B[j]$ must be a polynomial in $n$. (14 points)

(c) Based on your answers to Parts (a) and (b), indicate the time for computing all the entries of $B$ as a big-O estimate. Indicate how you arrived at the estimate. There is no need for any pseudocode. (4 points)
An Algorithm for Horn Clause Satisfiability:

1. Set \( x_i = \text{False}, 1 \leq i \leq n \). (Thus, all variables are set to \text{False} initially.)

2. while (there is an unsatisfied normal clause) do
   
   Change the assignment for the positive literal in the normal clause to \text{True}.

3. if (all purely negative clauses are satisfied) then
   Return the assignment.
   
   else
   
   Output “No Satisfying assignment exists”.

Iterative Version of Huffman’s Algorithm:

1. Create a node (of the tree) corresponding to each symbol in \( C \). Let \( S \) denote the resulting set of nodes.

2. while \((n \geq 2)\) do
   
   (a) Let \( x \) be a node with smallest frequency value in \( S \). Remove \( x \) from \( S \).
   
   (b) Let \( y \) be a node with smallest frequency value in \( S \). Remove \( y \) from \( S \).
   
   (c) Create a new node \( w \) with \( x \) and \( y \) as its children. Add \( w \) to \( S \).
   
   (d) Set \( f(w) = f(x) + f(y) \).
   
   (e) Set \( n = n - 1 \).

3. For each internal node of the tree, label the edge to the left child as 0 and the edge to the right child as 1.

4. For each leaf \( c \) (i.e., symbol in \( C \)), construct its codeword by concatenating the bits in the path from the root to \( c \).

Statement of Master Theorem:

Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function and let \( T(n) \) be defined on nonnegative integers by the recurrence

\[
T(n) = aT(n/b) + f(n)
\]

where \( n/b \) may be either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \).

Part 1: If \( f(n) = O(n^{\log_b a - \epsilon}) \) for some constant \( \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).

Part 2: If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \).

Part 3: If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some \( \epsilon > 0 \) and if \( a f(n/b) \leq c f(n) \) for some constant \( c < 1 \) and for sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).