This homework is to be done in groups of 3 (or 2 if necessary), students may select their own groups.

No additional problems this time, reading assignments will be updated on the class web page.

**Problems to be turned in:**

1. (10 pts, continued from HW5) Consider the following MANY-DISJOINT-SETS decision problem.

   Given an integer \( n \), a set \( S \) of subsets of \( \{1, 2, \ldots, n\} \), and a bound \( k \), does \( S \) contain \( k \) mutually disjoint subsets?

   For example, if given the input: \( n = 4 \), \( S = \{\{1, 2, 3\}, \{1, 4\}, \{3, 4\}, \{2, 3\}\} \), and \( k = 2 \), then the answer is “yes” since there are two subsets in \( S \) (\( \{1, 4\} \) and \( \{2, 3\} \)) that are disjoint. However, no three of the subsets in \( S \) are disjoint.

   Show that the language consisting of strings encoding “yes” instances of the MANY-DISJOINT-SETS problem is NP-complete. (Hint: there is a reduction from CLIQUE that is not too difficult).

2. (15 pts)

   This problem is about the 0-1 Knapsack problem (see page 425 of the text). The 0-1 Knapsack decision problem is:

   Given a weight bound \( W \), a value goal \( G \) and a set \( I \) of \( n \) items \( i_1, \ldots, i_n \) where each item \( i_j \) has an integer weight \( w_j \) and an integer value \( v_j \), is there a subset of the items that has combined weight at most \( W \) and total value at least \( G \)?

   a. (2 points) Describe a verification algorithm showing that the 0-1 Knapsack decision problem is in NP.

   b. (2 pts) Show how a polynomial time subroutine which solves the 0-1 Knapsack Decision problem can be used to create a polynomial time algorithm that solves the following 0-1 Knapsack value Problem:

      Given a weight bound \( W \) and a list of \( n \) items \( i_1, \ldots, i_n \) where each item \( i_j \) has an integer weight \( w_j \) and an integer value \( v_j \), what is the largest total value achievable by a subset of the items having weight at most \( W \)?

      Your algorithm can call the subroutine several times, but make sure your algorithm runs in polynomial time in the length of the input, even when numbers are written in binary.
c. (2 pts) Show how a polynomial time subroutine which solves the 0-1 Knapsack Decision problem can be used to create a polynomial time algorithm that solves the following 0-1 Knapsack Optimization Problem:

Given a weight bound $W$ and a list of $n$ items $i_1, \ldots, i_n$ where each item $i_j$ has an integer weight $w_j$ and an integer value $v_j$, find a subset of the items with total weight at most $W$ and total value as large as possible.

Your algorithm can call the subroutine several times, but make sure your algorithm runs in polynomial time in the length of the input even when numbers are written in binary.

d. (9 pts) Show that the 0-1 Knapsack decision problem is NP-complete by giving a polynomial time reduction from SubsetSum to the 0-1 Knapsack decision problem (i.e. show that $\text{SubsetSum} \leq_P \text{0-1Knapsack}$). Verify that your reduction runs in polynomial time and has the "if-and-only-if" property.