Total marks: 50 (10 points for each question)

1. Define a boolean array $R(i, j)$ for $0 \leq i \leq n$ and $0 \leq j \leq W$, where $W = \sum_{i=1}^{n} w_i$, as follows:

$$R(i,j) = \begin{cases} T & \text{if there exists an } S \subseteq \{1, \ldots, i\} \text{ such that } \sum_{k \in S} w_k = j \\ F & \text{otherwise} \end{cases}$$

To obtain the solution once R has been computed, use the following decision procedure: If W is odd, then the answer is NO. Otherwise, the answer is YES iff $R(n, \frac{1}{2}W) = T$.

To compute the array use the following recurrence: $R(0, 0) = T$, and $R(0, j) = F$, for $j > 0$ (initial conditions). Now, for $i, j > 0$,

$$R(i,j) = \begin{cases} T & \text{if } R(i-1, j) = T \text{ or } (w_i \leq j \text{ and } R(i-1, j-w_i) = T \\ F & \text{otherwise} \end{cases}$$

Note that it takes $O(nW)$ steps to compute the array R. The input is given in unary, so it is given in the form $\bar{w}_1\texttt{\#}\ldots\texttt{\#}\bar{w}_n$, where $\bar{w}_i$ denotes weight w_i in unary notation, i.e., $\bar{w}_i = \underbrace{\texttt{11}\ldots\texttt{1}}_{w_i}$. Thus the length of the input is $O(W)$. Since $n \leq W$, it follows that the running time of the above dynamic programming algorithm is in $O(W^2)$, and hence it is a polytime algorithm.

2. We show that MSSP $\xrightarrow{p}$ MSSD by giving a polytime algorithm for solving MSSP which has access to a ("black-box") solver for MSSD. Our algorithm has two parts: in the first part we compute the smallest $|S|$ such that $\sum_{i \in S} = t$ (if it exists). We can do this using linear search on $B = 1, 2, \ldots, d$, i.e., by quering MSSD($\langle w_1, \ldots, w_d, t, B\rangle$) with $B = 1$, then with $B = 2$, etc., and we stop the first time we get a YES answer. Let the result be $B_{\min}$. If we never get a YES answer, then no such set exists, so we stop. (Why can we get away with linear search, rather than binary search?)

   After part one is complete, we have $B_{\min}$ (or we know that no such set exists, in which case we stopped). In the second part we want to compute the members of a set S, where $|S| = B_{\min}$. Again, we query MSSD weight by weight as follows: if MSSD($\langle w_2, \ldots, w_d, t, B_{\min}\rangle$) is NO, then we know that weight w_1 is necessary, we output it, and consider w_2 with MSSD($\langle w_1, w_3, \ldots, w_d, t, B_{\min}\rangle$). If it is YES, we know that w_1 is *not* necessary, so we go on to consider MSSD($\langle w_2, \ldots, w_d, t, B_{\min}\rangle$).

   The guess-and-check algorithm that establishes that MSSD is in NP is the following: guess a subset $S \subseteq \{1, \ldots, d\}$ and check two things: (i) $|S| \leq B$, and (ii) $\sum_{i \in S} w_i = t$.

3. As in Problem 2, the idea is to query PMD with certain edges taken out to establish which edges form a perfect matching. Let $E = \{e_1, \ldots, e_m\}$ be the set of edges. First query PMD($e_1, \ldots, e_m$) to check *if* there is a perfect matching. Then query PMD($e_2, \ldots, e_m$). If the answer is YES, e_1 is not necessary, so we go on to e_2 and query PMD($e_3, \ldots, e_m$). If the answer is NO, then e_1 is needed for the perfect matching, we output e_1, and go on to consider e_2 by querying PMD($e_1, e_3, \ldots, e_m$), and so on, until we build a perfect matching.

   The guess-and-check algorithm: guess a subset $E_0 \subseteq E$, and check that it *is* a perfect matching (i.e., that no two edges share the same vertex, and that every vertex in U and V is included).

4. To do this question you have to design a Turing Machine with input alphabet $\Sigma = \{0, 1\}$, tape alphabet $\Gamma = \{0, 1, b\}$, a finite set of states Q (containing q_0, q_{accept} and q_{reject}), as well

as a transition function $\delta : (Q - \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$. You should present δ as a table (as in the notes). Your TM should always halt, and it should halt in the state q_{accept} iff the input is of the form xx (for example, your TM should accept 110110 or 010010, but it should reject 110111).

The design of such a TM allows a lot of freedom; it is just like writing a very low-level program. Therefore, there are going to be as many solutions to this problem as students, and so I will only give an outline of a possible solution (basically, I am going to repeat what was posted on the web).

You should assume that x is a string of 0s and 1s. The Turing machine that you design may write any symbol from $\{0, 1, b\}$. The first thing that your TM should do is find the middle of the string xx. One way to do it is to implement a counter; the Turing machine may use the portion of the tape to the left of xx to compute the number of bits of xx and divide it by two. There is, however, a simpler way to do it: let the TM go back in forth on the string xx, moving bits on the right end to the right, and on the left end to the left, for example if $x = 101$, so that $xx = 101101$, the TM would do the following:

$$101101 \to 1b01101 \to 1b0110b1 \to 10b110b1 \to 10b11b01 \to 101b1b01 \to 101bb101$$

(note that these are not single steps, "$\to$" means several steps of your TM), and once bb are together, the TM knows that it has $xbbx$ on the tape, so it has successfully "divided" the string in two. Of course you have to provide for cases where this cannot be done (for example, if the input has an odd number of bits–so it is *not* of the form xx–then the string would have to be rejected).

Once you have $xbbx$ on the input tape, the TM can now check that $x = x$; again it may be the case that you have $xbby$, where x is not y, in which case the TM must reject this input.

Note that it is good practice to "document" your program; that is, provide a clear (and brief) outline of the idea behind your TM.

5. To show that Partition $\leq_p$ DoublePartition we have to come up with a polytime function f which satisfies the following correctness condition:

   The original instance $I = \langle w_1, \dots, w_d \rangle$ is a YES instance (i.e., there is a set $S \subseteq \{1, \dots, d\}$ such that $\sum_{i \in S} w_i = \sum_{i \notin S} w_i$) iff the constructed instance $I' = f(I) = \langle w'_1, \dots, w'_{d'} \rangle$ is a YES instance (i.e., there is a set $S' \subseteq \{1, \dots, d'\}$ such that $\sum_{i \in S'} w'_i = 2 \sum_{i \notin S'} w'_j$).

   We now define the transformation f as follows: Let $W = w_1 + \cdots + w_d$.

   If W is odd, then I is a NO instance (of Partition), so we map it to a default NO instance of DoublePartition, for example $d' = 1, w'_1 = 1$.

   If W is even, we let $d' = d + 2$, we let $w'_i = w_i$ for $1 \leq i \leq d$, and we define two more weights $w'_{d+1} = \frac{5}{2}W$ and $w'_{d+2} = W$.

   This transformation is clearly polytime.

   We are now going to prove the correctness of this transformation (this was optional; in the future, when the proof of correctness is required, it will be asked for explicitly in the question).

   "$\Longrightarrow$" If S solves the instance I, then let $S' = S \cup \{d+1\}$. Then $\sum_{i \in S'} w'_i = \frac{1}{2}W + \frac{5}{2}W = 3W = 2\sum_{i \notin S'} w'_i$.

   "$\Longleftarrow$" Suppose S' solves instance I' of DoublePartition. Then $\sum_{i \in S'} w'_i = 3W$, so $d+1 \in S'$ but $d+2 \notin S'$. Let $S = S' - \{d+1\}$. Then $\sum_{i \in S} w_i = 3W - \frac{5}{2}W = \frac{1}{2}W$.