1. As discussed in class, if \( f_i(y) \) is the optimal profit for \( \text{KNAP}(1, j, y) \), the recurrence relation for \( f_i(y) \) is given by:\n\[
f_i(y) = \max \{ f_{i-1}(y), f_{i-1}(y - w_i) + p_i \}.
\]
Also, \( f_0(y) = 0 \) for all non-negative values of \( y \) and \( f_i(y) = -\infty \) when \( y \) is negative. From these relations we compute \( f_0(y), f_1(y), f_2(y), f_3(y), f_4(y) \) for all \( 0 \leq y \leq 5 \). These values are shown in the following table.

<table>
<thead>
<tr>
<th>Function</th>
<th>( y = 0 )</th>
<th>( y = 1 )</th>
<th>( y = 2 )</th>
<th>( y = 3 )</th>
<th>( y = 4 )</th>
<th>( y = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( f_1 )</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>0</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>35</td>
<td>35</td>
<td>45</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>45</td>
<td>45</td>
</tr>
</tbody>
</table>

For example, \( f_3(5) = \max \{ f_2(5), f_2(5 - 2) + 15 \} = \max \{ 30, 30 + 15 \} = 45 \). Also, \( f_4(4) = \max \{ f_3(4), f_3(4 - 3) + 25 \} = \max \{ 35, 20 + 25 \} = 45 \); and so on. Thus the optimal profit is 45.

2. A simple algorithm would be to use one of the graph traversal algorithms \( \text{BFT} \) or \( \text{DFT} \), and color nodes on the way.

Say we use BFT. Start with an arbitrary node, color it \textbf{Red}. Then go over to its neighbors and color them \textbf{Blue}. Continue in this way until -

(a) You attempt coloring a \textbf{Red} node \textbf{Blue} or vice versa. That means that there are 2 nodes of the same color that are neighbors. Report that the graphs is \textit{not bipartite}.

(b) You end the traversal, with no conflicts. Report that the graph is \textit{bipartite}.

3. Step 1: Create an array \( P \) of \( n \) elements and initialize all the elements to 0.
Step 2: Assign one processor to each element in the array \( A \). Processor \( i \) will attempt to write a 1 in \( P[A[i]] \) (i.e., at the location \( A[i] \) in \( P \)). After this parallel write step, if there are any repeated elements in \( A \) then there would be 0 in at least one location in \( P \).
Step 3: Find if there is a 0 in at least one location in \( P \). Allocate a location \( M \) in the common memory and initialize it to 0. Processor \( i \) will check the element at \( P[i] \) and will write 1 in the common location if \( P[i] \) is zero. At the end of this step, \( M \) will contain 1 if there is a repeated element in the array \( A \).

Each of the three steps takes \( O(1) \) time. Total time = \( O(1) \). Any version of the CRCW PRAM can be used here.

4. Let \( X = x_1, x_2, \ldots, x_n \) and \( Y = y_1, y_2, \ldots, y_n \). We will employ \( n^2 \) processors and compute the rank of each input key in the merged list. Once we know the ranks of the keys we can output them in sorted order in an additional one unit of time.

Assign \( n \) processors \( x_i \), for \( 1 \leq i \leq n \). The \( n \) processors assigned to \( x_i \) compare \( x_i \) with each element of \( Y \) in parallel and figure out the unique \( j \) such that \( y_{j-1} < x_i < y_j \). This takes \( O(1) \) time. Now we know that the rank of \( x_i \) is \( (i - 1) + (j - 1) + 1 \).

Similarly, we can assign \( n \) processors to \( y_j \), for \( 1 \leq j \leq n \), and compute the rank of \( y_j \) in \( X \) and hence compute the global rank of \( y_j \).

Clearly, the entire algorithm runs in \( O(1) \) time.
5. If $n$ is the size of the instance of $\pi_1$, we can translate this instance into an instance of $\pi_2$ in $n^5$ time. This implies that the size $m$ of the corresponding instance of $\pi_2$ cannot be more than $n^5$. Since the algorithm for $\pi_2$ takes $O(m^{20})$ time, this run time is no more than $O((n^5)^{20}) = O(n^{100})$. Thus the instance of $\pi_1$ can be solved in time $n^5 + O(n^{100}) = O(n^{100})$.

6. To decide the assignment for $x_1$ (i.e., whether $x_1$ is included or not), do the following: Call $\text{DK}(n - 1 \text{ objects}, m, r)$. Here, $n - 1$ objects are obtained by excluding $x_1$ from the given $n$ objects. If the answer is yes, $x_1 = 0$. If the answer is no, $x_1$ is included in the optimal solution and hence $x_1 = 1$.

Now, to decide whether $x_2$ is included or not, do the following:
Case 1: If $x_1$ is not included, call $\text{DK}(n - 2 \text{ objects}, m, r)$. $n - 2$ objects here are obtained by excluding both $x_1$ and $x_2$ from the $n$ objects.
Case 2: If $x_1$ is included, call $\text{DK}(n - 2 \text{ objects}, m - w_1, r - p_1)$.

The decision to include $x_2$ or not in both the cases above is similar to that of the decision made at $x_1$.
Repeat the above procedure for all the objects $x_3$ through $x_n$. 