1. Let \( P(i, k) \) be 1 if there exists a subset whose sum is \( k \) from among the first \( i \) items and zero otherwise. We are interested in computing \( P(n, K) \). A recurrence relation for \( P(i, k) \) is given by:

\[
P(i, k) = 1 \text{ iff either } P(i - 1, k) = 1 \text{ or } P(i - 1, k - k_i) = 1
\]

We can use the above recurrence relation to compute \( P(n, K) \) in \( O(nK) \) time. For example, we can compute the following sequence: \( P(1, 1), P(1, 2), \ldots, P(1, K), P(2, 1), P(2, 2), \ldots, P(2, K), \ldots, P(n, 1), \ldots, P(n, K) \).

2. A BFS starting from the node \( s \) can be used to solve this problem. Recall that BFS visits the nodes in the order of their distances (measured as the number of edges) from \( s \). While visiting any node, label it with the level number if this node has not been visited before.

When the BFS terminates, we can compute the shortest path weight to any node \( v \) as the label it got in the BFS multiplied by \( w \). Clearly, the entire algorithm takes \( O(|V| + |E|) \) time.

3. Let \( A \) and \( B \) be the input matrices and let \( C \) be the product. We know that \( C[i, j] = \sum_{k=1}^{n} A[i, k] B[k, j] \). Since we have Boolean matrices, we can replace the addition operation with Boolean OR and multiplication with Boolean AND. We allocate \( n \) processors per each output element. The \( n \) processors allocated for \( C[i, j] \) first compute \( C_{ikj} = A[i, k] B[k, j] \) for \( k = 1, 2, \ldots, n \). This takes \( O(1) \) time. Followed by this, they compute the Boolean OR of the bits \( C_{i1j}, C_{i2j}, \ldots, C_{ijn} \) using the algorithm discussed in class. This also takes \( O(1) \) time.

4. We use the arbitrary-CRCW PRAM model. Each processor is assigned \( \log n \) input keys. In \( O(\log n) \) time we identify the distinct keys in the input by writing into a common cell \( M \). To begin with each processor picks its first key and tries to write it in \( M \). After this, each processor reads \( M \). The key in \( M \) is one of the distinct input keys and is stored. In the next time step, each processor picks (if any) one of its keys other than the one in \( M \) and tries to write it in \( M \). As a result, we get a second distinct key, and so on. Let the distinct keys be \( k_1, k_2, \ldots, k_c \) (where \( c \) is a constant).

We write all the input keys whose values are equal to \( k_1 \) in successive memory cells using a prefix addition operation (similar to the homework algorithm). Then we write keys whose values equal \( k_2 \) and so on. Each prefix operation takes \( O(\log n) \) time and there are \( O(1) \) such operations.

Thus the entire algorithm runs in \( O(\log n) \) time.

5. Note that any Boolean formula \( F \) in DNF is satisfiable iff at least one of its clauses is satisfiable. Also, a clause is satisfiable iff it does not have a literal and its negation. To check if a clause is satisfiable, we can sort the literals in it (using radix sort) and scan through the sorted list to see if a literal and its negation are in it. Thus, we can check if a clause is satisfiable in time that is linear in the length of this clause. As a result, we can decide if \( F \) is satisfiable in time that is linear in the length of \( F \). Therefore, the given problem is in \( \mathcal{P} \).

6. Let \( G(V, E); k \) be any instance of the CLIQUE problem. We create the following instance of INDSET: \( G'(V, E'); k \). Here \( G' \) is the complement of \( G \). Clearly, \( G'(V, E') \) can be constructed in \( O(n^2) \) time. Now we have to show that \( G \) has a clique of size \( k \) iff \( G' \) has an independent set of size \( k \).
Let $G$ have a clique of size $k$. Let the nodes that form a clique be $v_1, v_2, \ldots, v_k$. In $G'$ no two of these $k$ nodes will be connected by an edge. Therefore the same nodes $v_1, v_2, \ldots, v_k$ will form an independent set in $G'$.

Let $G'$ have an independent set of size $k$ and let the nodes that form an independent set in $G'$ be $u_1, u_2, \ldots, u_k$. It is easy to see that these $k$ nodes will form a clique in $G$.

This completes the proof that CLIQUE $\preceq$ INDSET.