1. (a) By definition, there are poly-time functions f and g such that $x \in A \Leftrightarrow f(x) \in B$ and $y \in B \Leftrightarrow g(y) \in C$, thus $x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C$. Obviously, $g(f(\cdot))$ is poly time (since $|f(x)|$ is polynomial in $|x|$).

(b) We can define a non-deterministic polynomial-time verifier for A, denoted as V-A: ① Take the input string y (the certificate) for A.
② Apply the reduction function f to y to obtain $f(y)$
③ Use the verifier for B, V_B as subroutine, and run $V(f(y))$.
The verifier V-A works in polynomial time, as it consists of a polynomial-time reduction f and the non-deterministic polynomial-time verifier V for B. If y is valid certificate for A, then there exists an x such that $f(x)=y$ since $x \in A \Leftrightarrow f(x) \in B$, the verifier V can be used to verify $f(y)$ in polynomial time. If y is not valid, there is no x such that $f(x)=y$ and the verifier $V(f(y))$ will reject. Therefore, A is in NP as we have constructed a non-deterministic polynomial-time verifier for A.

(c) Let x be an input to A and let $f(x)$ be the output of the reduction. To simulate $f(x)$ using a Turing machine, we can simply feed x into the machine that decides B, and then simulate the computation of $f(x)$ on a separate tape. This can be done in polynomial time, since f is a polynomial-time reduction. Therefore, if $A \leq_P B$, then $A \leq_T B$

2. (a) Proof is
The set of S of items that are chosen and the verification process is to compute $\Sigma_{i \in S} w_i$ and $\Sigma_{i \in S} v_i$, which takes polynomial time in the size of input

(b) One common NP-Hard problem used for such reductions is the SUBSET-SUM problem. Subset-Sum problem:

Given a set of positive integers $S = \{a_1, a_2, \ldots a_n\}$, a target integer T.

Given an instance of the Subset-Sum problem, we create an instance of the Knapsack problem as follows: ① Create a set of items, where each item corresponds to an element in set S where $\omega_i = a_i$ and $v_i = a_i$, $W = T$.

If there exists a subset of S add up to T, then there is corresponding solution to the Knapsack problem. If there exists a solution to the Knapsack problem, then there is a corresponding solution to the Subset-Sum Problem. This reduction is polynomial-time because constructing the Knapsack instance involves creating a set of items and copying values and weights, which can be done in polynomial time. Since Subset-Sum is NP-Hard, we have shown a polynomial-time reduction from Subset-Sum to Knapsack, Knapsack is also NP-Hard.

(c) It does not prove P=NP because our time complexity O(nC) is not actually polynomial as it takes the value of C in it. It depends on not only the size of our input n but also on the capacity of our sack C.

3. Since an NP Complete problem, by definition, is a problem which is both in NP and NP hard, the proof for the statement that a problem is NP Complete consists of two parts:

1) We know that Half-Clique is in NP since Clique is in NP.

2) Half-Clique Problem is NP Hard : To prove that the half-clique problem is NP Hard, we take the help of a problem which is already NP Hard and show that this problem can be reduced to the half-clique problem. For this, we consider the Clique problem, which is NP Complete. Every instance of the clique problem consisting of the graph G (V,E) and an integer k can be converted to the required graph G' (V',E') and k' of the half clique problem. The deduction that can be made is that the graph G' will have a clique of size n/2 , if the graph G has a clique of size k. Let m be the number of nodes in the graph G. We will now prove that the problem of computing the clique indeed boils down to the computation of the independent set. The reduction can be proved by the following two propositions:

- If k > = m/2, then for a constant number t, we add t nodes each of degree 0, for a graph G'. The graph G' has a total number of nodes equivalent to n = m + t, that is, the summation of all the nodes of graph G along with the extra nodes, such that it is equivalent to 2k, for any arbitrary value of k. Now k = n/2. This can be done by taking t = 2k -m. Then, the graph G has a clique of size k if and only if the graph G' has a clique of size k.
- If k < m/2 , then we add t additional nodes for the creation of graph G'. Edges can also be added from each new node to every other node in the graph. Therefore, any k-clique in G, for any arbitrary value of k combines with the t new nodes to make a (k+t)-clique in G', since edges have been added between each pair of vertices. A k+t-sized clique in G' must include at least k old nodes, which form a clique in the graph G. Therefore, the value of t is picked such that k+t = (m+t)/2, or t = m-2k, which makes the clique size in G' equivalent to n/2 exactly.

4.

To prove that 3SAT is polynomial-time reducible to the POSS-CONFIG problem, we need to provide a reduction from 3SAT to POSS-CONFIG. The reduction should transform an instance of 3SAT into an instance of POSS-CONFIG in polynomial time such that the answer to the 3SAT instance is "yes" if and only if the answer to the corresponding POSS-CONFIG instance is "yes."

Let's go through the reduction step by step:
Given a 3SAT instance with variables {v_1, v_2, ..., v_n} and clauses C_1, C_2, ..., C_m, we will create an instance of POSS-CONFIG with clubs that represent the variables and clauses.
1. Variables:
For each variable v_i, create two students, v_i_T (representing v_i = True) and v_i_F (representing v_i = False).
Create two clubs for each variable v_i: Club_v_i_T and Club_v_i_F.
2. Clauses:
For each clause C_j, create a club, Club_C_j.
For each literal in clause C_j, create a student for that literal. For example, if C_j contains the literals v_i, ¬v_k, and ¬v_l, create students C_j_v_i, C_j_~v_k, and C_j_~v_l3
3. Connecting students and clubs:
Enroll students representing literals in the corresponding clubs. For instance:
Enroll v_i_T in Club_v_i_T and ¬v_i_F in Club_v_i_T.
Enroll v_k_T in Club_v_k_F and ¬v_k_F in Club_v_k_F.
Enroll v_l_T in Club_v_l_F and ¬v_l_F in Club_v_l_F.
Enroll students representing literals in clause clubs. For example, if clause C_j contains v_i, ¬v_k, and ¬v_l, enroll the students C_j_v_i, C_j_~v_k, and C_j_~v_l in Club_C_j.
Set the desired configuration q for each club:
For each variable club Club_v_i_T, set q = 1.
For each variable club Club_v_i_F, set q = 1.
For each clause club Club_C_j, set q = 1.
Now, the 3SAT instance is reducible to the POSS-CONFIG instance, and the transformation is done in polynomial time.

If there exists a satisfying assignment for the 3SAT instance, we can choose the corresponding students (either v_i_T or v_i_F) for each variable and ensure that each clause club (e.g., Club_C_j) has at least one student from the set of students that make the clause true. This means q can be set to 1 for all clause clubs, and the configuration is possible.
If there is a possible configuration for the POSS-CONFIG instance, it implies that there exists a set of students for each variable club such that exactly one student is chosen for each variable club, and for each clause club, at least one student is chosen to satisfy the clause. This means there exists a satisfying assignment for the 3SAT instance.

5.

To prove that NP ⊆ EXP, we need to show that any efficiently verifiable language in NP is decidable in exponential time.

Let L be a language in NP, and let V be a polynomial-time verifier for L. That is, for any input x in L, there exists a certificate y such that V(x,y) accepts in polynomial time.
Consider the set of all possible certificates Y for inputs of size n. Since V runs in polynomial time, the number of possible certificates is at most 2^p(n) for some polynomial p(n), which is polynomial in n. This means that for any input x of size n, there are at most 2^p(n) effectively distinct certificates.
To decide whether x is in L, we can simply check all possible certificates y for x and see if V(x,y) accepts. Since there are at most 2^p(n) effectively distinct certificates, this can be done in time 2^p(n) * poly(n), which is exponential in n.
Therefore, we have shown that any efficiently verifiable language in NP is decidable in exponential time. Thus, NP ⊆ EXP.