1. If the weight on every edge of the graph is increased by the same amount, $T$ would still continue to be a MCST.

Proof: Consider any optimal algorithm to construct a MCST. Let $(e_1, e_2, \ldots, e_{|E|})$ be the order in which the edges are considered for constructing the MCST $T$. Even after the weight on every edge is increased by the same amount, this order will remain the same and hence the tree output will be the same.

2. The weight of each object is $\geq \epsilon m$. Since the knapsack capacity is $m$, a maximum of $1/\epsilon$ objects can be included in the solution. The total number of subsets with $1/\epsilon$ objects or less is $\sum_{i=1}^{1/\epsilon} \binom{m}{i} = O(n^{1/\epsilon})$. Compute the profit for each such subset and pick the best subset.

Complexity: $O(n^{1/\epsilon})$.

3. 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: $P(1, 1), P(1, 2), \ldots, P(1, K), P(2, 1), P(2, 2), \ldots, P(2, K), \ldots, P(n, 1), \ldots, P(n, K)$.

4. Use BFT to find the connected components of the given graph. Fill the transitive closure matrix $A^*$ as follows: $A^*(i, j) = 1$ if either $i = j$ or $i \neq j$ and $i$ and $j$ are in the same connected component.

Total complexity = complexity of BFT + complexity of filling the $A^*$ matrix = $O(|V| + |E|) + O(|V|^2) = O(|V|^2)$.

5. Consider a complete binary tree with $k$ leaves where each leaf has one of the input polynomials. Perform a computation up the tree as follows. Each internal node multiplies the two children polynomials and sends the result to its parent. When the root completes its operation we get the product of the $k$ polynomials. There are $\log k$ levels in the tree and the time spent at each level is $O(n \log n)$. Thus the run time of the algorithm is $O(n \log n \log k)$.

6. Convert the polynomial $f(x)$, into the following form (into $n/m$ groups):

$$f(x) = x^{n-m}(a_n x^m + a_{n-1} x^{m-1} + \ldots + a_{n-m+1} x) + x^{n-2m}(a_{n-m} x^m + a_{n-m-1} x^{m-1} + \ldots + a_{n-2m+1} x) + \ldots$$

where $f_1(x), f_2(x), \ldots, f_{n/m}(x)$ are polynomials of degree $m$. $f(x)g(x) = x^{n-m}f_1(x)g(x) + x^{n-2m}f_2(x)g(x) + \ldots + f_{n/m}(x)g(x)$ requires multiplying $(n/m)$ pairs of polynomials of degree $m$ each and combining the $n/m$ resulting polynomials into one polynomial of degree $n + m$. The second step requires a scan of all the $n/m$ polynomials of degree $2m$.

Total time taken is $O(\frac{n}{m}(m \log m)) + O(\frac{n}{m}(2m)) = O(n \log m)$.