1. Consider the **partition problem**: given \( n \) positive integers, partition them into two disjoint subsets with the same sum of their elements. (Of course, the problem does not always have a solution.) Design an exhaustive-search algorithm for this problem. Try to minimize the number of subsets the algorithm needs to generate.

2. Consider the **clique problem**: given a graph \( G \) and a positive integer \( k \), determine whether the graph contains a clique of size \( k \), i.e., a complete subgraph of \( k \) vertices. Design an exhaustive-search algorithm for this problem.

3. Recall the **Knapsack problem**: Given \( n \) items with known weights, \( w_1, w_2, ..., w_n \) and values \( v_1, v_2, ..., v_n \) and a knapsack capacity \( W \), find the most valuable subset of items that will fit into the knapsack.

   We talked about a brute force solution that checks all subsets of the \( n \) items, but it requires exponential time. Consider the following approximation algorithm for the problem. Give a counterexample to demonstrate that the algorithm does not give the optimal (maximum) answer.

   ```
   Sort the items in decreasing order by value
   remaining_weight ← W
   total_value ← 0
   item_set ← {} 
   for each item \( i \) in decreasing order by value do 
     if \( w_i \leq \) remaining_weight then 
       item_set ← item_set ∪ \{i\} 
       remaining_weight ← remaining_weight - \( w_i \) 
       total_value ← total_value + \( v_i \) 
     end 
   end 
   
   Algorithm 1: Approximation algorithm for 0/1 Knapsack
   ```