1. Input is a (not necessarily sorted) sequence \( S = k_1, k_2, \ldots, k_n \) of \( n \) arbitrary numbers. Consider the collection \( C \) of \( n^2 \) numbers of the form \( \min\{k_i, k_j\} \), for \( 1 \leq i, j \leq n \). Present an \( O(n) \)-time and \( O(n) \)-space algorithm to find the median of \( C \).

2. Two sets \( A \) and \( B \) have \( n \) elements each. Assume that each element is an integer in the range \([0, n^{100}]\). These sets are not necessarily sorted. Show how to check whether these two sets are disjoint in \( O(n) \) time. Your algorithm should use \( O(n) \) space.

3. Let \( \hat{F}(I) \) be the value of the solution generated on problem instance \( I \) by \textbf{GreedyKnapsack} when the objects are input in nonincreasing order of the \( p_i \)'s. Let \( F^*(I) \) be the value of an optimal solution for this instance. How large can the ratio \( F^*(I) / \hat{F}(I) \) get?

4. Find a minimum spanning tree for the following graph \( G(V, E) \) either using Prim’s algorithm or using Kruskal’s algorithm: \( V = \{1, 2, 3, 4, 5\} \). The edge weights are: \( W(1, 2) = 11; W(1, 4) = 2; W(1, 3) = 2; W(2, 3) = 5; W(3, 4) = 4; W(3, 5) = 5; W(4, 5) = 7 \).

5. Use Dijkstra’s algorithm to solve the single source shortest path problem on the directed graph \( G(V, E) \): \( V = \{s, 1, 2, 3, 4, 5\} \). Edge weights are: \( W(s, 1) = 2; W(s, 2) = 15; W(1, 3) = 6; W(1, 4) = 3; W(2, 4) = 4; W(2, 5) = 2; W(3, 4) = 2; W(4, 2) = 5; W(4, 3) = 1; W(4, 5) = 5 \).

6. Solve the following instance of the 0/1 knapsack problem using dynamic programming:

\[
\begin{array}{c|c|c|c|c}
\text{Weight} & 1 & 2 & 3 & 2 \\
\hline
\text{Profit} & 10 & 15 & 25 & 12
\end{array}
\]

The capacity of the knapsack \( m = 5 \).

7. Let \( A \) be the adjacency matrix of a directed graph \( G \). The reflexive transitive closure \( A^* \) is a matrix with the property \( A^*(i, j) = 1 \) iff \( G \) has a path, containing zero or more edges, from \( i \) to \( j \). \( A^*(i, j) = 0 \) otherwise. Present an \( O(M(n) \log n) \) time algorithm to compute \( A^* \), where \( M(n) \) is the time needed to multiply two \( n \times n \) boolean matrices.