1. Let S be a collection of seven objects with benefit-weight values $(12, 4)$, $(10, 6)$, $(8, 5)$, $(11, 7)$, $(14, 3)$, $(7, 1)$, and $(9, 6)$. What is an optimal solution to the Fractional Knapsack problem for S assuming we have a sack that can hold objects with total weight 18?

2. Determine the complexity of finding the maximum-benefit subset in the Fractional Knapsack problem.

3. The start-time and the finish-time of a job is denoted by a pair (s, f). Given that each machine can handle only one job at a time, how many machines are required to handle seven jobs whose start finish times are $(1, 3)$, $(1, 4)$, $(2, 5)$, $(3, 7)$, $(4, 7)$, $(6, 9)$, and $(7, 8)$.

4. Solve the following recurrence equations using the Master Method. In each of the given equations the value of $T(n)$ is constant c, if the value of n is less than d.

   - $T(n) = 2T(n/2) + \log n$
   - $T(n) = 8T(n/2) + n^2$
   - $T(n) = 16T(n/2) + (n \log n)^4$
   - $T(n) = 7T(n/3) + n$
   - $T(n) = 9T(n/3) + n^3 \log n$

5. Let us recall the problem of paying money using minimum number of coins. Give an example set of denominations so that the greedy change

making algorithm (1) will use the minimum number of coins, (2) will not use the minumum number of coins.

6. A round-robin tournament is a collection of games in which each team plays each other team exactly once. Given a set P of n teams, describe an algorithm for planning a round-robin tournament. You may assume that n is a power of 2.

7. Design a divide-and-conquer algorithm for finding the minimum and maximum element of n numbers using no more that $3n/2$ comparisons.

8. Can we multiply two binary numbers of length n is sub-quadratic time? If yes, then prove it.

9. Suppose we are given a collection $A = \{a_1, a_2, \cdots, a_n\}$ of n positive integers that add up to N. Design an $O(nN)$-time algorithm for determining whether there is a subset $B \subset A$, such that $\sum_{a_i \in B} a_i = \sum_{a_i \in (A-B)} a_i$.

10. Suppose we are given an n-node rooted tree T, such that each node v in T is given a weight $w(v)$. An independent set of T is a subset S of the nodes of T such that no node in S is a child or parent of any other node in S. Design an efficient algorithm to find the maximum-weight independent set of the nodes in T, where the weight of a set of nodes is simply the sum of the weights of the nodes in the set. What is the running time of your algorithm.