1. Consider the following algorithm for the MAX-CUT problem.

A. Start with an arbitrary cut (S, T) of V.

B. So long as possible, repeat:

(i) If there exists $u \in S$ such that the cut $(S - u, T + u)$ has more cross edges than (S, T), delete u from S, and add u to T.
(ii) If there exists $v \in T$ such that the cut $(S + v, T - v)$ has more cross edges than (S, T), add v to S, and delete v from T.

C. Return (S, T).

(a) Prove that this algorithm terminates in polynomial time.

(b) Prove that the approximation ratio of this algorithm is 1/2.

(c) Prove that this approximation ratio is tight.

2. Consider the following algorithm for EUCLIDEAN-TSP.

a) Select the pair (u, v) such that $c(u, v)$ is smallest among all pairs.
Start with the tour $C = (u, v)$.

b) Repeat until C is a Hamiltonian cycle:

Find $i \in C$ and $j \notin C$ such that $c(i, j)$ is the minimum.
Let k be the city next to i in C.
Replace i, k by i, j, k in C.

Prove that this is a 2-approximation algorithm.

3. [Next-fit strategy for BIN-PACKING] Keep on adding the items to the most recently opened bin so long as possible. When the insertion of the next item lets the capacity of the current bin exceed, close the current bin, and open a new bin.

(a) Prove that this is a 2-approximation algorithm.

(b) Prove that the approximation ratio of 2 is tight, that is, given any $\varepsilon > 0$, there exists a collection for which the approximation ratio is $> 2 - \varepsilon$.

4. Assuming that $P \neq NP$, prove that the BIN-PACKING problem cannot have a ρ-approximation algorithm for any $\rho < 3/2$.

5. An algorithm is called pseudo-polynomial-time if it runs in time polynomial in the size of the input expressed in unary. An NP-Complete problem is called *weakly NP-Complete* if it admits a pseudo-polynomial-time algorithm. For example, we have seen that the KNAPSACK problem is weakly NP-Complete.

Prove that the SUBSET-SUM problem is weakly NP-Complete.

6. Let A = (a_1, a_2, . . . , a_n) be an array of n positive integers, and t a target sum (a positive integer again). The task is to find a subset $I \subseteq \{1, 2, 3, . . . , n\}$ for which the sum of a_i for $i \in I$ is as small as possible but at least as large as t. Assume that $t \le a_1 + a_2 + \cdot\cdot\cdot + a_n$ (otherwise the problem has no solution). Prof. Sad proposes the following algorithm to solve this problem.

Initialize sum = 0, and I = 0.
for i = 1, 2, 3, . . . , n (in that order), repeat {
  Update sum = sum + a_i , and $I = I \cup \{i\}$.
  If sum $\ge t$, break.
}
Return I.

(a) Prove that the approximation ratio of Prof. Sad's algorithm cannot be restricted by any constant value.

(b) Prof. Atpug suggests sorting the array A in the ascending order before running the algorithm. In view of this suggestion, we now have $a_1 \le a_2 \le a_3 \le \cdot\cdot\cdot \le a_n$. Prove that Prof. Atpug's suggestion makes Prof. Sad's algorithm a 2-approximation algorithm.

7.

You are given a stream of songs lasting for $t_1, t_2, t_3, \ldots, t_n$ seconds. You have two write-once devices D_1 and D_2, each capable of storing songs of total duration T seconds. When the i-th song starts, you have three options: (i) copy the song to D_1, (ii) copy the song to D_2, and (iii) discard the song. The copy of a song to a device is allowed only when there is enough memory left in that device. Assume that the individual song durations t_i are known to you beforehand, and these durations satisfy $t_i \leqslant T$ for all i, and $\sum_{i=1}^{n} t_i \leqslant 2T$. Your goal is to maximize the total copy time in the two output devices together. To that effect, you run the following greedy algorithm. Derive a tight approximation ratio of the algorithm. Prove that the approximation ratio is tight.

```
for i = 1,2,3,...,n (in that order) {
    If D_1 can accommodate the i-th song, copy the i-th song to D_1,
    else if D_2 can accommodate the i-th song, copy the i-th song to D_2,
    else discard the i-th song, and continue.
}
```

8. Let $G = (V, E)$ be a connected undirected graph. You make a DFS traversal of G starting from any arbitrary vertex. Let T be the DFS tree produced by the traversal. Take C to be the set of all internal (that is, non-leaf) nodes in T.

(a) Prove that C is a vertex cover for G.

(b) Prove that determining C using this method is a 2-approximation algorithm for the MIN-VERTEX-COVER problem.

9. Consider the knapsack problem. First, sort the objects in decreasing order of p_i / w_i . Call this sorted list $O_1, O_2, \ldots, O_n$. Assume that $w_1 + w_2 + \cdots + w_n > C$ (otherwise the solution is trivial). Let k be the index such that $w_1 + w_2 + \cdots + w_k \leq C$, but $w_1 + w_2 + \cdots + w_{k+1} > C$. Now, let j be the index such that p_j is maximum. Augment the greedy algorithm to output $\{1, 2, \ldots, k\}$ or $\{ j \}$, whichever gives better profit. Prove that this is a 1/2-approximation algorithm for the knapsack problem.