Total marks: 50 (10 points for each question)

1. (a) Before we give the algorithm, note that if we resolve on two clauses with two literals each, then the result is also a two-literal clause; the resolvent of $\{l_1, l\}$ and $\{\bar{l}, l_2\}$ is $\{l_1, l_2\}$. The algorithm is simple: systematically produce all possible resolvents.

   There are at most $\binom{2m}{2} + 2m = \frac{2m(2m-1)}{2} + 2m = m(2m-1) + 2m = 2m^2 + m$ resolvents, for m clauses, because each resolvent consists of two literals, or it consists of a single literal, and there are $2m$ literals in m clauses.

   The algorithm might have several stages: in the first stage it produces the resolvents of the original clauses; in the second stage it produces the resolvents from the original clauses, and the clauses from the first stage, etc. If at some point the empty clause is obtained, the original set of clauses is unsatisfiable; if, however, there are no more new resolvents, and the empty clause has not been derived, the original set of clauses must be satisfiable (by the **Fact**).

   Estimate of the running time: we know that there are at most $2m^2 + m$ resolvents, so together with the original clauses we will have at most $2m^2 + m + m = 2(m^2 + m)$ $(O(m^2))$ clauses throughout the procedure. Thus, everytime we resolve, we process pairs out of these $2(m^2 + m)$ clauses. So a rough estimate of the running time is $O(m^4)$.

   (b) We cannot use this method effectively for 3-SAT because the size of the resolvent might be bigger then the original clauses: $\{l_1, l_2, l\}$ and $\{\bar{l}, l_3, l_4\}$ produce $\{l_1, l_2, l_3, l_4\}$, so we cannot bound the number of resolvents as we did above, and in fact, in general there might be exponentially many resolvents in the number of original clauses.

2. (a) This was shown in class (the idea is to start colouring, until we have reached a conflict; output "NO", or until all vertices have been assigned colours; output "YES").

   (b) For a particular fixed $k \geq 3$, we reduce 3-COL to k-COL as follows: add $k-3$ new vertices, $V_{\text{new}} = \{v_1, \ldots, v_{k-3}\}$. Also add the following edges: $(V_{\text{new}} \times V_{\text{new}}) \cup (V_{\text{old}} \times V_{\text{new}})$, that is, edges among every pair of new vertices, and edges between every old vertices and every new vertex. To see that this works, just note that the set V_{new} is a clique, so it requires $k-3$ different colours; the remaining 3 colours must be used to colour V_{old}.

3. (a) NP-complete. For example 3SAT $\leq$ SAT, where:

$$\{l_1 \vee l_2 \vee l_3\} \longmapsto \{l_1 \vee l_2 \vee l_3 \vee p\}, \{l_1 \vee l_2 \vee l_3 \vee \bar{p}\}$$

   (b) NP-complete. Even though this is a TSP-like problem in unary, it is still NP-complete. The reason is because HC $\leq_p$ UNARY-VISIT-ALL-ROUTING: given an instance $G = (V, E)$ to HC, let l be the distance matrix $l(i,j) = 1$ if $(i,j) \in E$, and 2 otherwise (11 in unary), and let $D = |V|$ and $K = 1$. Then, G has a Hamiltonian Cycle iff there is a route of total weight at most D.

   (c) In P. There are efficient algorithms (Dijkstra's or Floyd's) for computing the shortest path from s to t. If the shortest path has length $l \leq B$, we answer "YES".

   (d) In P. We use Kruskal's algorithm with the edges ordered from the most expensive to the cheapest. We answer "YES" if the cost of the tree that we obtain is $\geq B$.

   (e) NP-complete. HC $\leq_p$ SEATING-DIPLOMATS: given a graph $G = (V, E)$, let V be the set of diplomats, and define (u, v) to like each other iff $(u, v) \in E$.

4. If $\{f(x) | x \in \Sigma^*\}$ is finite, then it is decidable.

   Suppose that it is infinite: We want a procedure that for any given $w \in \Sigma^*$, it decides if $w \in \{f(x) | x \in \Sigma^*\}$. The procedure is the following: let $x_1, x_2, x_3, \ldots$ be a listing of all the strings in Σ^* in such a way that $|x_i| \le |x_{i+1}|$ (for example, a lexicographic ordering of Σ^*). For a given w, compute $f(x_1), f(x_2), f(x_3), \ldots$ until one of the following happens: $w = f(x_i)$ for some i; we stop and accept, or we reach some x_i such that $|f(x_i)| > |w_i|$; we reject, because by the property of f, and by our ordering of Σ^*, if $j \ge i$, then $|x_j| \ge |x_i|$, so $|f(x_j)| \ge |f(x_i)| > |w_i|$, so we know that we are never going to encounter some x_j, $j > i$, such that $f(x_j) = w$. Note that since $\{f(x)|x \in \Sigma^*\}$ is infinite, if w is not in the set, then we will *always* find an x_i such that $|f(x_i)| > |w_i|$.

5. (a) Let $s_1, s_2, \ldots$ be an enumeration of all strings. Consider the following algorithm: in input $\langle M_1, M_2 \rangle$ it starts two loops, the outer loop is $t : 1..\infty$ and the inner loop is $i : 1..t$. For each pair (t, i), it simulates M_1 and M_2 for t steps on input s_i. If M_1 accepts s_i and M_2 rejects s_i during this simulation, then ACCEPT. This program accepts $\langle M_1, M_2 \rangle$ iff $\langle M_1, M_2 \rangle \in L$.

   (b) We show that $\overline{E} \le_m L$, where $E = \{\langle M \rangle | L(M) = \emptyset\}$. Given an input $\langle M \rangle$ to E, we construct an input $\langle M_1, M_2 \rangle$ to L such that: $\langle M \rangle \in \overline{E}$ iff $\langle M_1, M_2 \rangle \in L$. Choose a fixed M_2 such that $L(M_2) = \emptyset$. Choose $M_1 = M$. Then $\langle M \rangle \in \overline{E}$ iff $L(M) \ne \emptyset$ iff M accepts w for some w iff $\langle M_1, M_2 \rangle \in L$.