Name ________________________ Student No. ________________________

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.

Total Marks: 40

1. Recall that the decision problem k-COLOR is defined as follows:
Input: A graph $G = (V, E)$.
Question: Can the nodes of G be colored using at most k colors so that adjacent nodes have different colors?

[6] (a) Prove that 2-COLOR is in P.

We provide a polytime algorithm. Suppose our colors are red and blue. The idea is to color each connected component of the graph (if possible).

Pick a node v_0 in the first connected component, and color it red. If v_0 is connected to $v_1, v_2, \ldots, v_k$, then color all these nodes blue, and continue spreading out by coloring all the neighbors of $v_1, v_2, \ldots, v_k$ red, and so on. The point is that once you color a node, you have no choice of color for its neighbors, so you will either be able to color all nodes in a connected component, or somewhere, along the way, you will get a conflict, in which case you know that the graph is not 2-colorable.

If the algorithm colors all connected components without a conflict, it outputs "YES". Otherwise, as soon as a conflict arises, it outputs "NO". If the input is presented as an adjacency matrix (the usual way to present graphs) the algorithm runs in time $O(|V|^2)$.

[4] (b) We know that 3-COLOR is NP-complete, and assume that P $\neq$ NP. Explain, briefly, why it *is not* true that 3-COLOR $\leq_p$ 2-COLOR.

Suppose that 3-COLOR $\leq_p$ 2-COLOR. Then, since 2-COLOR is in P (by part 1(a)), we know that 3-COLOR is in P as well. But then, since 3-COLOR is NP-complete, it follows that P = NP, which contradicts our assumption.

[4] (c) Is it true that 2-COLOR $\leq_p$ 3-COLOR ? Explain briefly.

Yes, it is true. One way to see it is the following: since 2-COLOR is in P, and P $\subseteq$ NP, it follows that 2-COLOR is in NP, and hence, by definition of NP-completeness, 2-COLOR $\leq_p$ 3-COLOR.

2. Recall that VC is an NP-complete decision problem defined as follows:
Input: Graph $G = (V, E)$, positive integer B.
Question: Does G have a vertex cover of size B?

Let ODD-VC be the same as VC, except that B must be odd. Show that ODD-VC is NP-complete as follows:

[3] (a) Show that ODD-VC is in NP.

We provide a polytime "guess-and-check" algorithm:
guess a set V_0 of vertices, $|V_0| = B$.
check that every edge has at least one end-point in V_0. Note that this can clearly be done in polytime in the size of the input.

[6] (b) Show that VC $\leq_p$ ODD-VC. State the correctness condition of your reduction. (You do not have to prove the correctness condition).

Given an input $I = \langle G = (V, E), B \rangle$ to VC, we transform it (in polytime) to an input $I' = \langle G' = (V', E'), B' \rangle$ to ODD-VC as follows:

$$\begin{cases} V' = V,\ E' = E,\ B' = B & \text{if } B \text{ is odd} \\ V' = V \cup \{a, b\},\ E' = E \cup \{(a, b)\},\ B' = B + 1 & \text{if } B \text{ is even} \end{cases}$$

where a and b are *new* nodes.

Correctness condition: $I \in \text{VC} \iff I' \in \text{ODD-VC}$.

3. For each of the following assertions, state whether it is true or false, and justify it. If it is true, justify it with a proof; if it is false, justify it with a counter-example. (No marks for a guess without justification).

[4] (a) For all languages L_1 and L_2, if $L_1 \leq_m L_2$ and L_2 is semi-decidable, then L_1 is also semi-decidable.

Circle one: $\boxed{\text{True}}$ False

Justification:

Since $L_1 \leq_m L_2$, there exists a computable function f such that $x \in L_1 \iff f(x) \in L_2$. Since L_2 is semi-decidable, there exists a TM M_2 such that $L_2 = L(M_2)$. We design M_1 so that $L_1 = L(M_1)$ as follows: on input x, M_1 computes $f(x)$, and simulates M_2 on $f(x)$, and accepts if M_2 accepts.

[4] (b) For all languages L_1 and L_2, if $L_1 \subseteq L_2$ and $L_2 \in \mathrm{P}$, then $L_1 \in \mathrm{P}$.

Circle one: True $\boxed{\text{False}}$

Justification:

Let $L_2 = \Sigma^*$. Then, trivially $L_2 \in \mathrm{P}$. But $A_{\mathrm{TM}} \subseteq \Sigma^* = L_2$, and A_{TM} is not even decidable, let alone in P.

[4] (c) For all languages L, if both L and $\overline{L}$ are semi-decidable, then L is decidable.

Circle one: $\boxed{\text{True}}$ False

Justification:

Since L and $\overline{L}$ are semi-decidable, there exist TMs M_1 and M_2 such that $L = L(M_1)$ and $\overline{L} = L(M_2)$. We design M_3 so that $L = L(M_3)$ and M_3 halts on all inputs:

> On input x, M_3 simulates M_1 and M_2 "in parallel" on input x. If M_1 accepts, then M_3 accepts, if M_2 accepts, then M_3 rejects.

Clearly $L = L(M_3)$. Since $L \cup \overline{L} = \Sigma^*$, M_1 or M_2 always accepts, so M_3 always halts.

[5] (d) Let L be a language. Suppose that $L = L(E)$ where E is an enumerator that outputs the strings of L in order of their length, i.e. it outputs first all the strings of length 1 (if any), then all the strings of length 2 (if any), and so on. Then, L is decidable.

Circle one: $\boxed{\text{True}}$ False

Justification:

If L is finite, then L is decidable. Otherwise, we design a TM M based on E such that $L = L(M)$ and M halts on every input:

> On input x, M simulates E on a blank tape. If E outputs x, then M accepts. If E outputs a string y such that $|y| > |x|$, then M rejects.

Clearly $L = L(M)$. To see that M always halts, note that for any fixed string x there are finitely many strings y such that $|y| \leq |x|$. So, since L is infinite, at some point E is going to output a y such that $|y| > |x|$, after which we know for sure that E will never output x.