1. Consider the following operation on languages:

\[ \square L \overset{\text{def}}{=} \{xx \mid x \in L\} \]

(a) Prove that \( \mathcal{L}_3 \), the class of regular languages, is not closed under this operation.

\[ 14 \text{ marks} \]

\textbf{Hint:} Consider the language \( \{a^ib \mid i \in \mathbb{N}\} \).

(b) Prove that if \( L \in \mathcal{L}_3 \), we cannot even guarantee that \( \square L \in \mathcal{L}_2 \), the class of context free languages.

\[ 14 \text{ marks} \]

\textbf{Hint:} Consider the language \( \{a^ib^j \mid i, j \in \mathbb{N}\} \).

(c) Hence or otherwise prove that \( \mathcal{L}_2 \) is not closed under this operation.

\[ 5 \text{ marks} \]

2. (a) By using the pumping lemmas and constructing appropriate machines, find the smallest class in the Chomsky hierarchy to which the following language belongs: \( \{a^ib^j c^k \mid j = i + k\} \).

\[ 11 \text{ marks} \]

(b) By giving an appropriate construction or machine algorithm, show that \( \{a^ib^j c^k \mid i \neq j, i \neq k, j \neq k\} \) is in \( \mathcal{L}_1 \) — the class of context-sensitive languages.

\[ 10 \text{ marks} \]
(c) Give constructions to show that the classes of recursive and recursively enumerable languages are closed under language reversal:

\[ L^R \overset{\text{def}}{=} \{ x : \Sigma^* \mid x^R \in L \} \]

[12 marks]

3. A counting automaton is a non-deterministic finite state automaton but which can ‘remember’ an integer. The automaton starts off with its memory set at a particular value and every transition will add or subtract a number from the currently remembered value.

We depict counting automata in a similar fashion as finite state automata, but we add some extra information about the transitions:

- The start state now has an integer associated with it. This represents the starting value of the machine’s memory bank:

- Each transition also modifies the value of the memory by adding (or subtracting) a value from it. Transitions thus have a pair \((a, n)\) as a label, denoting the expected input and value to add to the memorised number respectively:

- Counting automata have no terminal states.

An example of a counting automaton is shown below:
(a) Formalise the concept of a counting automaton.  

(b) Define what we mean by the configuration of a counting automaton.

A string $w$ is said to be accepted by a counting automaton if, starting from the initial state with input $w$ the machine can finish with zero in its memory bank.

The automaton depicted earlier thus accepts all strings of the form $\{a^{2n}b^n \mid n \in \mathbb{N}\}$.

(c) Formalise the meaning of $T(M)$, the language accepted by counting automaton $M$.

(d) Construct a counting automaton which accepts strings over the alphabet $\{a, b\}$, of the form $a^i b^i$.

4. Prove that the following problem is $NP$-complete.

**Total Vertex Covering (TVC)**

**Instance:** Given a graph $G$ with $n$ nodes.

**Question:** Is there a total vertex cover of size $k$ in graph $G$?

A total vertex cover is a subset of vertices $V' \subseteq V$ such that:

- for every edge $(v, w)$ in the graph at least one of $v$ and $w$ is in $V'$ and
- unconnected nodes (nodes which are not the source or the destination of any edge) are in $V'$.

Hence, a total vertex covering is a vertex covering which also includes isolated nodes.

Clearly state any results proved in the course notes which you use.
Show how the result can be extended to prove the \( NP \)-completeness of the following problem:

**Low Cost Total Vertex Covering (LCTVC)**

**INSTANCE:** Given a graph \( G \), a cost function \( c \) associating each vertex with a (non-negative numeric) cost \( r \).

**QUESTION:** Is there a total vertex cover which costs no more than \( r \) in graph \( G \)?

[13 marks]

5. Prove that the following problem is \( NP \)-complete.

**Long Paths (LP)**

**INSTANCE:** Given a graph \( G = (V, E) \) and two positive integers \( l \) and \( n \) (such that \( l \leq |E| \)).

**QUESTION:** Are there \( n \) distinct simple paths in \( G \) each with \( l \) or more edges?

A simple path is a path which encounters no node more than once. Two paths are distinct if they do not follow the exact sequence of nodes (ie they may pass through common nodes).

Carefully consider every part of your proof. Clearly state any results proved in the course notes which you use.

[33 marks]