1. Answer True or false, tick the right choice or fill up as necessary - no proofs needed. (Negative 2 for each incorrect answer to the True/False and multiple choice type.)

   (a) The minimum state PDA to recognize the language strings not of the form $w \cdot w$ where $w \in (0 + 1)^*$ has one states. \boxed{2}

Since every PDA can be reduced to a 1 state machine when transformed from the equivalent CFL.

   (b) A shuffle of two strings $x, y \in \Sigma^*$ denoted by $x||y$ is the set of strings that can be obtained by interleaving the strings $x$ and $y$ in any manner. For example $ab||cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$. (The strings need not be of the same length.) For two sets of strings $A, B$, the shuffle is defined as $A||B = \bigcup_{x \in A, y \in B} x||y$. \boxed{4}$\times$2

   (i) If $L_1$ and $L_2$ are regular then $L_1||L_2$ is regular. \boxed{True}

We can define an NFA with transition function $\delta([q_1, q_2], a) = \{[\delta_1(q_1, a), q_2], [q_1, \delta_2(q_2, a)]\}$ where $\delta_1, \delta_2$ correspond to the transition function of $L_1, L_2$.

   (ii) If $L_1$ and $L_2$ are CFL, then $L_1||L_2$ is CFL. \boxed{False}

If $L_1 = a^n b^n$ and $L_2 = c^n d^n$ then $a^i c^j b^i d^j \in L_1||L_2$. If this is CFL then we can create strings with unequal number of a’s and b’s using P.L. that should not belong to $L_1||L_2$.

   (iii) If $L_1, L_2$ are recursively enumerable then $L_1||L_2$ is r.e. \boxed{True}

We can try all partitions of the given string and run the machine for $M_1$ and $M_2$ and accept if one of the partitions is accepted. We can assume that the machines are non-deterministic to avoid any complications of non-halting.

   (iv) If $L_1$ is CFL and $L_2$ is regular then $L_1||L_2$ is CFL. Like part (i) the PDA can maintain the states corresponding to both languages and one stack and has a choice of either making a transition according to $L_1$ or $L_2$.

   (c) Consider the following instance of the Post Correspondence Problem: $A_1 = 100, A_2 = 0, A_3 = 1$ and $B_1 = 1, B_2 = 100, B_3 = 00$. Compute a solution to the above instance and report the sequence of indices or answer No solution possible. \boxed{3}

It can be verified that the shortest sequence is 1,3,1,3,2,2

   (d) Classify the following languages under recursive/r.e. but not recursive/ not r.e. \boxed{3}$\times$2

   i. \{ $< M > | M$ takes fewer than 1000 steps for some input \} \boxed{recursive}

   ii. \{ $< M > | M$ takes fewer than 1000 steps for at least 1000 inputs \} \boxed{recursive}

   iii. \{ $< M > | M$ takes fewer than 1000 steps for all inputs \} \boxed{recursive} The answer is NO for any interesting language.
Since the number of steps is bounded, an $M_u$ can simulate it for some fixed number of steps and depending on whether it is accepted, it can decide for a given string. Note that no string with length $> 1000$ with satisfy this condition, so it will cycle through a maximum of $|\Sigma|^{1000}$ strings. For any longer string, the answer can be deduced on the basis of the prefix - if the tape head never reaches the 1001-th cell of the input, then the answer is YES, i.e., it is never trying to read the 1001 input position. The above argument generalizes to any $k$-tape machine.

(e) Every infinite recursively enumerable language has an infinite recursive subset. True (3)

Consider a generator for any infinite r.e. language - we print the next string that is larger than the previous number.

(f) Describe a CFG in Chomsky Normal Form for the set of strings $a^i b^j i \neq j$. (3)

\[
S \rightarrow AS_1 \mid S_1 B \mid A \mid B \quad S_1 \rightarrow X_a S_2 \quad S_2 \rightarrow S_1 X_b \\
A \rightarrow AX_a \mid a \quad B \rightarrow BX_b \mid b \quad X_a \rightarrow a \quad X_b \rightarrow b
\]

If the grammar is not in CNF then marks were deducted - saying that it can be converted in not acceptable for this problem.

(g) If $L_1 \leq_f L_2$ where $f$ uses deterministic logarithmic space then - If $L_2$ has a deterministic polynomial time algorithm then $L_1$ has a \textbf{det polynomial} time algorithm. (2)

(h) If $L \in \text{NSPACE}(f(n))$ then $L \in \text{NTIME}(2^{O(f(n))})$. (2)

(i) Determining if a given graph has a vertex cover of size 25 is NPC. \textbf{False} (2)

This can be checked in $(\binom{n}{25})$ steps by verifying all subsets of size 25.

(j) If the 3 colorability problem has an algorithm taking $2^{\log^2 n}$ time then the 3-SAT problem has $O(2^{\log^2 n}) = O(n^{O(\log n)})$ time algorithm. (2)

(k) A boolean formula is called a \textit{tautology} if it evaluates to True for all truth assignments. Then, Tautology belongs to \textbf{co-NP}. (2)

2. Consider the relation $L_1 \subset L_2 \subset L_3$. Is it possible that $L_1$ and $L_3$ are non-recursive and $L_2$ is recursive? Justify. Assume that $L_i \subset \Sigma^*$, i.e., unary alphabet. (10)

Remark: Answer not valid if only one of the relations is satisfied.

Consider any non-recursive r.e. language, say $L_u$ (over unary alphabet). Consider $L_1 = \{2x \mid x \in L_u\}$. Let $L_2 = \{2i \mid i \text{ is an integer}\}$. Define $L'_u = \{2x + 1 \mid x \in L_u\}$ and $L_3 = L_2 \cup L'_u$.

Note that $L_1$ is non-recursive, as $L_u \leq L_1$. Similarly $L_3$ is also non-recursive as $L_u \leq L'_u$. Indeed if $L_3$ is recursive, then $L'_u = L_3 \cap L_o$ is recursive as $L_o = \{2i + 1\}$ is recursive.

Many tried to show impossibility which is clearly not correct because of the given example. Many attempts were about designing Turing machines with some clever stopping conditions but these have to be backed up by the relationships between the languages accepted.
3. Let \( L \in \mathcal{P} \) (deterministic polynomial time). Which of the following statements are true? Justify.

(i) \( \bar{L} \in \mathcal{P} \) True

(ii) \( L^* \in \mathcal{P} \). True

Say there is a \( g(n) \) steps TM that accepts \( L \) where \( g \) is a polynomial.

(i) Note that \( x \notin L \) iff \( x \) is not accepted within \( g(|x|) \) steps and this can be easily counted. So \( \bar{L} \in \mathcal{P} \). Most answers used an incorrect definition of acceptance, i.e., TM will either accept or reject and therefore swap the accepting condition. Note that the time/space bound is only for strings in the language, so the complement of the language requires a different stopping criteria. Only 1 marks was deducted.

(ii) \( x \in L^* \) iff \( x = x_1 \cdot x_2 \ldots x_k \) where \( |x_i| \geq 1, k = |x| \) and \( x_i \in L \). One can design a dynamic prog based algorithm similar to CYK to do the job in polynomial time.

One common mistake was not accounting for the partitions or counting the number of partitions incorrectly. A common fallacy was claiming that the number of partitions is polynomial - note that \( \binom{n}{n/2} \geq 2^{n/2} \).
4. Consider a $S(n)$ space bounded non-deterministic Turing Machine $M$. Let $m$ denote the maximum number of distinct IDs in any valid sequence of computation.

Consider a deterministic TM $M'$ that recognizes the same language by simulating $M$ using its storage tapes. However, it doesn’t want to store all the $m$ IDs along any computational trail to save space. To check if the initial ID $I_0$ (containing an input $w$ of length $n$), is accepted, then there is a valid final ID $I_f$ such that $I_0 \rightsquigarrow I_f$. The notation $\rightsquigarrow$ denotes that there is a valid sequence of computation. For this, there must be some intermediate ID $I_k$ such that $I_0 \rightsquigarrow I_k$ and $I_k \rightsquigarrow I_f$ (alternately these are consecutive IDs).

Using this idea, design a deterministic TM $M'$ that uses at most $O(S^2(n))$ space for $S(n) \geq \log n$.

This is the statement of Savich’s theorem. We know that for any $k$-tape space bounded TM with tape alphabet size $t$ and $q$ states, the number of distinct IDs is bounded by $n \times t^{S(n)k} \times q \leq 2^{cS(n)}$ for some constant $c$ for $S(n) > \log n$. If any string $w \in L$, $I_0 \xrightarrow{2^{cS(n)-1}} I_f$ for some final ID $I_f$. This implies that for some ID $I_k$, we have $I_0 \xrightarrow{2^{cS(n)-1}} I_m$ and $I_k \xrightarrow{2^{cS(n)-1}} I_f$. Each of these can be solved recursively by cycling through all the $2^{cS(n)}$ options for $I_m$. These can be generated using a counter having $cS(n)$ bits and recursively solved. Note that the first and the second sub-problem can be solved using the same space, i.e., the space can be reused. So the space needed is the space for the counter and the ID for each recursive call that $dS(n)$ bits for some appropriate constant $d$. The total space required can be written as a recurrence

$$F(N) = F(N/2) + e \log N \text{ where } N = 2^{cS(n)}$$

This yields solution $F(N) = e \log^2 N$. This can be reduced to $S^2(n)$ by using space-compression.

Most attempts didn’t even try to formulate the reachability problem using an intermediate ID as a recursive function and instead claimed doing the DFS cleverly using $S^2(n)$ space by omitting crucial details. In fact, this is not possible as one has to remember enough information for back-tracking which will be proportional to the number of IDs which is exponential in space. No marks were given for clearly incorrect strategies. In many cases, the claim was that the number of IDs is bounded by $S^2(n)$.
5. The set intersection problem is defined as follows: Given finite sets \(A_1, A_2 \ldots A_m\) and \(B_1, B_2, \ldots B_n\), is there a set \(S\) such that

\[
|S \cap A_i| \geq 1 \text{ for } i = 1, 2, \ldots m \quad \text{and} \\
|S \cap B_j| \leq 1 \text{ for } j = 1, 2, \ldots n.
\]

Show that the set intersection problem is NP Complete. (10) The problem is in NP, since one can guess the set \(S\) and verify properties (i) and (ii) in polynomial number of steps.

For completeness, we reduce SAT to the set intersection problem as follows. For each clause \(i\), we define \(A_i\) to be the set of literals in clause \(i\), i.e., \((y_{i,1} \lor y_{i,2} \lor y_{i,3})\). On the other hand, \(B_i\) consists of elements \((x_i, \bar{x}_i)\). A truth assignment is the set \(S\) that cannot contain both the literal and its complement and each clause must have at least one true literal. For example, \(S = \{x_1, \bar{x}_2, \bar{x}_3\}\) implies the truth assignment \(x_1 = T, x_2 = F, x_3 = F\).

So the reduction function takes as input an instance \(\phi\) of 3-SAT problem and writes out \(A_i, 1 \leq i \leq m\) from each of the \(m\) clauses and \(B_i\) for \(i \leq n\). This is easily done in linear time.

Claim \(\phi\) is satisfiable iff the set-intersection problem has a solution.

(i) Suppose \(\phi\) is satisfiable, then construct \(S\) as the set of literals that are set to True. Clearly \(|S \cap A_i| \geq 1\) since each clause has at least one literal assigned to True. Moreover \(|S \cap B_j| \leq 1\) since a literal and its complement cannot be simultaneously True.

(ii) Conversely suppose \(S\) exists - then the satisfiable assignment can be constructed from all literals that are set to True. Because of the restriction on \(B\), the truth assignment is consistent. (If some variable is missing from \(S\), then its truth assignment is not required to make the formula satisfiable.)

Some answers did the reduction in the wrong direction. Although, notationally it was written as \(VC \leq SIP\), they actually created a graph from an instance of the SIP problem. Or created a boolean formula from an SIP instance. When you reduce \(L_1 \leq L_2\), then you have to take an arbitrary instance of \(L_1\) and map it to an instance of \(L_2\).

If the proof for the validity of reduction was missing and was penalised 3-4 marks. If you forgot to mention ”polynomial time” reduction, 1 mark was deducted.