1) $S = \{\langle M\rangle \mid M$ is a TM that accepts w^R whenever it accepts $w\}$.

we are given a set S, which consists of Turing Machines (TM) that have a specific property: a TM M belongs to S if and only if M accepts a string w if and only if it accepts the reverse of that string, w^R. We need to prove that the language S is undecidable. This means we need to show that there is no Turing machine that can decide for any given TM M whether M belongs to S.

An undecidable language is a language for which no Turing Machine exists that can decide membership in the language for every possible input. To prove that S is undecidable, we need to show that there is no Turing machine that can take any TM M as input and correctly decide whether M accepts w if and only if it accepts w^R for every string w.

A common technique to prove a language is undecidable is to use a reduction from a known undecidable problem. In this case, we will reduce from the Halting problem, which is known to be undecidable. The Halting problem asks whether a given Turing Machine M halts on input w.

Construct a Turing Machine M' to use in the reduction. Given an instance (M, w) of the Halting Problem, we will construct a new Turing Machine M' that will help us decide if M belongs to S. M' will work as follows:-

- On input x, M' simulates M on w
- If M halts on w, M' will then check if x is equal to its reverse x^R.
- If x is equal to x^R, M' accepts.
- If x is not equal to x^R, M' rejects.
- If M does not halt on w, M' will enter an infinite loop and never halt.

Notice that if M halts on w, M' accepts exactly those strings x that are equal to their reverse (palindromes). If M does not halt on w, M' does not accept any string because it never halts. Therefore, M' belongs to S if and only if M halts on w, because M' has the property of accepting a string x if only if it accepts x^R (in this case, palindromes) if M halts on w.

If we had a Turing Machine that could decide S, we could use it to decide whether any Turing Machine M halts on input w by constructing M' as described and checking if M' belongs to S. Since the Halting Problem is undecidable, ~~language~~ this implies that deciding S is also undecidable. Therefore, S is an undecidable language. By reducing the Halting problem to the problem of deciding membership in S, we have shown that S is undecidable. This concludes the proof.

2) We need to prove that the language HALT_ON_BLANK_TAPE_TM is not decidable. This means we need to show there is no Turing Machine (TM) that can decide for every TM M whether M halts when started on a blank tape.

The language HALT_TM ~~reduces~~ = {(M, w) | M is a TM and M halts on w} is known to be undecidable. This is a fundamental result in computability theory, known as the Halting problem.

To prove HALT_ON_BLANK_TAPE_TM is undecidable, we will show that HALT_TM reduces to HALT_ON_BLANK_TAPE_TM. This means, if we had a decider for HALT_ON_BLANK_TAPE_TM, we could use it to decide HALT_TM, which is impossible since HALT_TM is undecidable.

Construct a reduction from HALT_TM to HALT_ON_BLANK_TAPE_TM.
Given an arbitrary instance (M, w) of HALT_TM, we need to construct a Turing machine M' such that M' halts on a blank tape if and only if M halts on input w.

Construction of M' as follows:-

- M' starts with a blank tape.
- M' first writes the string w on its tape. This can be done by hardcoding the string w into the description of M' so that M' knows what to write.
- After writing w, M' then simulates M on input w.
- If M halts on w, M' halts. If M does not halt on w, M' also does not halt.

By design, M' halts on a blank ~~space~~ tape if and only if M halts on input w. This is because M' is constructed specifically to simulate M on w starting from a blank tape.

Since we can construct such an M' for any given (M, w), we have shown that if we could decide HALT_ON_BLANK_TAPE_TM, we could also decide HALT_TM by constructing M' and checking if M' halts on a blank tape. This would contradict the undecidability of HALT_TM.

Since we have shown that HALT_TM reduces to HALT_ON_BLANK_TAPE_TM, and since HALT_TM is undecidable, it follows that HALT_ON_BLANK_TAPE_TM is also undecidable. This p concludes the proof.

3) We have two languages A and B over the alphabet $\{0,1\}^*$. It is given that $A \leq_m B$, meaning A is many-one reducible to B. Additionally, it is stated that B is a regular language. We are asked to prove or disprove whether A must also be a regular language.

language A is many-one reducible to another language B if there exists a computable function f such that for every string w in $\{0,1\}^*$, w is in A if and only if f(w) is in B. This function f essentially transforms any instance of the problem A into an instance of the problem B.

A language is regular if it can be recognized by some finite automaton. Regular languages are closed under various operations, including union, intersection, complement and homomorphism.

If B is a regular language and there exists a computable function f that can transform any string w from A to a string in B such that w is in A if and only if f(w) is in B, we need to analyze the impact of this transformation on the regularity of A.

The function f, being computable, can be seen as a form of homomorphism when applied to strings. Since regular languages are closed under homomorphism, if we were to apply a homomorphism to a regular language, the resulting language would also be regular. However, this closure property applies to the transformation of B into another language, not necessarily from A to B.

The fact that $A \leq_m B$ means we can transform A into B, not necessarily the other way around. The regularity of B does not imply the regularity of A through the reduction because the reduction shows how to simulate A using B, not that A itself adheres to the properties that make B regular.

To disprove that A must be regular, we need to find an example where A is not regular, but B is regular, and $A \leq_m B$. consider the language $A = \{0^n 1^n \mid n \geq 0\}$, which is the set of strings of 0s followed by equal number of 1s. This language is not regular (as proven by the pumping lemma). let $B = \{0,1\}^*$, which is clearly a regular language since it can be recognized by a finite automaton that accepts any string of 0s and 1s.

Define a function f such that $f(w) = w$ for all w in $\{0,1\}^*$. This function trivially satisfies the condition for many-one reducibility since if w is in A, then $f(w) = w$ is in B (since B accepts all strings), and if w is not in A, $f(w) = w$ is still in B (but it does not affect the reducibility condition).

In this case, A is not-regular, but B is regular, and we have shown that $A \leq_m B$ through a simple identity function. This counterexample disproves the statement that if $A \leq_m B$ and B is a regular language, then A must also be regular.

The existence of a many-one reduction from A to a regular language B does not guarantee that A is also regular. The counterexample provided demonstrates a scenario where A is not regular, yet it is many-one reducible to a regular language B.