1. (a) $L = \{M : M$ has an useless state q $\}$
Language L is undecidable. Suppose L was decidable then we can use that as a sub routine to solve $\overline{HP}$ as follows:

   - Consider a new machine $\grave{M}$ which has two states - start and accept state. It writes its input y on a seperate tape.
   - It then runs M on input x . If M halts on x it accepts its input y.

   The machine $\grave{M}$ can be described as :

$$\grave{M} = \begin{cases} \grave{M} \textit{ has no useless state} & \text{if } M \text{ halts on } x \\ \grave{M} \textit{ has an useless accept state} & \text{if } M \text{ does not halt on } x \end{cases}$$

   Since our assumption is that L is decidable implies there exists a total turing machine K which accpets L . Giving $\grave{M}$ as input to K , we can totally compute if M halts on x . But we know there is no such total turing machine. Hence, language L is not decidable.

   (b) $L = \{(M, w)$: M moves its head left during computation of w $\}$
This language is decidable. Suppose, during computation of w, the head moves left once, we are done nad can accept (M, w) .
If it deosnt not move left till it reaches the first blank symbol - Let q_1 be the state at which it first reads the blank symbol. Let N be the number of states of machine M. Simulate the machine for a further $N + 1$ steps. If the head moved left during this time then accept (M, w) . Else reject. Because :
$\delta(q_1, blank) = q_j \ \forall j \in [1, N]$. By pigeon hole principle, it should return to a previously visited state after $N + 1$ steps.

   (c) $L = \{(M1,M2)$: $L(M1) = \overline{L(M2)}\}$
L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:
Consider two machines $M1$ and $M2$ which does the following:

   - Both machines simulate M on x . If M halts on x then machine $M1$ accepts its input y , while machine $M2$ rejects input y.

   So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

   Similarly the language accpeted by machine $M2$ can be described as:

$$L(M2) = \begin{cases} \varnothing & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines ($M1$, $M2$) as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total turing machine, hence no such total turing machine K exists.

(d) $L = \{M : \exists x \in \Sigma^*$ s.t. M runs forever on input $x\}$
This language is undecidable . If this language was decidable by a total turing machine K , then we can use it as a sub routine to solve $\overline{HP}$ as follows:
Consider machine $\grave{M}$ which does the following:

- $\grave{M}$ writes its input y on a seperate track.
- It runs machine M on input x. If M halts on x , then it accepts its input y.

The machine $\grave{M}$ can be described as follows:

$$\grave{M} = \begin{cases} For\ all\ inputs\ \grave{M}\ doesnt\ run\ forever & \text{if } M \text{ halts on } x \\ There\ exists\ an\ input\ for\ which\ \grave{M}\ runs\ forever & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving $\grave{M}$ as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total machine, hence no such total turing machine K exists.

(e) $L = \{M$: M accepts atleast one plaindromic string$\}$
L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:
Consider machine $\grave{M}$ which does the following:

- Machine $\grave{M}$ simulates M on input x.
- If M halts on x, then $\grave{M}$ accepts its input y

So the language of machine M can be described as :

$$L(M) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Since, Σ^* contains atleast one palindromic string, Giving machine M as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total turing machine, hence no such total turing machine K exists.

(f) $L = \{M$: M accepts only plaindromic strings$\}$
L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:
Consider machine $\grave{M}$ which does the following:

- Machine $\grave{M}$ simulates M on input x.
- If M halts on x, then $\grave{M}$ checks if its input y is a palindrome. If yes, it accepts y , else rejects y.

So the language of machine M can be described as :

$$L(M) = \begin{cases} P & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Where P is the set of all palindromes.
Giving machine M as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total turing machine, hence no such total turing machine K exists.

(g) $L = \{(M1{,}M2)\text{: } L(M1) \bigcap L(M2) \neq \varnothing \ \}$
L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:
Consider two machines $M1$ and $M2$ which does the following:

- Both machines simulate M on x . If M halts on x then both machines $M1$ and $M2$ accepts their inputs $y1$,$y2$ respectively.

So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Similarly the language accpeted by machine $M2$ can be described as:

$$L(M2) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

And the Language $L(M1) \bigcap L(M2)$ can be described as :

$$L(M1) \bigcap L(M2) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines ($M1$, $M2$) as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total turing machine, hence no such total turing machine K exists.

2. (a) (107) $L = \{M : L(M) \ = \ rev\ L(M)\}$
This language is undecidable.
If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:
Consider machine $\grave{M}$ which does the following:

- Machine $\grave{M}$ simulates M on input x.
- If M halts on x, then $\grave{M}$ checks if its input y is a palindrome. If yes, it accepts y , else rejects y.

So the language of machine M can be described as :

$$L(M) = \begin{cases} P & \text{if } M \text{ halts on } x \\ \varnothing & \text{if } M \text{ does not halt on } x \end{cases}$$

Where P is the set of all palindromes.
Giving machine M as input to machine K can help in totally computing if machine M halts on input x. Since, there is no such total turing machine, hence no such total turing machine K exists.

(b)

(c)

(d)

(e)

(f)

3. (a) $L = \{a^p \ : \ p \ is \ a \ prime \ number\}$
   This language is not regular but is decidable.

   (b) For a given turing machine M,
   $L = \{1^n \ : \ M \ halts \ on \ input \ 1^n\}$
   Since, its a fixed machine M , the transitions of M can be hardwired in the transitions of the machine K simulating it. And hence the set of tape alphabets is singleton.

   (c) $FIN = \{M \ : \ L(M) \ is \ finite\}$,
   Generally, we represent the encoding of M using the bits 0's and 1's. However, this can be viewed as the nth lexicographic permutation of 0's and 1's. Hence, encoding M as a^n , where n represents the nth lexicographic permutation of 0's and 1's. Hence every machine can be encoded over a singleton alphabet.

4. (a) $\leq_m$ is reflexive and transitive,but not symmetric.
   Reflexive: A $\leq_m$ A via the identity function $\sigma(x) = x$
   Transitive: If A $\leq_m$ B via a map σ and B $\leq_m$ C via the map τ , then A $\leq_m$ C via the map $\tau \circ \sigma$
   Symmetric: $\leq_m$ is not symmetric because the function σ is not invertible.

   $\leq_T$ is reflexive and transitive,but not symmetric.
   Reflexive: A $\leq_T$ A .Given a A as an oracle, the OTM just has to return the value of its query to the oracle to accept/reject A.
   Transitive: If A $\leq_T$ B , then given B as an oracle, A is totally computable. Similarly B $\leq_T$ C implies given C as an oracle , B is totally computable. Now, an OTM with C as an oracle, can compute A totally as follows. Using C as the oracle, compute and store intermidiate value of B. Now this value can serve as an answer from an oracle and hence A can be calculated. Calculation of B doesnt cause looping (B $\leq_T$ C). Hence, calculation of A doesnt cause looping. Hence A $\leq_T$ C.
   Symmetric: $\leq_T$ is not symmetric.Example : $FIN \leq_T REG$, but $REG \not\leq_T FIN$.

   (b) L is decidable => L $\leq_m$ 1^*0^*

   To show this we have to find a $\sigma : \Sigma^* \rightarrow \Sigma^*$:

   - σ is totally computable

- $\forall x, x \in L <=> \sigma(x) \in 1^*0^*$

Since, L is decidable , there exists a total turing machine K such that, $L(K) = L$.
Let σ be defined as follows :

$$\sigma(x) = \begin{cases} 10 & \text{if } K \text{ accepts } x \\ 01 & \text{if } K \text{ rejects } x \end{cases}$$

σ is totally computable since K is total.
$x \in L <=> K\ accepts\ x <=> (\sigma(x) = 10 \in 1^*0^*)$

Therefore, L $\leq_m$ 1^*0^*

L is decidable <= L $\leq_m$ 1^*0^*

RHS implies there exists a totally computable σ such that $\forall x, x \in L <=> \sigma(x) \in 1^*0^*$.
For any input x , calculating $\sigma(x) \in 1^*0^*$, can be done by a total turing machine.(Keep 4 states. when on state 1 implies seeing 1's. First time a 0 is seen, move to state 2. Or if end of input is seen, move to state 3 and accept. In state 2 ,if end of input is reached go to state 3 and accept. If a 1 is seen, move to state 4 and reject). Hence, $x \in L$ can be totally computed . Therefore , L is decidable.

5.