1) We will prove that TILE is undecidable by reducing from $\overline{HALT}$. Given a TM M with input w, we will construct an instance of TILE | if M halts on w, the resulting tiling is only possible up to a certain size. If M does not halt, a valid tiling exists $\forall$ grid sizes. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. Q is the set of all states, Σ is the input alphabet, Γ is the tape alphabet, $q_0 \in Q$ is the start state, $q_{accept} \subseteq Q$ is the set of accept states, and $q_{reject} \subseteq$ is the set of reject states. We will represent each tile as a 3-tuple (t_1, t_2, t_3) where t_1 is the left of the symbol, t_2 represents the symbol, and t_3 represents the right of t_2. Note that if we are labeling a tile with a symbol under the head, we will write it as a 2-tuple in the form (q, t), where q is the current state | we have a 3-tuple in the form $((q, t_1), t_2, t_3)$. $\therefore$ On an input $w = w_1, w_2, \ldots w_n$, our first tile which will go in the first part of our grid or $(1,1)$ is defined as $((q_0, w_1), w_2, w_3)$. Let the first row of our tiling be the start configuration of M from the input w | we add rows if they follow the compatibility rows below. $\therefore$ given $w = w_0 w_1 w_2 \ldots w_n$ we have a starting set of tiles as $((q_0, w_0), w_1, w_2)(w_1, w_2, w_3)(w_2, w_3, w_4) \ldots (w_{n-2}, w_{n-1}, w_n)(w_{n-1}, w_n, \sqcup)(w_n, \sqcup, \sqcup)$. We define horizontal compatibility as: given tile pair (T_1, T_2) where $T_1 = (t_1, t_2, t_3)$ and $T_2 = (s_1, s_2, s_3)$ we say that (T_1, T_2) are horizontally compatible if $t_2 = s_1$ and $t_3 = s_2$. Assume T_1 is the left tile and T_2 is the right. We define vertical compatibility as such: given a tile pair (T_1, T_2), the two tiles are vertically compatible iff T_2 can be reached from T_1 in one step as defined transition functions of M or the following transitions. Assume $T_1 = (t_1, t_2, t_3)$ and $T_2 = (s_1, s_2, s_3)$ where T_1 is the top tile and T_2 is the bottom one.

- if $t_1 = (q, t_1)$ and $\delta(q, t_1) = (q', s_1, L)$ then $s_2 = t_2$ and $s_3 = t_3$
- if $t_1 = (q, t_1)$ and $\delta(q, t_1) = (q', s_1, R)$ then $s_2 = (q', t_2)$ and $s_3 = t_3$
- if $t_2 = (q, t_2)$ and $\delta(q, t_2) = (q', s_2, L)$ then $s_1 = (q', t_1)$ and $s_3 = t_3$
- if $t_2 = (q, t_2)$ and $\delta(q, t_2) = (q', s_2, R)$ then $s_1 = t_1$ and $s_3 = (q', t_3)$
- if $t_3 = (q, t_3)$ and $\delta(q, t_3) = (q', s_3, L)$ then $s_1 = t_1$ and $s_2 = (q', t_2)$
- if $t_3 = (q, t_3)$ and $\delta(q, t_3) = (q', s_3, R)$ then $s_1 = t_1$ and $s_2 = t_2$
- if t_1, t_2, t_3, s_2 don't contain a state symbol, then $(t_1, t_2, t_3) = (s_1, s_2, s_3)$ where $s_2 = t_2$ and $s_1 = t_1$ or $s_1 = (q, t_1)$ and $s_3 = t_3$ or $s_3 = (q, t_3)$ where $q \in Q$.

$\therefore$ we have defined our compatibility rules. Also, to prevent moving off the left bound we make our starting input tape to have a starting symbol like \$ and copy our input tape w after it. So whenever the head of the tape reads the \$ the tape will not move left. Also we must ensure that once M reaches q_{accept} or q_{reject} there are no more transitions possible to leave these states (thus when M halts there are no legal transitions). Further notice that the way we have defined our transition function leads to the possibility of the head of M appearing to the left or right as it moves across tiles, $\therefore$ leading to an issue where the head can appear on the rightmost tile. To solve this issue, we modify our →

1) configuration | we have two versions of our symbols we can call type 1 and type 2. We let type 1 symbols occur in each row up to the cell with the head and let the head and directly right of the head be type 2. This way it's impossible for the head to appear in the rightmost cell as the rightmost cell is always symbol type 2. Note that this is indeed a reduction from $\overline{HALT}$. If we have a "yes" case in $\overline{HALT}$, we don't halt on an input w. Consider TILE | for an nxn grid we can tile each square following our rules for w. ∴ our reduction is clearly a "yes" instance of TILE. If we have a "no" case in $\overline{HALT}$ we halt on an input w after some number of steps $n' < n$. Notice that in TILE, if we have an input w for an nxn grid if we halt after some n' steps then we can only tile n' rows | our grid is incomplete and there are no legal transitions for M. ∴ we have created a "no" instance of TILE. Note that we may have to fill w with blanks to get to a width of n or shorten our input w to n if $|w| > n$. ∴ we have done a reduction from TILE to $\overline{HALT}$ and since $\overline{HALT}$ is undecidable, TILE must be too.

2) NTS that for any fixed graph H, we can determine whether an input graph G contains a subgraph isomorphic to H in polynomial time. Given graph H with k vertices, we must find a set of k vertices in G whose edges form the same structure as H. Since H is fixed, we can solve this problem by searching through all the possible vertex subsets and mappings. To do this we will choose a subset with k vertices from G. There are at most $\binom{|G|}{|H|}$ fitting subsets, which is upper bounded by $|G|^{|H|}$ There are k! ways to match the k vertices to the vertices of H since we need a bijective mapping. we must verify if the induced subgraph matches H by checking if every edge in H corresponds to an edge in G, which is done in $\leq O(k^2)$ time. The total time complexity of the algorithm we made is: $\binom{|G|}{|H|} k! \, k^{O(1)} \leq |G|^{|H|} \cdot k^k \cdot k^{O(1)}$. k is fixed, so the time complexity simplifies to $|G|^{O(k)}$, which is polynomial in the size of G because k is constant.

3) we will use dynamic programming to show that unary subset sum is in P. consider an array A with size B+1. Each entry A[B'] is true if $\exists$ a subset of the given numbers $\{x_1, x_2, \ldots x_n\}$ that sums to exactly B' and is false otherwise. Let A[0] = true since the empty subset sums to 0. Set all the other entries A[B'] to false initially. $\forall$ B' from 1 to B, if $x_i = B'$, then we set A[B'] = true. If $A[B' - x_i]$ = true, then we also set A[B'] = true since $\forall$ some set of x_i that sums to B'. $\therefore$ we accept iff A[B] is true. we will use induction to prove that after the B'-th iteration, A[B'] correctly represents whether a subset exists that sums to B'. Our base case is B'=0, which trivially holds since an empty subset sums to 0. our inductive hypothesis is to assume that for all $B' \leq k$, A[B'] is correctly computed. Induction: for B'=k+1. If $x_i = B'$, then A[B'] = true. If $T[B' - x_i]$ = true, then A[B'] must be true. From the hypothesis, $A[B' - x_i]$ is already correctly filled, so A[B'] is correct. From induction A is correct for all $B' \leq B$. The algorithm iterates over all B' values from 0 to B. In each iteration, all n values of x_i are checked, so one loop is in O(n) time. The total time complexity is O(Bn). B is given in unary with size O(B), so the runtime is polynomial in the input size. Since the algorithm runs in polynomial time relative to the input size, unary subset sum is in P.