Show that the following language $L$ is in NP.

$L = \{ \langle S, k \rangle \mid S \text{ is a finite set of binary strings, } k \text{ is a positive integer, and there is some binary string } w \text{ s.t. } |w| < k \text{ and } \forall x \in S, x \text{ is a substring of } w \}$

Also, show that if $|S| = 2$, then we can decide membership of $L$ in polynomial time.

(Throughout, we denote the length of the input as $n$.)

$L \in \text{NP.}$

There are $2^k - 1$ binary strings with length less than $k$. So, we non-deterministically pick out one of these strings $w$. We then check to see if each string $s_i \in S$ is a substring of $w$ by iterating $s_i$ over every possible substring of $w$. Each $s_i$ must be checked against the $k$ substrings of $w$ and each check takes (at most) $k$ steps. Since there are $|S|$ string to check, the whole algorithm runs in about $|S| \times k \times k$ time, which is polynomial.

Solution: We construct a non-deterministic, polynomial time decider for $L$.

$M$ on $\langle S, k \rangle$

1. Non-deterministically choose a binary string $w$ s.t. $|w| < k$.

2. For $i = 0$ to $|S|$
   a. For $j = 0$ to $|w|$
      i. For $\ell = 0$ to $|s_i|$
         - if $s_i[\ell] \neq w[j + \ell]$, break
      ii. If $s[\ell] = w[j + \ell]$, break
   b. If $s_i[\ell] \neq w[j + \ell]$, REJECT

3. ACCEPT

Correctness checklist:

✓ $M$ is a TM.
   There are finite instructions, and each instruction is computable.

✓ $M$ runs in polynomial time.
   Step 1 takes polynomial time. In step (2), each string $s_i$ is compared with every substring of $w$. In one such comparison the tapehead must travel back and forth $|s_i|$ times, yielding at most $|s_i| \times n = O(n^2)$ steps. There are at most $|S| \times k$ such comparisons, yielding a time complexity of $k|S| \times O(n^2) = O(n^c)$ for some constant $c$. So, the algorithm runs in polynomial time.

✓ If $\langle S, k \rangle \in L$ then $M$ accepts $\langle S, k \rangle$.
   Suppose $\langle S, k \rangle \in L$. Then there is some $w$ s.t. every string in $S$ is a substring of $w$. So, for each $s_i$, there is some starting index $j$ of $w$ such that for all $\ell$, $s_i[\ell] = w[\ell + j]$. So, the loop at step (i) always terminates with $s_i[\ell] = w[\ell + j]$. So, $M$ never rejects at step (b) and, once the outer loop completes, $M$ accepts $\langle S, k \rangle$. 

1
If \( \langle S, k \rangle \not\in L \) then \( M \) rejects \( \langle S, k \rangle \).

Suppose \( \langle S, k \rangle \not\in L \). Then for every string \( w \), there is some \( s_i \in S \) which is not a substring of \( w \). So, for some \( s_i \), every index \( j \) of \( w \) is s.t. there is some \( \ell \) where \( s_i[\ell] \neq w[j+\ell] \). So, each iteration of the loop at step (i) terminates with \( s_i[\ell] \neq w[j+\ell] \), and the loop at step (a) terminates with \( s_i[\ell] \neq w[j+\ell] \), so \( M \) rejects \( \langle S, k \rangle \).

Therefore, \( M \) is a polynomial time non-deterministic decider, so \( L \in \text{NP} \).

* Rather than using a polynomial time non-deterministic decider, we could also have used a polynomial time verifier. The construction is essentially the same:

\( M' \) on \( \langle \langle S, k \rangle, w \rangle \)

1. Check that \( w \) is a binary string s.t. \( |w| < k \)

Where steps (2) and (3) are the same as in \( M \). \( M' \) is a valid TM since \( M \) is, and step (1) takes at most polynomial time. So, by the reasoning above, \( M' \) is a polynomial time verifier.

\( L \) is decidable in polynomial time if \( |S| = 2 \).

If \( S = \{s_1, s_2\} \), we want to see if there is some string \( w \) with length less than \( k \) that has both \( s_1 \) and \( s_2 \) as substrings. If \( k \geq |s_1| + |s_2| \), then \( \langle S, k \rangle \in L \) as \( |s_1 s_2| < k \) and has \( s_1 \) and \( s_2 \) as substrings. The problem is when \( k < |s_1| + |s_2| \). In this case we must find if there is a shorter string containing both \( s_1 \) and \( s_2 \). If there is such a string, we can find it by ‘overlaying’ \( s_1 \) on \( s_2 \). The algorithm runs as follows.

We start with the concatenation ‘\( s_1 s_2 \)’. If this string has length greater than or equal to \( k \), we slide \( s_1 \) one place to the right, so its last digit lines up with \( s_2 \)’s first digit. If the overlaying digits match, we have found a shorter string with both \( s_1 \) and \( s_2 \) as substrings (just as ‘100’ and ‘01’ are both substrings of ‘1001’). If these digits do not match, or if the length is still greater than or equal to \( k \), we slide \( s_1 \) one more place to the right and check again. We continue in this way until we reach ‘\( s_2 s_1 \)’. If, at that point, we have found no suitable string, we reject.

We can see that this algorithm takes polynomial time. Checking for matches takes linear time (we need to pass over the overlaid strings only once). Comparing the resulting string length to \( k \) takes polynomial time (we decrement \( k \) by 1 for each digit in the overlaid string). Since there are \( |s_1| + |s_2| \) checks and comparisons, the whole algorithm takes polynomial time.
Solution: We construct a polynomial time decider. The following definitions will make this construction easier. We suppose that $S = \{s_1, s_2\}$. We also define $s'_2$ to be the following string:

$$
\underbrace{\cdots}_{|s_1|} s_2 \underbrace{\cdots}_{|s_1|}
$$

where there are $|s_1|$ dashes (`-`) on either side of $s_2$. Iterating $s_1$ over the indices of $s'_2$ rather than $s_2$ allows us to test all possible overlays of $s_1$ on $s_2$. When comparing $s_1$ to $s'_2$, we say that two characters `match` if (a) both are `1's, (b) both are `0's, or (c) one is a `1'-. Finally, when $s_1$ is overlaid on $s_2$, we let $d$ be the number of cells from the leftmost character of the overlaid string to the rightmost character. Finding $d$ and constructing $s'_2$ each take polynomial time, so these aspects of the algorithm will not affect its complexity class. We are now ready to construct the polynomial time decider.

$D$ on $\langle S, k \rangle$

1. Construct $s'_2$

2. For $i = 0$ to $|s_1| + |s_2|$
   (a) For $j = 0$ to $|s_1|$
      - if $s_1[j]$ does not match $s'_2[i + j]$, break
      - if $j = |s_1|$ and $d < k$, ACCEPT

3. REJECT

Correctness checklist:

✓ $D$ is a TM.

$D$ has finitely many instructions, and each instruction is computable.

✓ $D$ runs in polynomial time.

Step (1) takes $O(n^3)$ time. In step (2), $s_1$ is compared with every substring of $s'_2$. Each comparison takes $|s_1|$ steps, and checking that $d < k$ takes $O(n^2)$ steps. Since there are $|s_1| + |s_2|$ comparisons, the algorithm takes $O(n) \times O(n^2)$ steps, which is polynomial.

✓ If $\langle S, k \rangle \in L$, then $D$ accepts $\langle S, k \rangle$.

Suppose $\langle S, k \rangle \in L$. Then there is some $w$ s.t. $|w| < k$ and $s_1$ and $s_2$ are both substrings of $w$. So, one such $w$ must be either `$s_1 s_2$', `$s_2 s_1$', or some overlay of $s_1$ on $s_2$. So, there some value $i$ such that for every value $j$, $s_1[i]$ matches with $s'_2[j + i]$. Therefore, $D$ accepts $\langle S, k \rangle$.

✓ If $\langle S, k \rangle \notin L$, then $D$ rejects $\langle S, k \rangle$.

Suppose $\langle S, k \rangle \notin L$. Then for every $w$ s.t. $|w| < k$, either $s_1$ or $s_2$ is not a substring of $w$. So, no such $w$ is `$s_1 s_2$', `$s_2 s_1$', or some overlay of $s_1$ on $s_2$. So, for every value $i$ there is some value $j$ such that $s_i[i]$ does not match $s'_2[j + i]$. So, the loop at step (a) always terminates before accepting, and $D$ rejects $\langle S, k \rangle$. 