1. Consider a TM variant with a single two-dimensional tape, where the tape head can move either one step to the left, right, up or down. The input is arranged in a $\sqrt{n} \times \sqrt{n}$ grid, with the first $\sqrt{n}$ symbols on the first row, the second $\sqrt{n}$ symbols on the second row, and so on (there may be some empty spaces in the last row if $n$ isn’t a perfect square.) Show that this model can simulate a $k$-tape TM running in time $T(n)$ in time $O(T(n)^{3/2})$. Then show that there is a language solvable in linear time on a two-tape machine that requires $\Omega(n^{3/2})$ time with a single two dimensional tape.

We’ll simulate $k$ tapes by extending the cell alphabet to include $k$ concatenated symbols in a single cell, plus some helper markers: a special symbol meaning the $i$’th tape head is at this cell, and a different symbol marking the first cell in a row holding the $i$’th tape head. We’ll put the first cell on the first row, the next two on the second row, the next three on the third row and so on, only using the first $i$ symbols of the $i$’th row. Since the sum of the first $t$ numbers is $\Omega(t^2)$, we will use no more than $O(T^{1/2})$ rows, and use at most that many cells in each row.

To simulate each step, we find all the tape heads, by scanning the first cell in each row until we see the row marker, then scanning the row until we find the cell marker. This takes $O(T^{1/2})$ steps. Once we have all the contents of cells under tape heads, we can use the transition function for the $k$-tape machine to know which action to take and which state the machine will be in next. Accept or reject actions are immediate. If we are writing on tape $i$, we need to find the tape cell indicator as before and write at that cell (keeping the marker). If we are moving the head to the left, we find the tape head marker as before and erase the marker. If we are not at the first cell on the row, we move one left and make a mark. If we are at the first cell, we move down one row, and move to the right until we reach the end, and make the mark there. Moving to the right is similar. If we are not at the last non-blank symbol on our row, we simply erase the marker, move one to the right, and write the marker there. If we are at the last blank symbol, we look down and to the left one. If that is not blank, we go back and write the marker. If it is, we go all the way to the left and up one, and write the marker there. Thus, simulating each of $T$ time steps takes time $O(T^{1/2})$, for a total time of $O(T^{3/2})$.

(To initialize, we need to move the input bits to this form, using the same method to keep track of where we are writing from and where we are writing to. This takes time $O(n^{3/2})$ one tape cell marker to keep track of the row and cell we are currently copying from, and a second to the row and cell we are copying to. The cells we are copying from have a slightly different successor/predecessor rule, but otherwise it is the same.)

To prove the lower bound, we use the language $L = \{x2^n x\}$ defined in class. On an input of the form $x2^{2n} y$ where $x$ is a binary string of length $n$, $x$ will start on the first $1/2\sqrt{n}$ rows, then there will be $\sqrt{n}$ rows of 2’s, and the last $1/2\sqrt{n}$ rows will contain $y$. We use the $\Omega(n)$ lower bound for
the zero-error communication complexity of equality. The protocol will be as follows: On input \( x \) to \( Alice \) and \( y \) to \( Bob \), they pick a random \( i \in \{ 0.5\sqrt{n}, 1.5\sqrt{n} \} \). Alice will simulate the first \( i \) rows of the tape, Bob the rest. They can initialize the machine since row \( i \) is strictly between the \( x \) bits and the \( y \) bits. Whenever \( M \) crosses from row \( i \) to row \( i+1 \) Alice sends Bob the state, and Bob does likewise when \( M \) crosses from row \( i+1 \) to row \( i \). When \( M \) halts, either accepting or rejecting, they output the corresponding bit. Since there are only \( T(n) \) steps, and each one is only \( 1/\sqrt{n} \) likely to cross the row randomly chosen from \( \sqrt{n} \) possibilities, the expected communication of this protocol is \( O(T(n)/\sqrt{n}) \), which by the lower bound must by \( \Omega(n) \). Thus, \( T(n) \in \Omega(n^{3/2}) \), as claimed.

2. It would be nice to have a programming language \( PL \) where: A) (Recognizability) we could computably tell whether a string was a valid \( PL \) program ; B) (Termination guarantee) given a valid \( PL \) program and an input, we could simulate the program on the input computably, thus guaranteeing that all programs in our language halt; and C) (Generality) for every recursive language \( L \), there is a valid \( PL \) program that computes membership in \( L \).

Show that no programming language exists having all three properties.

Assume there were. Let \( V \) be an algorithm deciding whether \( p \) is a valid program, and let \( S(p,x) \) be the simulator that always halts when \( p \) is valid, and determines whether \( x \in L_p \).

We’ll show a decidable language that is not of the form \( L_p \) for any \( p \), contradicting generality.

Let \( Diag = \{ p | V(p) \text{ and } S(p,p) = 0 \} \)

\( Diag \) is decidable because we can first test \( V(p) \), and if false reject, and if true run \( S(p,p) \) and see if it is 0.

If \( Diag = L_{p_0} \) for some valid \( p_0 \), then \( p_0 \in Diag \) iff \( V(p_0) \) and \( S(p_0,p_0) = 0 \) iff \( S(p_0,p_0) = 0 \) (since \( p_0 \) is valid), iff \( p_0 \notin L_{p_0} \) (by definition of \( L_p \) iff \( p_0 \notin Diag \) (since \( Diag = L_{p_0} \)). So this contradicts generality.

3. Consider two kinds of constructions, making a new language from an existing language, \( L \): One of Two\( L \) = \{ \( (x_1, x_2) \) : exactly one of \( x_1 \in L \) and \( x_2 \in L \} \). \( Maj_3 L \) = \{ \( (x_1, x_2, x_3) \) : at least two of \( x_1 \in L, x_2 \in L, x_3 \in L \} \).

We say that a class is closed under a construction if whenever \( L \) is in the class, the constructed language is in the class as well. For each of the constructions above, is \( COMP \) closed under the construction? Is \( CE \) closed under the construction, where \( CE \) is the class of computably enumerable languages (also called \( RE \))? Prove all four parts of your answer (but some will be shorter than others).

If \( L \in COMP \), for either construction, we can run the algorithm for \( L \) on all the \( x_i \) in sequence, and use that to determine whether the input is in the constructed language. So \( COMP \) is closed under both constructions.

If \( L \in CE \), \( Maj_3 L \) can be weakly recognized as follows. Let \( M \) weakly recognize \( L \), i.e., it always halts and accepts if \( x \in L \), but might either reject or not halt if \( x \notin L \). Starting with \( T = 1 \), run \( M \) on each input for \( T \) steps. If \( M \) accepts two of its inputs, accept; if it rejects two of its inputs reject; and otherwise double \( T \) and repeat.

If two of the three inputs are in \( L \), there will be a time \( t \) when \( M \) has accepted both of them. By the first iteration when \( T > t \), our simulation will accept. If two inputs are not in \( L \), no iteration will accept, so we will either reject or continue indefinitely. Thus, this algorithm is a weak acceptor for \( Maj_3 L \), and \( CE \) is closed under this construction.

For One of Two\( L \), let \( L \in CE − COMP \), e.g., \( L = HALT \). Since \( L \) is not computable, it is not the empty language, so let \( x_{in} \in L \) be some element of \( L \). Then if we had a weak recognizer for One of Two\( L \), \( M \). given \( x \), we could run \( M \) on the pair \( (x, x_{in}) \). \( M \) would eventually halt iff \( x \notin L \). Thus, \( L \) would be in \( CE \). But then \( L \in COMP \), since \( COMP = CE \cap co − CE \).