2D TM We show how to simulate a 2D TM on a 5-tape TM with polynomial overhead.

The two dimensional TM only uses memory in the upper righthand quadrant of the plane, i.e., cells \( <x,y> \) with \( x, y \geq 1 \). We’ll use the following pairing function, \( p(x, y) = \frac{x + y - 12 + x}{2} \) and store the \( x, y \) th 2D tape cell in the \( p(x, y) \)’th linear tape cell on the first tape. Note that the difference between \( p(x, y) \) and \( p(x, y + 1) \) is \( x + y - 1 \) and to \( p(x + 1, y) \) is \( x + y \). We also keep the current positions \( x \) and \( y \) on two separate tapes (each initialized to 1). We’ll use the fourth and fifth tapes for arithmetic.

At the beginning of simulating one step of the blackboard TM whose tape head is at \( x, y \), the simulating 3-tape TM will have its first tape at \( p(x, y) \) and the values of \( x \) and \( y \) on the next two tapes. The two remaining tapes will be empty.

Since we know the current state of the 2D TM, and the contents of the cell it is reading, we know its next action. Writing is simulated by writing to the current cell. If the blackboard TM moves its tape head up, i.e., increases \( y \) to \( y + 1 \), the simulating machine computes \( x + y \) on the fourth tape, moves its tape head \( x + y \) cells to the right as follows, decrementing the fourth tape each time. This takes a total of \( O(x + y) \) steps, since the cost of addition is \( O(\log(x + y)) \) and the amortized cost of decrementing is constant. When it reaches 0, it erases the last zero from the fourth tape. It then increments the third tape, with \( y \) on it, in \( O(\log y) = O(y) \) steps. Similarly for moving down, left or right, using \( x + y - 1 \) moves to the left, \( x + y - 1 \) moves to the right, and \( x + y - 2 \) moves to the left, respectively, in place of \( x + y \).

We also need to initialize by moving the input to \( p(1,1), p(2,1),...p(n,1) \). This takes \( O(n^2) \) time
Each of the simulating steps could take up to \( O(x + y) \) time. Since the blackboard machine can only increase \( x + y \) by 1 every step, this is \( O(T(n)) \). So simulating \( T(n) \geq n \) steps takes \( O(T(n)^2) \) time.

In the reverse direction, we can simulate a 1-tape TM on a 2D TM by simply never moving up or down.

Thus, what is computable by a 2D TM in polynomial time is computable on a \( k \)-TM in polynomial time. What is computable on a \( K \)-TM in polynomial time is computable by a 1-TM in polynomial time, hence by a 2D TM in polynomial time. Thus, the two models define the same class of polynomial time computable languages.

Time-space Tradeoff We can look at external memory use by making the input tape read only, i.e., not allow \( \text{write}_1 \) commands in the program. Then the external memory is the total number of cells used on the other tapes, where a cell is considered used if the tape head is ever at that cell. Show that for the language \( L = \{xo^n x | n \geq 1, |x| = n \} \) considered in class, for any algorithm that solves this problem on a \( TM \) with at most \( S(n) \) external memory and \( T(n) \) time steps, \( T(n)S(n) \in \Omega(n^2) \).
Any algorithm using no external memory can be simulated by a finite automaton. Therefore, $S(n) \geq 1$.

We showed in class, and is in Chapter 13 of the text, that any communication protocol to test whether two $n$ bit strings are equal requires $\Omega(n)$ bits of communication.

If $M$ is a $k$-TM deciding $L$ and uses time $T(n)$ and space $S(n)$, consider the protocol $p_M$ as follows:

(This version was suggested by Geoff, and is a simplification over the one in class for this problem.)

Alice imagines $x$ written on the first $n$ cells of the input tape, and 0's written on the next $n$. Bob imagines the $n$ 0's and his input $y$ on the third group of $n$ cells. Since these tapes are read-only, these contents never change. Then they simulate the machine as follows:

Alice simulates $M$ until it goes to cell $2n+1$. At this point, she transmits the state and the contents and tape head positions of the other $k$ tapes to Bob. This takes $\log |Q| + kS(n) + k \log S(n)$ bits to describe, which is $O(S(n))$ since $k$ and $Q$ are fixed, and $S(n) \geq 1$. Then Bob simulates $M$ until it returns to cell $n$, transmitting the same information to Alice. They repeat until the machine accepts or rejects, and the last person tells the other which occurred.

Each transmission takes $O(S(n))$ bits. Since at least $n$ steps occur between transmissions, to allow the machine to go from cell $2n+1$ to cell $n$ or vice versa, there are at most $T(n)/n$ messages sent. Thus, the total communication is $O(T(n)S(n)/n) = \Omega(n)$ from the lower bound. Therefore $T(n)S(n) = \Omega(n^2)$.

**FP Bit-by-Bit**

Let $f$ be a function. Let $L_f$ be the language described in the problem. That is,

$$L_f = \{(x, i, b) | i \leq |f(x)| \land (f(x))_i = b\}$$

The forward direction ($f \in FP \Rightarrow L_f \in P$) and $|f(x)| \leq poly(|x|)$ is easy. Given input $(x, i, b)$, first compute $y = f(x)$. This takes time polynomial in $|x|$ and therefore polynomial in the input $(x, i, b)$. Next, check if $i \leq |y|$, and finally if $y_i = b$. This takes linear time in the RAM model. For the second part, a machine that outputs $f(x)$ in poly(n) time can only output one bit per time step, so the polynomial time bound is also a bound on $|f(x)|$.

For the converse, assume $L_f \in P$ and $|f(x)| \leq poly(|x|)$ for some polynomial $poly$. We need to show $f \in FP$. We compute $f$ using as a sub-routine a machine $M$ that decides $L_f$. The following machine computes $f(x)$ on input $x$. We give the description in a slightly higher-level pseudo-code, which can be translated to RAM machine operations in straightforward manner. We write $\epsilon$ to mean the empty string and $\circ$ to be the string concatenation operator.

- $\text{done} \leftarrow 0$.
- $i \leftarrow 1$.
- $y \leftarrow \epsilon$
- While NOT (done) do:
  - IF $M(x, i, 0)$ accepts
  - THEN $y \leftarrow 0 \circ y$
  - ELSE $M(x, i, 1)$ accepts
  - THEN $y \leftarrow 1 \circ y$
  - ELSE $\text{done} \leftarrow 1$.
- $i \leftarrow i + 1$.
- Return $y$. 


Is a language REC? RE? co-RE? Let \( L \) be the language given in the problem statement. We first show that \( L \) is not recursive by reducing \( \text{HALT} \), the halting language (p. 188 in the Sipser textbook), to \( L \). We will reduce \( \text{HALT} \) to \( L \) via a mapping reduction, although for decidability we could also use a Turing reduction. The mapping reduction requires that we transform an an instance \((M, \omega)\) of the halting problem to an instance \((M', \omega')\) for a (hypothetical) machine that decides \( L \), so that \((M, \omega) \in \text{HALT} \) if and only if \((M', \omega') \in L \). Given \( M \) and \( \omega \) the reduction constructs the following machine \( M' \):

\[
M'(\gamma) = 
\begin{itemize}
  \item Erase input \( \gamma \)
  \item Write \( \omega \) on the input tape
  \item Run \( M \) on \( \omega \)
  \item IF \( M \) halts THEN output 1.
\end{itemize}
\]

Also set \( \omega' = 1 \). We need to show that the above equivalence holds. Assume \((M, \omega) \in \text{HALT} \). Since \( M \) accepts \( \omega \), the machine \( M \) will also halt and output 1 if run long enough. Thus, if \( |\gamma| \) is greater than this amount of time, and greater than \( |\omega| \), after \( |\gamma| \) steps \( M \) will halt and output 1 = \( \omega' \). The number of steps \( M' \) takes on \( \gamma \) is at most the time \( M \) takes to halt, plus the times to erase \( \gamma \), write \( \omega \), and output 1. All of these times are less than \( |\gamma| \), so the total time is at most \( 5|\gamma| < 100|\gamma| \). Thus \((M', \omega') \in L \).

To show the other direction, assume \((M', \omega') \in L \). Since \( M' \) outputs 1 after simulating \( M \) on \( \omega \), then \( M \) must halt on \( \omega \), so \((M, \omega) \in \text{HALT} \).

Thus, the language \( L \) is not recursive, since otherwise so would \( \text{HALT} \).

We claim that \( L \) is recursively enumerable. We will show this directly. The following machine \( M'' \) recognizes \( L \): On input \( M, \omega \), \( M'' \) simulates \( M \) first on the empty string for 0 steps, then on the two strings of length 1 for 100 steps, and so on. Note that each of these simulations terminates, since the time is bounded by 100 times the input size. If \( M \) ever returns \( \omega \), then \( M'' \) accepts.

Since \( L \) is recursively enumerable but not recursive, it can’t be co-recursively enumerable.