1. **(20 points):** Do Kozen, Homework 8, problem 1.
Describe a TM that accepts the set $\{a^n | n \text{ is a power of } 2\}$. Your description should be at the level of the descriptions in Kozen Lecture 29 of the TM that accepts $\{ww | w \in \Sigma^*\}$ and the TM that implements the sieve of Eratosthenes. In particular, do not give a list of transitions.

   Solution: My machine has a tape alphabet of $\{a, b, \vdash, \square\}$; the input alphabet is $\{a\}$. My machine makes successive sweeps across the tape replacing every other a that it encounters with a b. If it completes a sweep with only one a on the tape, then it accepts; $1 = 2^0$. Otherwise, if it encounters an odd number of a's in a sweep, then it rejects (all other powers of 2 are even). If it encounters no as, then it rejects; 0 is not a power of 2. This is enough detail to get a full-credit answer. I'll provide some more details for those who are using this to study.

   My machine has states $\{s, t, r, 1, e, o, back\}$.

   In state s:

   The machine is making a left-to-right scan and hasn't encountered any a symbols yet.

   If reading a $\vdash$ or b, the machine writes the same symbol as it read and moves one square to the right, and remains in state s.

   If reading an a, the machine writes an a, moves one square to the right, and enters state 1.

   If reading an $\square$, the machine completed the scan without having encountered an a. The machine rejects.

   In state 1:

   The machine is making a left-to-right scan and has encountered exactly one a symbol so far.

   The machine cannot encounter a $\vdash$.

   If reading an a, The machine writes a b on the tape, moves to the right, and enters state e.

   If reading a b, the machine writes a b, moves one square to the right, and remains in state 1.

   If reading a $\square$, the machine completed the scan having encountered exactly one a. The machine accepts.

   In state e:

   The machine is making a left-to-right scan and has encountered an even number of a symbols (greater than zero).

   The machine cannot encounter a $\vdash$.

   If reading an a, The machine writes a b on the tape, moves to the right, and enters state o.

   If reading a b, the machine writes a b, moves one square to the right, and remains in state e.

   If reading a $\square$, the machine completed the scan having encountered an even number of a's (and more than zero). It has erased half of them and is ready to start the next scan. The machine writes a $\square$, moves one square to the left, and enters state $back$.

   In state o:

   The machine is making a left-to-right scan and has encountered an odd number of a symbols (greater than one).

   The machine cannot encounter a $\vdash$.

   If reading an a, The machine writes an a on the tape, moves to the right, and enters state e.

   If reading a b, the machine writes a b, moves one square to the right, and remains in state o.

   If reading a $\square$, the machine completed the scan having encountered an odd number of a's (and more than one). The machine rejects.

   In state $back$:

   The machine is making a right-to-left scan looking for the left endmarker.

   If reading an a or b, The machine writes the same symbol on the tape that it read, moves one square to the left, and remains in state $back$.

If reading a $\vdash$, The machine writes a $\vdash$ on the tape, moves one square to the right, and enters state s to begin the next left-to-right scan.

The machine cannot encounter a $\square$.

2. **(30 points):** Do Kozen, Homework 8, problem 2.
A *linear bounded automaton* (LBA) is exactly like a one-tape Turing machine, except that the input string $x \in \Sigma^*$ is enclosed in left and right endmarkers, $\vdash$ and $\dashv$, which may not be overwritten, and the machine is constrained to never move left of the $\vdash$ nor right of the $\dashv$. It may read and write all it wants between the endmarkers.

   (a) **(8 points)** Give a rigorous, formal definition of deterministic, linearly bounded automata, including a definition of configurations and acceptance. Your definition should begin as follows: "A *deterministic linearly bounded automaton (LBA)* is a 9-tuple

$$M = (Q, \Sigma, \Gamma, \vdash, \dashv, \delta, s, t, r),$$

   where Q is a finite set of *states*, ..."

   Solution: A *deterministic linearly bounded automaton (LBA)* is a 9-tuple

$$M = (Q, \Sigma, \Gamma, \vdash, \dashv, \delta, s, t, r),$$

   where Q is a finite set of *states*; Γ is the tape alphabet, $\Sigma \subset \Gamma$ is the input alphabet, $\vdash, \dashv \in \Gamma - \Sigma$ are the left and right tape endmarkers respectively; $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{L, R\})$ is the transition function; and s, t, and r are the start, accept, and reject states respectively.
   As with a Turing machine, $\delta(q, \vdash) \in Q \times \{\vdash\} \times \{R\}$; likewise, $\delta(q, \dashv) \in Q \times \{\dashv\} \times \{L\}$. In other words, the machine may not move left beyond the left endmarker, $\vdash$, nor may it move right beyond the right endmarker, $\dashv$, nor may it erase either of the endmarkers. We define configuration of an LBA in direct analogy with configurations for a TM. Finally, we define $L(M)$ as

$$L(M) \quad = \quad \{x \in \Sigma^* \mid \exists n.\ M \text{ reaches a configuration after } n \text{ moves where it is in an accepting state}\}$$

   (b) **(5 points)** Let M be a linear bounded automaton with state set Q of size k and tape alphabet Γ of size m. How many possible configurations are there on input x, $|x| = n$?

   Solution: Including the endmarkers, there are $n + 2$ tape squares. The n in the middle can each take on any of m values. The tape head can be at any of these $n + 2$ squares, and the machine may be in any of k states. Thus, there are $k(n + 2)m^n$ possible configurations.

   (c) **(7 points)** Argue that the halting problem for deterministic linearly bounded automata is decidable.

   Solution: Because the machine can be in at most $k(n + 2)m^n$ different configurations, it is sufficient to simulate the machine for $k(n + 2)m^n$ steps. If the machine terminates within this number of moves, then it halts. If not, it must have returned to a configuration that it has seen before; the machine is in a loop; and it will never terminate.
   I'll add that an LBA can be simulated by a TM by a construction very similar to that for a universal TM.

   (d) **(10 points)** Prove by diagonalization that there exists a recursive set that is not accepted by any LBA.

   Solution: The main idea is to follow the construction used to show that the halting problem for Turing machines is undecidable by any Turing machine to show that the halting problem for LBAs is undecidable by any LBA. The one catch is that in the construction for TMs, we constructed a Turing machine M, that given some input N, wrote $N\#N$ on its tape and then ran the hypothetical TM that accepts $P\#x$ if machine P halts on input x. The problem is that an LBA can't write anything longer than its input its tape. We consider instead the set:

$$Z \quad = \quad \{M \mid \text{LBA } M \text{ does not halt when run with input } M\}$$

The set Z is recursive. A Turing machine with input M can write $M\#M$ on its tape and then run the machine described above for determining whether or not LBA M halts with input M. If M halts with input M, the TM rejects; otherwise it accepts. This TM is total, thus Z is recursive.

I'll now show by diagonalization that Z is not accepted by any LBA. For the sake of contradiction, assume that M is an LBA that accepts Z. I'll now construct a machine M' with the same input and tape alphabet as M, and add one more state, q. Transitions of M to state a are replaced in M' with transitions to state q. Once in state q, M' loops (e.g. in state q, M' moves to the right unless it is reading the right endmarker, in which case it moves to the left, staying in state q, and always writing the same symbol that it read).

What happens if we run M' with the description of M' for its input? Machine M' will run M. On the other hand, if M accepts, then M' will accepts and halt as well. But, M accepting means that M' does not halt when run with input M', but accepting causes M' to halt. Thus, M cannot accept. If M rejects, then M' will enter state q and loop. But, M rejecting means that M' halts when run with input M', but M's rejecting causes M' to halt. Thus, M cannot reject. Finally, if M loops, then M' loops as well, which means that M' is in Z. But M' fails to accept M'. All cases lead to a contradiction. This disproves our assumption that there is an LBA M that accepts Z. Thus, Z is recursive set that is not accepted by any LBA.

3. **(20 points):** I mentioned in class that we can construct a Turing machine that addresses its tape as if it were memory. In this problem, you'll show me how to do it.

   Consider a Turing machine with tape alphabet $\{0, 0', 1, 1', \square, \vdash\}$. Start with a configuration where the tape is of the form:

   $$0^{k_0}10^{k_1}10^{k_2}1\ldots0^{k_n}1\square^\omega$$

   We can interpret this tape as storing the sequence of integers k_0, k_1, $\ldots k_n$. Describe a Turing machine that starts in state q_1 and ends in state q_2 after replacing k_n with k_{k_n}. For example, if the tape is

   $$000101100000100000000101010001\square^\omega$$

   when the machine is in state q_1, the machine will do its thing and reach state q_2 with the tape holding

   $$000101100000100000000101000001\square^\omega$$

   You should specify your machine as a labeled state transition diagram like the ones that I've presented in class.

   This operation corresponds to reading from memory.

   Solution: Figure 1 shows my solution. I omitted arcs for transitions that will never occur for valid inputs. For brevity, I wrote $c_1, c_2, \ldots c_k \rightarrow (\bullet, d)$ to indicate that if the current tape symbol is c_1, c_2, $\ldots$or c_k, the machine writes the same symbol that it read and moves in direction d. I wrote $\bullet'$ to indicate that it replaces a 0 with a $0'$ or a 1 with a $1'$.

   First, my machine finds the end of the string of numbers by moving to the left until it encounters a non-blank symbol (state *q1*), and then moving to the right until it encounters a blank (state *p1*). The machine then moves to the beginning of the rightmost number of the string (states *p2* and *p3*). Assuming that this isn't the only number on the tape, the machine marks the leftmost symbol of the number with a $'$ (the transition from state *p5* to state *p6*, I'll handle the case of a tape with only one number at the end, that's what state *p4* is for). In state *p6*, the machine moves to the left end of the tape, and marks the first digit of the leftmost number with a $'$ (the transition from state *p7* to *p8*). At this point, the machine has the leftmost symbol of the first an last numbers on the tape marked with a $'$.

   The machine will continue to work with two marked symbols. The right marker counts digits in the rightmost number, while the left marker moves over a complete number for each 0 symbol of the rightmost number (states *p7* $\ldots$ *p11*). Once it reaches the end of the rightmost number, the machine copies the currently marked number to the end of the tape (states *p12* $\ldots$ *p14*).

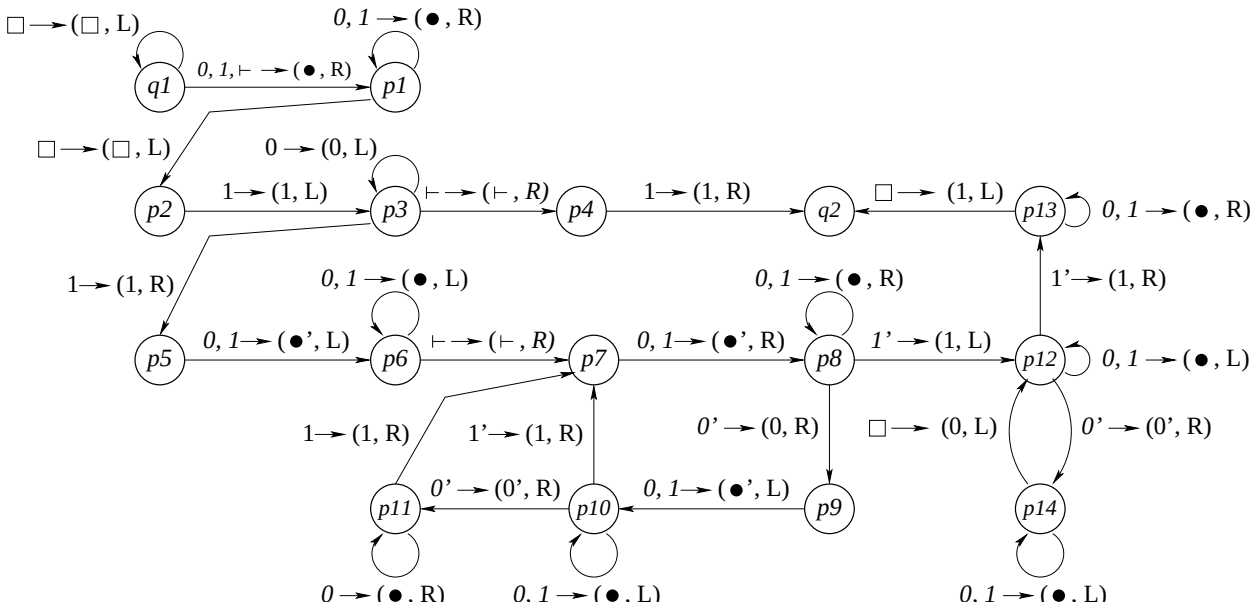

Figure 1: A Turing machine for reading from memory

Now, back to the case where the tape only has one number. Then, it can only reference itself, which means that this number must encode 0 for the input to be valid. If the rightmost number is preceded by the left endmarker (i.e., there is only one number on the tape), then the machine checks that this number is zero (which, ironically, means that it consists of a single 1 symbol). This is the transition from state *p3* to state *p4*. The machine writes one more 1 onto the tape. Now the tape holds two numbers, both of which have the value zero.

Figure 2 shows pseudo-code corresponding to the machine. I haven't included tests for invalid inputs:

4. **(30 points):** Do Kozen, Homework 9, problem 2.
Prove that it is undecidable whether two given Turing machines accept the same set. (This problem is analogous to determining whether two given PASCAL programs are equivalent.)

Solution: I will show that this problem is undecidable by reducing the halting problem for empty input to it. Let M be a Turing machine and x be a string for which we want to know if M halts when run with input x. I will construct two other Turing machines, M_1 and M_2 such that $L(M_1) = L(M_2)$ iff M halts when run with input x. M_1 accepts Σ^*. It is straightforward to construct M_1. When M_2 is run with input y, M_2 performs the following actions

- M_2 erases y and writes x onto its tape.
- M_2 performs the actions of M.
- If M reaches its accept or reject state, M_2 accepts.

Note that M_2 accepts iff M halts on input x. Thus $L(M_2) = \Sigma^*$ if M halts on input x, and $L_(M_2) = \emptyset$ otherwise. In other words, $L_(M_2) = L(M_1)$ iff M halts.

Note: I wrote this solution restricting myself to the material that we had covered when HW5 was assigned. Shortly after that, we learned about Rice's theorem. This problem is very easy to solve using Rice's theorem:

> Let M_1 be any Turing machine. If we could solve language equivalence, we could test whether an arbitrary Turing machine, M, recognized the same language as M_1. Clearly $L(M_1) = L(M_1)$.

```
        // find the right end of the string of numbers
q1:     while(currentSymbol == ❑) move(❑, L);
p1:     while(currentSymbol != ❑) move(currentSymbol, R);
        move(❑, L);

        // The rightmost number is of the form 0*1.
          // Move left past the 1, and then left past the 0's
p2:     move(1, L);
p3:     while(currentSymbol == 0) move(0, L);
        if(currentSymbol == ⊢) { // only one number on the tape
          move(⊢, R);
          if(currentSymbol == 1) { // good, it encodes zero
            move(1, R);
          }
        } else { // multiple numbers on the tape
          // mark the leftmost digit of the rightmost number on the tape
p3:       move(1, R);
p5:       move(currentSymbol', L);

          // find the leftmost number on the tape
p6:       while(currentSymbol ∈ {0, 1}) move(currentSymbol, L);
          if(currentSymbol == ⊢) move(⊢, R);

          // Determine which number we are supposed to copy
          while(true) {
            // mark the first digit of the current number
p7:         move(currentSymbol', R);

            // find the marked symbol of the rightmost number
p8:         while(currentSymbol ∈ {0, 1}) move(currentSymbol, R);
            if(currentSymbol == 1′) {
              move(1, R);
              break;
            } else /* currentSymbol == 0′ */
              move(0, L);

p9:         move(currentSymbol', L); // mark the next symbol over

            // move left to the marked number
p10:        while(currentSymbol ∈ {0, 1}) move(currentSymbol, L);
            if(currentSymbol == 1′) move(1, R); // erase the marker
            else {
              move(0, R); // erase the marker
              // now find the next number
p11:          while(currentSymbol == 0) move(0, R);
              if(currentSymbol == 1) move(1, R);
            }
          }

          // copy the marked number to the end of the tape
          while(true) {
p12:        while(currentSymbol ∈ {0, 1}) move(currentSymbol, L);
            if(currentSymbol == 1′) {
              move(1, R);
              break;
          } else /* currentSymbol == 0′ */
              move(0, R);
p14:        while(currentSymbol ∈ {0, 1}) move(currentSymbol, R);
            if(currentSymbol == ❑)
              move(0, L); // append the 0
          }
p13:      while(currentSymbol ∈ {0, 1}) move(currentSymbol, R);
          if(currentSymbol == ❑)
            move(1, L); // append the final 1
        }
q2:     DONE
```

Figure 2: Pseudo-Code corresponding to the Turing machine from figure 1

We can also construct a Turing machine M_2 such that $L(M_2) \neq L(M_1)$. For example, we can construct machines that accept Σ^* and $\emptyset$, and at least one of these must be different than $L(M_1)$. By Rice's theorem, it is undecidable whether or not an arbitrary Turing machine recognizes the same language as M_1.