Pledge: I pledge my honor that I have abided by the Stevens Honor System. -Eric Altenburg

1: (a) Let NONEMPTY = {<M>: Turing Machine M accepts some string}. Show that NONEMPTY is Turing-recognizable. Give a high-level description for your solution. (b) Is NONEMPTY decidable? Prove your answer. If you like, invoke a result discussed in class.

(a)

We construct a TM R that recognizes the language *NONEMPTY*. Let w_1, w_2, ... , w_n be a list of strings from Σ* that M will take as input.

"On input <M>:

```
1. In the following stages, repeat for $i = 1, 2, 3, ...$
2.    Run M for i steps on each input string from $w_1$, $w_2$, ... , $w_n$
3.    If any of the strings are accepted by M, let R ACCEPT."
```

For the language *NONEMPTY*, we need M to see if any of the strings in w_1, w_2, ... , w_n are accepted, however, there is a slight problem with this; the machine can loop on the first string and not evaluate any of the following strings when evaluated sequentially. By following the TM R above, it avoids this problem by looking at all strings and evaluating each one for i amount of steps, where i increases as R continues to run. Once M accepts at least one input string w_i, that means the language of M is not empty, hence R would accept. M can also reject a string saying that its language is still empty, and if that is the case, then M will continue to evaluate all input strings in w_1, w_2, ... , w_n.

(b)

To prove that NONEMPTY is undecidable, we will show that $A_{TM} \leq$ NONEMPTY.

Assume that NONEMPTY is decidable and let a TM R decide it.

Use R to construct another TM S that decides A_{TM}.

S: On input $< M, w >$

```
1. Construct TM $M_1$
      $M_1$: On input x
        if $x \neq w$ REJECT
        else run M(w)
          if it accepts, then ACCEPT
2. Run R($M_1$)
      If it accepts, then ACCEPT
      If it rejects, then REJECT
```

From this, S is proven to be decidable utilizing R which was assumed to also be decidable, however, this should not be possible as A_{TM} is proven to not be decidable, therefore, we have a contradiction. R cannot be decidable.

2: Define L_{17} = {<M> | TM M accepts at least 17 different input strings}. (a) Is L_{17} TM-Recognizable? Prove your answer. (b) is L_{17} TM-decidable? Prove your answer.

(a)

To prove that L_{17} is TM-recognizable, have TM D that recognizes the language, and using an enumerator ϵ that prints out all possible strings:

"On input w:

```
1. For every string $\epsilon$ prints out:
       Run TM M on that string
       If M accepts, then place a marker on the tape of D to signify a string has been accepted
       If M rejects, then go back to stage 1
2. If at any point, the amount of marks on the input tape is at least 17, then ACCEPT, else,
   M will continually run on all the strings from $\epsilon$ looping."
```

This proves that L_{17} is TM-recognizable because it keeps track of all the strings that it accepts using the input that, and if at any point the total amount of accepted strings becomes 17 or greater, then we accept.

(b)

To prove that L_{17} is undecidable, we will show that $A_{TM} \leq L_{17}$.

Assume that L_{17} is decidable and let a TM R decide it.

Use R to construct another TM S that decides A_{TM}.

S: On input $< M, w >$

```
1. Construct TM $M_1$
       $M_1$: On input x
         Run M(w)
         If it accepts, then ACCEPT
            Else, if it rejects, REJECT.
2. Run R($M_1$)
       If it accepts, then ACCEPT
       If it rejects, then REJECT
```

From this, S is proven to be decidable yet again thanks to the assumption that R is decidable, but S is not allowed to be decidable, therefore, there is a contradiction and R cannot be decidable.

3: What is the result of running Ben's program? What was the mistake in Ben's reasoning (other than thinking that the recursion theorem is false)?

The output of Ben's program would be a whole lot of nothing besides just continually constructing its own description. This is due to the way he constructed the program, in stage 2 he says "run B(x)," however, in doing so it will forever compute its own description from the call on itself, thus, never getting to the part of the program where it will accept or reject.

Ben is wrong when he says, "A program like this that contradicts itself cannot exist" because the program never actually contradicts itself. It will forever compute its own description since it runs itself in stage 2, therefore, it will never be able to reach the point in which it accepts or rejects.

4:

(a)

If D accepts <X> then run N on input w

(b)

If D rejects <X> then run M on input w

(3)

Since, in both cases X contradicts D, we conclude that L is undecidable