Problem 1

Give a high-level description of a Turing machine that recognizes the following language:

\[ L = \{w \mid w \in \{0, 1\}^* \text{ and } w \text{ contains at least twice as many 0s as 1s}\} \]

The following questions are lab problems.

Lab Problem 1

A standard Turing machine has a single tape that has a first cell on the left, but extends infinitely to the right. In this problem you will examine two variants of the standard Turing machine. There is no need to provide a formal (low-level) description of any Turing machines you construct; high-level descriptions will suffice.

a. A Turing machine with left-reset instead of left is defined as a Turing machine that, rather than moving either left or right at each step, can either move its head one space to the right or jump all the way to the start of the tape (on the left). Show that such a Turing machine can simulate a regular Turing machine and vice versa.
b. A Turing machine with stay-put instead of left is defined as a Turing machine that, rather than moving either left or right at each step, can either move its head one space to the right or not move at all. Prove that these machines can only recognize regular languages.

Lab Problem 2

A palindrome is a string $W$ with the property that $W = W^R$, where $W^R$ is the reverse of $W$. For example, 010 and 0110 are both palindromes.

a. Give a formal diagram of a Turing machine that recognizes the language of palindromes, with all of its states and transitions. Include an informal description of what your Turing machine does.

b. Analyze the runtime of your TM, in big theta ($\Theta$) notation.

Lab Problem 3

In the Klingon programming language, a program is a finite sequence of lines, each containing exactly one command. The interpreter reads the lines one at a time, and keeps track of two stacks, $A$ and $B$, as well as the current symbol $x$. $A$ and $B$ are strings over the tape alphabet $\Gamma$, and $x \in \Gamma$. As in the case of Turing machines, the input string is over a smaller alphabet $\Sigma$. $\Gamma$ contains a special symbol $\bot$, and $\Sigma$ does not. The input is initially stored as stack $B$ with the first character on top, while stack $A$ is initially empty, and $x$ is initially equal to $\bot$.

The following commands are permitted in the program:

- push $A$: Push $x$ onto stack $A$.
- push $B$: Push $x$ onto stack $B$.
- pop $A$: Replace $x$ with the top symbol of $A$, and delete the top of $A$.
- pop $B$: Replace $x$ with the top symbol of $B$, and delete the top of $B$.
- set <symbol>: Set $x = \langle\text{symbol}\rangle$.
- if <symbol>: Unless $x = \langle\text{symbol}\rangle$, skip the following line.
• **goto <string>:** If there is a label <string> line in the program, go there. If there is none or there is more than one, ignore this command.

• **label <string>:** Ignore and move to the next line.

Note that:

• If **pop** is called on an empty stack, then \( x \) is replaced with the special symbol \( \_ \).

• If the interpreter would read past the last line of the program, it halts.

When (if) the program halts, if \( x = \_ \) then the original string is considered rejected; otherwise it is considered accepted. The language of a Klingon program \( K \) is the set of strings \( s \in \Sigma \) such that \( K \) accepts \( s \).

Prove that Klingon is equivalent to a Turing machine in the following sense:

a. Every Turing machine can be converted to an equivalent a Klingon program.

b. Every Klingon program can be converted to an equivalent a Turing machine.

For either part, if it is more convenient, you may choose to use a Turing machine with a doubly-infinite tape or some other equivalent kind of Turing machine. You don’t have to prove that the kind of Turing machine you use is equivalent to a standard Turing machine, as long as equivalence has been shown in class or in the book.