Collaborator: yc2454 (Yalu Cai)

We can construct a multi-tape Turing machine to solve this in $O(T(n))$. Our Turing machine will have

- an alphabet of {0,1, □, ▷} plus natural numbers 1 through m, with m being the bigger between the number of lines in the program and most number of commands in a line.
- an input tape (containing whatever $\{0,1\}^n$ is the input to the program)
- a work tape we shall call A
- another work tape we'll call $label_{current}$
- another work tape we'll call $command_{current}$
- an output tape

First, we pre-process by copying the first $n + 1$ cells of the input tape to A (the start symbol ▷ plus the n-long input). A will function as the array A in the program. The reason we had to copy the input tape to A is because we need to be able to edit it, which we can't do to the input tape because input tapes are read-only. $label_{current}$'s and $command_{current}$'s second cell characters are set to 0.

Only one cell in $label_{current}$ and $command_{current}$ are in use, and they display the current label and current command within the current label the program is on, respectively. The register's pointers initially point to the ith cell of A after the starting symbol (whatever i is initialized to in the program), the second cells of $label_{current}$ and $command_{current}$ (the only relevant cells in those tapes), and the second cells of the input and output tapes. The only pointer to move is the one pointing to A. **q_{halt} is when the second cell of the output tape is 0 or 1.**

The variable i is maintained in our Turing machine by the pointer to A. Our transition function makes it so that whenever the character at the cell pointed to in A (i.e. $\mathrm{A}[i]$ in the program) equals σ, $command_{current}$ is incremented from 0 to 1, meaning we're going to read the first command in line $label_{current}$. If it doesn't equal σ, then $command_{current}$ stays as 0 but $label_{current}$ increments by one, meaning we're moving on to the next line. So, if $command_{current}$'s character isn't 0 and the corresponding command in the program is to increment i, the transition function moves the pointer to A right by one; if it's a command to decrement i, we move the pointer left by one. If the command is to set $A[i]$ to τ, then write τ to the cell pointed to in A. If the command is to output b and halt, then write b to the cell pointed to in the output tape (the second cell), which will make the Turing machine reach q_{halt}. When we do these, the transition function will also increment $command_{current}$'s character by one (i.e. 56 to 57), meaning we're moving on to the next command in the list of commands on the current line. We don't increment it if it's the last command on the line, in which case the transition function will make it 0 again and increment $label_{current}$. If it's a Goto command, then $command_{current}$ is also set to 0 again and $label_{current}$ is set to the label specified in the Goto.

Our Turing machine applies the transition function an $O(T(n))$ times to reach q_{halt} from q_{start} because each operation in the program corresponds to a constant number of transition function applications in our Turing machine.