605.621 Foundations of Algorithms Spring 2024
Project Assignment #2

1. **Natural Language Description of Algorithm**

   The algorithm starts by accepting an input that defines the DTM's configuration, including symbols, states, and transition functions. It then begins to simulate that DTM, processing input strings and applying stransition rules to modify the tape and machine state as defined. The simulator will attempt to solve specific computational problems, such as addition, subtraction, and multiplication. The algorithm will output detailed transition states.

2. **Pseudocode**

```
SimulateDTM(InputString)
 1  Tape = InputString + BlankSymbol
 2  HeadPosition = 0
 3  CurrentState = 'q0'
 4  TransitionCount = 0
 5  Delta = DTM Operating mode instructions based on type
 6  while CurrentState not in {qY, qN}
 7      ReadSymbol = Tape[HeadPosition]
 8      if TransitionFunction exists for (CurrentState, ReadSymbol)
 9          (NextState, WriteSymbol, MoveDirection) = TransitionFunction(CurrentState, ReadSymbol)
10          Tape[HeadPosition] = WriteSymbol
11          if MoveDirection = 'R'
                HeadPosition = HeadPosition + 1
12          elseif MoveDirection = 'L'
                HeadPosition = HeadPosition - 1
13          CurrentState = NextState
14          TransitionCount = TransitionCount + 1
15          if TransitionCount > 30
                Write Tape to File
16      else
17          Break
18  return Tape, CurrentState
```

3. **Analytically derived Asymptotic family for algorithm**

	SimulateDTM(*InputString*)		
1	Tape = InputString + BlankSymbol	c_1	1
2	HeadPosition = 0	c_2	1
3	CurrentState = q_0	c_3	1
4	TransitionCount = 0	c_4	1
5	Delta = DTM Operating mode instructions based on type	c_5	1
6	**while** CurrentState not in $\{q_A, q_N\}$	c_5	$T(n)$
7	ReadSymbol = Tape[HeadPosition]	c_6	1
8	**if** TransitionFunction exists for (CurrentState, ReadSymbol)	c_7	1
9	(NextState, WriteSymbol, MoveDirection) = delta(CurrentState, ReadSymbol)	c_8	1
10	Tape[HeadPosition] = WriteSymbol	c_9	1
11	**if** MoveDirection = 'R'	c_{10}	1
12	HeadPosition = HeadPosition + 1	c_{11}	1
13	**Else If** MoveDirection = 'L'	c_{12}	1
14	HeadPosition = HeadPosition - 1	c_{13}	1
15	CurrentState = NextState	c_{14}	1
16	TransitionCount = TransitionCount + 1	c_{15}	1
17	**Else**	c_{16}	1
18	Break	c_{17}	1
19	**return** Tape, CurrentState	c_{18}	1

As we can see in the above analysis there is only 1 real variable to the DTM which I consider T(n). As the length of input does not actually affect the speed at which the while loop increases I had to use a different name than n. T(n) can be described as the function of the Turing Machine and its overall time complexity would be O(T(n)). As n only really affects the function T, not directly the time complexity.

4. **Analysis**

   As I determined in my Asymptotic analysis of the pseudocode O(T(n)) seemed to be the best representation of how it ran. This can be seen with a few black box testings I performed:

   Base DTM:
   Input: 1110, Times While Loop ran: 7
   Input: 11110, Times While Loop ran: 7
   Input: 11101, Times While Loop ran: 10
   DTM Addition:
   Input: 100+11, Times While Loop ran: 25
   Input: 111+1011, Times While Loop ran: 67
   Input: 101100101+101101101, Times While Loop ran: 232
   DTM Subtraction:
   Input: 100-10, Times While Loop ran: 25
   Input: 101-10, Times While Loop ran: 25
   Input: 101111-101, Times While Loop ran: 43
   DTM Multiplication:
   Input: 1*1, Times While Loop ran: 295
   Input: 10*10, Times While Loop ran: 557
   Input: 1110*10, Times While Loop ran: 825

   As we can see, while input size can make a difference to the DTM Runtime, what truly makes a difference is what instructions it is using and what is input, neither of which can easily be graphed in a simple n format.

