Turing Machine

Jan Nogué

January 2023

1 Description

We present a theoretical analysis of a Turing Machine that, given an unsorted array of binary numbers, returns the array sorted in ascending order. The code used to create the file.txt with the transition functions, examples with different inputs and overall code used to perform this work can be found on my Github. The high-order implementation of such a TM is

1. The TM looks for the global minimum in the Input Tape.
2. Copies the global minimum to the Output Tape
3. Erases the global minimum in the Input Tape and restarts until reaching the Halt state.

The Input alphabet for this TM is $\Sigma = \{1, 0, S, E, X, H\}$. The S,E characters denote the beginning/end of a Tape. The X character is used to track the global minimum's position in the Input Tape, and the H character yields the TM to the Halt state.

Following the instructions, we supposed that the head on the Turing machine moves left (L) / right (R) direction (D) or doesn't move (*) upon reading a symbol on the Tape. The other actions of the transition functions are Read (R) and Write (W). The $\downarrow$ denotes the position of the Head.

1. Input Tape: Contains the unsorted binary numbers separated by an underscore; the first and last characters are S/E. Actions $= \{R_I, W_I, H_I\}$. For example

$$\overset{\downarrow}{S}\,10\_00\_11E$$

2. Working Tape: It temporarily stores the numbers of the Input Tape to compare with each other. It's initialised with the same lenght of the Input Tape. An advanced version of the algorithm would initialise it with the size of the largest Input number to minimise the memory used. Actions $= \{R_W, W_W, H_W\}$. Initially

$$\overset{\downarrow}{S}\,\_\_\_\_\_\_\_\_\_\_\_\_\_\_E$$

3. Track Tape: It's initialised with the same length as the Input Tape. Its function is to keep track of the position in the Input Tape of the number copied to the Working Tape. If the number copied is found to be the global minimum, we will use this Tape to erase the number from the Input Tape. Actions $= \{R_T, W_T, H_T\}$. Initially

$$\overset{\downarrow}{S}\,\_\_\_\_\_\_\_\_\_\_\_\_\_\_E$$

4. Output Tape: It's also initialised with blank spaces the same length as the Input Tape. It will contain the sorted array of binary numbers. Actions = $\{R_O, W_O, H_O\}$.

$$\overset{\downarrow}{S}\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_H$$

In this algorithm we made that the Head of the Input Tape (H_I) and the Head of the Track Tape (H_T) move accordingly. Therefore, the algorithm only has eleven variables

$$\{R_I, W_I, H_I, R_W, W_W, H_W, R_T, W_T, R_O, W_O, H_O\}$$

2 States and Transitions functions

For clarity, in each transition table, we will only present the actions used in the state. The default values for the actions that do not appear in the tables are:

1. Head: Stays in the same place.
2. Write: Writes the same that has been read.
3. Read: Reads the same character every time (the Head does not move)

2.1 Intial States

The root of the algorithm is the state q_0, which reads the Input Tape.

q_0 (Read Input Tape)		
R_I	H_I	State
_	R	q_0 (Read I)
E	*	q_8 (Read O Tape)
1/0	*	q_1 (Read W)
S	R	q_0 (Read I)

q_1 (Read Working Tape)		
R_W	H_W	State
_	*	q_2 (copy)
1/0	*	q_3 (compare)
S	R	q_1

Table 1: Transitions functions of the state q_0 and q_1

1. If it reads 1/0, it goes to read the Working Tape (State q_1). If it reads the character "E" in the Input Tape, it goes to the state q_8 (read Output Tape).
   (a) If the Working Tape is empty, it copies the number from the Input Tape (State q_2 - copy), resets H_I, H_W to S (states q_{2A}, q_{2B}) and starts rereading the Input Tape (state q_0).

q_2 (Copy)						
R_I	H_I	W_W	H_W	W_T	H_T	State
1/0	R	1/0	R	X	R	q_2 (copy)
_	*	_	*	_	*	q_{2A} (Reset H_W)
E	*	*	*	*	*	q_8 (Read Output Tape)

Table 2: Transitions functions of the state q_2, which copies the digit read in the Input Tape to the Working Tape.

q_{2A} Reset H_W		
R_W	H_W	State
1/0	L	q_{2A}
_	L	q_{2A}
S	*	q_{2B}

q_{2B} Reset H_I		
R_I	H_I	State
1/0	L	q_{2B}
_	L	q_{2B}
S	*	q_0

Table 3: Transitions functions of the state q_{2A}, q_{2B} which reset the Head of both Input and Working Tapes.

   (b) Otherwise, it means that the Working Tape has already a number written. The TM moves to the compare state (q_3) to determine which number is smaller.

2. If it reads the character E, it has finished reading the Tape and moves on to check if the Output Tape is full (State q_8)

   (a) If it is, it goes to Halt state (State q_{10}).

   (b) If the Output Tape is empty copies the number from the Working Tape to the Output Tape and starts over (State q_9).

2.2 Compare State

In this state, we compare the digits from the Input State with the digits of the number copied in the Working Tape. Let A be the number from the Input Tape and B the number from the Working Tape.

Since the numbers are the same lenght, we place both Heads to the first digit on the left and move to right comparing digit by digit (let D_i with $i \in A, B$ be the digit of the Tape A or B). We determine which one is smaller following the next logic:

1. If $D_A = D_B$, with $D_{A,B}$ being 0 or 1, move to the next digit
2. If $D_A = 1$ and $D_B = 0$, $A > B$
3. If $D_A = 0$ and $D_B = 1$, $B > A$.
4. If $D_A = D_B = \_$, then $A = B$ and we move to the next number in the Input Tape.

q_3 (Compare) A (Input) vs B (Working)				
R_I	H_I	R_W	H_W	State
1	R	1	R	q_3 (compare)
0	R	0	R	q_3 (compare)
0	*	1	*	q_6 (A<B)
1	*	0	*	q_4 (A>B)
_	*	_	*	q_4 (A=B)

2.2.1 A>B and A=B

In all cases where $A > B$ and the case where $A = B$, move the H_I to the right until it finds an underscore (next number in the Input Tape). Then reset H_W (State q_{4A}) and start over. If it finds the character E, the TM goes to the state q_8 (to write the number in the Output Tape)

q_{4A} (Next number H_I)		
R_I	H_I	State
1/0	R	q_{4A}
_	*	q_0
E	*	q_8

2.2.2 A<B

In the case where $A < B$, we erase the Working Tape and copy the new number:

1. First move H_W to right until found first blank space (state q_6). Then erase the Working Tape (state q_{6A})

q_6			
R_W	W_W	H_W	State
1/0	1/0	R	q_6
_	_	R	q_6

q_{6A}			
R_W	W_W	H_W	State
1/0	_	L	q_{6A}
_	_	L	q_{6A}
S	S	*	q_{6B} (Reset H_I)

2. Move H_I to the left until found first underscore.

q_{6B} (Reset H_I)		
R_I	H_I	State
1/0	L	q_{6B}
_	*	q_0 (Read Input Tape)

3. After state q_{6B}, we move to the state q_0 (read Input Tape), it will read the number and move to copy state ($q_2 - q_{2A} - q_{2B}$).

2.3 End of the Input Tape

When the TM reads the character E, it means that it has reached the end of the Input Tape and moves to state q_8. At this point, the current global minimum is copied in the Working Tape. The TM reads the current character in the Output Tape:

1. If it reads the character E, it means that the Output Tape is full and contains the sorted numbers. The TM goes to the Halt state (State q_H).

2. If it reads an underscore, it copies the number from the Working Tape to the Output Tape and resets the algorithm.

q_8 Read Output Tape		
R_O	H_O	State
_	*	q_9 (copy output Tape)
S	R	q_8
H	*	q_H (Halt)

We divide the state q_9 in the subsections:

1. Copy the number in the Working Tape to the Ouput Tape. To do so we need to
   (a) Reset the H_W (state q_9)
   (b) Copy the Working Tape number to the Output Tape (state q_{9A})

q_9 (Reset H_W)		
R_W	H_W	State
1/0	L	q_9
_	L	q_9
S	*	q_{9A}

q_{9A} Copy to Output Tape				
R_W	H_W	H_O	W_O	State
1/0	R	R	1/0	q_{9A}
_	*	R	_	q_{9B} (Erase copied in Input Tape)
S	R	S	R	q_{9A}

2. Erase the Working Tape number from the Input Tape (States $q_{9B} - q_{9E}$).
   (a) Move left simultaneously H_T, H_I until H_T reads X (State q_{9B}).

q_{9B}(Move H_I, H_T to same position)					
R_T	H_T	R_I	H_I	W_I	State
E	L	E	L	E	q_{9B}
_	L	1/0	L	1/0	q_{9B}
_	L	_	L	_	q_{9B}
X	*	1/0	*	1/0	q_{9C}

   (b) For each X character, put an underscore in the Input Tape (thus deleting the global minimum from the Input Tape) (State q_{9C}). Erase and reset Working Tape (State q_{9D}).

q_{9C} Erase Input Tape						
R_T	W_T	H_T	R_I	W_I	H_I	State
X	_	L	1/0	_	L	q_{9C}
_	_	*	_	_	*	q_{9D}

q_{9D} Reset Working Tape			
R_T	W_T	H_T	State
X	_	L	q_{9D}
_	_	L	q_{9D}
S	S	*	q_{9E}

   (c) Reset Input and Track Tape (State q_{9E}).

q_{9E}						
R_I	W_I	H_I	R_T	W_T	H_T	State
1/0	1/0	L	X	_	L	q_{9E}
1/0	1/0	L	_	_	L	q_{9E}
_	_	L	_	_	L	q_{9E}
S	S	*	S	S	*	q_0

3 Complexity Analysis

Sorting algorithms are a fundamental part of computer science, and are used in a wide range of applications. There are several different sorting algorithms, each with its own strengths and weaknesses, here we present the fundamentals [1]

1. Bubble sort: Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. Bubble sort has a time complexity of $\mathcal{O}(n^2)$, making it inefficient for large data sets.

2. Selection sort: Selection sort works by dividing the unsorted list into two parts: the sorted part and the unsorted part. The algorithm repeatedly selects the minimum element from the unsorted part and moves it to the end of the sorted part. The time complexity of selection sort is also $\mathcal{O}(n^2)$.

3. Insertion sort: Insertion sort is similar to selection sort, but instead of selecting the minimum element, it inserts elements one by one into the sorted part of the list. The time complexity of insertion sort is also $\mathcal{O}(n^2)$.

4. Merge sort: Merge sort is a divide-and-conquer algorithm that recursively splits the data set into smaller parts, sorts each part, and then merges the sorted parts back together. The time complexity of merge sort is $\mathcal{O}(n \log n)$, making it an efficient and scalable sorting algorithm.

5. Quick sort: Quick sort is another divide-and-conquer algorithm that works by selecting a pivot element from the data set and partitioning the other elements into two parts, according to whether they are less than or greater than the pivot. The time complexity of quick sort is $\mathcal{O}(n \log n)$ on average, but its worst-case time complexity is $\mathcal{O}(n^2)$.

6. Heap sort: Heap sort is a comparison-based sorting algorithm that works by building a binary heap data structure, which is then used to sort the elements. The time complexity of heap sort is $\mathcal{O}(n \log n)$, making it an efficient and scalable sorting algorithm.

7. Radix sort: Radix sort is a non-comparison-based sorting algorithm that works by sorting the data set digit by digit, from least significant digit to most significant digit. The time complexity of radix sort is $\mathcal{O}(nk)$, where k is the number of digits, making it an efficient sorting algorithm for data sets with a small number of digits.

As seen, most of this algorithms have a worse case time complexity of $\mathcal{O}(n^2)$. We choose our algorithm to find the global minimum because, as we will see, in the worse (average) case scenario, it has a time complexity of $\mathcal{O}(n^3)$ ($\Theta(n^2)$). We decided to perform our algorithm because it performs on average like the worse case of most of the sorting algorithms so we would have the higher complexity possible between all sorting algorithms.

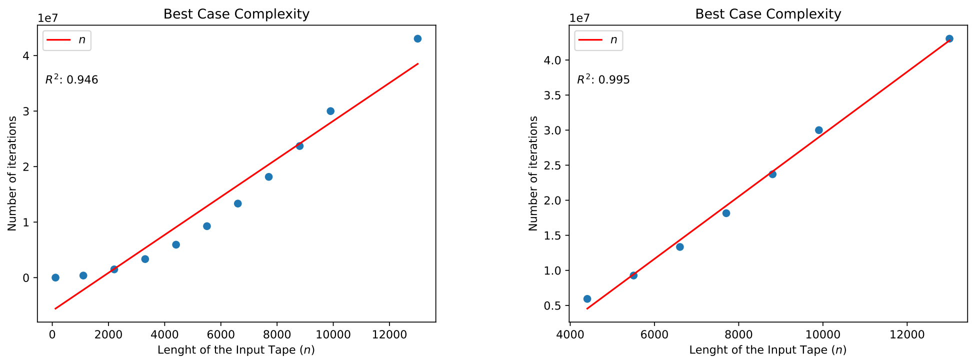


Figure 1: Plots for the Best Case Complexity

3.1 Space complexity

Regarding the space complexity, this algorithm is inefficient because it initialises the four tapes with the same lenght. While the Track Tape and the Output Tape need to be the same size than the Input Tape, the Working Tape only needs to be of the size of the largest sequence of bits from the Input Tape. As mentioned, an advanced version of the algorithm would look for the largest sequence of bits, before initialising the other tapes. This would enter in the phase of pre processing the data. In our case, each tape is used independently to store data. Since the tapes are of lenght n, we need $4n$ of space memory resulting in a space complexity of $\mathcal{O}(n)$.

3.2 Time complexity

For the time complexity analysis, we will consider three scenarios. The best case scenario, the average case scenario and the worse case scenario. Let n be the lenght of the Input Tape.

1. **Best case scenario:** In the best case scenario, the input tape is already sorted in ascending order. The algorithm reads the first element, copies it in the Working Tape and continues until reaching the end of the Input Tape. In this best case scenario, the algorithm only reads the tape once. Therefore, the time complexity is $\mathcal{O}(n)$.

   In FIG.1 we plotted the iterations of different sized already sorted input arrays. In the first plot we can see that for small sizes of input tape the algorithm doesn't evolve like $\mathcal{O}(n)$.
   In the second figure we plotted the iterations as a function of the Lenght of the Input Tape for larger Input Tapes and confirmed that it evolves like $\Omega(n)$. Thus, we can conclude that the time complexity of the algorithm evolves like $\mathcal{O}(n)$ for the best case scenario.

2. **Worse case scenario:** In the worse case scenario, the input array is sorted in decreasing order. To find the global minimum, the algorithm runs the tape twice, since the smaller number is the last element of the tape. We get that for each iteration, the time complexity is $\mathcal{O}(n^2)$. After an iteration, the algorithm erases the global minimum from the Input Tape

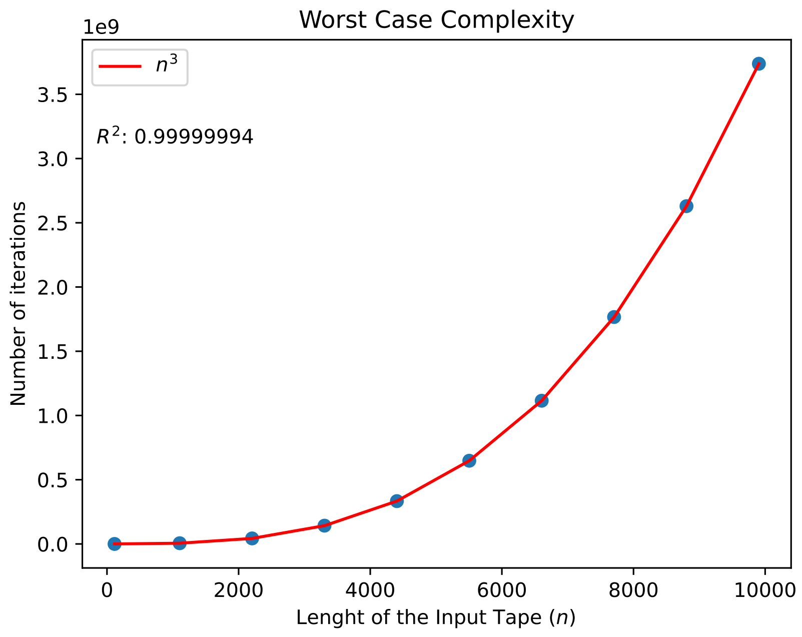


Figure 2: Number of iterations as a function of the Input Tape's length for the worse case scenario.

and restarts the process. Therefore, the total time complexity is $\mathcal{O}(n^2 \cdot (n-1)) = \mathcal{O}(n^3)$.

In FIG.2 we plotted the number of iterations as a function of the Input Tape's length for the case where the Input Tape was already sorted in decreasing order. We can see that the time complexity evolves like $\mathcal{O}(n^3)$ for the worse case scenario.

3. **Average case scenario:** In the average case scenario, half the elements need to be read to find the global minimum, resulting in a $\mathcal{O}(n/2)$ time complexity per iteration. Therefore, the total time complexity is $\Theta(n^2)$.

   In FIG.3 we plotted the number of iterations as a function of the Input Tape's length for the case where the Input Tape was unsorted. We can check that the time complexity evolves like $\mathcal{O}(n^2)$ for the average case scenario.

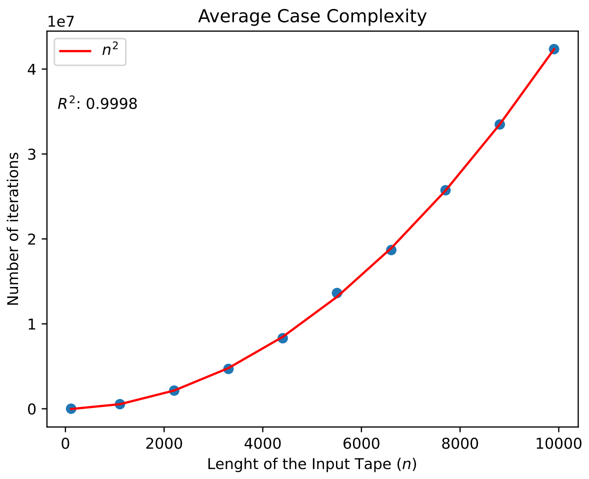
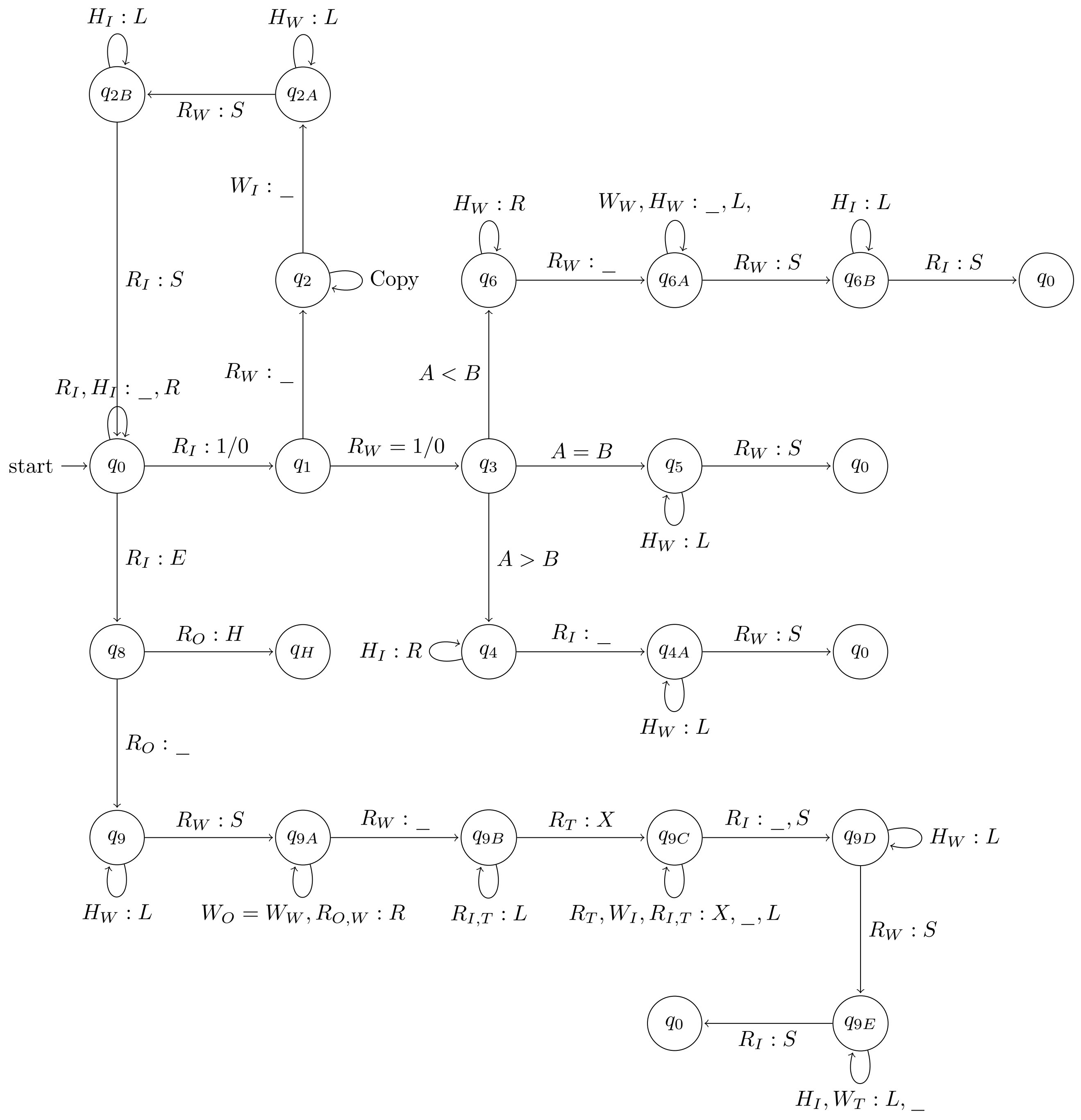


Figure 3: Number of iterations as a function of the Input Tape's length for the average case scenario.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

4 Diagram of the Turing Machine


$H_I : L$
$H_W : L$
q_{2B}
q_{2A}
$R_W : S$
$W_I : \_$
$H_W : R$
$W_W, H_W : \_, L,$
$H_I : L$
$R_I : S$
q_2
Copy
q_6
$R_W : \_$
q_{6A}
$R_W : S$
q_{6B}
$R_I : S$
q_0
$R_W : \_$
$A < B$
$R_I, H_I : \_, R$
start
q_0
$R_I : 1/0$
q_1
$R_W = 1/0$
q_3
$A = B$
q_5
$R_W : S$
q_0
$H_W : L$
$R_I : E$
$A > B$
q_8
$R_O : H$
q_H
$H_I : R$
q_4
$R_I : \_$
q_{4A}
$R_W : S$
q_0
$H_W : L$
$R_O : \_$
q_9
$R_W : S$
q_{9A}
$R_W : \_$
q_{9B}
$R_T : X$
q_{9C}
$R_I : \_, S$
q_{9D}
$H_W : L$
$H_W : L$
$W_O = W_W, R_{O,W} : R$
$R_{I,T} : L$
$R_T, W_I, R_{I,T} : X, \_, L$
$R_W : S$
q_0
q_{9E}
$R_I : S$
$H_I, W_T : L, \_$