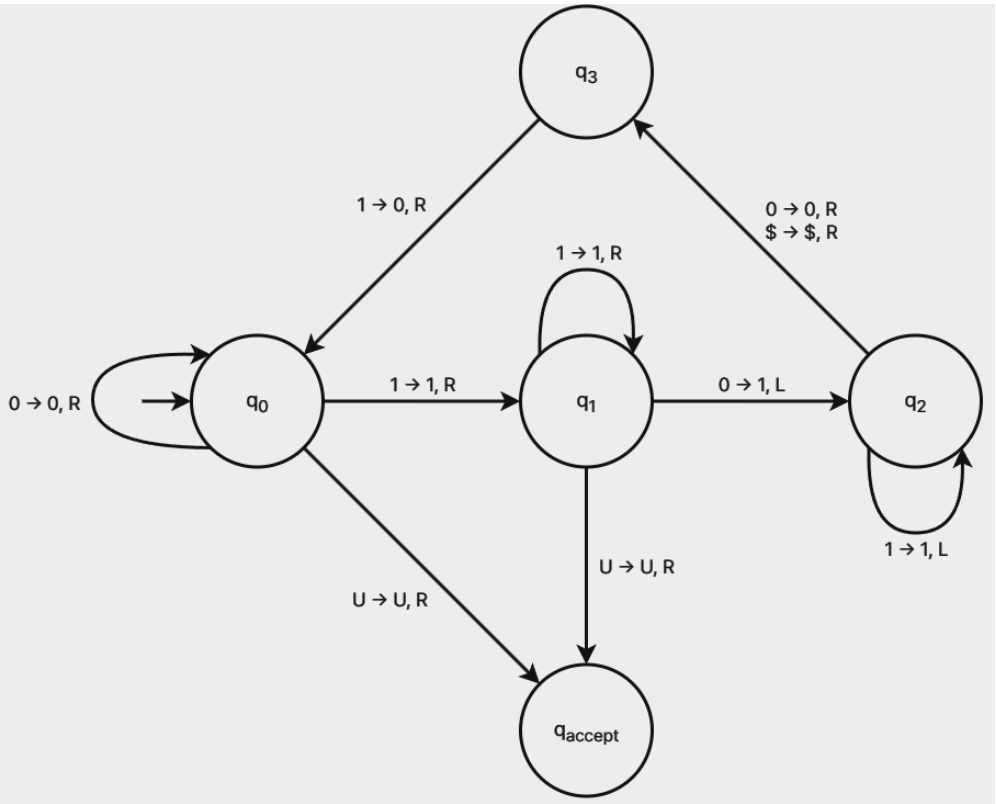
Theory of Computation '23 Quiz 3
Time : 50 minutes

Name : **Roll :** **Marks :**

Problem 1. (10 points) In the lectures, we have seen Turing Machines that can recognize/decide languages. In this problem, you will design a Turing Machine that can 'compute'. Given a binary string s in $\{0,1\}^\star$, we define a function $SORT(s)$ which takes s as input and outputs another binary string s' such that s' is of the form $0^\star1^\star$ and the number of 0s and 1s in s' is the same as those in s. Give the **state diagram level** description of a single tape TM that implements the function $SORT(s)$. For simplicity, assume that the TM always gets a valid binary string as input (including ε) along with a left-end marker \$ and goes to an 'accept' state only when the tape has $SORT(s)$ written on it. You cannot use more than 4 states (excluding the 'accept' state, you may skip showing the 'reject').

Solution.



Any string $s \in \{0,1\}^*$ can be written as $(0^*1^*)^*$. So, the idea is as follows:

1. Read the initial 0^*1^*.
2. If, at this point, you encounter a space, we accept the string – this is the sorted version of the string.
3. If we encounter a 0 after the initial 0^*1^+, we know it is in an unsorted location. So, we cross out the 0, and write a 1, and start moving left.

4. We move left until we encounter the last 0 in the 0^*1^+ sequence and replace the first 1 with a 0 (this is the swapping step - at step 3., we replace 0 with an extra 1, and at 4., we replace a 1 with a 0.

5. Repeat steps 1. through 4. until the string is sorted.

A turning machine with more than these 4 states is simpler to come up with.

Rubric:

- **+10** for *any* correct construction with not more than 5 states (excluding q_{acc})
- **+4 to +7** for partially correct TMs [i.e. logic can be understood)
- **-0.5** for every two incorrect/missing transitions

Problem 2. (10 points) A **write-once** TM is exactly same as a vanilla TM except that each tape-cell can be *modified* only once. Show that a write-once TM is equivalent to an ordinary TM (ordinary means 'as done in lecture'). An implementation level description (that is clearly stating how the head behaves over the tape) is enough in your proofs (Hint: First try write-twice. This will fetch you 50% credit).

Solution.

The idea is to keep an *active* work area over the tape, read the input string, and repeatedly copy the *updated* string contents and allocate another work area further down the tape. Note that simply copying uses 1 write operation on a cell.

Write-Twice TM:
Let M be a regular Turing machine whose equivalent write-twice machine M_{W_2} we intend to construct. M may update its input string as $s \mapsto M^{(1)}(s)$ in the first step. M_{W_2} reads s and writes $M^{(1)}(s)$ on a fresh work area A_1 on the tape. It must also scratch out all cells in the previous active area, say A_0, to mark it as *used*, which uses 1 write operation per cell.
In its second step, M updates its input to $M^{(2)}(s)$. M_{W_2} reads $M^{(1)}(s)$ from A_1, writes $M^{(2)}(s)$ on a fresh area A_2, and scratches off the cells in A_1. It is not hard to see that in the i^{th} step, $s \mapsto M^{(i)}(s)$ through M, so M_{W_2} reads $M^{(i-1)}(s)$ from A_{i-1} and writes $M^{(i)}(s)$ on an area A_i. If M halts in n steps, we repeat this for n steps. If M does not halt on s, so won't M_{W_2}.
It is clear to see that M_{W_2} uses at most 2 writes per cell on its tape. Moreover, the above description implicitly entails the operations performed by its head.

Write-Once TM:
At the i^{th} step, instead of the verbose notation above, we call $M^{(i-1)}(s)$ the *old* string and $M^i(s)$ the *updated* string. Now, instead of copying the updated string to the next active area, we use twice as many cells and write each character of the updated string spaced by one cell (i.e. only write on consecutive cells). These *blank* cells will be used to identify whether the preceding cells have been *used* or not. Hence, each cell will be written on exactly once!
Formally, any regular Turing machine M can be converted into an equivalent write-once machine M_{W_1} as follows. M_{W_1} reads input (the old string) s and writes the updated string on an area A_1 spaced by one cell and scratches off all cells in A_0. Following this, it reads blank-separated old

strings from A_{i-1} and writes blank-separated updated strings to A_i and scratches off the blank cells to indicate that the area has been used.

This problem actually implies something – you can convert any write-k-times TM to an equivalent write-$(k-1)$-times (and hence, write-once) TM, and using a similar strategy, you can show that all such variants are equivalent to the vanilla TM model!

There are other ways to prove this equivalence. Any **mathematically sound** proof will be given **full credit**.

Rubric:

- **+3** for correctly proving simulation of write-once TM on vanilla TM.
- **+3.5** for correctly proving simulation of vanilla TM on write-twice TM (the update and copy idea), and **+0.5** to **+3** for partially correct proofs. **+3.5** for correctly proving write-twice TM on write-once TM (the idea of leaving blank cells and simulating write-twice), and **0.5** to **+3** for partially correct proofs.
- If simulation on write-twice TM hasn't been attempted, **+7** for correctly proving simulation on write-once TM. **+0.5** to **+6.5** for partially correct solutions.