Theory of Computation, CSCI 438 spring 2022
Introduction to Complexity Theory and Complexity Relationships among Models, pg. 275-284, April 22nd

Let $A=\{0^k1^k \mid k \geq 0\}$

1. Find the time complexity of deciding A using a single tape Turing machine.

M= "On input w where $w \in \{0,1\}^*$

1. If 1 reject. If blank accept. If 0, replace by \$ and move right across 0s and 1s until reach blank.
2. Move left and replace 1 with a blank. If was a 0, reject. If was a \$, reject.
3. Move right across 1s and 0s until hit \$.
4. Move right and go to step 1."

Analyze the time complexity of this Turing machine algorithm. The loop is executed n/2 times and each step is O(n), giving $O(n^2)$.

(This also makes sense using n + (n-1) + (n-2) + … + 1 which is $\frac{n(n+1)}{2}$ in our case we have n+(n-2) + (n-4) + … + 2 which is $\frac{n(n+2)}{2}$.)

2. If your above algorithm runs in $O(n^2)$, define an algorithm that runs in $O(n\mathrm{Log}_2 n)$.

M= "On input w where $w \in \{0,1\}^*$

1. Scan over the input to make certain that no 0 appears after a 1.
2. Scan across the tape to see if the total number of symbols on the tape is odd and if so reject. If the tape is blank or only marked symbols are on the tape, accept.
3. Beginning with the first 0, mark out every other 0. Beginning with the first 1, mark out every other 1. (Note that if the parity of the number of 0s and the number of 1s is not the same, a different number of 0s will be marked off than the number of 1s.)
4. Go back to step 2."

This works because the number of 0s and 1s will keep being cut in half, and this will happen equally as long as the number of 0s and the number of 1s have the same parity.

Step 1 O(n)
Step 2 O(n).
Step 3 O(n)

Repeat steps 2 and 3 $\log_2 n$ times since each time the number of 0s and s is cut in half.

Overall $O(n\log_2 n)$

It can be shown that this is the best that can be done using a single tape Turing machine.

3. Find the time complexity of deciding A using a 2-tape Turing machine.

M= "On input w where w$\in$\{0,1\}*

1. Mark front on 2nd tape.
2. Travel right, copying 0s to second tape.
3. When see 1, travel right across 1s on first tape and left across 0s on second tape.
4. If see blank on first tape and \$ on second tape at the same time, accept; anything else, reject."

Notice that this is O(n) running time. So, as stated earlier, the model makes a difference for running times.

4. Describe why the following theory makes sense:
   Theorem 7.8 (page 282)
   Let t(n) be a function, where $t(n) \geq n$. Then every t(n) time multi-tape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

   Consider the method we used to simulate a multi-tape TM on a single tape. Simulate the multi-tape TM on the single tape by placing the significant contents of each tape, separated by a new tape symbol, say #, onto the single tape. By significant contents, use at least one blank for each tape. If the tape has non-blank symbols, keep those on the single tape, along with all interspersed blanks.

   Simulate the position of each read/write head on the tape by introducing the new symbols x-dot for every $x \in \Gamma$ where x-dot indicates that the read/write head is at that location.

   To simulate each move: Read across the tape to see what is under each read/write head in order to decide which transition to perform. Perform the transition on each tape by writing the appropriate symbol, unmarking the symbol where the read/write head was, and marking the symbol where the read/write head is after the move.

   Thus, for each move, the contents of the tape were traversed twice, making this simulation require $O(t^2(n))$ time, where execution on the multi-tape machine required t(n) time.

5. Describe why the following theory makes sense:
   Theorem 7.11 (page 284)
   Let t(n) be a function, where $t(n) \geq n$. Then every t(n) time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic Turing machine.

   Consider the method we used to simulate a non-deterministic TM on a 3-tape machine.

   High Level Plan

   View the deterministic execution of a particular string by a particular nondeterministic Turing machine as the exploration of a search tree with states as vertices, symbols on the branches, and branches representing moves in the nondeterministic Turing machine. When a state and symbol in the nondeterministic machine have multiple choices, say x choices, corresponding vertex in the tree will have x children, one for each choice. When a state and symbol in the nondeterministic machine has only 1 choice, the corresponding vertex will have only 1 child. Accept and reject vertices do not need to have children.

   The deterministic TM can simulate the nondeterministic TM by traversing this search looking for an 'accept' vertex. The deterministic TM will find an 'accept' vertex exactly when the nondeterministic TM will accept. When the deterministic TM finds a 'reject' vertex, it will not stop, rather it will continue to search because, a different branch may lead to an accept.

   Thus, if at any point the deterministic machine enters an accept state, it accepts. If it enters a reject state, it explores the next branch. Thus, the deterministic TM does a breadth-first search of all possible executions. If one of these enters an accept state, accept. Otherwise, an exhaustive search may occur and the deterministic TM may reject, or the deterministic machine may search forever.

   More Detailed Plan

   Use a 3-tape machine to accomplish the breadth-first search.
   1. Input tape: holds the original input and is never changed.
   2. Simulation tape: For each simulation, a fresh copy of the input is copied onto this tape. A deterministic Turing machine is simulated on the tape, choosing the transitions indicated on the progress tape.
   3. Progress tape: This tape controls the breadth-first search. It holds what moves to try next. The progress tape tells what transition to make each move, making a particular simulation deterministic.

   Once all of the choices have been pursued at the given level, explore the nodes at the next level.

For each state/symbol combination, number the transitions 1, 2, … n. If there is only one choice for a state/symbol pair, it will be numbered 1.

Initially the progress tape holds 1. The simulation is run, using the 1st rule of the start symbol and the tape symbol under the read/write head. If an accept state is entered, the string is accepted. If an accept state is not entered, the progress tape is updated and a new simulation is started, beginning back at the start state, and with fresh input on the simulation tape.

Updating the progress tape: If there is another move to make for the state/symbol combination, that position on the tape is updated. If there is not another move, the position on the tape is set to 1, and the previous position on the tape is updated. If there is not another move at that position, that position is set to 1 and the previous position on the tape is updated. If we get back to the first position, and there are no more moves for that position, add a new 1 to the end of the progress tape. This indicates that the search will go another transition further, i.e. another level deeper on the tree.

With considerable thought, it can be seen that this new deterministic TM will have a finite number of states, and will accept exactly those strings as were in the language of the nondeterministic machine.

It also makes sense that this simulating can be done in $2^{O(t(n))}$ time.

6. Exercise 7.5

Is the following formula satisfiable?

$$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y})$$

There are many ways to solve this. I'll use a truth table.

x	**y**	**statement**
T	T	$T \wedge T \wedge T \wedge F \equiv F$
T	F	$T \wedge T \wedge F \wedge T \equiv F$
F	T	$T \wedge F \wedge T \wedge T \equiv F$
F	F	$F \wedge T \wedge T \wedge T \equiv F$

There are no truth values for x and y which make this statement evaluate to T. Thus, this statement is not satisfiable.

8. Using the fact that A_{TM} is not decidable, prove that HALT is not decidable.

$HALT_{TM}$ = {<M,w> | M encodes a TM, w is a string in the alphabet of M, and M halts on w}

Proof: Suppose, by way of contradiction, that $HALT_{TM}$ is decidable. Then there is a TM, call it H, that decides it. Using H, the following TM is definable:

D = "On input <M,w> that encodes a TM and w in the alphabet of M

1. Feed <M,w> into H. If H says that M doesn't halt on w, reject. If H says that M does halt on w, then simulate M on w and output the result."

This new TM decides A_{TM}. However, A_{TM} is not decidable. This contraction shows that our assumption was wrong. $HALT_{TM}$ must not be decidable.