Exercise One

Remarks: After describing a program for a Turing machine, add a brief analysis of the number of head moves it does on a size \( n \) input, on the worst case.

1. Question 1: Write a program for a Turing machine that recognizes the string \( 0^n 10^n \).
   The input is legal in the sense that it starts with a (perhaps empty) string of 0. Then, it has a 1 (the 1 appears there in any case) and then another (perhaps empty) string of 0. The goal of the program is just to check if there is the same amount of 0 to the left and to the right of the 1.
   
   Answer: We change the leftmost 0 to \( x \). Then, we go for the rightmost 0 and change it to \( y \). Then we iterate in the same way. This is a way to count that there is the same number of 0 to the left and to the right of 1.
   
   The general state of the string will be:
   \[
   x x x \ldots 0 0 \ldots 1 0 0 0 \ldots 0 y y \ldots y.
   \]

   Stopping conditions 1: If when we go right searching for 0 we see a \( y \) then there are more 0 to the left of the 1 than to its right.

   Stopping condition 2: The machine changed a 0 to \( y \), went back to find an \( x \) and moved one step right to find the leftmost 0. If the head now points to 1 this means that we have changed all the left 0 to \( x \). Thus, we need to go and check that there is no more zero to the left of the leftmost \( y \).

   The states: The states are
   \[
   Q = \{q_0, q_1, q_2, q_3, q_4, q_N, q_Y\}.
   \]
   
   (a) State \( q_0 \): The state \( q_0 \) is the initial state with the head pointing at the leftmost 0. But more generally, the head at \( q_0 \) points to the leftmost 0 that was not changed to \( x \). In state \( q_0 \) the 0 is changed to \( x \). The machine then switches to state \( q_2 \).
   
   A special case is when in state \( q_0 \) the head “sees” a 1. This means that all the 0 to the left of the 1 were exchanged by \( x \). Thus, we need to see that no 0 remain unchanged to the right of the 1. The machine switches in this case to state \( q_1 \) and moves one cell to the right. Immediately to the right of the 1 we have to meet a \( y \) otherwise we meet a 0 and we go to \( q_N \) (rejecting state).

   (b) State \( q_1 \): State \( q_1 \) stops and reject if it sees a 0. Because there are more 0 to the right of the 1 than to its left. The rejecting state is \( q_N \). If in \( q_1 \) we see a \( y \), the machine accepts (moves to \( q_Y \)).
(c) **State** \( q_2 \): This state enters into play if we changed the currently leftmost 0 to \( x \). This state keeps moving to the right to find the rightmost 0. This is done by finding a \( y \) and then going one step to the left. After that, the machine switches to state \( q_3 \).

(d) **State** \( q_3 \): When we are in state \( q_3 \), the head is supposed to be looking at a 0. If at state \( q_3 \) we encounter the 1, this means that there are more 0 to the left of 1 then to the right of the 1. Then we move to a rejection state \( q_N \). If we meet a 0, then it is change it to \( y \) move the head one step left and switch to state \( q_4 \) which is a “go back to the beginning of the string” state.

(e) **State** \( q_4 \): This is a returning left state. This state returns to the rightmost \( x \) and then moves one step to the right and switches to state \( q_0 \). The symmetry is completed, and the machine iterates.

The transitions:

\[
(q_0, 0) \mapsto (q_2, x, R)
\]

\[
(q_0, 1) \mapsto (q_1, 1, R)
\]

\[
(q_1, 0) \mapsto (q_N, 1, R)
\]

\[
(q_1, y) \mapsto (q_Y, y, R)
\]

\[
(q_2, 0) \mapsto (q_2, 0, R)
\]

\[
(q_2, 1) \mapsto (q_2, 1, R)
\]

\[
(q_2, y) \mapsto (q_3, y, L)
\]

\[
(q_3, 0) \mapsto (y, q_4, L)
\]

\[
(q_3, 1) \mapsto (1, q_N, L)
\]

\[
(q_4, 0) \mapsto (0, q_4, L)
\]

\[
(q_4, 1) \mapsto (1, q_4, L)
\]

\[
(q_4, x) \mapsto (x, q_0, R)
\]

For an input of size \( n \) the TM performs \( O(n^2) \) moves.

2. **Question 2**: Write a program that gets as input a string of the form \( 0^m1^n \). If \( m > n \) it leaves \( 0^{m-n} \) zeros on the tape. Otherwise, it leaves the empty string (note that the 1
is erased in any case). As in Question 1, you do not need to check that input, namely, it starts with a string of 0 and ends with a string of 0 and it has 1 in the middle.

**Answer:** The machines operates by replacing the 0 to the left of 1 by $B$ (namely, by a blanc) and replacing the 0 to the right of 1 by 1. A typical state of the tape is:

$$B B \ldots B 0 0 \ldots 0 1 1 1 \ldots 1 1 0 0 0 \ldots 0.$$  

A general loop of the machine is described as follows: It changes the leftmost 0 by $B$, and then goes to the right and finds the rightmost 1, and then points to the 0 immediately to its right. Then it replaces this 0 by 1 and returns to the left to find the leftmost $B$. Then moves one step to the right to point to the leftmost 0.

**First stopping conditions:** The first stopping condition is when searching right for a 0, the machine finds a blanc, $B$ immediately to the right of the rightmost 1. This is the case that all the $n$ 0 to the right of the 1 were replaced by 1. In addition, $n + 1$ 0 to the left of the 1 were replaced by $B$. Note that we say $n + 1$ and not $n$ because one 0 that was replaced by $B$ did not find a matched 0 to be replaced by 1. Now, the 1 are erased. Thus the tape contains now $m - n - 1$ 0. Observe that it does not contain $m - n$ 0 as an “extra” 0 was erased. Then, one additional 0 is added.

**Second stopping condition:** It may be that upon return to the left, in order to find the rightmost $B$ and then moving one step right to find the leftmost 0, the head detects a 1. This means that all the 0 to the left of 1 were changed into $B$. It also implies that $n \geq m$ hence in this case we empty the tape (fill it with blanc).

The transitions:

In this case, the machine erases all the $B$. The tape contains now 0

We assume that in the initial state the head points to the leftmost 0 that was not changed into a blanc. The initial state is $s$

$$(s, 0) \rightarrow (q_1, B, R)$$

The state $q_1$ loops to the right searching for the 0 immediately to the right of the rightmost 1.

$$(q_1, 0) \rightarrow (q_1, 0, R)$$

$$(q_1, 1) \rightarrow (q_2, 1, R)$$

The state $q_2$ starts at the leftmost 1 in the string of 1, and loops to the right looking for the rightmost 1.

$$(q_2, 1) \rightarrow (q_2, 1, R)$$
\((q_2, 0) \mapsto (q_3, 1, L)\)

After finding the first 0 (immediately to the right of the 1 chain) this 0 is changed to 1. The state \(q_3\) is a “getting back to the leftmost 0” state.

\((q_3, 0) \mapsto (q_3, 0, L)\)

\((q_3, 1) \mapsto (q_3, 1, L)\)

\((q_3, \mathcal{B}) \mapsto (s, \mathcal{B}, R)\)

In state \(q_3\) we reach the rightmost \(\mathcal{B}\) and move one step right. A loop has closed so we move to \(s\).

**Dealing with the first stopping condition:**
\((q_2, \mathcal{B}) \mapsto (q_4, \mathcal{B}, L)\)

\(q_4\) is a state that deletes all 1 and adds one 0 to the tape.

\((q_4, 1) \mapsto (q_4, \mathcal{B}, L)\)

\((q_4, 0) \mapsto (q_4, \mathcal{B}, L)\)

\((q_4, \mathcal{B}) \mapsto (h, 0, R)\)

The last transition rule changes a \(\mathcal{B}\) to 0 to add 1 0 to the \(m - n - 1\) existing 0. The state \(h\) is a stopping condition.

**Dealing with the second stopping condition:** In case in state \(s\) we see a 1, we need to erase all the tape. This is done in state \(q_5\).

\((s, 1) \mapsto (q_5, \mathcal{B}, R)\)

\((q_5, 0) \mapsto (q_5, \mathcal{B}, R)\)

\((q_5, 1) \mapsto (q_5, \mathcal{B}, R)\)

\((q_5, \mathcal{B}) \mapsto (h, \mathcal{B}, R)\)

The running time here is \(O(n^2)\) as well.

3. **Question 3:**
(a) Explain how to simulate a Turing machine with \( c > 2 \) strings \((c\ \text{a constant})\) by a Turing machine of 1 string. How many states does the new single string Turing machine have (as a function of \(|\Sigma|, |Q|\) and \(c\))? How large is the new alphabet?

(b) Say that we simulate \( n \) moves of the \( c \) strings Turing machine, by the single string Turing machine. How many moves by the single string Turing machine are required for that?

**Answer:** As in class, we construct a new alphabet. Let \( \Gamma \) be the original alphabet. A new letter in our alphabet corresponds to \( c \) pairs of the form \(<0,x>\) or \(<1,x>\) with \( x \in \Sigma \). If the \( j \) letter of the tape is \( C \), the \( c \) pairs give the content and head position of the \( j \) letters in all \( c \) tapes. If the \( i \) pair in \( C \) is \(<0,x>\) it means that the \( i \) tape has \( x \) in cell number \( j \), and that the head is not located at \( j \) (for the \( i \) tape). A \(<1,x>\) indicates that on tape \( i \) the head points to cell \( j \).

The size of the new alphabet is \( 2^c \cdot |\Gamma|^c \). Because \( c \) and \( |\Gamma| \) are constants, this number is a constant as well.

A simulation of a move has to scan to the right, read the heads of the \( c \) tapes and then return left and scan right again doing the appropriate changes. It requires \( f(n) \) steps with \( f(n) \) the maximum number of operations the Turing machine performs on an input of size \( n \). Hence simulating \( f(n) \) steps requires \( O((f(n))^2) \).

4. **Question 4:** Assume we add a multiplication operation to the RAM requiring one time unit. Are the Turing machine and RAM polynomially equivalent in that case?

**Answer:** Let \( n \) be the size of the input. Let \( f(n) \) be the bound on the number of operations done by the RAM on input of size \( n \) in the worst case.

The claim shown in class is the following. Let \( g(n,t) \) be the number of used registers, plus the largest value of a register after \( t \) \((t \leq f(n)) \) steps. Then, the Turing machine requires \( O((n + g(n))^3) \) basic steps to simulate \( f(n) \) steps by the RAM.

We showed in class that in case the Turing machine does not have multiplication, after \( t \) steps the number of used registers and the largest number in any register is at most \( O(t + n) \). Since \( t \leq f(n) \), this number is bounded by \( O(f(n) + n) \) The time required to simulate the RAM by the Turing machine is thus \( O((n + f(n))^3) \).

However, if we have multiplication, then each multiplication can double the maximum number of a register. Say that \( x = 2^k \) (and requires \( k \) bits to be represented) and \( y = 2^k \) as well. Then \( x \cdot y \) requires \( 2 \cdot k \) bits to be represented.

This means that after \( t \leq f(n) \) steps the size of the largest number may be \( 2^t \) and \( 2^{f(n)} \) at the end. Hence, the TM requires exponential time to simulate the RAM (around \((2^{f(n)})^3 = 2^{3f(n)}\)).

5. **Question 5:** Give a description of a short certificate and a non-deterministic algorithm for the following problems
(a) **Input:** An undirected graph $G(V, E)$ and a number $\ell$

**Question:** Does $G$ admit a spanning tree with at most $\ell$ leaves? (Remark: leaves are degree 1 vertices).

**Answer:** The witness is a subset $E'$ of the edges so that the graph $G'(V, E')$ is a tree that has at most $\ell$ leaves.  
Given $E'$ we need to check in polynomial time that $G'(V, E')$ is a tree (checking degree 1 vertices is done by computing all degrees and requires only $O(|E|)$ time).  
For the algorithm we use few simplifying claims.

**Claim 1:** A connected graph $H(E, V)$ can not have less than $n - 1$ edges.

**Proof:** Indeed, let $H'$ be a spanning tree of a connected graph $H$. We now show (as some of you may remember) that $H'$ has $|V| - 1$ edges. We use the following easy fact (perhaps you should try to prove it): every tree $H'$ has at least two vertices of degree 1. Thus, iteratively remove a leaf and its only edge from $H'$.  
Clearly, at the end a single vertex remain. And we have removed (other then the last vertex) the same amount of vertices and edge. Thus the tree has one more vertex than edge. Hence $|V| = |E'| - 1$. Thus, $|E| \geq |E'| = n - 1$ and the claim has been proved.

**Claim 2:** The graph $G'(V, E')$ is a tree (namely, connected and without cycles) if and only if $|E'| = n - 1$ and $G'$ is connected.

**Proof:** For the sake of deriving a contradiction, say that $G'(V, E')$ has a cycle (the fact that it is connected is a given). Choosing iteratively a cycle in $G'$, and removing an edge $e$ on the cycles, we get $G''(V, E'')$, $|E''| < |E'| = n - 1$ and $G''(V, E'')$ connected. However, this can not be as $|E''| \leq n - 2$ contradicting Claim 1. This gives the proof.

Checking if $G'$ is a tree is thus equivalent to checking it is connected and counting that $|E'| = n - 1$. Checking if $G'$ is connected is done by computing all distances of $V \setminus \{s\}$ from some $s$. The graph $G'$ is connected if all distances are finite. You should know the procedure to compute (unweighted) distances, namely, the BFS algorithm (those who did not learn this algorithm should go over it by themselves). The BFS algorithm runs in $O(|E| + |V|)$ time. Counting the edges is done in $O(|E|)$ time. So in total, $O(|E| + |V|)$ namely a linear algorithm.

(b) **Input:** An undirected graph $G(V, E)$ with a weight $w(e) > 0$ over the edges and a number $\ell$

**Question:** Does $G$ admit a spanning tree of cost $\ell$ or less?

**Answer:** This problem belongs to $P$ (is polynomially solvable) and so certainly belongs to $NP$. In fact we can use an empty witness (we do not need a witness) to solve it.
(c) **Input:** A set $S = x_1, x_2, \ldots, x_n$ of integers so that $\sum_i x_i$ is an even number

**Question:** Can $S$ be decompose into the union of two disjoint subsets $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \emptyset$, so that:

$$\sum_{x_i \in S_1} x_i = \sum_{x_j \in S_2} x_j?$$

**Answer:** A witness is the partition of $S$ into $S_1$ and $S_2$. Remember, the size of the input here is $n \cdot \log(M)$ with $M$ the maximum number in the input. The size of the input is **NOT** $n \cdot M$.

We have to show that $n$ numbers can be added in $\text{poly}(\log M, n)$ time. First, observe that the addition of two numbers requires $O(\log M)$ This is clear because of the “school addition method” (you add two digits and have a carry, etc).

Thus, the sum of $S_1$ and $S_2$ can be computed in $O(n \log M)$ time each. Thus, In total the time to verify that the sum of $S_1$ numbers equal to the sum of $S_2$ numbers is linear in the input.