1. Any language in $SPACE(f(n))$ as defined using the two-tape read-only model can be simulated with a single tape model using at most $O(n)$ space. Similarly, any language in $SPACE(f(n))$ as defined using a single tape model can be simulated with a two-tape read-only model with an improvement of a most $O(n)$ space. Thus, the complexity classes are equivalent where $f(n) \geq n$.

2. The winning strategy for X is to move to the top-right position. O can then move to block only either the top-centre or centre-right position. If O moves to the top-centre, X moves to the centre-right. If O move to the centre-right, X moves to the top-centre.

3. Player I has a winning strategy as follows:

   - Player I begins at node 1.
   - Player 2 chooses node 2.
   - Player I chooses node 4. Node 3 has only one outgoing edge which connects to node 6. As node 6 has no outgoing edges, this path would guarantee a win for Player II.
   - Player II chooses node 5.
   - Player I chooses node 6. As no unchosen nodes remain, Player I wins.

4. Let L_i be a language decided by PSPACE turing machine M_i. We define languages $L_\cup = L_i \cup L_j$, $\bar{L_i} = \{w \mid w \notin L_i\}$, $L_i^* = \{x_1 x_2 \dots x_k \mid k \geq 0$ and each $x_i \in L_i\}$.

   We define $M_\cup$, $\bar{M_i}$ and M_i^* as follows:

   $M_\cup =$ "On input w

   1. Run $M_i\langle w\rangle$. If M_i accepts, *accept*.
   2. Run $M_j\langle w\rangle$. If M_j accepts, *accept*.
   3. If neither $M_i\langle w\rangle$, $M_j\langle w\rangle$ accepted, *reject*."

   $\bar{M_i} =$ "On input w

   1. Run $M_i\langle w\rangle$. If M_i accepts, *reject*, else *accept*."

   $M_i^* =$ "On input w

   1. If $w = \epsilon$, *accept*.
   2. For each m, where $1 \leq m \leq n$, $n = |w|$.
      3. Split w into m pieces, such that $w = w_1 w_2 \dots w_k$.
      4. For all i, $1 \leq i \leq m$, run $M_i\langle w_i\rangle$. If M_i rejects, go to step 2.

5. M_i has accepted for all i, *accept*.
6. M_i has rejected for all m, *reject*."

5. Construct a TM M to decide A_{DFA} When M receives input $\langle A, w \rangle$, a DFA and a string, M simulates A on w by keeping track of A's current state and its current head locations, and updating them appropriately. The space required to carry out this simulation is $O(\log n)$ because M can record each of these values by storing a pointer into its input.

6. Construct a language L, such that L is PSPACE-hard. As L is PSPACE-hard, TQBF $\leq_p L$. We know SAT $\leq_p TQBF$, which gives us SAT $\leq_p L$. As SAT is NP-complete, L is NP-hard, and the proof is complete.

7. Let A_1 and A_2 be languages that are decided by NL-machines N_1 and N_2. Construct three Turing machines: $N_\cup$ deciding $A_1 \cup A_2$; $N_\circ$ deciding $A_1 \circ A_2$; and N_* deciding A_1^*. Each of these machines operates as follows.

   Machine $N_\cup$ nondeterministically branches to simulate N_1 or to simulate N_2. In either case, $N_\cup$ accepts if the simulated machine accepts.

   Machine $N_\circ$ nondeterministically selects a position on the input to divide it into two substrings. Only a pointer to that position is stored on the work tape - insufficient space is available to store the substrings themselves. Then $N_\circ$ simulates N_1 on the first substring, branching nondeterministically to simulate N_1's nondeterminism. On any branch that reaches N_1's accept state, $N_\circ$ simulates N_2 on the second substring. On any branch that reaches N_2's accept state, $N_\circ$ accepts.

   Machine N_* has a more complex algorithm, so we describe its stages.

   $N_* =$ "On input w:

   1. Initialise two input position pointers p_1 and p_2 to 0, the position immediately preceding the first input symbol.
   2. *Accept* if no input symbols occur after p_2.
   3. Move p_2 forward to a nondeterministically selected position.
   4. Simulate N_1 on the substring of w from the position following p_1 to the position at p_2, branching nondeterministically to simulate N_1's nondeterminism.
   5. If this branch of the simulation reaches N_1's accept state, copy p_2 to p_1 and go to stage 2. If N_1 rejects on this branch, *reject*."

8. We let $s_1, s_2, \ldots, s_k$ be the starting position, and $s'_1, s'_2, \ldots, s'_k$ the position after the move is made. We define NIM_s to be the result of performing an XOR operation on each column in $s_1, s_2, \ldots, s_k$, such that $\text{NIM}_s = s_1 \oplus s_2 \oplus \ldots \oplus s_k$. We

note that $s_1, s_2, \ldots, s_k$ is balanced if and only if $\mathrm{NIM}_k = 0$.

Let s_l denote the pile from which sticks are removed, such that $s_i = s'_i$ for all $i \neq l$ and $s_l > s'_l$. We then have

$$\begin{aligned}
\mathrm{NIM}'_s &= 0 \oplus \mathrm{NIM}'_s \\
&= \mathrm{NIM}_s \oplus \mathrm{NIM}_s \oplus \mathrm{NIM}'_s \\
&= \mathrm{NIM}_s \oplus (s_1 \oplus s_2 \oplus \ldots \oplus s_k) \oplus (s'_1 \oplus s'_2 \oplus \ldots \oplus s'_k) \\
&= \mathrm{NIM}_s \oplus (s_1 \oplus s'_1) \oplus \ldots \oplus (s_k \oplus s'_k) \\
&= \mathrm{NIM}_s \oplus 0 \oplus \ldots \oplus (s_l \oplus s'_l) \oplus \ldots \oplus 0 \\
&= \mathrm{NIM}_s \oplus s_l \oplus s'_l
\end{aligned}$$

Assume $\mathrm{NIM}_s \neq 0$. Let d be the most significant non-zero bit of NIM_s. To move to a balanced position, We choose l such that $s_l \neq 0$, and set $s'_l = \mathrm{NIM}_s \oplus s_l$. Note that we chose d in a way that maintains the constraint that $s'_l < s_l$. We then have

$$\begin{aligned}
\mathrm{NIM}'_s &= \mathrm{NIM}_s \oplus s_l \oplus s'_l \\
&= \mathrm{NIM}_s \oplus s_l \oplus (\mathrm{NIM}_s \oplus s_l) \\
&= (\mathrm{NIM}_s \oplus s_l) \oplus (\mathrm{NIM}_s \oplus s_l) \\
&= 0
\end{aligned}$$

Assume $\mathrm{NIM}_s = 0$. To move to an unbalanced position, choose any l. We then have $\mathrm{NIM}'_s = s_l \oplus s'_l$, where $s_l \neq s'_l$.

9. Let a_i, b_i, c_i be the i-th digits of a, b, c and r_i the carry generated for c_i. We then have

$$\begin{aligned}
r_i &= \Big(c_{i-1} + \sum_{j+k=i+1} a_j b_k\Big) \bmod 2 \\
c_i &= \Big(\sum_{m=0}^{i-2} \sum_{j+k=i-m} a_j b_k\Big)/D
\end{aligned}$$

We define machine M, which decides $\mathrm{MULT}\langle a\#b\#c\rangle$.

$M =$ "On input $a\#b\#c$:

1. Initialise $t = 0$.
2. Let $a = a_1 a_2 \ldots a_p$, $b = b_1 b_2 \ldots b_q$, $c = c_1 c_2 \ldots c_{p+q-1}$.
3. For $k = 0$ to $k = p + q - 1$
   4. For $j = max(1, k - p + 1)$ to $min(k, q)$.
      5. Let $i = k - j + 1$.
      6. Update $t = t + (a_i b_j)$.

7. If $c_k = t \bmod 2$, continue, else *reject*.
8. Update $t = \lfloor t/2 \rfloor$.

9. If $c_{p+q} = t \bmod 2$, *accept*, else *reject*.

As M stores i, j, k only as pointers to the input tape, they require only $O(\log n)$ of worktape space. We note that the sum and carry can never exceed values larger than $O(\log n)$ bits, meaning that t requires no more than $O(\log n)$ worktape space, and completing the proof that MULT $\in L$.