Exercise 1: Let $A$ be an algorithm of space complexity $s(n)$. Show that there is an algorithm $A'$ such that

1. $L(A) = L(A')$
2. $A'$ has space complexity $s'(n) = O(s(n) + \log n)$
3. $A'$ does not scan the input tape beyond the boundaries of the input

Proof: We proceed in two steps.

1) We prove that on input $x$, there is an algorithm $A_x$ such that

- $L(A_x) = L(A)$
- $A_x$ does not scan the input tape beyond location $2O(s(n) + \log n)$ from the input

This proof is analogous to the one that we did in class to show that a poly-space bounded (N)TM is equivalent to one that has running time $t(n) \leq Cq(n)$ with $q(n) = O(s(n))$ (where $s(n)$ is a polynomial space bound).

We showed $q(n) = 2s(n) + \delta$, where $\delta = |\Gamma| + |Q|$

In our case:

- $q(n) = 2s(n) + \delta = 2s(n)$

- We have also the position on the input tape that contributes to the configuration:

$$2O(s(n)) = 2\log_2 2O(s(n)) = 2O(s(n) + \log n)$$

- Different configurations at most

Note: A TM with running time $t(n) \leq O(s(n) + \log n)$ can scan at most $2O(s(n) + \log n)$ cells of the input tape.
2) We modify the algorithm $A$ in such a way that it does not move beyond the input.

The resulting algorithm $A'$ works as follows:
- Whenever $A$ would move right past the end of the input, $A'$ instead:
  - does not move past the end of the input, but maintains a counter on the work tape.
  - whenever $A$ moves right, the counter is incremented; left, decremented

In this way, $A'$ can keep track of the position of the input head of $A$.

Whenever $A$ moves back again over the input symbol, $A'$ does not update the counter (leaving it to 0).

$A'$ operates similarly whenever $A$ moves left past the beginning of the input.

How much space does the counter use?
Since $A$ does not scan the input tape beyond $N(n) = 2^{O(n + \log n)}$ the counter takes $\log_2 N(n) = O(n(m) + \log n)$

Hence, the total space used by $A'$ is:
$N(n) + O(N(n) \log 2) = O(N(n) + \log n)$
Exercise 2: Let $A$ be an algorithm of space complexity $S$. Show that there is an algorithm $A'$ such that

- $A'$ computes the same function as $A$, i.e., $A'(w) = A(w) \forall w \in \{0,1\}^*$
- $A'$ has space complexity $S'(n) = S(n) + O(\log l(n))$

where $l(n) = \max_{w \in \{0,1\}^n}|A(w)|$ is the size of the maximum output for input $x$ of length $n$

- $A'$ never overwrites on the same location of its output tape

Proof:

$A'$ proceeds in successive iterations, each time simulating the whole computation of $A$:

In the $i$-th iteration, $A'$ outputs the $i$-th bit of $A(x)$

When simulating $A$, in its $i$-th iteration, $A'$ proceeds as follows:

- it does not directly (re)write on the output tape
- instead, it maintains on the work tape:
  - the counter $i$ of the next output bit that will be written
  - the counter $c$ of the bit that $A$ is currently writing
  - the value of the bit written by $A$ in position $i$
- when $A$ would write an output bit, $A'$ operates depending on the values of $i$ and $c$:
  - if $i \neq c$, then $A'$ does not output anything
  - if $i = c$, then $A'$ stores the written bit on its worktape

At the end of its simulation, $A'$ outputs the stored bit to the $i$-th position of the output tape
How much space \( S(n) \) does \( A \) use on the working tape for inputs of length \( n \)?

- \( S(n) \) cells, since it performs the computation of \( A \)
- The space for the counters \( i, j, \) and \( c \)
- \( c, e \) and so here to count positions on the output tape, and hence will use \( \log_2 l(n) \) bits each.

We get that \( S'(n) = S(n) + O(\log_2 l(n)) \)
Exercise 1: Consider the boolean expression

\[ E = (x_3 \land \neg((x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2))) \lor (\neg x_3 \land (\neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2)) \]

Construct a boolean circuit that computes the value of \( E \), given inputs for \( x_1, x_2, x_3 \).

What is the size of this circuit? 6
What is the depth? 4

How does the size compare to the length of \( E \)?
Exercise: Reduction from Reachability to Circuit Value

Reachability: given a directed graph \( G = (V, E) \) with \( V = \{1, \ldots, n\} \) and \( E \subseteq V \times V \), is there a path from node 1 to node \( n \) in \( G \)?

Circuit Value: given a boolean circuit \( C \) without input variables, is the output of \( C \) equal to \( T \)?

We show a logspace reduction of Reachability to Circuit Value, i.e., we show how to construct in logspace from a directed graph \( G \) a circuit \( R(G) \) such that:

1. \( i \) is reachable from \( n \) in \( G \) iff
   
   - the value of \( R(G) \) is \( T \).

Notice that the key point is to compute \( R(G) \) in logspace.

In \( R(G) \), we use gates of two forms:

1. \( g_{i,j,k} \), with \( 1 \leq i, j \leq n \) and \( 0 \leq k \leq n \)

   Intuitively, \( g_{i,j,k} \) is true iff
   
   \[
   \begin{array}{c}
   i \rightarrow 0 \rightarrow 0 \rightarrow \cdots \rightarrow 0 \rightarrow j \\
   \forall \ell \leq k
   \end{array}
   
   \]

   i.e., there is a path from \( i \) to \( j \) not using any intermediate node bigger than \( k \).

2. \( h_{i,j,k} \), with \( 1 \leq i, j, k \leq n \) iff

   \[
   \begin{array}{c}
   i \rightarrow 0 \rightarrow 0 \rightarrow k \rightarrow \cdots \rightarrow j \\
   \forall \ell \leq k
   \end{array}
   
   \]

   i.e., there is a path from \( i \) to \( j \) not using any intermediate node bigger than \( k \), but \( k \) is an intermediate node.
We describe now the gates and how they are connected.

- For $k = 0$, all $g_{ij0}$ gates are constant gates

$$g_{ij0} = T \iff i \neq j \lor i \rightarrow j \in G_1$$

This is how $G$ is reflected in $R(G)$ (note that there are no $g_{ij0}$ gates)

- For $k = 1, 2, \ldots, n$
  - $g_{ijk}$ is an AND gate with predecessors
    
    $g_{i,k, k-1}$ and $g_{k,j, k-1}$

  - $g_{ijk}$ is an OR gate with predecessors

  $g_{ijk}, k-1$ and high

The output gate is $g_{i,n,n}$

$R(G)$ can be computed from $G$ in logarithmic space.

Note that the circuit $R(G)$ is legitimate, since it contains no cycles: we can reverse the gates $1, 2, \ldots, 2n^3 + n^2$ in non-decreasing order of the third index, and with high preceding $g_{ijk}$

We have to show that the value of the output gate of $R(G)$ is $T$ iff there is a path from $i$ to $n$ in $G$.

We prove by induction on $k$ that the values of the gates correspond to the informal meaning we gave them:

- For $k = 0$: this holds.
- If it is true up to $k-1$, the definitions of $g_{ijk}$ and high guarantee that it is true also for $k$. 

(\text{proof details omitted})
Exercise: A boolean function \( f \) is said to be monotone if it has the following property: if one of the values changes from 0 to 1, then the value of \( f \) does not change from 1 to 0.

We show that \( f \) is monotone iff it can be expressed by a circuit with only AND and OR gates.

\[ \leq \]

Consider a circuit \( C \) with only AND and OR gates expressing \( f \).

We show by induction on the depth of a node \( N \):

- If the value of an input \( x \) changes from 0 to 1, then the value of \( N \) does not change from 1 to 0.

Base: depth \((N) = 0\), then \( N \) is either an input or a constant node.

- if it is a constant, its value does not change
- if it is an input different from \( x \),
- if it is input \( x \), its value changes from 0 to 1 (and not from 1 to 0)

Induction: suppose that for all nodes of level \( k \), the value does not change from 1 to 0.

Consider a node \( N \) at level \( k+1 \). We show that the value of \( N \) does not change from 1 to 0.

Case 1: \( N \) is an AND node

\[ y_1 \land y_2 \]

\[ y_{\text{AND}} \]

Case 2: \( N \) is an OR node

\[ y_1 \lor y_2 \]

\[ y_{\text{OR}} \]
We need to consider various subcases, corresponding to the changes of \( y_1, y_2 \) from 0 to 1.

<table>
<thead>
<tr>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y'_1 )</th>
<th>( y'_2 )</th>
<th>( \text{AND} )</th>
<th>( \text{OR} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

We see that both for an AND node and for an OR node, the output value never changes from 1 to 0.

"\( \Rightarrow \)" We show by induction on \( \alpha \) that every monotone boolean function \( f(x_1, \ldots, x_\alpha) \) of \( \alpha \) variables can be represented by a circuit with AND and OR gates only.

**Base case:** 0 arguments. \( f \) is constant, and hence monotone.

**Inductive case:** Assume the claim holds for \( \alpha \).

We show it holds for a function \( f(x_1, \ldots, x_{\alpha+1}) \).

We exploit the fact that

\[
f(x_1, \ldots, x_\alpha, x_{\alpha+1}) = (x_{\alpha+1} \land f(x_1, \ldots, x_\alpha, 1)) \lor (\overline{x_{\alpha+1}} \land f(x_1, \ldots, x_\alpha, 0))
\]
Hence, we can construct a circuit $C_f$ computing $f(x_1, \ldots, x_n, x_{n+1})$ as follows:

![Diagram of the circuit $C_f$]

Observe that, since $f(x_1, \ldots, x_{n+1})$ is monotone, we have that also $f(x_1, \ldots, x_n, 0)$ and $f(x_1, \ldots, x_n, 1)$ are monotone.

Hence, since $f(x_1, \ldots, x_n, 0)$ and $f(x_1, \ldots, x_n, 1)$ are $n$-variable monotone functions, by inductive hypothesis, they can be represented by circuits with $A\&D$ and $O\&R$ gates only.

Hence, it suffices to show that we can get rid of the only remaining $N\&O$ gate.

Consider the following circuit $C'_f$ in which we have eliminated the $N\&O$ gate.
Let us consider the possible values of $f$ in $C_f$ and $C'_f$. It depends on the values of $k_m$, $v_0$, $v_1$.

<table>
<thead>
<tr>
<th>$k_m$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$C_f$</th>
<th>$C'_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>71</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>81</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that $C_f = C'_f$, except for case (7).

However, since $f(k_1, \ldots, k_{n+1})$ is monotone, cases (3) and (7) cannot occur, since they would mean that $f(k_1, \ldots, k_m, k_{n+1})$ changes from $v_0 = 1$ for $k_{n+1} = 0$ to $v_0 = 0$ for $k_{n+1} = 1$.

Hence, $C'_f$ is the correct circuit consisting of AND and OR gates only and computing $f(k_1, \ldots, k_{n+1})$. 