1. (a) M_f compute f in time $O(n+f(n))$, and space $O(f(n))$, M_g compute g in time $O(n+g(n))$, and space $O(g(n))$;

   - M_1 will compute $f(g(n))$:
     i. M_1 will simulate M_g over the input string, and will write $g(n)$ on string k_1.
     ii. M_1 will simulate M_f and will take k_1 as its input string. M_1 will write the result on the output string.

     Time complexity: $O(n+g(n))+O(g(n)+f(g(n)))$. As $f(n) \geq n$, time complexity is $O(n + f(g(n)))$.
     Space complexity: $O(g(n)) + O(f(g(n))) = O(f(g(n)))$.
   - M_2 will compute $f + g$:
     i. M_2 will simulate M_f and write its output on the output string.
     ii. M_2 will simulate M_g and write its output concatenated to the first stage output.

     Time complexity: $O(n + f(n)) + O(n + g(n)) = O(n + f(n) + g(n))$.
     Space complexity: $O(f(n)) + O(g(n)) = O(f(n) + g(n))$.
   - M_3 will compute $f \cdot g$:
     i. M_3 will simulate M_f over the input string, and will write $f(n)$ on string k_1.
     ii. M_3 will simulate M_g over the input string, and will write $g(n)$ on string k_2.
     iii. For each 1 on k_1, M_3 will copy all k_2 to the output string.

     Time complexity: $O(n + f(n)) + O(n + g(n)) + O(f \cdot g(n)) = O(n + f \cdot g(n))$.
     Space complexity: $O(f(n)) + O(g(n)) + O(f \cdot g(n)) = O(f \cdot g(n))$.
     (We assume $f(n) > 0, g(n) > 0$)
   - M_4 will compute 2^g:
     i. M_4 will simulate M_g over the input string, and will write $g(n)$ on string k_1.
     ii. M_4 will write 1 on string k_2.
     iii. M_4 will use strings k_2 and k_3 to compute 2^g, for each 1 on k_1, M_4 will copy twice the 1's from k_2 to k_3 and vice versa.

     Time complexity: $O(n + g(n) + 2^{g(n)}) = O(n + 2^{g(n)})$.
     Space complexity: $O(g(n) + 2^{g(n)}) = O(2^{g(n)})$.

   (b) First we will see that $\log n$ is proper complexity function:

   M will count the number of 1's on the input string, and compute n in binary representation (exact method was needed for full grade) over string k_1. M will copy each character $(0, 1)$ from k_1 to 1 over the output string.
   Time complexity: $O(n + \log n)$.
   Space complexity: $O(\log n)$.
   $\log n$ is proper complexity function. $\log^2 n = \log n \cdot \log n$, and as shown in 1a if f,g are proper complexity functions than $f \cdot g$ is proper complexity function $\Rightarrow \log^2 n$ is proper complexity function.
   $n^2 = n \cdot n$, n is proper complexity function (why?) $\Rightarrow n^2$ is proper complexity function.
   As shown in 1a if g is proper complexity functions than, 2^g is proper complexity function $\Rightarrow 2^n$ is proper complexity function.
   Express $\sqrt{n} = 2^{0.5 \cdot \log n}$, $0.5 \cdot \log n$ is proper complexity function (why?) $\Rightarrow \sqrt{n}$ is proper complexity function.

2. Let $p(n)$ be a polynomial with degree m, $P(n) = \Theta(n^m)$.

   (a) $C = \{n^k | k > 0\}$:
   Left: $p(f(n)) = p(n^k) = O(n^{km})$, $n^{km} \in C \Rightarrow$ closed.
   Right: $f(p(n)) = p(n)^k \leq (cn^m)^k = O(n^{km})$, $n^{km} \in C \Rightarrow$ closed.

   (b) $C = \{k \cdot n | k > 0\}$:
   Let $p(n) = n^2$.
   Left: $p(f(n)) = (kn)^2 = \Omega(n^2)$, $n^2 \notin C \Rightarrow$ not closed.
   Right: $f(p(n)) = kn^2$, $n^2 \notin C \Rightarrow$ not closed.

   (c) $C = \{k^n | k > 0\}$:
   Left: $p(f(n)) = p(k^n) = O((k^n)^m)$, for $k' = k^m$, $k'^n \in C \Rightarrow$ closed.
   Right: Let $p(n) = n^2$. $f(p(n)) = f(n^2) = k^{n^2}$, $k^{n^2} \notin C \Rightarrow$ not closed.

   (d) $C = \{2^{n^k} | k > 0\}$:
   Left: $p(f(n)) = O((2^{n^k})^m) = O(2^{mn^k}) = O(2^{n^{k+1}})$, for $k' = k + 1$, $2^{n^{k'}} \in C \Rightarrow$ closed.
   Right: $f(p(n)) \leq f(cn^m) = 2^{c^k n^{km}} = O(2^{n^{km+1}}) \Rightarrow$ closed.

   (e) $C = \{log^k n | k > 0\}$:
   Left: $p(f(n)) = p(\log^k n) = O((\log^k n)^m) = O(\log^{km} n)$, for $k' = km$, $\log^{k'} n \in C \Rightarrow$ closed.
   Right: $f(p(n)) \leq f(cn^m) = \log^k cn^m = (\log c + m \log n)^k = O(\log^k n) \Rightarrow$ closed.

   (f) $C = \{k \cdot \log n | k > 0\}$:
   Left: Let $p(n) = n^2$. $p(f(n)) = (k \log n)^2 = \Omega(\log^2 n) \notin C \Rightarrow$ not closed.
   Right: $f(p(n)) \leq f(cn^m) = k \cdot \log cn^m = k(\log c + m \log n) \leq k' \log n$ for $k' = k(|\log c| + m) \Rightarrow$ closed.

   Note: there is need to prove that $f(p(n)) \leq f(cn^m)$ whenever this was used.

3. (a) Consider the following algorithm:

Algorithm 1 DECIDE-PAL(x)

Require: $x \in \{0,1\}^n$.
Ensure: Returns *true* if $x \in PAL$ and $false$ otherwise.

```
for i ← 1 to n do
    if x[i] ≠ x[n + 1 − i] then
        return false
return true
```

Space Analysis:
$O(\log n)$ for $i \Rightarrow O(\log n)$.

Correctness:
($\Leftrightarrow$) $x \in PAL$ if and only if $\forall i\ x[i] = x[n - i + 1]$; DECIDE-PAL returns *true* if and only if $x[i] = x[n - i + 1]$ for all i.

Algorithm 2 DECIDE-NOT-PAL(x)

Require: $x \in \{0,1\}^n$.
Ensure: Returns $false$ if $x \in PAL$ and $true$ otherwise.

guess $1 \leq i \leq n$ {actually $i \leq \lfloor \frac{n}{2} \rfloor$ is enough}
if $x[i] \neq x[n-i+1]$ **then**
    return $true$
else
    return $false$

(b) Consider the following algorithm:

Space Analysis:
$O(\log n)$ for $i \Rightarrow O(\log n)$.

Time Analysis:
One sweep forward to find n, one sweep backward to gather $x[i], x[n-i+1] \Rightarrow O(n)$. As we saw in Exercise 1, incrementing a counter from 0 to n only costs $O(n)$.

Correctness:
($\Leftrightarrow$) $x \notin PAL$ if and only if $\exists i$ such that $x[i] \neq x[n-i+1]$; DECIDE-NOT-PAL returns $true$ if and only if for some execution path, i.e., a guess for i, $x[i] \neq x[n-i+1]$.

Note: $PAL \in L \subseteq NL \Rightarrow \overline{PAL} \in$ co-$NL = NL$ is a simple result of *Immerman-Szelepcsényi theorem*, but we needed the algorithm presented to enforce linear running time.

4. Consider the following algorithm:

Algorithm 3 ADD-BINARY-NUMBERS(x, y)

Require: Two integers $x, y < 2^n$ in binary representation, least-to-most significant bit order.
Ensure: Returns $x + y$ in binary representation, least-to-most significant bit order.

$carry \leftarrow 0$ {can be embodied into the internal state}
for $i \leftarrow 1$ to n **do**
    $output[i] \leftarrow x[i] \oplus y[i] \oplus carry$ { assume 0 if $x[i]$ or $y[i]$ don't exist }
    $carry \leftarrow maj(x[i], y[i], carry)$
$output[n+1] \leftarrow carry$

Space Analysis:
$O(\log n)$ for i, $O(1)$ for $carry \Rightarrow O(\log n)$.

Correctness:
Trivial - we add bit by bit using a full adder.

Note: The algorithm shown uses least-to-most sigfinicant bit order. If we require most-to-least order, this can be achieved as a composition of this algorithm with a rather trivial $O(\log n)$-space REVERSE algorithm, in $O(\log n)$-space (as done with composing log-space reductions).

5. Solution draft: We will use a binary encryption for each letter of Σ. To read and to write those extended letters we will need to expand our set of machine states and the transition

function. For each old state we will now need $|\Sigma|$ new stats to read the encrypted letter in this state, and $|\Sigma|$ new stats to write a new letter in its place (we will actually need double of that amount since we also need to remember whether we should move left or right). We will also need additional $2\log(|\Sigma|)$ new states to move left and right. The size of each 'letter' is now $\log(|\Sigma|)$ so the time complexity should not exceed $4\log(|\Sigma|)$ times the old time complexity, since for a standard TM operation we will need to read the old 'letter' then write a new one and then move at most $2\log(|\Sigma|)$ spaces (a move left).