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. We describe a k'-string $O(f(n))$ time bounded block respecting TM M' with $k' = 2k + O(1) = O(k)$ equivalent to M. We must assume $f(n) \geq n$ is a proper complexity function as described in Question 1.

   First, M' calculates the unary representation of $\sqrt{f(n)}$ on the clock string. This can be done in $O(\sqrt{f(n)})$ space and $O(n + \sqrt{f(n)}) = O(f(n))$ time (prove! use results from Question 1 carefully since $\sqrt{n} \not\geq n$). We use a constant number of strings such that on every string we don't cross the $\sqrt{f(n)}$ boundary.

   M' simulates M, keeping the even and odd blocks of each string s on the strings s_0, s_1, respectively. Every step is accompanied with a to-and-fro movement on the clock string, so we *know* when it is safe to cross. Whenever M moves from an odd block to an even block or vice versa, M' uses the respective string. When M' needs to cross a boundary, it waits until due time and then crosses safely[1].

   Correctness: M' is equivalent to M since it simulates it, perhaps adding null steps; M' respects blocks, since it only crosses block boundaries on steps which are integer multiples of $\sqrt{f(n)}$.

[1] M' looks left, right and left again before crossing.

Time complexity: M' only waits when M passes from one odd/even block to the adjacent odd/even block. In any case, M needs to stroll along a whole block, and this takes $\geq \sqrt{f(n)}$ steps. M only runs for $f(n)$ steps, so no more than $\sqrt{f(n)}$ stalls may occur. For each stall, M' waits no more than $\sqrt{f(n)}$ steps, so the total cost of stalls is bounded by $f(n)$. Hence the total running time of M' is $O(f(n))$.

4. We build the required circuit inductively. For $n = 1$ naught is to be done, as the parity function is the identity function. For $n = 2$, naming the inputs x_1, x_2

$$XOR(x_1, x_2) \stackrel{def}{=} (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$$

is a circuit of depth $3 = O(1)$ and size $8 = O(1)$ that calculates the parity function (verify!).

For the general circuit ($n > 2$), we rely on the associativity of addition modulo 2 to represent the parity function as a binary tree of XOR gates.

We split the n inputs into two groups of size $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$. For each group we recursively build a parity circuit of depth $3 \log_2 \frac{n}{2} = 3(\log_2 n - 1)$ and size $O(\frac{n}{2})$. Finally, we connect the outputs of the two circuits via a XOR gate.

The depth of the resulting circuit is $3 + 3(\log_2 n - 1) = 3 \log_2 n$;

The size of the resulting circuit is $O(n)$ since it contains exactly $n - 1$ XOR gates (prove!) of size $O(1)$ each[2].

5. We use the subroutine $CVAL(C, x)$ that computes the value of a circuit C given the input x. This can be computed in polynomial space.

Let C be the input circuit, $n = |C|$. C has $k \leq n$ inputs.

Algorithm: Enumerate (e.g., in lexicographic order) all circuits C' of size $|C'| \leq |C|$. If C' doesn't have k inputs, skip it. For each C', we enumerate (e.g., in lexicographic order) all inputs $x \in \{0,1\}^k$. If for any x $CVAL(C, x) \neq CVAL(C', x)$, skip C' altogether; otherwise return "true". If all C' were skipped and we've reached C itself, return "false".

Correctness: Obviously, the algorithm finds an equivalent C' if and only if such exists.

Space complexity: Each of C', x and the temporary calculation of $CVAL$ consume polynomial space which can be recycled between iterations. Thus the whole algorithm uses polynomial space.

[2]Notice that the depth bound is calculated using the induction step whereas the size bound is proved directly. Use the method easier for you.