1. Circuit-size Hierarchy:

Let $F_n$ be the set of binary functions on $n$ variables, and $C_n(t(n))$ be the set of binary functions on $n$ variables computable by circuits of size at most $t(n)$.

We use that every $f \in F_n$ can be computed by some $c_f \in C_n(O(2^n/n))$, and therefore $F_n \subseteq C_n(O(2^n/n))$. Furthermore, we use that $|C_n(t(n))| = 2^{O(t(n) \ln t(n))}$.

Set $x(n) = \ln(f(n) \ln f(n))$. Note that $x(n) \leq n$. Set $A = C_n(o(f(n)))$, $B = F_x$ and $C = C_n(O(2^{x/x}))$. We will show that $A \subsetneq B \subseteq C$ which would imply that there exists $g \in F_x \subseteq F_n$ computable by a $(f(n) \ln f(n))$-size circuit, and not computable by a $o(f(n))$-size circuit. $B \subseteq C$ follows immediately from the facts above. For $A \subsetneq B$ we use that $|A| \leq 2^{o(f(n) \ln f(n))} \ll f(n)f(n) = |B|$.

2. Poly-size Circuits:

If NP $\subseteq$ P/poly then Ladner’s theorem provides the desired language. Otherwise, NP $\not\subseteq$ P/poly and hence no NP-hard language is in P/poly. On the other hand, the unary halting problem (in P/1) is not in NP. And we are done.

$\text{TIME}(2^{n \log n}) \not\subseteq P/poly$: We would like to build a machine $M$ such that for all input lengths $n \in \mathbb{N}$, and all circuits $C \in NC$ of size at most $g = n^{\sqrt{\log n}}$ (super-polynomial), there exists $x \in \{0, 1\}^n$ with $M(x) \neq C(x)$. Since $g(n)$ is super-polynomial, eventually $M$ will differ from all polynomial-size circuits.

Let $C = c_1, \ldots, c_m$ be an enumeration of all circuits on $n$ inputs of size at most $g = n^{\sqrt{\log n}}$. By counting, $m \leq 3^g(g + n)^{2g} = 2^{O(g \log g)}$. Let $\alpha_1, \ldots, \alpha_{2^n}$ be all possible values of the input. $M(x)$ is computed as follows:

1. Set $i \leftarrow 1$ and $R \leftarrow C$

2. While $R \neq \emptyset$, repeat:
   i. $M(\alpha_i) \leftarrow \neg \text{Maj}(R)$
   ii. $R \leftarrow \{c \in C \mid c(\alpha_i) \neq M(\alpha_i)\}$
   iii. $i \leftarrow i + 1$

3. For $j \geq i$ set $M(\alpha_j) \leftarrow 0$

We are thus left to show that $M$ runs in time $2^{n \log n}$. It is easy to see that $M$ simulates at most $2m$ circuits, each requiring $n^{\sqrt{\log n}}$ steps, for a total:

$$2^{O(n \sqrt{\log n} \log^{3/2} n)} \ll 2^{O(n \log n)}$$
3. CNF, DNF, and Branching Programs:

Let $k$-DNF formula $\Phi_{\text{DNF}}$ and an $l$-CNF formula $\Phi_{\text{CNF}}$ be given for a boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. It then immediately follows that the following hold:

\[
\bigwedge_{C \in \Phi_{\text{CNF}}} \bigwedge_{T \in \Phi_{\text{DNF}}} \bigvee_{l \in T} C \ni l \quad (\dagger)
\]

\[
\bigwedge_{T \in \Phi_{\text{DNF}}} \bigwedge_{C \in \Phi_{\text{CNF}}} \bigvee_{l \in C} T \ni l \quad (\ddagger)
\]

Where $T$ is a term, $C$ is a clause and $l$ is a literal.

Let $A_{k,l}$ be the set of binary functions on $n$ variables that have a $k$-DNF and an $l$-CNF formulas. For a fixed function $\Phi \in A_{k,l}$ we describe a depth-$k$ branching program (BP) that either evaluates $\Phi$ on an input $x \in \{0, 1\}^n$ or makes a recursive call to a BP evaluator for $\Phi' \in A_{k,l-1}$ on $x' \in \{0, 1\}^{n-k}$, where $x' \subset x$.

Let $T \in \Phi_{\text{DNF}}$ be a DNF term and assume $T = x_1 \ldots x_k$. If $T$ is true in $x$ then halt and output “1”. Otherwise, according to $(\dagger)$ we can remove at least one literal (in $T$) from each clause in $\Phi_{\text{CNF}}$ thus obtaining $\Phi'$, which we evaluate on $x_{k+1}, \ldots, x_n$ recursively.

Similarly, for $\Phi \in A_{k,l}$ we have a depth-$l$ BP that either evaluates $\Phi$ on $x$, or makes a recursive call to a BP evaluator for $\Phi' \in A_{k-1,l}$ on $x' \in \{0, 1\}^{n-l}$, where $x' \subset x$.

Using these two constructions, it is straightforward that $f(k, l) \leq kl + \min\{k, l\}$. (You can also do a little better and get $kl$.)

4. Finding a satisfying assignment when there are many:

(This solution is adapted from Hirsch’98.) For a formula $f$ on $n$ variables $X = \{x_1, \ldots, x_n\}$ we let $f[l_1, \ldots, l_k]$, where $L = \{l_1, \ldots, l_k\}$ are literals in $X$, be the formula obtained by restricting $f$’s inputs correspondingly.

For a $k$-CNF $f$ consider the following algorithm:

1. $i \leftarrow 0$, $\Phi_0 = \{f\}$ and $\Phi_1 = \cdots = \Phi_n = \emptyset$

2. For all $g \in \Phi_i$, do:
   i. Let $(l_1 \lor \cdots \lor l_r)$ be the shortest clause in $g$
   ii. Consider the restrictions $g[l_1], g[l_1, l_2], \ldots, g[l_1, \ldots, l_{r-1}, l_r]$. If any one of them is $\equiv 1$ then halt the algorithm and output a satisfying assignment for $f$. Otherwise, set $\Phi_{i+1} \leftarrow \Phi_{i+1} \cup \{g[l_1], g[l_1, l_2], \ldots, g[l_1, \ldots, l_{r-1}, l_r]\}$

3. $i \leftarrow i + 1$. If $i > d$, where $d$ is a threshold to be specified later, halt the algorithm and output “$f$ has less than $\epsilon 2^n$ satisfying assignments”

4. Go to step 2
Let $T$ be the abstract tree induced by this algorithm, where each node $v$ corresponds to a restriction $v = f[\ldots]$ and a node $u$ is a parent of $v$ if $v$ is a restriction of $u$ created in step 2.ii. of the algorithm. Let $T$ be a subtree and $v$ be a node in it. We define the floor of $v$ with respect to $T$, denoted $\varphi_T(v)$, to be the number of variables in $T$’s root that are restricted in $v$. Furthermore, we refine our notation by $\Phi_i^T := \{v \in T \mid \varphi_T(v) = i\}$, and thus $\Phi_i = \Phi_i^R$ where $R$ is the whole tree.

Let’s make an argument for soundness first. Assume the algorithm has reached to the point when $\Phi$ is complete, but it hasn’t been processed yet (step 2). This implies that there are no satisfying partial assignments of at most $i$ variables and furthermore any potential satisfying assignment must also be satisfying for some $v \in \Phi_i \cup \cdots \cup \Phi_{i+k-1}$. Every $v \in \Phi_{i+j}$, where $0 \leq j < k$, can have at most $2^{n-i-j}$ satisfying assignments. Therefore, at this point of the execution we have certified that $f$ has at most $M_i = \sum_{j=0}^{k-1} |\Phi_{i+j}| \cdot 2^{n-i-j}$ satisfying assignments in total. The threshold $d$ is chosen so that the algorithm stops as soon as $M_d/2^n < \epsilon$. This completes soundness. We now analyze the running time.

We begin by deriving that $|\Phi_i| \leq \lambda_k^i$ where $\lambda_k$ is the unique positive solution of $h_k(x) = 1 - x^{-1} - \cdots - x^{-k}$. Induct on the size of $T$. In the base, $|T| = 1$ and we check that $\Phi_0 = 1 \leq \lambda_k^0 = 1$ and $\Phi_i = 0 < \lambda_k^i = 1$ for $i > 0$. For the step, we let $R$ be the root of the tree and $T_1, \ldots, T_l$, where $0 \leq l \leq k$, be the subtreess of its children:

$$|\Phi_i^R| = \sum_{j=1}^{l} |\Phi_{i-j}^T| \leq \sum_{j=1}^{l} \lambda_k^{i-j} = \lambda_k^i \sum_{j=1}^{l} \lambda_k^j \leq \lambda_k^i \sum_{j=1}^{k} \lambda_k^{-j} = \lambda_k^i \cdot (1 - h_k(\lambda_k)) \leq \lambda_k^i$$

Using this bound we can now derive $d \geq \frac{\log 2/\epsilon}{\log 2/\lambda_k}$ using $M_d/2^n < \epsilon$ and:

$$\sum_{j=0}^{k-1} |\Phi_{d+j}| \cdot 2^{n-d-j} \leq \sum_{j=0}^{k-1} \lambda_k^d \cdot 2^{n-d-j} \leq 2 \cdot \lambda_k^d \cdot 2^{n-d}$$

We have thus far shown that if $f$ has “many” satisfying assignments, then it has a partial satisfying assignment on $\frac{\log 2/\epsilon}{\log 2/\lambda_k} + k - 1$ variables.

Finally, we need to show that the size of the tree is small:

$$|T| = \sum_{i=0}^{i-1} |\Phi_i| + \sum_{j=0}^{k-1} |\Phi_{i+j}| \leq \sum_{i=0}^{i-1} \lambda_k^i + k \cdot \lambda_k^i = \cdots = O \left( k (2/\epsilon)^{\left( \frac{\log \lambda_k}{2-1} \right)^{-1}} \right)$$

The algorithm spends $L$ steps at each tree node, where $L$ is the size of $f$. This concludes the proof that the algorithm runs in polynomial time.

5. Majority:

As seen from Smolensky’s proof that $\oplus_2 \not \in AC^0$, it is the case that $\oplus_3 \not \in AC^0_{\oplus_2}$ (where $AC_{\oplus_2}$ stands for AC with parity gates). Therefore, it would be sufficient to show that $\oplus_3 \leq \text{Maj}$ using a constant depth, polynomial size reduction.

For $x \in \{0,1\}^n$, let $\geq_{1,t}(x)$ be a circuit that determines if $\#_1(x) \geq t$, where $\#_1(x)$ is the number of 1’s in $x$. To implement $\text{GE}_{1,t}(x)$, assume w.l.o.g. $t \leq n/2$ and verify that $\text{GE}_{1,t}(x) := \text{Maj}(1^n-2t \cdot x)$.
works. We can now also implement $\text{EQ}_{1,t}(x)$ which decides whether $x$ has exactly $t$ entries 1, as $\text{EQ}_{1,t}(x) := \text{GE}_{1,t}(x) \land \text{GE}_{0,|x|-t}(x)$.

Then, set $k := \lceil n/3 \rceil$ and define:

$$\oplus_3(x) := \begin{cases} 0, & \text{if } \bigvee_{i=0}^k \text{EQ}_{1,3k}(x) \\ 1, & \text{if } \bigvee_{i=0}^k \text{EQ}_{1,3k+1}(x) \\ 2, & \text{otherwise} \end{cases}$$

This completes the reduction $\oplus_3 \leq \text{Maj}$ (and the problem).

6. A Lower Bound via Communication Complexity:

We assume for contradiction that there exists a 1-tape Turing machine $M$ that solves the PALINDROME language in $o(n^2)$ time. Using $M$ we build a communication protocol for EQ which has worst-case complexity $o(n)$, leading to a contradiction.

Let Alice be given $x \in \{0, 1\}^n$ and Bob be given $y \in \{0, 1\}^n$. (Making sure that $|x| = |y|$ is trivial using two log $n$-bit rounds.)

Alice runs $M$ on input $w_x = x \cdot 0^n \cdot x^R$ and recognizes a location $i_x \in [n + 1, 2n]$ on the tape such that the number of times $M$’s pointer passes through $i_x$ is $o(n)$. Let’s see why such an $i$ always exists. Let $c_i(w)$ be the number of times $M$’s pointer passes through $i$ during computation on $w$. Since $M$ runs in $o(n^2)$, we have that $\sum_{n < i \leq 2n} c_i(w) = o(n^2)$, and thus by averaging there is an $i_x$ for which $c_{i_x}(w) = o(n)$. Bob performs a similar computation with $w_y = y \cdot 0^n \cdot y^R$.

Next, Alice and Bob exchange the indices $i_x$ and $i_y$ in two rounds of log $n$-bits each. If the indices differ, then $x \neq y$ and the protocol is over. Otherwise:

Alice and Bob simulate an imaginary run of $M$ on tape $x \cdot 0^n \cdot y^R$, such that at any point either Alice or Bob is simulating and they alternate whenever $M$’s cursor passes through the agreed upon location $i$. Control is transferred by sending $M$’s state in $O(1)$-bits.

If $x = y$ the simulated computation will go exactly as Alice and Bob expect, in $o(n)$ rounds and bits and they will accept. If $x \neq y$, it must be the case that Alice or Bob halts $M$ pre-maturely, or the state of $M$ at hand-off is not what is expected. We have thus obtained a protocol for EQ which requires $o(n)$-bits communication complexity in the worst case – a contradiction.