1. (a) Note: it is most convenient to think of $\pi\pi'$ as the permutation $k \mapsto \pi'(\pi(k))$ rather than the more conventional $k \mapsto \pi(\pi'(k))$ – the two notions are equivalent by taking inverses; however the second is somewhat more cumbersome notationally for this problem. We start with $m + 1$ levels $\ell_1, \ell_2, \ldots, \ell_{m+1}$ of 5 nodes each. We describe the edges directed from level $i$ to level $i + 1$ based on the $i$-th instruction $(i_j, r_j, t_j)$: connect the outgoing “0” edges from node $k$ to $\tau(k)$ for $k \in \{1, 2, 3, 4, 5\}$, and the outgoing “1” edges from node $k$ to $\tau(k)$ for $k \in \{1, 2, 3, 4, 5\}$. Suppose on input $x \in \{0, 1\}^n$ the instructions yield $\alpha \in S_5$. Then the path in the branching program starting at node $k$ in level 1 and dictated by $x$ leads to node $\alpha(k)$ in level $m + 1$. Since $\pi \neq e$, we can find some $k \in \{1, 2, 3, 4, 5\}$ for which $\pi(k) \neq k$. We designate node $k$ in the first level as the start node, node $\pi(k)$ in level $m + 1$ as the accept node, and node $k$ in level $m + 1$ as the reject node, and we discard the other nodes in level $m + 1$. The result is a width 5 branching program with $m$ levels. For every $x \in A$, the path dictated by $x$ from the start node leads to the accept node (formerly node $\pi(k)$ in level $m + 1$), and for every $x \not\in A$, the path dictated by $x$ from the start node leads to the reject node (formerly node $\alpha(k) = k$ in level $m + 1$), as required.

(b) For every pair of 5-cycles $\pi$ and $\pi'$ we can find an element $\alpha \in S_5$ for which $\alpha\pi\alpha^{-1} = \pi'$. We replace each instruction $(i_j, r_j, t_j)$ with the instruction $(i_j, \alpha r_j \alpha^{-1}, \alpha t_j \alpha^{-1})$.

(c) We replace the last instruction $(i_m, r_m, t_m)$ with the instruction $(i_m, \alpha r_m \alpha^{-1}, \alpha t_m \alpha^{-1})$. The resulting sequence of $m$ instructions yields $e$ on inputs $x \in A$ and $\pi^{-1}$ on inputs $x \not\in A$. Thus the modified sequence of $m$ instructions $\pi^{-1}$-accepts the complement of $A$. Since $\pi$ is a 5-cycle, $\pi^{-1}$ is a 5-cycle, and we can apply the previous part to obtain a sequence of $m$ instructions that $\pi^{-1}$-accept the complement of $A$ as required.

(d) We concatenate the following 4 sequences: (1) a sequence of $m$ instructions that $\sigma$-accepts $A$, obtained using part (b); (2) a sequence of $m'$ instructions that $\tau$-accepts $B$, obtained using part (b); (3) a sequence of $m$ instructions that $\tau^{-1}$-accepts $B$, obtained using part (b); (2) a sequence of $m'$ instructions that $\tau^{-1}$-accepts $B$, obtained using part (b). We claim that this sequence $\sigma\tau\sigma^{-1}\tau^{-1}$-accepts $A \cap B$. If $x \in A \cap B$, then clearly this sequence yields $\sigma\tau\sigma^{-1}\tau^{-1}$. However, if $x \in A - B$ the sequence yields $e\tau e\tau^{-1} = e$, and if $x \in B - A$ then it yields $\sigma\tau\sigma^{-1}e = e$; finally if $x \not\in (A \cup B)$ then it yields $e$. So this sequence of $2(m + m')$ instructions $\sigma\tau\sigma^{-1}\tau^{-1}$-accepts $A \cap B$ as required.

(e) Observe that $(A \cup B) = (\overline{A \cap B})$. We use part (c) to obtain a sequence of $m$ instructions that $\pi$-accepts $\overline{A}$ and a sequence of $m'$ instructions that $\pi'$-accepts $\overline{B}$. Using part (d), we obtain a sequence of $2(m + m')$ instructions that $\sigma\tau\sigma^{-1}\tau^{-1}$-accepts $\overline{A \cap B}$. Finally, we use part (c) one more time to convert it into a sequence of the same length that $\sigma\tau\sigma^{-1}\tau^{-1}$-accepts $A \cup B$. 

3-1
(f) As suggested we prove this by induction on \( d \). If \( d = 0 \), then the circuit is just a single literal \( x_i \), and the sequence \((i, e, \pi)\) clearly \( \pi \)-accepts the language decided by the circuit. Now, for \( d > 0 \), we have 3 cases. If the last gate is \( \neg \), then by induction we have a sequence of \( 4^{d-1} \) instructions that \( \pi \)-accepts \( \neg \), and by part (c) there is a sequence of \( 4^d \) instructions that \( \pi \)-accepts \( A \). If the last gate is \( \lor \), then by induction there is a sequence of \( 4^d \) instructions that \( \pi \)-accepts the language decided by the left sub-formula and a sequence of \( 4^{d-1} \) instructions that \( \pi \)-accepts the language decided by the right sub-formula, and by part (e) we can obtain a sequence of \( 2(4^{d-1} + 4^{d-1}) = 4^d \) instructions that \( \pi \)-accepts \( A \). If the last gate is \( \land \) we use part (d) in an identical way. Since \( d = O(\log n) \) we have \( 4^d = \text{poly}(n) \) and we conclude that every language in non-uniform \( \text{NC}_1 \) has polynomial-size width-5 branching programs.

(g) We are given a polynomial-size width-5 branching program. By adding dummy levels, we may assume it has \( 2^d \) levels, for \( d = O(\log n) \) – this at most doubles the size. Once an input \( x \) is fixed, for every pair of adjacent levels \( \ell_i \) and \( \ell_i + 1 \) there is a function \( f_i \) from \( \{1, 2, 3, 4, 5\} \) to \( \{1, 2, 3, 4, 5\} \) that is “computed” in level \( i \). Specifically, for \( k \in \{1, 2, 3, 4, 5\} \) we have \( f_i(k) \) equal to the destination of the outgoing 0 or 1 edge from node \( k \) in \( \ell_i \), depending on whether the variable labelling node \( k \) in \( \ell_i \) is 0 or 1 in the input. If we can compute the composition \( f = f_1 \circ f_2 \circ \cdots \circ f_{2^d} \) we can simply examine \( f(k) \) (where \( k \) is the number of the start node in level 1) and see if it leads to accept or reject.

The set of all functions from \( \{1, 2, 3, 4, 5\} \) to \( \{1, 2, 3, 4, 5\} \) is finite, so there is a constant size (and constant depth) “function composition circuit” \( C \) that takes as input the description of two functions \( g : \{1, 2, 3, 4, 5\} \rightarrow \{1, 2, 3, 4, 5\} \) and \( h : \{1, 2, 3, 4, 5\} \rightarrow \{1, 2, 3, 4, 5\} \), and outputs a description of the function \( g \circ h \). We can assemble these in a tree to obtain a circuit that computes the composition of more than 2 functions: for example, to produce a circuit that computes the composition of 4 functions, we have a copy of \( C \) computing the composition of the first two, a copy of \( C \) computing the composition of the last two, and a copy of \( C \) whose two sets of inputs are wired to the outputs of the two other copies. In general, this gives us a “function composition” circuit of depth \( O(\log n) \) with \( 2^d \) sets of inputs (each group expecting the description of a function from \( \{1, 2, 3, 4, 5\} \) to \( \{1, 2, 3, 4, 5\} \)), and a single set of outputs.

Now we add a constant-depth “pre-processing” circuit that supplies the inputs with descriptions of \( f_1, f_2, \ldots, f_{2^d} \) which are determined by the input \( x \). We also add a constant-depth “post-processing” circuit that takes the output and determines whether \( f(k) \) leads to accept or reject (recall the discussion above). We output 1 or 0 accordingly.

The overall circuit depth is \( O(\log n) \) and it has polynomial size; thus every language decided by a polynomial-size width-5 branching program is in non-uniform \( \text{NC}_1 \), as required.

2. Assume SAT can be decided in polynomial time by a TM \( M \) utilizing \( O(\log n) \) bits of advice. Given an instance \( \phi \), we determine whether \( \phi \in \text{SAT} \) as follows. For each possible advice string \( A \) (there are polynomially many), we use \( M \) with advice \( A \) in the standard self-reducibility argument to determine a satisfying assignment if there is one. Upon finding a purported satisfying assignment we check it, and accept if it indeed satisfies \( \phi \). Otherwise we continue, trying the other possible advice strings. If \( \phi \in \text{SAT} \), then when we try the correct advice
Let $n = 2^k$. We prove that $L(\bigoplus_n) \leq n^2$ by induction on $k$. When $k = 0$ we have $n = 1$, and the formula of size 1 consisting of a single literal computes $\bigoplus_1$. Otherwise, let $C$ be a formula of size $(n/2)^2$ computing the parity of $x_1, x_2, \ldots, x_{n/2}$ and let $C'$ be a formula of size $(n/2)^2$ computing the parity of $x_{n/2+1}, x_{n/2+2}, \ldots, x_n$. We can compute $C \oplus C'$ using $\land, \lor, \neg$ as: $(C \land \neg C') \lor (\neg C \land C')$. This formula has leaf-size $4(n/2)^2 = n^2$ as required.

(b) Let $C$ be an optimal formula for $f$. If $L(f) = 1$ then clearly $C$ is a single literal $x_i$ or $\neg x_i$, and we have $1 = FC(x_i) \leq L(f)$ as well as $1 = FC(x_i) = FC(\neg x_i) \leq L(f)$. If $L(f) > 1$, then we have two cases, depending on the last gate of $C$ (we can push all the negations down to the leaves, so that the only possible last-gates are $\land$ and $\lor$). If the last gate of $C$ is $\land$, then $f = g \land h$, and the subformulas for $g$ and $h$ are optimal (this is a special feature of formulas – if they were not optimal, we could replace them with smaller sub-formulas, contradicting our initial choice of $C$ to be optimal). Thus we have $L(g) + L(h) = L(f)$ and by induction we have $FC(g) \leq L(g)$ and $FC(h) \leq L(h)$. We conclude $FC(f) \leq FC(g) + FC(h) \leq L(g) + L(h) = L(f)$ as required. The argument for $\lor$ is identical, after observing that $g \lor h = \neg(\neg g \land \neg h)$, and using $FC(f) = FC(\neg f)$.

(c) We need to verify the three properties of a formal complexity measure. (i) consider $K(x_i)$. If we let $B$ be the all-zeros vector, and $A$ be the $i$-th unit vector, we see that $K(x_i) \geq 1$; in the other direction, for every vector in $x_i^{-1}(0)$, there is exactly one vector in $x_i^{-1}(1)$ that differs in exactly one coordinate, and vice versa, so $|H(A, B)| \leq |A|$ and $|H(A, B)| \leq |B|$. We conclude that $K(x_i) \leq 1$.

For part (ii), we see that the definition of $K(f)$ is symmetric with respect to $f^{-1}(0)$ and $f^{-1}(1)$, and so $K(f) = K(\neg f)$.

For part (iii), take $A$ and $B$ to be subsets maximizing the expression that defines $K(f \lor g)$ as suggested, and partition $A$ into (disjoint sets) $A_f \subseteq f^{-1}(1)$ and $A_g \subseteq f^{-1}(1)$. This partitions $H(A, B)$ into (disjoint sets) $H(A_f, B)$ and $H(A_g, B)$. The particular sets $A_f \subseteq f^{-1}(1), B \subseteq f^{-1}(0)$ and $A_g \subseteq g^{-1}(1), B \subseteq g^{-1}(0)$ imply that

$$\frac{|H(A_f, B)|^2}{|A_f||B|} \leq K(f)$$

$$\frac{|H(A_g, B)|^2}{|A_g||B|} \leq K(g)$$

To simplify expressions, we use $h_f = |H(A_f, B)|$, $h_g = |H(A_g, B)|$, $a_f = |A_f|$, $a_g = |A_g|$, $b = |B|$, and observe that $|A| = a_f + a_g$, and $|H(A, B)| = h_f + h_g$. To prove that $K(f \lor g) \leq K(f) + K(g)$, we will show $(h_f + h_g)^2/(a_f + a_g)b \leq h_f^2/(a_f b) + h_g^2/(a_g b)$.

Multiplying both sides by $(a_f + a_g)a_f a_g b$, we see that this inequality is equivalent to

$$(h_f + h_g)^2 a_f a_g b \leq h_f^2 a_g (a_f + a_g) + h_g^2 a_f (a_f + a_g).$$
Multiplying out and cancelling terms we get the equivalent inequality

\[ 2hfha_fag \leq h_f^2a_g^2 + h_g^2a_f^2 \]

which can be rewritten as \( 0 \leq (h_f a_g - h_g a_f)^2 \), which is clearly true. Thus

\[
K(f \lor g) = \frac{(h_f + h_g)^2}{(a_f + a_g)b} \leq \frac{h_f^2}{a_f b} + \frac{h_g^2}{a_g b} \leq K(f) + K(g)
\]

as required.

(d) Let \( A = \bigoplus_n^{-1}(0) \) and \( B = \bigoplus_n^{-1}(1) \). Then \( H(A, B) = n2^{n-1} \), and so \( K(\bigoplus_n) \geq (n2^{n-1})^2/(2^{n-1})^2 = n^2 \). Since \( K \) is a formal complexity measure, we have \( L(\bigoplus_n) \geq n^2 \).