Solutions

Exercise 1. There exists an inductive algorithm that uses at most $2n$ flips. First bring the largest pancake to the bottom, and then sort recursively. To bring the largest one to the bottom, insert the spatula below the largest one, and flip. Then flip the whole pile. Note that we can slightly improve this to $2n-3$ flips, because 2 pancakes can be sorted using at most 1 flip.

Exercise 2. Use a greedy algorithm to color the graph and check in the end whether this is indeed a coloring, i.e., whether no edge connects two nodes of the same color.

1. Take a random vertex and color it red (or blue, the color does not matter here).
2. Color all neighboring uncolored vertices with the opposite color. Repeat this step until no uncolored neighbors are left.
3. If there are uncolored nodes left (these must belong to a different connected component), repeat the previous step.
4. Check if some edge connects two vertices of the same color. If this is the case, output "not 2-colorable", otherwise output "2-colorable".

We have to show that this algorithm works. Clearly, if our algorithm outputs "colorable", this means it has constructed a coloring and hence, the result is correct. Otherwise, if it says "not colorable", it has colored 2 neighboring vertices with the same color. We show that vertices lie on a cycle of odd length, and hence the graph is not colorable.
All pairs of vertices with the same color are connected by some path of even length, and pairs with different colors are connected by a path of odd length. This follows by induction from the algorithm.

Exercise 3. Let $0 \leq d_1 < d_2 < \cdots < d_\ell$ such that $b = \sum_{i=1}^{\ell} 2^{d_i}$. Note that $d_\ell \leq \log b$. The idea is to factorize a^b using this decomposition. Every factor $a^{2^{d_i}} \bmod c$ can be computed by iterative squaring. To compute all these factors, we use d_ℓ multiplications modulo c. Then we multiply the factors. In total we performed $2d_\ell$ multiplications. On a Turing machine this requires time $(\log b)(\log c)^2$. This is polynomial in the input length (which is proportional to $\log(abc)$).

Exercise 4. The algorithm processes the input strings a_1, a_2, …, a_n one by one and maintains a set S of all lengths that can be realized by concatenating processed strings. Initially, S equals the singleton $\{0\}$. For each string of length i, it replaces S by $S \cup \{j+i \colon j \in S\}$. After all strings have been processed, the algorithm checks whether the number $|a_0|$ is in the set.
We show that this algorithm runs in polynomial time in the input length. The size of S is bounded by the value of its maximal element, and this equals $|a_1| + \cdots + |a_n|$. This is bounded by the input length. Each update can be computed in polynomial time, and the number of updates equals the number of input strings, which is bounded by the input length. Hence, the algorithm runs in polynomial time.

Exercise 5. We discussed one algorithm in detail in class. Here is a different algorithm. This algorithm is simpler but slower (still polynomial though). On input $w = w_1 w_2 \cdots w_n \in \Sigma^*$, the algorithm computes the set of pairs (i,j) with $i < j$ such that $w_i w_{i+1} \ldots w_j \in A^*$.
We can compute this set iteratively. Initially, let

$$S = \{(i,j) \colon i < j \text{ and } w_i \cdots w_j \in A\}$$

In each iteration, we add to S all pairs (i,k) for which there exists a j such that $(i,j) \in S$ and $(j+1,k) \in S$. If during some iteration no new elements are added to S, we are finished. We accept w if $(1,n) \in S$, otherwise, we reject.
This algorithm runs in polynomial time, because in each iteration, it suffices to check all triples $i < j < k \leq n$. The maximal size of S is $n(n-1)/2$, hence the algorithm must stop after at most polynomially many iterations.

Exercise 6. We use the verifier definitions. For the union, the certificate that $x \in A \cup B$ is a bit that indicates to which of the two sets x belongs, together with the certificate for that set. For the intersection, the certificate is given by the pair of certificates of both sets.

Exercise 7. On input $w \in A^*$, a nondeterministic machine can guess the lengths of the parts of w that belong to A. We can also prove this using the verifier definition: a certificate $w \in A^*$ is given by a splitting of w, together with certificates for each part in the splitting.

Exercise 8. The problem is in NP, because a certificate for a satisfiable formula is given by the values that make the formula true. Assume $\mathsf{P} = \mathsf{NP}$. Hence, there exists a polynomial time algorithm that decides whether a formula is satisfiable.
The algorithm that computes the assignment is defined by recursion. If the formula has no arguments, then it is easy to decide whether it is satisfiable, and if it is satisfiable, then the assignment is given by the empty string. Now assume that ϕ has at least 1 argument. Then, we obtain two formulas ϕ_0 and ϕ_1 by setting the first variable to 0 and to 1. If ϕ is satisfiable, then at least one of these formulas must be satisfiable and by assumption, we can compute which one. We have now computed the first variable, and we obtain the others by a recursive call with argument ϕ_0 or ϕ_1. The total number of recursive calls equals the number of variables. Hence the algorithm runs in polynomial time.

Exercise 9. To show that a language A is NP-complete, we need to show that every language $X \in \mathsf{NP}$ reduces to A. Assume A is neither empty nor equal to Σ^*. This implies that there exist $a_{\text{in}} \in A$ and $a_{\text{out}} \notin A$. The reduction f from X to A is defined by

$$f(x) = \begin{cases} a_{\text{in}} & \text{if } x \in X \\ a_{\text{out}} & \text{otherwise.} \end{cases}$$

By construction $x \in X \Leftrightarrow f(x) \in A$. This mapping can be computed in polynomial time because $X \in \mathsf{NP} = \mathsf{P}$ can be decided in polynomial time. Hence, f is indeed a reduction.

Exercise 10. We need to map a graph (V, E) and a number k to a set of linear inequalities. There will be a variable x_v for each vertex $v \in V$, and an inequality for each edge. The idea is that vertices in the independent set are represented by variables $x_v = 1$ and the others by variables $x_v = 0$. The pair $((V, E), k)$ is mapped to the equations:

$$x_u + x_v \leq 1 \quad \text{for all} \quad (u, v) \in E$$

and the equation

$$\sum_{u \in V} -x_u \leq -k$$

These inequalities can be represented using an integer matrix and vector.

1) We show that this reduction can be computed in polynomial time. Recall our convention that we write graphs using the incidence matrix representation. A graph on n vertices is represented as a binary $n \times n$ matrix (hence by a bitstring of length n^2). A and b have dimensions $n \times m$ and m with $m = |E| + 1 \leq n^2 + 1$.

2) If there exists an independent set of size k, then the equations are all satisfiable. As explained above, we set $x_v = 1$ for all vertices in the independent set, and $x_v = 0$ for the others. By definition of independent set, no edge $(u, v) \in E$ connects two elements, thus $x_u + x_v \leq 1$.

3) Finally we need to show that if the equations are satisfiable, the graph has an independent set of size k. Fix some solution of the equation. By the last equation, there are at least k vertices v for which $x_v = 1$. No two such vertices can be connected, because otherwise, the equation corresponding to this edge would be violated. We have proven that the mapping is a reduction.

Exercise 11. Let ϕ be a formula in 3CNF. We map ϕ to (a representation of) the pair (M, ϕ), where M is a Turing machine that on input (a binary representation of) a 3CNF-formula ψ uses brute force search to find a satisfying assignment of ψ, and if such an assignment is found, halts, and otherwise goes to an infinite loop. We must show that this is a reduction.

1) The mapping can be computed in polynomial time, because the machine simply adds a description of the machine M and computes some pairing function.

2) and 3) By construction, ϕ is satisfiable if and only if $M(\langle\phi\rangle)$ halts.

Exercise 12. We map a graph (V, E) and an integer k to a new graph. There are 3 cases.

If $|V| = 2k + 1$ then we simply output the same graph.

If $|V| < 2k + 1$, we add ℓ isolated vertices to V (vertices that are not incident on any edges), where ℓ is the difference between the right and left hand-side of the inequality.

If $|V| > 2k + 1$, we add ℓ vertices to V that are all connected to all vertices in V. Again we choose ℓ to be the difference between the left and the right hand-side of the inequality.

It can be verified that this is indeed a reduction.