Problem 7.7 (5 pts)
Show that NP is closed under union and concatenation.

Union
Let $L_1$ and $L_2$ be languages in NP and for $i = 1, 2$, let $V_i(x, c)$ be an algorithm that, for a string $x$ and a possible certificate $c$, verifies whether $c$ is a certificate for $x \in L_i$. Therefore, $V_i(x, c) = 1$ if certificate $c$ verifies $x \in L_i$, and $V_i(x, c) = 0$ otherwise.

Since both $L_1$ and $L_2$ are in NP, we know that $V_i(x, c)$ terminates in polynomial time $O(|x|^d)$ for some constant $d$.

To show that $L_3 = L_1 \cup L_2$ is also in NP, we will construct a polynomial-time verifier $V_3$ for $L_3$. Since a certificate $c$ for $L_3$ will have the property that either $V_1(x, c) = 1$ or $V_2(x, c) = 1$, we can construct a verifier $V_3(x, c) = V_1(x, c) \lor V_2(x, c)$. Thus, $x \in L_3$ if and only if there is a certificate $c$ such that $V_3(x, c) = 1$. The proposed verifier $V_3$ will run in time $O\left(2(|x|^d)\right)$, which is polynomial. Therefore, the union $L_3$ of two languages is also in NP, so NP is closed under union.

Concatenation
Now we will show that $L_4 = L_1 \circ L_2$ is in NP, where $L_1$ and $L_2$ are languages in NP with verifiers $V_1$ and $V_2$ as in the solution for the previous part. Construct a verifier $V_4(x, c)$ for the string $x$ and possible certificate $c$.

Suppose $|x| = n$. Define $V_4(x, c) = 1$ if and only if $c = k\#y\#z$, where $\#$ is a new symbol, $k \in \{0, 1, \ldots, n\}$, and $V_1(x_1 \ldots x_k, y) = 1$ and $V_2(x_{k+1} \ldots x_n, z) = 1$.

The variable $k$ specifies the position where the original string $x$ should be split into two parts, and $y$ and $z$ are the certificates for the two parts. The verifier $V_4$ will run in time $O(|x|^d)$ since $|x_1 \ldots x_k| \leq |x|$ and $|x_{k+1} \ldots x_n| = |x|$. Additionally, $V_4(x, c) = 1$ if and only if $x \in L_4$. Therefore, the language $L_4$, the concatenation of two languages in NP, is also in NP. So, NP is closed under concatenation.

Problem 7.10 (5 pts)
Show that $ALL_{DFA}$ in $P$.

$ALL_{DFA} = \{ \langle M \rangle \mid M$ is a DFA and $L(M) = \Sigma^* \}$

Given a DFA $M$, we must determine whether $M$ accepts all strings from $\Sigma^*$ or not. To check whether $M$ rejects some string, we search to find if a non-accepting state is reachable from the initial state (taking only polynomial time using a search, i.e. depth first search). If a non-accepting state is reachable in $M$ then there is a string not belonging to $L(M)$, meaning $L(M) \neq \Sigma^*$ and $\langle M \rangle \notin ALL_{DFA}$. If only accepting states are reachable in the DFA $M$ then $L(M) = \Sigma^*$ and $\langle M \rangle \in ALL_{DFA}$. Thus, we just described a polynomial time algorithm for deciding $ALL_{DFA}$ and so $ALL_{DFA} \in P$. 
Problem 7.12 (5 pts)
Call graphs $G$ and $H$ isomorphic if the nodes of $G$ may be reordered so that it is identical to $H$. Let $ISO = \{ \langle G, H \rangle \mid G$ and $H$ are isomorphic graphs $\}$. Show that $ISO \in NP$.

We construct a polynomial time verifier $V(\langle G, H, c \rangle)$ such that $V(\langle G, H, c \rangle) = 1$ iff $G$ and $H$ are isomorphic graphs. We check that the certificate $c$ is an isomorphic mapping between $G$ and $H$ as follows:

Let $G = (V, E)$ and $H = (U, D)$
1. If $|V| \neq |U|$ or $|E| \neq |D|$, reject.
2. For all $v \in V$, if $c(v) \notin U$, reject.
3. For all $v_1, v_2 \in V$, if $c(v_1) = c(v_2)$ for $v_1 \neq v_2$, reject.
4. For all $(v_1, v_2) \in E$, if $(c(v_1), c(v_2)) \notin D$, reject.
5. Accept as $c$ has been shown to be an isomorphic mapping from $G$ to $H$.

All the above steps take polynomial time. Because we have found a polynomial time certifier, $ISO$ is in $NP$.

Problem 7.13 (5 pts)
Let $MODEXP = \{ \langle a, b, c, p \rangle \mid a, b, c, \text{ and } p \text{ are positive binary integers such that } a^b \equiv c \pmod{p} \}$. Show that $MODEXP \in P$.

Because $b$ is a binary integer, let $b = b_1 b_2 \ldots b_{n-1} b_n$. (So the decimal representation of $b$ is $\sum_{k=1}^{n} b_{n-k+1} 2^{k-1}$.) By the hint, we observe that $a^{(10)_2} = a^2$ and $a^{(1000)_2} = ((a^2)^2)^2$.

Construct the following algorithm to decide $MODEXP$:
$A =$ “On the input $\langle a, b, c, p \rangle$, where $a, b, c, \text{ and } p \text{ are binary integers}:
1. Let $T = 1$ (and $n = \lfloor \log_2 b \rfloor$).
2. For $i = 1$ to $n$,
   a. if $b_i = 1$, $T = (a(T^2) \pmod{p})$,
   b. if $b_i = 0$, $T = (T^2 \pmod{p})$;
3. Return $T \pmod{p}$.
4. If $T = c \pmod{p}$, accept; otherwise, reject.”

Assume that $a$, $b$, $c$ and $p$ are at most $m$ bits (so $n \leq m$). It is known that two $m$-bit numbers cost $O(m^2)$ unit time to do multiplication and division (and hence modular), so each $i$ costs $O(m^2)$ time. The total cost of the for loop is $O(m^2) \times n = O(m^3)$, which is the dominant cost of all steps. The time complexity of the algorithm $A$ is polynomial in the length of its input.

Problem 7.18 (5 pts)
Show that if $P = NP$, then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.
Let $A$ be any language in NP and let $B$ be another language not equal to $\emptyset$ or $\Sigma^*$. Then there exist strings $x \in B$ and $y \notin B$. To reduce an instance $w$ of $A$ to that of $B$, we just check in polynomial time if $w \in A$. If yes, we output $x$ and $y$ when $w \notin A$. That is, $f(w) = x$ if $w \in A$, and $f(w) = y$ if $w \notin A$. So $w \in A$ iff $f(w) \in B$.

The languages $\emptyset$ and $\Sigma^*$ cannot be NP-complete, because to reduce a language $A$ to a language $B$, we need to map instances in $A$ to instances in $B$ and those outside of $A$ to outside $B$. However, for $B = \emptyset$, there are no instances in $B$ (and none outside $B$ for $B = \Sigma^*$) which means there cannot be such a reduction from any language $A \neq \emptyset, \Sigma^*$.

**Problem 7.20 (5 pts)**
We generally believe that $PATH$ is not NP-complete. Explain the reason behind this belief. Show that proving $PATH$ is not NP-complete would prove $P \neq NP$.

We know that $PATH$ is in P. If $PATH$ would be NP-complete, this would imply that for all $L \in NP$, $L \leq_p PATH$. But this again implies that for all $L$ in NP, $L$ is in $P$. Thus $P = NP$ would follow, which we believe is not true.

We show that proving that $PATH$ is not NP-complete implies that $NP \neq P$. We show this by contraposition:

$P = NP$ implies that $PATH$ is NP-complete by Problem 7.18. The contraposition of this implication is exactly that $PATH$ is not NP-complete implies $P \neq NP$. 