Solution of Homework 4

1. (a) Problem U is NP-complete. It is in NP because given (G, F, k) where $F \subseteq G, |F| \leq k$ is a set of fire-stations it is possible to verify (G, F, k) in polynomial time.

   Problem U is NP-hard because *Vertex Cover* can be reduced to it. Given an instance $(G = (V, E), k)$ of *Vertex Cover* it is converted into into an instance $(G', k' = k, w' = 1)$ of problem U as follows. Vertices of G', $V' = V_V \cup V_E$ represent both vertices and edges of G. Edges of $G', E' = E_1 \cup E_2$ where $E_1 = \{(i, v_j) | i \in V_V, v_j \in V_E$ if edge j is adjacent to vertex i in graph $G\}$. All edges in E_1 have weight 1. Edges $E_2 = \{(i, j) | i, j \in V_V\}$ with weights 2/3.

   By construction it follows that $(G, k) \in$ *Vertex Cover* iff $(G', k', w') \in U$. There is a brute force algorithm for U that takes $\binom{n}{k}$ time, hence upper bound of U is $\binom{n}{k}$. Lower bound is $O(n)$.

   (b) The upper and lower bound of V is $O(n)$. The lower bound follows from the upper bound and the fact that every clause needs to be examined at least once. Upper bound can be established as follows. Every clause $a \bigvee b$ is equivalent to $\overline{a} \to b$. To determine whether a formula is satisfiable it is sufficient to verify whether the set of equivalent implications is not contradictory, i.e whether there are any implications of the type $x \to \overline{x}$. (Note that due to the transitivity of implications, $a \to b \bigwedge b \to c$ is equivalent to $a \to c$).

   Formula S can be converted into a directed graph $G = (V \cup V', E)$ where V represent set of variables of S, V' of negations of variables of S and every clause $a \bigvee b$ of S will correspond to two (directed) edges in E, one is $(\overline{a}, b)$ another is $(a, \overline{b})$, where $a, b \in V, \overline{a}, \overline{b} \in V'$.

   The formula S is satisfiable iff the graph G constructed above does not contain any cycles (or strongly connected components) that contain both x and $\overline{x}$ for some variable x. This property of G can be verified in linear time using DFS and graph G can be constructed from S also in linear time, thus upper bound of problem V follows.

   (c) W is NP-hard, upper and lower bounds are unknown.

   (d) Upper bound of W is 2^n, lower bound is $O(n)$.

   (e) Upper and lower bounds of X are unknown.

   (f) Upper bound of Y is $O(n \log n)$, lower bound is $O(n)$.

   (g) Upper and lower bounds of Z are unknown.

2. The idea is to have infinitely many entries of M^* in the table, so eventually when $g(n)$ becomes greater (as well as asymptotically greater) than $f(n), \forall n > N$ there will be at least one copy of M^* for this N, resulting in contradiction.

Technically this can be done by making M^* simulate M whenever it receives an input of the form $< M > 10*$.

3. (a) Class P is closed under union, intersection and complement. This can be shown by constructing appropriate TMs. Let M_1 be TM for A, M_2 be TM for B. TM for $A \cup B$ runs M_1 and M_2 on input, accept if at least one of them accept. TM for $A \cap B$ runs M_1 and M_2 on input, accept if both M_1 and M_2 accept. TM for $\overline{A}$ negates the output of M_1.

   (b) Class NP is closed under union and intersection, proof is similar to the 3(i), except that now we have nondeterministic TMs.

   (c) The idea of r.e proof was the following: if you have an enumeration of machines in a class (r.e), we can take the complement of the diagonal set $\overline{H}$ (namely the complement of the halting problem) and clearly it cannot be in the class (r.e). But the diagonal set, i.e, H itself is in r.e. So, the complement is in co-r.e

   Suppose that we were to try to imitate that proof. We can enumerate all NP machines. We can also diagonalize over them, creating set G. Also $\overline{G} \notin NP$ by construction. However set G is also not in NP, since for every $A \in$ NP that runs in $p(n)$ time, algorithm for G should run at least for $p(n)$ steps, but there is no such polynomial $q(n)$ (running time of G) such that $q(n) \geq p(n)$ for all polynomials $p(n)$ (take $p(n) = nq(n)$ to obtain a contradiction).

   (d) Let A be NP-complete, then $\forall X \in NP, X \leq_m^P A$ and $A \in NP$. Then $\overline{A} \in co-NP$. Let $\overline{X} \in co-NP$ then $x \in X \Leftrightarrow f(x) \in A$ therefore $x \notin X \Leftrightarrow f(x) \notin A$ therefore $x \in \overline{X} \Leftrightarrow f(x) \in \overline{A}$ therefore $\overline{X} \leq_m^P \overline{A}$.

   (e) Let S be an NP-complete set and $S \in co-NP, S = \overline{A}, A \in NP$. Then $\overline{S} = A \in NP$. Also by previous exercise $\overline{S}$ is co-NP-complete. Therefore $\overline{L} \leq_m^P \overline{S} \in NP, \forall \overline{L} \in co-NP$. Thus $\overline{L} \in NP$. Similarly all L that are in NP are also in co-NP.

   (f) No. Machine M in proof of Theorem 4.16 would be non-deterministic polynomial time (since both M_1 and M_2 are) hence it cannot be used as deterministic polynomial time machine.

   (g) Assume that $NP \neq co-NP$ and suppose that $P = NP$ therefore $co-P = co-NP$ therefore $P \neq co-P$, contradiction since P is closed under complementation.

   It is possible that $P \subset NP = co-NP$.

4. (a) **9.10**

   Given any language $L \in NP$, the algorithm that reduces L to SAT may potentially take more than $O(n^k)$ time, or the output size of the algorithm may differ from the original input size. (For the later, think

SAT is in $O(n^5)$. Assume the reduction takes $O(n^2)$ time, but the output size is $O(n^2)$. Clearly then to solve the original problem by reducing to SAT will take time $O(n^{10})$.) Therefore, if the reduction algorithm takes $t(n)$ time and the size of the output is $f(n)$, then to solve the problem we need time $O(t(n) + (f(n))^k)$ which may not be in $O(n^k)$.

(b) **9.18**

Let $A \in TIME(n^2)$, machine M decides A in time $O(n^2)$. Given a string $x = s\sqcup^j$ following machine M' decides x in $O(|x|)(= O(n^2))$ time. It simply simulates M on s, and this takes $O(n^2) = O(|x|)$.

(c) **9.19**

First we note that it is enough to show that NEXP $\subseteq$ EXP under the given hypothesis. Let's take an arbitrary language L in NEXP. Assume $L \subseteq NTIME(2^{n^k})$ for some constant k, that is there is a non-deterministic TM, say N , that accepts the language L in time 2^{n^k}. We shall now decide the language L in deterministic exponential time, i.e. in TIME(2^{n^k}).

We define $f(n) = 2^{n^k}$. Now given any arbitrary x, we produce $y = pad(x; f(n))$, where $n = |x|$. (Note this can be done in deterministic exponential time, since we need only to add $2^{n^k} - n$ many \$s.) We next define a TM N', that takes a padded string and simulates N (recall N decides L) only on the portion having no \$, that is on x. For example, N' takes the string $y = x\$^{f(n)-n}$ and simulates N on x and accepts if N accepts and rejects otherwise.

We note that our N' is a non-deterministic machine that decides the language $pad(L; f(n))$. (Note that N accepts x if and only if N' accepts $pad(x; f(n))$.) However, to decide $pad(L; f(n)), N'$ simply simulates N. Hence, it takes time at most $f(n)$. Since the input to N' has size $f(n)$, we conclude N' decides $pad(L; f(n))$ in $NTIME(n)$ i.e. in non-deterministic linear time. Therefore, its language is in NP, i.e. $pad(L; f(n))$ is in NP. By the hypothesis, NP = P. Therefore, there is a deterministic machine, say M' that decides $pad(L; f(n))$ in polynomial time, say in time n^l, for some constant l.

Therefore, without simulating $pad(L; f(n))$ using N', we can as well simulate it using M' and thus decide $pad(L; f(n))$ which will take at most $f(n)^l$ deterministic time.

Summarizing, we describe our final deterministic machine that decides L in deterministic exponential time. To decide whether x is in L or not, M first pads x up with enough \$ i.e. with $(f(n) - n)\$s$ (say the resulting string be y). Next M simulates M' on y. Finally, M accepts if M' accepts y and reject otherwise.

First we note that M correctly decides L. (Recall $x \in L$ if and only if $pad(x; f(n)) \in pad(L; f(n))$.) How much time then M takes to decide L? This is clearly $(f(n) - n) + f(n)^l$ i.e. to pad x we need $(f(n) - n)$ and for simulating M' we need another $f(n)^l$.

But, this is at most $O(2^{n^k l})$ i.e. which is at most $O(2^{n^{k+1}})$ which is (deterministic) exponential. Thus we proved that if NP = P, then EXP = NEXP.

(d) **9.20**

Let assume first that $A \in P$. Therefore, there is an algorithm, say M , that decides A in polynomial time, say n^l time. We shall use M to decide $pad(A; n^k)$. Given $x\$^l$ and k we first check whether it is of the desired form or not. (We simply count the \$'s and decide.) If it is not of the desired form, we reject. Otherwise, we simulate M on x and accept if M accepts. It is easy to see that this algorithm correctly decides $pad(A; n^k)$. However, the time taken to decide $pad(A; n^k)$ is no more than $((n + max(n^k - n; 0)) + (n^l))$ (to count we need $(n + n^k)$ and an additional n^l for simulating M), which is polynomial in the size of the input which is $(n + max(n^k - n; 0))$ (for any k) i.e. at least n. Therefore $A \in P \Rightarrow pad(A; n^k) \in P$.

To prove the reverse, assume for some k there is an algorithm that decides $pad(A; n^k)$ in poly time, we shall prove that A is in P. (Using the first part, we then claim that for all natural number k' , $pad(A; n^{k'})$ is in P.) To prove this, assume there is an algorithm, say M, that decides $pad(A; n^k)$ in time m^l (here m has been taken to be the input size, where $m = n + max(n^k - n; 0)$, which is at most $(n + n^k)$). To decide A we pad it up and then simulate N. Formally, given x, we first produce $pad(x; n^k)$ which takes at most $(n + n^k)$ time. We then simulate M on $pad(x; n^k)$ and accepts if M does. Total time taken by this procedure is no more than $((n + n^k) + (n + n^k)^l)$ which is again a polynomial in n i.e. the input size. We use the first part of this proof to extend the result for all natural numbers. Thus, we established the desired result.