Honors Theory of Computation/Yap/May 8, 2000
Final Exam

Answer all questions. This is an OPEN book, in-class exam. Please write ONLY in complete English sentences (even if they involve equations). Up to 10% of the points are for the clarity of your proofs.

Problem 1 (Short Questions) [10 points for each part]
For multiple choice questions, a short justification is needed. In particular, for True/False/Unknown questions, please give some supporting facts even in the UNKNOWN case.

(a) State two natural problems that are in *NP* and which are NEITHER known to be *NP*-complete NOR known to be in *P* .

SOLUTION: Primes, Graph-Isomorphism.
REMARKS: Actually, for any problem L in $NP \cap$ co-NP that is a valid answer, co-L would also be valid. So Primes is of this form, so "Composites" can be offered as an answer. Some students also point out LINEAR DIVISIBILITY as a possibility. Given integers a, b, does there exist an integer of the form $ax + 1$ that divides b.

(b) True/False/Unknown: For any complexity function f, TIME(f) is properly contained in TIME($f(n) \log n$).

SOLUTION: false, as we know from Gap theorem, there is an f such that TIME(f) = TIME($2^f(n)$). We can make sure that $f(n) \log n \leq 2^f(n)$.

NOTE: you might be tempted to say YES because of the time hierarchy theorem (this theorem says, in the strongest form which we mentioned in class and in the midterm that if $f(n)$ is proper then TIME(f) is properly contained in TIME($f \log f$). The trick, of course, is that you need f to be proper. I realize that some of you assume (as does Christos) that "complexity function" always implies "proper". Adjustment will be made for this fact.

(c) True/False/Unknown: If $\Sigma_2 P \subseteq BPP$ then $\Sigma_2 P = \Pi_2 P$.

SOLUTION: T. Since *BPP* has polynomial circuits, this implies *NP* has poly. circuits. Then Theorem 17.13 says that the polynomial hierarchy collapses to the second level.

(d) If Hamiltonian Path is in $TIME(n^3)$, how fast can you solve 3SAT? Hint: Theorem 9.7 of book.

SOLUTION: Then 3SAT is in $TIME(n^6)$. To see this, a 3CNF formula F has size $3m$. The resulting graph $G = G(F)$ constructed by the reduction in Theorem 9.7 has $O(m)$ vertices and $O(m^2)$ edges. This will require $O(m^6)$ time to solve.

(e) True/False/Unknown: The STEINER TREE has an ε-approximation algorithm for some $\varepsilon < 1$. [This problem is MAXSNP-complete, see Exercise 13.4.9 of text.]

SOLUTION: True, as this is a direct consequence of theorem 13.8 (see discussion following the theorem). Point is, you need not know anything about the nature of the STEINER TREE problem.

(f) True/False/Unknown: Let g be a n-input quantum gate whose corresponding unitary transformation is given by a $2^n \times 2^n$ unitary matrix M_g. Then g is a classical gate iff M_g is a 0/1 matrix in which each row and each column has exactly one 1.

SOLUTION: TRUE. This means that g converts pure states to pure states. To say that M_g has one 1 in each row and column is equivalent to saying that g is a permutation of $\mathbb{B}^n$.

(g) True/False/Unknown: The RSA cryptosystems is secure if and only if $P \neq NP$.
SOLUTION: UNKNOWN. If RSA are secure, then necessarily $P \neq NP$. But $P \neq NP$ does not mean the functions used in the RSA system is secure (we still do not know if one-way functions exist).

(h) A Boolean circuit $U : \mathbb{B}^{n+m} \to \mathbb{B}$ is said to be (n, m)-universal if for all $f : \mathbb{B}^n \to \mathbb{B}$, there exists a $y \in \mathbb{B}^m$ such that for all $x \in \mathbb{B}^n$, $f(x) = U(x, y)$. Where is the place of the language

$$L_u := \{(U, n, m) : U \text{ is } (n, m)\text{-universal}\}$$

in the polynomial hierarchy?
SOLUTION: The quantifier structure is
(for all circuits C on n inputs) (there is a setting y) (for all inputs x) [U(x,y)=C(x)].
So this is in $\Pi_3 P$.

Problem 2 (FNP and completeness) [30 points]
Prove that $FSAT$ is FNP-complete (see chapter 10 for definitions). HINT: recall the completeness proof of SAT.

SOLUTION: By definition of FNP, we immediately see that $FSAT$ is in FNP. So we must show that any $F \in FNP$ is reducible to $FSAT$. Basically, use the construction in Cook's theorem. Let $R(u, v)$ be a balanced, polynomial-time relation such that $F(x) = y$ imply $R(x, y) = 1$. Let M be a P-machine for $R(u, v)$. Then there is a log-space computable function which, for any given input x to M, computes a Boolean circuit $f(u, v)$ such that $f(x, y) = 1$ iff $R(x, y) = 1$.

To reduce F to $FSAT$, we need to show two reduction functions, r and s. Given any input x for F, we define $r(x)$ to be the Boolean circuit $C(v) = f(x, v)$ (where v are the free variables of the circuit). If we feed $C(v)$ as input to $FSAT$, we will receive an output y such that $C(y) = 1$. We can choose the function s to be the identity function: $s(y) = y$.

REMARKS: Here is a bad argument to show that $FSAT$ is in FNP: guess an assignment for an input formula, and verify it, then accept. THis is not how the class FNP is defined! Also, most people make this fundamental mistake: corresponding to an FNP problem F, there is a corresponding polynomial-time decidable and polynomially balanced relation $R(x, y)$ such that $F(x) = y$ implies $R(x, y) = 1$ (recall that $F(x)$ can also give the specially coded answer "NO"). BUT, given F, there will generally be many possible choices for the relation R. Alternatively, let $L_R = \{x : (\exists y)R(x, y)\}$.

So the following attempted proof is simply wrong: "Since $L_R \in NP$, let L_R be reduced to SAT via some polynomial-time function f. Recall that to reduce F to $FSAT$, we need to produce two log-space computable functions r, s with the appropriate properties. Let r be chosen to be f, etc."

Problem 3 (BPP and NP) [30 points]
Suppose $NP \subseteq BPP$. Prove that $NP = RP$. HINT: Construct an RP-machine to accept SAT: construct a satisfying assignment $(b_1, \ldots, b_n) \in \mathbb{B}^n$ for an input Boolean formula $F(x_1, \ldots, x_n)$, by repeated invocation of a BPP-machine which accepts SAT.

SOLUTION: That is a well-known result of Ko (1982). It is an exercise in Christos' book. It is also an old theory exam question (with answers), or you can read about it in in my complexity book (chapter 8).

If $b_1, b_2, \ldots, b_k$ ($k \leq n$) are Boolean values, let $F_{b_1, b_2, \ldots, b_k}$ denote the formula in which we replace x_i by b_i ($i = 1, \ldots, k$). The idea is this: operate in stages (stage $0, 1, \ldots, n$). In stage i, we determine b_{i+1}.

Stage 0: we first ask the BPP-machine whether F_0 is satisfiable. If the answer is yes, we set $b_1 = 0$ and go to stage 2. If the answer is no, then we ask the BPP-machine whether F_1 is satisfiable. If the second answer is yes we set $b_1 = 1$ and go to stage 2. If the second answer is no, we answer NO. [Clearly, this NO can be wrong, but hopefully with small probability.]

Stage $i = 1, \ldots, n-1$: Assuming $b_1, \ldots, b_i$ has been determined, we ask the BPP-machine if $F_{b_1,\ldots,b_i 0}$ is satisfiable. If yes, set $b_{i+1} = 0$ and go to stage $i+1$. If no, we ask the BPP-machine if $F_{b_1,\ldots,b_i 1}$ os satisfiable. If the second answer is yes, we set $b_{i+1} = 1$ and go to the next stage, else we answer NO.

In stage n, we simply check whether $F_{b_1,\ldots,b_n}$ is satisfiable or not (directly!) and answer YES or NO accordingly.

ANALYSIS: the YES answer happens only in stage n, and it is never wrong. The NO answer can happy in any stage, with small probability of error. We leave it as an exercise to determine this probability. A simple trick is to assume that the probability of error in our BPP-machine is n^{-2} (polynomially small). We can do this because of the trick of majority voting.

Problem 4 (Factorization) [20+15+20 points]
Consider the following language:

$$L_f = \{x\#y : (\exists z)[z \le y \text{ and } z \text{ divides } x]\}$$

where x, y, z are all binary numbers. We may regard L_f as encoded over the alphabet $\{0, 1, \#\}$.

(i) Let $Factor(x)$ be the function which on input x (a binary number), outputs some representation of the prime factorization of x. E.g. $Factor(300) = 2^2 \cdot 3 \cdot 5^2$. Show that $Factor \in FP^{(L_f)}$ (i.e., it is computed by an oracle FP-machine with L_f as oracle).

SOLUTION: Let x be the input. Use binary search to find the smallest prime factor. (a) Determine the number of bits in the binary representation of the smallest factor. For $i = 1, 2, 3, \ldots$, let $y = 1^i$ and we query if $x\#y \in L_f$. If the first positive answer is at the kth step, we know that $x\#1^k \in L_f$ and $x\#1^{k-1} \notin L_f$. Hence the smallest factor has k bits. (b) Determine the remaining $k-1$ bits of this factor. For $i = 1, \ldots, k-1$: assume that the first i bits have been determined and is the bit-string w_i. Initially, $w_1 = 1$. We then query if $x\#w_i 0 1^{k-i-1} \in L_f$. If so, let $w_{i+1} = w_i 0$. Otherwise, $w_{i+1} = w_i 1$. (c) Now we have obtained the smallest prime factor p_1. It is easy to determine the largest exponent e such that $p_1^e | x$. Once this is done, we can replace x by x/p_1^e, and repeat the process. It is easy to see that we only make a linear number of queries to L_f.

(ii) Prove that L_f is in $NP \cap$ co-NP.

SOLUTION: Clearly $L_f \in NP$. To show that $x\#y \notin L_f$: guess a factorization of x together with certificate for each prime factor. Verify this factorization and certificates. Finally, accept iff y is smaller than each of the prime factors.

(iii) Suppose L_f is NP-complete. Show that an open problem of complexity theory would be solved using this result.

SOLUTION: If L_f is NP-complete, then $NP =$ co-NP (a major open problem) NOTE: it is not enough to just quote Proposition 10.2 of the book, you will need Proposition 10.1 also. Anyway, here is a direct argument: We first need to show that co-NP is closed under reduction (recall that we have only proved that NP is closed under reduction): if $L \in$ co-NP and $L' \le L$ via f, then we can accept L' using the following co-NP-machine. On input x, we compute $f(x)$ and then run any co-NP-machine for L. Then $x \in L'$ iff every computation path accepts. By the NP-completeness of L_f, it follows that every language in NP is reducible to L_f. Since $L_f \in$ co-NP, this proves $NP \subseteq$ co-NP. Finally, $L \in$ co-NP implies co-L $\in NP \subseteq$ co-NP and thus $L \in NP$. Hence $NP =$ co-NP.