Q1a. False. By linear speedup theorem as $600n^5 + 600$ is bounded by $600 * (5n^5 + 5)$.

Q1b. False. As $H(n)$ is bounded by $\log\log\log n$, for large enough n. We have shown in class that $DSPACE(1) = DSPACE(\log\log\log n)$.

Q1c. True.

Consider any language L. $Size(C_0) \leq 1$.

For C_{n+1}, note that $L(x_1, x_2, \ldots, x_{n+1}) =$

(x_1 AND $L(1, x_2, \ldots, x_{n+1})$) OR

(($\neg x_1$) AND $L(0, x_2, \ldots, x_{n+1})$).

Thus, $Size(C_{n+1}) \leq 3 + Size(C_n)$

Thus, $Size(C_n) \leq 3n + 1$.

Q1d. True. Algorithm for this would just consider all possible sets U of at most five variables, and check if all the clauses are satisfied if only the variables in U are set to be true, and all other variables are set to be false. If any of these gives that all clauses are satisfied, then answer YES. Otherwise answer NO. It is polynomial time algorithm as the number of possibile sets U considered above is at most $O(n^5)$, where n is the number of variables.

Q2. Let

$T_1(0) = T_2(0) = 1$.

$T_1(2i + 1) = 2^{T_1(2i)}$, $T_1(2i + 2) = 2^{T_2(2i+2)}$,

$T_2(2i + 1) = 2^{T_1(2i+1)}$ and $T_2(2i + 2) = 2^{T_2(2i+1)}$.

Now, similar to time-hierarchy theorem, as $\lim_{i\to\infty} \frac{T_1(2i+1)*\log T_1(2i+1)}{T_2(2i+1)} = 0$, we have a language which is in $DTIME(T_2(\cdot)) - DTIME(T_1(\cdot))$. Similarly, as $\lim_{i\to\infty} \frac{T_2(2i+2)*\log T_2(2i+2)}{T_1(2i+2)} = 0$, we have a language which is in $DTIME(T_1(\cdot)) - DTIME(T_2(\cdot))$.

Q3. It is easy to see that the problem is in NP by just guessing the mapping h, and then verifying that it satisfies the property mentioned in QUESTION, for all $u \in V$, $h(u) = min(\mathbf{N} - \{h(w) : (u, w) \in E\})$.

To see NP-hardness, we reduce 3-SAT to the problem.

Suppose (U, C) is the 3-SAT instance where U is the set of variables and C is the set of clauses. Below is the construction of the instance of the given problem.

For each $u \in U$, form four vertices a_u, b_u, c_u, d_u, with edges $(a_u, b_u), (b_u, c_u), (c_u, d_u), (d_u, a_u)$. Let a_u represent u and b_u represent $\overline{u}$. There will not be any other edges with tail in the above vertices. Thus, the only possible way for h to do the mapping would be, $h(a_u) = h(c_u) = 1$ and $h(b_u) = h(d_u) = 0$ or $h(a_u) = h(c_u) = 0$ and $h(b_u) = h(d_u) = 1$. Intuitively, $h(a_u) = 1$ will represent that u is true.

For each clause $c \in C$, form three vertices e_c, f_c, g_c. Then, we have the edges $(e_c, f_c), (f_c, g_c), (g_c, e_c)$. Additionally, we have the edges, (e_c, a_u), if $\overline{u}$ is a literal in the clause c, and (e_c, b_u), if u is a

literal in the clause c.

To see that the above works, suppose the 3-SAT problem is satisfiable. Then let $h(a_u) = h(c_u) = 1$, $h(b_u) = h(d_u) = 0$ if u is true (and correspondingly $h(a_u) = h(c_u) = 0$, $h(b_u) = h(d_u) = 1$ if u is false). Let $h(e_c) = 2, h(g_c) = 0$ and $h(f_c) = 1$ for all c. Note that this is consistent as at least one of the tail ends of edges starting in e_c ends in a vertex with h value 0.

On the other hand, suppose we have a h as claimed for the instance. Then, we claim that having u to be true iff $h(a_u) = 1$ gives us a satisfying assignment. For this, note that g_c and f_c get h value 0 and 1 (or vice versa) as they have only one edge going out of them. Thus, e_c should get h value at least 2 (it will be exactly 2). But this implies that $h(g_c) = 0$ and $h(f_c) = 1$. But then at least one of the edges starting from e_c should end up in a vertex with h value 0, giving us the literal which is true.

Q4. (a) To show that the algorithm runs in polynomial time, it is sufficient to show that the number of elements in any X_j is bounded by a polynomial in n and $\log K$. To see this, note that each subsequent element added in X_j in $REDUCE(X'_j)$ has value ratio with the previous item added by at least $1/(1 - \delta/n)$. As the maxium value of the sets in X_j is at most K, the total number of items in X_j is bounded by $\frac{-\ln K}{\ln(1-\delta/n)}$, which is bounded by $\frac{n \ln K}{\delta}$.

(b) Let P_j denote the set of all subsets Y of $\{1, 2, \ldots, j\}$ such that $Val(Y) \leq K$.

By induction on j, we claim that, for each $Y \in P_j$, there exists a $Y' \in X_j$ such that $Val(Y) \geq Val(Y') \geq (1 - \delta/n)^j Val(Y)$.

Above is clearly true for $j = 0$.

Suppose this holds for $j = i$. Then, we show it for $j = i + 1$.

Let $Y \in P_{i+1}$ be given. Let $Z = Y \cap \{1, 2, \ldots, i\}$.

Let $Z' \in X_i$ be such that $Val(Z) \geq Val(Z') \geq (1 - \delta/n)^i Val(Z)$.

If $Y = Z$, then let $Z'' = Z'$. Otherwise, $Y = Z \cup \{i + 1\}$, and let $Z'' = Z' \cup \{i + 1\}$. Note that $Z'' \in X'_{i+1}$ as constructed in the algorithm, and $Val(Y) \geq Val(Z'') \geq (1 - \delta/n)^i Val(Y)$.

Now, if $Z'' \in X_{i+1}$, then we are done. Else, by construction there exists a $Y' \in X_{i+1}$ such that $Val(Z'') \geq Val(Y') \geq (1 - \delta/n) Val(Z'')$ (otherwise Z'' would have been placed in X_{i+1}). It follows that $Val(Y) \geq Val(Z'') \geq Val(Y') \geq (1 - \delta/n) Val(Z'') \geq Val(Y)(1 - \delta/n)(1 - \delta/n)^i$.

Thus, we have that the answer given by the algorithm has value at least $(1 - \delta/n)^n$ of optimal value. As $(1 - \delta/n)^n \geq (1 - \delta)$, we are done.

Q5.

First boost the probabilities so that the error probability is below $2^{-|x|}$, where x is the input.

Without loss of generality assume that V tosses all its coins in the beginning (as the coins are private). Let us call these coin tosses z, which is of length say $p(|x|)$.

Now, similar to the proof of $BPP \subseteq \Sigma_2^p$, one can show that there exists sequence $(y_1, y_2, \ldots, y_{q(|x|)})$, where y_i is of length $p(|x|)$ such that:

(a) If $x \in L$, then for all z, there exists a k such that the coin tosses $z \oplus y_k$ leads V to acceptance.

(b) If $x \notin L$, then for any sequence $(y'_1, y'_2, \ldots, y'_{q(|x|)})$: at most 0.4 fraction of the z satisfy that for some k, coin tosses $z \oplus y'_k$ leads to acceptance.

Thus, one can use the protocol that initially the prover provides $(y_1, y_2, \ldots, y_{q(|x|)})$ to the verifier. Then the verifier proceeds to toss the coins z, and does the original protocol for the coin tosses corresponding to $y_i \oplus z$, for $1 \leq i \leq q(|x|)$, and accepts iff at least one of these leads to acceptance.