Question 1

(b) and (c) are polytime.

Question 2

(a) and (d) are TRUE. (b) and (c) are FALSE.

Question 3

Let's call the version of *3SAT* in which each clause contains appearances of three distinct variables *DISTINCT-3SAT*. Obviously *DISTINCT-3SAT* is in NP. We give a reduction from *3SAT* to *DISTINCT-3SAT* that operates in polynomial time. Suppose the original Boolean formula has n distinct variables $x_1, x_2, ..., x_n$ and m clauses $C_1, C_2, ..., C_m$. The reduction transform ϕ to another Boolean formular ϕ' by the following rules:

1. If C_i contains three distinct variables, we add C_i to ϕ'.
2. If C_i contains a variable and the negation of the same variable, we do not add C_i to ϕ' because C_i is always TRUE.
3. If C_i is of the form $(x_a \vee x_a \vee x_b)$, we replace C_i by 2 new clauses $(x_a \vee x_b \vee y_i)$ and $(x_a \vee x_b \vee \overline{y_i})$ where y_i is a new variable. It is easy to see that $(x_a \vee x_a \vee x_b)$ is TRUE if and only if $(x_a \vee x_b \vee y_i) \wedge (x_b \vee x_b \vee \overline{y_i})$ is TRUE.
4. If C_i is of the form $(x_a \vee x_a \vee x_a)$, we replace C_i by 4 new clauses $(x_a \vee y_i \vee z_i)$, $(x_a \vee y_i \vee \overline{z_i})$, $(x_a \vee \overline{y_i} \vee z_i)$ and $(x_a \vee \overline{y_i} \vee \overline{z_i})$ where y_i and z_i are new variables. By the same argument as in the previous case, C_i is TRUE if and only if all the 4 new clauses are TRUE.

By the above construction, all the clauses in ϕ' contains three distinct variables. It is clear that ϕ‘ can be constructed from ϕ in polynomial time. Finally, it is easy to see that ϕ is satisfiable if and only if ϕ‘ is satisfiable. It completes the proof.

Question 4

Since the graph is unweighted, we can determine the shortest path from a to b by simply doing BFS starting from a. Then, we answer "yes" to the decision problem *SPATH* if and only if the length of the shortest path is at most k. BFS can be implemented in polynomial time easily.

It is clear that *LPATH* is in NP. To show that *LPATH* is NP-complete, you just have to observe that *UHAMPATH* is a special case of *LPATH* by substituing $k = n - 1$.

Question 5

We give a reduction from *CLIQUE* to *HALF-CLIQUE*. The reduction takes a graph G with n vertices and construct a graph G' with n' vertices. Graph G has a clique of size k if and only if the graph G' has a clique of size $\frac{n'}{2}$. We consider two cases.

1. If $k \leq \frac{n}{2}$, construct G' from G by adding $n - 2k$ universal vertices (a vertex is universal if it has an edge with every other vertex in the graph). So $n' = 2n - 2k$. If G has a clique C of size k, then G' has a clique C' with the vertices in C and all new universal vertices. And C' is of size $k + n - 2k = n - k$ which is a half clique in G'. If G' has a clique of size $n - k$, at most $n - 2k$ vertices are new and so G has a clique of size k.

2. If $k > \frac{n}{2}$, construct G' from G by adding $2k - n$ degree 0 vertices. So $n' = n + 2k - n = 2k$. If G has a clique C of size k, C is the half clique in G'. If G' has a half clique, since no degree 0 vertices are in C, C is a clique of size k in G.

Question 6

Suppose, without loss of generality, we use binary representation for all numbers in this question. By the assumption that P = NP, we know that $PRIME(x)$ is in P. Now consider another decision problem $PREFIX(x, y)$. $PREFIX(x, y)$ is the decision problem asking if a number x has a nontrivial factor x' with the bit string y as prefix. Clearly $PREFIX(x, y)$ is in NP. By the assumption that P = NP, $PREFIX(x, y)$ can be solved in polynomial time. We use the following algorithm to factorize x.

```
FACTOR(x)
{
        if (PRIME(x) == TRUE)
            return {1, x};   // only trivial factors
        y = "1";
loop:   if (PREFIX(x, y + "0") == TRUE)
            y = y + "0";    // y = y + "0" means add a 0 at the end of y
        else
            y = y + "1";
        if (x is divisible by y)
            return {y + FACTOR(x/y)};
        goto loop;
}
```

The function $FACTOR$(x) will return a list of factors of x. First, we ask the question if x is a prime number. If x is a prime number, then x has only two trivial factors 1 and x. If x is not a prime number, we search the smallest nontrivial factor by asking a number of decision problem $PREFIX(x, y)$. By asking one more question of $PREFIX(x, y)$, we know one more bit of the nontrivial factor. By asking at most n questions where n is the length of the binary string of x, we can find the smallest nontrivial factor y of x. And then we can repeat the procedure to find the list of factors of x/y. Each iteration takes polynomial time and there are at most polynomial number of iterations. So it is a polynomial time algorithm.

Question 7

By showing that a specific algorithm to solve the SAT problem requires exponential time does not imply all possible algorithms that solve the SAT problem require exponential time. For example, one can design an exponential time algorithm to sort a list of integers by generating all the permutations and test if it is sorted. However, it does not imply the sorting problem has exponential time complexity.