Answer four (4) questions from six (6). Time Allowed: 3 hours.

1.

2.

3. (a) $L = \{w : w \in \{a, b\}^*, w = a^n b^n, n \in \mathbb{N}\}$
   $L = \{w : w, v \in \{a, b\}^*, w = v w R\}$
   $L = \{w : w \text{ is a syntactically correct Java program}\}$

(b)

(c) $\Sigma^*$ is not finite, but is countable.

4.

5.

6. (a)

(b)

(c)

(d) $2^X$ will be countable exactly when $X$ is finite. $2^X$ will be finite exactly when $X$ is finite. ($2^X$ will either be uncountable or finite.)

7.

8.

9.

10. (a) $L = \{w \in \{a, b\}^* : w = a^n b^n a^n, n \in \mathbb{N}\}$ is r.e. but not c.f.

(b)

(c) $L = \{w \in \{t : t \text{ is a Turing machine}\} \times \{s : s \text{ is a state}\} : \text{where TM } t \text{ enters state } s\}$

(d)

11.

12.
13. **Note:** this solution is quite long in order to make it understandable. A far more concise proof would be more acceptable in an exam.

To prove that \textsc{halts} and \textsc{change} are equally difficult problems we need to show two reductions: \textsc{halts} \leq \textsc{change} proves that \textsc{change} is at least as hard as \textsc{halts} and \textsc{change} \leq \textsc{halts} proves that \textsc{halts} is at least as hard as \textsc{change}.

Assume we have a program \texttt{halts()} that takes a function \texttt{P()} and returns true if \texttt{P()} is a halting function and returns false otherwise. Assume also that we have a program \texttt{change()} that takes a function \texttt{Q()}, and a variable \texttt{a} declared in \texttt{Q()}, and returns true if \texttt{a} would be changed during the execution of \texttt{Q()}, and returns false otherwise.

First, we show \textsc{halts} \leq \textsc{change} (a reduction from \textsc{halts} to \textsc{change}). This reduction will prove that if we have a program to solve the latter, we can solve the former. Given a program \texttt{change()} we will solve the halting problem for a function \texttt{P()} as follows.

Construct a function \texttt{Q()} of the following form:

```c
void Q(void) {
    int a;
    P(); // call function P()
    a = 5;
}
```

If we have a program \texttt{change()}, we could execute it passing \texttt{Q()} and \texttt{a}. If \texttt{change(Q, a)} returns true then that means that \texttt{halts(P)} would have returned true and if \texttt{change(Q, a)} returns false that means that \texttt{halts(P)} would have returned false. More concisely we say that \texttt{change(Q, a)} returns true iff (if and only if) \texttt{halts(P)} would have returned true. Therefore we effectively have simulated the program \texttt{halts()}.

Secondly, we show a reduction from \textsc{change} to \textsc{halts}. This will prove that problem \textsc{halts} is at least as hard as problem \textsc{change}. Given a program \texttt{halts()} that solves \textsc{halts} we can simulate a program \texttt{change()} that solves \textsc{change} for a function \texttt{Q()} as follows.

Assume \texttt{Q()} contains \textit{N} statements or function calls and is of the form:

```c
void Q(void) {
    statement0;
    statement1;
    ...
    statementi;
    a = 5;
    statementi+2;
    ...
    statementN;
}
```

Let us extract the block of statements before the line where \texttt{a} is first changed (all statements from \texttt{statement0} to \texttt{statementi}). Let us put them into a function of their own and call it \texttt{P()}. \texttt{P()} would look like:
void P(void) {
    statement0;
    : 
    statementi;
}

Next we pass \( P() \) to halts(). halts(\( P \)) returns true iff \( \text{change}(Q, a) \) would have returned true. Therefore we have simulated \( \text{change}(Q, a) \).

This method works even if "\( a = 5; \)" is part of a conditional statement or part of a function call. For example, imagine if the function looked like:

void Q(void) {
    statement0;
    : 
    statementi;
    if (condition) {
        a = 5;
    } else {
        statementi+2;
    }
    statementi+3;
    : 
    statementN;
}

Then we would pass the following function

void P(void) {
    statement0;
    : 
    statementi;
    if (condition) {
        a = 5;
    } else {
        while (true) {
            // infinite loop
        }
    }
}

To halts(). In this case, halts(\( P \)) also returns true iff \( \text{change}(Q, a) \) would have returned true.

Finally, if there had been many statements in \( Q() \) that assign some value to \( a \), then we just repeat the procedure for each one. As long as the piece of code immediately before any one statement assigning a value to \( a \) halts, then \( a \) will be changed.

By showing that HALTS \( \leq \) CHANGE and that CHANGE \( \leq \) HALTS we prove that HALTS and CHANGE are equally difficult problems.

15.

16. (a) i. (1,1,1,1,1,1) : because a FA exists to accept such unary numbers
   ii. (1,1,1,1,1,1) : because a FA exists to accept such a language
iii. (0,0,1,0,0,0): (0,0,1,0,... because the halting problem can be reduced to it, and
....,0,0) because at least one EXPTIME-complete language is accepted by a TM
that goes into a state 01.

iv. (0,0,1,0,0,0): (0,0,1,0,... because the halting problem can be reduced to it, and
....,0,0) because at least one EXPTIME-complete language is accepted by a Java
program that executes a print statement.

v. (1,1,1,1,1,1): because a FA exists to accept such a language

vi. (1,1,1,1,1,1): because the following FA accepts such a language:

```
  a
  |
  v
b   b
  |
  v
a
```

(b)

17.

18. (a)

(b) To prove that the class $\mathcal{P}$ is closed under complement we must show that if a language
is in $\mathcal{P}$ then its complement must also be in $\mathcal{P}$. Consider a language $L$ that is in $\mathcal{P}$.
Since $L$ is in $\mathcal{P}$, some polynomial-time function $\text{boolean } f(\text{String } x)$ must
exist that returns true if $x \in L$ and returns false otherwise. Consider the following
function:

```java
boolean g(String x) {
    if (!f(x)) {
        return true;
    } else {
        return false;
    }
}
```

This function decides exactly the language $\overline{L}$, the complement of $L$. Moreover this
function is also polynomial so $\overline{L}$ must be in $\mathcal{P}$.

(c) To prove that SAT is in $\mathcal{NP}$ we must show that each instance of SAT has a wit-
ness (solution) that can be verified in polynomial time. We do this by outlining a
polynomial-time algorithm to verify any satisfying truth assignment for any instance
of SAT.

Given a conjunctive normal form with a list of $n$ clauses, each with at most $m$ literals
(boolean variables), from an alphabet of $a$ literals, and a truth assignment for each of
the $a$ literals (this is our witness), the algorithm could be defined as follows.

Stage 1. Take each clause, and iterate through the literals in the clause, replacing
each literal with a T or an F as appropriate from the list of truth assignments. If the
negation of a literal is encountered, replace it with the negation of the value from the
truth assignment. This substitution stage will take at most $(n \times m \times a)$ copy and
comparison operations (assuming the list of truth assignments is not sorted).
Stage 2. Iterate through the clauses, ensuring that each clause has at least one T in it. If a single clause does not have at least one T in it then the truth assignment was not valid. If the end of the list of clauses is reached then the truth assignment was a valid one. This checking stage requires at most $n \times m$ comparison operations.

The total time complexity is $nma + nm$, which can be approximated as a polynomial function $c^3 + c^2$ for some $c$ where $c \geq \max(m, n, a)$.

19. (a)

(b)

(c) $P = NP$ implies that if a language $L$ is in $P$ then $\overline{L}$ is also in $NP$. We also know that $P$ is closed under complement so that if a language $L$ is in $P$ then $\overline{L}$ is also in $P$. Therefore, if $L$ is in $NP$ then $\overline{L}$ is in $NP$. This is equivalent to saying that $NP$ is also closed under complement, or that $NP = coNP$. 