1. **Enunciare e discutere la tesi di Church-Turing, chiarendo perché viene formulata come tesi e non come teorema.**

Risposta: La tesi di Church Turing: "è calcolabile tutto ciò che può essere calcolato da una MdT". Si tratta di una tesi e non di un teorema perché non è possibile dimostrare l'esistenza di una macchina più potente di quella di Turing e i problemi incalcolabili con MdT non sono calcolabili nemmeno umanamente come ad esempio l'Halting Problem.

2. **Definire le classi P, NP e indicare le relazioni di contenimento fra le classi.**

Risposta: le classi rispettivamente P (polynomial) ed NP ( Nondeterministic polynomial) sono due importanti classi di complessità che rappresentano:
P l'insieme dei linguaggi per cui esiste una MdT deterministica che in tempo polinomiale decide se una stringa in input appartiene ai linguaggi,
NP l'insieme dei linguaggi per cui esiste una MdT non deterministica che in tempo polinomiale decide se una stringa in input appartiene ai linguaggi.
I problemi $P \subseteq NP$ (esempio, MdT è un caso particolare di MdTN) ma ad oggi non sappiamo dimostrare se vi sia equivalenza stretta o meno: ovvero se esista un problema NP che non appartenga a P. Questo problema resta tutt'ora irrisolto.
Ad esempio, sappiamo che SAT è un problema in NP ma non sappiamo se appartenga a P

3. **Definire la classe dei problemi NP-completi e indicare le relazioni di contenimento rispetto alle classi P e NP.**

Risposta: la classe NP-completo è l'insieme dei linguaggi L tali che appartengono a NP e ogni altro linguaggio di NP sia polinomialmente riducibile a quei linguaggi L.
Il teorema di Cook-Levin afferma che SAT è NP-completo.

4. **Indicare un possibile risultato che permetterebbe di stabilire che P ≠ NP**

Risposta: Se riuscissimo a dimostrare che un problema NP non appartiene anche a P potremmo stabilire la disuguaglianza tra P e NP. Ad esempio, sappiamo che il problema SAT è NP e se fossimo in grado di dimostrare che abbia tempo risolutivo con lowerbound esponenziale (e dunque che non ci sia un algoritmo polinomiale in grado di risolvere SAT) allora potremmo verificare la diseguaglianza.

5. **Indicare un possibile risultato che permetterebbe di stabilire che P = NP**

Risposta: Se riuscissimo a dimostrare che un problema NP appartiene anche a P potremmo stabilire l'uguaglianza tra P e NP. Ad esempio, sappiamo che il problema SAT è NP e se fossimo in grado di dimostrare che abbia tempo risolutivo con lowerbound polinomiale (e dunque che ci sia almeno un algoritmo polinomiale in grado di risolvere SAT) allora potremmo verificare l'uguaglianza.

6. **Per dimostrare che un problema A è NP-completo cosa è necessario dimostrare?**

Risposta: Un problema A è NP-completo se:

- $A \in NP$
- $\forall A' \in NP, A' \leq_p A$.

7. **Definire il problema della soddisfacibilità di una formula logica e cosa è una formula in forma normale congiuntiva (CNF). A quale classe di complessità temporale appartiene il problema della soddisfacibilità di una formula CNF?**

Risposta: Data una formula F del calcolo proposizionale il problema della soddisfacibilità consiste nel verificare l'esistenza di un'assegnazione di valori tali da rendere vera la formula. La CNF è una formula che consiste nella congiunzione della disgiunzione di valori:
$\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{k} l_{ij})$ e appartiene alla classe NP-completo.

8. **Definire il problema della soddisfacibilità di una formula logica e cosa è una formula in forma normale congiuntiva (DNF). A quale classe di complessità temporale appartiene il problema della soddisfacibilità di una formula DNF?**

Risposta: (vedi sopra) La DNF è una formula che consiste nella disgiunzione della congiunzione di valori:
$\bigvee_{i=1}^{n}(\bigwedge_{j=1}^{k} l_{ij})$ e appartiene alla classe NP-completo.

9. **Nel caso di problemi NP-completi non rinunciamo a cercare soluzioni. Illustrare un possibile approccio per il problema della soddisfacibilità di formule logiche**

Risposta: Un approccio per risolvere il problema della soddisfacibilità è tramite l'algoritmo DPLL. Si applica ad un insieme di clausole ed è basata sulla tecnica del backtracking. Infatti, prova ricorsivamente tutte le assegnazioni per vedere se soddisfa la formula.
Si compone di funzioni di controllo:

- UNIT PROPRAGATION (If Clausola unitaria -> rende vera la clausola)
- PURE LITERAL ELIMINATION (if letterale appare uguale e in tutte le clausole -> rende vero il letterale)

E di funzioni di appoggio:

- UNIT_PROPRAGATE(l,F): (if l uguale in ogni clausola -> if l positivo -> cancella clausola, else-> cancella letterale dalla clausola)
- PURE LITERAL ASSIGN(l,F): (cancella tutte le clausole che contengono l)
- CHOOSE_LITEREAL(F): seleziona casualmente una lettera in F

**10. Illustrare una riduzione – a vostra scelta - che mostra che un problema A è NP-completo**

Risposta: Sia A un problema 3SAT (e ricordando che SAT è NP-completo) è possibile dimostrare che, se SAT $\leq_p$ 3SAT, allora 3SAT è NP-completo. Data F una formula con k clausole possiamo definire F' tale che è soddisfacibile se e solo se F è soddisfacibile.
Ad esempio, siano n il numero dei letterali:
n=1) $F' = x \Leftrightarrow F = (x \vee y_0 \vee y_1) \wedge (x \vee y_0 \vee \neg y_1) \wedge (x \vee \neg y_0 \vee y_1) \wedge (x \vee \neg y_0 \vee \neg y_1)$
n=2) $F' = (x_1 \vee x_2) \Leftrightarrow F = (x_1 \vee x_2 \vee y_0) \wedge (x_1 \vee x_2 \vee \neg y_0)$
n>3) $F' = (x_1 \vee x_2 \vee \ldots) \Leftrightarrow F = (x_1 \vee x_2 \vee y_0) \wedge (y_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (x_n \vee x_{n-1} \vee \neg y_{n-4})$
la soddisfacibilità dei letterali al variare della loro cardinalità è costante per la soddisfacibilità delle formule quindi 3sat lo è.

11. **Per mostrare che un problema B è NP-difficile è sufficiente fornire una riduzione da un problema A a B. Quali proprietà deve verificare A e quali deve verificare la riduzione?**

Risposta: Dati due problemi A, B, B è NP-difficile se A $\in$NP-difficile e A $\leq_p$ B.

**12. Descrivere le fasi di analisi lessicale e di analisi sintattica di un compilatore; per ciascuna delle due fasi descrivi l'input e l'output e come viene elaborato il programma.**

Risposta: L'analisi lessicale di un compilatore prende in input il codice da elaborare e lo scompone in unità detti TOKEN che costituiscono l'output dell'analisi. I token vengono generati con l'utilizzo di espressioni regolari e di automi a stati finiti.
L'analisi sintattica prende in input i token generati dall'analisi lessicale e studia la struttura del programma e delle singole istruzioni e restituisce in output l'albero di derivazione tramite l'uso di parser di tipo TOP-DOWN o BUTTOM-UP che lavorano esclusivamente con grammatiche CF.

**13. Illustrare la gerarchia di Chomsky delle grammatiche; indicare per ciascuna categoria la proprietà/caratteristica a vostro giudizio maggiormente rilevante.**

Risposta: La gerarchia delle grammatiche di Chomsky si compone di 4 tipi di grammatiche:
TIPO 3: Sono grammatiche regolari in cui le produzioni sono del tipo $A \rightarrow aB$ oppure $A \rightarrow a$ con $A, B \in V_N$ e $a \in V$. Queste grammatiche generano linguaggi riconosciuti da ASF.
TIPO 2: Sono grammatiche Context Free in cui le produzioni sono del tipo $A \rightarrow a$ con $a \in (V \cup T)(V \cup T)^*$ e generano linguaggi riconosciuti da automi a pila non deterministici.
TIPO 1: Sono grammatiche contestuali del tipo $a \rightarrow b$ con $a \in (V \cup T)^*V(V \cup T)^*$, $b \in (V \cup T)(V \cup T)^*$ e $|b| \geq |a|$ e generano linguaggi riconosciuti da MdT lineari.
TIPO 0: Sono grammatiche non limitate dalla lunghezza delle produzioni e generano linguaggi riconosciuti da MdT.

**14. Descrivi l'algoritmo per eliminare le ricorsioni sinistre di una grammatica di tipo 2**

Risposta: l'algoritmo che elimina le ricorsioni sinistre di produzioni immediate del tipo $A \rightarrow Aa$ si realizza in una serie di operazioni:

1. Separa le ricorsioni sinistre dalle altre ($A \rightarrow Aa_1$, $A \rightarrow Aa_2$, $A \rightarrow Aa_3$ … $A \rightarrow b_1$, $A \rightarrow b_2$, $A \rightarrow b_{3\ldots}$)
2. Introduciamo A' e sostituiamo le produzioni non ricorsive ($A \rightarrow b_1$ diventa $A \rightarrow b_1A'$ ecc.)
3. Sostituiamo le ricorsioni sinistre $A \rightarrow Aa_i$ con $A' \rightarrow a_iA'$
4. Aggiungiamo la produzione $A \rightarrow \lambda$

**15. Descrivere un analizzatore sintattico di tipo top-down**

Risposta: L'analisi TOP-DOWN costruisce in pre-ordine un albero di derivazione a partire dalla radice. Sceglie un non terminale ed applica ricorsivamente a sinistra. Se terminando con successo tutte le produzioni le foglie dell'albero rappresentano la stringa dell'input termina con SUCCESSO. Altrimenti termina con FALLIMENTO.

**16. Descrivere un analizzatore sintattico di tipo bottom-up**

Risposta: L'analisi BOTTOM-UP costruisce in post-ordine un albero di derivazione destro di un input x a partire dalle foglie, usando le produzioni della relativa grammatica G. Il metodo che utilizza è detto SHIFT-REDUCE e viene applicato dai parer LR sull'input in questo modo:
È basato su una pila inizialmente vuota, il parser legge l'input un token alla volta eseguendo ad ogni lettura SHIFT per spostare in pila il prossimo token o REDUCE per derivare una produzione. Per non incorrere in errori del tipo REDUCE-REDUCE ( quando è possibile eseguire reduce con più produzioni differenti) o SHIFT-REDUCE (quando è possibile eseguire indifferentemente le azioni), la decisione è condizionata dallo stato in cui si trova in quel momento la pila.

**17. Quali sono le proprietà di una grammatica LL(1) e per quale ragione sono la migliore scelta per descrivere i linguaggi di programmazione.**

Risposta: definita SELECT(A→a) la funzione tale che = First(a)∪Follow(a) se a è annullabile, altrimenti = First(a), una grammatica CF si dice LL(1) se per ogni coppia di produzioni A→a, A→b vale SELECT(A → a) ∩ SELECT(A → b) = Ø.
Le grammatiche LL(1) sono particolarmente interessanti per descrivere i linguaggi di programmazione perché sono semplici da formulare e molto efficienti.

**18. Data una Grammatica regolare G, è possibile stabilire se G genera il linguaggio vuoto? Come?**

Risposta: Sapendo che per una grammatica regolare G esiste un ASF che riconosce tutte e sole le stringhe che appartengono a G è possibile dimostrare che la grammatica generi il linguaggio vuoto. Se l'ASF non avesse stato finale potremmo subito concludere la dimostrazione quindi prendiamo ad esempio che l'ASF abbia k stati stato k-esimo finale: se tutte le stringhe di G di lunghezza < k non vengono riconosciute dall'ASF possiamo concludere che G abbia generato il linguaggio vuoto senza andare a studiare anche le stringhe con lunghezza > di k per le quali, di fatto, si sarebbe dovuto verificare il pumping lemma.

**19. Data una Grammatica regolare G, è possibile stabilire se G genera un linguaggio finito o infinito? Come?**

Risposta: Sapendo che per una grammatica regolare G esiste un ASF che riconosce tutte e sole le stringhe che appartengono a G è possibile dimostrare che la grammatica generi il linguaggio finito o infinito. Se l'ASF della grammatica è privo di cicli possiamo affermare che genera solo linguaggi finiti. Altrimenti è sufficiente che almeno uno stato abbia almeno un ciclo per far valere il pumping lemma e considerare il linguaggio infinito.

**20. definire la classe dei linguaggi LR e LR(0).**

Risposta: Definiamo il parser LR come il metodo che scansiona Left to right l'input e che costruisce la derivazione da destra (Rightmost). Viene detto anche parser shift-reduce perché si usa per costruire l'albero di derivazione di grammatiche CF col metodo BOTTOM-UP. I parser LR fanno uso di stati da inserire in una pila per rappresentare il contesto sinistro corrente in cui si trovano. Lo stato permette di decidere se effettuare una Shift o una Reduce insieme all'uso di due tavole: action e goto. Lo stato è l'insieme risultante dall'operazione di chiusura di un set di item dove per item intendiamo una produzione soggetta alla notazione punto che permette di capire cosa è stato letto e cosa deve ancora leggere il parser di tale produzione.

Una grammatica è detta LR(0) se è possibile prendere la decisone Shift/reduce senza guardare l'input (infatti lookhead è nullo). Ciò è possibile solo se è presente al massimo una produzione col punto finale e non sono contemporaneamente presenti una produzione col punto finale e l'altra senza il punto finale (quindi assenza di conflitti reduce-reduce e shift-reduce)

.