I can provide all the answers, but I cannot create a PDF file. I can, however, format the answers in a clear, organized way that you can easily copy and paste into a document to create your own PDF. I will use tables where they are most helpful for clarity.

---

1. Prove the vertex cover problem is NP-Complete. (6 Marks)

The Vertex Cover problem is proven to be

NP-complete by satisfying two conditions[1]:

1. **It is in NP:** Given a graph $G=(V,E)$ and a positive integer k, we can check if a given vertex set V' is a vertex cover of size at most k in polynomial time[2]. We simply verify that

   $|V'|$[citestart]$\le k$ and then iterate through every edge $(u,v)\in E$ to confirm that either $u\in V'$ or $v\in V'$[3]. This process is efficient.

2. **It is NP-hard:** We can demonstrate this by reducing a known NP-complete problem, such as the **3-SAT problem**, to the Vertex Cover problem in polynomial time[4]. The reduction involves creating a graph from a 3-SAT formula such that a vertex cover of a certain size exists if and only if the formula is satisfiable[5]. Since 3-SAT is a well-established NP-complete problem, this reduction proves that Vertex Cover is at least as hard, making it NP-hard[6].

---

2. Differentiate between (any two) (4 Marks)

I'll provide a differentiation between

P class and NP class problems and **Deterministic and Non-deterministic algorithms**, as requested[7].

P class vs. NP class problems

Feature		**P class** [8]		**NP class** [9]
Solvability	Problems that can be solved by a	**deterministic** algorithm in **polynomial time**[10].	Problems where a given solution can be	**verified** by a **deterministic** algorithm in **polynomial time**[11].
Tractability	Considered "tractable" or "efficiently solvable"[12].	May or may not be "tractable"; finding a solution can be very difficult[13].		
Examples	Sorting, finding the shortest path in a graph[14].	Traveling Salesperson Problem, satisfiability problems[15].		

Deterministic vs. Non-deterministic Algorithms

Feature		**Deterministic Algorithms** [16]		**Non-deterministic Algorithms** [17]
Execution	The next step is always uniquely	The algorithm can "guess" and make a		

	determined by the current state[18].	choice from a set of possible next steps[19].
Output	Always produces the same output for a given input[20].	Not a practical model for real computers, but useful for defining complexity classes[21].
Role	Used for most practical computational tasks[22].	A theoretical model used to define complexity classes like NP[23].

---

3. Show that Hamiltonian problem is NP hard (4 or 6 Marks)

The

Hamiltonian Cycle problem is proven to be **NP-hard** by demonstrating that any NP-complete problem can be reduced to it in polynomial time[24]. A common way to show this is by performing a polynomial-time reduction from the

3-SAT problem to the Hamiltonian Cycle problem[25]. This reduction involves constructing a graph based on a 3-SAT formula[26]. The graph is carefully designed to contain a Hamiltonian cycle if and only if the original 3-SAT formula is satisfiable[27]. Because 3-SAT is a known NP-complete problem, this reduction proves that the Hamiltonian Cycle problem is at least as hard as 3-SAT, which is the definition of being NP-hard[28].

---

4. What is satisfiability problem? (4 Marks)

The

Satisfiability problem, or **SAT**, is a decision problem for boolean formulas[29]. It asks whether there is an assignment of boolean values (true or false) to the variables in a given boolean formula such that the entire formula evaluates to

true[30]. SAT is a foundational problem in computer science and was the first problem proven to be

NP-complete[31].

---

5. What is SAT and 3-SAT problem? Prove that 3-SAT problem is NP complete. (2 Marks)

- **SAT (Satisfiability):** The general problem of determining if a given boolean formula has a satisfying truth assignment[32].
- **3-SAT:** A special case of SAT where the boolean formula is in Conjunctive Normal Form (CNF) and each clause contains exactly three literals[33].

To prove that

3-SAT is NP-complete, we must show it is both in NP and is NP-hard[34].

1. **3-SAT is in NP:** Given a 3-SAT formula, we can verify a potential solution (a truth assignment) by substituting the values into the formula and checking if it evaluates to true[35]. This verification can be done in polynomial time[36].
2. **3-SAT is NP-hard:** This is proven by a polynomial-time reduction from the general SAT problem to 3-SAT[37]. This reduction shows that 3-SAT is at least as hard as SAT, which is a known NP-complete problem[38]. Therefore, 3-SAT is also NP-complete[39].

---

6. Define P, NP, NP hard and NP-Complete (6 or 8 Marks)

Class	Definition	
P (Polynomial Time)	The class of decision problems that can be solved by a	**deterministic** algorithm in **polynomial time**[40].
NP (Nondeterministic Polynomial Time)	The class of decision problems where a "yes" answer can be	**verified** in **polynomial time**[41].
NP-hard	A problem is NP-hard if any problem in the NP class can be	**reduced** to it in **polynomial time**[42]. These problems are at least as difficult as any NP problem[43].
NP-Complete	A problem that is both in	**NP** and is **NP-hard**[44]. These problems represent the "hardest" problems in the NP class[45].

---

7. State and explain NP hard, Hamiltonian cycle, Vertex cover problem (4 Marks)

- **NP-hard:** A problem is NP-hard if any problem in the NP complexity class can be polynomially reduced to it[46]. This means they are at least as computationally difficult as any NP problem.
- **Hamiltonian Cycle Problem:** This is a decision problem that asks whether a given undirected graph contains a **Hamiltonian cycle**—a simple cycle that visits every vertex exactly once[47].

- **Vertex Cover Problem:** This is a decision problem that asks if a given graph has a **vertex cover** of size at most k[48]. A vertex cover is a subset of vertices where every edge is incident to at least one vertex in the subset[49].

---

8. What is NP complete problem? Explain steps to prove that problem is NP-complete. (6 Marks)

An

NP-complete problem is a decision problem that lies at the intersection of the NP and NP-hard complexity classes[50]. This means it can be verified in polynomial time, and it is also at least as hard as any other NP problem[51].

To prove that a problem, let's call it

L, is **NP-complete**, you must follow a two-step process[52]:

1. **Show that L is in NP:** You must demonstrate that a potential solution to problem L can be verified in **polynomial time**[53].
2. **Show that L is NP-hard:** You must choose a problem L′ that is already known to be NP-complete and show that L′ can be **polynomially reduced** to L[54]. This reduction must transform any instance of

   L′ into an equivalent instance of L in polynomial time[55].

---

9. Explain vertex cover problem with example. (4 or 6 Marks)

The

Vertex Cover problem asks for a subset of vertices in a graph such that every edge has at least one endpoint in the subset[56]. The goal is often to find a vertex cover of the minimum possible size.

Example:
Consider the following graph:
.
The edges are: {(1,2), (1,3), (2,4), (3,5), (4,5)}[57].

We want to find a vertex cover. If we choose the set {1, 5}, let's check if it covers all edges:

- (1,2) is covered by vertex 1.
- (1,3) is covered by vertex 1.
- (2,4) is covered by vertex 4... wait, that's not right. Let's restart the check.
- Let's pick the set of vertices {1, 4, 5}.
- Edge (1,2): Covered by 1.
- Edge (1,3): Covered by 1.
- Edge (2,4): Covered by 4.
- Edge (3,5): Covered by 5.
- Edge (4,5): Covered by 4 and 5.
- So, {1, 4, 5} is a vertex cover.
  However, a smaller vertex cover is possible. The set {1, 5} covers all edges.
- Edge (1,2) is covered by 1.
- Edge (1,3) is covered by 1.
- Edge (2,4) is not covered by either 1 or 5.
- Let's try another set, {2, 3, 4}.
- Edge (1,2): Covered by 2.
- Edge (1,3): Covered by 3.
- Edge (2,4): Covered by 2 and 4.
- Edge (3,5): Covered by 3.
- Edge (4,5): Covered by 4.
- So, {2, 3, 4} is a vertex cover of size 3.

It can be shown that the minimum vertex cover size for this graph is 2. The set **{1, 5}** is not a vertex cover because the edge (2,4) is not covered. However, the set **{2, 3, 5}** is a vertex cover of size 3. The minimum vertex cover is a problem of finding the smallest possible such set, which is **NP-hard**.

---

10. Explain polynomial reduction problem. (4 Marks)

A

polynomial reduction is a method used to prove the relative hardness of problems[58]. If a problem

A can be transformed into a problem B in polynomial time, we say that A is polynomially reducible to B[59]. This transformation must ensure that solving the instance of

B allows us to solve the original instance of A[60]. This technique is central to complexity theory because if we can reduce a known NP-complete problem to a new problem, we prove that the new problem is also

NP-hard[61].

---

11. Define Asymptotic Notations. Explain their significance in analyzing algorithms. (6 Marks)

Asymptotic notations are mathematical tools used to describe the limiting behavior of an algorithm's running time or space requirements as the input size grows infinitely large[62]. They are essential for classifying and comparing the efficiency of algorithms[63].

- **Big O (O):** Provides an **upper bound** on the growth rate, representing the worst-case scenario[64].
- **Omega (Ω):** Provides a **lower bound** on the growth rate, representing the best-case scenario[65].
- **Theta (Θ):** Provides a **tight bound**, describing the exact asymptotic behavior[66].

Their significance lies in providing a machine-independent way to analyze an algorithm's performance[67]. This allows us to predict how an algorithm will scale with larger inputs and to choose the most efficient algorithm for a given task[68].

---

12. Explain deterministic and non-deterministic algorithms. (4 or 6 Marks)

(This is a rephrased version of a previous answer.)

	Deterministic Algorithms	Non-deterministic Algorithms
Description	For a given input, the algorithm follows a single, predictable sequence of steps and always produces the same output[69].	A theoretical model that can make "lucky guesses" to arrive at a solution[70].
Application	The vast majority of algorithms used in practice are deterministic[71].	Primarily used as a theoretical concept to define complexity classes like NP[72].
Examples	Standard sorting algorithms like Merge Sort[73].	Algorithms for solving NP problems by guessing a solution and then verifying it[74].

---

13. Explain class NP Hard. Differentiate between NP hard and NP complete algorithms. (6 or 8 Marks)

- **Class NP-hard:** A problem is in the NP-hard class if every problem in the NP class can be polynomially reduced to it[75]. This implies that NP-hard problems are at least as computationally difficult as any problem in NP[76].
- **Difference between NP-hard and NP-complete:**

Feature	**NP-hard problems**	**NP-complete problems**
Class Membership	Are not necessarily in the NP class (e.g., the Halting Problem)[77].	Must be in the NP class[78].
Relationship	The NP-hard class contains the NP-complete class[79].	Are a subset of the NP-hard problems that are also in NP[80].
Solvability	May or may not have solutions that can be verified in polynomial time[81].	Solutions can always be verified in polynomial time[82].

---

14. Define Big O, omega and Theta notations. (4 Marks)

(This is a concise rephrasing of a previous answer).

- **Big O (O):** Defines the **upper bound** of an algorithm's runtime, indicating its worst-case performance[83].
- **Omega (Ω):** Defines the **lower bound**, indicating the best-case performance[84].
- **Theta (Θ):** Defines the **tight bound**, providing a precise description of the algorithm's average-case performance[85].

---

15. State whether the following functions are CORRECT and INCORRECT and justify your answer. (6 Marks)

- **i) 3n+2=O(n)**
  - **CORRECT.** As n grows, the linear term 3n dominates the function[86]. We can find constants

$c=4$ and $n0=2$ such that for all $n\geq 2$, $3n+2\leq 4n$.

- **ii) 100n+6=O(n)**
    - **CORRECT.** The dominant term is 100n[87]. We can choose

      $c=101$ and $n0=6$ such that for all $n\geq 6$, $100n+6\leq 101n$.
- **iii) 10n2+4n+2=O(n2)**
    - **CORRECT.** The quadratic term 10n2 determines the growth rate[88]. We can find constants

      $c=11$ and $n0=1$ such that for all $n\geq 1$, $10n2+4n+2\leq 11n2$.

---

16. What are different time complexities? (4 Marks)

Time complexity measures how the runtime of an algorithm scales with the input size[89]. Common types of time complexities, expressed in Big O notation, include:

- **O(1) - Constant:** The runtime does not change with the input size.
- **O(logn) - Logarithmic:** The runtime grows slowly as the input size increases (e.g., binary search).
- **O(n) - Linear:** The runtime grows in direct proportion to the input size (e.g., searching an unsorted array).
- **O(nlogn) - Log-linear:** The runtime is highly efficient for sorting (e.g., Merge Sort).
- **O(n2) - Quadratic:** The runtime grows as the square of the input size (e.g., nested loops).
- **O(2n) - Exponential:** The runtime doubles with each addition to the input size, making it impractical for large inputs.

---

17. What is meant by Best case, average case and Worst case? (6 Marks)

These terms describe different scenarios for an algorithm's performance[90]:

- **Worst-Case:** The maximum time or space an algorithm might take for any input of a given size[91]. It provides a guaranteed upper bound on the performance[92].

- **Best-Case:** The minimum time or space an algorithm takes, corresponding to the most favorable input[93]. This is rarely a useful measure of performance[94].
- **Average-Case:** The expected performance of an algorithm over all possible inputs of a given size[95]. This analysis can be complex but provides a more realistic measure of performance[96].