🔹 Short Notes on Complexity Theory Topics (Plain Text Version)

---

1. Complexity Classes

Complexity classes group problems based on how much time and space (memory) they need.

Examples:

P: Problems solved in polynomial time.

NP: Problems whose solutions can be verified in polynomial time.

PSPACE: Problems solvable using polynomial memory.

EXP: Problems solvable in exponential time.

---

2. Time Complexity

Measures how long an algorithm takes to run depending on input size n.

Expressed as Big-O notation:

Example: Linear search → O(n), Binary search → O(log n).

Helps compare algorithms and predict performance.

---

3. Space Complexity

Measures the total memory required by an algorithm to execute completely.

Includes input, output, and extra space for computation.

Example: Recursive programs may use extra stack memory.

---

4. Class P (Polynomial Time)

The class P includes all problems that can be solved efficiently by a normal (deterministic) computer in time proportional to a polynomial function of input size.

Examples:

Sorting (Merge Sort, Quick Sort)

Shortest Path (Dijkstra)

Matrix Multiplication

---

5. Class NP (Nondeterministic Polynomial Time)

The class NP includes problems for which a given solution can be verified quickly (in polynomial time).

These may not be easy to solve, but easy to check once a solution is guessed.

Examples:

Traveling Salesman Problem

Subset Sum

Satisfiability (SAT)

It is known that P is contained in NP, but we do not know if P equals NP (the famous P vs NP question).

---

6. Polynomial-Time Reductions

A reduction converts one problem (A) into another (B) using a polynomial-time function.

If A can be reduced to B and B is easy to solve, then A is also easy.

Reductions are used to prove that problems are NP-hard or NP-complete.

---

7. NP-Completeness

A problem is NP-complete if it is:

1. In NP (its solutions can be verified quickly), and

2. Every problem in NP can be converted to it using a polynomial-time reduction.

NP-complete problems are the hardest problems in NP.

Examples: SAT, Vertex Cover, Hamiltonian Cycle, Subset Sum.

If any NP-complete problem is solved efficiently, then every NP problem becomes solvable efficiently.

---

8. Cook–Levin Theorem

The first formal proof that an NP-complete problem exists.

It states that the Boolean Satisfiability (SAT) problem is NP-complete.

This theorem (proved by Stephen Cook in 1971 and Leonid Levin in 1973) started the field of NP-completeness.

Other NP-complete problems are proved by reducing SAT to them.

---

9. Vertex Cover Problem

Given a graph and a number k, decide whether there exists a set of at most k vertices such that every edge touches at least one of these vertices.

Used in network protection or facility placement problems.

NP-complete – proved by reducing from Clique or SAT.

Can be approximated but not solved exactly in polynomial time.

---

10. Hamiltonian Path Problem

Ask whether there exists a path in a graph that visits every vertex exactly once.

If the path returns to the starting point, it is a Hamiltonian Cycle.

Used in routing, scheduling, and games.

It is an NP-complete problem (very hard to solve for large graphs).

---

11. Subset Sum Problem

Given a set of numbers and a target sum S, check whether any subset adds up exactly to S.

Example: For {3, 34, 4, 12, 5, 2}, can we form 9? → Yes, using {4, 5}.

Important in cryptography and resource allocation.

NP-complete problem.

---

12. Hierarchy Theorems

These theorems show that giving an algorithm more time or more space allows it to solve strictly more problems.

Time Hierarchy Theorem: More time $\Rightarrow$ more computational power.

Space Hierarchy Theorem: More space $\Rightarrow$ more computational power.

These imply that P is smaller than EXP ($P \subset EXP$) and L is smaller than PSPACE.

---

13. Circuit Complexity

Studies how large or deep a Boolean circuit (a network of logic gates) must be to compute a given function.

The size = number of gates; depth = number of layers.

Used in hardware design, cryptography, and proving computational lower bounds.

Example: Parity and Majority functions are studied for their circuit complexity.
