Complexity Theory and Computability Concepts

1. Halting Problem

The **Halting Problem** determines whether a given program will halt or run indefinitely for a given input. Alan Turing proved that no algorithm can solve this problem for all possible programs. It is a classic example of an undecidable problem. The result shows fundamental limits of computation. The halting problem is not solvable by any Turing machine. **Examples:**

- Determining termination of arbitrary programs
- Infinite loop detection

2. Hamiltonian Cycle Problem

The **Hamiltonian Cycle problem** asks whether a given graph contains a cycle that visits each vertex exactly once and returns to the starting vertex. It is a decision problem in graph theory. The problem belongs to NP because a given cycle can be verified in polynomial time. It is also NP-Complete. **Examples:**

- Travelling Salesman Problem (decision version)
- Route planning problems

3. Undecidability

A problem is said to be **undecidable** if no algorithm exists that can solve the problem correctly for all possible inputs in finite time. Such problems cannot be decided by any Turing machine. Undecidable problems highlight the theoretical limits of algorithms. **Examples:**

- Halting Problem
- Post Correspondence Problem
- Entscheidungsproblem

4. Lower Bound Arguments

A **lower bound** specifies the minimum time or space required by any algorithm to solve a problem. It helps prove that an algorithm is optimal. Lower bounds are independent of implementation. They are expressed using asymptotic notation. **Examples:**

- Sorting: $\Omega(n \log n)$
- Linear search: $\Omega(n)$
- Finding maximum element: $\Omega(n)$

5. Class P

Class **P** consists of decision problems that can be solved in polynomial time using a deterministic Turing machine. These problems are efficiently solvable in practice. Polynomial time includes $O(n)$, $O(n^2)$, etc. **Examples:**

- Binary Search
- Minimum Spanning Tree
- Shortest Path Problem

6. Class NP

Class **NP** contains decision problems whose solutions can be verified in polynomial time. The solution itself may not be computable efficiently. NP problems play a central role in computational complexity theory. **Examples:**

- SAT Problem
- Hamiltonian Cycle
- 0/1 Knapsack (decision version)

7. P vs NP

The **P vs NP** problem asks whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time. It remains an open problem in computer science. Solving it would have major implications in cryptography and optimization. **Facts:**

- $\mathrm{P} \subseteq \mathrm{NP}$
- Unknown whether $\mathrm{P} = \mathrm{NP}$
- Millennium Prize Problem

8. NP-Hard

A problem is **NP-Hard** if it is at least as hard as the hardest problems in NP. NP-Hard problems may not be decision problems. They may not belong to NP. Solving an NP-Hard problem efficiently would solve all NP problems. **Examples:**

- Travelling Salesman Problem (optimization)
- Halting Problem
- Scheduling problems

9. NP-Complete

A problem is **NP-Complete** if it is both in NP and NP-Hard. These problems are the hardest in NP. If any NP-Complete problem is solved in polynomial time, then P = NP. **Examples:**

- SAT
- 3-SAT
- Hamiltonian Cycle Problem

NP (Nondeterministic Polynomial Time)

The class **NP** contains decision problems for which a given solution can be verified in polynomial time using a deterministic Turing machine. These problems may not be solvable in polynomial time. Using a nondeterministic Turing machine, they can be solved in polynomial time. NP problems are central to computational complexity theory.

Examples:

- 0/1 Knapsack (decision version)
- Hamiltonian Cycle
- Boolean Satisfiability Problem (SAT)

Key Point: Every problem in P is also in NP, i.e., $P \subseteq NP$.