**Chapter 6: DECIDABILITY & COMPUTATIONAL COMPLEXITY**

**Topic – 1: Summary**

* **Decision** & **Optimization Problems**
* **Polynomial Time Algorithms** (**Tractable**)
* **Decidable** v/s **Undecidable Problems**
* **Class P** and **NP**
* **Polynomial-Time Reducibility**
* **NP-Hardness** & **NP-Completeness**
* Proving **NP-Completeness**

**Topic – 2: Decision & Optimization Problems**

* **Solvable problem:** We see if a **theoretically** solvable problem is solvable in **practical sense** or not.
* **Decision problems:** These are problems with their answer in form of **Yes** or **No**.
* **Optimization problems:** Problems where we try to **maximize** or **minimize** a value.
* We introduce a parameter **'k'** & use it as a **benchmark** to compare optimizations.

**Topic – 3: Polynomial Time Algorithms (Tractable)**

**P-Class Algorithms**

* Most **searching**, **sorting** & **path finding** algorithms we have studies so far take **O(nk)** time to be done.
* We call these as **polynomial-time algorithms**.
* These are generalized under **P-class**.

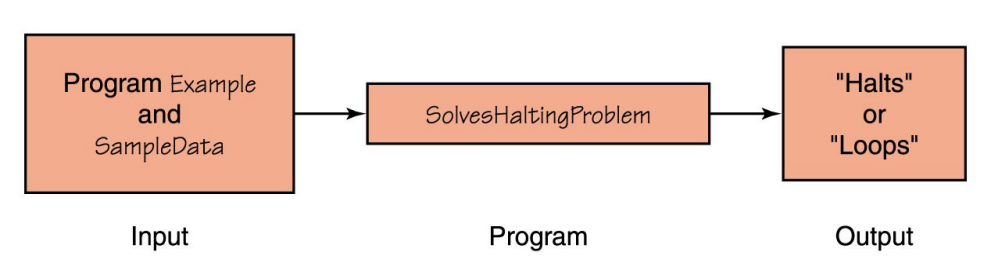
**Topic – 3: Decidable v/s Undecidable Problems**

**Undecidable Problems**

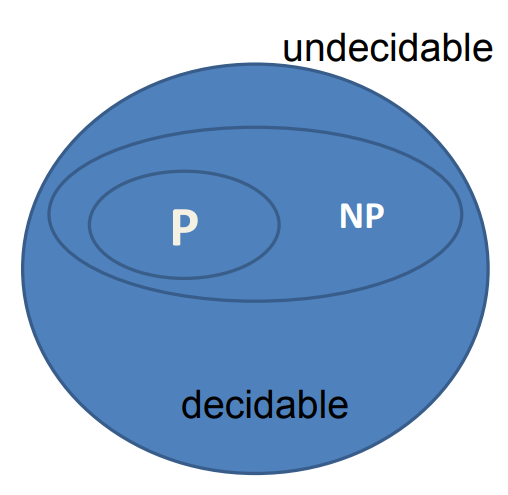
* Undecidable problems **can’t** be solved at all.
* For example, the ***halting problem***.

**Halting Problem**

* This problem is about **determining in advance** if an algorithm will terminate in future or not.



**NP-Class Algorithms**



* **NP** stands for **non-deterministic polynomial** problem.
* **NP-class** of problems are the ones which take **more** than **polynomial time** to be solved.
* Are **non-deterministic** by algorithms.
* No one has yet proven if **N** is **equal** to **NP**.

***// Deterministic algorithm***

***if (x<5) {***

***/\* Statements \*/***

***}***

***// Non-deterministic algorithm***

***if (\*) {***

***/\* Statements \*/***

***}***

**Certificate**

* We take input **'x'** and **'y'** is called ***certificate***.
* Certificate **'y'** is used to store some relevant values to input **'x'**.
* Also, **certificate** is used to **verify** the input.

**Hamiltonian Cycle Problem**

* Is an **NP-class** problem.
* Finding a **cycle** in graph which covers all vertices.
* Certificate here contains the **whole path** with it.
* We can verify it in **O(nk)**.

**Note!**

**🡪 A problem solvable in polynomial time must also verifiable in polynomial time (P ⊆ NP).**

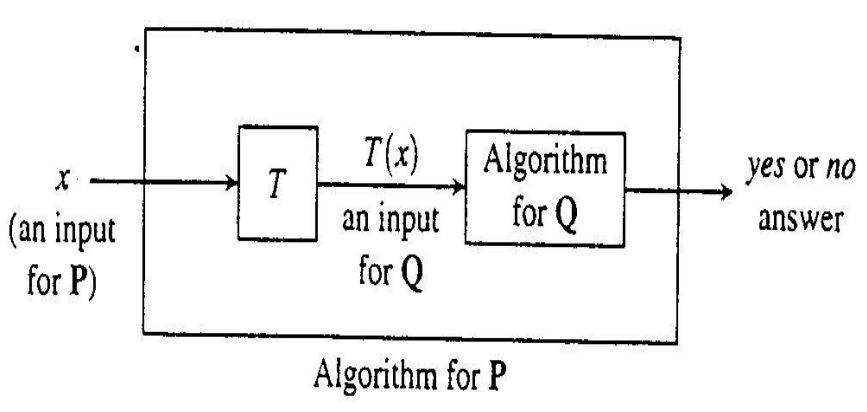
**NPC Class**

* **NPC** stands for **NP-complete** class.

**P ⊆ NP ⊆ NPC**

**NP Hard Algorithms**

* **NP-hard** is always an **NP** problem.
* **P** problems are **reducible** to another form **Q**.
* This transformation occurs through a **function**, refer to the example below.



* **T** above can be computed in **polynomial bounded time**.
* We basically transform inputs of **P** into inputs of **Q**.

**Topic – 4: Travelling Salesman Problem (TSP)**

* We are given a graph with **each edge** having a value.
* We have to find a way which sums up to **least travelling cost**.
* **Travelling cost** is calculated by **summing** the cost of each edge salesman travels through.
* Our goal is to **cover all vertices** with **least travelling cost**.

