7. COMPUTATIONAL COMPLEXITY THEORY

Theory of computation

Er. Shiva Ram Dam
Assistant Professor, Pokhara University

Syllabus:

7.1 Computable Languages and functions

7.2 Class P and class NP problems

7.3 NP-complete problems

7.1 Computational complexity theory

- Branch of TOC, **focuses on classifying computational problems according to their inherent difficulty.**
- **Deals with resources required** during the computation **to solve a given problem.**
- Common resources dealt are: **time and space**
- Also deal with decision problem.
- A decision problem is a problem where the answer is always YES/NO.

7.2 Computability Theory

- **Deals with only with whether a problem can be solved at all, regardless of the resource required.**
- **Does not deal with the space and time factor** of a machine.
- Is a **problem solvable** at all on the computer?
- Addresses **four main questions**:
  1. What problems can TM solve?
  2. What other systems are equivalent to TMs?
  3. What problems require more powerful machines?
  4. What problems can be solved by less powerful machines?

7.3 Space and Time complexity

- **Two** most common measures of complexity.
  1. Time (how many steps it takes to solve a problem)
  2. Space(how much memory it takes)
- Sometimes, there are **more than one way** to solve a problem.
- We need to learn how to compare the performance different algorithms and **choose the best one to solve** a particular problem.
- While analyzing an algorithm, **we mostly consider time complexity and space complexity**.
- Time complexity of an algorithm quantifies **the amount of time taken by an algorithm to run** as a function of the length of the input.
- Similarly, Space complexity of an algorithm quantifies the **amount of space or memory** taken by an algorithm to run as a function of the length of the input.
- Other resources may be: **how many parallel processors are needed** to solve a problem in parallel

Space Complexity

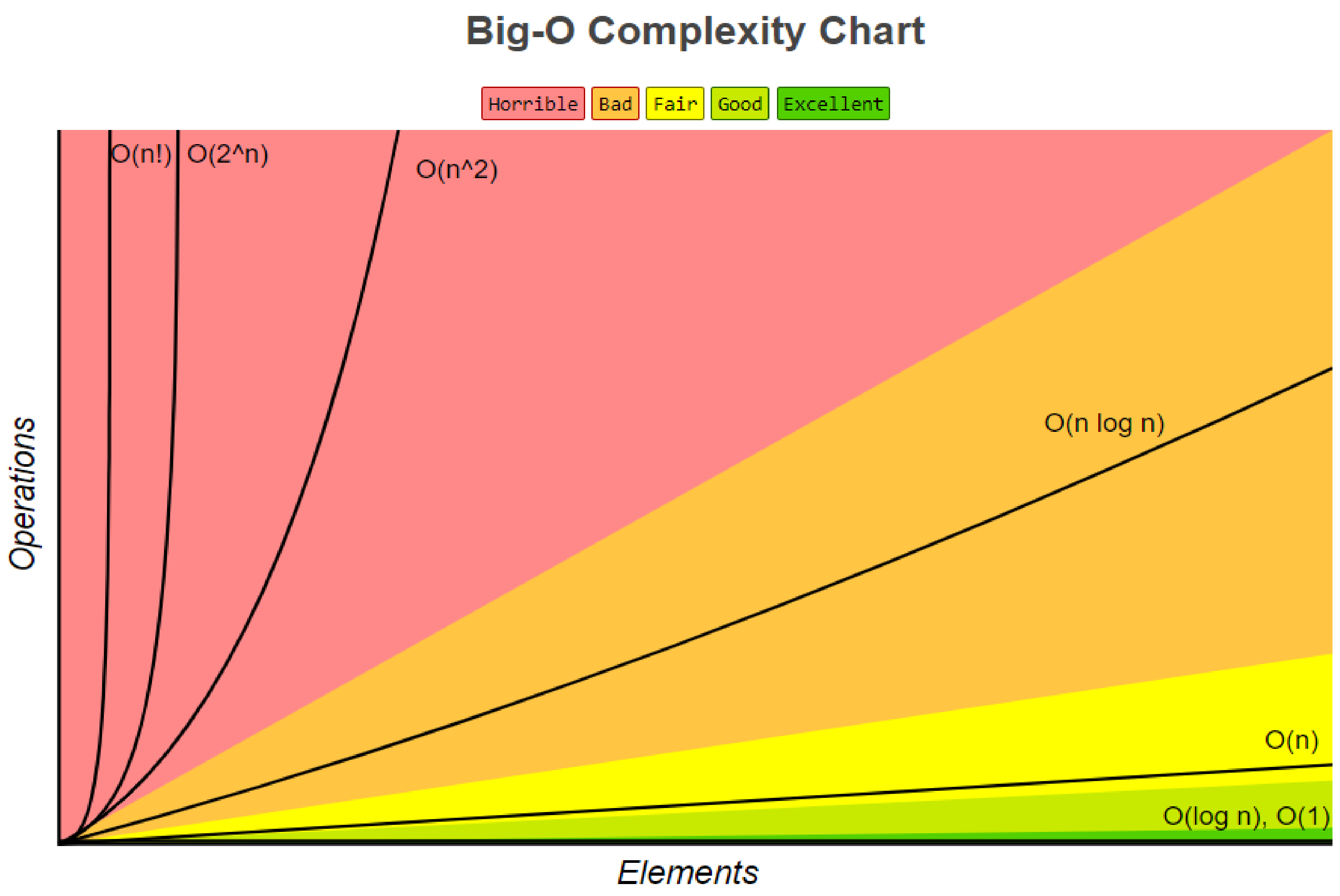
- The space complexity of an algorithm is the amount of space (or **memory**) taken by the algorithm to run as a function of its input length, *n*.
- Space complexity includes both *auxiliary space* and space used by the input.
- Auxiliary space is the temporary or **extra space used by the algorithm** while it is being executed.
- Space complexity of an algorithm is commonly expressed using Big O (O(n)) notation.
- Space Complexity is defined as the process of determining a formula for prediction of how much memory space will be required for the successful execution of the algorithm.
- The memory space we generally consider is the space of primary memory.
- Space complexity specifies the amount of temporary storage for running the algorithm.

Time complexity

- The time complexity of an algorithm is the **amount of time taken by the algorithm** to complete its process as a function of its input length, *n*.
- The time complexity of an algorithm is commonly expressed using *asymptotic notations*:
  - Big O - O(n),
  - Big Theta - Θ(n)
  - Big Omega - Ω(n)
- Time complexity is defined as the process of determining a formula for total time required towards execution of that algorithm
- This calculation will be independent of implementation details and programming language.

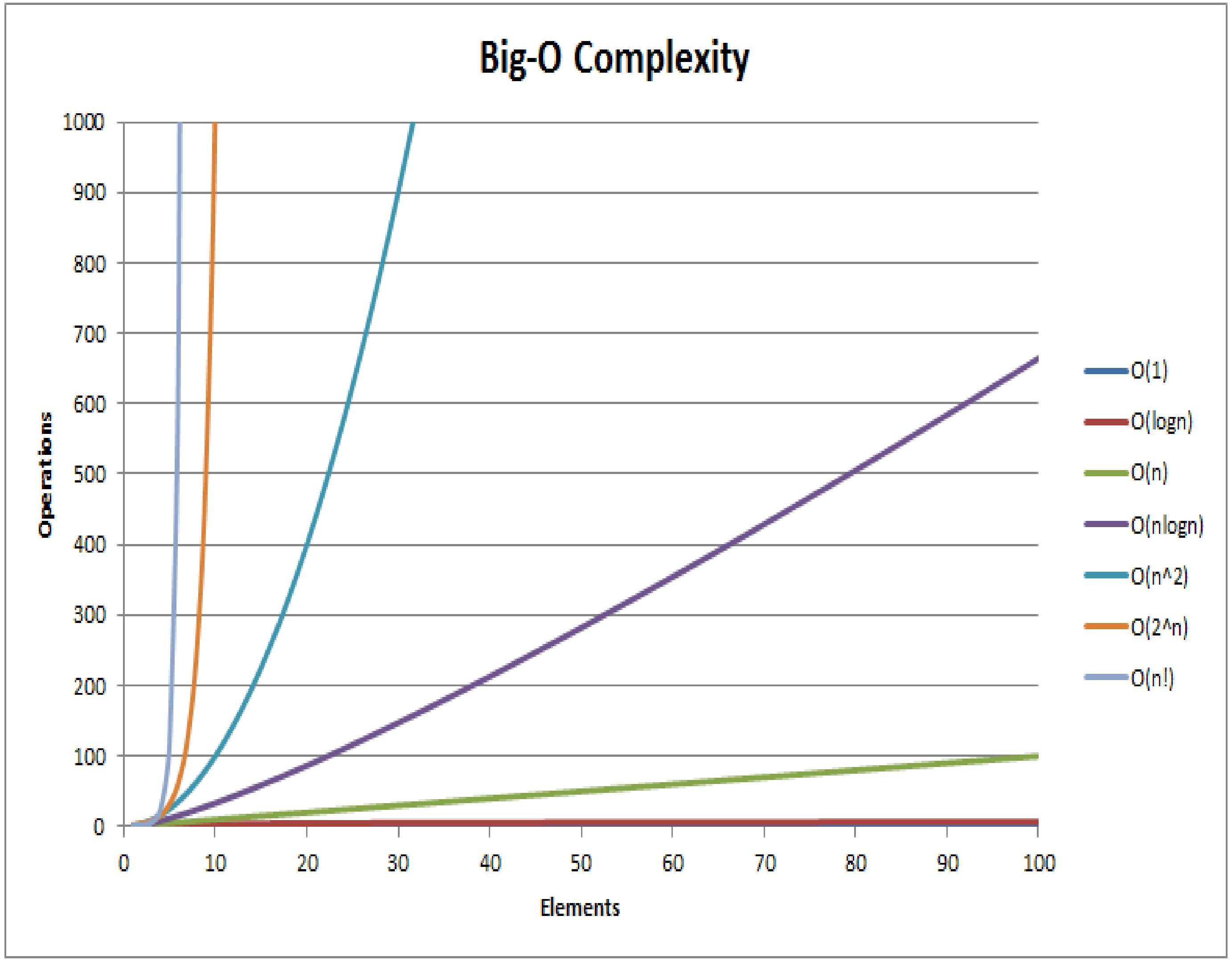
7.4 Polynomial-time and Exponential Algorithm

- The basic idea for problem classification.
- Classification basis:
  1. Algorithm based on polynomial time
  2. Algorithm based on exponential time



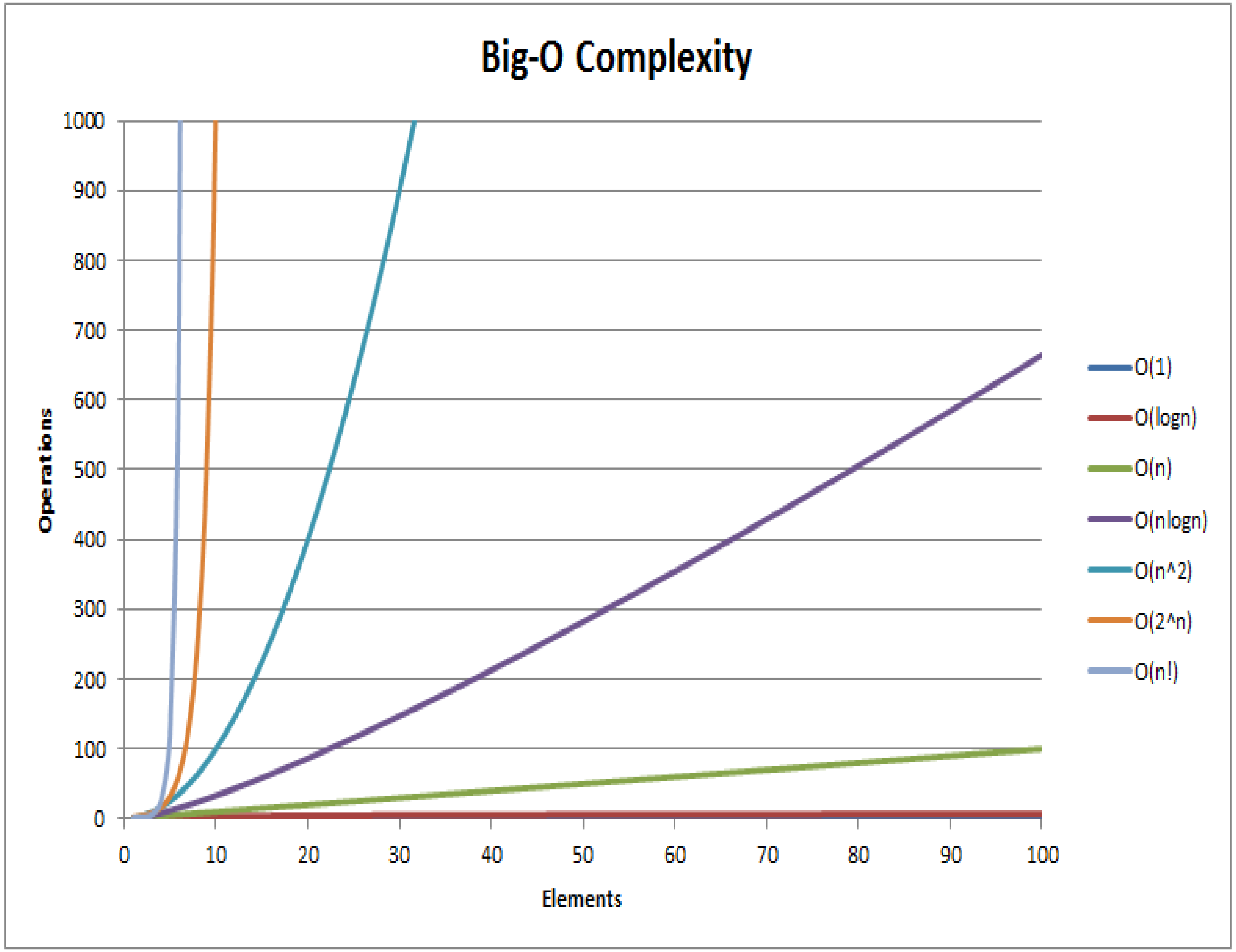
Polynomial-time Algorithms

- Algorithms run in polynomial-time if for a problem size is n, the number of steps needed to find the solution is polynomial function of n.
- The order-of-magnitude time performance is bounded by a polynomial function of n, where n is the size of its input.
- Polynomial functions are like O(1), O(log n), O(n), O(n x log n), $O(n^2)$, $O(n^3)$ and so on.
- Polynomial algorithms are considered to be efficient because **time grows less rapidly as the problem size increases.**



Exponential Algorithms

- Exponential algorithms are **not bounded by polynomial-time**.
- Exponentials functions are like: $O(k^n)$, $O(n!)$, $O(n^n)$.
- Exponential functions are considered to be inefficient because **the execution time of the latter grow much rapidly as the problem size increases.**



7.5 Tractable and Intractable Problems

Tractable Problems :

- Which can be **solved in polynomial time or less**.
- Class P

Intractable problems:

- Which **cannot be solved in polynomial time**
- Class NP, NP-hard and NP-complete

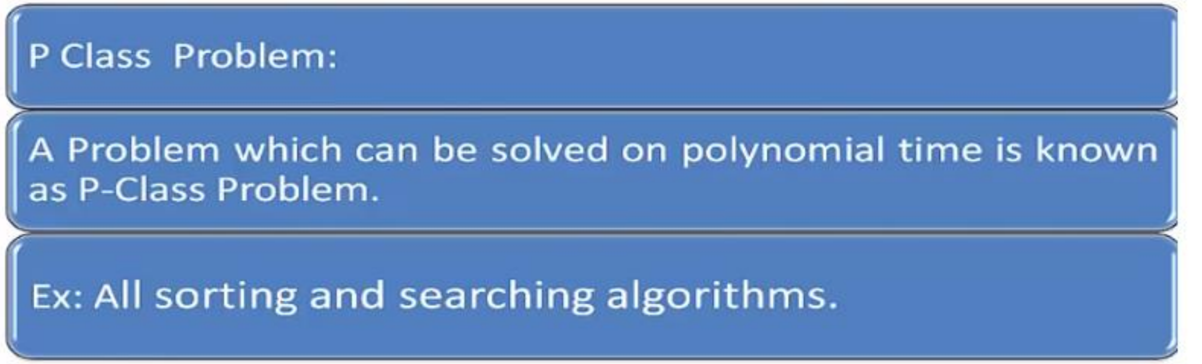
7.5.1 Tractable problems

- A decision problem is tractable if:
  - There **exists an algorithm to solve the given problem**, and
  - **time required is expressed in polynomial P(n),** n being the length of the input string.
- Tractable problems have algorithms of **polynomial complexity to solve problems.**
- The **P class** of problems are tractable problems.
- Examples of tractable problem:
  1. Searching an unordered list
  2. Searching an ordered list
  3. Sorting a list
  4. Multiplication of integers
  5. Finding a minimum spanning tree

7.5.2. Intractable problems

- Are computationally **hard or complex to solve.**
- A problem is said to be computationally intractable if the optimal algorithm for solving the problem **cannot solve all of its instances in polynomial time.**
- **Time required** for any algorithm to solve is at least f(n) where f is an **exponential** function of n.
- Problems that are solvable in theory , but **cannot be solved in practice** are intractable.
- The **NP, NP-hard and NP-complete** problems fall in this category.
- Some examples of intractable problems:
    1. Towers of Hanoi
    2. Travelling Salesman problem

7.6 Tractable problems : Class P



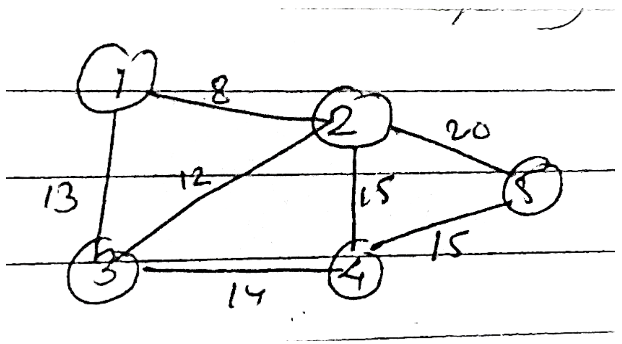
- A problem which **can be solved on Polynomial time**
- E.g.: All sorting and searching algorithms
- P is a class of problems that can be solved deterministically in polynomial time.
- These problems can be solved in time $O(n^k)$ in worst-case, where k is constant.
- The class P is important because: It is invariant over all models of computations
- Practical problems in P-class have efficient (low-degree polynomial) algorithms
- Class P is also a sub-set of NP class, but not P=NP
- The P-class problems are **easy to solve and also easy to verify in polynomial time.**

Examples of P-class

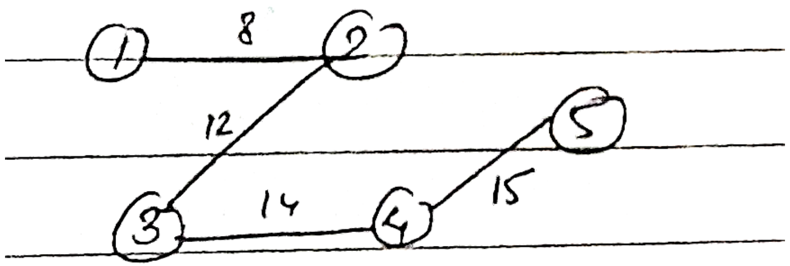
1. Kruskal's Algorithm: Minimum Weight-Spanning Tree

- A minimum weight spanning tree is a sub-graph that connects all the nodes (or vertices) together with least possible weight of the edges, yet there are no cycles.
- E.g. :

  XYZ company is providing cable services to 5 new housing developments. The goal is to determine the most economical cable network. Find the minimal spanning tree for below:



The solution is:



- This problem is class P problem because it is possible to implement Kruskal's algorithm (using computer) on a graph with m nodes and x edges in time $O(m + x \log x)$.

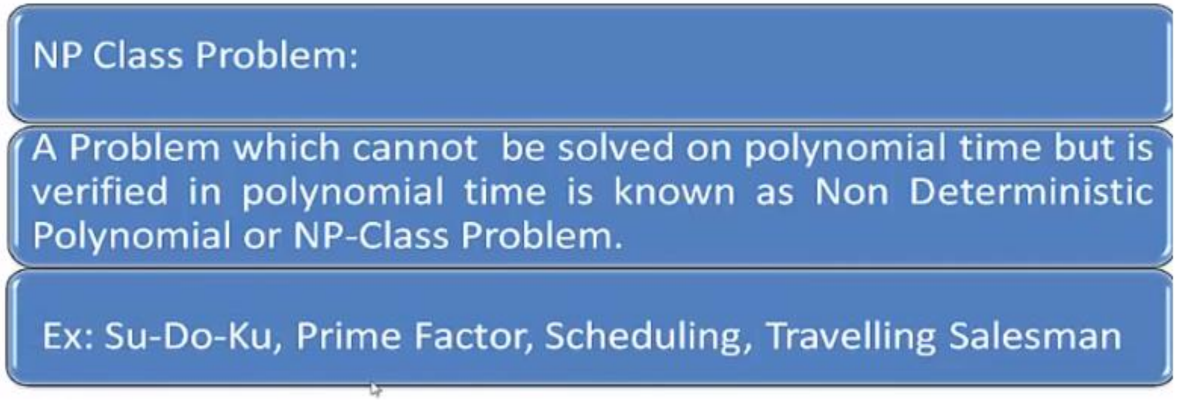
2. Problem of Finding the Maximum number

- Let us consider there are 5 numbers (n1,n2,n3,n4,n5)
- The problem is to find the largest among them.
- This can be solved as:

```
int max;
    max=n1;
if(n2>max)
    max=n2;
if(n3>max)
    max=n3;
if(n4>max)
    max=n4;
if(n5>max)
    max=n5;
```

- If there are N numbers in the list, this takes roughly N steps
- N is a polynomial function of N
- So this problem is the example of P-class

7.7 Intractable Problem: Class NP



- NP is the set of problems that **can be solved in non-deterministic polynomial time.**
- A problem which **cannot be solved on polynomial time** (**but in exponential time**), but **can be verified in polynomial time** is known as Non-deterministic polynomial or NP class problem.
- For e.g.: Su-Do-Ku, prime factor, scheduling, Travelling Salesman problem
- For class NP problems, **solving the problem is difficult, but verifying it is easy**
- The class NP is also invariant over all reasonable models of computation.

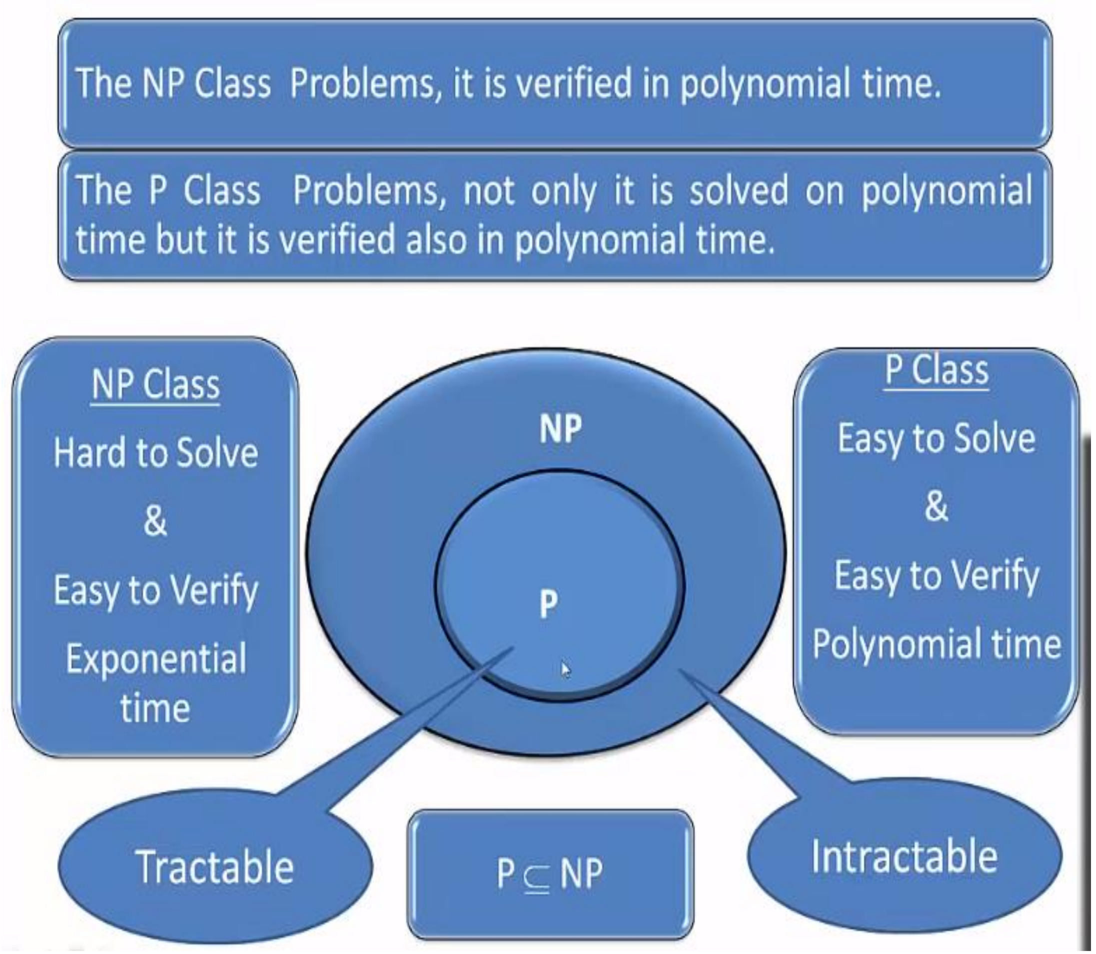
Example: The Su-Do-Ku problem

- Goal: Find the 9x9 grid with numbers so that each row, column and 3x3 section contains all of the digits between 1 and without repetition.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- It is very difficult to solve the problem.
- It to solve this problem takes an exponential time .
- However, it takes very less time to verify if the problem is solved or not.
- Hence, it is a class NP problem.

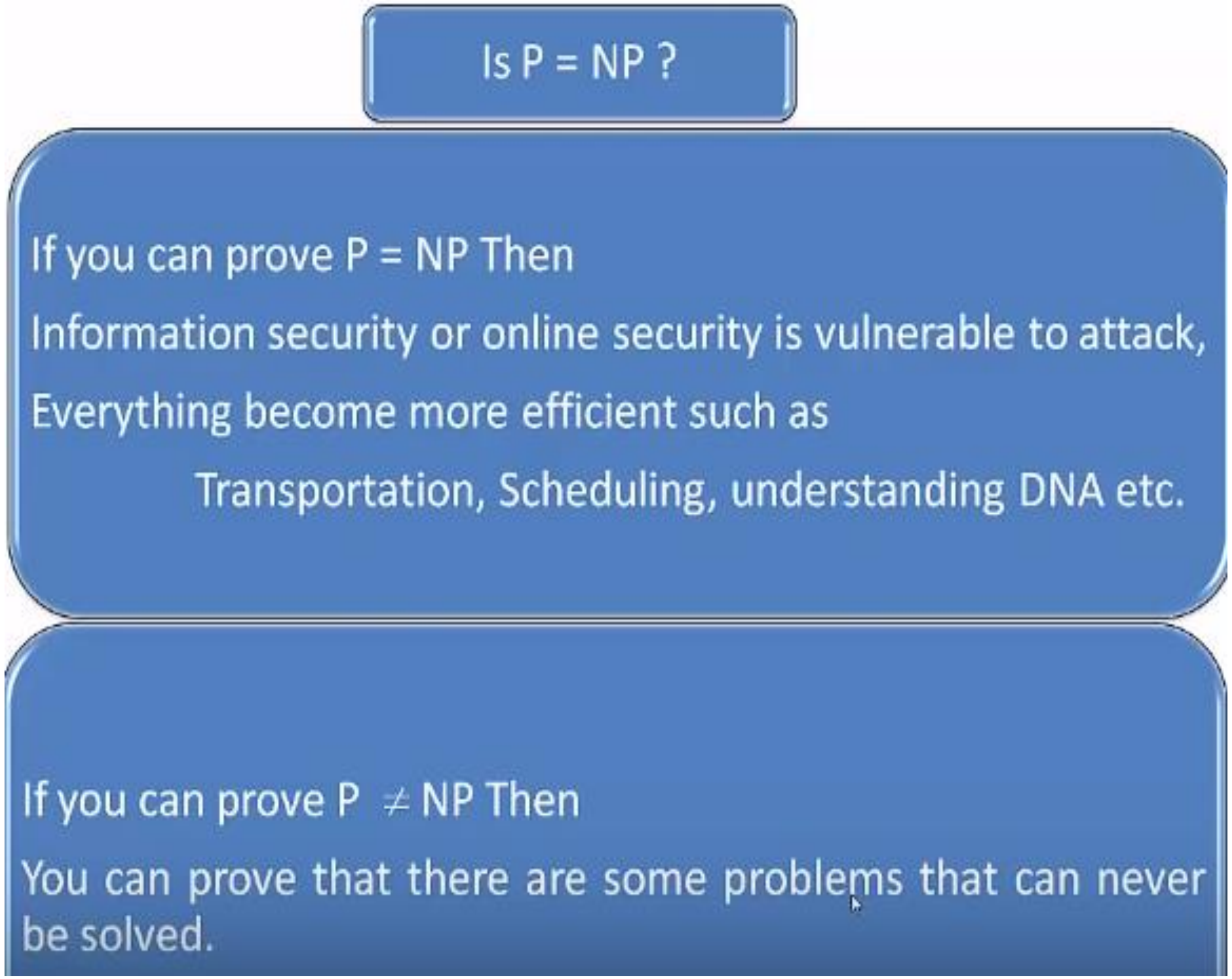
P-class VS NP-class



P-class	NP-Class
• P-class problems are solved in polynomial time by using deterministic algorithm.	• NP-class are verified in polynomial time by using non-deterministic algorithms.
• All the P-class problems are basically deterministic.	• All the NP-class problems are basically non-deterministic.
• Every problem which is a P-class is also in NP-class.	• Every problems in NP-class is not in P-class.
• P-class problems are easy to solve.	• NP-class problems are difficult to solve.
• Eg: ➢ Linear search ➢ Binary search ➢ Bubble sort ➢ Matrix addition	• Eg: ➢ TSP ➢ HCP

Reference Video:
https://www.youtube.com/watch?v=DumOqL85Ryc

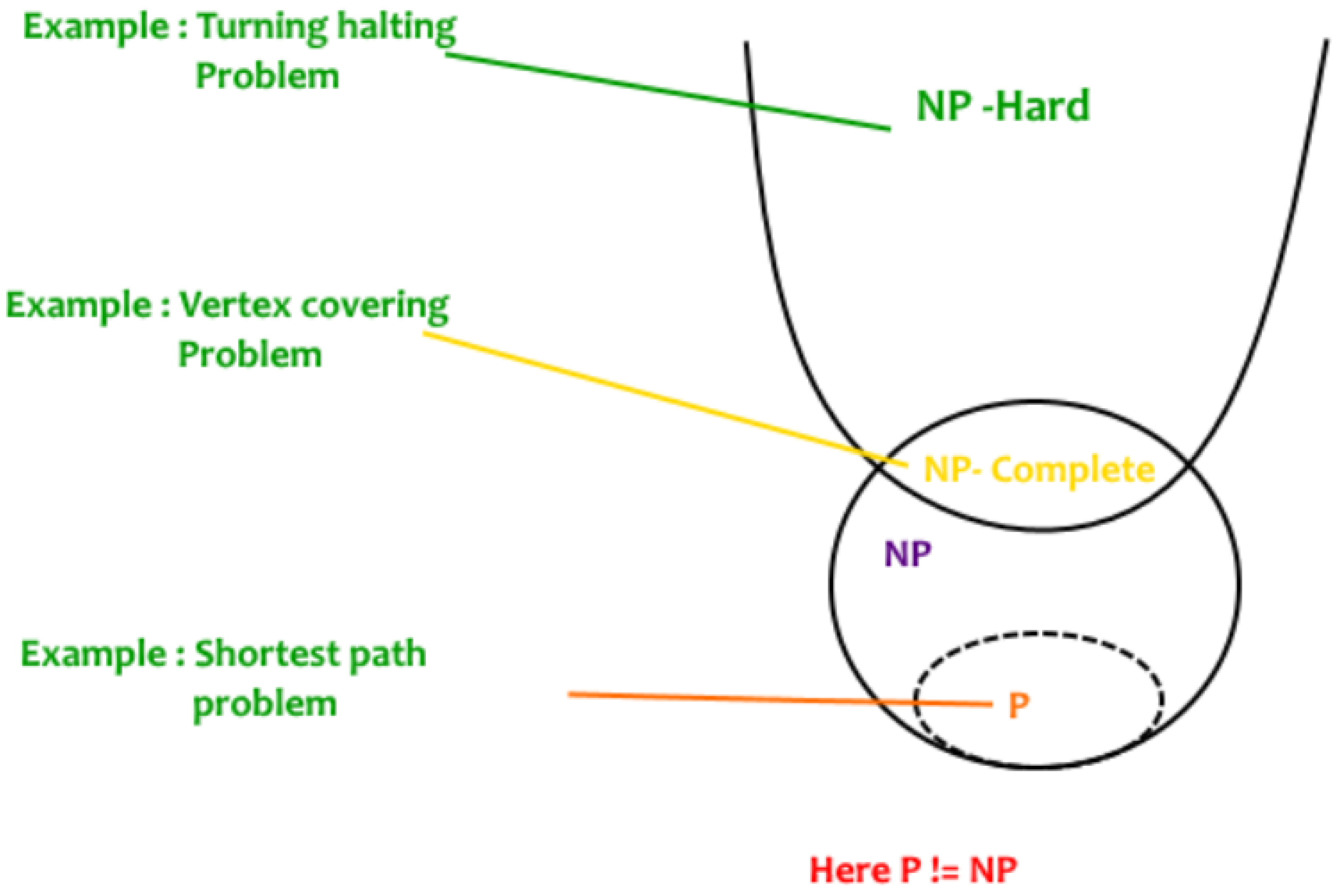
Is P=NP?



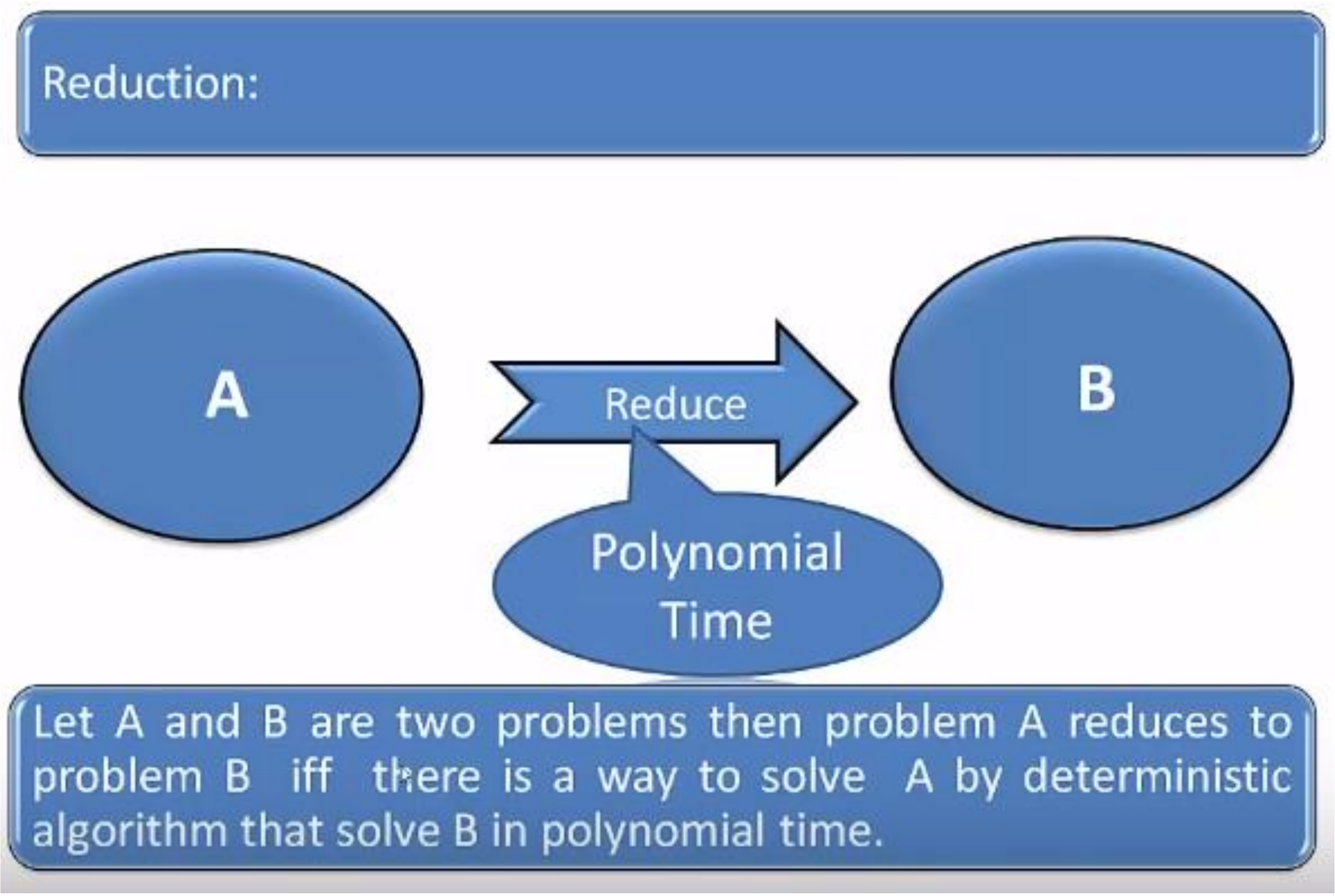
- An unsolved problem
- If solved, **$ 1 million reward !!!**
- Simply asks whether computationally hard problems actually contain hidden and computationally easy solutions.
- **If P≠NP** then we could prove that there are some problems that can never be solved.
- **If P=NP** then all cryptography would have been easier.
- All transportation, scheduling, etc. would have been very easy to be solved.

7.8 Intractable Problems: NP-Hard and NP-complete

- Which cannot be solved in polynomial time
- Class NP further classified as: NP-hard and NP-complete

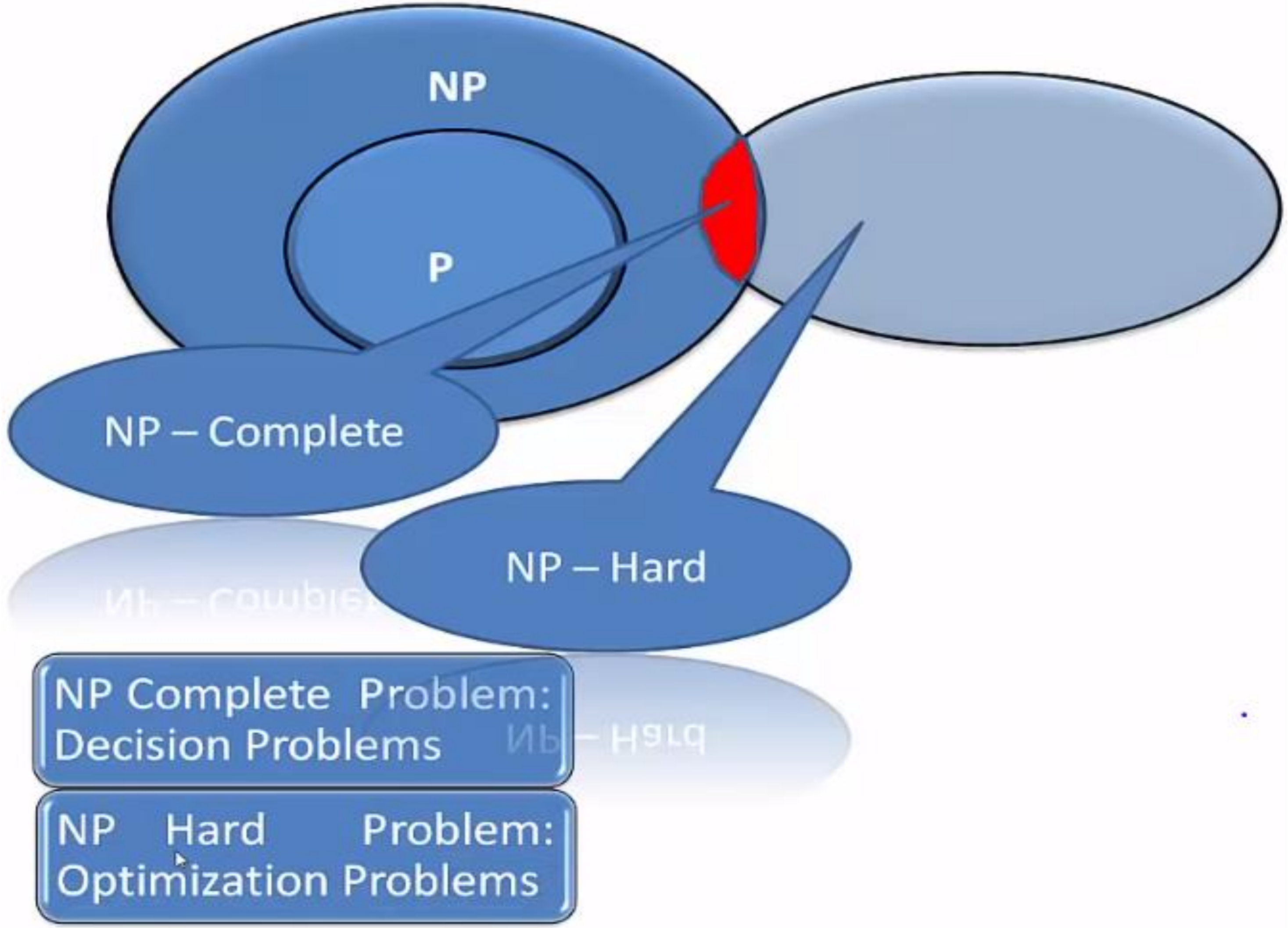


Polynomial Time Reduction



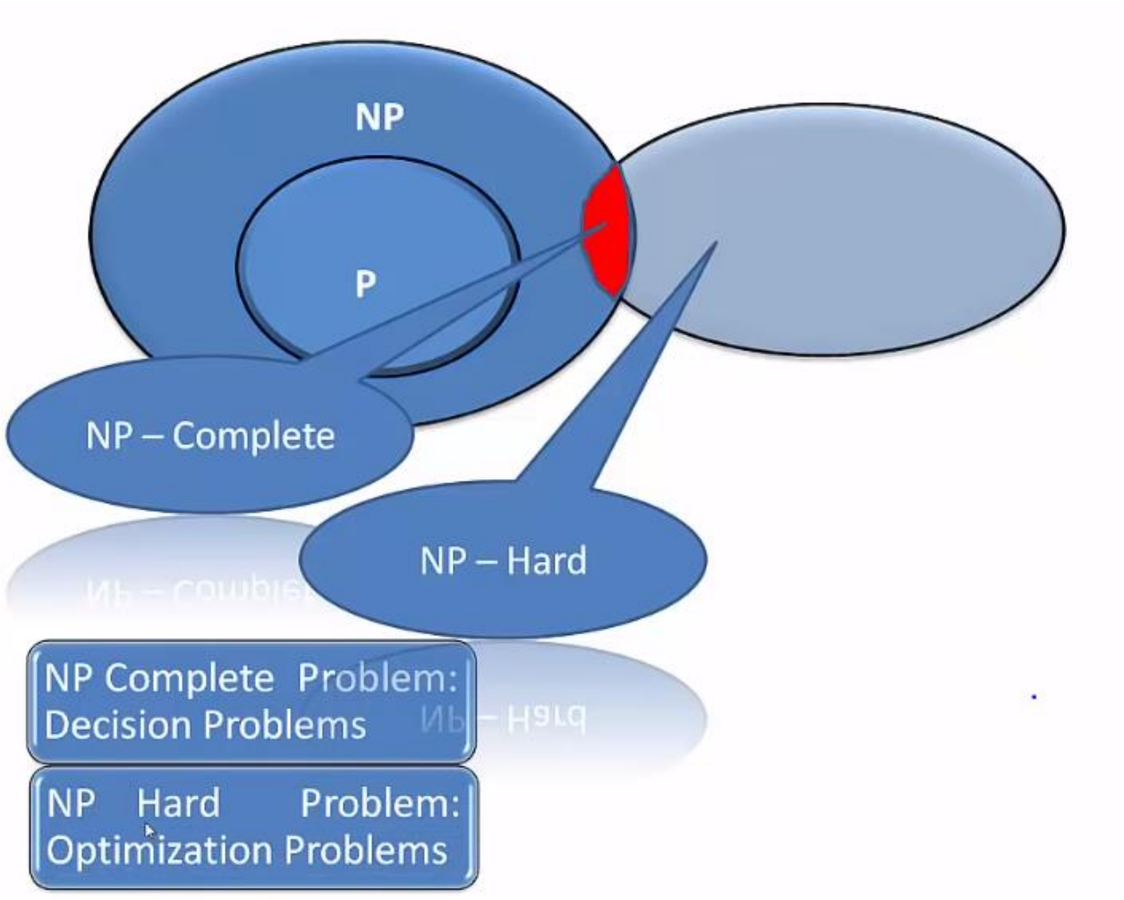
- Let A and B are two problems.
- Then A reduces to B iff there is a way to solve A by deterministic algorithm that solve B in polynomial time.
- If A is reducible to B, we denote it by $A \propto B$

NP-hard and NP-Complete



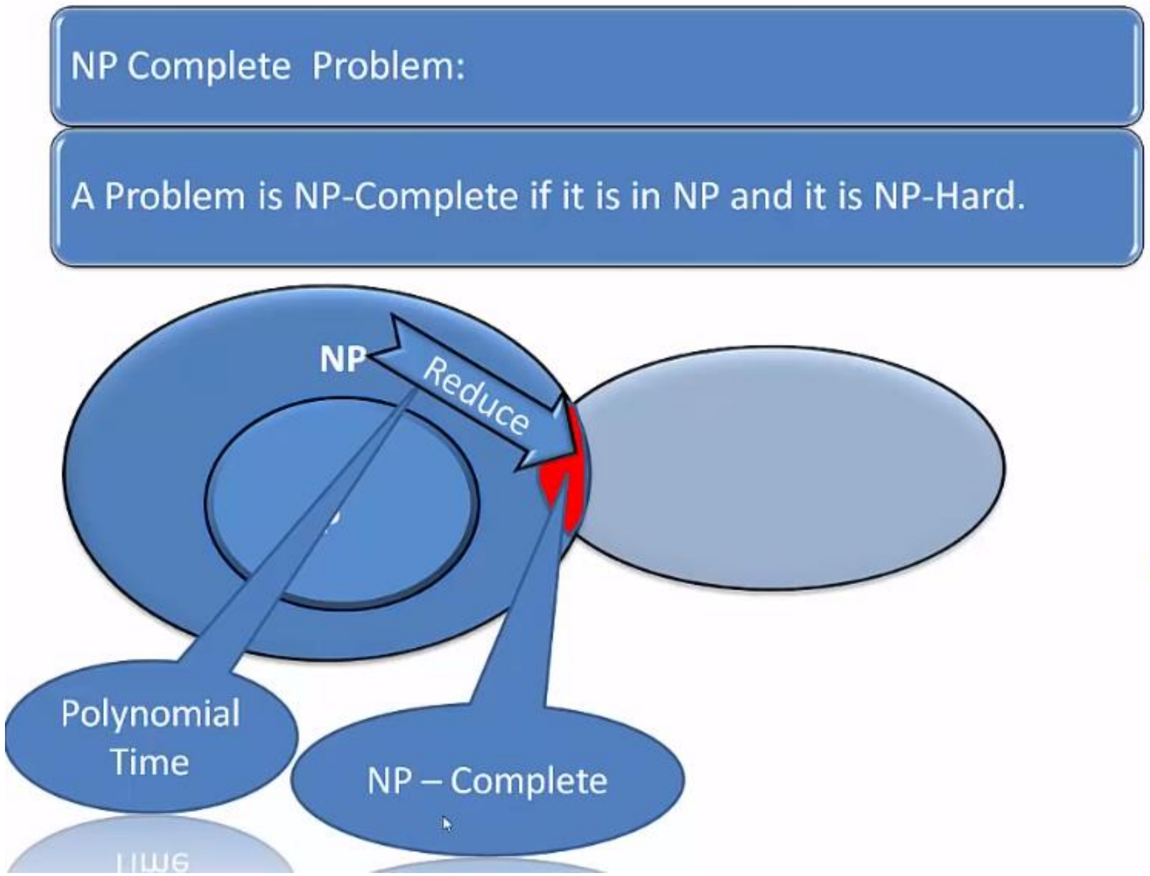
- A problem is **NP-hard** problem if every problem in NP can be polynomially reduced to it.
- A problem is **NP-complete** if it is NP and it is NP-hard.

7.8.1 NP-hard



- A problem is NP-hard problem if every problem in NP can be polynomially reduced to it.
- The NP-hard problem may or may not fall within NP
- NP-hard problems are basically optimization problems.
- NP-hard means "at least as hard as many NP-problems"
- Some common examples or NP-hard problems are:
  - The Halting Problem
  - Hamilton Cycle problem (HCP)
  - Travelling salesman Problem (TSP)
  - Vertex Cover Problem (VCP)
  - Partition problem

7.8.2 NP-complete

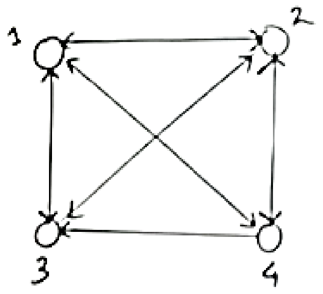


- A problem is NP-complete if it is NP and it is NP-hard.
- It is a subset of NP-hard.
- So **every NP-complete problems are NP-hard but not vice-versa.**
- NP-complete problems are decision problems.
- Some common examples of NP-complete problems are:
  - Hamilton Cycle problem (HCP)
  - Travelling salesman Problem (TSP)
  - Vertex Cover Problem (VCP)
  - Partition problem

1. Travelling Salesman Problem (TSP)

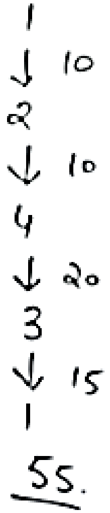
- Given **n** cities $c_1, c_2, ..c_n$ and distance $d_{ci, cj}$ between any two cities c_i and c_j
- TSP asks for the total distance of the shortest tour of the cities.
- A travelling salesman wants to visit each of n cities exactly once and return to its starting point with minimum distance/mileage.
- It is a minimization problem, i.e. to find the minimum distance tour.

Given problem of TSP

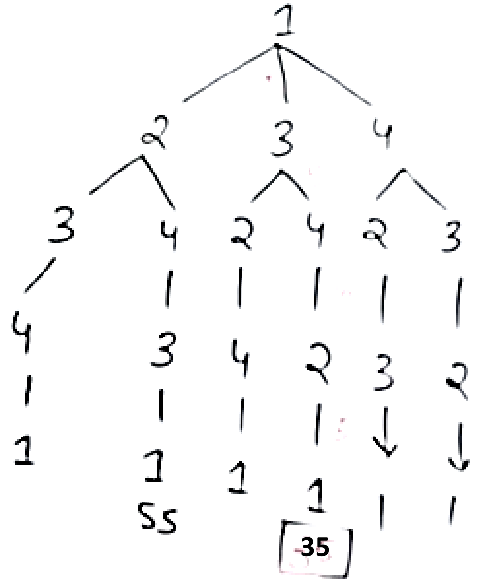


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 25 & 10 \\ 3 & 15 & 30 & 0 & 5 \\ 4 & 15 & 10 & 20 & 0 \end{array}$$

Using Greedy Algorithm



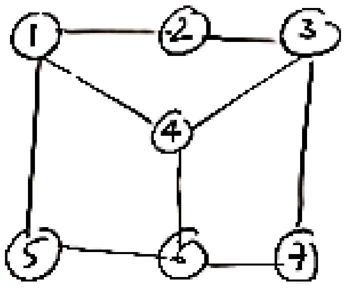
Using Dynamic programming

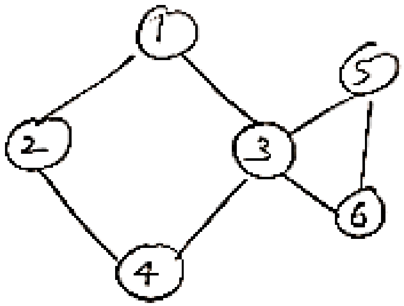


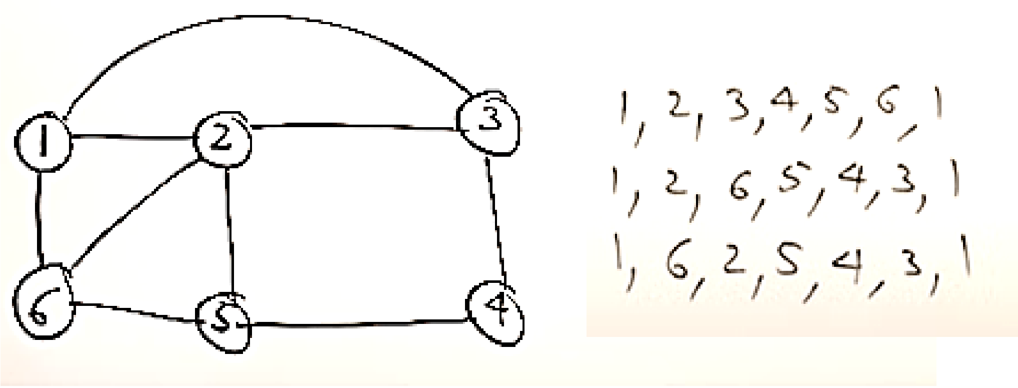
Reference: https://www.youtube.com/watch?v=3QiSyc7KyC4

2. Hamilton Cycle problem (HCP)

- Hamiltonian Path in a directed graph G is a directed path that goes through each node exactly once.
- Hamiltonian cycle or circuit is a Hamiltonian path, that there is an edge from the last vertex to the first vertex.
- In this problem, we will try to determine whether a graph contains a Hamiltonian cycle or not.
- Unlike TSP, it is to find all possible tours.
- The first and second graphs do not contain the Hamiltonian cycle, but the last one does.





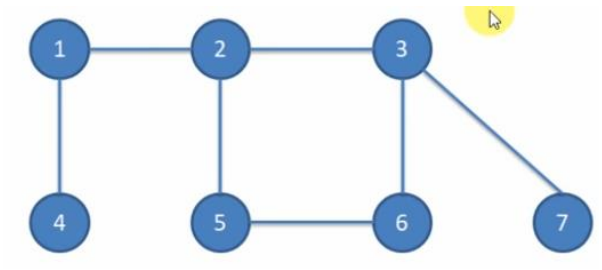


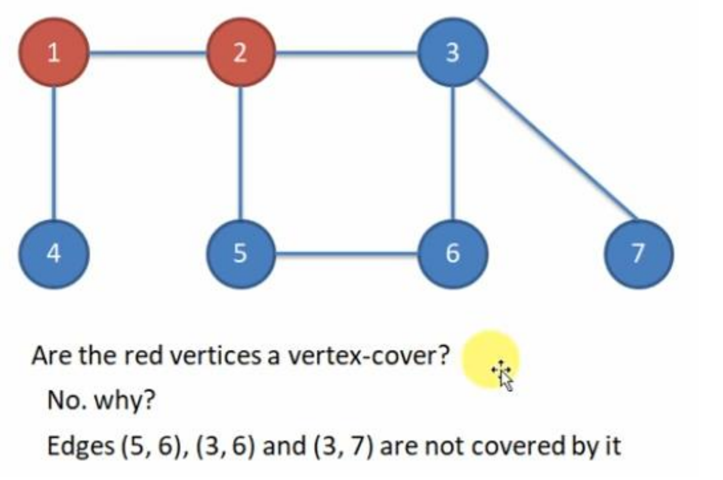
- In the first, starting from 1, we move 1-2-3-7-6-4-1. So 5 is missing.
- In the second, starting from 1, we move 1-2-4-3-6-5-?. We cannot revisit 3.

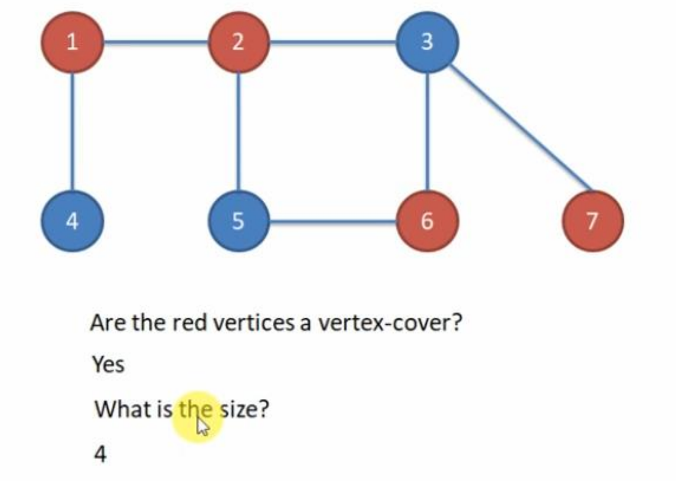
Reference: https://www.youtube.com/watch?v=dQr4wZCiJJ4

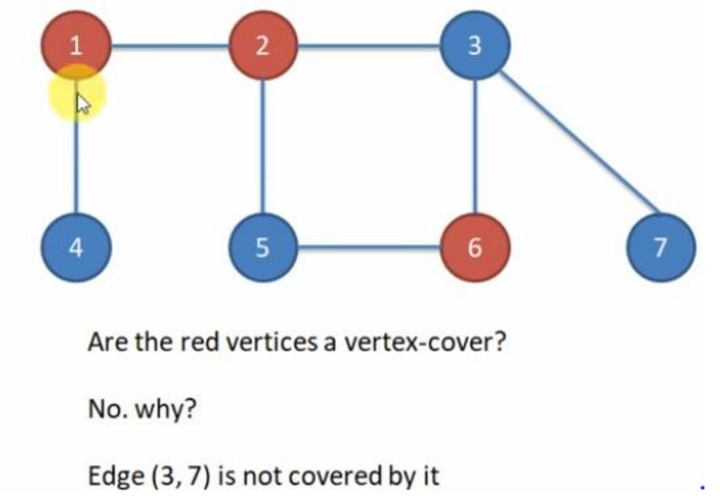
3. Vertex Cover Problem

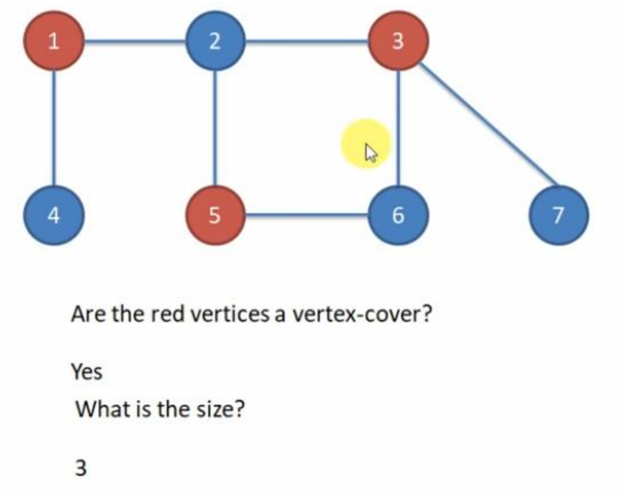
- **A Vertex Cover of a graph is a subset of vertices which covers every edge.**
- **An edge is covered if one of its endpoint is chosen.**
- **The vertex cover problem: What is the minimum size of vertex covered in graph G?**
- **Idea: Keep finding a vertex which covers the maximum number of edges.**
- **E.g: Given a undirected graph, find a vertex cover with minimum size.**











Reference: https://www.youtube.com/watch?v=ZZxj9hqldng

4. Partition Problem

- The task of deciding whether a given multiset **S** of positive integers can be partitioned into two subsets $\mathbf{S_1}$ and $\mathbf{S_2}$ such that:
  - Sum of numbers in S_1 = sum of numbers in S_2
  - The subsets S_1 and S_2 must form a partition in the sense that they are disjoint and they cover **S.**
- A variation of the partition problem is the 3-partition problem, in which the set S must be partitioned into $|S|/3$ triples each with the same sum.

- **Example:**
  - Given $S = \{3,1,1,2,2,1\}$,
  - a valid solution to the partition problem is the two sets :
  - $S_1 = \{1,1,1,2\}$ and $S_2 = \{2,3\}$.
  - Both sets sum to 5, and they partition S.
- Note that this solution is not unique.
- $S_1 = \{3,1,1\}$ and $S_2 = \{2,2,1\}$ is another solution.

References:

- https://www.youtube.com/watch?v=ZSyZozw6JMQ
- https://www.youtube.com/watch?v=2cyryXRmN5Q&list=PLPmNdapEAffgvtjCJLG558pQNweui0Mos
- https://www.youtube.com/watch?v=DumOqL85Ryc&list=PLPmNdapEAffgvtjCJLG558pQNweui0Mos&index=7
- https://www.youtube.com/watch?v=RiDzt22KUd8&list=PLPmNdapEAffgvtjCJLG558pQNweui0Mos&index=2
- https://www.youtube.com/watch?v=EkmjDoiKZ8Q
- https://www.geeksforgeeks.org/np-completeness-set-1/

Reference videos for Complete TOC tutorial: (from Neso Academy)

- https://www.youtube.com/watch?v=58N2N7zJGrQ&list=PLBlnK6fEyqRgp46KUv4ZY69yXmpwKOIev

End of chapter