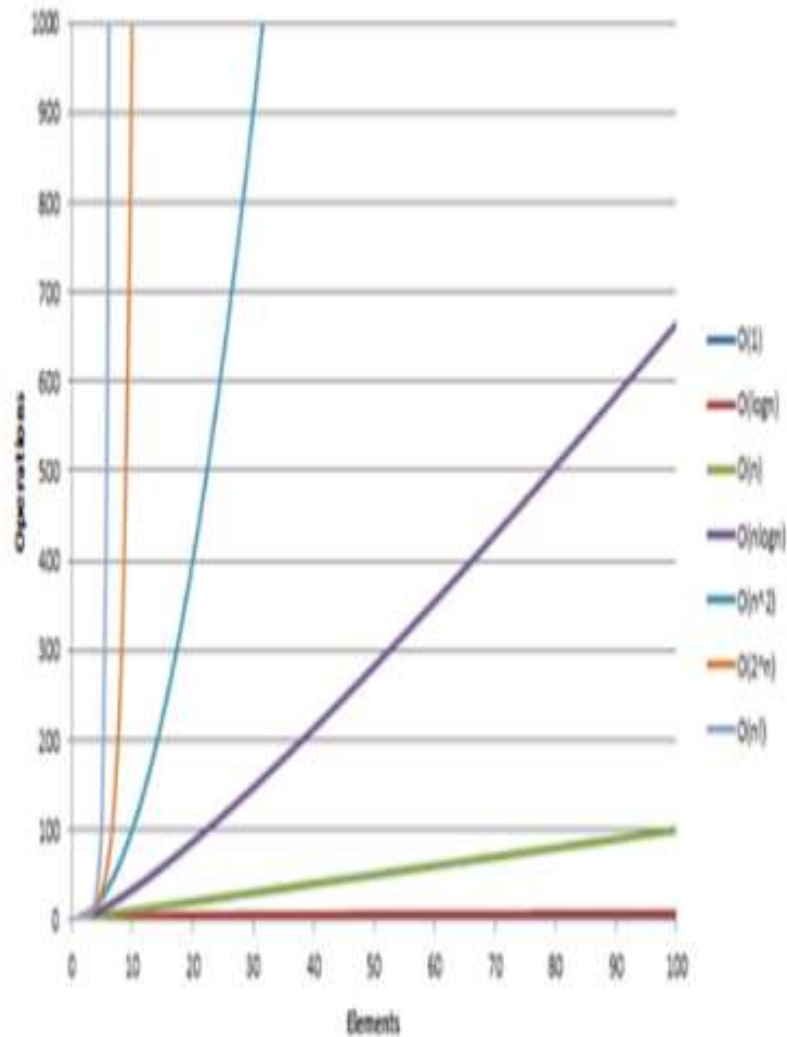




P Versus NP Problems



Big-O Complexity



Input Size (n)	Time Complexity				
	n	$n \log_2 n$	n^2	n^3	2^n
10	< .001 second	< .001 second	< .001 second	< .001 second	< .001 second
20	< .001 second	< .001 second	< .001 second	< .001 second	.001 second
30	< .001 second	< .001 second	< .001 second	< .001 second	1 second
50	< .001 second	< .001 second	< .001 second	< .001 second	13 days
100	< .001 second	< .001 second	< .001 second	.001 second	4×10^{11} centuries
1000	< .001 second	< .001 second	.001 second	1 second	4×10^{282} centuries
100,000	< .001 second	.002 second	10 seconds	11.57 days	–
one million	.001 second	.02 second	1.67 minutes	32 years	–
ten million	.01 second	0.24 second	1.2 days	317 centuries	–
one billion	1 second	30 seconds	32 years	4×10^8 centuries	–
100 billion	1.67 minutes	1 hour	3171 centuries	4×10^{14} centuries	–

▶ **Polynomial time**

- ▶ Linear Searching
- ▶ Binary Searching
- ▶ Insertion sort
- ▶ Bubble sort
- ▶ Merge sort

▶ **Exponential time**

- ▶ 0/1 knapsack
- ▶ TSP
- ▶ Sum of subsets

Need a faster Algorithm.....

NP hard and NP Complete are guidelines made for doing research on exponential time problems



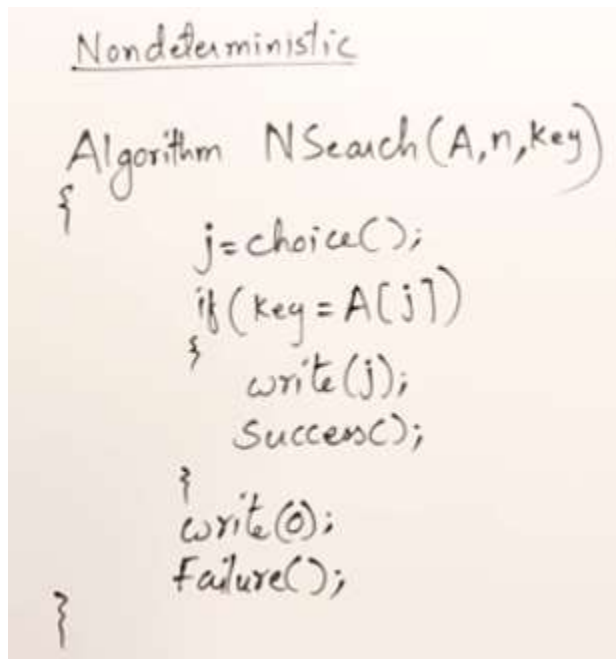
-
- ▶ Points to consider if you are unable to solve exponential time algorithm using polynomial time
 - ▶ If not able to write deterministic algms, write non-deterministic algms
 - ▶ Try to find a relationship between exponential algorithms so that if one is solved in polynomial time, others are also solved in polynomial time

In Deterministic algorithms, we know the working of the algorithmic.



Non- Deterministic Algorithm

- ▶ Don't know how they will work
- ▶ Most of the statements are deterministic
- ▶ Few are non-deterministic----preserved for future use



Handwritten code for a Non-deterministic Search algorithm:

```
Non-deterministic  
Algorithm NSearch(A, n, key)  
{  
    j = choice();  
    if (key == A[j])  
    {  
        write(j);  
        Success();  
    }  
    write(0);  
    Failure();  
}
```

▶ We define 2 classes here

▶ P

▶ NP



-
- ▶ **Need a base problem**
 - ▶ Satisfiability problem



▶ **Exponential time**

- ▶ 0/1 knapsack
- ▶ TSP
- ▶ Sum of subsets





P versus NP

Decision problem: problem for which the answer must be either YES or NO

Polynomial time algorithm: there is a constant c such that the algorithm solves the problem in time $O(n^c)$ for every input of size n

P (*polynomial time*) - class of decision problems for which there is a polynomial time deterministic algorithm **solving** the problem

NP (*nondeterministic polynomial time*) - class of decision problems for which there is a certifier which can **check** a witness in polynomial time



P

NP

$$\begin{array}{r} 51 \\ \times 3 \\ \hline 153 \end{array}$$

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

Sudoku



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

Sudoku

P vs NP problem

=

F	2				6	C	B	3
C			4	8	E	A		0
D	A	8		3	2	7	F	
6		E	D	F	C	8		7
9	3	7				A		2
E			6	F	5	8	4	3
C	8	1	3	9	D	0	2	E
D	6	5	E	B	1			0
9	6			1	F	3	2	0
		4	A	8	D	0	9	B
2	A	0	D	5	6	C		F
5			2			A	4	8
B			4	1	A	2	F	0
0	7		F	3	C	D		2
	5	1		A	9	0	B	
2	D	A		9				1

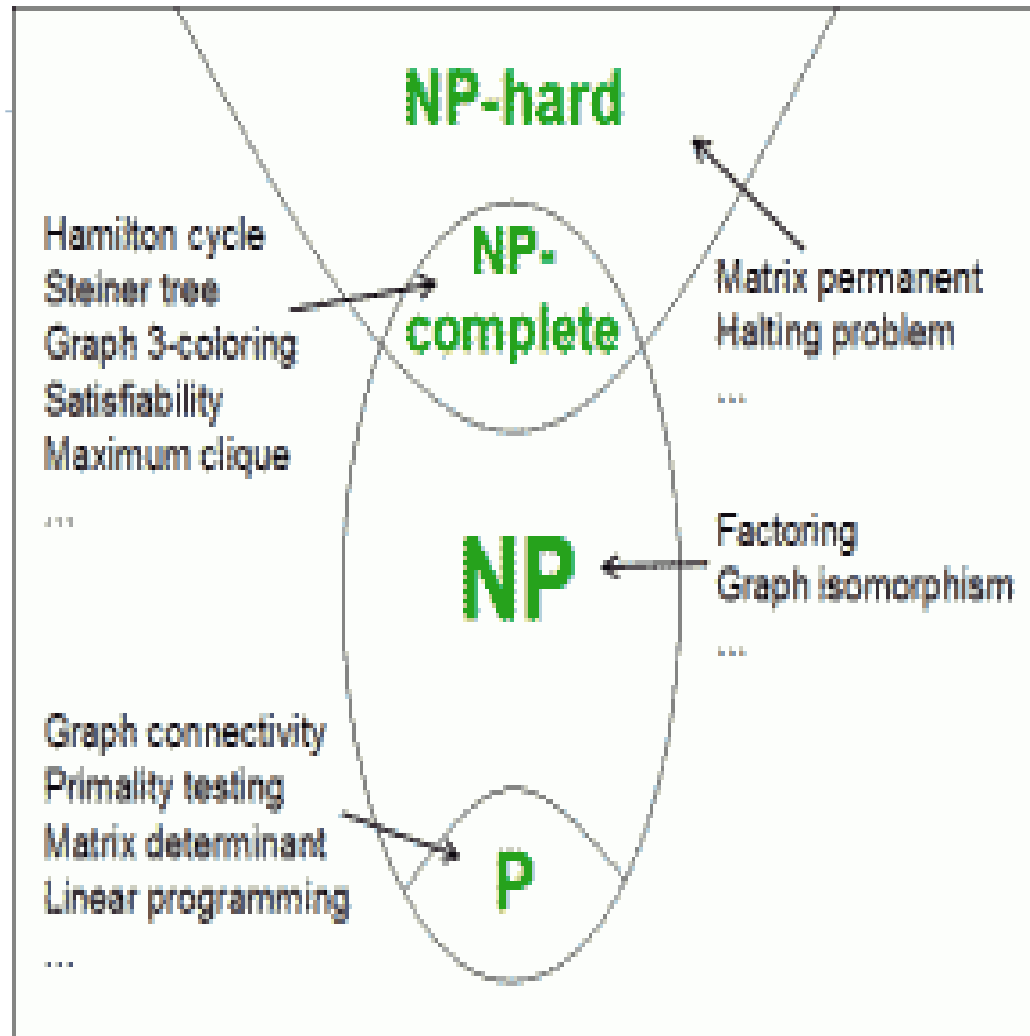
■
■
■

$n \times n \times n$

Does there exist an algorithm for solving $n \times n \times n$ Sudoku that runs in time $p(n)$ for some polynomial $p()$?

FORMAL DEFINITION OF P VERSUS NP PROBLEM

- Formally, P is the class of problems solved by a deterministic Turing machine in a polynomial time. NP is the class of problems which are solved by a non-deterministic Turing machine in a polynomial time, which means it can be solved by brute force algorithm in an exponential time.
- We know that any P problem is also an NP one. In fact the fast algorithm that solves the P problem can be its fast verification algorithm. The question is could we find an NP problem which doesn't have definitely a fast algorithm to solve it. If so then NP is different from P. If no then NP is identical to P.



NP-Complete & NP-Hard

What's P and NP ?

- ▶ $P = \{ \text{problems solvable in polynomial time} \}$
- ▶ $NP = \{ \text{decision problems solvable in non-deterministic polynomial time} \}$
 - can be verified in polynomial time.

Is $P=NP$?



NP-COMPLETE

- ▶ problems whose status is unknown.
- ▶ hardest problems in NP set.
- ▶ No polynomial time algorithm has yet been discovered for any NP complete problem.
- ▶ If any one of the NP complete problems can be solved in polynomial time, then all of them can be solved.
- ▶ If two problems can be reduced to each-other, they are in some sense, "equivalent". All problems in NP that are so equivalent, are called **NP-Complete**.



How can we say that a problem is NP complete?

2 conditions-

▶ $X \in NP$

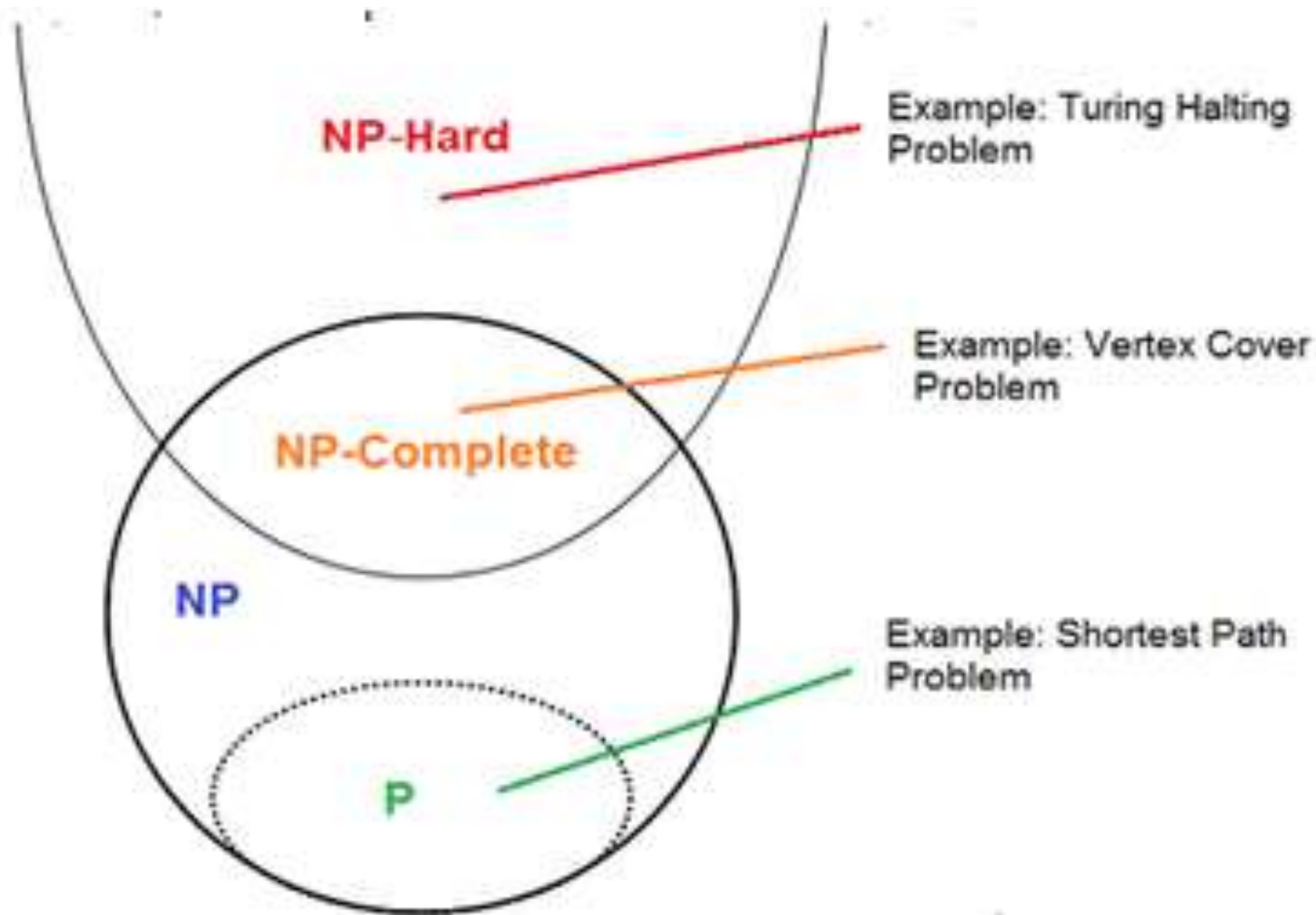
▶ X is NP hard



NP-HARD

- ▶ The precise definition here is that *a problem X is NP-hard, if there is an NP-complete problem Y , such that Y is reducible to X in polynomial time.*
- ▶ these are the problems that are *at least as hard as the NP-complete problems.*
- ▶ I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B.



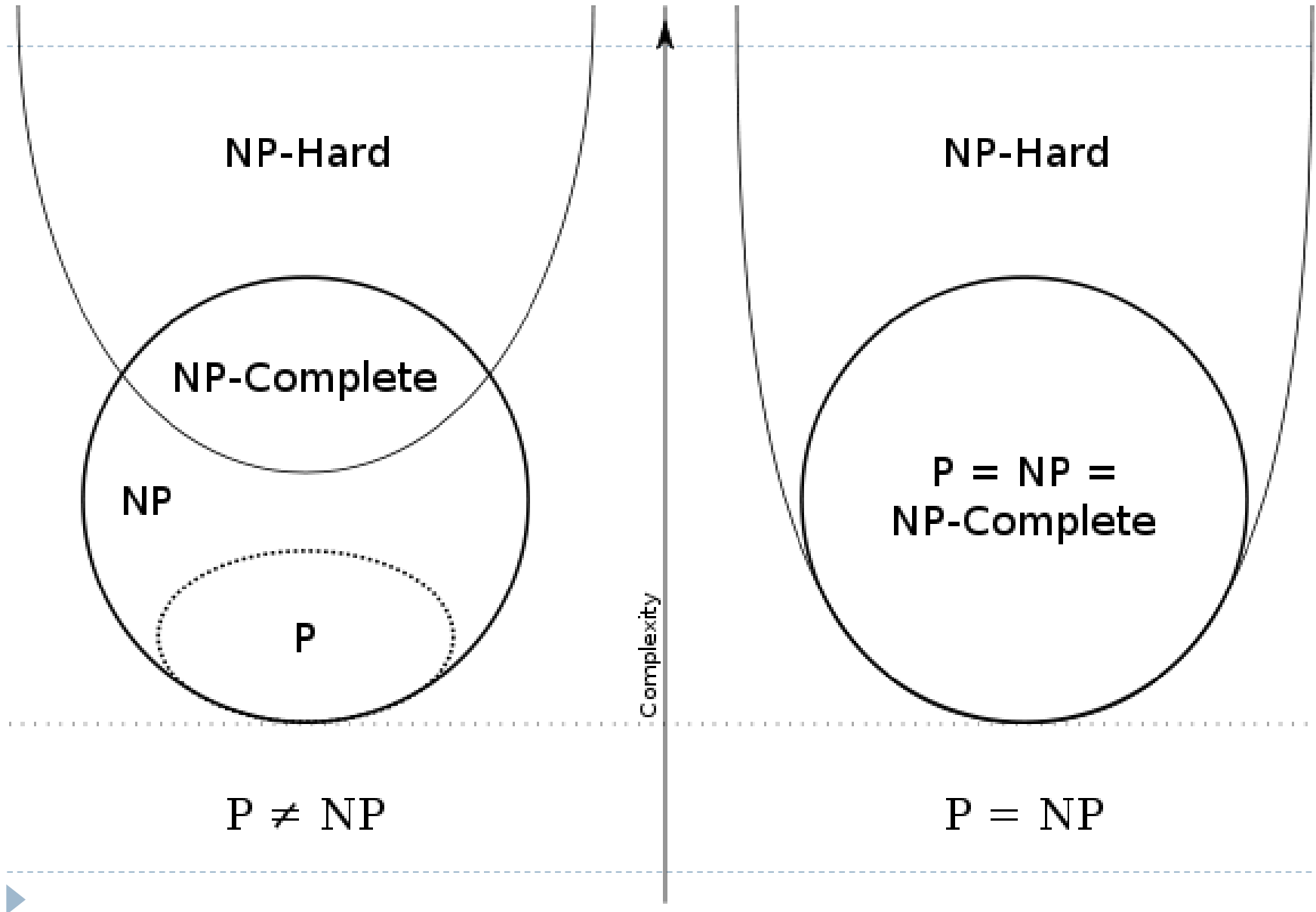


This diagram assumes that $P \neq NP$

*NP-complete problems are either optimization, search or decision problems. The classical TSP is an optimization problem. For a problem to be strictly NP-hard it should be neither an **optimization/decision/search** problem*

But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.





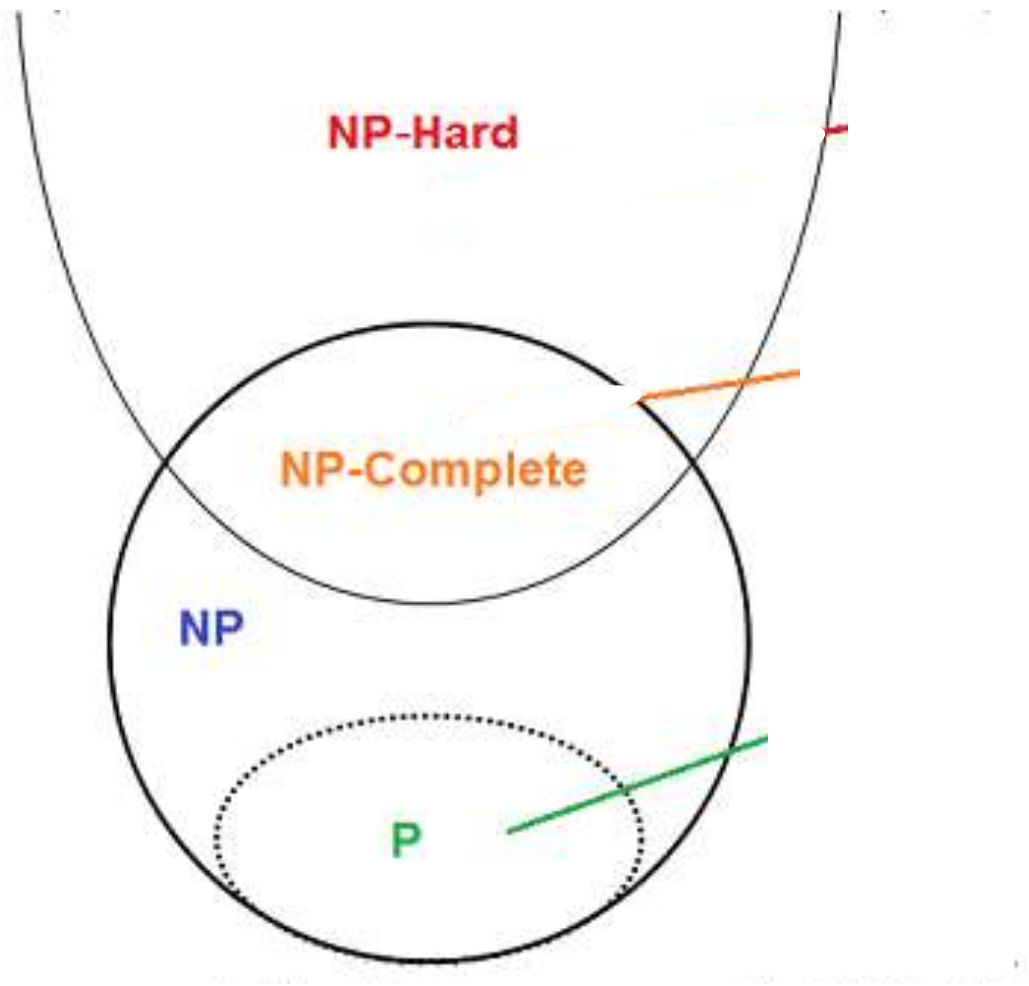
NP-completeness and reducibility

NP-completeness

NP-complete problems are the hardest problems in NP set. A decision problem L is NP-complete if:

- 1) L is in NP (Any given solution for NP-complete problems can be verified quickly, but there is no efficient known solution).
- 2) Every problem in NP is reducible to L in polynomial time





NP-completeness and Reducibility

A (class of) problem P_1 is *poly-time reducible* to P_2 , written as $P_1 \leq_p P_2$ if there exists a poly-time function $f: P_1 \rightarrow P_2$ such that for any instance of $p_1 \in P_1$, p_1 has “YES” answer if and only if answer to $f(p_1)$ ($\in P_2$) is also “YES”.

