

Introduction to

# Algorithm Design and Analysis

[18] NP Complete Problems 1

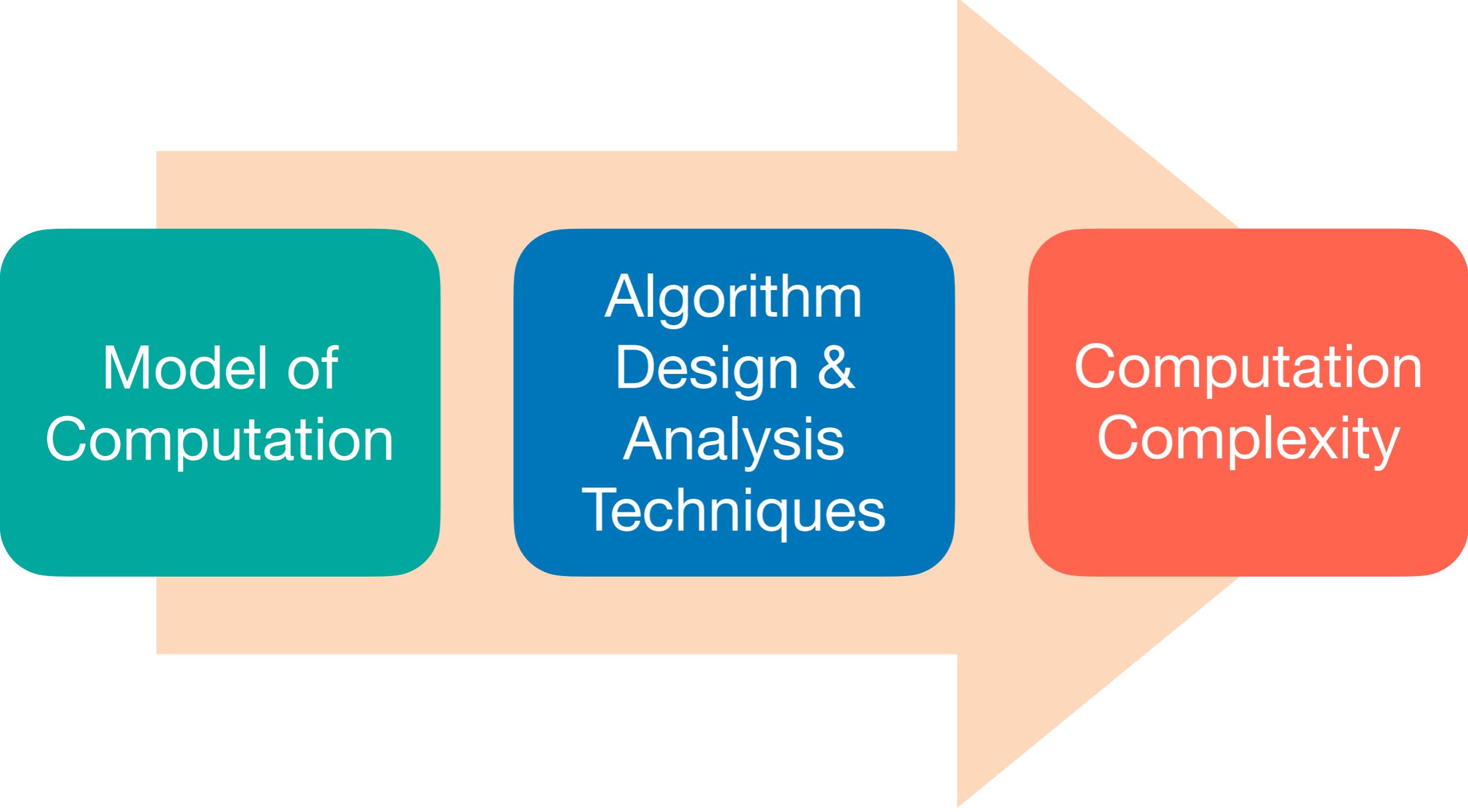
Jingwei Xu

[https://cs.nju.edu.cn/ics/people/jingweixu/  
index.html](https://cs.nju.edu.cn/ics/people/jingweixu/index.html)

Institute of Computer Software

Nanjing University

# Syllabus



Model of Computation

Algorithm Design & Analysis Techniques

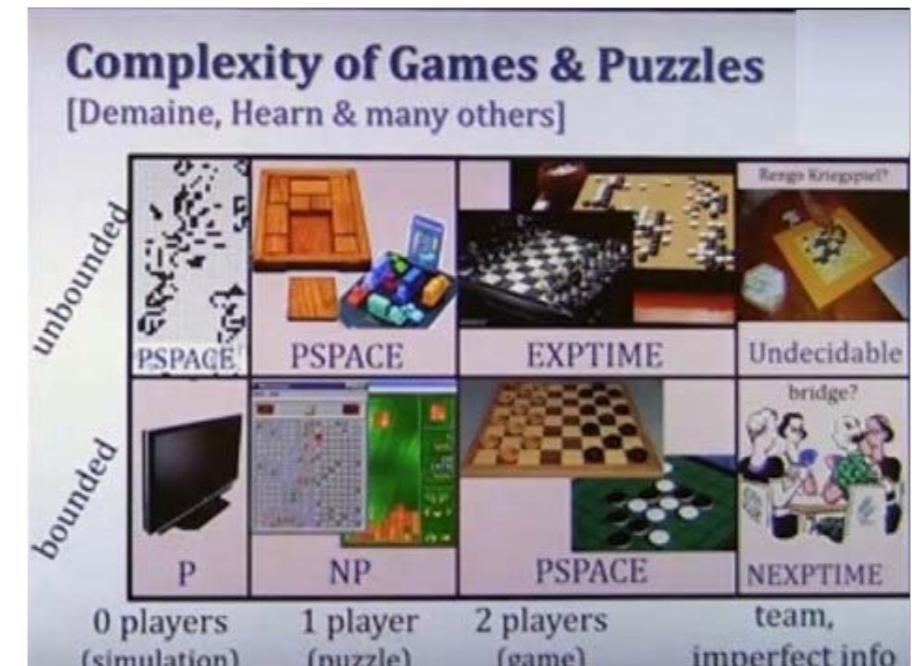
Computation Complexity

# Hard Games & Puzzles

Complexity of games & puzzles



Complexity of algorithmic problems



# NP-Complete Problems

- Decision Problem
- The class P
- The class NP
- Reduction between problems

# How Functions Grow

Algorithm	1	2	3	4	
Time function(ms)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	$2^n$
Input size( $n$ )	Solution time				
10	0.00033 sec.	0.0015 sec.	0.0013 sec.	0.0034 sec.	0.001 sec.
100	0.0033 sec.	0.03 sec.	0.13 sec.	3.4 sec.	$4 \times 10^{16}$ yr.
1,000	0.033 sec.	0.45 sec.	13 sec.	0.94 hr.	
10,000	0.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26

# Hanoi Tower Revisited

- It is **easy** to provide a recursive algorithm to resolve the problem of Hanoi Tower
  - The solution requires  $2^N-1$  moves of disc.
- It is **extremely difficult** to achieve the result for an input of moderate size
  - For the input of 64, it takes half a million years even if the Tibetan priest has superhuman strength to move a million discs in a second.

# Max Clique - an Example

- A complete subgraph of a graph  $G$  is called a clique, whose size is the number of vertices in it.
- Optimization problem: Find the maximal clique in a given graph  $G$ .
- Decision problem: Has  $G$  a clique of size at least  $k$  for some given  $k$ ?

# Decision Problem

- Statement of a decision problem
  - Part 1: instance description defining the input
  - Part 2: question stating the actual yes-or-no question
- A decision problem is a mapping from all possible inputs into the set {yes , no}

# Optimization v.s. Decision

- Usually, an optimization problem can be rephrased as a decision problem.
- For some cases, it can be proved that the decision problem can be solved in polynomial time if and only if the corresponding optimization problem can.
- We can make the statement that **if the decision problem cannot be solved in polynomial time then the corresponding optimization problem cannot either.**

# Max Clique Revisited

- The max clique problem can be solved in polynomial time iff. The corresponding decision problem can be solved in polynomial time.
  - If the size of a max clique can be found in time  $g(n)$ , the corresponding decision may be settled in that time of course.
  - If deciClique is algorithm for the decision problem with  $k$  in the complexity of  $f(n)$ , then we apply the algorithm at most  $n$  time, for  $k=n, n-1, \dots, 2, 1$ , and we can solve the optimization problem, and with the complexity no worse than  $nf(n)$ , which is polynomial only if  $f(n)$  is polynomial.

# Some Typical Decision Problems

- Graph coloring
  - Given an undirected graph  $G$  and a positive integer  $k$ , is there a coloring of  $G$  using at most  $k$  colors?
- Job scheduling with penalties
  - Given a group of jobs, each with its execution duration, deadline and penalty for missing the deadline, and a nonnegative integer  $k$ , is there a schedule with the total penalty bounded by  $k$ ?

# Some Typical Decision Problems

- Bin packing
  - Given  $k$  bins each of capacity one, and  $n$  objects with size  $s_1, \dots, s_n$ , (where  $s_i$  is a rational number in  $(0,1]$ ). Do the  $n$  objects fit in  $k$  bins?
- Knapsack
  - Given a knapsack of capacity  $C$ ,  $n$  objects with sizes  $s_1, \dots, s_n$  and “profits”  $p_1, \dots, p_n$ , and a positive integer  $k$ . Is there a subset of the  $n$  objects that fits in the knapsack and has total profit at least  $k$ ?  
**(Subset sum** as a simplified version)

# Some Typical Decision Problems

- CNF-Satisfiability
  - Given a CNF formula, is there a truth assignment that satisfies it?
- Hamiltonian cycles or Hamiltonian paths
  - Given an undirected graph G. Does G have a Hamiltonian cycle or Hamiltonian path?
- Traveling salesperson
  - Given a complete, weighted graph and an integer k, is there a Hamiltonian cycle with total weight at most k?

# Theory of NP-Completeness

- What it **cannot** do

- Provide a method of obtaining polynomial time algorithms for those “hard” problems
- Negate the existence of algorithms of polynomial complexity for those problems

- What it **can** do

- Show that many of the problems for which there is no known polynomial time algorithm are computationally related

# The Class P

- A **polynomially bounded algorithm** is one with its worst-case complexity bounded by a polynomial function of the input size.
- A **polynomially bounded problem** is one for which there is a polynomially bounded algorithm.
- The **class P** is the class of decision problems that are polynomially bounded.

# Notes on the Class P

- Class P has a too broad coverage
  - In the sense that not every problems in P has an acceptable efficient algorithm.
- However
  - The problem not in P must be extremely expensive and probably impossible to solve in practice.
  - The problems in P have nice “closure” properties for algorithm integration.
  - The property of being in P is independent of the particular formal model of computation used.

# Nondeterministic Algorithm

```
void nondetA(String input)
    String s=genCertif(); ←
    Boolean CheckOK=verifyA(input,s);
    if (checkOK)
        Output "yes";
    return;
```

Phase 1 Guessing:  
generating **arbitrarily**  
“certificate”, i.e.  
proposed solution

The algorithm may  
behave differently on  
the same input in  
different executions:  
“yes” or “no output”.

Phase 2 Verifying: determining if  $s$  is a valid  
description of a object for answer, and  
satisfying the criteria for solution

# Nondeterministic Algorithm

- For a particular decision problem with input  $x$ :
  - The answer computed by a nondeterministic algorithm is defined to be yes if and only if there is **some** execution of the algorithm that gives a yes output.
  - The answer is no, if for **all**  $s$ , there is no output.

# Nondeterministic Algorithm

```
void nondetSearch(int k; int[ ] S)
    int i =genCertif();
    if (S[i]==k)
        Output "yes";
    return;
```

In  $O(n)$

Note:  $\Omega(n \log n)$  for  
deterministic algorithm

In  $O(1)$

Note:  $\Omega(n)$  for  
deterministic algorithm

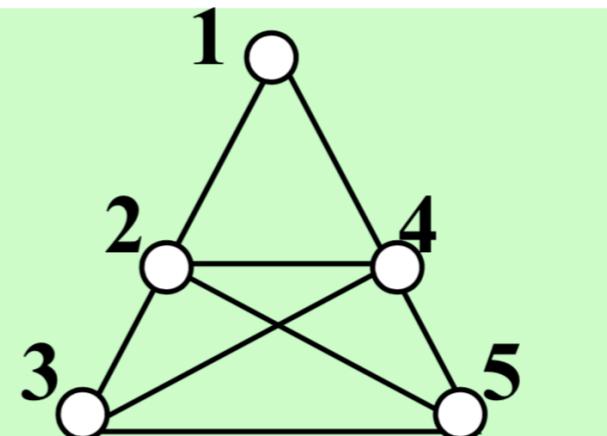
```
void nondetSort(int[ ] S; int n)
    int i, j; int[ ] out=0;
    for i =1 to n do
        j= genCertif();
        if out[j]≠0 then return;
        out[j]=S[i ];
    for i =1 to n-1 do
        if out[i ]>out[i+1] then return;
    S=out;
    Output(yes);
    return
```

# Nondeterministic Graph Coloring

Problem instance  $G$

Input string:

4,5,(1,2)(1,4)(2,4)(2,3)(3,5)(2,5)(3,4)(4,5)



$s$	Output	Reason	verified by phase 2
RGRBG	false	$v_2$ and $v_5$ conflict	←
RGRB	false	Not all vertices are colored	
RBYGO	false	Too many colors used	
RGRBY	true	A valid 4-coloring	→ (G,4) → yes
R%*,G@	false	Bad syntax	

generated by phase 1

# The Class NP

- A **polynomial bounded nondeterministic algorithm** is one for which there is a (fixed) polynomial function  $p$  such that for each input of size  $n$  for which the answer is yes , there is some execution of the algorithm that produces a yes output in at most  $p(n)$  steps.
- The **class NP** is the class of decision problems for which there is a polynomial bounded nondeterministic algorithm.

# The Class NP

- NP means Non-deterministic P
  - From “deterministic” to “non-deterministic”
  - From “solve a problem” to “verify the answer of a problem”
- What does NP indicate?
  - Harder problems
  - Not too hard
  - At least, you can quickly understand the answer

# Proof of Being in NP

- Graph coloring is in NP
  - Description of the input and the certificate
  - Properties to be checked for an answer “yes”
    - There are  $n$  colors listed:  $c_1, c_2, \dots, c_n$  (not necessarily different)
    - Each  $c_i$  is in the range  $1, \dots, k$
    - Scan the list of edges to see if a conflict exists
  - Proving that each of the above statement can be checked in polynomial time.

# CLIQUE is in NP

```
void nondeteClique(graph G; int n, k)
set S=∅;
for int i=1 to k do
    int t=genCertif();
    if t∈S then return;
    S=S∪{t};
for all pairs (i,j) with i,j in S and i≠j do
    if (i,j ) is not an edge of G
        then return;
Output("yes");
```



In  $O(n)$



In  $O(n^2)$

So, we have an algorithm for the maximal clique problem with the complexity of  $O(n+k^2)=O(n^2)$

# SAT

An example of propositional conjunctive normal form (CNF) is like this:

$$(p \vee q \vee s) \wedge (\overline{q} \vee r) \wedge (\overline{p} \vee r) \wedge (\overline{r} \vee s) \wedge (\overline{p} \vee \overline{s} \vee \overline{q})$$

## Satisfiability Problem

Given a CNF formula, is there a truth assignment that satisfies it?

In other words, is there a assignment for the set of propositional variable in the CNF, such that the value of the formula is **true**.

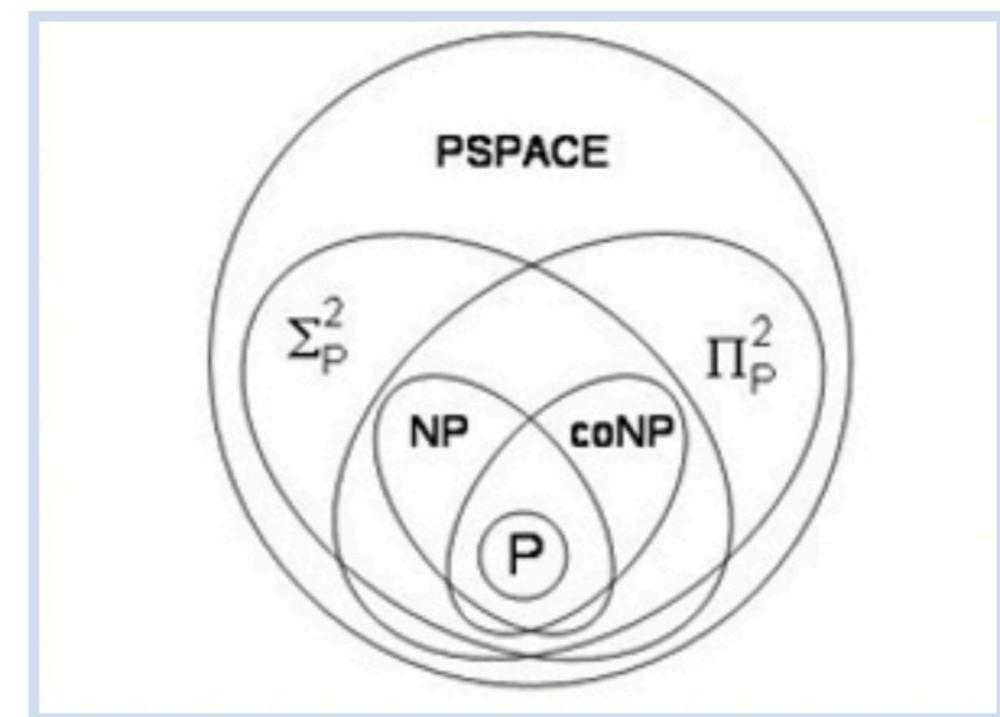
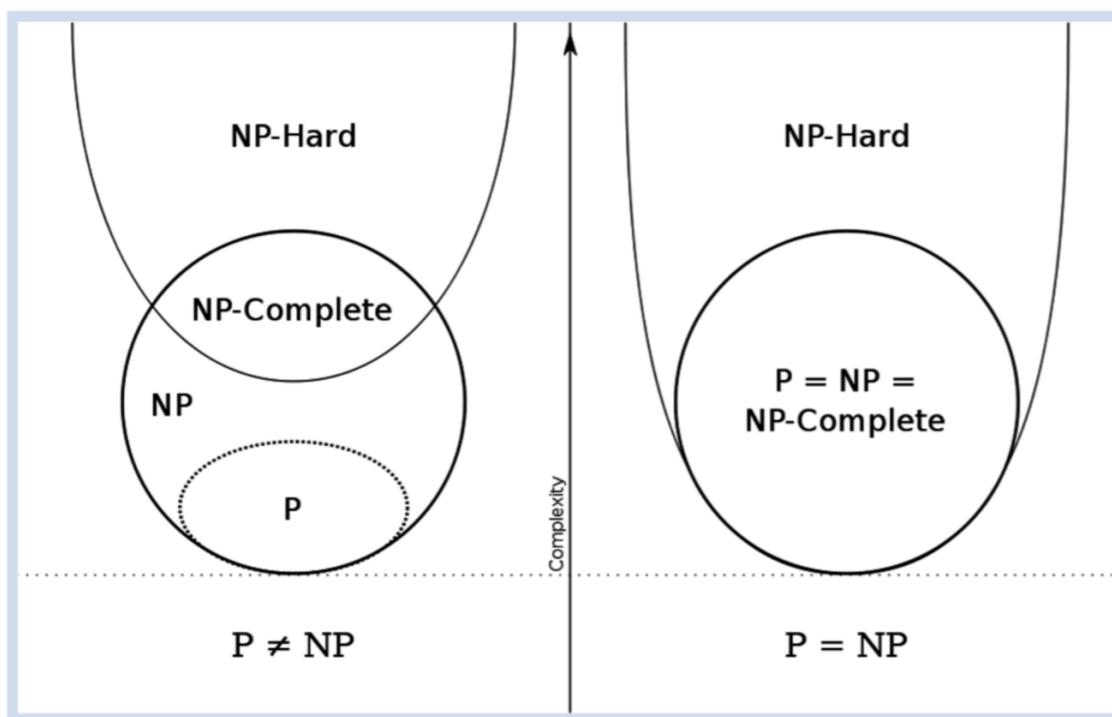
```
void nondetSat(E, n)
    boolean p[ ];
    for int i =1 to n do
        p[i ]= genCertif(true, false);
    if E(p[1], p[2], ..., p[n])=true
        then Output("yes");
```

So, the problem is in **NP**

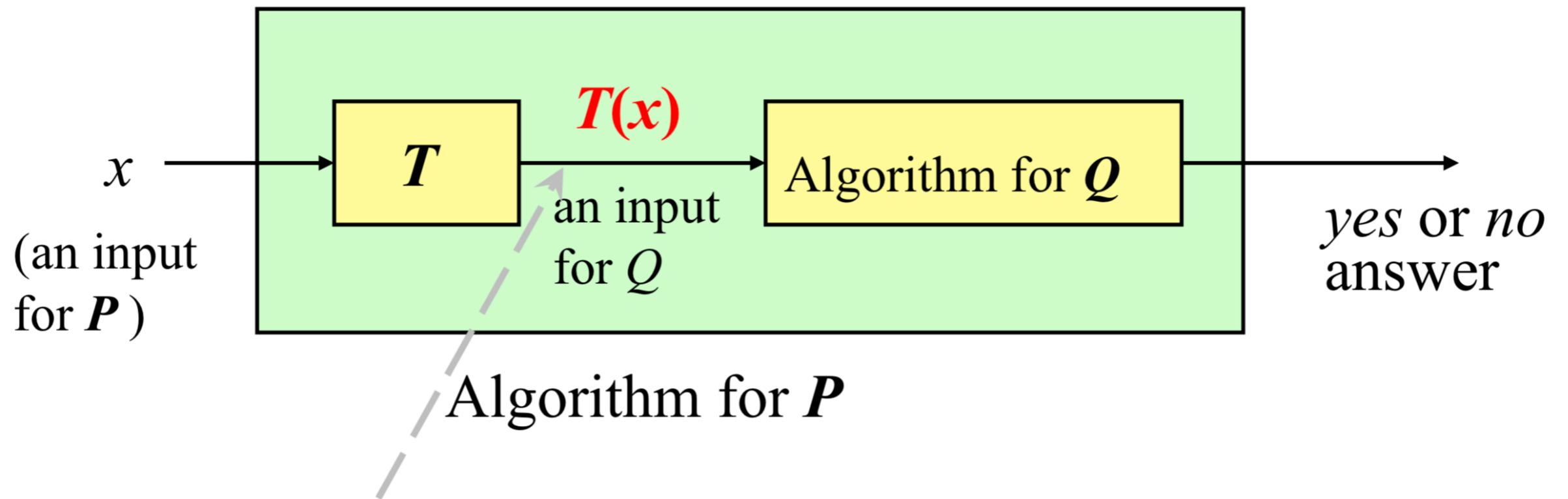
# Relation between P and NP

- An deterministic algorithm for a decision problem is a special case of a nondeterministic algorithm, which means:  $P \subseteq NP$ 
  - The deterministic algorithm is looked as the phase 2 of a nondeterministic one, which always ignore the s the phase 1 has written.
- Intuition implies that **NP** is a much larger set than **P**.
  - The number of possible s is exponential in  $n$ .
  - No one problem in **NP** has been proved not in **P**.

# P, NP and ...



# Reduction



The correct answer for  $P$  on  $x$  is *yes if and only if* the correct answer for  $Q$  on  $T(x)$  is *yes*.

# Polynomial Reduction

- Let  $T$  be a function from the input set for a decision problem  $P$  into the input set for  $Q$ .  $T$  is a polynomial reduction from  $P$  to  $Q$  if:
  - $T$  can be computed in polynomial bounded time
  - $x$  is a *yes* input for  $P \rightarrow T(x)$  is a *yes* input for  $Q$
  - $x$  is a *no* input for  $P \rightarrow T(x)$  is a *no* input for  $Q$

**An example:**

**$P$ :** Given a sequence of Boolean values, does at least one of them have the value true?

**$Q$ :** Given a sequence of integers, is the maximum of them positive?

$T(x_1, \dots, x_n) = (y_1, \dots, y_n)$ ,  
where:  $y_i=1$  if  $x_i=\text{true}$ , and  
 $y_i=0$  if  $x_i=\text{false}$

# Relation of Reducibility

- Problem  $P$  is **polynomially reducible** to  $Q$  if there exists a polynomial reduction from  $P$  to  $Q$ , denoted as:  
 $P \leq_P Q$
- If  $P \leq_P Q$  and  $Q$  is in  $\text{P}$ , then  $P$  is in  $\text{P}$ 
  - The complexity of  $P$  is the sum of  $T$ , with the input size  $n$ , and  $Q$ , with the input size  $p(n)$ , where  $p$  is the polynomial bound on  $T$ ,
  - So, the total cost is:  $p(n) + q(p(n))$ , where  $q$  is the polynomial bound on  $Q$ .

(If  $P \leq_P Q$ , then  $Q$  is at least as “hard” to solve as  $P$ )

**Thank you!**  
**Q & A**