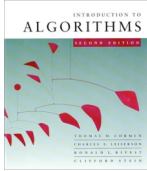


# Introduction to Algorithms


6.046J/18.401



**Lecture 23**

Prof. Piotr Indyk


## P vs NP (interconnectedness of all things)



- A whole course by itself
- We'll do just two lectures
- More in 6.045, 6.840J, etc.

© Piotr Indyk      Introduction to Algorithms      May 6, 2008    L20.2


## Have seen so far



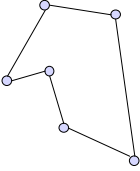
- Algorithms for many interesting problems:
  - Running times  $O(nm^2), O(n^2), O(n \log n), O(n), \dots$
  - I.e., polynomial in the input size
- Can we solve all (or most of) interesting problems in polynomial time ?
- Not really...

© Piotr Indyk      Introduction to Algorithms      May 6, 2008    L20.3

## Example difficult problem




- Traveling Salesperson Problem (TSP)
  - Input: undirected graph with lengths on edges
  - Output: shortest tour that visits each vertex exactly once
- Best known algorithm:  $O(n 2^n)$  time.

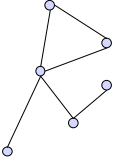


© Piotr Indyk      Introduction to Algorithms      May 6, 2008    L20.4

## Another difficult problem




- Clique:
  - Input: undirected graph  $G=(V,E)$
  - Output: largest subset  $C$  of  $V$  such that every pair of vertices in  $C$  has an edge between them
- Best known algorithm:  $O(n 2^n)$  time



© Piotr Indyk      Introduction to Algorithms      May 6, 2008    L20.5

## What can we do ?



- Spend more time and money designing efficient algorithms for those problems
  - People tried for a few decades, no luck
  - Outstanding \$1000,000 prize for finding one
  - It seems very likely that such algorithms do not exist
- Prove there is **no** polynomial time algorithm for those problems
  - Would be great
  - Seems *really* difficult
  - Best lower bounds for “natural” problems:
    - $\Omega(n^2)$  for restricted computational models
    - $\Omega(n)$  for unrestricted computational models

© Piotr Indyk      Introduction to Algorithms      May 6, 2008    L20.6



## What else can we do ?

- Show that those hard problems are essentially equivalent.  
I.e., if we can solve one of them in poly time, then all others can be solved in poly time as well.
- Works for at least **few thousand** hard problems

© Piotr Indyk

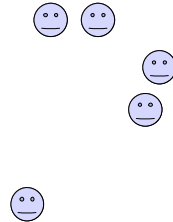
Introduction to Algorithms

May 6, 2008 L20.7



## The benefits of equivalence

- Combines research efforts
- If one problem has polytime solution, then all of them do



© Piotr Indyk

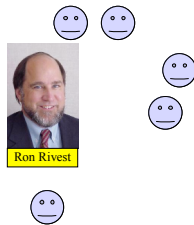
Introduction to Algorithms

May 6, 2008 L20.8



## A more realistic scenario

- Once an exponential **lower bound** is shown for one problem, it holds for all of them
- But someone *is* happy...



© Piotr Indyk

Introduction to Algorithms

May 6, 2008 L20.9



## Summing up

- If we show that a problem  $\Pi$  is equivalent to a few thousand other well studied problems without efficient algorithms, then we get a very strong evidence that  $\Pi$  is hard.
- We need to:
  1. Identify the class of problems of interest
  2. Define the notion of equivalence
  3. Prove the equivalence(s)

© Piotr Indyk

Introduction to Algorithms

May 6, 2008 L20.10



## 1. Class of problems (informally)

- Decision problems: answer YES or NO. E.g., "is there a tour of length  $\leq K$ " ?
- Solvable in *non-deterministic polynomial* time:
  - Intuitively: the solution can be **verified** in polynomial time
  - E.g., if someone gives as a tour  $T$ , we can verify if  $T$  is a tour of length  $\leq K$ .
- Therefore, TSP is in NP.

© Piotr Indyk

Introduction to Algorithms

May 6, 2008 L20.11



## 1. Class of problems: NP

- A problem  $\Pi$  is solvable in poly time (or  $\Pi \in P$ ), if there is a poly time algorithm  $V(\cdot)$  such that for any input  $x$ :

$$\Pi(x)=\text{YES} \text{ iff } V(x)=\text{YES}$$

- A problem  $\Pi$  is solvable in *non-deterministic* poly time (or  $\Pi \in NP$ ), if there is a poly time algorithm  $V(\cdot, \cdot)$  such that for any input  $x$ :

$$\Pi(x)=\text{YES} \text{ iff there exists a certificate } y \text{ of size } \text{poly}(|x|) \text{ such that } V(x,y)=\text{YES}$$

© Piotr Indyk

Introduction to Algorithms

May 6, 2008 L20.12

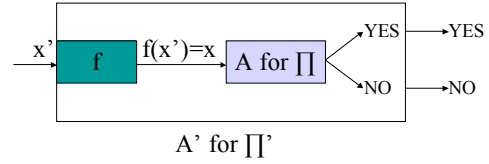


## Examples of problems in NP

- Is “Does there exist a clique in  $G$  of size  $\geq K$ ” in NP?  
Yes:  $\forall(x,y)$  interprets  $x$  as a graph  $G$ ,  $y$  as a set  $C$ , and checks if all vertices in  $C$  are adjacent and if  $|C| \geq K$
- Is **Sorting** in NP?  
No, not a decision problem.
- Is “Sortedness” in NP?  
Yes: ignore  $y$ , and check if the input  $x$  is sorted.



## 2. Reductions: $\Pi'$ to $\Pi$



## 2. Reductions (formally)

- $\Pi'$  is poly time reducible to  $\Pi$  ( $\Pi' \leq \Pi$ ) iff there is a poly time function  $f$  that maps inputs  $x'$  to  $\Pi'$  into inputs  $x$  of  $\Pi$ , such that for any  $x'$

$$\Pi'(x') = \Pi(f(x'))$$

- Fact 1: if  $\Pi \in P$  and  $\Pi' \leq \Pi$  then  $\Pi' \in P$
- Fact 2: if  $\Pi \in NP$  and  $\Pi' \leq \Pi$  then  $\Pi' \in NP$
- Fact 3: if  $\Pi' \leq \Pi$  and  $\Pi'' \leq \Pi'$  then  $\Pi'' \leq \Pi$