# Week 12 - 02
# P & NP

50.004 Introduction to Algorithm
Dr. Subhajit Datta
Based on original slides by Dr. Simon LUI
ISTD, SUTD

# How important is P & NP?

- People have been thinking about it since 1956
- No one has seen able to solve it yet!
- Perhaps you will solve it
- So pay attention to these lectures ☺
- Simply,
  - If you can prove P=NP or P!=NP
  - That's a big deal!

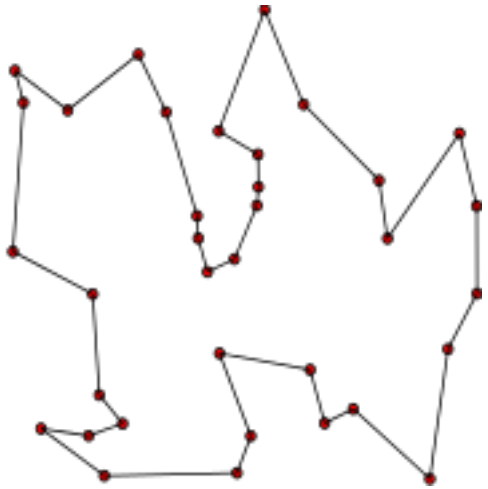
# TRACTABLE PROBLEMS
# &
# SOLVABLE PROBLEMS

# Tractable Problems

- Tractable problems = problems that can be solved in **polynomial time**.
  - shortest path
  - sort *n* number
  - etc

# Intractable Problems

- Problems that cannot be solved in **polynomial time**.
- e.g. The travelling salesman problem (TSP)
  - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?



# Unsolvable Problems?

- Are there computational problems that cannot be solved?
- Is a python program P syntactically correct?
  - If it is correct: yes, we can tell it is correct in polynomial time
  - If it is not correct: …. How long does it take to CONFIRM that it is not correct? Can be forever!

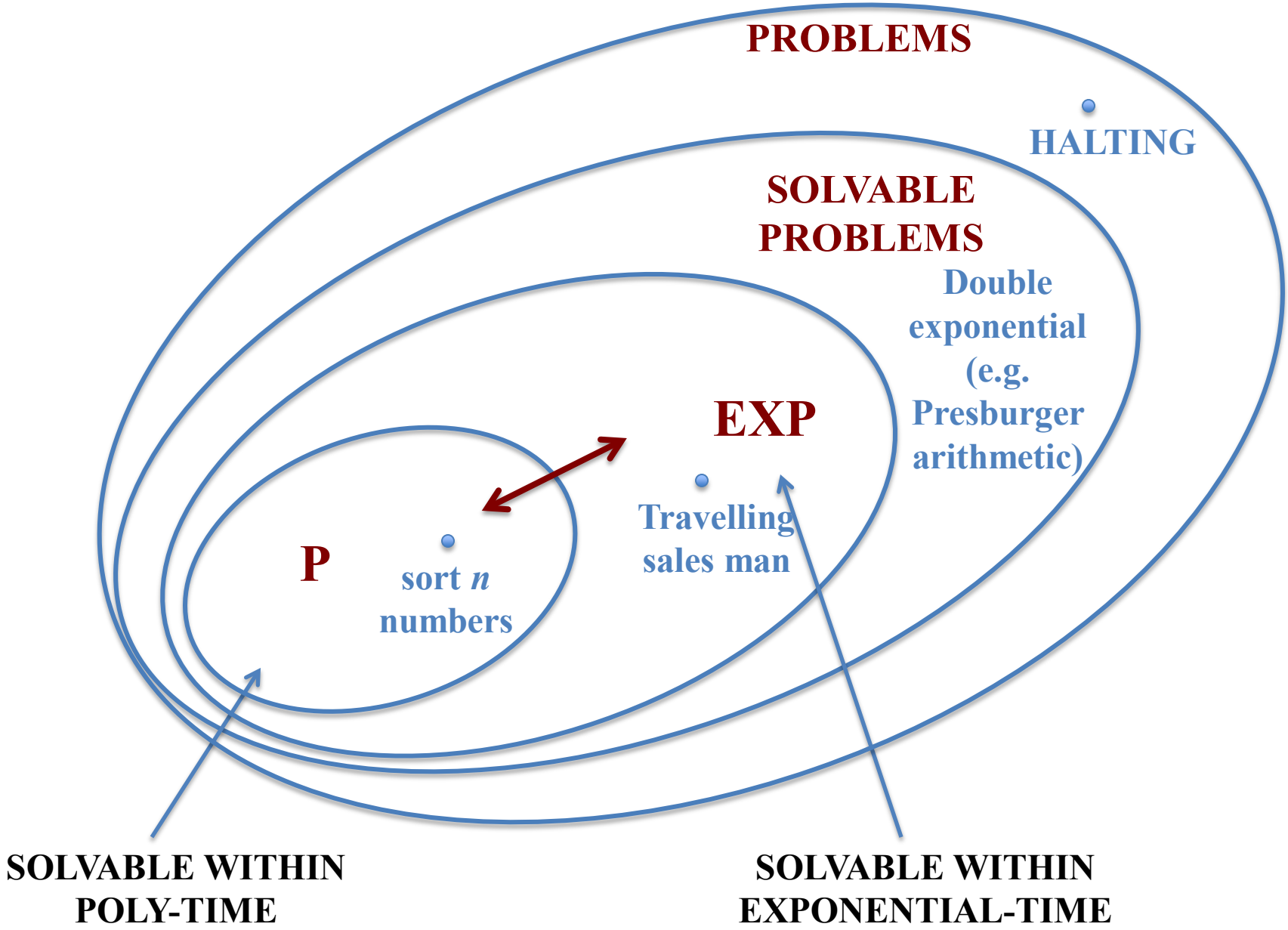
  - Function(P) = 1 if P is syntactically correct
  - Function(P) = 0 if P is syntactically incorrect
  - Function() is unsolvable. Since we can never confirm any answer which is 0

# Halting problem

- H(P,X)=1 if program P on input X terminates
- H(P,X)=0, if program P on input X runs forever

- Program A(P)
  - Input: Program P
  - if H(P,P)=1, then enter infinite loop
  - else if H(P,P)=0, then stop

- Question: Does A(A) terminate?
  - suppose it does, then … H(A,A)=0 (A will run forever)
  - suppose it does not, then…H(A,A)=1 (A will stop)
  - Contradiction! So there is no algorithm that can solve this problem...


PROBLEMS
HALTING
SOLVABLE PROBLEMS
Double exponential (e.g. Presburger arithmetic)
EXP
Travelling sales man
P
sort *n* numbers
SOLVABLE WITHIN POLY-TIME
SOLVABLE WITHIN EXPONENTIAL-TIME

# What we will learn…

- Classification of problems: P, EXP, unsolvable
- Problems for which we can't decide yet
- the class NP
- the P vs NP question

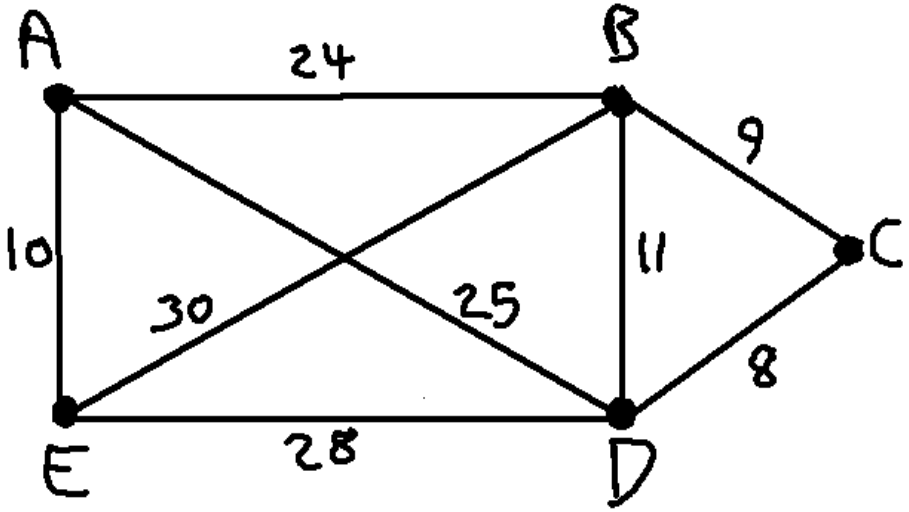
# PROVING A PROBLEM IS UNSOLVABLE IN P

# 1. Knapsack Problem

- **Input:**
  - Knapsack of size $S$
  - Collection of $n$ items
    - Item $i$ has (integer) size $s_i$ and (integer) value $v_i$
- DP algorithm: O($n$ $S$)
- In Polynomial time

# 2. Traveling Salesperson Problem (TSP)

- **Input:** Undirected graph with lengths on edges.
- **Output:** Shortest tour that visits each vertex exactly once.
- Best known algorithm: $O(n^2 2^n)$ time.
- In Exponential time



# 3. Unsolvable problem

- Can we prove that a certain problem is not in P?
  - (Problem in P: problem solvable in polynomial time)

# Proving a negative

- How to prove there is no polynomial time algorithm for a problem?
  - i.e show that there is no algorithm of time O(n), OR $O(n^2)$, OR $O(n^3)$…

Short Answer: We don't know how to prove

We don't even have general technique to show that there is no O(n) algorithm

# Proving a negative

- How to prove no perpetual motion machine?
  - We can prove one exists by building it.
  - We can't prove none exists.
  - Especially if the only "evidence" is that we tried and failed.
- Many have tried to build one and failed
  - But an absolute evidence is impossible

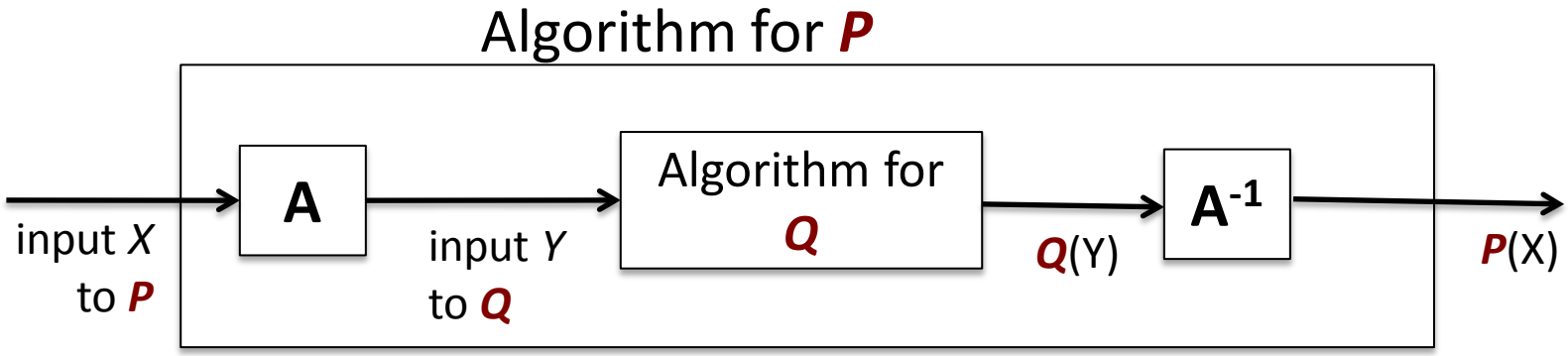
# POLYNOMIAL REDUCTION

# Polynomial-Time Reduction

- A huge number of fundamental problems are hard
- Show that these fundamental problems are "computationally equivalent" to each other
  - Hence all of them appear to be different representation of one really hard problem
  - Hence we can say “they are probably not solvable in P”
- Technique: Polynomial-time reductions

- Problem ***P*** can be reduced to problem ***Q*** in polynomial-time
- if an arbitrary instances of problem ***P*** can be solved using:
  - Polynomial number of standard computational steps, plus
  - Polynomial number of calls that solves problem ***Q*** .
- Notation: $\boldsymbol{P} \leq_P \boldsymbol{Q}$
- We can solve ***P*** in polynomial time IF we can solve ***Q*** in polynomial time!

# Polynomial-Time Reduction




A: Algorithm that in polynomial time transforms the input for P to an input for Q

$A^{-1}$: Algorithm that in polynomial time transforms the answer for Q to an answer for P

- Classify problems according to relative difficulty.
- Design algorithms: If $P \leq_P Q$ and $Q$ can be solved in polynomial-time, then $P$ can also be solved in polynomial time.
- Establish intractability: If $P \leq_P Q$ and $P$ cannot be solved in polynomial-time, then $Q$ cannot be solved in polynomial time.
- Establish equivalence. If $P \leq_P Q$ and $Q \leq_P P$ , we use notation $P \cong_P Q$.
- $P \leq_P Q$ : "$Q$ is at least as hard as $P$"

# Algorithmic Hardness "Proof"

- Suppose you want evidence that there is no poly-time algorithm for your problem ***Q***.
- Take a problem ***P*** where many scientists have tried and failed to find a poly-time algorithm.
- Find a poly-time reduction $Q \leq_P P$ of the solution of ***P*** to the solution of ***Q***.
- If no poly-time algorithm for ***P***, then no poly-time algorithm for ***Q***.
- Evidence from scientists that ***P*** is hard becomes evidence that ***Q*** is hard.