# P, NP Theory I

Prof Devesh C Jinwala, PhD
Professor, Department of Computer Science and Engineering
http//www.svnit.ac.in/dcj/

**Indian Institute of Technology Jammu**

# Contents

- What is an Algorithm ? Solving problems algorithmically
- Classifying the problems
- A Motivating Example
- Some Hard Problems
- Reductions
- Non-determinism
- Class P, NP
- NP Complete Problems
- Concluding Remarks

# Problems you must have studied so far

- Sorting
- Searching
- Minimum spanning tree
- Fractional Knapsack problem
- Shortest paths in a graph
- Primality testing
- Towers of Hanoi
- Travelling Salesperson problem
- Program termination
- … … … … … … …

# Tutorial Time Complexity

- What is the worst case time complexity of the algorithms to solve these problems viz.

- Finding maximum/minimum in an array
- Finding the sum of n element array
- Comparison based Sorting
- Binary search
- Kruskal's and Prim's MST algorithms
- Fractional Knapsack problem
- Container Loading problem
- Activity Selection
- Job scheduling to minimize average completion time
- Dijkstra and Bellman Ford SSSP Algorithms
- Floyd Warshall's APSP Algorithms

# Polynomial or Intractable ?

| | $n$ | $n \lg n$ | $N^2$ | $N^3$ | $1.5^n$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| **n=10** | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 4 sec |
| **n=30** | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 18 min | $10^{25}$ yrs |
| **n=50** | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 11 min | 36 yrs | very long |
| **n=100** | < 1 sec | < 1 sec | < 1 sec | 1 sec | 12.89 yrs | $10^{17}$ yrs | < 4 sec |
| **n=1000** | < 1 sec | < 1 sec | 1 sec | 18 min sec | very long | very long | very long |
| **n=10K** | < 1 sec | < 1 sec | 2 min | 12 days | very long | very long | very long |
| **n=100K** | < 1 sec | 2 sec | 3 hrs | 32 yrs | very long | very long | very long |
| **n=1M** | 1 sec | 20 sec | 12 days | 31.71 yrs | very long | very long | very long |

# Efficient Solutions

- Efficient solutions
  - A solution to a problem is generally considered to be efficient if its running time is bounded by a polynomial in input n i.e.
  - ∃ a constant c, such that the running time of the algorithm T(n) is
    - $O(n^c)$, where c is some positive integer and n is the input size.
    - c could be either constant degree – $n^2$, $n^3$, $n^4$ … OR $n^{\log n}$
  - Searching `(O lg n)`, Sorting `(O(n lg n))`, Polynomial evaluation `(O(n)` …….

effectively Polynomial time …..

# Why polynomial only ?

- Three reasons
  - Machine Independence
  - Limited to $O(n^2)$
  - Closure property

# Can every problem be solved efficiently?

- Can every problem be solved efficiently ?

# Inefficient Solutions

- Non-efficient solutions
  - A solution to a problem is generally considered to be non-efficient if it is found using the brute force approach.
  - No $O(n^k)$ algorithm is known
  - $O(n^n)$ **or** $O(n!)$ **or** $O(k^n)$ time required
  - Traveling Salesperson $(O(n^2 2^n))$, 0/1 Knapsack $(O(2^{n/2}))$ ……

  - Though this may not be true always……

# Inefficient Solutions…

- Why are inefficient solutions are those that are obtained using the brute-force ?
  - Bruteforce algorithms
    - search exhaustively through the entire solution space i.e……???
    - e.g. how do we carry out sorting using brute force ?
    - complexities of sorting – comparison based, non-comparison based, non-comparison based brute force…
  - If the input size is n, typical time taken is $2^n$.

# Classifying Problems

Tractable problems = Efficiently solvable ????

Intractable problems = Inefficiently solvable ????

# Why learn classifying problems ?

- How do we know whether the problem can be solved efficiently or not ?
- What if we encounter a new problem tomorrow ?

Decidable or Undecidable problem ??

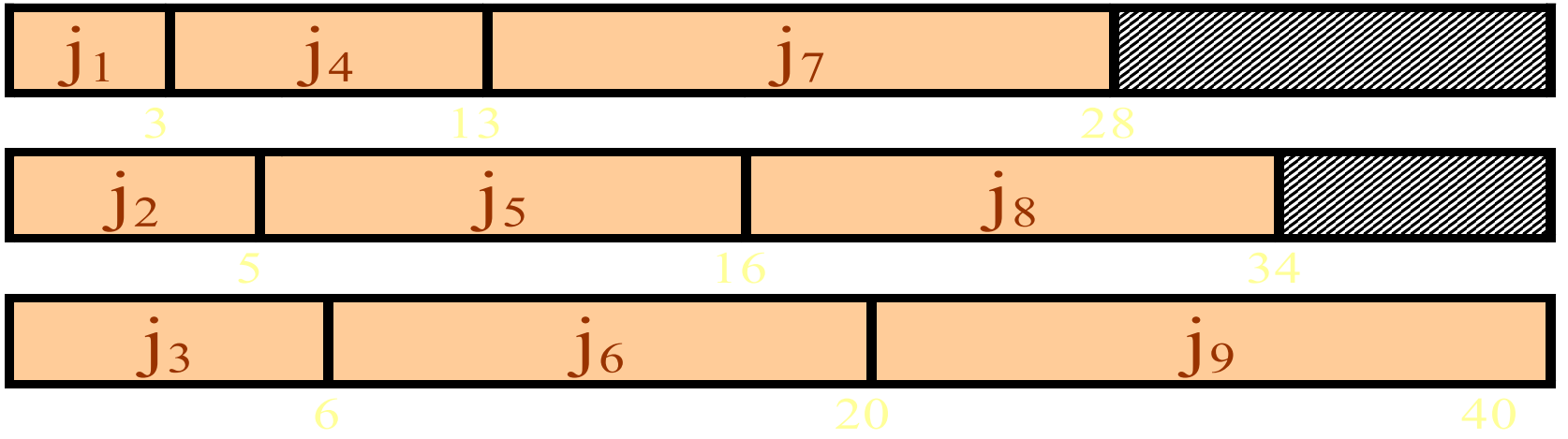
Tractable or Intractable problem ??

# Contents

# Motivating Example

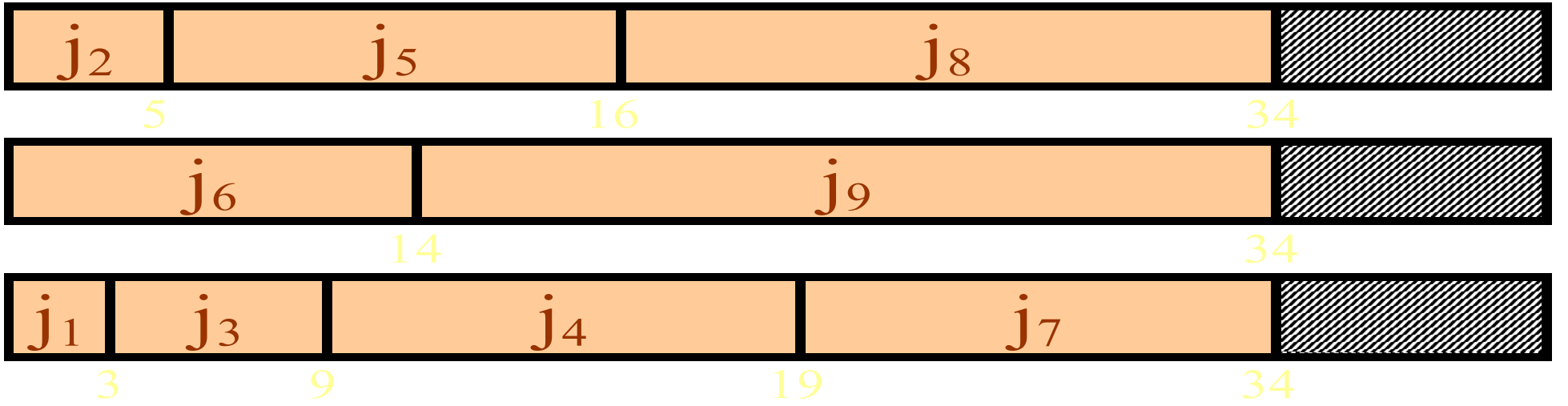
- Machine scheduling to minimize the finish times of machines used.
- That is……
  - Processes
    - $j_1, j_2, j_3 \ldots j_m$ with running times $t_1, t_2, t_3, \ldots t_o$
    - to be scheduled on specified m no of machines $m_1, m_2, m_3, \ldots\ldots m_m$ such that
      - no machine processes more than one process at a time
      - no process is executed by more than one machine
      - there is non-preemptive scheduling
      - the average response time is minimized.

# Motivating Example…..




| $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ |
|---|---|---|---|---|---|---|---|---|
| $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ | $j_7$ | $j_8$ | $j_9$ |
| 3 | 5 | 6 | 10 | 11 | 14 | 15 | 18 | 20 |
| 3 | 5 | 6 | 13 | 16 | 20 | 28 | 34 | 40 |

# Motivating Example…..




| $M_3$ | $M_1$ | $M_3$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ |
|---|---|---|---|---|---|---|---|---|
| $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ | $j_7$ | $j_8$ | $j_9$ |
| 3 | 5 | 6 | 10 | 11 | 14 | 15 | 18 | 20 |

# Motivating Example

- Machine scheduling to minimize the finish times of machines used.
- That is……
  - Processes
    - $j_1, j_2, j_3 \ldots j_m$ with running times $t_1, t_2, t_3, \ldots t_o$
    - to be scheduled on specified m no of machines $m_1, m_2, m_3, \ldots\ldots m_m$ such that
      - no machine processes more than one process at a time
      - no process is executed by more than one machine
      - there is non-preemptive scheduling
      - the final completion time is minimized.

# Motivating Example…..

- Jobs $j_1, j_2, j_3 \ldots j_n$ with running times $t_1, t_2, t_3, \ldots t_n$
  - to be scheduled on say TWO processors - i.e. multiprocessors
  - such as to minimize the final completion time, i.e. last job finishes earliest
- That is, given the jobs $j_1, j_2, j_3 \ldots j_7$ with running times 2,100,2,100,2,100,2
  - and we schedule all the ODD jobs on P1
    - and the even jobs on P2, then, the last job is …… & finishes at time …….
- We wish to improve upon this finish time of whatever is the last job to execute i.e.
  - we wish to evenly distribute the jobs across the two processors s.t. ……

# Motivating Example…..

- How to devise the algorithm to solve the problem ?
- Algorithm Attempt #1
  - For the given job mix, put the first job on P1, second job on P2 i.e.

| $P_1$ | $P_2$ |
|---|---|
| $j_1$ | $j_2$ |
| | |

  - When scheduling $j_3$, schedule it on the least lightly loaded processor i.e.
    - Check if $t_1 > t_2$, if so put $j_3$ on $P_2$ otherwise on $P_1$.
- What would be the schedule in the previous example with this approach ?

# Motivating Example…..

- For the job mix $j_1, j_2, j_3 \ldots j_7$ with running times 2,100,2,100,2,100,2,
  - Scheduling as per the policy just defined we get the schedule as

| $P_1$ | $P_2$ | Comment |
|---|---|---|
| $j_1(2)$ | $j_2(100)$ | |
| $j_3(2)$ | | $t_1<t_2$ |
| $j_4(100)$ | | $t_1+t_3<t_2$ |
| | $j_5(2)$ | $t_2<t_1+t_3+t_4$ |
| | $j_6(100)$ | $t_2+t_5<t_1+t_3+t_4$ |
| $j_7(2)$ | | ..... |

- What is the earliest finish time ?

# Motivating Example…..

- However, will this work always ?
- Think of a counterexample that shows it doesN't work….
- Algorithm Attempt #2
  - Sort the jobs in ascending order of their execution times…. and then
  - Follow the same approach as before i.e.
    - for the given job mix, put the first job on P1, second job on P2 i.e.
    - When scheduling $j_3$, schedule it on the least lightly loaded processor i.e.
      - Check if $t_1 > t_2$, if so put $j_3$ on $P_2$ otherwise on $P_1$.
  - The schedule that we obtain for the previous example with this approach is ……

# Motivating Example…..

- For the job mix $j_1, j_2, j_3 \ldots j_7$ with running times 2,100,2,100,2,100,2,
  - Scheduling as per the policy just RE-defined we get the schedule as

| $P_1$ | $P_2$ | Comment |
|---|---|---|
| $j_1(2)$ | $j_3(2)$ | |
| $j_5(2)$ | | $t_1 <= t_3$ |
| | $j_7(2)$ | $t_1+t_5<t_3$ |
| $j_2(100)$ | | $t_1+t_5<=t_3+t_7$ |
| | $j_4(100)$ | $t_3+t_7<t_1+t_5+t_2$ |
| $j_6(100)$ | | ..... |

- What is the earliest finish time ?

# Motivating Example…..

- However, this approach also does not work on all the inputs as seen before OR
- Try the same on jobs with running times 2,2,2,3,3.

| $P_1$ | $P_2$ | Comment |
|---|---|---|
| $j_1(2)$ | $j_2(2)$ | |
| $j_3(2)$ | | $t_1<=t_3$ |
| | $j_4(3)$ | $t_1+t_3<t_2$ |
| $j_5(3)$ | | $t_1+t_3+t_5<=t_2+t_4$ |

| $P_1$ | $P_2$ | Comment |
|---|---|---|
| $j_1(2)$ | $j_4(3)$ | |
| $j_2(2)$ | $j_5(3)$ | OPTIMAL SCHEDULE |
| $j_3(2)$ | | |
| | | |

- Thus, the observation is……..
  - No heuristics would work on ALL the inputs.

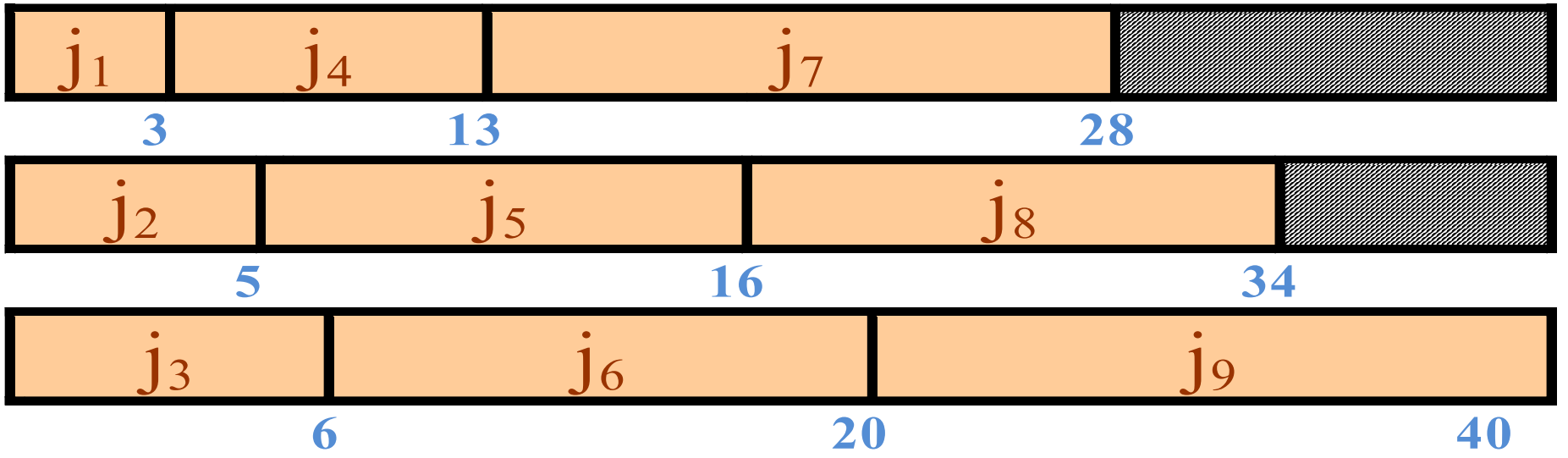
# Motivating Example…..Generalized version

- Jobs $j_1, j_2, j_3 \ldots j_n$ with running times $t_1, t_2, t_3, \ldots t_n$
  - to be scheduled on say three processors multiprocessors
  - such as to minimize the final completion time, i.e. last job finishes earliest

| $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ | $j_7$ | $j_8$ | $j_9$ |
|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 10 | 11 | 14 | 15 | 18 | 20 |

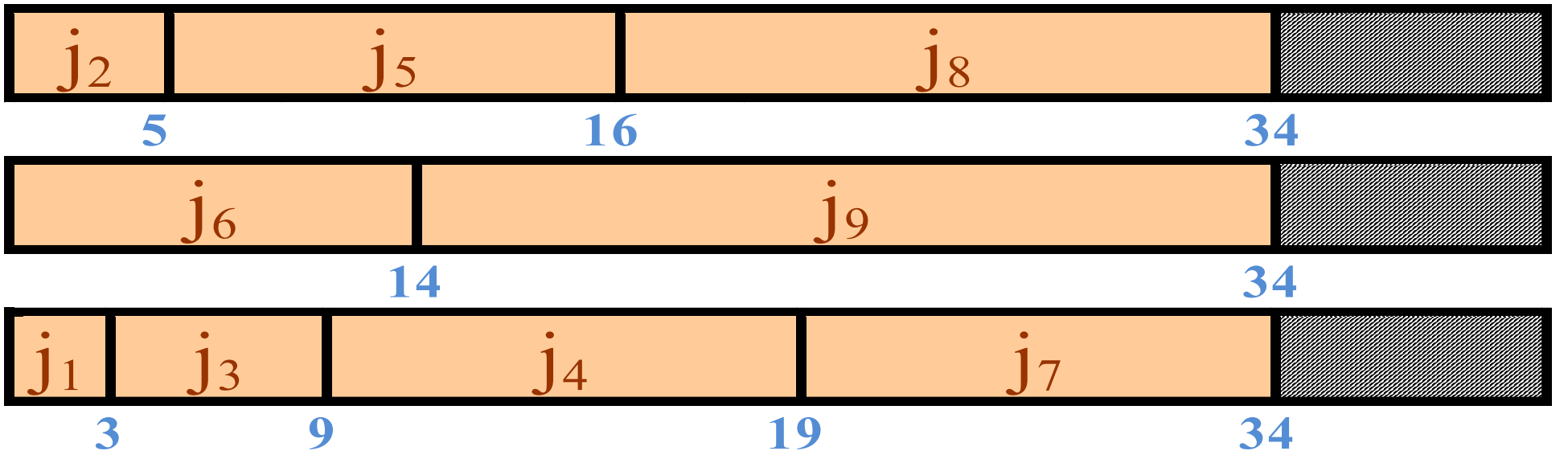
- A typical schedule and time completion of each job

# Motivating Example…..




| $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ |
|---|---|---|---|---|---|---|---|---|
| $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ | $j_7$ | $j_8$ | $j_9$ |
| 3 | 5 | 6 | 10 | 11 | 14 | 15 | 18 | 20 |
| 3 | 5 | 6 | 13 | 16 | 20 | 28 | 34 | 40 |

# Motivating Example…..Generalized version




| $M_3$ | $M_1$ | $M_3$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ |
|---|---|---|---|---|---|---|---|---|
| $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ | $j_7$ | $j_8$ | $j_9$ |
| 3 | 5 | 6 | 10 | 11 | 14 | 15 | 18 | 20 |

# Motivating Example…..

- How was the  schedule earlier obtained…?????
- Can we apply the brute force approach ?

# Motivating Example…Time Complexity

- The given problem
  - will take exponential time complexity and
  - is not feasibly solvable with the existing computational power.
- So, now what is the rationale behind exploring the NP theory ?
  - The rationale is…….

# Another Example

- Consider Time table scheduling problem
  - Say there are two subjects, four teachers and there are four lectures of 1 hour each everyday.
  - Design an algorithm to solve this problem……

# Classifying Problems…

Tractable problems = Efficiently solvable ????

Intractable problems = Inefficiently solvable ????

# Problems defying classification

- Frustrating news…….is there is another class of problems for which


No reasonably fast algorithms have been found


Intractability of these problems cannot be proved

# Why do we need to classify problems?

- Computation has become pervasive in all walks of life
  - a standard tool in about every academic field viz.
    - whole subfields of Chemistry, Biology, Physics, Economics, OR and others
    - devoted to large-scale computational modeling, simulations and problem solving.
- We need to understand therefore, the limitations of computational power….

# Why do we need to classify problems?

- Study of P, NP theory
  - helps understand, handle various topics in allied sciences
  - helps what can be feasibly solved and what cannot be
  - also, enables one to EXPLOIT the advantage due to HARDNESS of various computational problems
    - e.g……

# Summarizing……..Problem Complexity

- Is there a polynomial-time algorithm that solves the problem?
- Possible answers
  - yes
  - no
    - because it can be proved that all algorithms take exponential time
    - because it can be proved that no algorithm exists at all to solve this problem
  - don't know
  - don't know, but if such algorithm were to be found,
    - then it would provide a means of solving many other problems in polynomial time

# Contents

- What is an Algorithm ? Solving problems algorithmically
- Classifying the problems
- A Motivating Example
- Some Hard Problems
- Reductions
- Non-determinism
- Class P, NP
- NP Complete Problems
- Concluding Remarks

# Other interesting problem(s)

- A large group of students to be grouped to work on projects so as to ensure compatibility
  - Matching students in pairs …….solvable
- Hard problems
  - Making a group of three so that each pair in each trio is compatible to each other…… ???
  - Finding as large group of students as possible so that each pair therein is compatible to each other…….???
  - Wanting the student to sit across a table so that no incompatible students sit next to each other ….???
  - Putting students into three groups so that each student is in the same group with his/her compatible partner….???

# Other interesting problem(s)

- A large group of students to be grouped to work on projects so as to ensure compatibility
  - Matching students in pairs …….solvable
- Hard problems
  - Making a group of three so that each pair in each trio is compatible to each other……Partitioning into triangles
  - Finding as large group of students as possible so that each pair therein is compatible to each other…….
  - Wanting the student to sit across a table so that no incompatible students sit next to each other ….
  - Putting students into three groups so that each student is in the same group with his/her compatible partner….

# Other interesting problem(s)

- A large group of students to be grouped to work on projects so as to ensure compatibility
  - Matching students in pairs …….solvable
- Hard problems
  - Making a group of three so that each pair in each trio is compatible to each other……Partitioning into triangles
  - Finding as large group of students as possible so that each pair therein is compatible to each other…….Maximum Clique problem
  - Wanting the student to sit across a table so that no incompatible students sit next to each other ….
  - Putting students into three groups so that each student is in the same group with his/her compatible partner….

# Other interesting problem(s)

- A large group of students to be grouped to work on projects so as to ensure compatibility
  - Matching students in pairs …….solvable
- Hard problems
  - Making a group of three so that each pair in each trio is compatible to each other……Partitioning into triangles
  - Finding as large group of students as possible so that each pair therein is compatible to each other…….Maximum Clique problem
  - Wanting the student to sit across a table so that no incompatible students sit next to each other ….Hamiltonian cycle problem
  - Putting students into three groups so that each student is in the same group with his/her compatible partner….

# Other interesting problem(s)

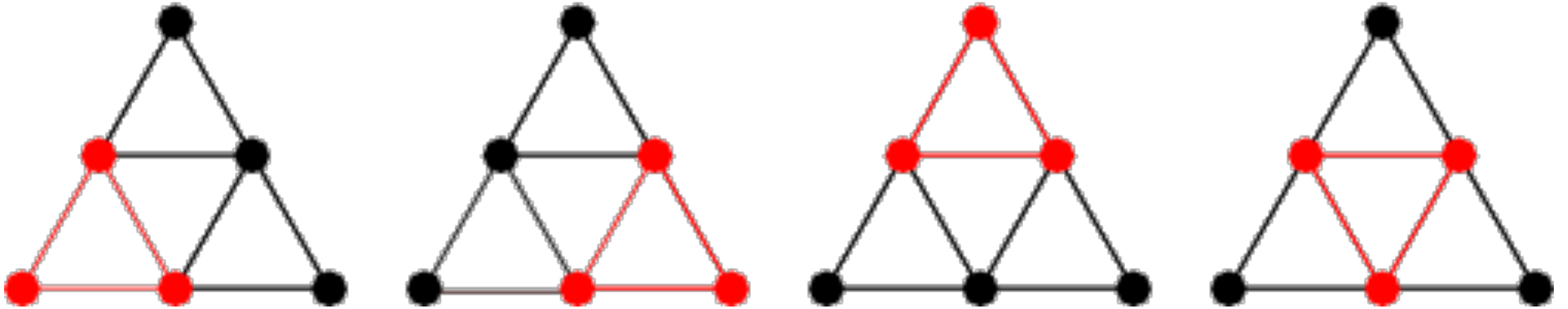
- A large group of students to be grouped to work on projects so as to ensure compatibility
  - Matching students in pairs …….solvable
- Hard problems
  - Making a group of three so that each pair in each trio is compatible to each other……Partitioning into triangles
  - Finding as large group of students as possible so that each pair therein is compatible to each other…….Maximum Clique problem
  - Wanting the student to sit across a table so that no incompatible students sit next to each other ….Hamiltonian cycle problem
  - Putting students into three groups so that each student is in the same group with his/her compatible partner….3-coloring problem

# A few other hard problems

- Determining Hamiltonian cycle in an unweighted graph
- Determining the minimal tour of cities – Travelling Salesperson problem
- Scheduling with profits and deadlines, to maximize the profit.
- Cliques in Social Networks
  - human groups form "cliques" on the basis of age, gender, race, ethnicity, religion/ideology, and many other things
- What is a clique in a graph G(V,E) ?

# Problem k-Clique

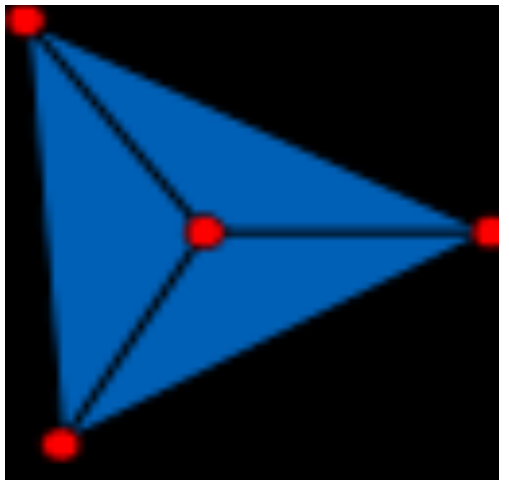
- A graph G=(V, E) is a Clique
  - if every two nodes in V are connected by an edge
- K-CLIQUE
  - Given a graph G=(V,E), if k-vertices in a graph are connected to each other then, it is a k–clique ….e.g.



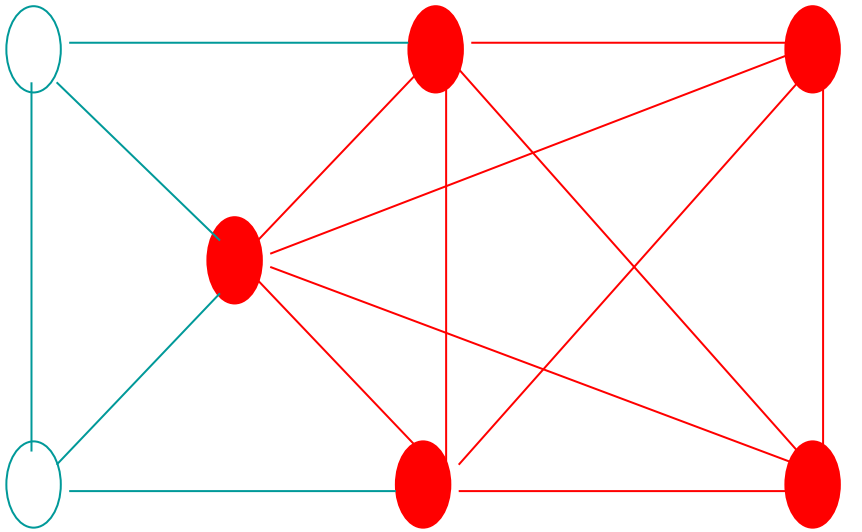
3-Clique

# Problem k-Clique…

- A graph G=(V, E) is a Clique
  - if every two nodes in V are connected by an edge
- K-CLIQUE
  - Given a graph G=(V,E), if k-vertices in a graph are connected to each other then, it is a k–clique ….e.g.

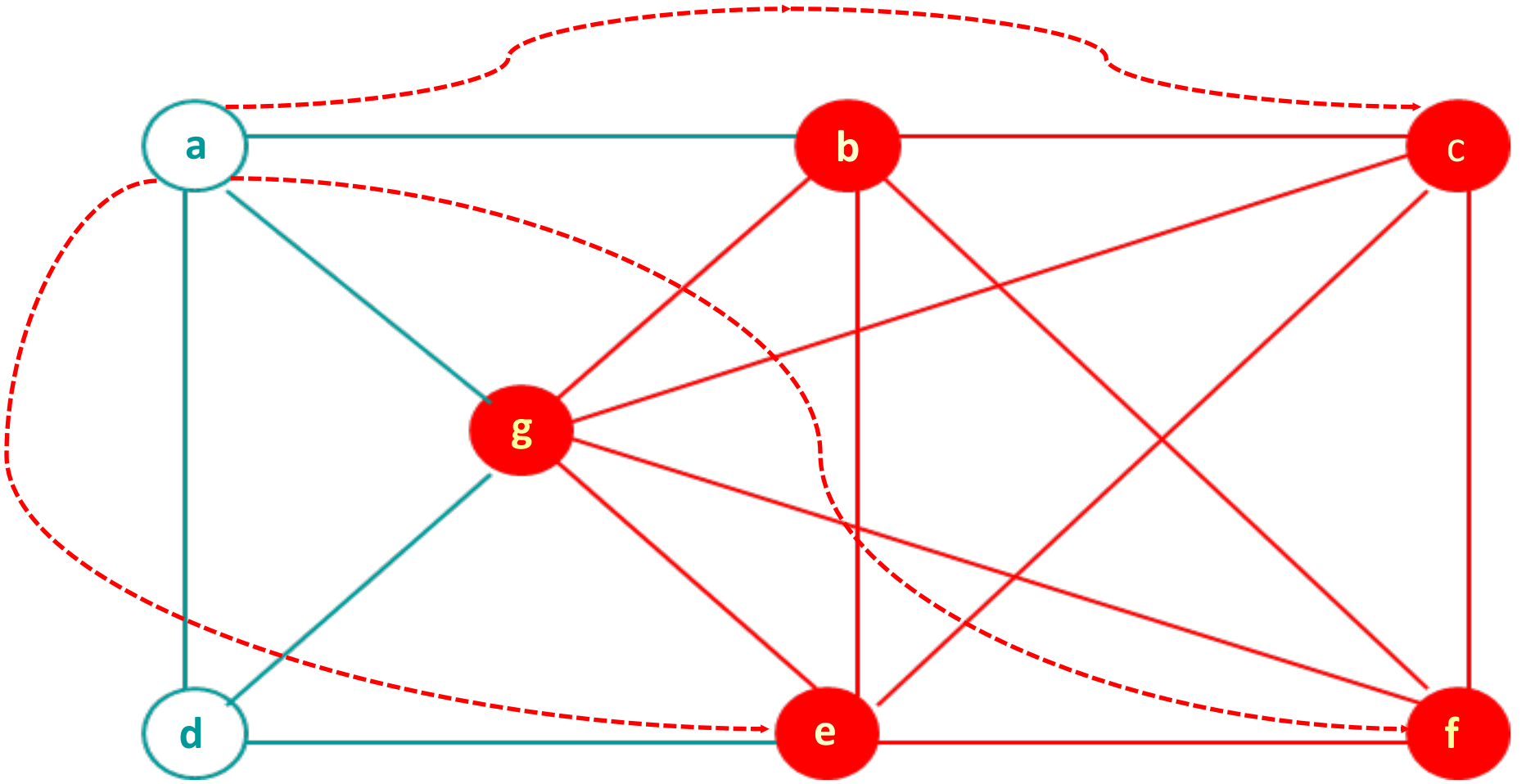


4-Clique



5-Clique

# Problem k-Clique…

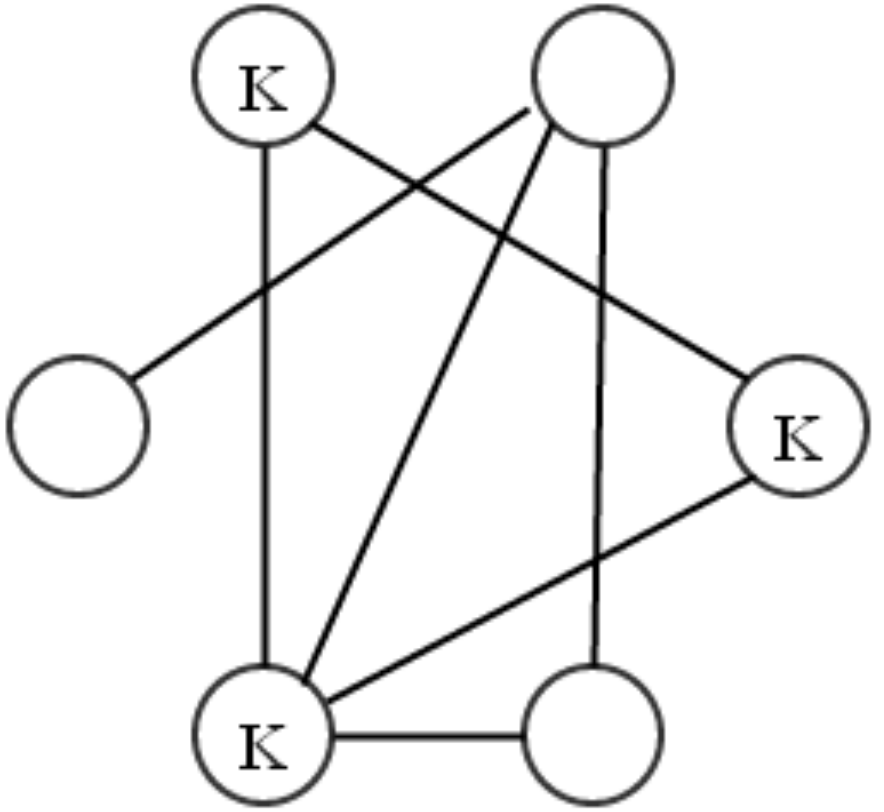



- What if we add edges from a to nodes c, e, and f ?

# Problem k-Clique

- Clique number ω(*G*) of a graph
  - The clique number ω(*G*) of a graph *G* is the number of vertices in a maximum clique in *G*.
- In passing…..
  - the clique and independent set problems ? Are they related ?

# Problems k-Clique and IS




Clique

# Boolean Satisfiability

- Terminologies
  - Propositional (boolean) variable
    - a variable that may be assigned value true or false
  - Literal
    - A Boolean variable or its negation.
  - Propositional formula
    - an expression that is either a propositional variable or a propositional constant or an expression of boolean operator and its operands
  - Clause
    - a disjunction – sequence of literals separated by V
  - Conjunctive normal form
    - A regular form of propositional formula $\Phi$ that is the conjunction of clauses

# Boolean Satisfiability…

- An example propositional CNF formula is

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_2 \vee x_3\right) \wedge \left(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}\right)$$

  where $x_1$, $x_2$, $x_3$ are propositional variables
- Truth assignment
  - a boolean valued function on the set i.e.
    - an assignment of values true or false to each propositional variable in the set
- Satisfiability
  - When does a truth assignment is said to satisfy a formula ?
- SAT
  - Given CNF formula $\Phi$, does it have a satisfying truth assignment?

# Problem Boolean Satisfiability

- Input
  - A boolean formula F in CNF
- Goal
  - Check if F is satisfiable or not.
    - e.g. if **`F=(x1+x2)`** can we assign at least one set of values to the literals of the formula so that F=1.
- How to solve this problem deterministically ?
  - What could be the Brute-force approach ?
  - Time complexity ??
    - $O(2^n|F|)$
  - Cannot do any better than that.

# Applications

- Bioinformatics
  - clustering gene expression data
  - modelling ecological niches in food webs.
  - Modelling  protein structure prediction
- Electrical engineering
  - analyzing communications networks,
  - designing efficient circuits for computing partially specified Boolean functions.
  - automatic test pattern generation
  - finding a hierarchical partition of an electronic circuit into smaller
- … … … … … …

# A few other hard problems

- Determining Hamiltonian cycle in an unweighted graph
- Determining the minimal tour of cities – Travelling Salesperson problem
- Scheduling with profits and deadlines, to maximize the profit.
- Cliques in Social Networks
  - human groups form "cliques" on the basis of age, gender, race, ethnicity, religion/ideology, and many other things
- Various problems in
  - Computer Vision/Pattern Recognition, Information/Coding Theory, Map Labeling, Molecular Biology, Scheduling

# A few other hard problems …

- Aerospace engineering
  - optimal mesh partitioning for finite elements.
- Biology
  - protein folding.
- Chemical engineering
  - heat exchanger network synthesis.
- Civil engineering
  - equilibrium of urban traffic flow.
- Economics
  - computation of arbitrage in financial markets with friction.
- Electrical engineering
  - VLSI layout

*Ref Klienberg Tardos*

# A few other hard problems …

- Environmental engineering
  - optimal placement of contaminant sensors.
- Financial engineering
  - find minimum risk portfolio of given return.
- Game theory
  - find Nash equilibrium that maximizes social welfare.
- Genomics
  - phylogeny reconstruction.
- Mechanical engineering
  - structure of turbulence in sheared flows.
- Medicine
  - reconstructing 3-D shape from biplane angiocardiogram.

*Ref Klienberg Tardos*

# A few other hard problems …

- Operations research
  - optimal resource allocation.
- Physics
  - partition function of 3-D Ising model in statistical mechanics.
- Politics
  - Shapley-Shubik voting power.
- Pop culture
  - Minesweeper consistency.
- Statistics
  - optimal experimental design.

*Ref Klienberg Tardos*

# Reviewing Objectives

- To understand
  - that many of the problems that have polynomial time algorithms are computationally related.
  - a few of the real world problems defy classification…..
  - how nondeterminism property can be exploited to understand the classes of problems.
  - the classification of the problems viz. NP-Hard and NP-Complete
  - how one problem can be reduced to another…… rendering them to be similar in nature to one another……..