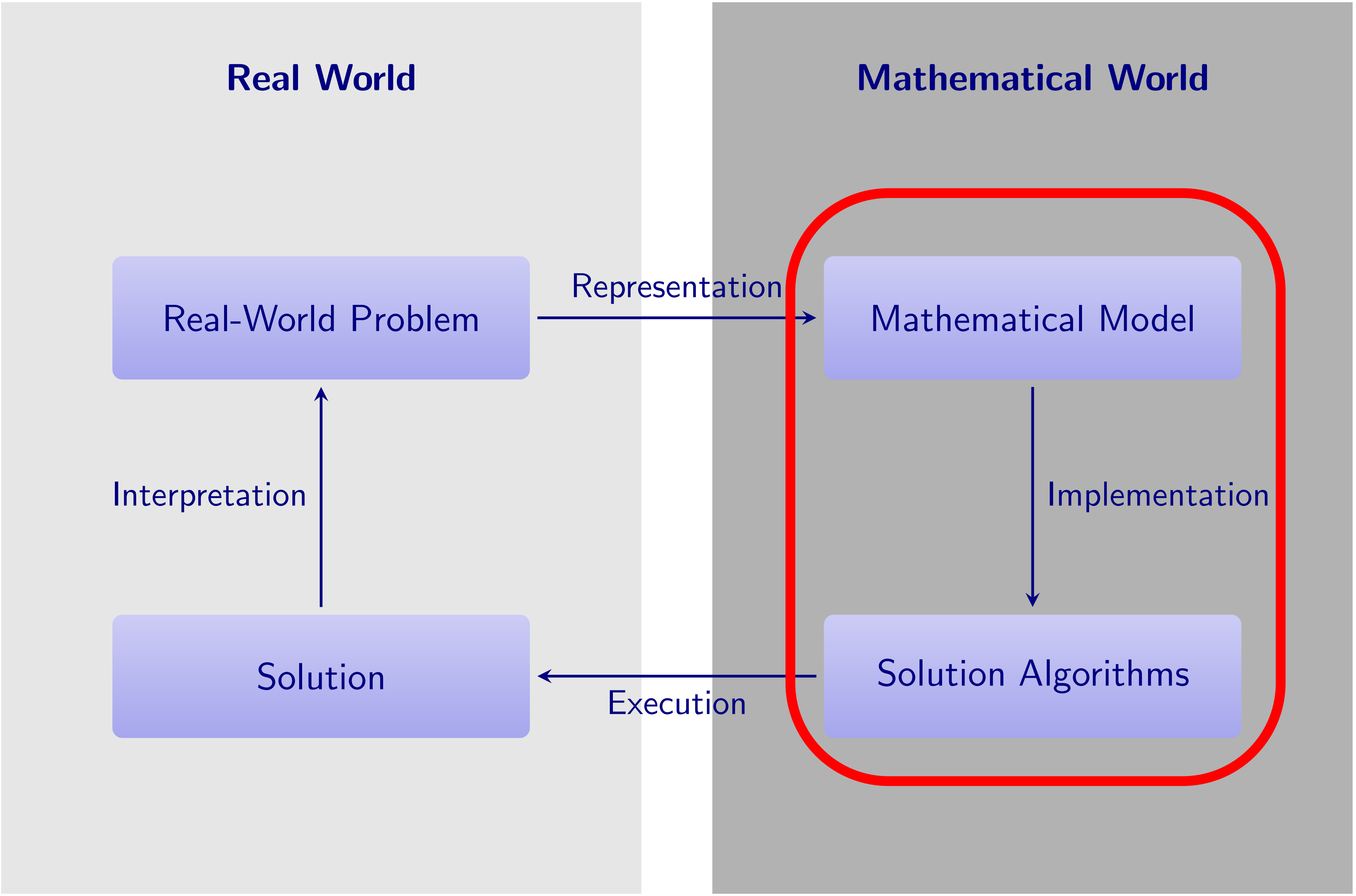
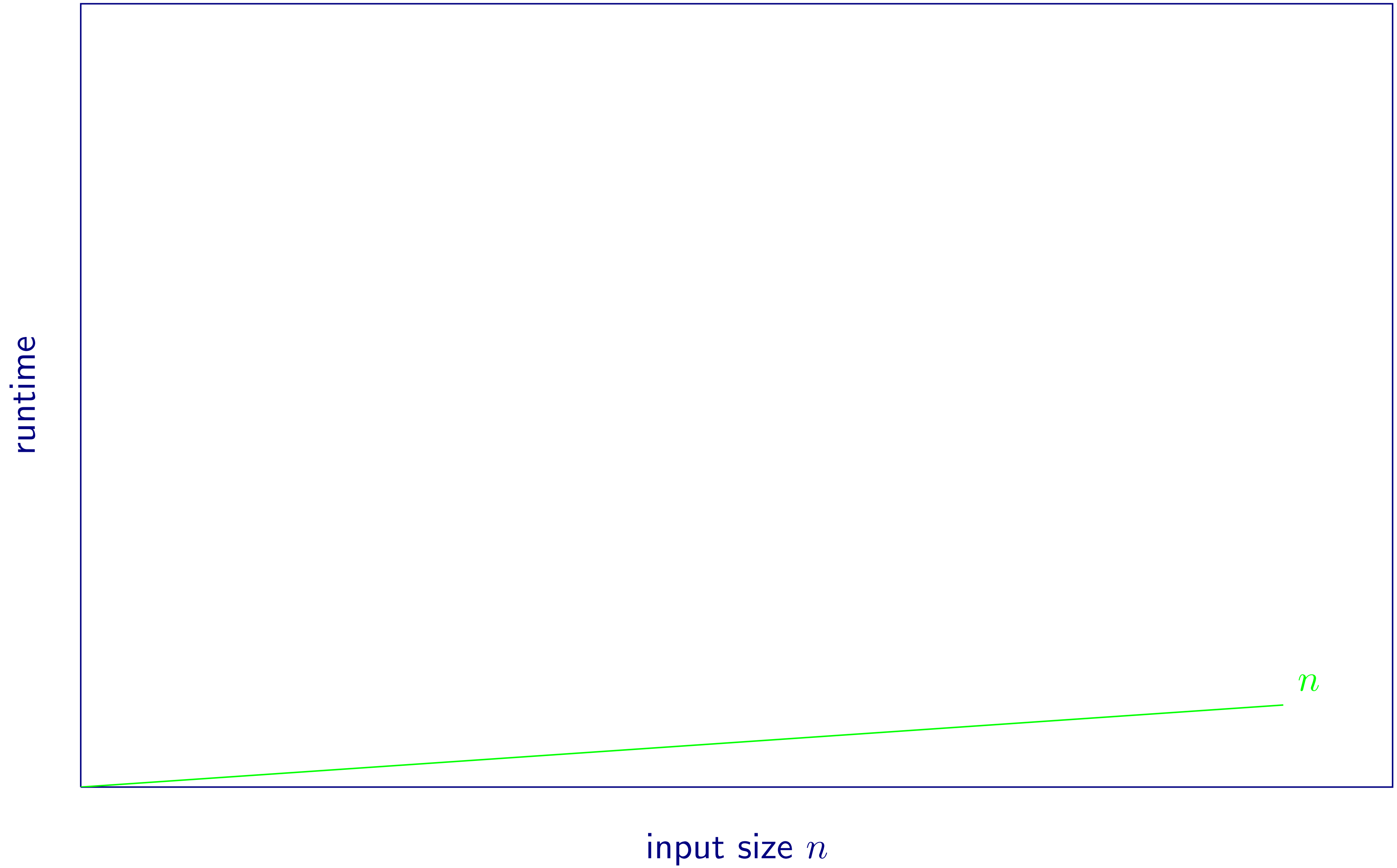
Mathematical Tools for Engineering and Management

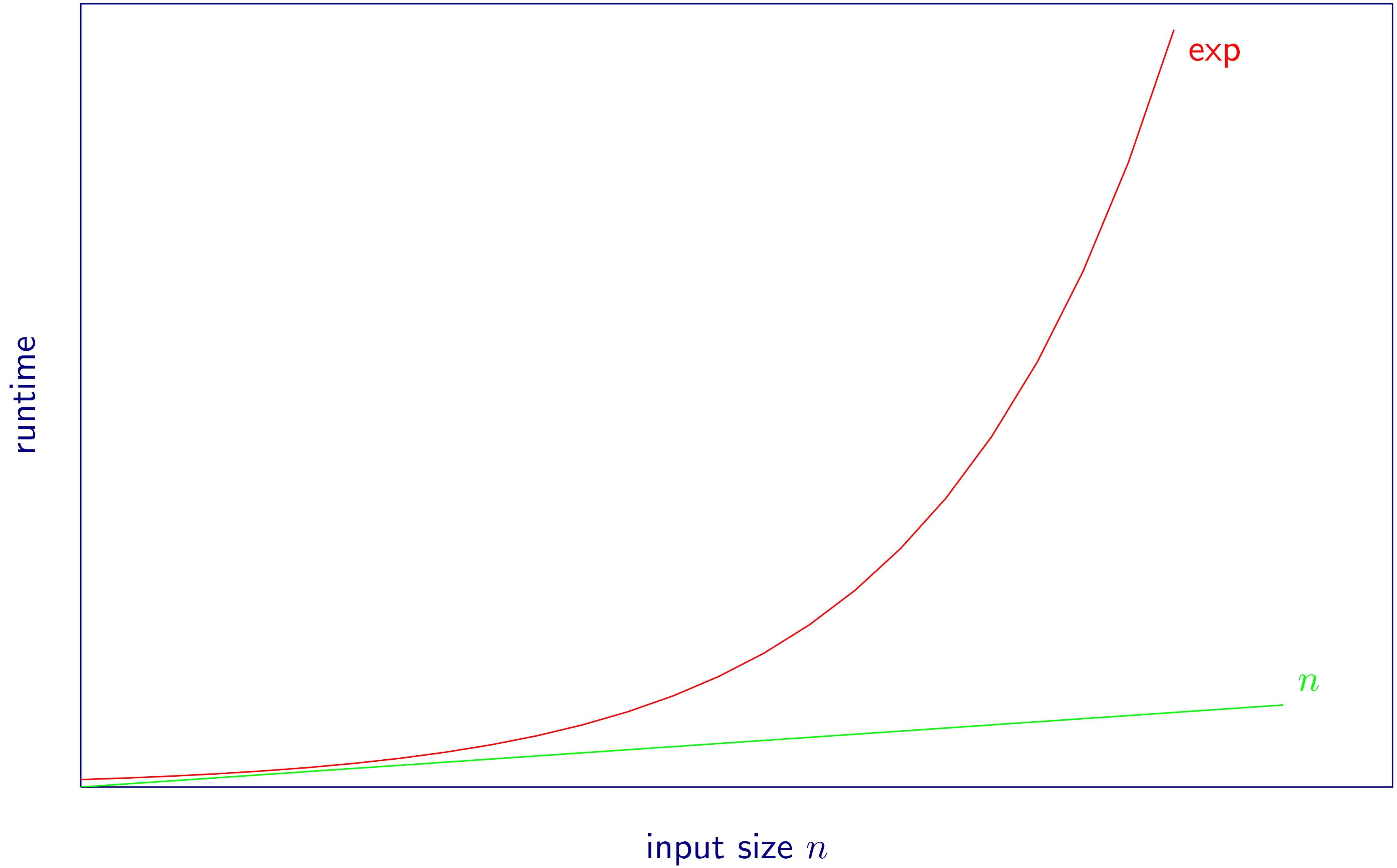
Lecture 11

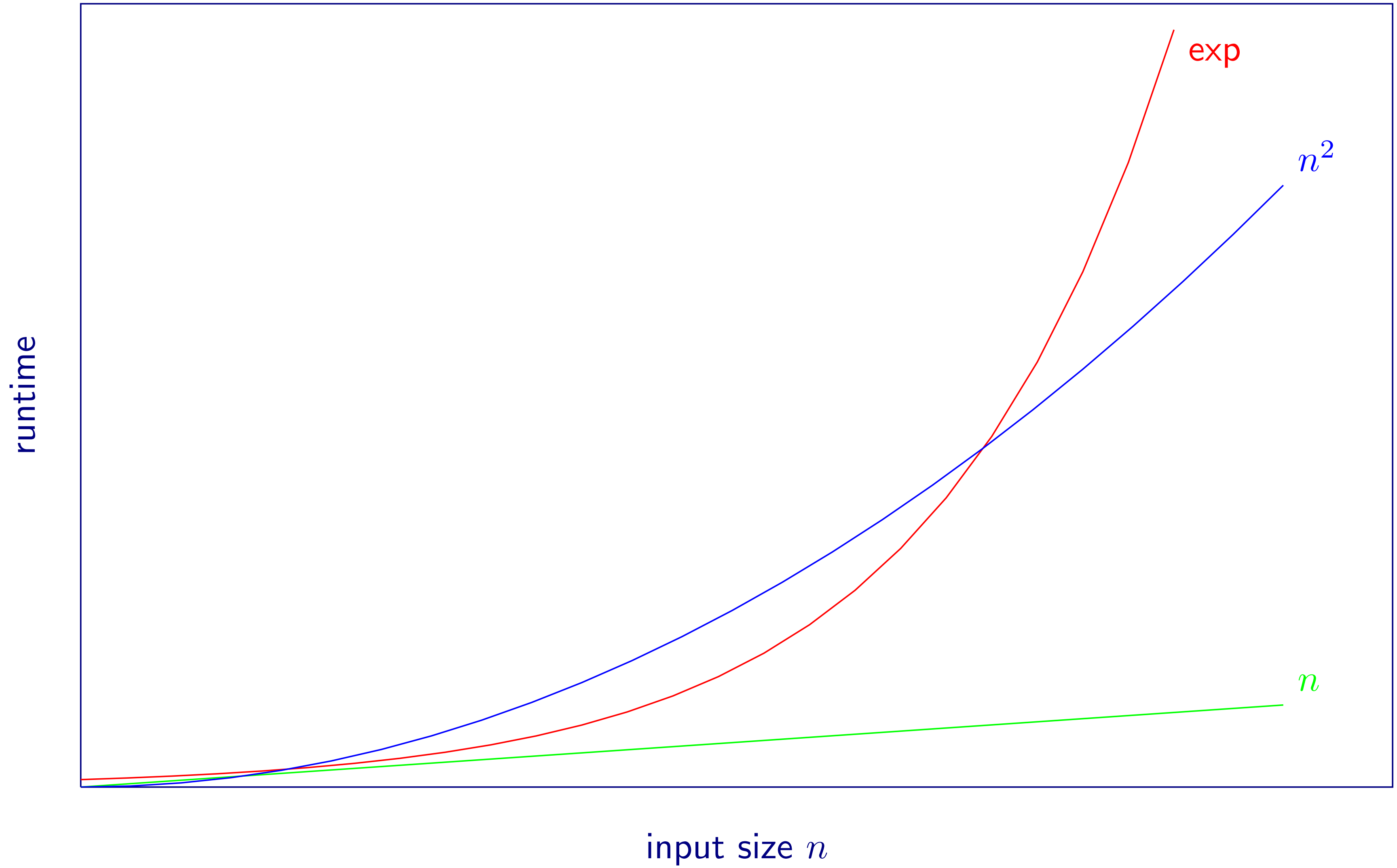
11 Jan 2012

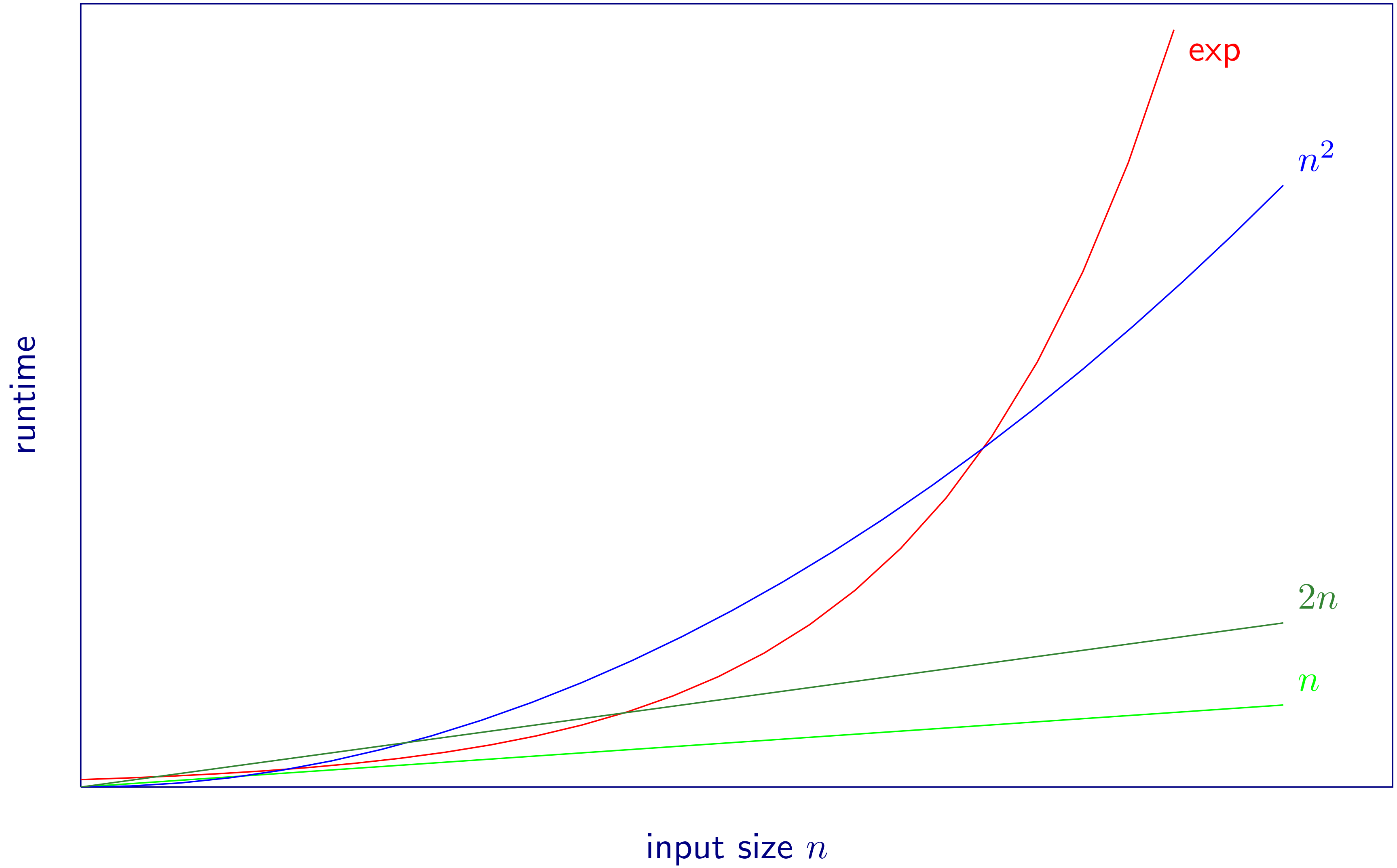
- Models, Data and Algorithms
- Linear Optimization
- Mathematical Background: Polyhedra, Simplex-Algorithm
- Sensitivity Analysis; (Mixed) Integer Programming
- MIP Modelling
- MIP Modelling: More Examples; Branch & Bound
- Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- TSP-Heuristics
- Network Flows
- Shortest Path Problem
- Complexity Theory
- Nonlinear Optimization, Scheduling
- Lot Sizing, Multicriteria Optimization
- Oral exam

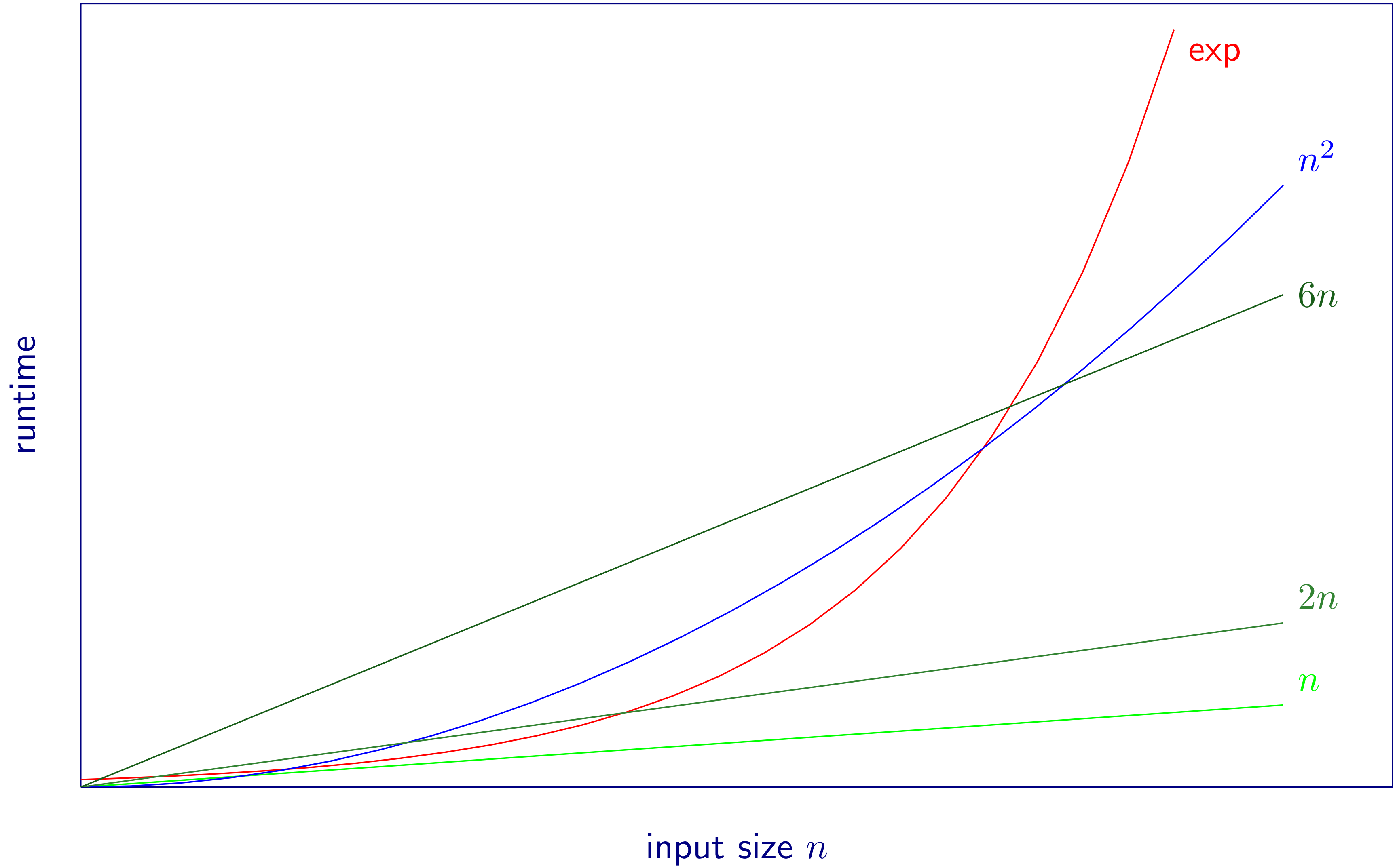

Real World
Mathematical World
Real-World Problem
Representation
Mathematical Model
Interpretation
Implementation
Solution
Execution
Solution Algorithms

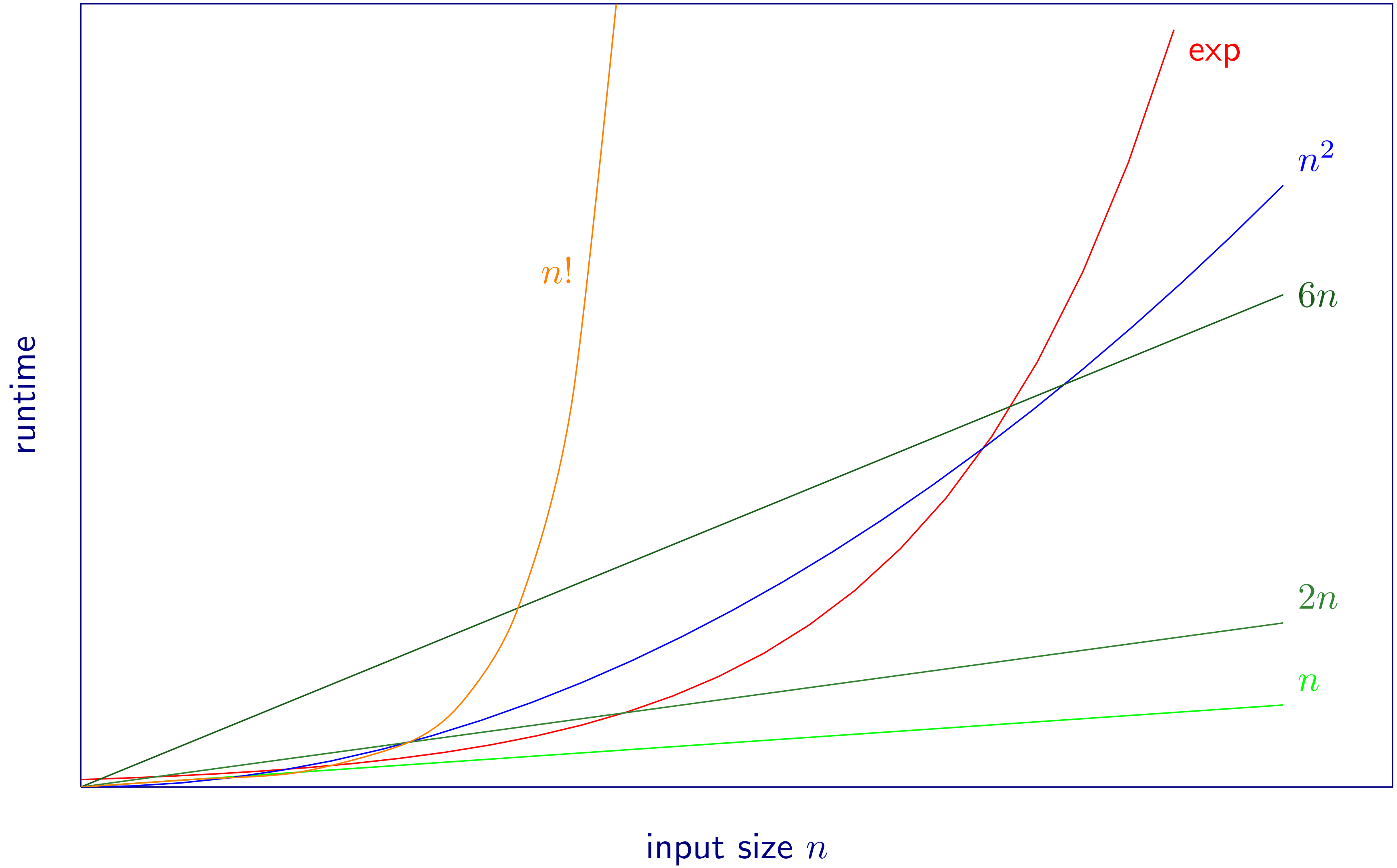

runtime
n
input size n


exp
n
runtime
input size n

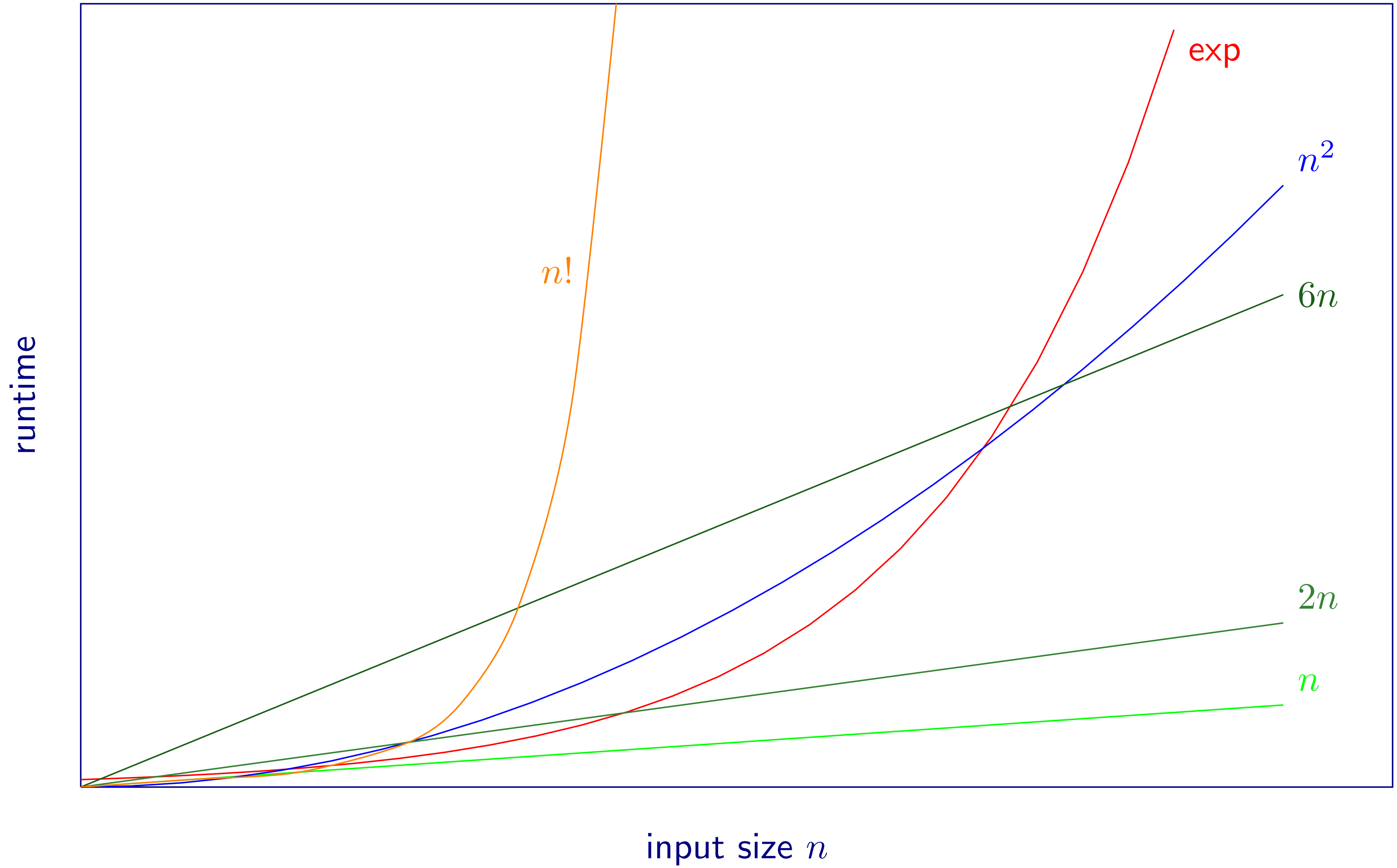

exp
n^2
n
runtime
input size n


exp
n^2
$2n$
n
runtime
input size n


exp
n^2
$6n$
$2n$
n
runtime
input size n



exp
n^2
$n!$
$6n$
$2n$
n
runtime
input size n



linear — polynomial — exponential

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

▷ Examples:

- Efficient: Dijkstra's algorithm, Kruskal's algorithm, TSP heuristic using MST

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

▷ Examples:

- Efficient: Dijkstra's algorithm, Kruskal's algorithm, TSP heuristic using MST
- Not efficient: Branch & bound method, Simplex algorithm (?)

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called exact if it guarantees to return an optimal solution (i.e. it can be proved that it always (i.e. for every input!) returns a solution with best possible objective).

▷ Examples:

- Efficient: Dijkstra's algorithm, Kruskal's algorithm, TSP heuristic using MST
- Not efficient: Branch & bound method, Simplex algorithm (?)

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called exact if it guarantees to return an optimal solution (i.e. it can be proved that it always (i.e. for every input!) returns a solution with best possible objective).

▷ Examples:

- Efficient: Dijkstra's algorithm, Kruskal's algorithm, TSP heuristic using MST
- Not efficient: Branch & bound method, Simplex algorithm (?)
- Exact: Dijkstra, Kruskal, Simplex, Branch & bound, Complete enumeration

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called exact if it guarantees to return an optimal solution (i.e. it can be proved that it always (i.e. for every input!) returns a solution with best possible objective).

▷ Examples:

- Efficient: Dijkstra's algorithm, Kruskal's algorithm, TSP heuristic using MST
- Not efficient: Branch & bound method, Simplex algorithm (?)
- Exact: Dijkstra, Kruskal, Simplex, Branch & bound, Complete enumeration
- Not exact: heuristics, approximation algorithms

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called exact if it guarantees to return an optimal solution (i.e. it can be proved that it always (i.e. for every input!) returns a solution with best possible objective).

- Examples:

	efficient	not efficient
exact	Dijkstra's algorithm	Simplex algorithm (?)
	Kruskal's algorithm	Branch & bound
	Ellipsoid method	Complete enumeration
not exact	TSP heuristic using MST	
	approximation algorithms	

Efficient and exact algorithms

An algorithm is called efficient if it has polynomial runtime (i.e. its runtime can be bounded by a polynomial in the input size).

An algorithm is called exact if it guarantees to return an optimal solution (i.e. it can be proved that it always (i.e. for every input!) returns a solution with best possible objective).

- Examples:

	efficient	not efficient
exact	Dijkstra's algorithm Kruskal's algorithm Ellipsoid method	Simplex algorithm (?) Branch & bound Complete enumeration
not exact	TSP heuristic using MST approximation algorithms	

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

- Examples:

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

▷ Examples:

- Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

▷ Examples:

- Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Minimum Spanning Tree ➡ Kruskal's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

▷ Examples:

- Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Minimum Spanning Tree ➡ Kruskal's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Linear Programming ➡ Ellipsoid method ➡ in $\mathcal{P}$

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

▷ Examples:

- Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Minimum Spanning Tree ➡ Kruskal's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Linear Programming ➡ Ellipsoid method ➡ in $\mathcal{P}$
- Integer Programming ➡ no polynomial algorithm known! ➡ not known if in $\mathcal{P}$

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

▷ Examples:

- Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Minimum Spanning Tree ➡ Kruskal's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
- Linear Programming ➡ Ellipsoid method ➡ in $\mathcal{P}$
- Integer Programming ➡ no polynomial algorithm known! ➡ not known if in $\mathcal{P}$

➡ How hard can a problem be...?

A mathematical problem for which an exact and efficient algorithm is known, is a member of the complexity class $\mathcal{P}$.

- ▷ Examples:
  - Shortest Path Problem ➡ Dijkstra's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
  - Minimum Spanning Tree ➡ Kruskal's algorithm ($\mathcal{O}(n^2)$) ➡ in $\mathcal{P}$
  - Linear Programming ➡ Ellipsoid method ➡ in $\mathcal{P}$
  - Integer Programming ➡ no polynomial algorithm known! ➡ not known if in $\mathcal{P}$

➡ How hard can a problem be...?

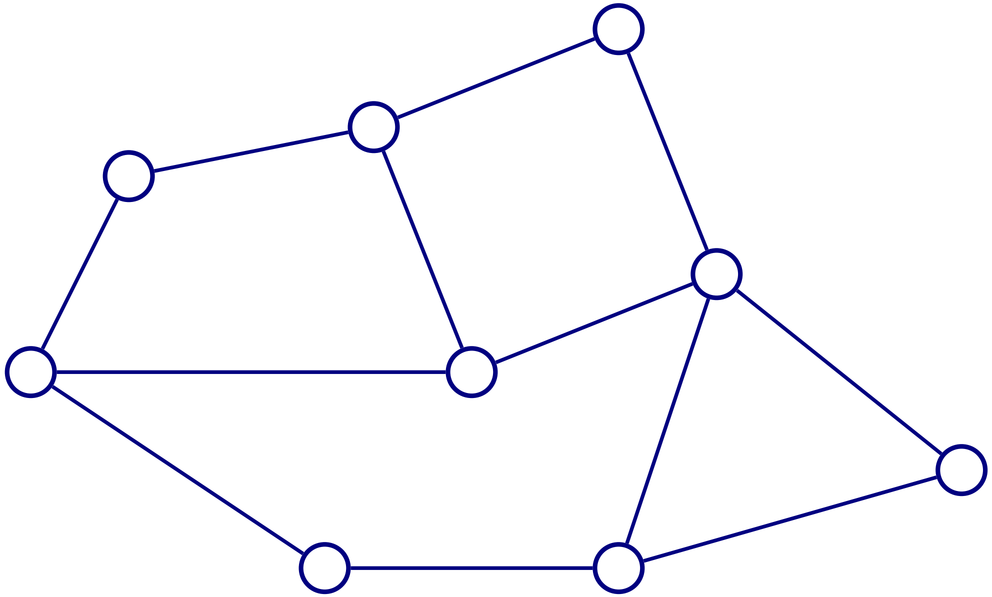
A mathematical problem for which it is possible to verify feasibility of a given solution in polynomial time, is a member of the complexity class $\mathcal{NP}$.

- Given an undirected graph, find a stable set with maximal cardinality!

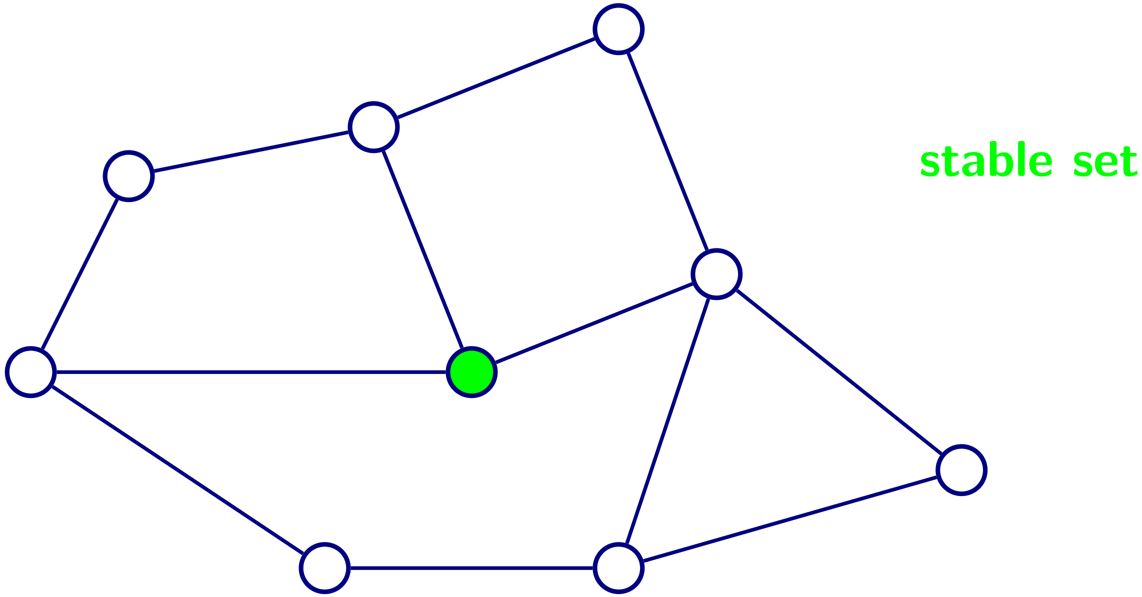
- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge

▷ Given an undirected graph, find a stable set with maximal cardinality!

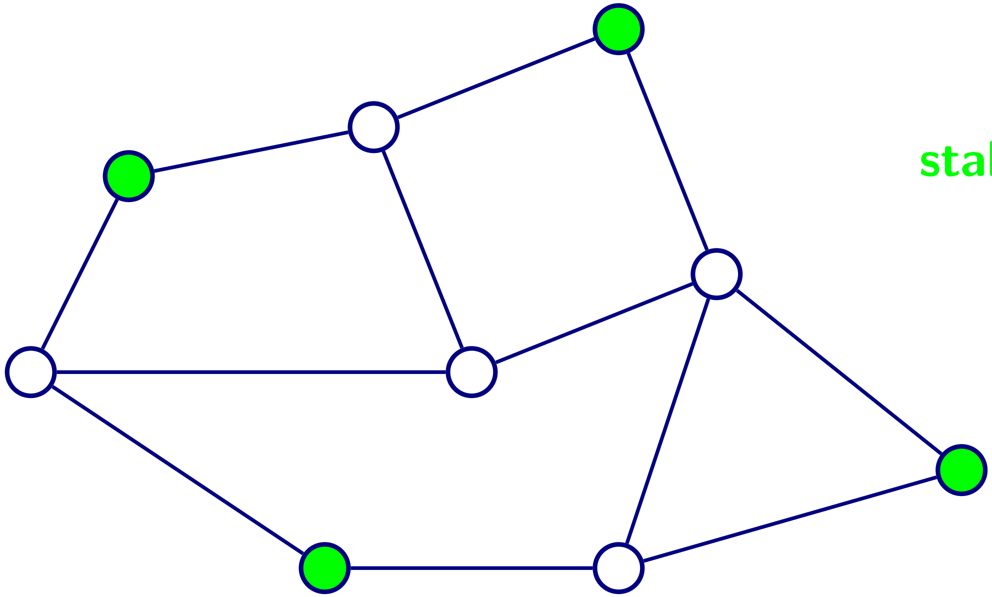
▷ Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge

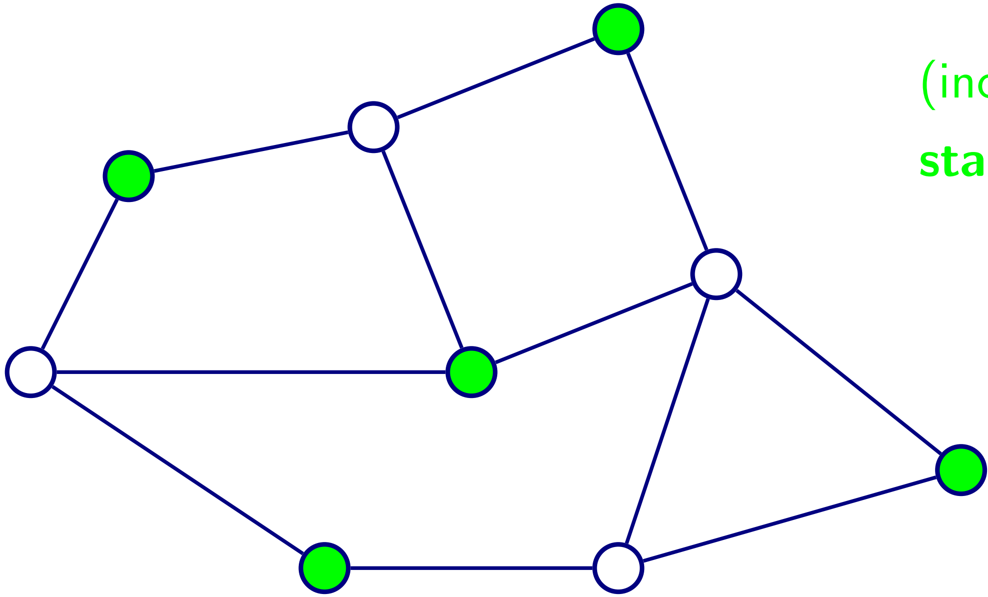


- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



stable set

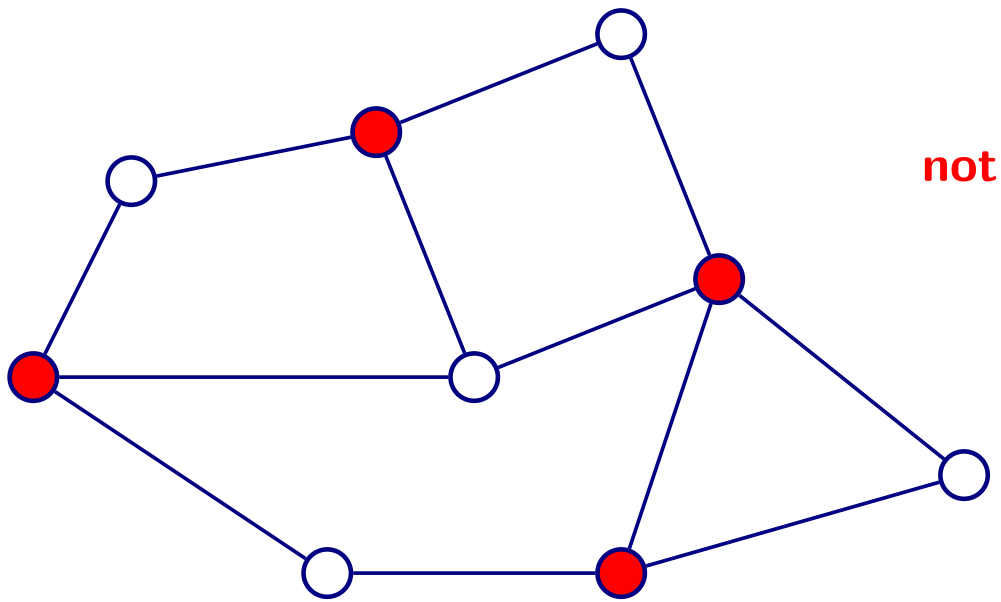
- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



(inclusion-maximal)
stable set

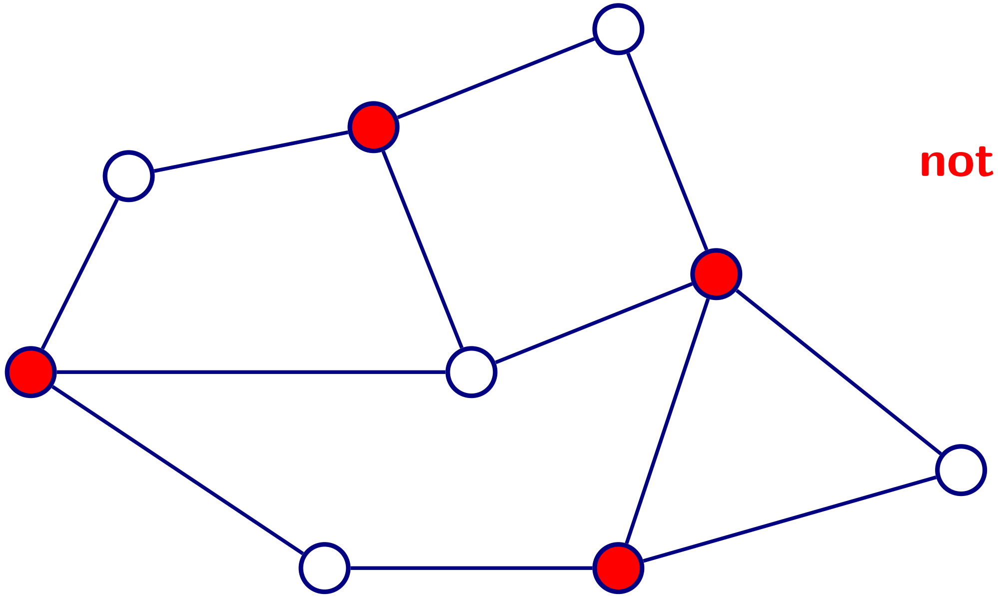
▷ Given an undirected graph, find a stable set with maximal cardinality!

▷ Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



not a stable set!

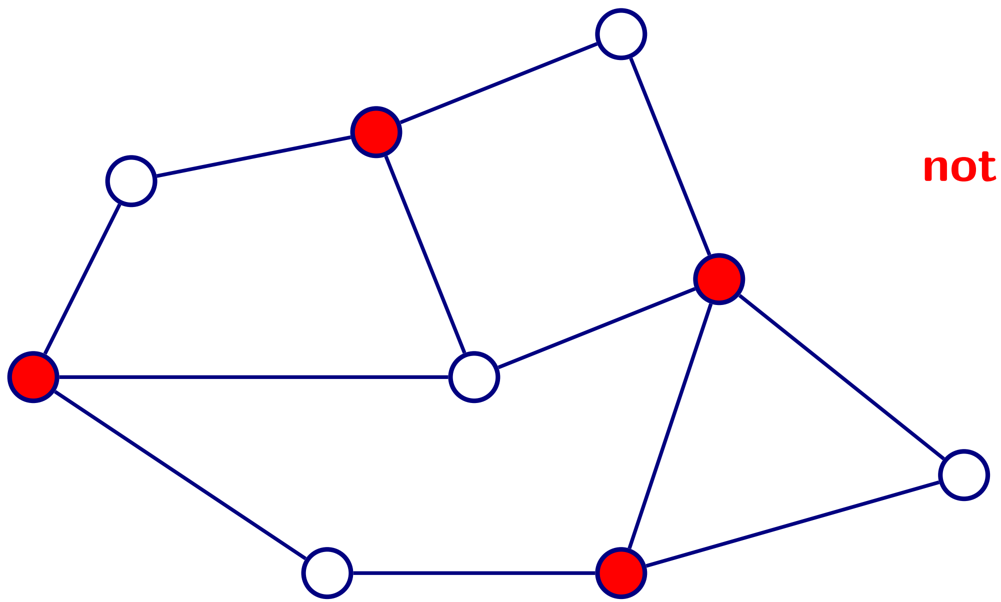
- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



not a stable set!

- No efficient exact algorithm known ➡ not known to be in $\mathcal{P}$

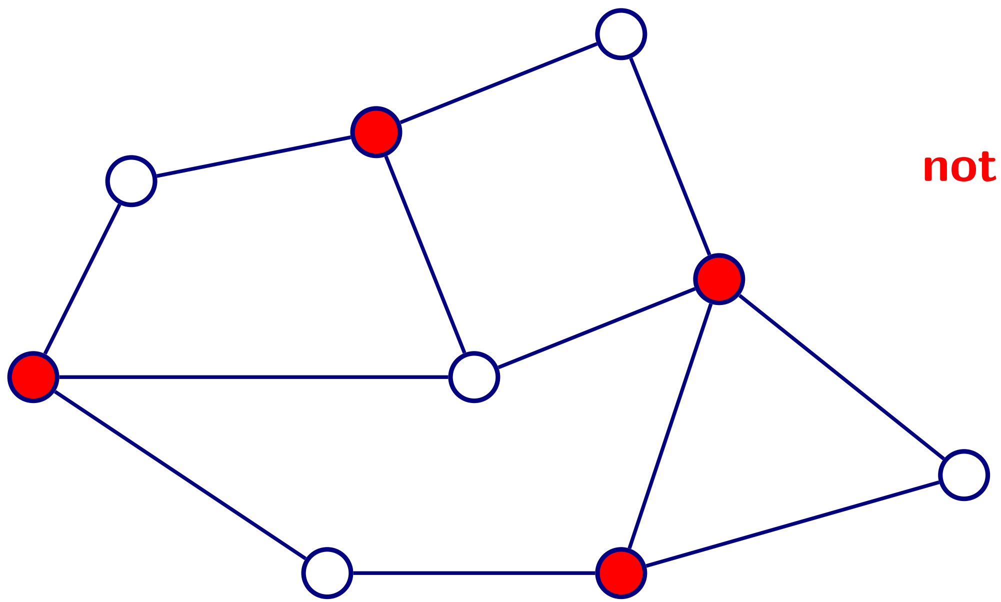
- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



not a stable set!

- No efficient exact algorithm known ➡ not known to be in $\mathcal{P}$
- But: given a set S of vertices, testing if S is a stable set is easy

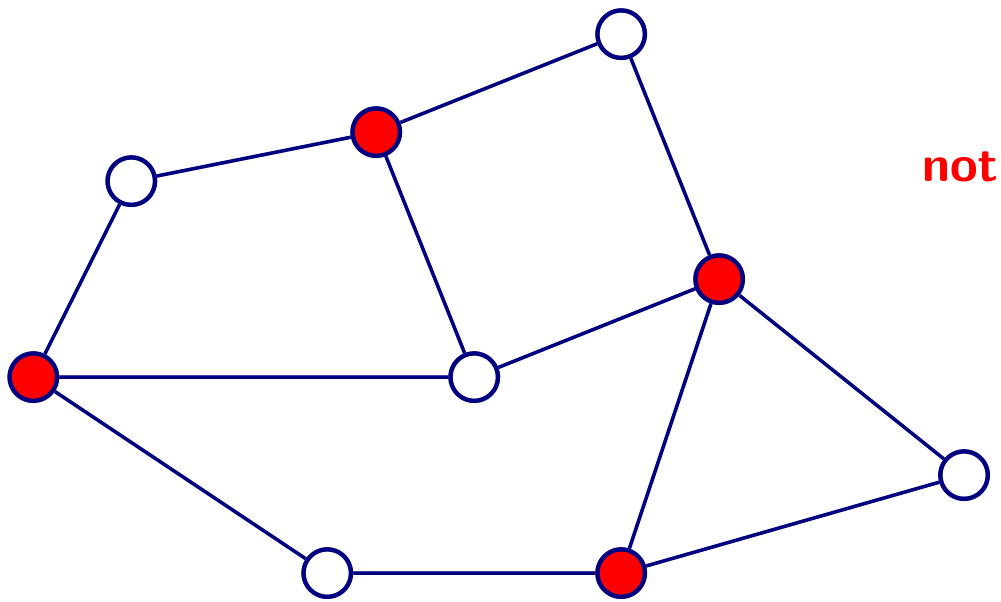
- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



not a stable set!

- No efficient exact algorithm known ➡ not known to be in $\mathcal{P}$
- But: given a set S of vertices, testing if S is a stable set is easy ($\mathcal{O}(n^2)$)

- Given an undirected graph, find a stable set with maximal cardinality!
- Stable set: a subset of the vertices, such that no two vertices of the subset are directly connected via an edge



not a stable set!

- No efficient exact algorithm known ➡ not known to be in $\mathcal{P}$
- But: given a set S of vertices, testing if S is a stable set is easy ($\mathcal{O}(n^2)$)

  ➡ Stable Set Problem is in $\mathcal{NP}$

▷ Examples:

▷ Examples:

- Stable Set Problem

▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem

▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem
- (Mixed) Integer Programming

▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem
- (Mixed) Integer Programming
- For every problem in $\mathcal{P}$ we can obviously check feasibility in polynomial time

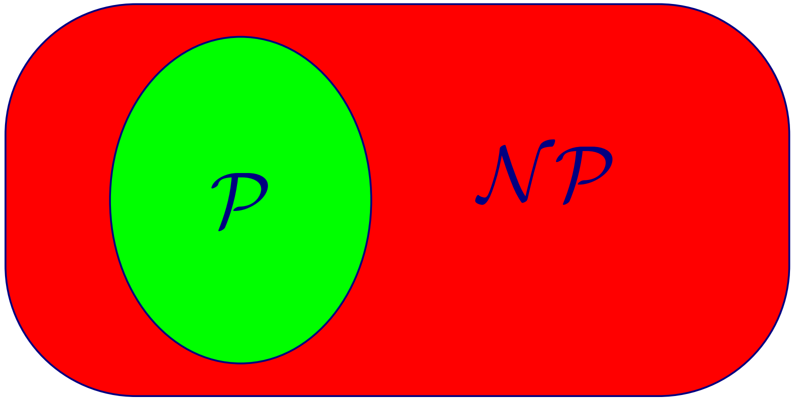
- Examples:
  - Stable Set Problem
  - Travelling Salesman Problem
  - (Mixed) Integer Programming
  - For every problem in $\mathcal{P}$ we can obviously check feasibility in polynomial time

    ➡ $\mathcal{P} \subseteq \mathcal{NP}$

▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem
- (Mixed) Integer Programming
- For every problem in $\mathcal{P}$ we can obviously check feasibility in polynomial time

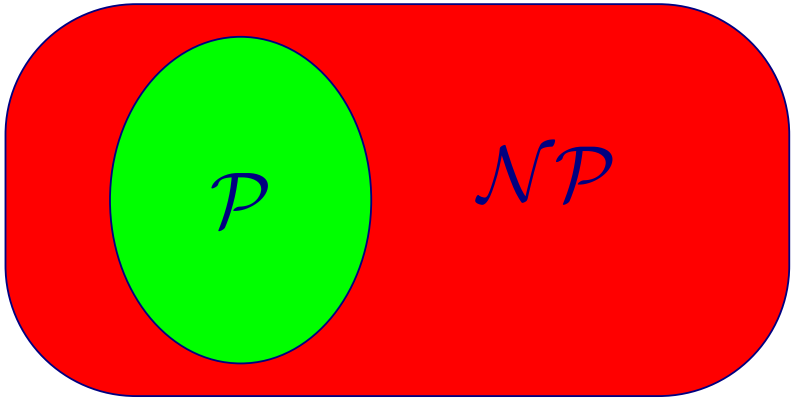
  ➡ $\mathcal{P} \subseteq \mathcal{NP}$



▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem
- (Mixed) Integer Programming
- For every problem in $\mathcal{P}$ we can obviously check feasibility in polynomial time

  ➡ $\mathcal{P} \subseteq \mathcal{NP}$

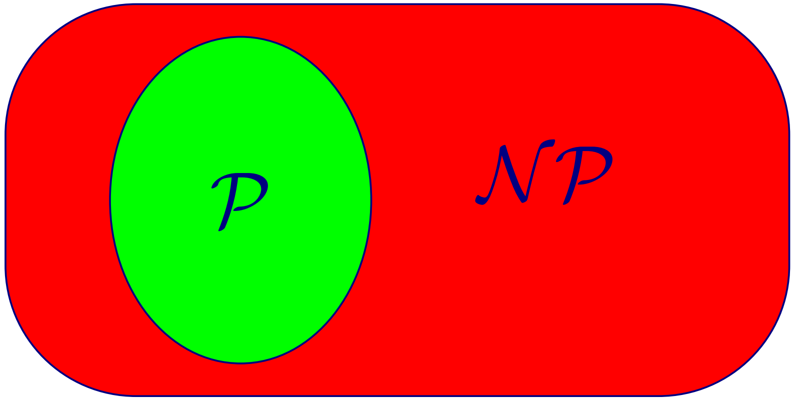


➡ Question: is $\mathcal{P} = \mathcal{NP}$?

▷ Examples:

- Stable Set Problem
- Travelling Salesman Problem
- (Mixed) Integer Programming
- For every problem in $\mathcal{P}$ we can obviously check feasibility in polynomial time

  ➡ $\mathcal{P} \subseteq \mathcal{NP}$



➡ Question: is $\mathcal{P} = \mathcal{NP}$? ➡ $1,000,000 for the answer to this question!
see `http://www.claymath.org/millennium/P_vs_NP/`

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **A** is called $\mathcal{NP}$-complete if every problem in $\mathcal{NP}$ is reducible to **A**.

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **A** is called $\mathcal{NP}$-complete if every problem in $\mathcal{NP}$ is reducible to **A**.

➡ If some $\mathcal{NP}$-complete problem can be solved in polynomial time then $\mathcal{P} = \mathcal{NP}$!

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **A** is called $\mathcal{NP}$-complete if every problem in $\mathcal{NP}$ is reducible to **A**.

➡ If some $\mathcal{NP}$-complete problem can be solved in polynomial time then $\mathcal{P} = \mathcal{NP}$!

▷ Nearly every combinatorial optimization problem is known to be either in $\mathcal{P}$ or $\mathcal{NP}$-complete

Problems in $\mathcal{P}$	**$\mathcal{NP}$-complete problems**
Minimum Spanning Tree	Stable Set
Linear Programming	Integer Programming
Shortest Path	Travelling Salesman

A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

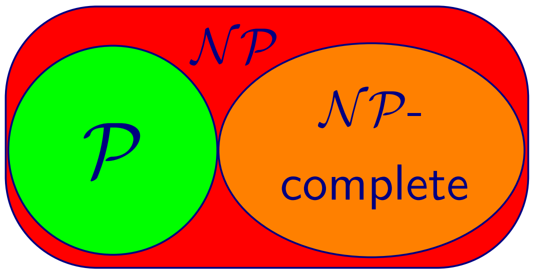
▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **A** is called $\mathcal{NP}$-complete if every problem in $\mathcal{NP}$ is reducible to **A**.

➡ If some $\mathcal{NP}$-complete problem can be solved in polynomial time then $\mathcal{P} = \mathcal{NP}$!

▷ Nearly every combinatorial optimization problem is known to be either in $\mathcal{P}$ or $\mathcal{NP}$-complete

Problems in $\mathcal{P}$	**$\mathcal{NP}$-complete problems**
Minimum Spanning Tree	Stable Set
Linear Programming	Integer Programming
Shortest Path	Travelling Salesman



A mathematical problem **X** is called reducible to another problem **A** if every instance of **X** can be transformed into an instance of **A** by a polynomial-time transformation.

➡ If **A** can be solved efficiently (i.e. by a polynomial-time algorithm), then also **X** can be solved efficiently!

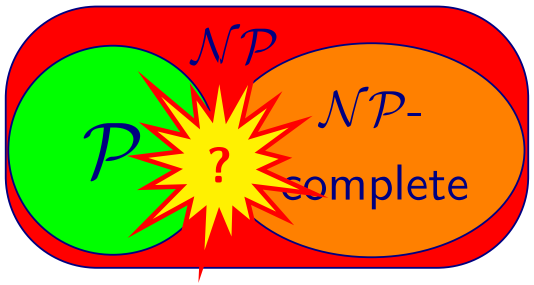
▷ Example: Shortest Path Problem is reducible to Integer Programming Problem

A mathematical problem **A** is called $\mathcal{NP}$-complete if every problem in $\mathcal{NP}$ is reducible to **A**.

➡ If some $\mathcal{NP}$-complete problem can be solved in polynomial time then $\mathcal{P} = \mathcal{NP}$!

▷ Nearly every combinatorial optimization problem is known to be either in $\mathcal{P}$ or $\mathcal{NP}$-complete

Problems in $\mathcal{P}$	**$\mathcal{NP}$-complete problems**
Minimum Spanning Tree	Stable Set
Linear Programming	Integer Programming
Shortest Path	Travelling Salesman



- Models, Data and Algorithms
- Linear Optimization
- Mathematical Background: Polyhedra, Simplex-Algorithm
- Sensitivity Analysis; (Mixed) Integer Programming
- MIP Modelling
- MIP Modelling: More Examples; Branch & Bound
- Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- TSP-Heuristics
- Network Flows
- Shortest Path Problem
- Complexity Theory
- Nonlinear Optimization
- Scheduling, Lot Sizing, Multicriteria Optimization
- Oral exam