Applied Artificial Intelligence

Session 4: Searching as an AI Method

Fall 2018
NC State University
Instructor: Dr. Behnam Kia
Course Website: https://appliedai.wordpress.ncsu.edu/

Computational Complexity
&
Complexity Classes

Computational Complexity

Computational complexity theory focuses on classifying computational problems according to their inherent difficulty, and relating the resulting complexity classes to each other.

– Wikipedia

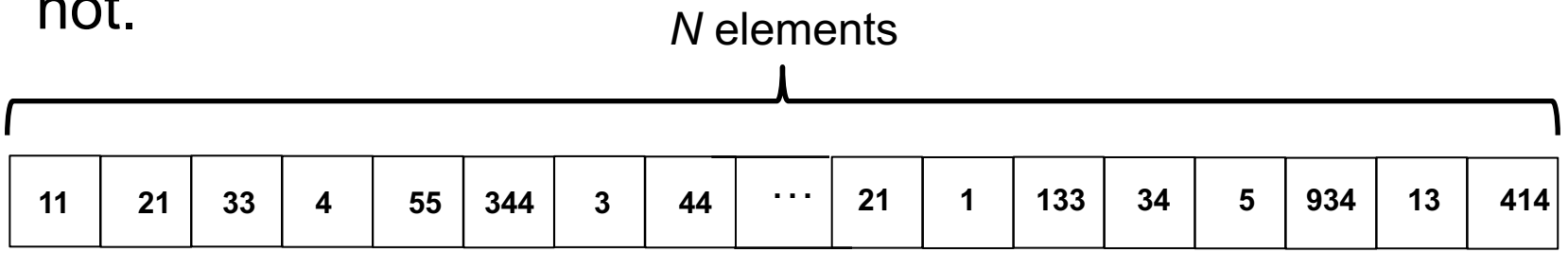
Computational Complexity:

Why We Need It?

- We would like to get an idea of hardness of a problem we like to solve.
- It tells us how many resources we need to solve the problem.
- Or is it even practical trying to directly solve it?

Example: Searching an Array of Size N for a specific Element

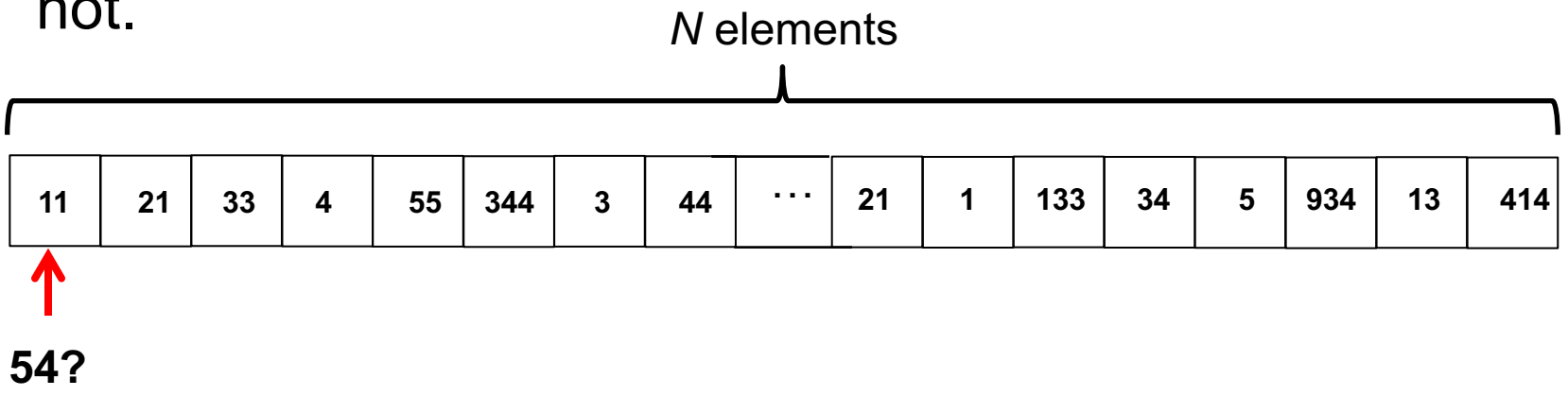
- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



54?

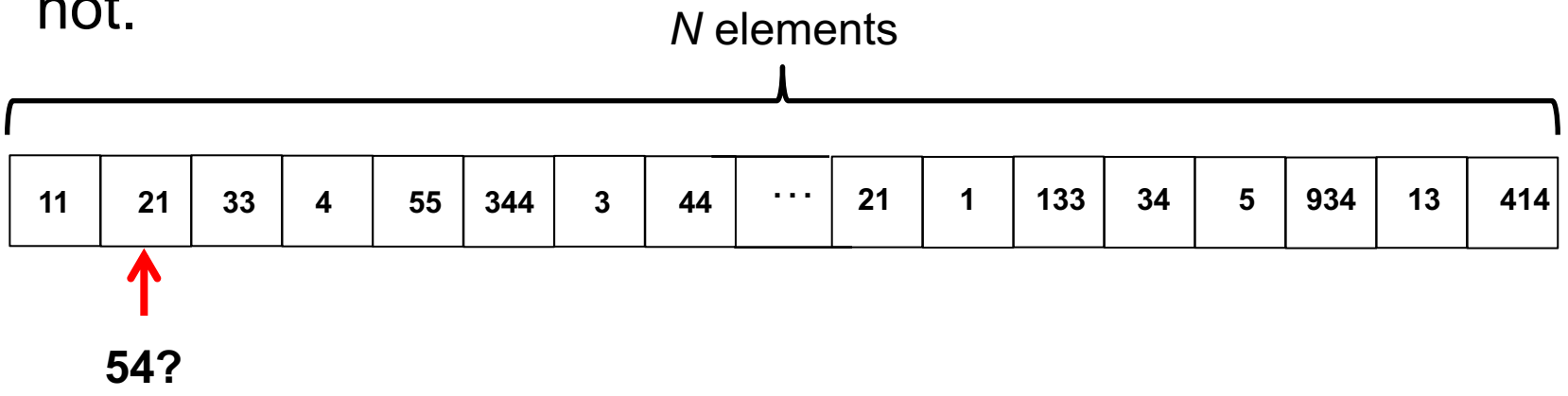
Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



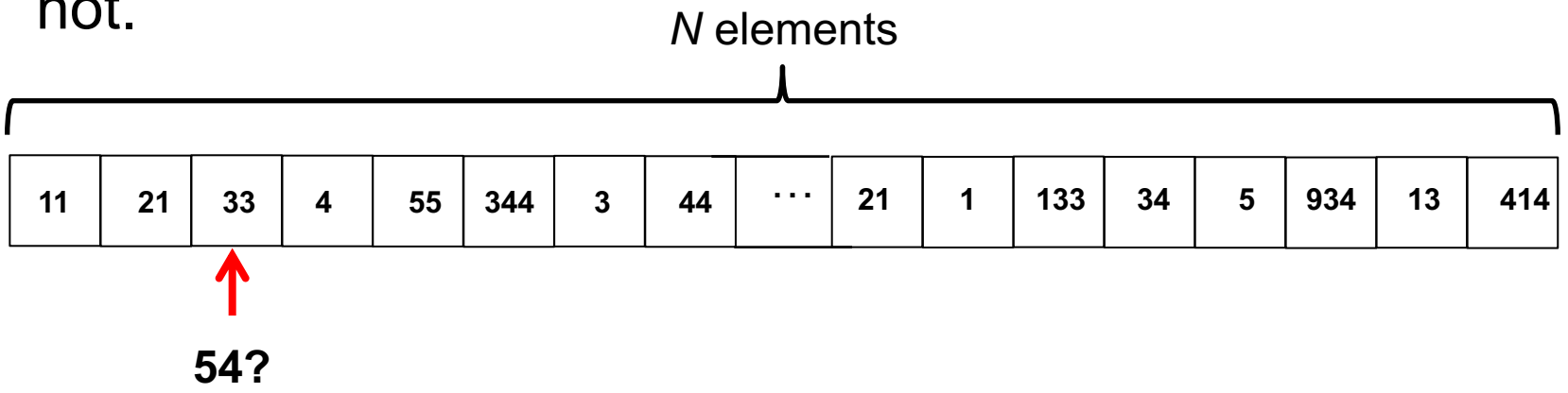
Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



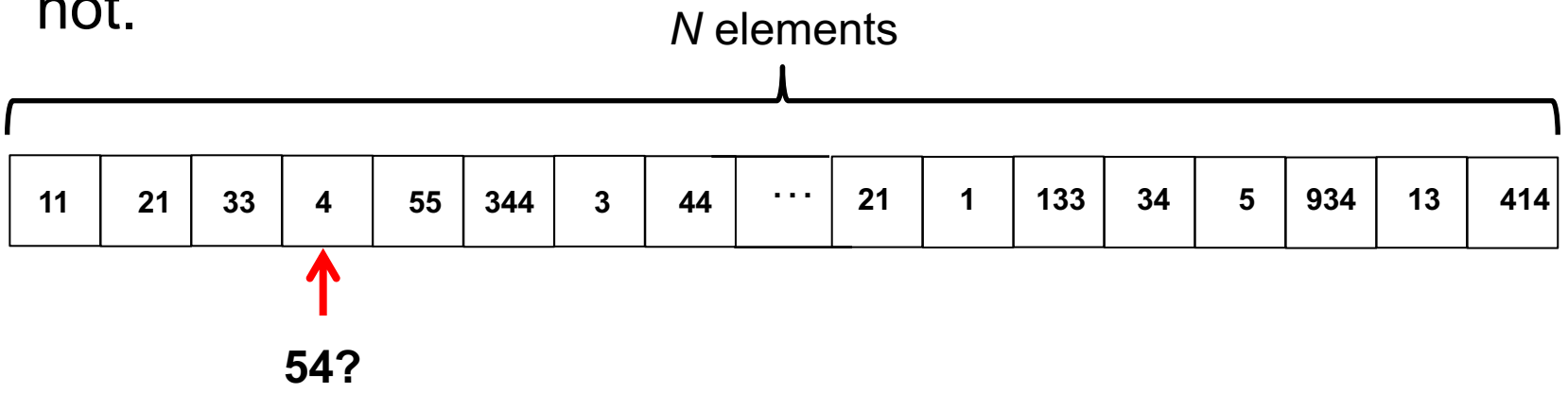
Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



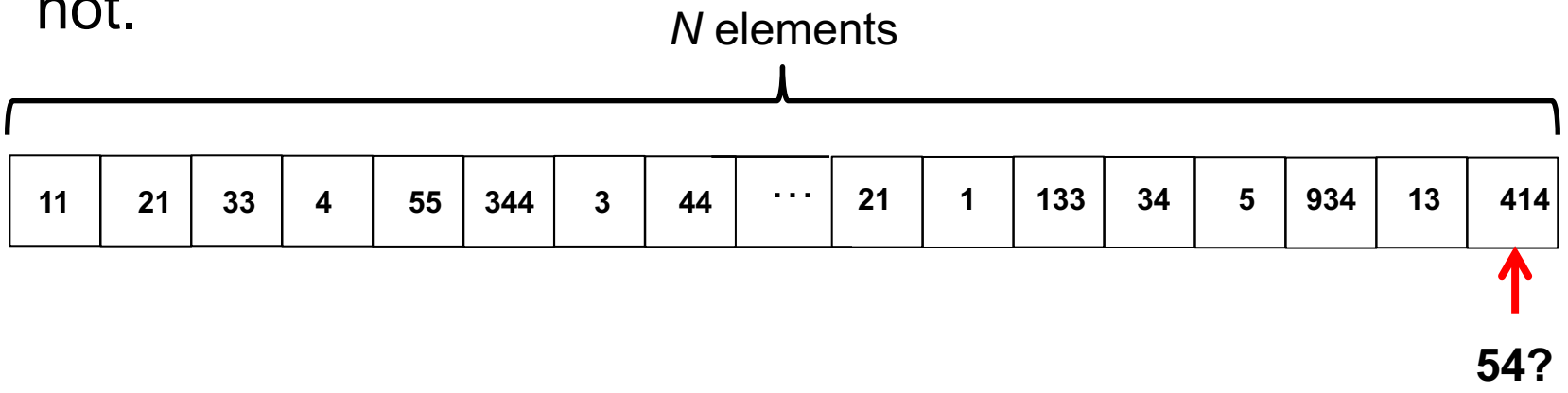
Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



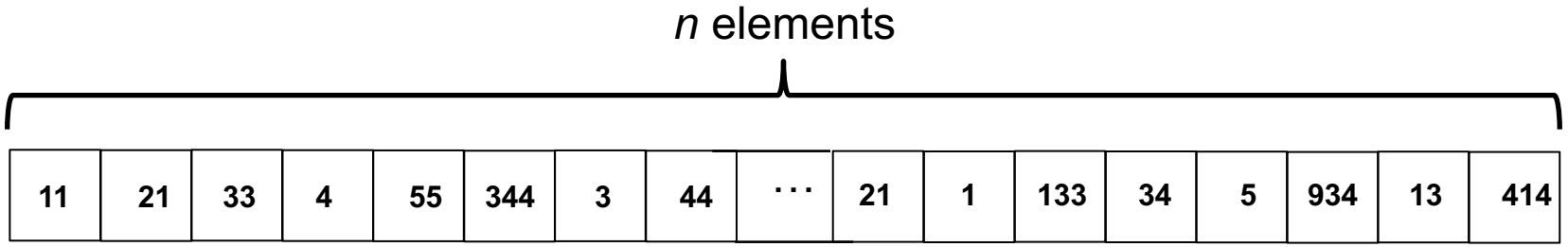
Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



Searching an Array of Size N for A specific Element

- An array of size *n* is filled with unsorted randomly placed, numbers.
- Goal: Find whether a specific item *x* is in this array or not.



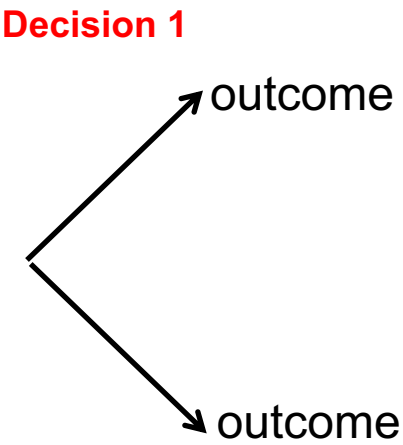
- We observe that this problem can be solved in *n* steps, where *n* is the size of the input.
- This problems belongs to computational complexity class of P.
- The class P is composed of such problems that can be solved in Polynomial time (or number of steps) with respect to the size of the input. (O(n)=n, n^2, n^3,…)

Complexity Classes

P: Can be solved in polynomial time.

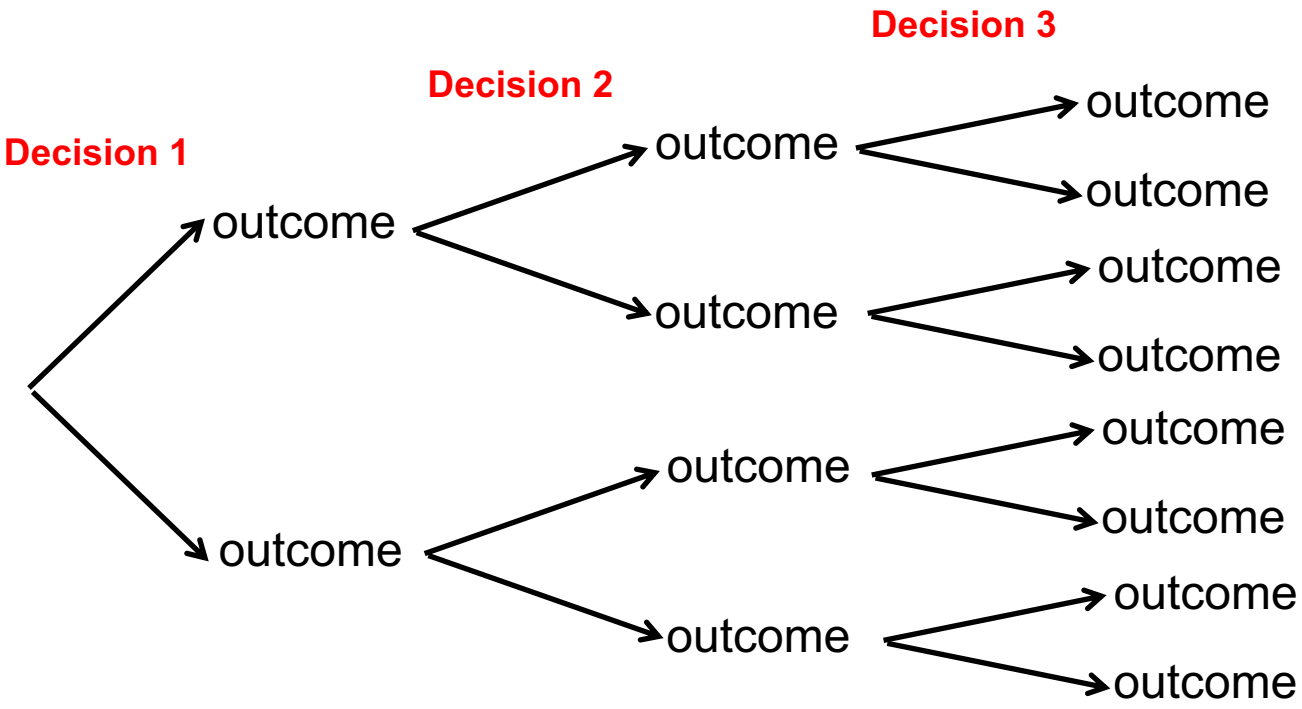
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes.



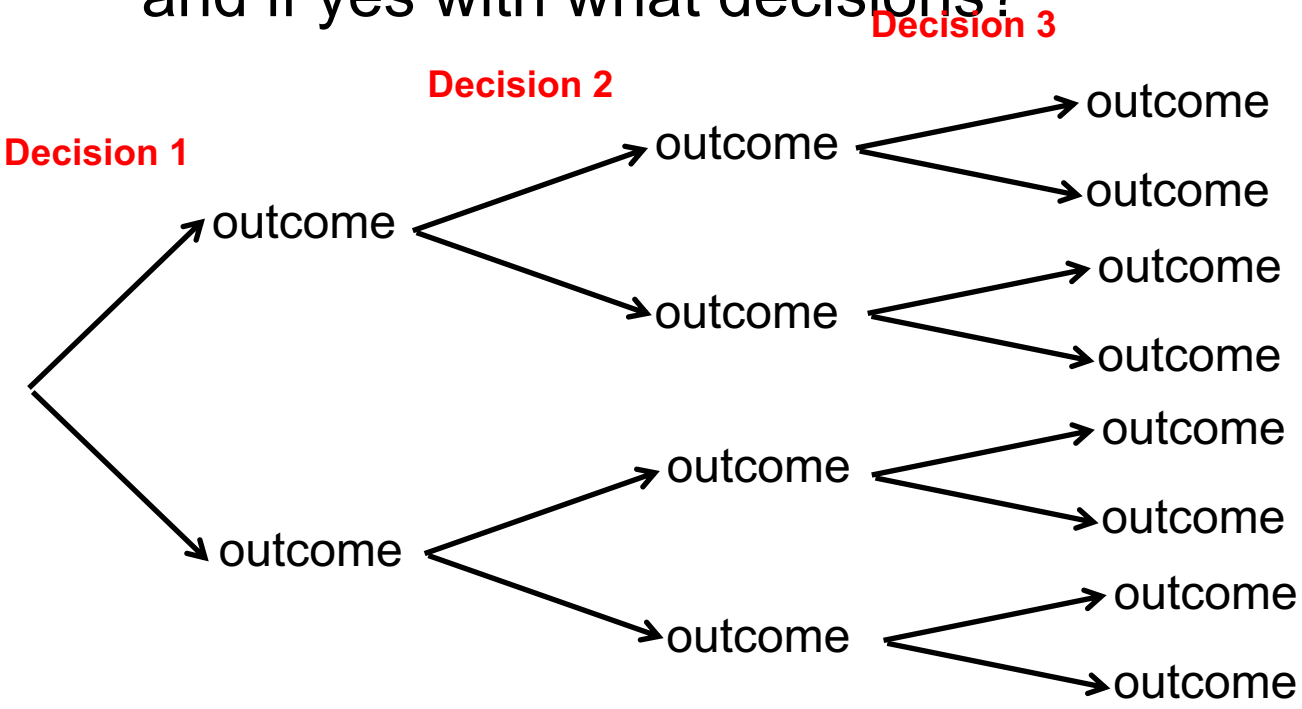
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes. And making further decisions with two outcomes for each decision.



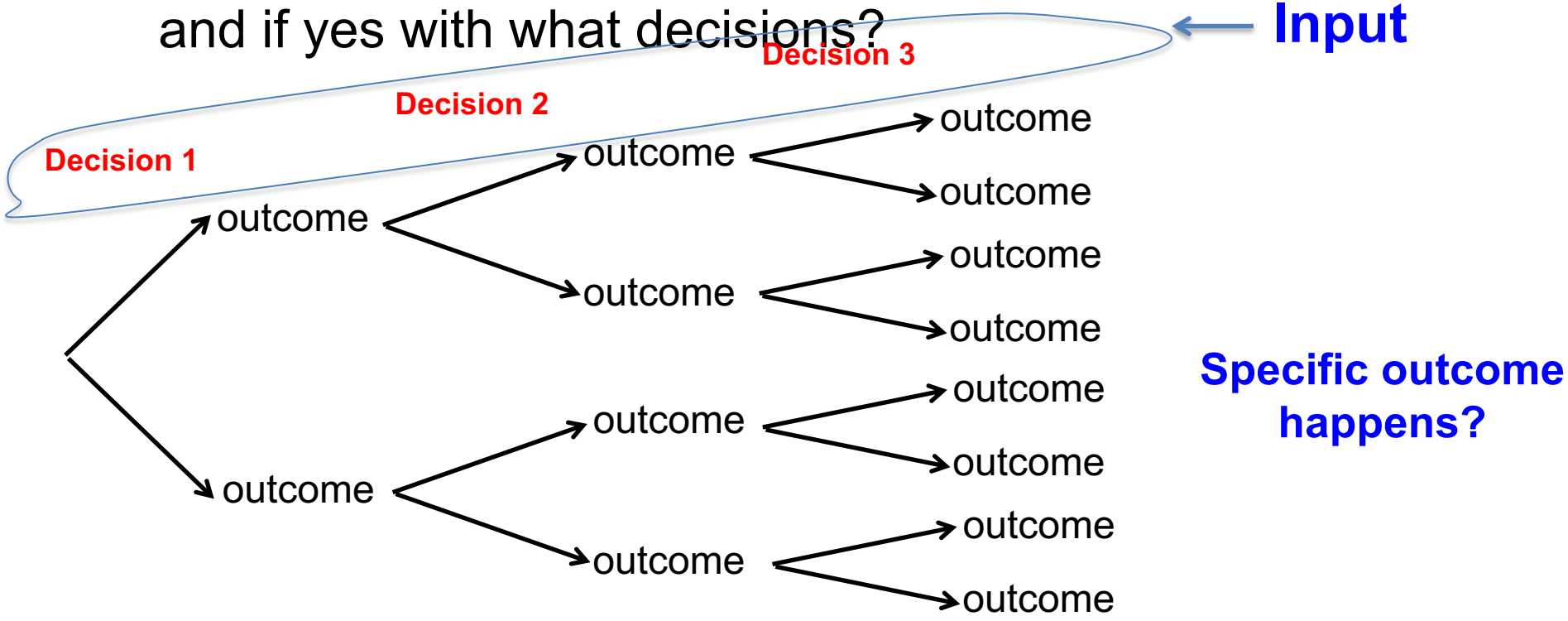
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



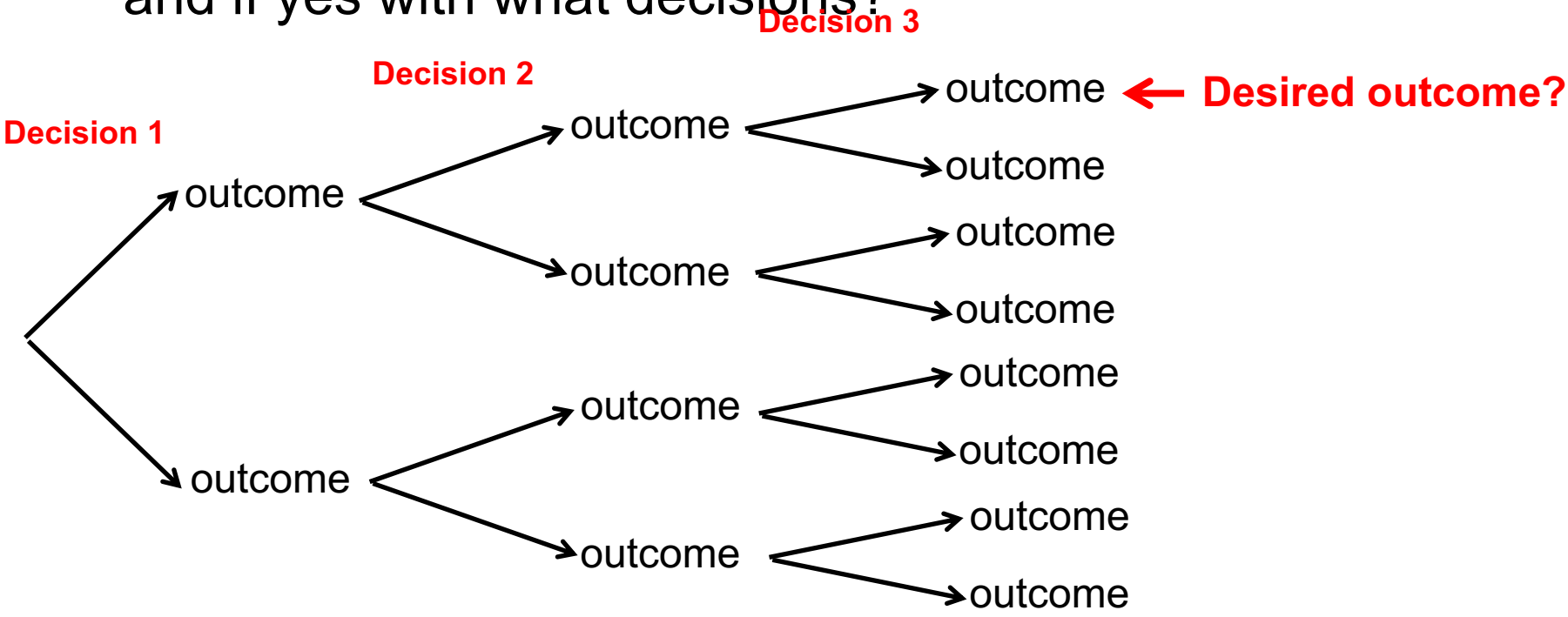
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



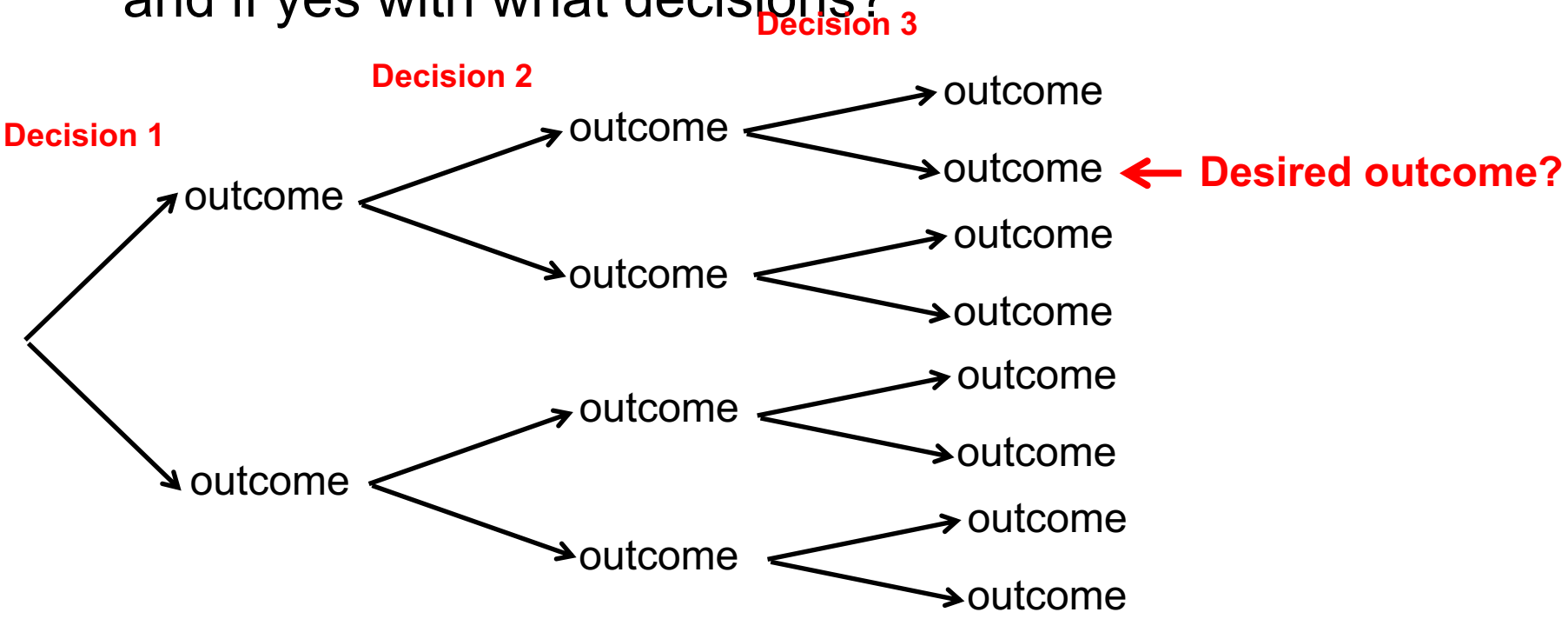
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



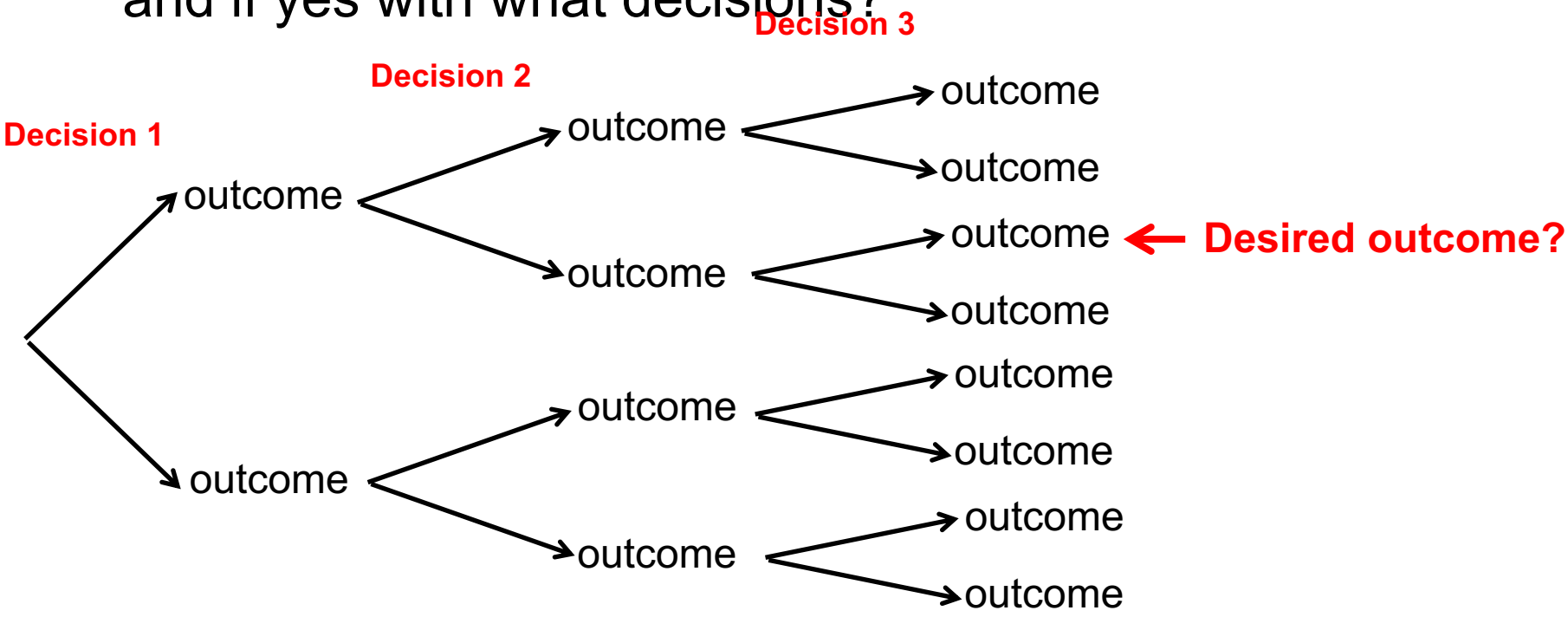
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible options. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



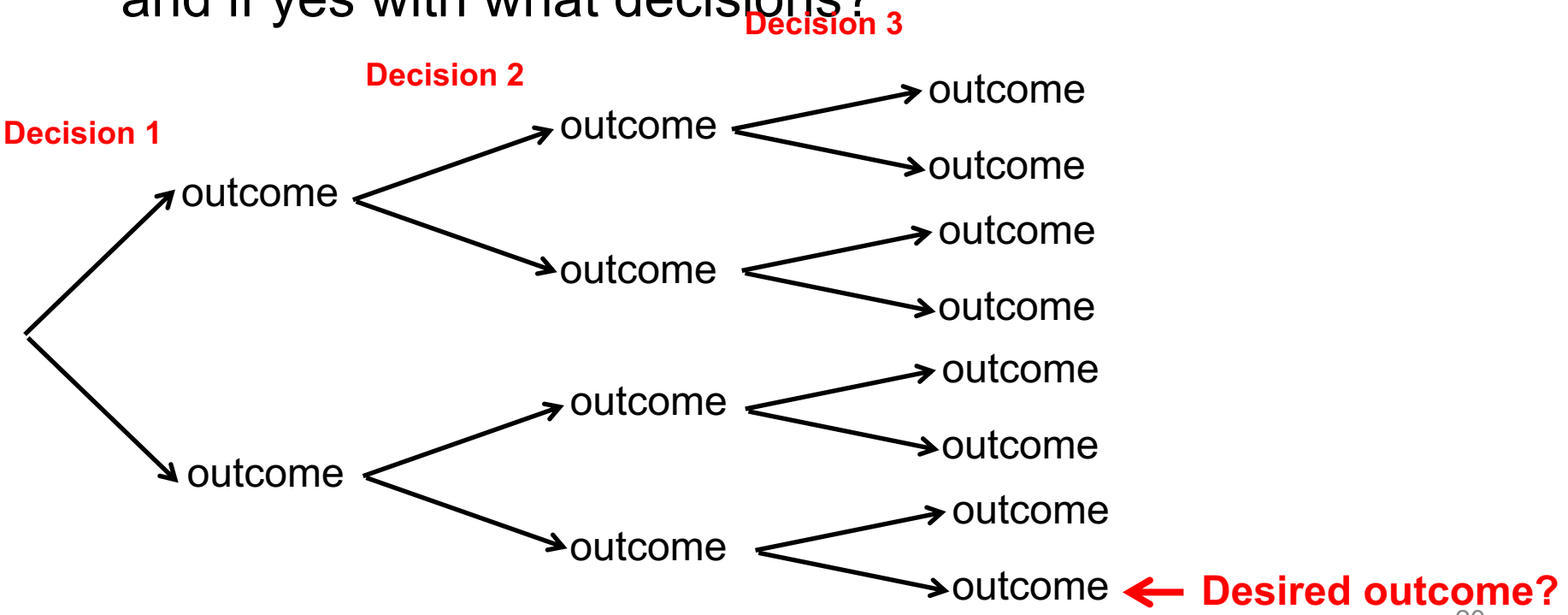
A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible options. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



A Series of Decisions to Obtain a Desired Outcome.

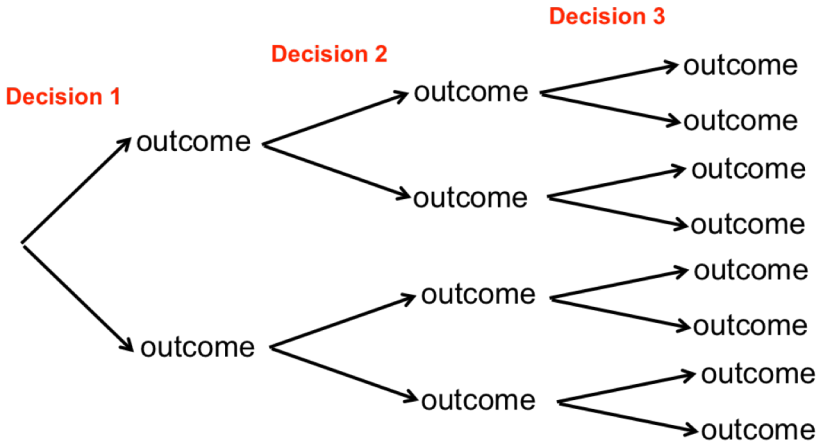
- Imagine making a decision with two possible options. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?



A Series of Decisions to Obtain a Desired Outcome.

- Imagine making a decision with two possible outcomes. And making further decisions with two outcomes for each decision.
- Goal: Find whether a specific outcome happens or not, and if yes with what decisions?

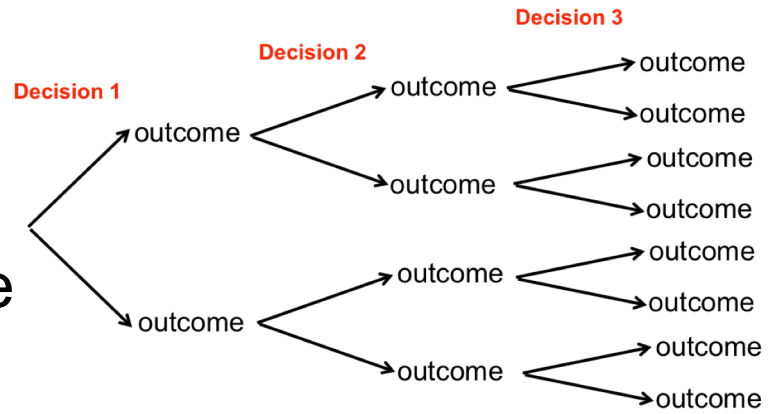
- Here the number of decisions is the size of the problem.
- The number of possible outcomes to be checked exponentially increases by the size of the input.



A Series of Decisions to Obtain a Desired Outcome.

- This problem belongs to complexity class of NP.
- NP is a class of decision problems that:
  1. A possible solution (in this case a series of decisions) can be verified in polynomial time.
  2. The problem can be solved in Non-deterministic Polynomial time.

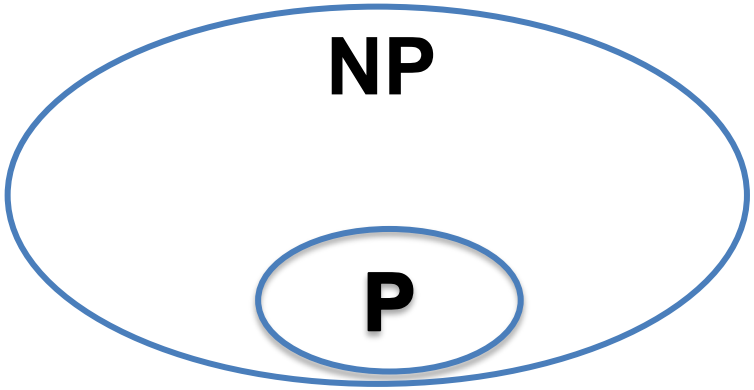
Informal version of (2): We need a "lucky" algorithm to solve in polynomial time. In practice we cannot design a lucky algorithm.



So far practical algorithms require exponential time.

Complexity Classes

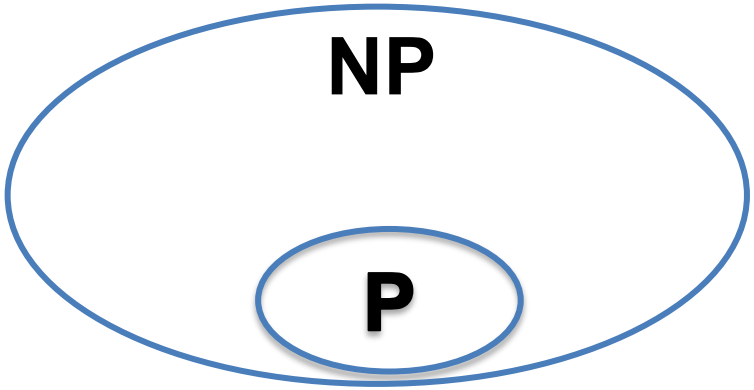
P: Can be solved in polynomial time.
NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.



Complexity Classes

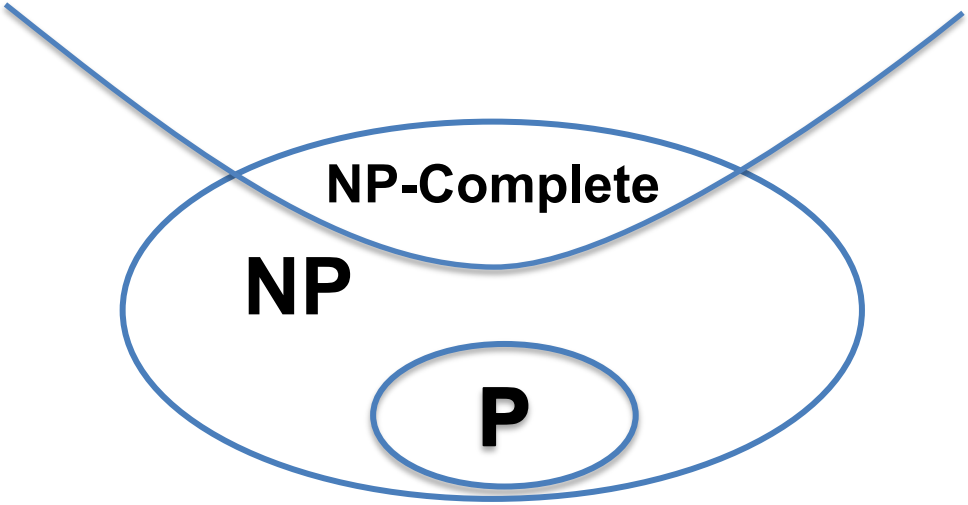
P: Can be solved in polynomial time.
NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.

Not all NP problems are equally hard!



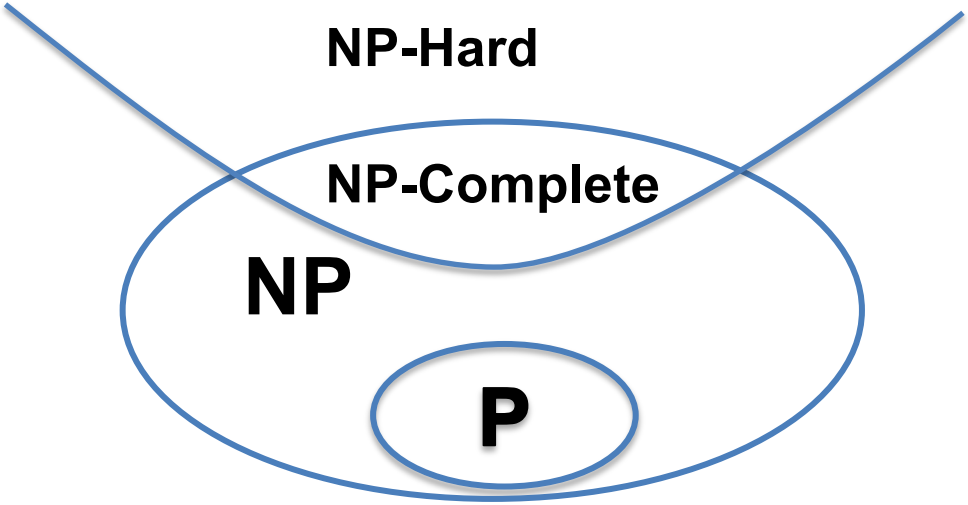
Complexity Classes

P: Can be solved in polynomial time.
NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.
NP-Complete: The hardest NP problems.



Complexity Classes

P: Can be solved in polynomial time.
NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.
NP-Complete: The hardest NP problems.
NP-Hard: At least as hard as the hardest NP problem.



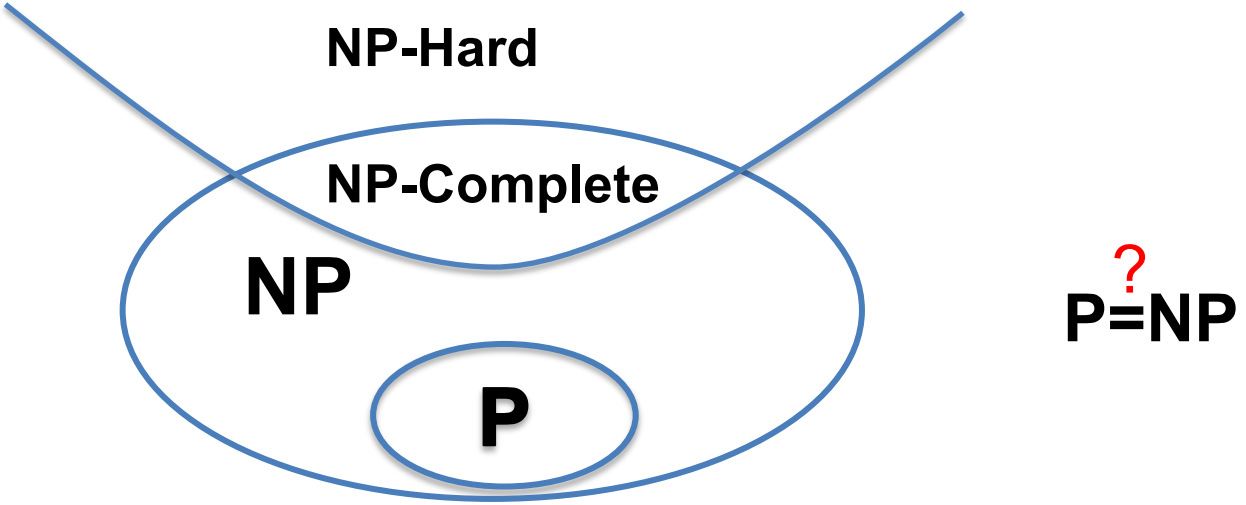
Complexity Classes: One Million Dollar Question!

P: Can be solved in polynomial time.

NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.

NP-Complete: The hardest NP problems.

NP-Hard: As hard as the hardest NP problem.



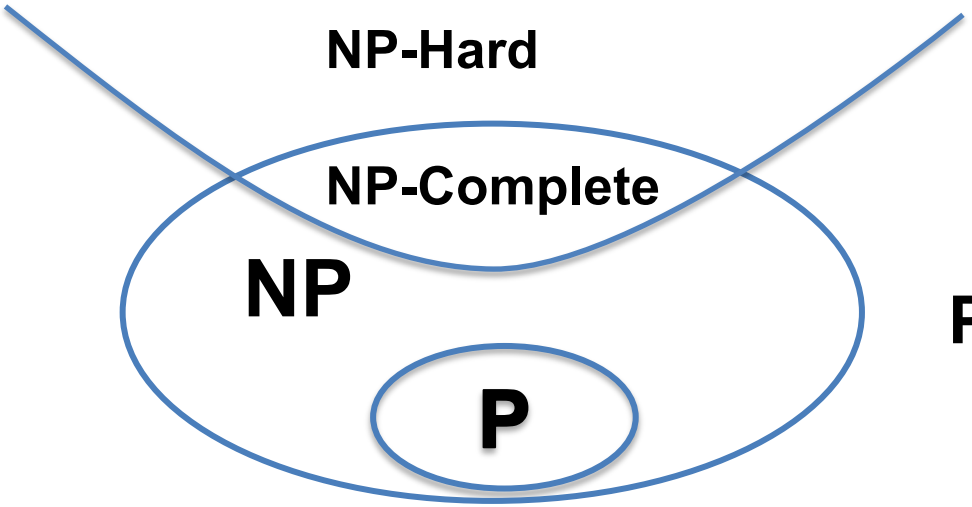
Complexity Classes: One Million Dollar Question!

P: Can be solved in polynomial time.

NP: Solutions verifiable in polynomial time. Can be solved in-deterministic polynomial time. (informal definition) No polynomial solution yet.

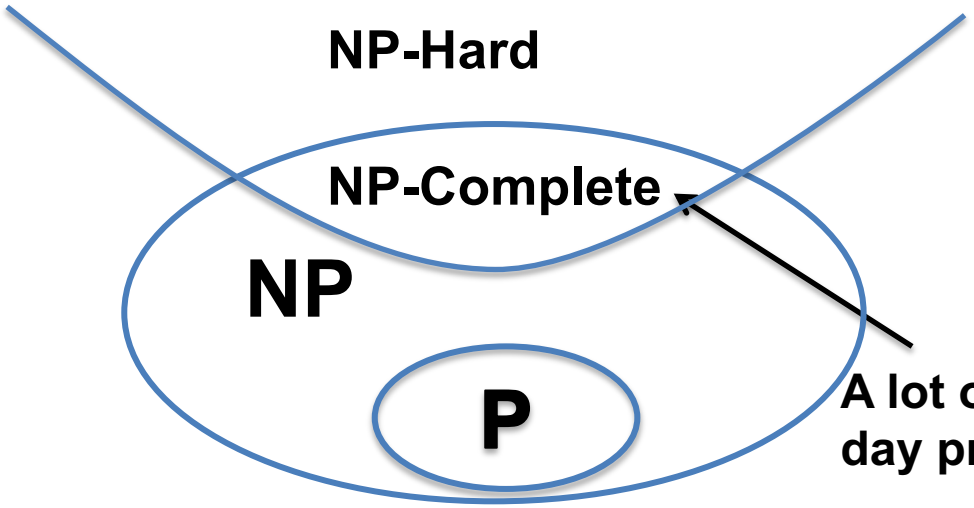
NP-Complete: The hardest NP problems.

NP-Hard: As hard as the hardest NP problem.

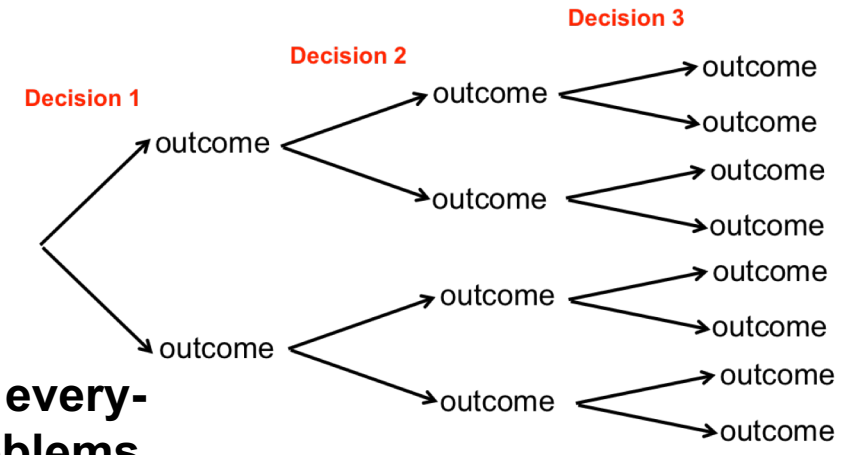


P≠NP many believe

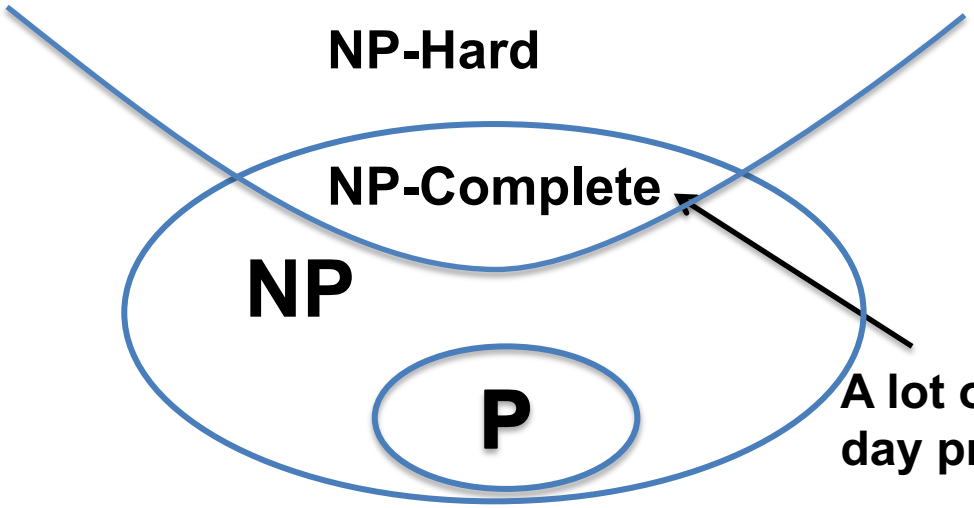
N≠NP Implications



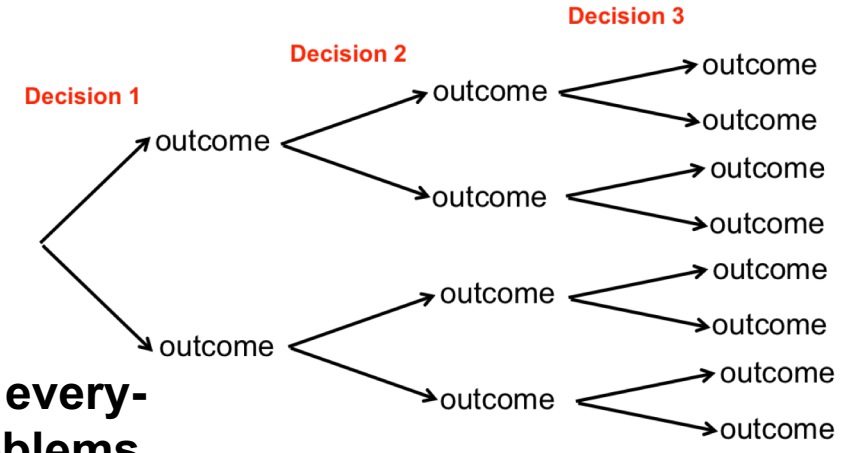
A lot of every-day problems



N≠NP Implications

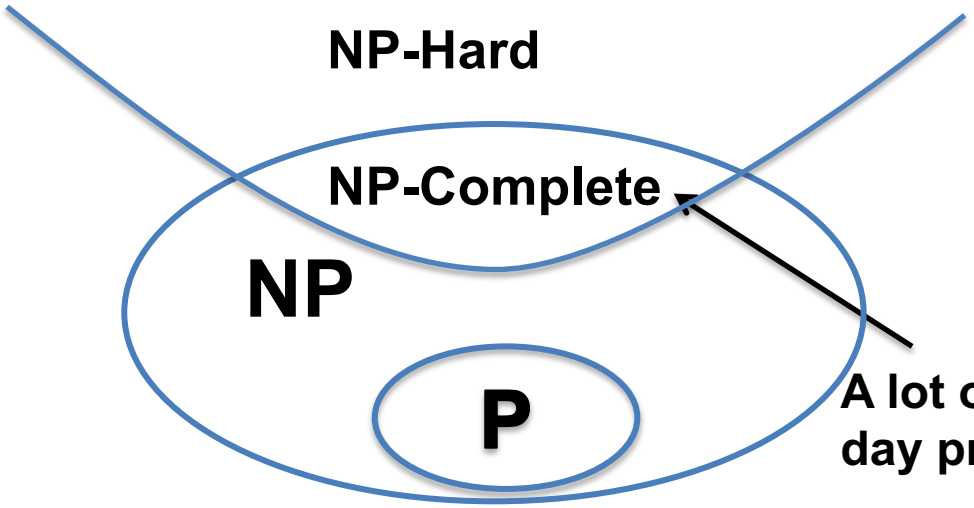


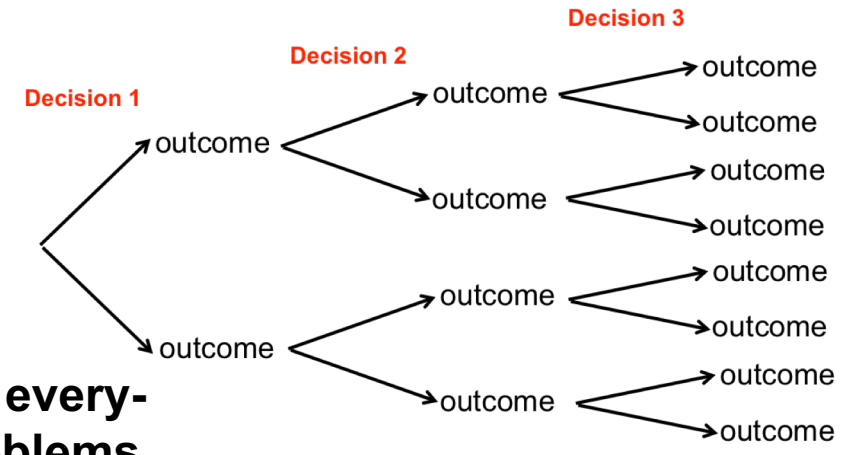
A lot of every-day problems



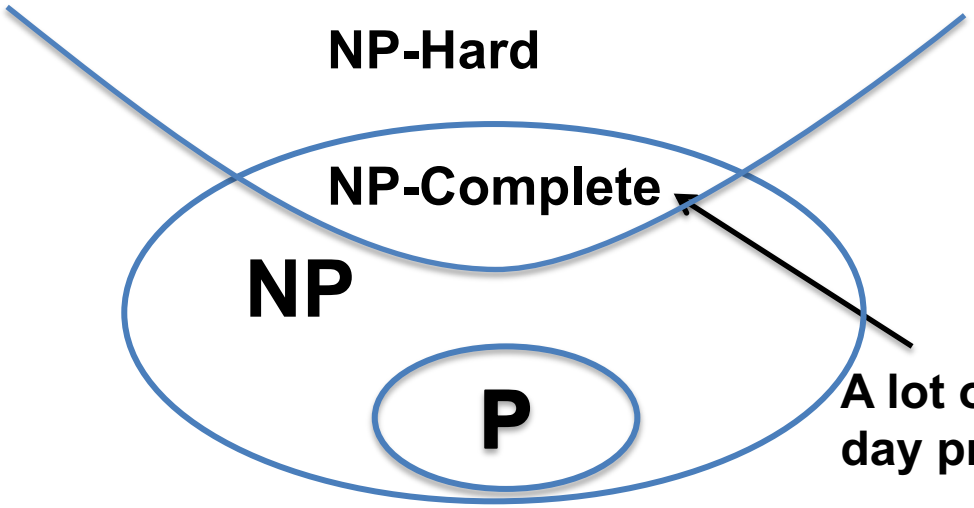
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

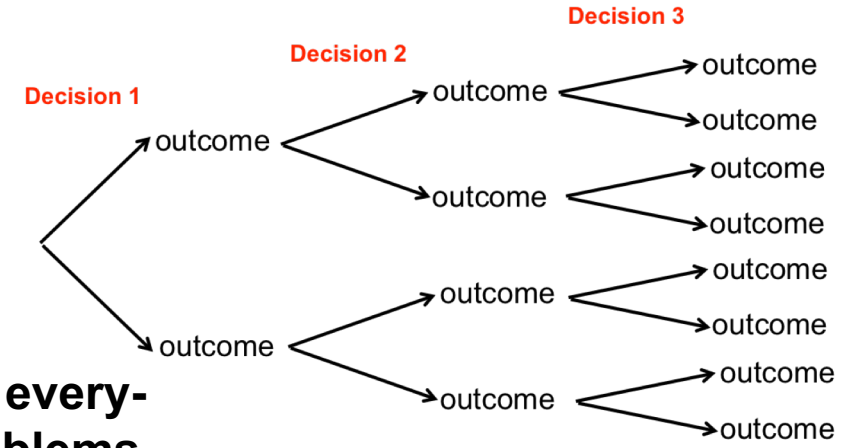
Complexity Classes





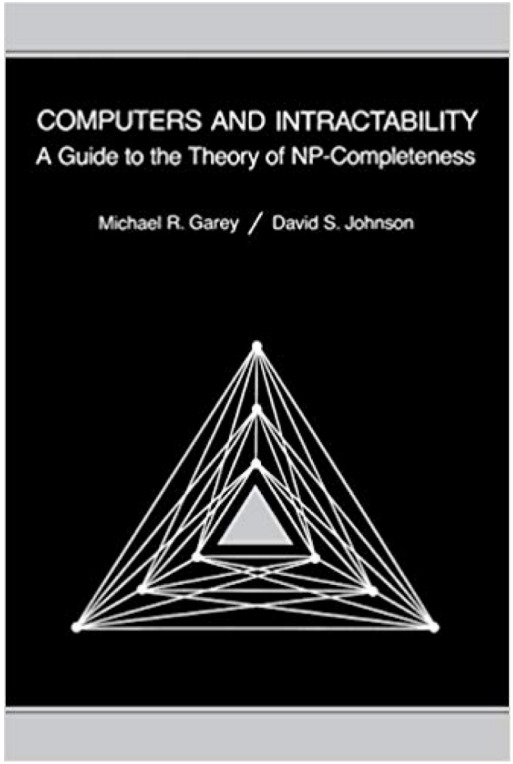
Complexity Classes





Strange: I went forward in time to see all the possible outcomes of the present situation.
Peter: How many did you see?
Strange: Fourteen million six hundred and five.

$Log_2(14{,}000{,}605) \approx 24$

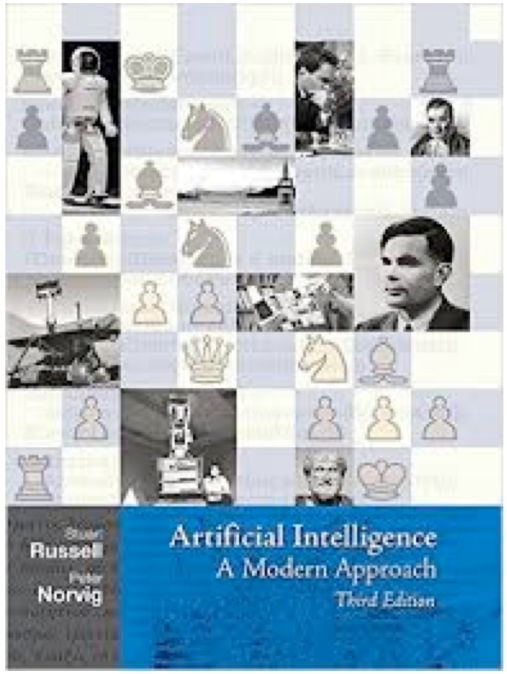


Computers and intractability : a guide to the theory of NP-completeness
Michael R. Garey, David S. Johnson.

- Computational complexity is about the order of growth with respect to input size, not absolute time given a specific sized input.

Four Different Text-Book AI Approaches

- Search (chapters 3-6)
- Logic and Reasoning (chapters 7-11)
- Probabilistic Reasoning (chapters 12-17)
- Learning (chapters 18-21)



video

$$\nabla \cdot \boldsymbol{E} = \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} + \frac{\partial E_z}{\partial z}$$

$$(\nabla \times \boldsymbol{E})_x = \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}$$

$$(\nabla \times \boldsymbol{E})_y = \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}$$

$$(\nabla \times \boldsymbol{E})_z = \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}$$

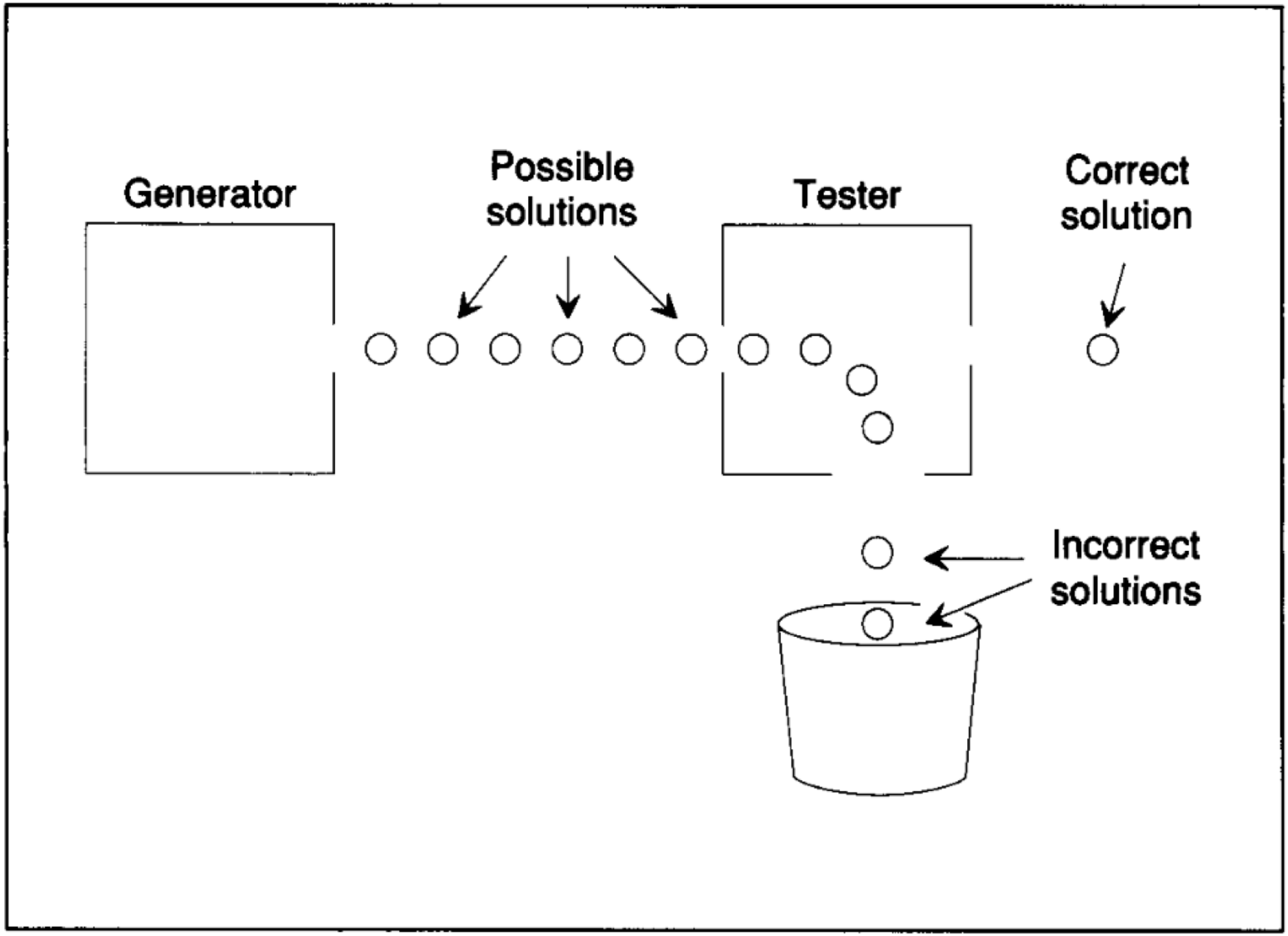
$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon_0} \qquad \nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}$$

$$\nabla \cdot \boldsymbol{B} = 0 \qquad c^2 \nabla \times \boldsymbol{B} = \frac{\boldsymbol{j}}{\epsilon_0} + \frac{\partial \boldsymbol{E}}{\partial t}$$

$$\boldsymbol{E} = \frac{q}{4\pi\epsilon_0}\left[\frac{\boldsymbol{e}_{r'}}{r'^2} + \frac{r'}{c}\frac{d}{dt}\left(\frac{\boldsymbol{e}_{r'}}{r'^2}\right) + \frac{1}{c^2}\frac{d^2}{dt^2}\boldsymbol{e}_{r'}\right],$$

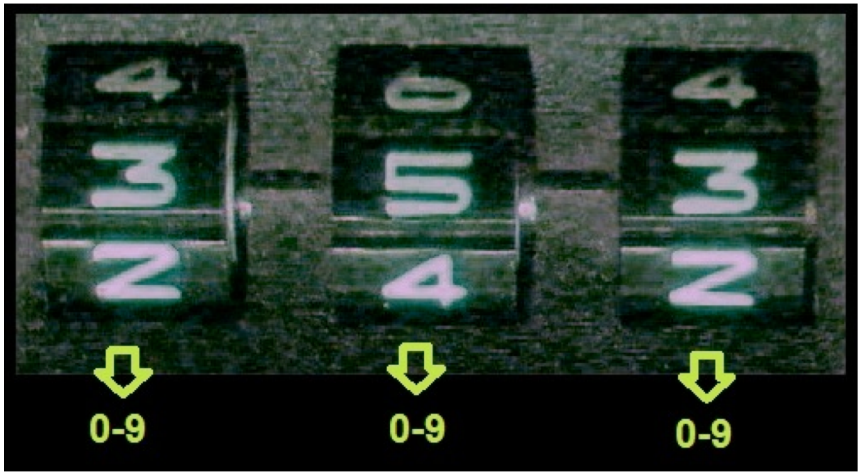
$$c\boldsymbol{B} = \boldsymbol{e}_{r'} \times \boldsymbol{E}. \qquad \text{Eq. (28.3), Vol. I.}$$

Search as a tool for problem solving and AI



Picture from Artificial Intelligence, Patrick Winston

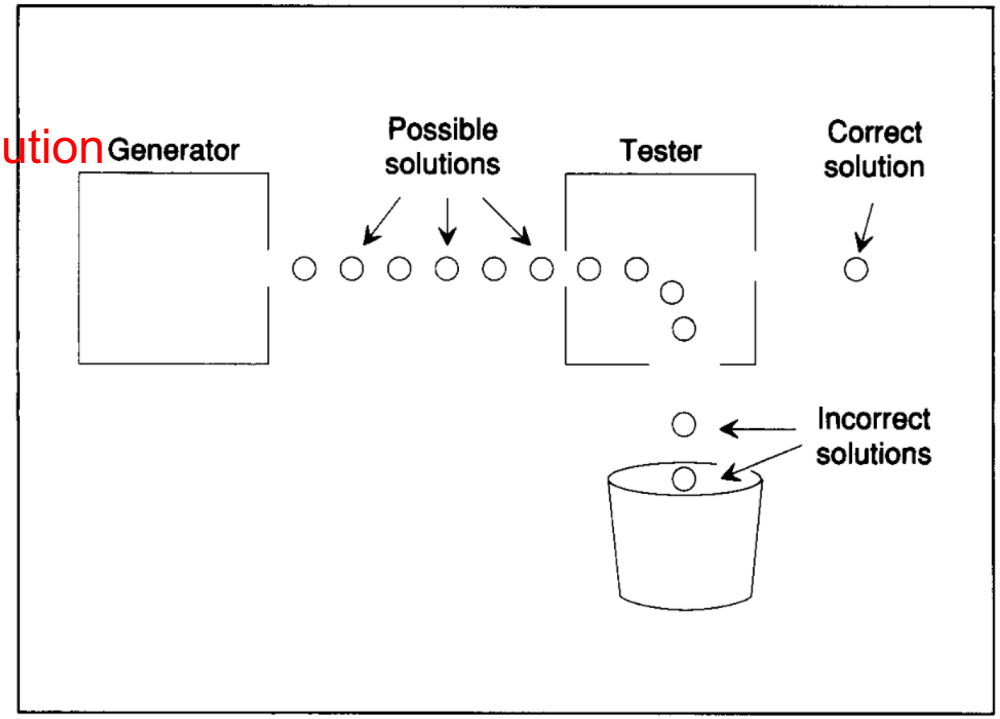
Search Mechanisms
Brute-Force (Exhaustive) Search



Random Search Methods: Pure Random Search

- Randomly (with a uniform distribution) choose candidates in the solution space up until max number of iterations performed, or an adequate fitness reached.

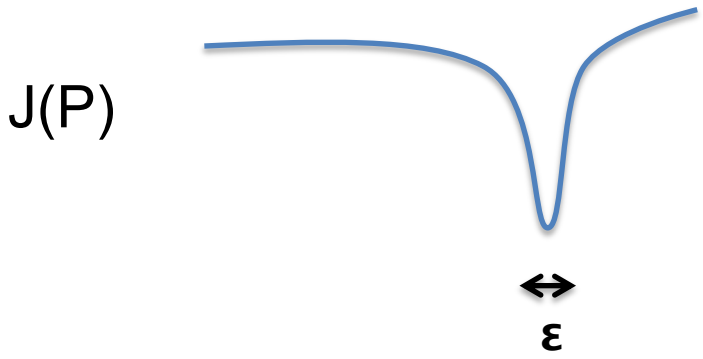
Randomly solution



Random Search Methods: Pure Random Search

- Performance of pure random search method experiences large variation.
- In an n-dimensional function J, the expected number of iterations until a pure random sample point falls within the **ε** neighborhood of the minimal point of J is:

$$\left(\frac{1}{\varepsilon}\right)^n$$



- The process of learning a function can be considered as a search in Euclidean space for a set of weights that implements the function.