Computational Complexity Theory

Lecture 1: Intro; Turing machines; Class P and NP

Indian Institute of Science

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

a. Decision problem

Example: Is vertex t reachable from vertex s in graph G?

(…output is YES/NO)

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

  a. Decision problem

  b. Search problem

Example: Find a satisfying assignment of a boolean formula, if it exists.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

  a. Decision problem

  b. Search problem

  c. Counting problem

  Example: Find the number of cycles in a graph

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

    a. Decision problem

    b. Search problem

    c. Counting problem

    d. Optimization problem

Example: Find a minimum size vertex cover in a graph

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- **Algorithms** are methods of solving problems; they are studied using formal models of computation, like Turing machines.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- **Algorithms** are methods of solving problems; they are studied using formal models of computation, like Turing machines.
  - a memory with head (like a RAM)

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- **Algorithms** are methods of solving problems; they are studied using formal models of computation, like Turing machines.
    - a memory with head (like a RAM)
    - a finite control (like a processor)

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- **Algorithms** are methods of solving problems; they are studied using formal models of computation, like Turing machines. (…more later)

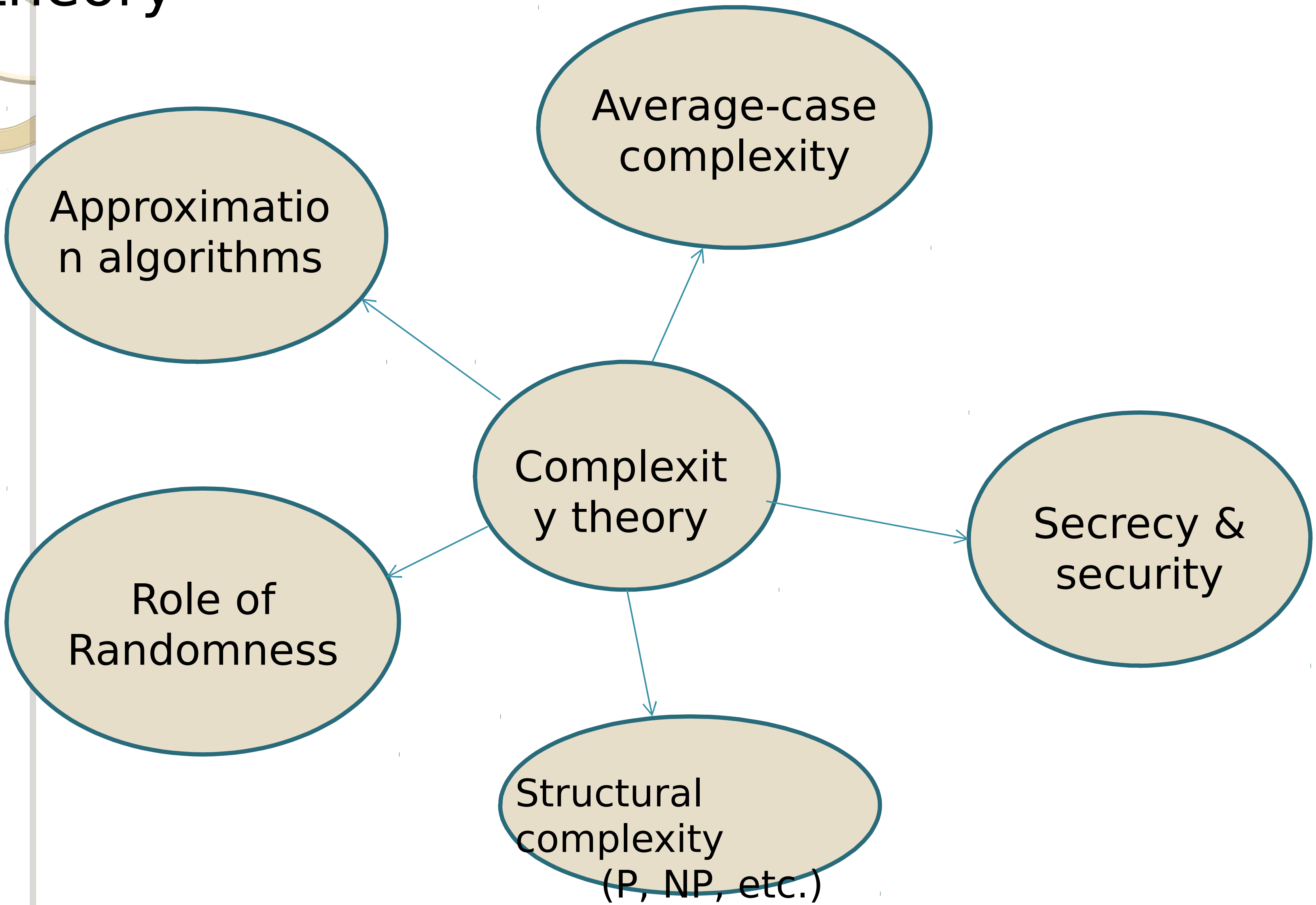
About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **resources** (required by models of computation) can be:
  - Time (bit operations)
  - Space (memory cells)

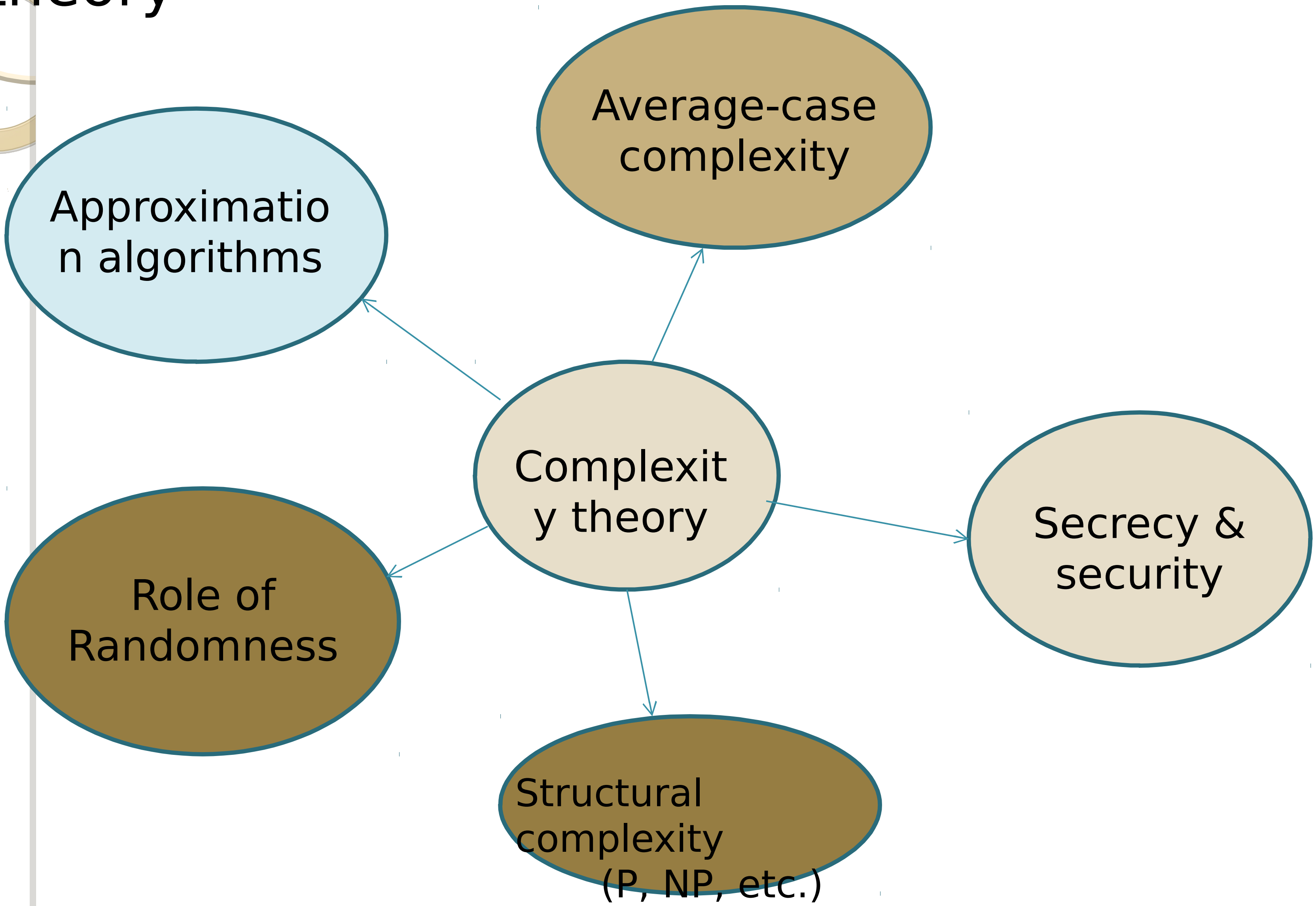
About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **resources** (required by models of computation) can be:
  - Time (bit operations)
  - Space (memory cells)
  - Random bits
  - Communication (bit exchanges)

Some topics in complexity theory



Some topics in complexity theory



Structural Complexity

- Classes P, NP, co-NP... NP-completeness.

Structural Complexity

- Classes P, NP, co-NP... NP-completeness.

  How hard is it to check satisfiability of a boolean formula that has exactly one or no satisfying assignment?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.

How much space is required to check s-t connectivity?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.

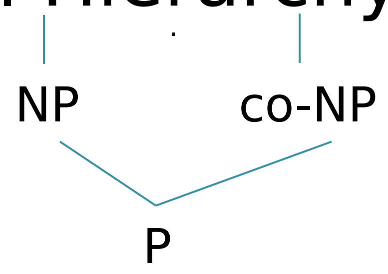
Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.

How hard is it to count the number of perfect matchings in a graph?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.
- Polynomial Hierarchy.



How hard is it to check that largest independent set in G has size exactly k ?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.
- Polynomial Hierarchy.

How hard is it to find a minimum size circuit computing the same boolean function as a given boolean circuit?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.
- Polynomial Hierarchy.
- Boolean circuits and circuit lower bounds.

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Counting complexity.
- Polynomial Hierarchy.
- Boolean circuits and circuit lower bounds.

A central topic in classical complecity theory;
Proving P≠NP boils down to showing circuit lower bounds.

Role of Randomness in Computation

- Probabilistic complexity classes.

Role of Randomness in Computation

- Probabilistic complexity classes.
  - Does randomization help in improving efficiency?
  - Quicksort has O(n log n) expected time but O(n^2) worst case time.
  - Can SAT be solved in polynomial time using randomness?

    Theorem (Schoening, 1999): 3SAT can be solved in *randomized* $O((4/3)^n)$ time.

Role of Randomness in Computation

- Probabilistic complexity classes.
- Probabilistically Checkable Proofs (PCPs).

Role of Randomness in Computation

- Probabilistic complexity classes.
- Probabilistically Checkable Proofs (PCPs).

Unconditional hardness of approximation results

Theorem (Hastad, 1997): If there’s a poly-time algorithm
to compute an assignment that satisfies at least
7/8 + e fraction of the clauses of an input 3SAT,
for a constant
e > 0, then P = NP.

Role of Randomness in Computation

- Probabilistic complexity classes.
- Probabilistically Checkable Proofs (PCPs).
- A glimpse of randomness extractors and pseudorandom generators (if time permits).

Role of Randomness in Computation

- Probabilistic complexity classes.
- Probabilistically Checkable Proofs (PCPs).
- A glimpse of randomness extractors and pseudorandom generators (if time permits).

Can every polynomial-time randomized algorithm be derandomized to a deterministic polynomial-time algorithm?

Average-case Complexity

- Distributional problems (if time permits).

How hard is it to solve the clique problem on inputs chosen from a "real-life" distribution?

Average-case Complexity

- Distributional problems (if time permits).
- Hardness amplification: From weak to strong hardness.

In cryptographic applications, we need hard on average functions for secure encryptions.

Basic Course Info

- **Course title:** Computational Complexity Theory
- **Credits:** 3:1 **Instructor:** Chandan Saha
- **TA:** Nikhil Gupta and Vineet Nair
- **Class timings** : Tuesday, Thursday: 3:30-5 pm.
- **Venue:** CSA lecture hall 252.
- **Primary reference:** Computational Complexity – A Modern Approach by Sanjeev Arora and Boaz Barak.

Basic Course Info

- **Prerequisites** : Basic familiarity with algorithms; Some *mathematical maturity* will be helpful.

- **Grading policy** : Assignments - 30%
  Mid-term - 35%
  End-term - 35%

Course homepage:

drona.csa.iisc.ernet.in/~chandan/courses/complexity17/home.html

Let’s begin…

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model.

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a <u>model of computation</u>. Turing machine is one such *natural* model.
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a <u>model of computation</u>. Turing machine is one such *natural* model.
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules
- Turing machines $\longleftrightarrow$ A mathematical way to describe algorithms.

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model.
- A TM consists of:
    - Memory tape(s)
    - A finite set of rules

(e.g. of a physical realization a TM is a simple adder)

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)

has a blank symbol

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)
- Q is a finite set of states. (special states: q_{start} , q_{halt})
- δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L,S,R\}^k$

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)
- Q is a finite set of states. (special states: q_{start} , q_{halt})
- δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L,S,R\}^k$

known as transition function; it captures the dynamics of M

Turing Machines: Computation

- Start configuration.
  - All tapes other than the input tape contain blank symbols.
  - The input tape contains the input string.
  - All the head positions are at the start of the tapes.
  - The machine is in the start state q_{start} .

Turing Machines: Computation

- Start configuration.
  - All tapes other than the input tape contain blank symbols.
  - The input tape contains the input string.
  - All the head positions are at the start of the tapes.
  - The machine is in the start state q_{start} .
- Computation.
  - A **step of computation** is performed by applying δ.
- Halting.
  - Once the machine enters q_{halt} it stops computation.

Turing Machines: Running time

- Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ and $T: \mathbb{N} \rightarrow \mathbb{N}$ and M be a Turing machine.

- Definition. We say M ***computes*** f if on every x in $\{0,1\}^*$, M halts with $f(x)$ on its output tape beginning from the start configuration with x on its input tape.

Turing Machines: Running time

- Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ and $T: \mathbb{N} \rightarrow \mathbb{N}$ and M be a Turing machine.

- Definition. We say M ***computes*** f if on every x in $\{0,1\}^*$, M halts with f(x) on its output tape beginning from the start configuration with x on its input tape.

- Definition. M computes f *in T(|x|)* ***time***, if M computes f and for every x in $\{0,1\}^*$, and M halts within T(|x|) steps of computation.

Turing Machines

- In this course, we would be dealing with
  - Turing machines that halt on every input.
  - Computational problems that can be solved by Turing machines.

Turing Machines

- In this course, we would be dealing with
  - Turing machines that <u>halt on every input</u>.
  - Computational problems that can be solved by Turing machines.
- Can every computational problem be solved using Turing machines?

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - A typical input instance:

$$x^2y + 5y^3 = 3$$

$$x^2 + z^5 - 3y^2 = 0$$

$$y^2 - 4z^6 = 0$$

Integer solutions for x, y, z?

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

(Hilbert's tenth problem, 1900)

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

- Theorem. There does not exist any algorithm (realizable by a TM) to solve this problem.

Why Turing Machines?

- TMs are natural and intuitive.

- Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated by a Turing machine".*

--- [quote from Arora-Barak's book]

Why Turing Machines?

- TMs are natural and intuitive.

- Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated by a Turing machine".*

--- [quote from Arora-Barak's book]

- Several other computational models can be simulated by TMs.

Basic facts about TMs

Turing Machines

- Time constructible functions. A function $T: \mathbb{N} \to \mathbb{N}$ is *time constructible* if $T(n) \geq n$ and there's a TM that computes the function that maps x to $T(|x|)$ in $O(\underbrace{T(|x|)}_{\text{in binary}})$ time.

- Examples: $T(n) = n^2$, or 2^n, or $n \log n$

Turing Machines: Robustness

- Let f: {0,1}* → {0,1}* and T: N → N be a time constructible function.

- Binary alphabets suffice.
  - If a TM M computes f in $T(n)$ time using Γ as the alphabet set then there's another TM M' that computes f in time $4.\log|\Gamma| . T(n)$ using {0, 1, blank} as the alphabet set.

Turing Machines: Robustness

- Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ and $T: \mathbb{N} \rightarrow \mathbb{N}$ be a time constructible function.

- Binary alphabets suffice.
  - If a TM M computes f in $T(n)$ time using Γ as the alphabet set then there's another TM M' that computes f in time $4.\log |\Gamma| \, . \, T(n)$ using $\{0, 1, \text{blank}\}$ as the alphabet set.

- A single tape suffices.
  - If a TM M computes f in $T(n)$ time using k tapes then there's another TM M' that computes f in time $5k \, . \, T(n)^2$ using a single tape that is used for input, work and output.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

  ...simply encode the description of the TM.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

...invalid strings map to a fixed, trivial TM.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

- Every TM has infinitely many string representations.

  … allow padding with arbitrary number of 0's

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

- Every TM has infinitely many string representations.

{0,1} string $\quad \alpha \longleftrightarrow M_\alpha \quad$ TM corresponding to α

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.
- Every string over {0,1} represents some TM.
- Every TM has infinitely many string representations.
- A TM (i.e. its string representation) can be given as an input to another TM !!

Universal Turing Machines

- Theorem. There exists a TM U that on every input x, α in $\{0,1\}^*$ outputs $M_\alpha(x)$.

- Further, if M_α halts within T steps then U halts within $C.\ T.\ \log T$ steps, where C is a constant that depends only on M_α 's alphabet size, number of states and number of tapes.

Universal Turing Machines

- Theorem. There exists a TM U that on every input x, α in $\{0,1\}^*$ outputs $M_\alpha(x)$.

- Further, if M_α halts within T steps then U halts within $C.\ T.\ \log T$ steps, where C is a constant that depends only on M_α's alphabet size, number of states and number of tapes.

- Physical realization of UTMs are modern day electronic computers.

Complexity classes
P and NP

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.

- Decision problems can be naturally identified with boolean functions, i.e. functions from $\{0,1\}^*$ to $\{0,1\}$.

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.
- Decision problems can be naturally identified with boolean functions, i.e. functions from $\{0,1\}^*$ to $\{0,1\}$.
- Boolean functions can be naturally identified with sets of $\{0,1\}$ strings, also called languages.

Decision Problems

Decision problems ⟷ Boolean functions ⟷ Languages

- Definition. We say a TM M *decides a* language $L \subseteq \{0,1\}^*$ if M computes f_L, where $f_L(x) = 1$ if and only if $x \in L$.

Complexity Class P

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.

- Definition: A language L is in $DTIME(T(n))$ if there's a TM that decides L in time $O(T(n))$.

- Defintion: Class $P = \cup_{c>0} DTIME(n^c)$.

Complexity Class P

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.

- Definition: A language L is in $\mathrm{DTIME}(T(n))$ if there's a TM that decides L in time $O(T(n))$.

- Defintion: Class $P = \cup_{c \geq 0} \mathrm{DTIME}(n^c)$.

Deterministic polynomial-time

Complexity Class P : Examples

- Cycle detection *(DFS)*
  - Check if a given graph has a cycle.

Complexity Class P : Examples

- Cycle detection

- Solvabililty of a system of linear equations *(Gaussian elimination)*
  - Given a system of linear equations over   check if there exists a common solution to all the linear equations.

Complexity Class P : Examples

- Cycle detection
- Solvabililty of a system of linear equations
- Perfect matching *(Edmonds 1965)*
  - Check if a given graph has a perfect matching

Complexity Class P : Examples

- Cycle detection
- Solvabililty of a system of linear equations
- Perfect matching
- Primality testing *(AKS test 2002)*
  - Check if a number is prime