CS640: Computational Complexity Theory

- Computation, we all understand these days, is a process running on a computer-like device.

- while, complexity refers to the resources that this process requires, eg.

  how much time?
  — space?
  — randomness?

- The theory of computation is motivated by the qn.: Does every problem has a solution?

- The theory of complexity by: What is the "cheapest" solution?

- Eg. We know how to <u>add</u> two numbers a & b, given in bits. How cheaply?

  If we do it carefully then in time $\approx (\log_2 a + \log_2 b)$.

  Any faster?

- On the other extreme: We have uncomputable problems.

  Can you write a computer program that takes a C-program M as input & <u>decides</u> whether M, on execution, halts or not?

- Think of the program M:

```
for (n=2,4,6,8,...) {
    flag=0;
    for( primes p ≤ n)
        if(n-p is a prime)
        then flag=1;
    if(flag==0) then HALT;
}
```

- This program HALTs at a counter-eg. of the Goldbach Conjecture!

- So, in this way we can encode almost any open math. qn. as a C-program!!

- This seems incredibly hard....

- This problem is, naturally, called the <u>Halting problem</u>.

  Known to be <u>uncomputable</u>!

- To prove this one needs a rigorous definition of <u>computation</u>.

- This is done on a mathematical model of a machine - <u>Turing machine</u>.

  Defined by Alan Turing in 1936.

- This course will deal with various notions of complexity of problems that can be solved on a Turing machine.

- Let us quickly see an outline of the course:

– (1) <u>Halting problem</u> (& the like).
Hilbert's 10th?

(2) <u>SATisfiability</u>:

Given a boolean formula ϕ in $\vee, \wedge, \sim$. e.g. $\phi = (x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$.
Decide whether ϕ is <u>satisfiable</u>, i.e. whether there is a way to set $x_i \in \{\text{True}, \text{False}\}$ such that $\phi = \text{True}$.

$P \neq NP$?

(3) <u>QBF</u> (quantified boolean formula):

Given a formula with quantifiers, $\exists, \forall$. e.g. $\phi = \exists x_1 \exists x_2 \forall x_3 \; (x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$.
Decide whether ϕ is true.

$NP \neq PSPACE$?

(4) <u>Formula optimization</u>:

Given a formula ϕ. Decide whether there is a smaller formula $\psi \equiv \phi$.

$NP \neq PH$?

(5) Identity testing:

Given a polynomial $f(x_1, .., x_n)$. Decide whether $f = 0$.

$P = BPP$?

(6) Graph reachability:

Given a huge graph G, two vertices s & t, and very little computing space. Decide whether there is a path $s \rightsquigarrow t$.

$L = RL$?

(7) Permanent:

Given a matrix $A \in \mathbb{Q}^{n \times n}$ compute its $\mathrm{per}(A) := \sum_{\pi \in S_n} \prod_{i=1}^{n} A_{i,\pi(i)}$.

$FP \neq \#P$?

(8) Graph isomorphism:

Given two graphs G_1, G_2. Decide whether $G_1 \cong G_2$.

$P \neq IP$?

[In 2016, Babai gave a breakthrough: GI has a exp(poly(log n))-time algorithm.]

- <u>Course webpage</u>:

  Go to www.cse/users/nitin.

- <u>Grading</u>:

  5% - Class participation

  25% - Assignments (no copying!)

  30% - Mid-sem

  40% - End-sem

  Bonus marks - Talk (advanced topic)

Status: $\mathbb{L} \stackrel{?}{=} RL \neq? \; P \neq? \; NP \neq? \; PH \neq? \; P^{\#P}$

$P \stackrel{?}{=} BPP$

$P \neq EXP \neq? \; Pspace = IP$

$IP \neq? \; P^{\#P}$