Thanks to the success of the Kinect, researchers all over the world believe they see the future of HCI and it is voice, speech and gestures. They are already developing a “gesture-controlled system for monitoring air-traffic”…

www.technologyreview.com/computing/37201
Computer Science … A UCB view

- **CS research areas:**
  - Artificial Intelligence
  - Biosystems & Computational Biology
  - Database Management Systems
  - Graphics
  - Human-Computer Interaction
  - Networking
  - Programming Systems
  - Scientific Computing
  - Security
  - Systems
  - **Theory**
    - Complexity theory
  - ...

UC Berkeley CS10 “The Beauty and Joy of Computing” : Limits of Computability (2)
Let’s revisit algorithm complexity

- Problems that...
  - are tractable with efficient solutions in reasonable time
  - are intractable
  - are solvable approximately, not optimally
  - have no known efficient solution
  - are not solvable
Recall our algorithm complexity lecture, we’ve got several common orders of growth:
- Constant
- Logarithmic
- Linear
- Quadratic
- Cubic
- Exponential

Order of growth is polynomial in the size of the problem.

E.g.,
- Searching for an item in a collection
- Sorting a collection
- Finding if two numbers in a collection are same

These problems are called being “in P” (for polynomial).
Intractable problems

- Problems that can be solved, but not solved fast enough

- This includes exponential problems
  - E.g., \( f(n) = 2^n \)
  - as in the image to the right

- This also includes poly-time algorithm with a huge exponent
  - E.g, \( f(n) = n^{10} \)

- Only solve for small \( n \)

Imagine a program that calculated something important at each of the bottom circles. This tree has height \( n \), but there are \( 2^n \) bottom circles!
Peer Instruction

What's the most you can put in your knapsack?

a) $4
b) $7
c) $8
d) $15
e) $19

Knapsack Problem
You have a backpack with a weight limit (here 15kg), which boxes (with weights and values) should be taken to maximize value?
Solvable approximately, not optimally in reasonable time

- A problem might have an optimal solution that cannot be solved in reasonable time
- BUT if you don’t need to know the perfect solution, there might exist algorithms which could give pretty good answers in reasonable time

Knapsack Problem
You have a backpack with a weight limit (here 15kg), which boxes (with weights and values) should be taken to maximize value?
Have no known efficient solution

- Solving one of them would solve an entire class of them!
  - We can transform one to another, i.e., reduce
  - A problem P is “hard” for a class C if every element of C can be “reduced” to P
- If you’re “in NP” and “NP-hard”, then you’re “NP-complete”

Subset Sum Problem
Are there a handful of these numbers (at least 1) that add together to get 0?

- If you guess an answer, can I verify it in polynomial time?
  - Called being “in NP”
  - Non-deterministic (the “guess” part) Polynomial

UC Berkeley CS10 “The Beauty and Joy of Computing” : Limits of Computability (8)
The fundamental question. Is $P = \text{NP}$?

- This is THE major unsolved problem in Computer Science!
  - One of 7 “millennium prizes” w/ a $1M reward

- All it would take is solving ONE problem in the NP-complete set in polynomial time!!
  - Huge ramifications for cryptography, others

If $P \neq \text{NP}$, then

- **Other NP-Complete**
  - Traveling salesman who needs most efficient route to visit all cities and return home
  - 3SAT
Problems NOT solvable

- Decision problems answer YES or NO for an infinite # of inputs
  - E.g., is N prime?
  - E.g., is sentence S grammatically correct?

- An algorithm is a solution if it correctly answers YES/NO in a finite amount of time

- A problem is decidable if it has a solution

Alan Turing
He asked:
"Are all problems decidable?"
(people used to believe this was true)
Turing proved it wasn’t for CS!
Turing’s proof: The Halting Problem

- Given a program and some input, will that program eventually stop? (or will it loop)
- Assume we could write it, then let’s prove a contradiction
  - 1. write Stops on Self?
  - 2. Write Weird
  - 3. Call Weird on itself
Conclusion

- Complexity theory is an important part of CS.
- If given a hard problem, rather than try to solve it yourself, see if others have tried similar problems.
- If you don’t need an exact solution, many approximation algorithms help.
- Some not computable!