More History

- 1930’s
  - What is (is not) computable
- 1960/70’s
  - What is (is not) feasibly computable
- Goal
  - A (largely) technology independent theory of time required by algorithms
- Key modeling assumptions/approximations
  - Asymptotic (Big-O), worst case is revealing
  - Polynomial, exponential time – qualitatively different

Another view of Poly vs Exp

Next year’s computer will be 2x faster. If I can solve problem of size $N_0$ today, how large a problem can I solve in the same time next year?

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Increase</th>
<th>Example</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$n_0 \rightarrow 2n_0$</td>
<td>$10^{12}$</td>
<td>$2 \times 10^{12}$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>$n_0 \rightarrow \sqrt{2} n_0$</td>
<td>$10^6$</td>
<td>$1.4 \times 10^6$</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>$n_0 \rightarrow 3\sqrt{2} n_0$</td>
<td>$10^4$</td>
<td>$1.25 \times 10^4$</td>
</tr>
<tr>
<td>$2^{n/10}$</td>
<td>$n_0 \rightarrow n_0 + 10$</td>
<td>400</td>
<td>410</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$n_0 \rightarrow n_0 + 1$</td>
<td>40</td>
<td>41</td>
</tr>
</tbody>
</table>

Polynomial versus exponential

- We’ll say any algorithm whose run-time is
  - Polynomial is good
  - Bigger than polynomial is bad
- Note – of course there are exceptions:
  - $n^{100}$ is bigger than $(1.001)^n$ for most practical values of $n$ but usually such run-times don’t show up
  - There are algorithms that have run-times like $O(2^{n/2})$ and these may be useful for small input sizes, but they’re not too common either

Some Convenient Technicalities

- ”Problem” – the general case
  - Ex: The Clique Problem: Given a graph $G$ and an integer $k$, does $G$ contain a $k$-clique?
- ”Problem Instance” – the specific cases
  - Ex: Does $\square \bigcirc$ contain a 4-clique? (no)
  - Ex: Does $\square \bigcirc$ contain a 3-clique? (yes)
- Decision Problems – Just Yes/No answers
- Problems as Sets of ”Yes” Instances
  - Ex: CLIQUE = { $(G,k) \mid G$ contains a $k$-clique }
Decision problems
Computational complexity usually analyzed using decision problems
- answer is just 1 or 0 (yes or no).

Why?
- much simpler to deal with
- deciding whether \( G \) has a k-clique, is certainly no harder than finding a k-clique in \( G \), so a lower bound on deciding is also a lower bound on finding
- Less important, but if you have a good decider, you can often use it to get a good finder. (Ex.: does \( G \) still have a k-clique after I remove this vertex?)

Computational Complexity
Classify problems according to the amount of computational resources used by the best algorithms that solve them

Recall:
- worst-case running time of an algorithm:
  - max # steps algorithm takes on any input of size \( n \)
Define:
- \( \text{TIME}(f(n)) \) to be the set of all decision problems solved by algorithms having worst-case running time \( O(f(n)) \)

Polynomial time
Define \( P \) (polynomial-time) to be
- the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

- \( P = \bigcup_{k \geq 0} \text{TIME}(n^k) \)

Beyond \( P \)?
There are many natural, practical problems for which we don’t know any polynomial-time algorithms
- e.g. decisionTSP:
  - Given a weighted graph \( G \) and an integer \( k \), does there exist a tour that visits all vertices in \( G \) having total weight at most \( k \)?

Solving TSP given a solution to decisionTSP
Use binary search and several calls to decisionTSP to figure out what the exact total weight of the shortest tour is.
- Upper and lower bounds to start are \( n \) times largest and smallest weights of edges, respectively
- Call \( W \) the weight of the shortest tour.
Now figure out which edges are in the tour
- For each edge \( e \) in the graph in turn, remove \( e \) and see if there is a tour of weight at most \( W \) using decisionTSP
  - if not then \( e \) must be in the tour so put it back

More examples
Independent-Set:
- Given a graph \( G=(V,E) \) and an integer \( k \), is there a subset \( U \) of \( V \) with \( |U| \geq k \) such that no two vertices in \( U \) are joined by an edge.

Clique:
- Given a graph \( G=(V,E) \) and an integer \( k \), is there a subset \( U \) of \( V \) with \( |U| \geq k \) such that every pair of vertices in \( U \) is joined by an edge.
### Satisfiability

- Boolean variables $x_1, \ldots, x_n$
  - taking values in $\{0, 1\}$: 0=false, 1=true
- Literals
  - $x_i$ or $\neg x_i$ for $i=1, \ldots, n$
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses

### CNF formula example

- $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12}) \land (x_2 \lor \neg x_4 \lor x_7 \lor x_3)$

### If there is some assignment of 0’s and 1’s to the variables that makes it true then we say the formula is satisfiable

- the one above is, the following isn’t
- $x_1 \land (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \neg x_3$

### Satisfiability: Given a CNF formula $F$, is it satisfiable?

### More History – As of 1970

- Many of the above problems had been studied for decades
- All had real, practical applications
- *None* had poly time algorithms; exponential was best known
- But, it turns out they all have a very deep similarity under the skin

### Common property of these problems

- There is a special piece of information, a short hint or proof, that allows you to efficiently verify (in polynomial-time) that the YES answer is correct. This hint might be very hard to find
- e.g.
  - DecisionTSP: the tour itself,
  - Independent-Set, Clique: the set $U$
  - Satisfiability: an assignment that makes $F$ true.

### The complexity class $\textbf{NP}$

- $\textbf{NP}$ consists of all decision problems where
  - You can verify the YES answers efficiently (in polynomial time) given a short (polynomial-size) hint
  - And
  - No hint can fool your polynomial time verifier into saying YES for a NO instance

### More Precise Definition of $\textbf{NP}$

- A decision problem is in $\textbf{NP}$ iff there is a polynomial time procedure $v(\ldots)$, and an integer $k$ such that
  - for every YES problem instance $x$ there is a hint $h$ with $|h| \leq |x|^k$ such that $v(x,h) = \text{YES}$ and
  - for every NO problem instance $x$ there is *no* hint $h$ with $|h| \leq |x|^k$ such that $v(x,h) = \text{YES}$
Example: CLIQUE is in NP

procedure v(x,h)
  if
    x is a well-formed representation of a graph G = (V, E) and an integer k,
    and
    h is a well-formed representation of a k vertex subset U of V,
    and
    U is a clique in G,
    then output "YES"
  else output "I'm unconvincited"

Is it correct?

For every x = (G,k) such that G contains a k-clique, there is a hint h that will cause v(x,h) to say YES, namely h = a list of the vertices in such a k-clique
and
No hint can fool v into saying yes if either x isn’t well-formed (the uninteresting case) or if x = (G,k) but G does not have any cliques of size k (the interesting case)

Keys to showing that a problem is in NP

What’s the output? (must be YES/NO)
What’s the input? Which are YES?
For every given YES input, is there a hint that would help?
  OK if some inputs need no hint
For any given NO input, is there a hint that would trick you?

Solving NP problems without hints

The only obvious algorithm for most of these problems is brute force:
  try all possible hints and check each one to see if it works.
  Exponential time:
    2^n truth assignments for n variables
    n! possible TSP tours of n vertices
    \(^k\) possible k element subsets of n vertices
    etc.

What We Know

Nobody knows if all problems in NP can be done in polynomial time, i.e. does P=NP?
  one of the most important open questions in all of science.
  huge practical implications
Every problem in P is in NP
  one doesn’t even need a hint for problems in P so just ignore any hint you are given
Every problem in NP is in exponential time

P and NP
P vs NP

Theory
- P = NP?
- Open Problem!
- I bet against it

Practice
- Many interesting, useful, natural, well-studied problems known to be NP-complete
- With rare exceptions, no one routinely succeeds in finding exact solutions to large, arbitrary instances

More Connections

Some Examples in NP
- Satisfiability
- Independent-Set
- Clique
- Vertex Cover
- All hard to solve; hints seem to help on all
- Very surprising fact:
  - Fast solution to any gives fast solution to all!

NP-hardness & NP-completeness

Some problems in NP seem hard
- people have looked for efficient algorithms for them for hundreds of years without success

However
- nobody knows how to prove that they are really hard to solve, i.e. \( P \neq NP \)

NP-hardness & NP-completeness

Alternative approach
- show that they are at least as hard as any problem in NP

Rough definition:
- A problem is NP-hard iff it is at least as hard as any problem in NP
- A problem is NP-complete iff it is both
  - NP-hard
  - in NP

P and NP

Polynomial Time Reduction

\( L \leq_p R \) if there is a poly time algorithm for \( L \) assuming a poly time subroutine for \( R \)

Thus, fast alg for \( R \) implies fast alg for \( L \)
- If you can prove there is no fast alg for \( L \), then that proves there is no fast alg for \( R \)
What to do?  Hopeless?

- Heuristics: perhaps there’s an alg that’s:
  - usually fast, and/or
  - usually succeeds
- Approximation Algorithms: Would you settle for an answer within 1% of optimal? 10% ? 10x ?

Is NP as bad as it gets?

- NO! NP-complete problems are frequently encountered, but there’s worse:
  - Some problems provably require exponential time.
    - Ex: Does P halt on x in $2^n$ steps?
    - Some require $2^n$, $2^{2^n}$, $2^{2^{2^n}}$... steps
  - And of course, some are just plain uncomputable

Summary

- Big-O – good
- P – good
- Exp – bad
- Hints help? NP
- NP-hard, NP-complete – bad (I bet)