Lecture 16

Complexity Classes

Defn:

TIME(T(n)) = the set of languages decidable by single-tape TMs in time O(T(n))

E.g. $\{ a^n b^n \mid n \geq 0 \} \in \text{TIME}(n \log n)$

A Church-Turing thesis for "time"?

Church-Turing thesis: all "reasonable" models of computation are equivalent – i.e. all give the same set of decidable problems

"Extended" Church Turing thesis: All "reasonable" models of computation are equivalent *up to a polynomial difference in time complexity*

E.g. from above, this is true of deterministic singe- and multi-tape TMs and RAMs

More on what "reasonable" means later…

The class P

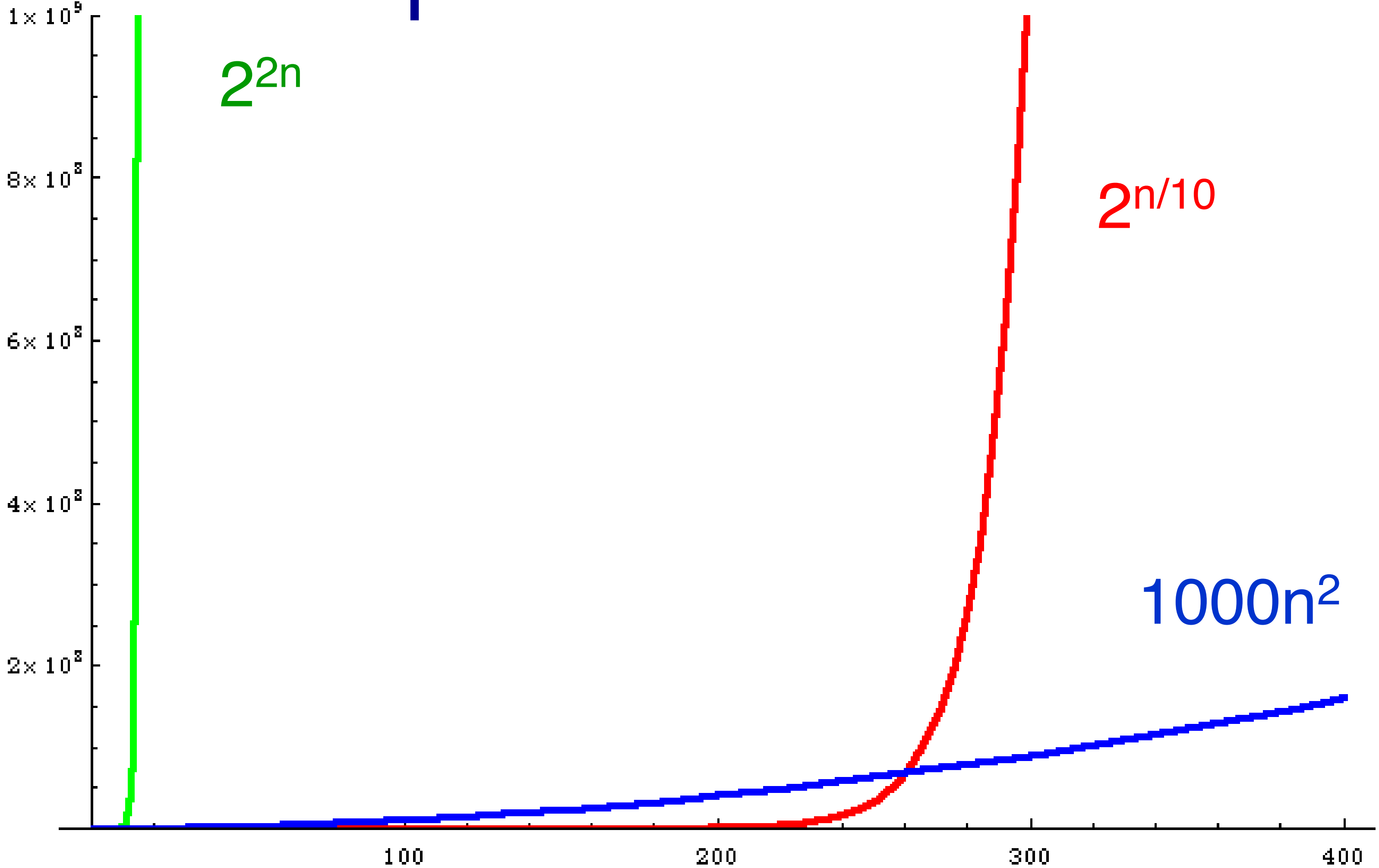
Definition:

$P = \bigcup_{k\geq 1} \mathrm{TIME}(n^k)$

I.e., the set of (decision) problems solvable by computers in *polynomial time*. I.e., $L \in P$ iff there is an algorithm deciding L in time $T(n) = O(n^k)$ for some fixed k (i.e., k is independent of the input).

Examples: sorting, shortest path, MST, connectivity, …

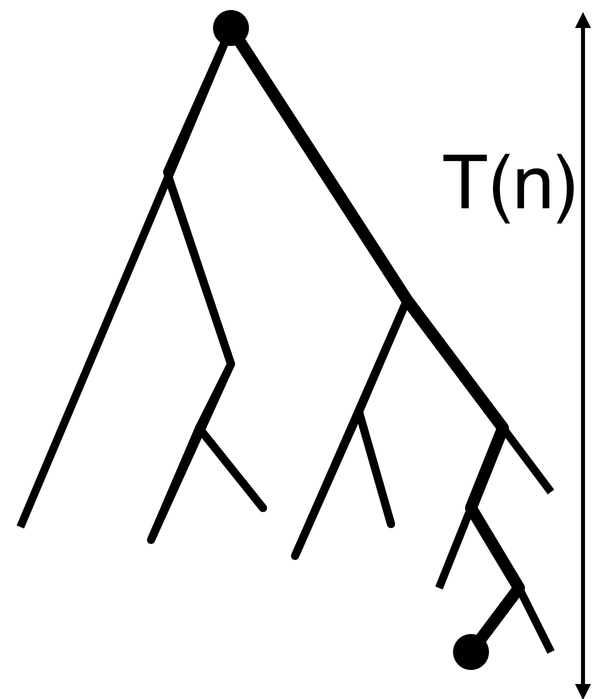
Polynomial vs Exponential Growth



Nondeterministic Time

Given a nondeterministic TM M that always halts, its run time T(n) is the length of the longest computation path (accepting or rejecting) on any input of length n.

(In fact, the theory doesn't change much if you make it "shortest accepting path", but that's just a detail.)



The class NP

Definition:

$NP = \bigcup_{k \geq 1}$ Nondeterministic-TIME(n^k)

I.e., the set of (decision) problems solvable by computers in *Nondeterministic* polynomial time. I.e., $L \in NP$ iff there is a nondeterministic algorithm deciding L in time $T(n) = O(n^k)$ for some fixed k (i.e., k is independent of the input).

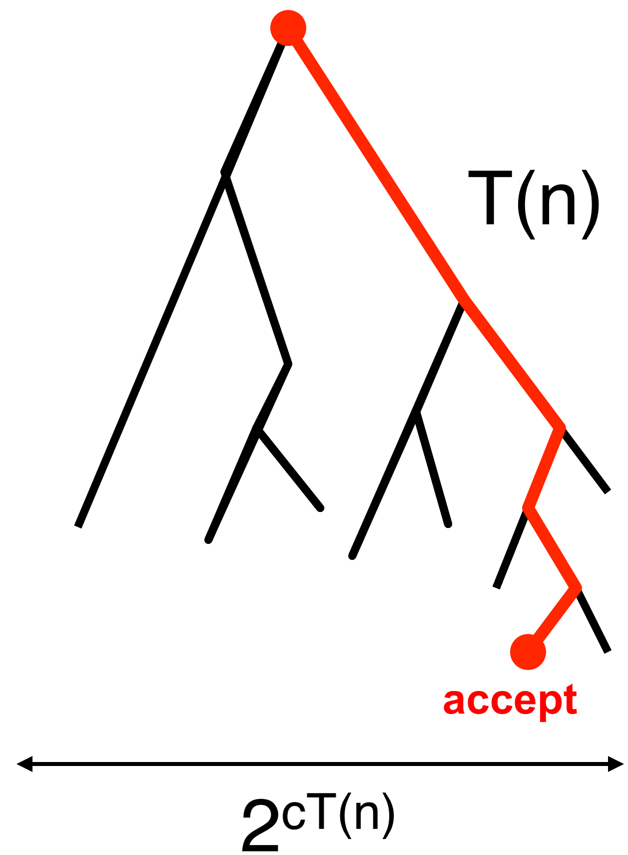
Examples: sorting, shortest path, …, *and more!*

NTIME(T) $\subseteq$ DTIME($2^{O(T)}$)

Theorem: Every problem solvable in nondeterministic time T(n) can be solved *deterministically* in time exponential in T(n)

Proof:

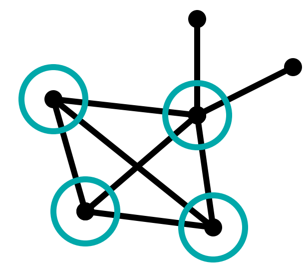
As before, do breadth first simulation. (Depth-first works too.)



The Clique Problem

Given: a graph G=(V,E) and an integer k

Question: is there a subset U of V with
|U| ≥ k such that every pair of vertices in U is joined by an edge.

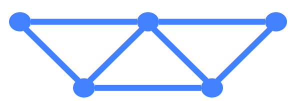


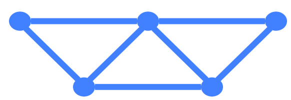
Some Convenient Technicalities

"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

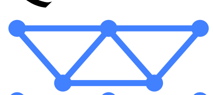
Ex: Does  contain a 4-clique? (no)

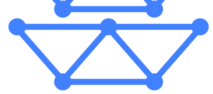
Ex: Does  contain a 3-clique? (yes)

Decision Problems – Just Yes/No answer

Problems as Sets of "Yes" Instances

Ex: CLIQUE = { (G,k) | G contains a k-clique }

E.g., ( , 4) $\notin$ CLIQUE

E.g., ( , 3) $\in$ CLIQUE

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. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$

CNF formula ("conjunctive normal form")

a logical AND of a bunch of clauses

Satisfiability

CNF formula example

$(x_1 \vee \neg x_3 \vee x_7) \wedge (\neg x_1 \vee \neg x_4 \vee x_5 \vee \neg x_7)$

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

$x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$

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

Satisfiable?

$$(x \lor y \lor z) \land (\neg x \lor y \lor \neg z) \land$$
$$(x \lor \neg y \lor z) \land (\neg x \lor \neg y \lor z) \land$$
$$(\neg x \lor \neg y \lor \neg z) \land (x \lor y \lor z) \land$$
$$(x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

$$(x \lor y \lor z) \land (\neg x \lor y \lor \neg z) \land$$
$$(x \lor \neg y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land$$
$$(\neg x \lor \neg y \lor \neg z) \land (\neg x \lor y \lor z) \land$$
$$(x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

Common property of these problems: Discrete Exponential Search Loosely–find a needle in a haystack

"Answer" is literally just yes/no, but there's always a somewhat more elaborate "solution" (aka "hint" or "certificate") that *transparently*‡ justifies each "yes" instance (and only those) – but it's *buried in an exponentially large search space of potential solutions.*

‡*Transparently* = verifiable in polynomial time