Computational Complexity

Decision Problem

- Running a decision problem on a TM.
  - Once encoded, the encoded instance is provided as input to a TM.
  - The TM must then
    - Determine if the input is a valid encoding
    - Run and halt:
      - In accept state if the answer for the input is yes
      - In reject state if the answer for the input is no
  - If such a TM exists for a given decision problem, the problem is decidable or solvable. Otherwise the problem is called undecidable or unsolvable.

What Makes a Good Algorithm?

- Suppose we know that a decision problem is decidable. How do we know that there is a good algorithm that solves the problem?
  - Consider the complexity of a TM that can solve the problem.

Complexity

- Complexity refers to the rate at which the storage or time required to solve the problem grows as a function of the length of the input to the algorithm
  - T(n) = time complexity (amount of time an algorithm will take based on input)
  - S(n) = space complexity (amount of space an algorithm will take based on input)
- We'll focus on time complexity

Complexity

- Definition 7.1
  - Let M be a deterministic TM that halts on all inputs. The time complexity of M is the function T: N → N where T(n) is the maximum number of steps that M uses on any input of length n.
  - n is used to represent the length (encoding) of the input string
  - T(n) can be thought of as the number of TM "moves" needed to accept or reject

Asymptotic Analysis

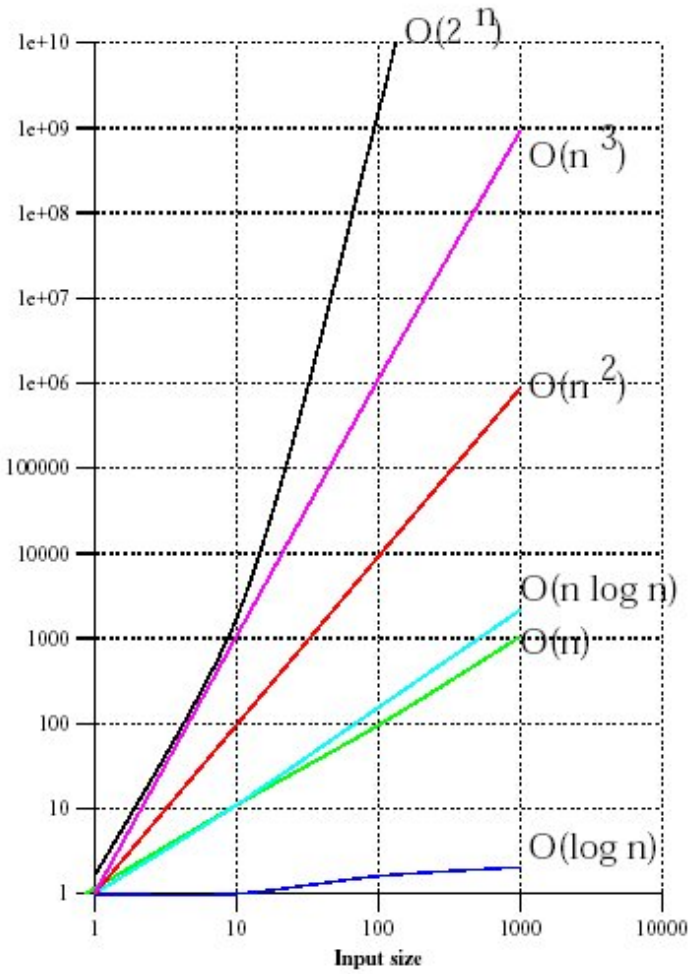
- Based on the idea that as the input to an algorithm gets large:
  - The complexity will become proportional to a known function.
  - Only worry about the highest order term
  - Notation:
    - O (Big-O) – upper bound on the complexity
    - Θ (Big-Theta) – tight bounds on the complexity

Asymptotic Analysis

- Big-O (order of)
  - $T(n) \in O(f(n))$ if and only if there are constants:
    - $c > 0$ and $n_0 \geq 0$ such that
    - $T(n) \leq cf(n)$ for all $n \geq n_0$
  - What this means
    - The algorithm may have some lower order "start-up" costs
    - Eventually, when the size of the input gets large enough ($n \geq n_0$), the runtime will be bounded above by a scaled ($c > 0$) function f(n).
      - Using c allows us to ignore constant factors when discussing asymptotic complexity

Algorithm Efficiencies

Axes are log scale, so all polynomial terms look linear here



What Complexity is Acceptable?

- So what should be the cutoff between a "good" algorithm and a "bad" algorithm?
  - In the 1960s, it was proposed:
    - A "good" algorithm is one whose running time is a polynomial function of the size of the input
    - Other algorithms are "bad"
  - This definition was adopted:
    - A problem is called tractable if there exists a "good" (polynomial time) algorithm that solves it.
    - A problem is called intractable otherwise.

Is this a Valid Cutoff?

Assuming 1 million computations per second

Time complexity function	Size n					
	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

The Class P

- The class P contains all decision problems that are decidable by an algorithm that runs **in polynomial time in the size of the input**.
  - Does this define a class of languages?
  - Yes:
    - Each language is a specific decision problem whose encodings of "yes instances" (the language) is decided by a deterministic TM, M, where M decides that particular language in polynomial time.
  - Recall: These languages are decidable

The Class P

- The class P is robust
  - It is the same class for all reasonable deterministic computational models
    - All reasonable deterministic computational models are polynomially equivalent
    - So if an algorithm runs in polynomial time using one given reasonable deterministic computational model, it also runs in polynomial time using other reasonable computational models
    - This means we don't have to worry about the specific computational model in use when showing that a language is contained in P

The Class P

- To show that a decision problem is in P, we have to consider two aspects:
  - The algorithm runs in polynomial time in the size of the input string
  - A “reasonable encoding” is used to convert objects into strings
    - Polynomial-time (with respect to the size of the original input) to convert to a string

The Class P

- Thus to show that a problem is in P, it suffices to analyze a high-level description
  - Describe a TM algorithm
    - Describe/assume a reasonable encoding that takes polynomial time
    - Show that the number of stages is polynomial in the size of the input
    - Show that the cost of running each stage is polynomial in the size of the input
- In doing so, we are discarding polynomial differences in algorithms.
  - Not that these are unimportant!

CFLs in P

- Sipser, pg. 290-291
  - Every context-free language is a member of P
  - Proof uses dynamic programming technique (covered in Analysis of Algorithms class)

Polynomially Verifiable

- A verifier for a language A is an algorithm V, where A = { w | V accepts <w,c> for some string c}.
- A polynomial time verifier runs in polynomial time in the length of w.
- The string c is called the certificate, or proof, that w is in A.

NP

- NP is the class of languages that have polynomial-time verifiers

- So, it may be that the algorithm can't run in polynomial time, but at least a specific certificate (or proof) that shows that the string is in the language can be checked in polynomial time
  - (that's all polynomial verifiability means – given a certificate, you can use that to confirm in polynomial time that the string should be accepted)

Another View of the Class NP

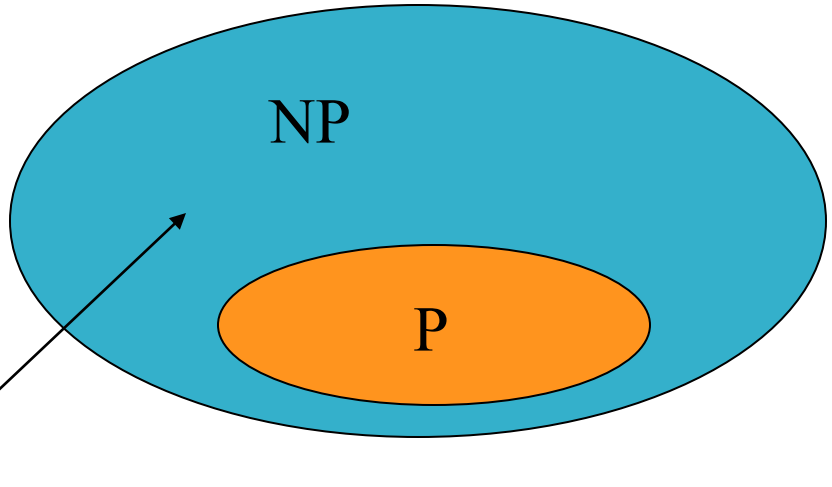
- Theorem 7.20:
  - A language is in NP if and only if it is decided by some non-deterministic polynomial time Turing Machine

Remember that P requires the language to be decidable in polynomial time by a deterministic TM. So NP allows extra flexibility. There may be an exponential number of paths, but as long as they all compute in polynomial time, it's in NP.

The Class NP

- Clearly P is a subset of NP
- Does P=NP?

(a deterministic TM is just a special case of a NDTM that only has one branch of computation, so anything in P is in NP)



Is there something in here?

After 40+ years of research…nobody knows for certain