GBL rev l 4/20

**CSCE686 Lecture 1: *Computational Complexity Theory*** *(review, Talbi p9)*

A fundamental area of theoretical computer science is complexity theory, the analysis of the resources needed to solve computational problems. CSCE586   
(or a version of CSCE532) presented some computational models, such as Turing machines, automata languages, NP-Complete problems , and resource measures such as space and time. A *complexity class* is the set of problems solvable in a particular model under particular resource constraints. Models must be simple enough to allow mathematical analysis yet general enough to be useful in a wide context of application. That is why determining and solving *complex* NP-Complete problems are important! <http://en.wikipedia.org/wiki/NP-complete>

* ***Computational complexity*** is defined in terms of the natural entities of time and space, and the term complexity is used to denote the time or space used in the computation. You have studied P-time, NP-Time, NP-C and HP-Hard problems. How about P-complete, co-NP-time, PSpace, and EXPTIME?

(see <http://en.wikipedia.org/wiki/Complexity> , and <http://en.wikipedia.org/wiki/Computational_complexity_theory> , and <https://en.wikipedia.org/wiki/Time_complexity> )

* Perhaps the hardest thing to prove about a computational model is that it may not solve a given problem within particular resource constraints (time, space). Such ``lower bounds'' are currently obtainable only when the model is very specialized or the constraints are very severe (see problem domain literature).
* Another complexity research study area is called [“descriptive complexity](http://www.cs.umass.edu/~immerman/descriptive_complexity.html)”. Rather than checking whether an input satisfies a property S, a more natural question might be, what is the complexity of expressing the property S? These two issues -- checking and expressing -- are closely related. It is startling how closely tied they are when the latter refers to expressing the property in first-order logic of finite and ordered structures (see literature for math).

**Note:** for your CSCE686 project you need to prove that your problem is of a specific complexity! (*Drives selection of solution algorithm*)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Decision problems vs optimization problems:*** *(review; Talbi, p11)*

* Any problem for which the answer is either zero or one is called a *decision problem*. An algorithm for a decision problem is termed a *decision algorithm*.
* Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an *optimization problem*. An optimization algorithm is used to solve an *optimization problem*.

***General Computational Problem Definitions*:** *(review; Talbi, p9)*

* **P**. [*P-time* problems](https://en.wikipedia.org/wiki/P_(complexity)) (P complexity) can be solved in polynomial time and a solution validated in polynomial time ("P" stands for polynomial.); i.e., the set of problems (languages) that can be solved by a deterministic Turing machine in polynomial time.   
  P-time languages can also be viewed as solved with a *uniform* family of Boolean circuits; i.e., a circuit for each member of the language. {Note: P-time problems formed the main material of CSCE486 and CSCE586 courses.}
* **NP**. *NP-time* stands for "nondeterministic polynomial time." [NP-*time* problems](https://en.wikipedia.org/wiki/NP_(complexity)) (NP complexity) is the set of all problems (languages) that can be solved by a non-deterministic Turing machine in polynomial time. A problem is in NP-time if you can quickly (in polynomial time) test (verify) whether a solution is correct (without worrying about how difficult it might be to find the solution). Some problems in NP-time are relatively easy: if only we could guess the right solution, we could then quickly test it. {What if *P = NP?*}

*Remember: NP does not stand for "non-polynomial", but for “nondeterministic polynomial time*!”

***\*\*\*\*\*\*\*\*\*\*\*\*\*\****

[***Problem Reduction***](https://en.wikipedia.org/wiki/Polynomial-time_reduction) ***(Reducibility): (Turing Reduction, Mapping:*** *Talbi, p13)*

* Reducing problem L to another problem H means describing an algorithm to solve problem L under the assumption that an algorithm for solving problem H already exists. This reduction or data structure mapping from L to H is by definition polynomial (accomplished by a deterministic Turing machine). “One-one reductions (Cook) and many-one reductions (Karp)”

***\*\*\*\*\*\*\*\*\*\*\*\*\*\****

[***P-Complete Definition***](https://en.wikipedia.org/wiki/P-complete)***:*** a decision problem is P-complete if it is in P and every problem in P can be reduced to it by an appropriate reduction (data structure mapping).

***[Classical P-Complete problems](https://en.wikipedia.org/wiki/P-complete)*[:](https://en.wikipedia.org/wiki/P-complete)**

* Horn satisfiability of Horn clauses; Boolean satisfiability problem
* Linear Programming model approximation solution *(will use later in CSCE686)*
* Graph depth first search (DFS) …

[***NC complexity***](https://en.wikipedia.org/wiki/NC_%28complexity%29) ***class:*** NC (Nick’s Class) is a subset of P because *polylogarithmic* *parallel* computations can be simulated by *polynomial-time* sequential computation.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\****

[***NP-Complete definition:***](https://en.wikipedia.org/wiki/NP-completeness)*(Talbi, p14)*

1. NP-C; The [complexity class](http://xlinux.nist.gov/dads/HTML/complexityClass.html) of [decision problems](http://xlinux.nist.gov/dads/HTML/decisionProblem.html) for which answers can be checked (verified) for correctness, given a [certificate](http://xlinux.nist.gov/dads/HTML/certificate.html), by an algorithm whose run time is [polynomial](http://xlinux.nist.gov/dads/HTML/polynomialtm.html) in the size of the input (that is, it is [NP](http://xlinux.nist.gov/dads/HTML/np.html)) and no other NP problem is more than a polynomial factor harder. Informally, a problem is NP-complete if answers can be verified quickly, and an algorithm to solve this problem can be used to solve all other NP problems.
2. A problem R is *NP-complete* if
   1. R is NP-Time, and
   2. There exists a known NP-complete problem that reduces to R in polynomial time.(example: SAT;Cook)

[***Classical NP-Complete problems*:**](https://en.wikipedia.org/wiki/List_of_NP-complete_problems)

*(Polynomial reducibility or mapping data structures to each other in* ***P****-Time by definition)*

* Satisfiability Problems (3-SAT, …) “Cook’s original problem”
* Assignment Problem (planning, weapon-target, placement, scheduling, …)
* Vehicle routing (mission planning, communication scheduling, …)
* [Set/Vertex Covering Problem](http://en.wikipedia.org/wiki/Set_covering_problem) (logic minimization, routing, scheduling, imaging)
* [Maximal Independent Sets](https://en.wikipedia.org/wiki/Maximal_independent_set) - Clique Problem (weighted, scheduling, imaging, ...)
* Graph Coloring Problem (planar, layout, scheduling, imaging...)
* Knapsack Problem (assignment, scheduling, imaging, packing, ...)
* Network flow problem domains (multicommodity, constraints, …)
* Traveling Salesperson Problem-decision (Hamiltonian-cycle, scheduling, imaging, routing)
* Partitioning Problems (3D-matching, coloring, …)
* [Secretary problem](https://en.wikipedia.org/wiki/Secretary_problem) (NPC?) *“Use of heuristics*”
* Sailor/Airman Assignment Problem (NPC?)
* Minesweeper Problem, Sudoku, … (How many NPC problems?)

*“Which of these are linear problem models (objective(s) and constraints)?”*

***\*\*\*\*\*\*\*\*\*\*\*\*\****

***NP-Hard definition:*** (<http://en.wikipedia.org/wiki/NP-hard> ) (Talbi, p14)   
**NP-Hard** ([non-deterministic polynomial-time](http://en.wikipedia.org/wiki/NP_(complexity)) hard), in [computational complexity theory](http://en.wikipedia.org/wiki/Computational_complexity_theory), is a class of problems that are, informally, "at least as hard as the hardest problems in NP". A problem H is NP-hard [if and only if](http://en.wikipedia.org/wiki/If_and_only_if) there is an [NP-complete](http://en.wikipedia.org/wiki/NP-complete) problem L that is [polynomial time Turing-reducible](http://en.wikipedia.org/wiki/Polynomial-time_Turing_reduction) to H and H is not in NP. *Note that NP-C problems maybe NP-Hard and in NP (see Venn diagram).”see TSP papers”*

***Classical NP-Hard problems (includes*** [***NP-Complete problems***](https://en.wikipedia.org/wiki/List_of_NP-complete_problems)***)***(see Venn diagram)

*-*NP-hard but not NP-complete problems (i.e., NP-hard but not in NP)

-- Tower of Hanoi (not in NP since solution of exponential length - verifiable in …)

-- Halting Problem?

-- [Matrix permanent](https://en.wikipedia.org/wiki/Permanent)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref: many textbooks; MIT OpenCourseWare – [Automata, Computability, and Complexity](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/index.htm)

 ***NP Review:***

[NP](https://en.wikipedia.org/wiki/NP_(complexity)) (NP-Time)  
 Solution verifiable in polynomial time

[NP-complete](http://en.wikipedia.org/wiki/NP-complete)  
 Hardest problems in NP. Exponential time to find solution.

[NP-hard](https://en.wikipedia.org/wiki/NP-hardness)   
 At least as hard as the hardest problems in NP. Such problems need not be in NP; indeed, they may not even be decision problems. What NP-Hard Problems are not even NP-Complete?

[NP-easy](http://en.wikipedia.org/wiki/NP-easy)  
 At most as hard as NPC, but not necessarily in NPC, since they may not be decision problems; (example: sorting of strings)

[NP-equivalent](http://en.wikipedia.org/wiki/NP-equivalent)  
 Exactly as difficult as the hardest problems in NP; NPC: but not necessarily in NP.

[NP-intermediate](https://en.wikipedia.org/wiki/NP-intermediate)  
 If P and NP are different, then there exist decision problems in the region of NP that fall between P and the NP-complete problems

***Additional Problem Classes*** (see different Venn diagram)

***Co-NP Class******Definition:*** The NP-class also has a counterpart, called **Co-NP**. A problem is in the class of co-NP problems if, given a problem statement there is no solution, that is the decision answer is no, there must exist a decision certificate to show this. For example, for the SAT problem to be in co-NP, we would need a “short” way to show that there is no solution to SAT; i.e. then Co-SAT in Co-NP. (Primes?) Co-NP Problems are essentially the opposite of NP. If an answer to a decision problem in co-NP is NO, then is can be checked in polynomial time. Co-NP may not be the same as the set NP. Note that P-time is in both NP-time and Co-NP-time.

[***PSPACE***](https://en.wikipedia.org/wiki/PSPACE) ***Class Definition.*** Set of all decision problems that can be solved by a Turing machine using a polynomial amount of space. Co-PSPACE = PSPACE

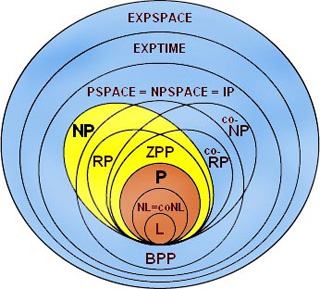
***PSPACE Complete*.** a decision problem is PSPACE-complete if it can be solved using an amount of memory that is polynomial in the input length (polynomial space) and if every other problem that can be solved in polynomial space can be transformed to it in polynomial time ([examples](https://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems))

***EXPTIME Class Definition:*** not all-inclusive class of problems solvable by deterministic Turing machines solvable in an exponent time bound; order (*2p(n)*), where *p(n)* is a polynomial in n.. Whereas P machines are bounded by polynomial time, the running time of an EXP machine may take exponential time. EXP contains nearly every other class that we regularly concern ourselves with, including [PSPACE](https://complexityzoo.uwaterloo.ca/Petting_Zoo#PSPACE) and [the polynomial hierarchy](https://complexityzoo.uwaterloo.ca/Petting_Zoo#PH). Of course, we can always construct classes of still harder problems, such as [NEXP](https://complexityzoo.uwaterloo.ca/Complexity_Zoo:N#nexp) and [EEXP](https://complexityzoo.uwaterloo.ca/Complexity_Zoo:E#eexp), but from a simplistic point of view, EXP is "big enough" to contain most problems we ever hope to solve computationally!

***EXPTIME Complete.*** A decision problem is EXPTIME-complete if it is in [EXPTIME](http://www.liquisearch.com/what_is_exptime), and every problem in EXPTIME has a polynomial-time many-one reduction to it

***EXPSPACE Class Definition.*** The set of all decision problems solvable by a deterministic Turing machine in order (2*p*(*n*)) space, where *p*(*n*) is a polynomial function of *n* [(examples)](https://en.wikipedia.org/wiki/EXPSPACE)

***EXPSPACE complete.*** A decision problem is EXPSPACE-complete if it is in EXPSPACE, and every problem in EXPSPACE has a [polynomial-time many-one reduction](https://en.wikipedia.org/wiki/Polynomial-time_many-one_reduction) to it.



**“Why are complexity classes useful?” They can provide an effective and efficient algorithm for a particular problem in the associated complexity class? See various complexity classes for associated algorithm discussions.**

***Some more complexity classes:*** *(*see additional Venn diagram - UMASS*)*

***NC Problem Class Definition:*** the class **NC** (for "Nick's Class") is the set of [decision problems](http://en.wikipedia.org/wiki/Decision_problem) decidable in [polylogarithmic time](http://en.wikipedia.org/wiki/Polylogarithmic_time) on a [parallel computer](http://en.wikipedia.org/wiki/Parallel_computing) with a polynomial number of processors. In other words, a problem is in **NC** if there exist constants *c* and *k* such that it can be solved in time [*O*](http://en.wikipedia.org/wiki/Big_O_notation)(log*c* *n*) using [*O*](http://en.wikipedia.org/wiki/Big_O_notation)(*nk*) parallel processors. [Stephen Cook](http://en.wikipedia.org/wiki/Stephen_Cook) coined the name "Nick's class" after [Nick Pippenger](http://en.wikipedia.org/wiki/Nick_Pippenger), who had done extensive research on Boolean circuits with polylogarithmic depth and polynomial size. (Depth/length of circuit and size/gates of circuit)

***NL Definition.*** [**NL**](https://en.wikipedia.org/wiki/NL_(complexity)) consists of [decision problems](https://en.wikipedia.org/wiki/Decision_problem) that can be solved by a nondeterministic Turing machine with a read-only input tape and a separate read-write tape whose size is limited to be proportional to the logarithm of the input length. Similarly, **L** consists of the languages that can be solved by a deterministic Turing machine with the same assumptions about tape length. Because there are only a polynomial number of distinct configurations of these machines, both **L** and **NL** are subsets of the class [**P**](https://en.wikipedia.org/wiki/P_(complexity)) of deterministic polynomial-time decision problems

***AC Problem Class Definition:*** In [circuit complexity](http://en.wikipedia.org/wiki/Circuit_complexity), **AC** is a [complexity class](http://en.wikipedia.org/wiki/Complexity_class) hierarchy. Each class, **ACi**, consists of the [languages](http://en.wikipedia.org/wiki/Formal_language) recognized by [Boolean circuits](http://en.wikipedia.org/wiki/Boolean_circuit) with depth *O*(log*in*) and a [polynomial number](http://en.wikipedia.org/wiki/Polynomial) of [unlimited-fanin](http://en.wikipedia.org/wiki/Fanin) [AND](http://en.wikipedia.org/wiki/AND_gate) and [OR gates](http://en.wikipedia.org/wiki/OR_gate). The name "AC" was chosen by analogy to [NC](http://en.wikipedia.org/wiki/NC_(complexity)), with the "A" in the name standing for "alternating" and referring both to the alternation between the AND and OR gates in the circuits and to [alternating Turing machines](http://en.wikipedia.org/wiki/Alternating_Turing_machine).The smallest AC class is [AC0](http://en.wikipedia.org/wiki/AC0), consisting of constant-depth unlimited-fanin circuits.

***FO Problem Class Definition:*** the class of structures which can be recognised by formula of [first-order logic](http://en.wikipedia.org/wiki/First-order_logic) (FO). It is the foundation of the field of [descriptive complexity](http://en.wikipedia.org/wiki/Descriptive_complexity) and is equal to the complexity class [AC0](http://en.wikipedia.org/wiki/AC0) FO-regular. Various extensions of FO, formed by the addition of certain operators, give rise to other well-known complexity classes, allowing the complexity of some problems to be proven without having to go to the [algorithmic](http://en.wikipedia.org/wiki/Algorithmic) level.

***PP Complexity Definition:*** the class of [decision problems](https://en.wikipedia.org/wiki/Decision_problem) solvable by a [probabilistic Turing machine](https://en.wikipedia.org/wiki/Probabilistic_Turing_machine) in [polynomial time](https://en.wikipedia.org/wiki/Polynomial_time), with an error probability of less than 1/2 for all instances. The abbreviation **PP** refers to probabilistic polynomial time.

***BPP Complexity Definition:*** the class of problems (bounded error, probabilistic), polynomial time) that are solvable efficiently by randomized algorithms. Formally, BPP is defined as the class of languages which can be solved in polynomial time by a machine with access to a fair coin, with the constraint that the algorithm's error rate is bounded. The choices of constants in the definition are arbitrary, as we can amplify the accuracy of a BPP algorithm by repeating it several times and taking a majority vote, provided that the error rate is bounded by *some* constants; a subset of **PP.**

***ZPP Complexity Definition:*** [**ZPP**](https://en.wikipedia.org/wiki/ZPP_(complexity)) can be defined as the class of problems for which probabilistic Turing Machine exists with these properties: (example: [Las Vegas algorithm](http://en.wikipedia.org/wiki/Las_Vegas_algorithm)).

* It always runs in polynomial time.
* It returns an answer YES, NO or DO NOT KNOW.
* The answer is always either DO NOT KNOW or the correct answer.
* It returns DO NOT KNOW with probability at most 1/2 (and the correct answer otherwise).

***RP Complexity Definition:* randomized polynomial time** (**RP**) is the class of problems for which a [probabilistic Turing machine](https://en.wikipedia.org/wiki/Probabilistic_Turing_machine) exists with these properties:

* It always runs in polynomial time in the input size
* If the correct answer is NO, it always returns NO
* If the correct answer is YES, then it returns YES with probability at least 1/2 (otherwise, it returns NO).

***TC Complexity Definition:*** [**TC**](https://en.wikipedia.org/wiki/TC_(complexity)) is a complexity class of decision problems that can be recognized by threshold circuits, which are Boolean circuits with AND, OR, and [Majority gates](https://en.wikipedia.org/wiki/Majority_gate). For each fixed *i*, the complexity class **TCi** consists of all languages that can be recognized by a family of threshold circuits of logi depth, polynomial size, and unbounded fanin.

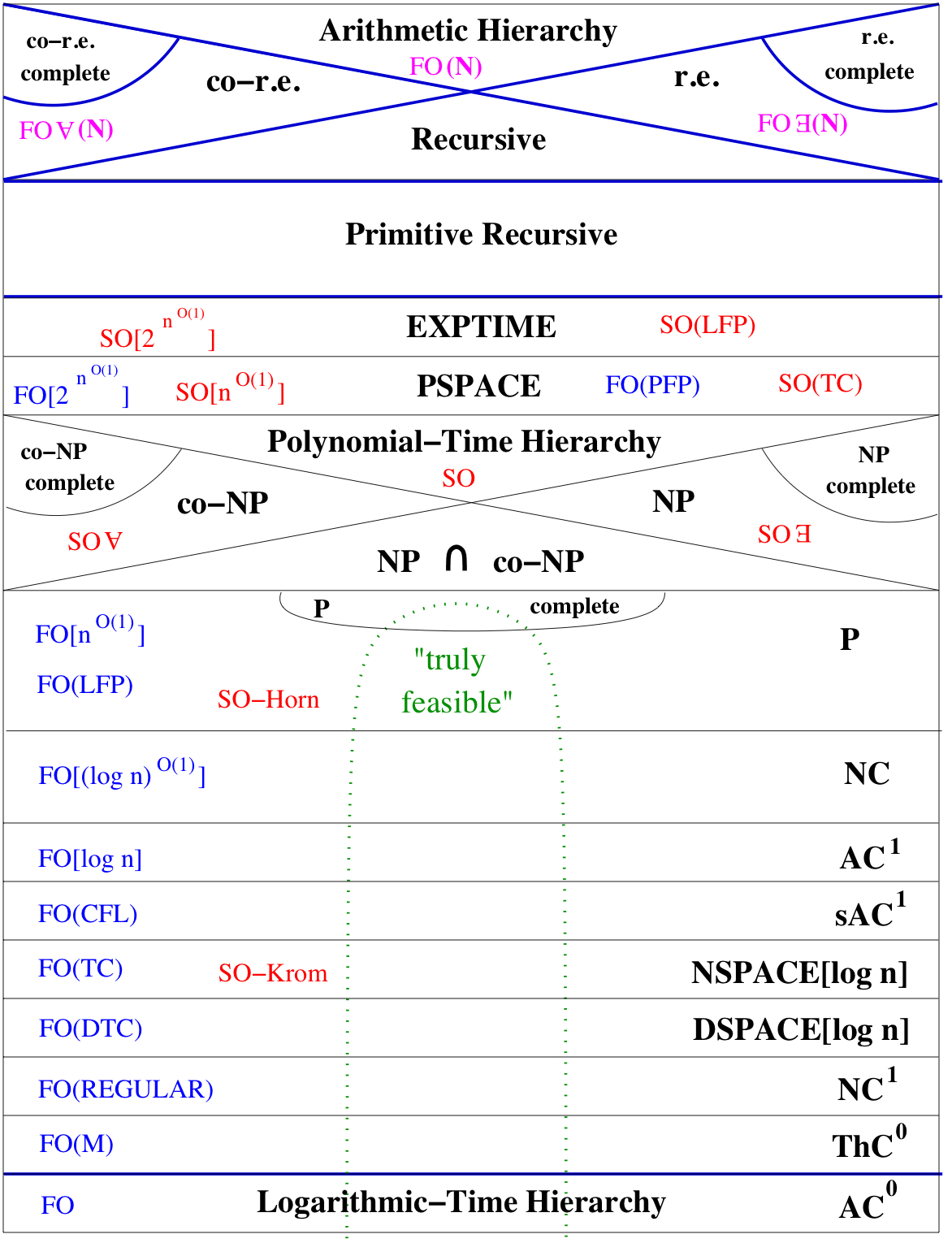
\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Source Listing of Specific Problem Classes:** [petting zoo](https://complexityzoo.uwaterloo.ca/Petting_Zoo)**,** [complexity zoo](https://complexityzoo.uwaterloo.ca/Complexity_Zoo) , [game complexity](https://en.wikipedia.org/wiki/Game_complexity) , …

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Computational Problems:** [complexity garden](https://complexityzoo.uwaterloo.ca/Complexity_Garden), Garey & Johnson (reference 1)

***Another diagram*** of the world of computability and complexity (UMass): **WOW!!***(What do all the notations mean? Some from Automata Theory)*

  
**FO – first order logic; SO – second order logic; … (UMASS)**

**Some additional NP problem classes of interest**

***Strongly NP-Complete:*** A problem is said to be strongly [NP-complete](http://en.wikipedia.org/wiki/NP-complete) (NP-complete in the strong sense), if it remains so even when all of its numerical parameters are bounded by a polynomial in the length of the input.  
(Examples: **Clique (MIS)**, Set-Covering (SCP), Bin Packing, 3-Partition, Satisfiability (SAT), Hamiltonian Cycle, TSP, 3-Colorability, …)

***Strongly NP-Hard:*** A problem is said to be strongly [NP-hard](http://en.wikipedia.org/wiki/NP-hard) if a strongly NP-complete problem has a polynomial reduction to it; in combinatorial optimization, particularly, the phrase "strongly NP-hard" is reserved for problems that are not known to have a polynomial reduction to another strongly NP-complete problem In other words, The [complexity class](http://xlinux.nist.gov/dads/HTML/complexityClass.html) of [decision problems](http://xlinux.nist.gov/dads/HTML/decisionProblem.html) which are still [NP-hard](http://xlinux.nist.gov/dads/HTML/nphard.html) even when all numbers in the input are bounded by some [polynomial](http://xlinux.nist.gov/dads/HTML/polynomial.html) in the length of the input. From a theoretical perspective any strongly NP-hard optimization problem with a polynomially bounded objective function cannot have a **fully polynomial-time approximation scheme (**[FPTAS](http://en.wikipedia.org/wiki/FPTAS) – *Talbi p22*).

***weak NP-Hard: (weakly NP-Complete)***A [NP-complete](http://en.wikipedia.org/wiki/NP-complete) (or [NP-hard](http://en.wikipedia.org/wiki/NP-hard)) problem is **weakly NP-complete** (or weakly NP-hard), if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem (pseudo-polynomial) and the magnitudes of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes. Such algorithms are technically exponential functions of their input size and are therefore not considered polynomial.(such problems have a FPTAS approximation)  
(Examples: 0-1 Knapsack, Subset Sum, Partition)

***\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***NP-Complete (another quick review):*** The [complexity class](http://xlinux.nist.gov/dads/HTML/complexityClass.html) of [decision problems](http://xlinux.nist.gov/dads/HTML/decisionProblem.html) for which answers can be checked for correctness, given a [certificate](http://xlinux.nist.gov/dads/HTML/certificate.html), by an algorithm whose run time is [polynomial](http://xlinux.nist.gov/dads/HTML/polynomialtm.html) in the size of the input (that is, it is [NP](http://xlinux.nist.gov/dads/HTML/np.html)) and no other NP problem is more than a polynomial factor harder. Informally, a problem is NP-complete if answers can be verified quickly, and a quick algorithm to solve this problem can be used to solve all other NP problems quickly. Again, *"NP" comes from the complexity class of language problems for which a* [***N***ondeterministic Turing machine](http://xlinux.nist.gov/dads/HTML/nondetermTuringMach.html) *accepts a string of the language in* ***P****olynomial time.(problems are encoded as strings)*(Examples: <http://en.wikipedia.org/wiki/List_of_NP-complete_problems>)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Pseudo-polynomial algorithm*** (review?) An algorithm which running time is polynomial in the numeric value (size of the numbers) in the input is called a pseudo-polynomial algorithm {*note that many strongly NP-C problems can use a* ***polynomial-time approximation scheme*** *(****PTAS – Talbi p22****) resulting in solutions some distance from the optimal solution; i.e., Euclidean TSP!!*}

“Is quantum computing more powerful than digital computing?”

CSCE686 Lecture 1, ver c, 6/19

***Quantum complexity class:*** uses a quantum model of computation, such as a standard [quantum computer](https://en.wikipedia.org/wiki/Quantum_computer) or a [quantum Turing machine](https://en.wikipedia.org/wiki/Quantum_Turing_machine). Two important quantum complexity classes are [BQP](https://en.wikipedia.org/wiki/BQP) and [QMA](https://en.wikipedia.org/wiki/QMA) which are the bounded-error quantum analogues of [P](https://en.wikipedia.org/wiki/P_(complexity)) and [NP](https://en.wikipedia.org/wiki/NP_(complexity)). One of the main aims of quantum complexity theory is to find out where these classes lie with respect to classical complexity classes such as P, NP, [PP](https://en.wikipedia.org/wiki/PP_(complexity)), [PSPACE](https://en.wikipedia.org/wiki/PSPACE) and [other complexity classes](https://en.wikipedia.org/wiki/List_of_complexity_classes).   
(see various YouTube videos)  
  
***BQP, bounded-error quantum polynomial time:*** class of [decision problems](https://en.wikipedia.org/wiki/Decision_problems) solvable by a [quantum computer](https://en.wikipedia.org/wiki/Quantum_computer) in [polynomial time](https://en.wikipedia.org/wiki/Polynomial_time), with an error probability of at most 1/3 for all instances. A decision problem is a member of **BQP** if there exists a [quantum algorithm](https://en.wikipedia.org/wiki/Quantum_algorithm) (an [algorithm](https://en.wikipedia.org/wiki/Algorithm) that runs on a quantum computer) that solves the decision problem with *high* probability and is guaranteed to run in polynomial time. A run of the algorithm will correctly solve the decision problem with a probability of at least 2/3. It is the quantum analogue of the complexity class [BPP](https://en.wikipedia.org/wiki/BPP_(complexity)). The complexity class [BQP](https://en.wikipedia.org/wiki/BQP) is defined to be the set of problems solvable by a quantum computer in polynomial time with bounded error (efficient algorithm).   
P ≤ BPP ≤ BQP ≤ AWPP ≤ PP ≤ PSPACE (Generate Venn diagram? See references)  
***AWPP, almost wide probabilistic polynomial-time***: a complexity class for problems in the context of [quantum computing](https://en.wikipedia.org/wiki/Quantum_computing). AWPP contains the [BQP](https://en.wikipedia.org/wiki/BQP) (bounded error, quantum, polynomial time) class, which contains the [decision problems](https://en.wikipedia.org/wiki/Decision_problem) solvable by a quantum computer in [polynomial time](https://en.wikipedia.org/wiki/Polynomial_time), with an error probability of at most 1/3 for all instances. In fact, it is the best known classical upper bound for BQP. Furthermore, it is contained in the [APP](https://en.wikipedia.org/w/index.php?title=APP_(complexity_class)&action=edit&redlink=1) class.  
***MA, Arthur–Merlin protocol:*** is an [interactive proof system](https://en.wikipedia.org/wiki/Interactive_proof_system) in which the verifier's coin tosses are constrained to be public (i.e. known to the prover too). This notion was introduced by [Babai (1985)](https://en.wikipedia.org/wiki/Arthur%E2%80%93Merlin_protocol#CITEREFBabai1985). [Goldwasser & Sipser (1986)](https://en.wikipedia.org/wiki/Arthur%E2%80%93Merlin_protocol#CITEREFGoldwasserSipser1986) proved that all (formal) [languages](https://en.wikipedia.org/wiki/Formal_language) with interactive proofs of arbitrary length with private coins also have interactive proofs with public coins.   
***QMA****,* ***Quantum*** [***Merlin Arthur***](https://en.wikipedia.org/wiki/Arthur%E2%80%93Merlin_protocol) : quantum analog of the nonprobabilistic [complexity class](https://en.wikipedia.org/wiki/Complexity_class) [NP](https://en.wikipedia.org/wiki/NP_(complexity)) or the probabilistic complexity class [MA](https://en.wikipedia.org/wiki/Arthur%E2%80%93Merlin_protocol). [QMA](https://en.wikipedia.org/wiki/QMA) is related to [BQP](https://en.wikipedia.org/wiki/BQP) in the same way [NP](https://en.wikipedia.org/wiki/NP_(complexity)) is related to [P](https://en.wikipedia.org/wiki/P_(complexity)), or [MA](https://en.wikipedia.org/wiki/Arthur%E2%80%93Merlin_protocol) is related to [BPP](https://en.wikipedia.org/wiki/Bounded-error_probabilistic_polynomial). Informally, it is the set of [decision problems](https://en.wikipedia.org/wiki/Decision_problem) for which when the answer is YES, there is a polynomial-size quantum proof (a quantum state) which convinces a polynomial-time quantum verifier of the fact with high probability. Moreover, when the answer is NO, every polynomial-size quantum state is rejected by the verifier with high probability. More precisely, the proofs have to be verifiable in [polynomial time](https://en.wikipedia.org/wiki/Polynomial_time) on a [quantum computer](https://en.wikipedia.org/wiki/Quantum_computer), such that if the answer is indeed YES, the verifier accepts a correct proof with probability greater than 2/3, and if the answer is NO, then there is no proof which convinces the verifier to accept with probability greater than 1/3. As is usually the case, the constants 2/3 and 1/3 can be changed. Changing 2/3 to any constant strictly between 1/2 and 1, or changing 1/3 to any constant strictly between 0 and 1/2, does not change class QMA.   
P ≤ NP ≤ MA ≤ QMA ≤ PP ≤ PSPACE  
***QAM:***  a related complexity class, in which fictional agents Arthur and Merlin carry out the sequence: Arthur generates a random string, Merlin answers with a quantum [certificate](https://en.wikipedia.org/wiki/Certificate_(complexity)) and Arthur verifies it as a BQP machine.   
Refs: [MIT course on Quantum Computing Complexity](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-845-quantum-complexity-theory-fall-2010/); [Watrous, QC Complexity](https://arxiv.org/pdf/0804.3401.pdf)

**quantum circuit**: a [model](https://en.wikipedia.org/wiki/Model_(abstract)) for quantum computation in which a computation is a sequence of [quantum gates](https://en.wikipedia.org/wiki/Quantum_gate), which are reversible transformations on a [quantum mechanical](https://en.wikipedia.org/wiki/Quantum_mechanics) [analog](https://en.wikipedia.org/wiki/Quantum_register) of an *n*-[bit](https://en.wikipedia.org/wiki/Bit) [register](https://en.wikipedia.org/wiki/Processor_register). This analogous structure is referred to as an *n*-[qubit](https://en.wikipedia.org/wiki/Qubit) register. The objective is study the **computational complexity** of compiling **quantum circuits** so that they can be run efficiently (quantum supremacy) on a given hardware configuration; use previous definitions of QC complexity.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Some Unsolved Complexity Problems!: Questions? Impact?***

* [P = NP problem](http://en.wikipedia.org/wiki/P_%3D_NP_problem)
* [NC = P problem](http://en.wikipedia.org/wiki/NC_%3D_P_problem)
* [NP = co-NP problem](http://en.wikipedia.org/wiki/NP_%3D_co-NP_problem)
* [P = BPP problem](http://en.wikipedia.org/wiki/P_%3D_BPP_problem)
* [P = PSPACE problem](http://en.wikipedia.org/wiki/P_%3D_PSPACE_problem)
* [L = NL problem](http://en.wikipedia.org/wiki/NL_(complexity)) (Nondeterministic-Logarithmic-Space)
* L = P problem (L is polynomial-time-logarithmic-space)
* L = RL problem (Randomized Logarithmic-space Polynomial-time)
* [Unique games conjecture](http://en.wikipedia.org/wiki/Unique_games_conjecture)
* Is the [exponential time hypothesis](http://en.wikipedia.org/wiki/Exponential_time_hypothesis) true?

***Some unsolved Problems in algorithms:***

* What is the fastest [algorithm for multiplication](http://en.wikipedia.org/wiki/Multiplication_algorithm#Fast_multiplication_algorithms_for_large_inputs) of two *n*-digit numbers?
* What is the fastest algorithm for [matrix multiplication](http://en.wikipedia.org/wiki/Matrix_multiplication#Algorithms_for_efficient_matrix_multiplication)?
* Can [integer factorization](http://en.wikipedia.org/wiki/Integer_factorization) be done in [polynomial time](http://en.wikipedia.org/wiki/Polynomial_time) on a classical computer?
* Can the [discrete logarithm](http://en.wikipedia.org/wiki/Discrete_logarithm#Algorithms) be computed in polynomial time on a classical computer?
* Can the [graph isomorphism problem](http://en.wikipedia.org/wiki/Graph_isomorphism_problem) be solved in polynomial time?
* Can [parity games](http://en.wikipedia.org/wiki/Parity_game) be solved in polynomial time?

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***References:***

1. [Garey, M. R.](http://en.wikipedia.org/wiki/Michael_R._Garey); [Johnson, D. S.](http://en.wikipedia.org/wiki/David_S._Johnson) (1979). In [Victor Klee](http://en.wikipedia.org/wiki/Victor_Klee). [*Computers and Intractability: A Guide to the Theory of NP-Completeness*](http://en.wikipedia.org/wiki/Computers_and_Intractability:_A_Guide_to_the_Theory_of_NP-Completeness). A Series of Books in the Mathematical Sciences. W. H. Freeman and Co. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-7167-1045-5](http://en.wikipedia.org/wiki/Special:BookSources/0-7167-1045-5).
2. Kleinberg and Tardos, Algorithms Design, Chapter 8 (**CSCE586 text**)
3. Complexity Zoo: <https://complexityzoo.uwaterloo.ca/Complexity_Zoo:C> **(Fantastic!)**
4. Wikipedia per the various complexity class terms: NP, Co-NP, QMA, …
5. Arora and Barak, Computational Complexity: A Modern Approach, 2007, <http://theory.cs.princeton.edu/complexity/bppchap.pdf>
6. Perry, Quantum Computing from the Ground Up, World Scientific, 2012