Complexity theory concerns itself with two kinds of measures: time and space.

Time complexity is a measure of how long a computation takes to execute. For a digital computer, this could be represented by the number of machine cycles required for the computation.

Space complexity is a measure of the storage required for a computation. For a digital computer, this could be the number of bytes used.

Such measures are functions of a single input parameter, the **size of the input**.

Given a particular input problem class, we might be interested in either the **average case** or the **worst case**. The worst-case complexity tends to be preferred because

- it may be difficult or impossible to define an "average" case. For many problems, the notion of "average case" doesn't even make sense,
- It is usually much easier to compute worst-case complexity.

In complexity theory we generally subject our equations to some extreme simplifications. For example, if a given algorithm takes exactly $5n^3 + 2n^2 - n + 1003$ machine cycles where *n* is the size of the input problem, we will simplify this to $O(n^3)$ (read: order *n*-cubed). This is called an **order statistic**. Specifically, we:

- drop all the terms except the one of the highest-order,
- drop the coefficient of the highest-order term.

This is because:

- for very large values of *n*, the effect of the highest-order term completely overrides that of lower-order terms.
- improving the code may improve the coefficients, but not the order statistic.

A **polynomial-time algorithm** is an algorithm whose execution time is bounded by a polynomial. Problems that can be solved by a polynomial-time algorithm are called **tractable problems**.

There are also a large number of practical problems, for which no polynomial algorithm has yet been found. These are called **intractable**.

To demonstrate that intractability is a serious problem, let us take a look at finding a minimum Hamiltonian cycle. In terms of the worst case, all algorithms solving this problem amount to the "exhaustive search" or "rude force" algorithm. With *n* vertices, there are $\frac{(n-1)!}{2}$ different cycles.

Suppose it takes 0.0000001 sec. to check each cycle. We get the following table:

n	*Time needed to solve the problem*
5	0.0000012 **seconds**
15	1 **hour** 12 **minutes**
20	192.86 **years** 10 months 2 weeks
30	14 000 000 000 000 000 **years**

Some problems can still be solved in polynomial time, even if a polynomial **deterministic** algorithm does not exist. All we need is "a little bit of luck". For example, the problem of determining whether a given graph contains a Hamiltonian cycle is among those known to be intractable. But, with a little bit of luck we may hit a cycle that **happens to be** Hamiltonian. Then it takes very little time to check this. These considerations lead to the concept of non-deterministic computation.

A **nondeterministic computation** can be viewed in either of two ways:

- When a choice point is reached, an infallible ***oracle*** can be consulted to determine the correct choice.

The Delphic oracle is a temple in the town of Delphi in Greece where, in ancient times, a priestess named Pythia gave answers from the god Apollo to questions people asked him. His answers were often mysterious and difficult to understand, and were often in the form of a riddle.

- When a choice point is reached, all choices can be made and computation can proceed simultaneously.

	1	2	3	4	5
1	_	304	258	634	494
2	304	_	53	740	382
3	258	53	_	350	615
4	634	740	350	_	442
5	494	382	615	442	_

(1,4,2,3,5)	☼ - - - **Processor 1** - - - - - >	2536	
(1,2,5,4,3)	☼ - - - **Processor 2** - - - - - >	1736	
(1,4,2,5,3)	☼ - - - **Processor 3** - - - - - >	2629	
(1,2,3,4,5)	☼ - - - **Processor 4** - - - - - >	1643	**BEST RESULT**
(1,2,4,3,5)	☼ - - - **Processor 5** - - - - - >	2503	
(1,4,5,2,3)	☼ - - - **Processor 6** - - - - - >	1769	
(1,2,4,5,3)	☼ - - - **Processor 7** - - - - - >	2359	
(1,5,2,3,4)	☼ - - - **Processor 8** - - - - - >	1913	
(1,2,5,3,4)	☼ - - - **Processor 9** - - - - - >	2285	
(1,5,4,2,3)	☼ - - - **Processor 10** - - - - >	1987	
(1,2,3,5,4)	☼ - - - **Processor 11** - - - - >	2048	
(1,5,2,4,3)	☼ - - - **Processor 12** - - - - >	2224	

A **nondeterministic polynomial-time problem** or an **NP problem** is a problem that can be solved in polynomial time on a nondeterministic machine.

What follows is a list of some NP problems.

Integer Bin Packing

Given a set of *n* positive integers, arrange them into two bins so that the sum of the integers in either bin is the same.

For example, given the integers 19, 23, 32, 42, 50, 62, 77, 88, 89, 105, 114, 123, 176 whose sum is 1000, can they be divided into two bins *A* and *B* so that the sum of each bin is 500? There are variations of this problem such as:

- there are more than two bins,
- for a single bin find, find a subset of integers that fit into the bin and sum up to the greatest value possible - also known as the **knapsack problem**.

Boolean satisfiability problem (SAT). Having a Boolean expression written using only operators AND, OR, and NOT, variables, and parentheses, the question is: is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true?

The problem can be further restricted. We can assume that NOT operators are only applied to variables, not expressions. If we OR together a group of literals, we get a *clause*, such as (x_1 *or* not(x_2)). If we consider formulas that are a conjunction (AND) of clauses, we call this form conjunctive normal form. Another restriction is that each clause is limited to at most three literals. This last problem is called 3SAT, 3CNFSAT, or 3-satisfiability.

Hamiltonian cycle problem. Given an ordinary graph with n vertices, determine whether it has a Hamiltonin cycle, that is, a cycle that contains all the n vertices of the graph.

Note that when we know of a sufficient condition for the existence of a Hamiltonian cycle that such a graph fulfils, such as $\deg(u)+\deg(v)\geq n$ for any two non-neighbouring vertices u, v of the graph, then the solution is easy but, generally, the problem is NP.

Travelling Salesman Problem (TSP). In a complete ordinary graph $G=(V,E)$, each $e\in E$ is assigned a positive number $c(e)$. Find a Hamiltonian cycle $H=(v_1,e_1,v_2e_2,\ldots v_{n-1},e_{n-1},v_n,e_n,v_1)$ of G such that $c(H)=\sum_{i=1}^{n}e_i$ is minimal.

There is a weaker form of TSP which is still NP:

Given a complete ordinary graph $G=(V,E)$ with each $e\in E$ assigned a positive number $c(e)$ and a positive number K, determine whether a Hamiltonian cycle H of G exist such that $c(H)\geq K$.

Some of the NP problems have a remarkable property: they are all reducible to each other. It means that, given any two NP problems *X* and *Y*,

- there exists a polynomial-time algorithm to restate a problem of type *X* as a problem of type *Y*, and
- there exists a polynomial-time algorithm to translate a solution to a type *Y* problem back into a solution for the type *X* problem.

Therefore the set of all the NP problems that can be reduced to each other are referred to as **NP-complete problems**. All the above listed NP problems happen to be also NP complete.

This means is that, if anyone ever discovers a polynomial-time algorithm for **any** of these problems, then there is an easily-derived polynomial-time algorithm for **all** of them. This leads to the famous question:

P = NP

No one has ever found a deterministic polynomial-time algorithm for any of these problems. However, no one has ever succeeded in proving that no deterministic polynomial-time algorithm exists, either. Most computer scientists are convinced that a polynomial-time algorithm cannot exist, but no one knows for sure.