Algorithm

Broadly-defined, an algorithm is an interpretable, finite set of instructions for dealing with contingencies and accomplishing some task which can be anything that has a recognizable end-state, end-point, or result. Usually steps in the procedure may repeat (iteration) or require decisions (logic and comparison) until the task is completed. Different algorithms may complete the same task with a different set of instructions in more or less time, space, or effort than others. A cooking recipe is an example of an algorithm. One of two different recipes for making potato salad may have "peel the potato" before "boil the potato" while the other vice versa, but they both call for those steps to be repeated for all potatoes and end when the potato salad is ready to eat.

Computational complexity theory

Complexity Theory is part of the theory of computation dealing with the resources required during computation to solve a given problem. The most common resources are time (how many steps does it take to solve a problem) and space (how much memory does it take to solve a problem). Other resources can also be considered, such as how many parallel processors are needed to solve a problem in parallel. Complexity theory differs from computability theory, which deals with whether a problem can be solved at all, regardless of the resources required.

A single "problem" is an entire set of related questions, where each question is a finite-length string. For example, the problem FACTORIZE is: given an integer written in binary, return all of the prime factors of that number. A particular question is called an instance. For example, "give the factors of the number 15" is one instance of the FACTORIZE problem.

The time complexity of a problem is the number of steps that it takes to solve an instance of the problem, as a function of the size of the input, (usually measured in bits) using the most efficient algorithm. This can be intuitively thought of as: If an instance that is \(n\) bits long can be solved in \(2n\) steps, then we say it has a time complexity of \(2n\). Of course, the exact number of steps will depend on exactly what machine or language is being used. To avoid that problem, we generally use Big \(O\) notation. If a problem has time complexity \(O(2n)\) on one typical computer, then it will also have complexity \(O(2n)\) on most
other computers, so this notation allows us to generalize away from the details of a particular computer.

**Big O notation**

We say a function $f = O(g)$ if there exist constants $c_1$ and $c_2$ such that $f \leq c_1g + c_2$ for all possible input values.

**Decision Problems**

Much of complexity theory deals with decision problems. A decision problem is a problem where the answer is always YES/NO. For example, the problem IS-PRIME is: given an integer written in binary, return whether it is a prime number or not.

Decision problems are often considered because an arbitrary problem can always be reduced to a decision problem. For example, the problem HAS-FACTOR is: given integers $n$ and $k$ written in binary, return whether $n$ has any prime factors less than $k$. If we can solve HAS-FACTOR with a certain amount of resources, then we can use that solution to solve FACTORIZE without much more resources. Just do a binary search on $k$ until you find the smallest factor of $n$. Then divide out that factor, and repeat until you find all the factors.

Complexity theory often makes a distinction between YES answers and NO answers. For example, the set NP is defined as the set of problems where the YES instances can be checked quickly. The set Co-NP is the set of problems where the NO instances can be checked quickly. The "Co" in the name stands for "complement". The complement of a problem is one where all the YES and NO answers are swapped, such as IS-COMPOSITE for IS-PRIME.

**Famous Complexity Classes**

The following are some of the classes of problems considered in complexity theory, along with rough definitions.

- **P**: Solvable in polynomial time
- **NP**: YES answers checkable in polynomial time
- **Co-NP**: NO answers checkable in polynomial time
- **NP-complete**: The hardest problems in NP
- **Co-NP-complete**: The hardest problems in Co-NP
- **NP-hard**: Either NP-complete or harder
- **NP-easy**: non-decision-problem analogue to NP
- **NP-equivalent non-decision-problem analogue to NP-complete**
The P=NP Question

The question of whether P is the same set as NP is the most important open question in theoretical computer science. There is even a $1,000,000 prize for solving it.

Comparison of time complexities

Fill in the following table. Consider that you are using a very fast computer, lets say 1 gigahertz, that is $10^9$ computations per second.

<table>
<thead>
<tr>
<th>n</th>
<th>$n^2$</th>
<th>$n^{10}$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$10^{-7}$ seconds</td>
<td>2.84 hours</td>
<td>1.07 seconds</td>
<td>77.14 years</td>
</tr>
<tr>
<td>20</td>
<td>2.84 hours</td>
<td>1.07 seconds</td>
<td>77.14 years</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1.07 seconds</td>
<td>77.14 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>77.14 years</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>