For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability.  
Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash.  
When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear.  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.  
Normally the first step in debugging is to attempt to reproduce the problem.  
It is usually easier to code in "high-level" languages than in "low-level" ones.  
 Programmable devices have existed for centuries.  
Transpiling on the other hand, takes the source-code from a high-level programming language and converts it into bytecode.  
 Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA).  
Proficient programming thus usually requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.  
 The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems.