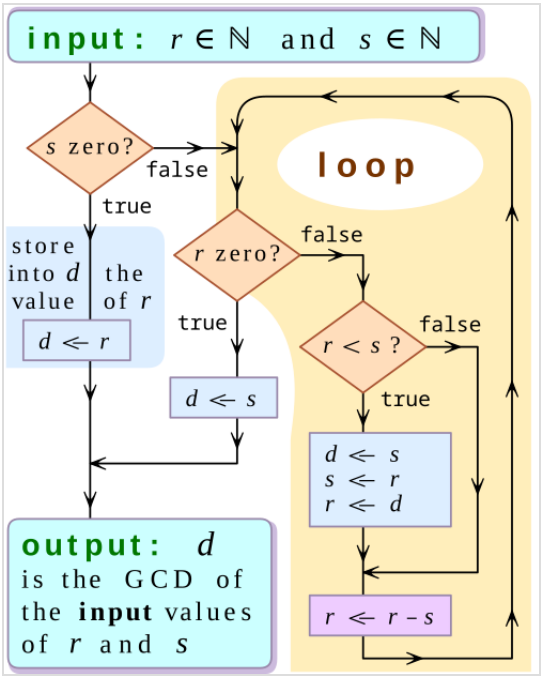
WIKIPEDIA
The Free Encyclopedia

Algorithm

In mathematics and computer science, an **algorithm** (/ˈælɡərɪðəm/ ⓘ) is a finite sequence of mathematically rigorous instructions, typically used to solve a class of specific problems or to perform a computation.[1] Algorithms are used as specifications for performing calculations and data processing. More advanced algorithms can use conditionals to divert the code execution through various routes (referred to as automated decision-making) and deduce valid inferences (referred to as automated reasoning).



Flowchart of using successive subtractions to find the greatest common divisor of number *r* and *s*

In contrast, a heuristic is an approach to solving problems without well-defined correct or optimal results.[2] For example, although social media recommender systems are commonly called "algorithms", they actually rely on heuristics as there is no truly "correct" recommendation.

As an effective method, an algorithm can be expressed within a finite amount of space and time[3] and in a well-defined formal language[4] for calculating a function.[5] Starting from an initial state and initial input (perhaps empty),[6] the instructions describe a computation that, when executed, proceeds through a finite[7] number of well-defined successive states, eventually producing "output"[8] and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.[9]

Etymology

Around 825 AD, Persian scientist and polymath Muḥammad ibn Mūsā al-Khwārizmī wrote *kitāb al-ḥisāb al-hindī* ("Book of Indian computation") and *kitab al-jam' wa'l-tafriq al-ḥisāb al-hindī* ("Addition and subtraction in Indian arithmetic"). In the early 12th century, Latin translations of these texts involving the Hindu–Arabic numeral system and arithmetic appeared, for example *Liber Alghoarismi de practica arismetrice*, attributed to John of Seville, and *Liber Algorismi de numero Indorum*, attributed to Adelard of Bath.[10] Here, *alghoarismi* or *algorismi* is the Latinization of Al-Khwarizmi's name;[1] the text starts with the phrase *Dixit Algorismi*, or "Thus spoke Al-Khwarizmi".[2]

The word *algorism* in English came to mean the use of place-value notation in calculations; it occurs in the *Ancrene Wisse* from circa 1225.[11] By the time Geoffrey Chaucer wrote *The Canterbury Tales* in the late 14th century, he used a variant of the same word in describing *augrym stones*, stones used for place-value calculation.[12][13] In the 15th century, under the influence of the Greek word ἀριθμός (*arithmos*, "number"; *cf.* "arithmetic"), the Latin word was altered to *algorithmus*.[14] By 1596, this form of the word was used in English, as *algorithm*, by Thomas Hood.[15]

Definition

One informal definition is "a set of rules that precisely defines a sequence of operations",[16] which would include all computer programs (including programs that do not perform numeric calculations), and any prescribed bureaucratic procedure[17] or cook-book recipe.[18] In general, a program is an algorithm only if it stops eventually[19]—even though infinite loops may sometimes prove desirable. Boolos, Jeffrey & 1974, 1999 define an algorithm to be an explicit set of instructions for determining an output, that can be followed by a computing machine or a human who could only carry out specific elementary operations on symbols.[20]

Most algorithms are intended to be implemented as computer programs. However, algorithms are also implemented by other means, such as in a biological neural network (for example, the human brain performing arithmetic or an insect looking for food), in an electrical circuit, or a mechanical device.

History

Ancient algorithms

Step-by-step procedures for solving mathematical problems have been recorded since antiquity. This includes in Babylonian mathematics (around 2500 BC),[21] Egyptian mathematics (around 1550 BC),[21] Indian mathematics (around 800 BC and later),[22][23] the Ifa Oracle (around 500 BC),[24] Greek mathematics (around 240 BC),[25] Chinese mathematics (around 200 BC and later),[26] and Arabic mathematics (around 800 AD).[27]

The earliest evidence of algorithms is found in ancient Mesopotamian mathematics. A Sumerian clay tablet found in Shuruppak near Baghdad and dated to c. 2500 BC describes the earliest division algorithm.[21] During the Hammurabi dynasty c. 1800 – c. 1600 BC, Babylonian clay tablets described algorithms for computing formulas.[28] Algorithms were also used in Babylonian astronomy. Babylonian clay tablets describe and employ algorithmic procedures to compute the time and place of significant astronomical events.[29]

Algorithms for arithmetic are also found in ancient Egyptian mathematics, dating back to the Rhind Mathematical Papyrus c. 1550 BC.[21] Algorithms were later used in ancient Hellenistic mathematics. Two examples are the Sieve of Eratosthenes, which was described in the *Introduction to Arithmetic* by Nicomachus,[30][25]:Ch 9.2 and the Euclidean algorithm, which was first described in *Euclid's Elements* (c. 300 BC).[25]:Ch 9.1 Examples of ancient Indian mathematics included the Shulba Sutras, the Kerala School, and the Brāhmasphuṭasiddhānta.[22]

The first cryptographic algorithm for deciphering encrypted code was developed by Al-Kindi, a 9th-century Arab mathematician, in *A Manuscript On Deciphering Cryptographic Messages*. He gave the first description of cryptanalysis by frequency analysis, the earliest codebreaking algorithm.[27]

Computers

Weight-driven clocks

Bolter credits the invention of the weight-driven clock as "the key invention [of Europe in the Middle Ages]," specifically the verge escapement mechanism[31] producing the tick and tock of a mechanical clock. "The accurate automatic machine"[32] led immediately to "mechanical automata" in the 13th century and "computational machines"—the difference and analytical engines of Charles Babbage and Ada Lovelace in the mid-19th century.[33] Lovelace designed the first algorithm intended for processing on a computer, Babbage's analytical engine, which is the first device considered a real Turing-complete computer instead of just a calculator. Although the full implementation of Babbage's second device was not realized for decades after her lifetime, Lovelace has been called "history's first programmer".

Electromechanical relay

Bell and Newell (1971) write that the Jacquard loom, a precursor to Hollerith cards (punch cards), and "telephone switching technologies" led to the development of the first computers.[34] By the mid-19th century, the telegraph, the precursor of the telephone, was in use throughout the world. By the late 19th century, the ticker tape (c. 1870s) was in use, as were Hollerith cards (c. 1890). Then came the teleprinter (c. 1910) with its punched-paper use of Baudot code on tape.

Telephone-switching networks of electromechanical relays were invented in 1835. These led to the invention of the digital adding device by George Stibitz in 1937. While working in Bell Laboratories, he observed the "burdensome" use of mechanical calculators with gears. "He went home one evening in 1937 intending to test his idea... When the tinkering was over, Stibitz had constructed a binary adding device".[35][36]

Formalization

In 1928, a partial formalization of the modern concept of algorithms began with attempts to solve the *Entscheidungsproblem* (decision problem) posed by David Hilbert. Later formalizations were framed as attempts to define "effective calculability"[37] or "effective method".[38] Those formalizations included the Gödel–Herbrand–Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines of 1936–37 and 1939.

Modern Algorithms

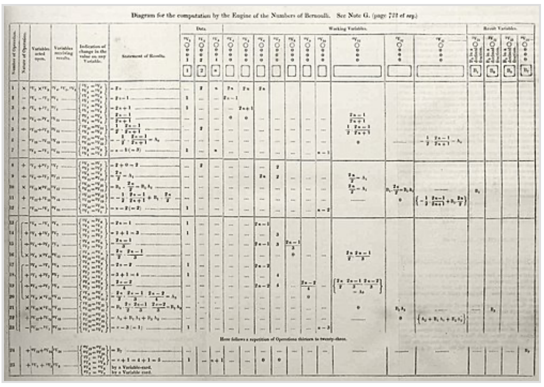


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Ada Lovelace's diagram from "Note G", the first published computer algorithm

Algorithms have evolved and improved in many ways as time goes on. Common uses of algorithms today include social media apps like Instagram and YouTube. Algorithms are used as a way to analyze what people like and push more of those things to the people who interact with them. Quantum computing uses quantum algorithm procedures to solve problems faster. More recently, in 2024, NIST updated their post-quantum encryption standards, which includes new encryption algorithms to enhance defenses against attacks using quantum computing.

Representations

Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, drakon-charts, programming languages or control tables (processed by interpreters). Natural language expressions of algorithms tend to be verbose and ambiguous and are rarely used for complex or technical algorithms. Pseudocode, flowcharts, drakon-charts, and control tables are structured expressions of algorithms that avoid common ambiguities of natural language. Programming languages are primarily for expressing algorithms in a computer-executable form but are also used to define or document algorithms.

Turing machines

There are many possible representations and Turing machine programs can be expressed as a sequence of machine tables (see finite-state machine, state-transition table, and control table for more), as flowcharts and drakon-charts (see state diagram for more), as a form of rudimentary machine code or assembly code called "sets of quadruples", and more. Algorithm representations can also be classified into three accepted levels of Turing machine description: high-level description, implementation description, and formal description.[39] A high-level description describes the qualities of the algorithm itself, ignoring how it is implemented on the Turing machine.[39] An implementation description describes the general manner in which the machine moves its head and stores data to carry out the algorithm, but does not give exact states.[39] In the most detail, a formal description gives the exact state table and list of transitions of the Turing machine.[39]

Flowchart representation

The graphical aid called a flowchart offers a way to describe and document an algorithm (and a computer program corresponding to it). It has four primary symbols: arrows showing program flow, rectangles (SEQUENCE, GOTO), diamonds (IF-THEN-ELSE), and dots (OR-tie). Sub-structures can "nest" in rectangles, but only if a single exit occurs from the superstructure.

Algorithmic analysis

It is often important to know how much time, storage, or other cost an algorithm may require. Methods have been developed for the analysis of algorithms to obtain such quantitative answers (estimates); for example, an algorithm that adds up the elements of a list of n numbers would have a time requirement of $O(n)$, using big O notation. The algorithm only needs to remember two values: the sum of all the elements so far, and its current position in the input list. If the space required to store the input numbers is not counted, it has a space requirement of $O(1)$, otherwise $O(n)$ is required.

Different algorithms may complete the same task with a different set of instructions in less or more time, space, or 'effort' than others. For example, a binary search algorithm (with cost $O(\log n)$) outperforms a sequential search (cost $O(n)$) when used for table lookups on sorted lists or arrays.

Formal versus empirical

The analysis, and study of algorithms is a discipline of computer science. Algorithms are often studied abstractly, without referencing any specific programming language or implementation. Algorithm analysis resembles other mathematical disciplines as it focuses on the algorithm's properties, not implementation. Pseudocode is typical for analysis as it is a simple and general representation. Most algorithms are implemented on particular hardware/software platforms and their algorithmic efficiency is tested using real code. The efficiency of a particular algorithm may be insignificant for many "one-off" problems but it may be critical for algorithms designed for fast interactive, commercial, or long-life scientific usage. Scaling from small n to large n frequently exposes inefficient algorithms that are otherwise benign.

Empirical testing is useful for uncovering unexpected interactions that affect performance. Benchmarks may be used to compare before/after potential improvements to an algorithm after program optimization. Empirical tests cannot replace formal analysis, though, and are non-trivial to perform fairly.[40]

Execution efficiency

To illustrate the potential improvements possible even in well-established algorithms, a recent significant innovation, relating to FFT algorithms (used heavily in the field of image processing), can decrease processing time up to 1,000 times for applications like medical imaging.[41] In general, speed improvements depend on special properties of the problem, which are very common in practical applications.[42] Speedups of this magnitude enable computing devices that make extensive use of image processing (like digital cameras and medical equipment) to consume less power.

Best Case and Worst Case

The best case of an algorithm refers to the scenario or input for which the algorithm or data structure takes the least time and resources to complete its tasks.[43] The worst case of an algorithm is the case that causes the algorithm or data structure to consume the maximum period of time and computational resources.[44]

Design

Algorithm design is a method or mathematical process for problem-solving and engineering algorithms. The design of algorithms is part of many solution theories, such as divide-and-conquer or dynamic programming within operation research. Techniques for designing and implementing algorithm designs are also called algorithm design patterns,[45] with examples including the template method pattern and the decorator pattern. One of the most important aspects of algorithm design is resource (run-time, memory usage) efficiency; the big O notation is used to describe e.g., an algorithm's run-time growth as the size of its input increases.[46]

Structured programming

Per the Church–Turing thesis, any algorithm can be computed by any Turing complete model. Turing completeness only requires four instruction types—conditional GOTO, unconditional GOTO, assignment, HALT. However, Kemeny and Kurtz observe that, while "undisciplined" use of unconditional GOTOs and conditional IF-THEN GOTOs can result in "spaghetti code", a programmer can write structured programs using only these instructions; on the other hand "it is also possible, and not too hard, to write badly structured programs in a structured language".[47] Tausworthe augments the three Böhm-Jacopini canonical structures:[48] SEQUENCE, IF-THEN-ELSE, and WHILE-DO, with two more: DO-WHILE and CASE.[49] An additional benefit of a structured program is that it lends itself to proofs of correctness using mathematical induction.[50]

Legal status

By themselves, algorithms are not usually patentable. In the United States, a claim consisting solely of simple manipulations of abstract concepts, numbers, or signals does not constitute "processes" (USPTO 2006), so algorithms are not patentable (as in *Gottschalk v. Benson*). However practical applications of algorithms are sometimes patentable. For example, in *Diamond v. Diehr*, the application of a simple feedback algorithm to aid in the curing of synthetic rubber was deemed patentable. The patenting of software is controversial,[51] and there are criticized patents involving algorithms, especially data compression algorithms, such as Unisys's LZW patent. Additionally, some cryptographic algorithms have export restrictions (see export of cryptography).

Classification

By implementation

Recursion

A recursive algorithm invokes itself repeatedly until meeting a termination condition and is a common functional programming method. Iterative algorithms use repetitions such as loops or data structures like stacks to solve problems. Problems may be suited for one implementation or the other. The Tower of Hanoi is a puzzle commonly solved using recursive implementation. Every recursive version has an equivalent (but possibly more or less complex) iterative version, and vice versa.

Serial, parallel or distributed

Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time on serial computers. Serial algorithms are designed for these environments, unlike parallel or distributed algorithms. Parallel algorithms take advantage of computer architectures where multiple processors can work on a problem at the same time. Distributed algorithms use multiple machines connected via a computer network. Parallel and distributed algorithms divide the problem into subproblems and collect the results back together. Resource consumption in these algorithms is not only processor cycles on each processor but also the communication overhead between the processors. Some sorting algorithms can be parallelized efficiently, but their communication overhead is expensive. Iterative algorithms are generally parallelizable, but some problems have no parallel algorithms and are called inherently serial problems.

Deterministic or non-deterministic

Deterministic algorithms solve the problem with exact decisions at every step; whereas non-deterministic algorithms solve problems via guessing. Guesses are typically made more accurate through the use of heuristics.

Exact or approximate

While many algorithms reach an exact solution, approximation algorithms seek an approximation that is close to the true solution. Such algorithms have practical value for many hard problems. For example, the Knapsack problem, where there is a set of items, and the goal is to pack the knapsack to get the maximum total value. Each item has some weight and some value. The total weight that can be carried is no more than some fixed number X. So, the solution must consider the weights of items as well as their value.[52]

Quantum algorithm

Quantum algorithms run on a realistic model of quantum computation. The term is usually used for those algorithms that seem inherently quantum or use some essential feature of Quantum computing such as quantum superposition or quantum entanglement.

By design paradigm

Another way of classifying algorithms is by their design methodology or paradigm. Some common paradigms are:

Brute-force or exhaustive search

Brute force is a problem-solving method of systematically trying every possible option until the optimal solution is found. This approach can be very time-consuming, testing every possible combination of variables. It is often used when other methods are unavailable or too complex. Brute force can solve a variety of problems, including finding the shortest path between two points and cracking passwords.

Divide and conquer

A divide-and-conquer algorithm repeatedly reduces a problem to one or more smaller instances of itself (usually recursively) until the instances are small enough to solve easily. Merge sorting is an example of divide and conquer, where an unordered list is repeatedly split into smaller lists, which are sorted in the same way and then merged.[53] In a simpler variant of divide and conquer called prune and search or *decrease-and-conquer algorithm*, which solves one smaller instance of itself, and does not require a merge step.[54] An example of a prune and search algorithm is the binary search algorithm.

Search and enumeration

Many problems (such as playing chess) can be modelled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration, and backtracking.

Randomized algorithm

Such algorithms make some choices randomly (or pseudo-randomly). They find approximate solutions when finding exact solutions may be impractical (see heuristic method below). For some problems, the fastest approximations must involve some randomness.[55] Whether randomized algorithms with polynomial time complexity can be the fastest algorithm for some problems is an open question known as the P versus NP problem. There are two large classes of such algorithms:

1. Monte Carlo algorithms return a correct answer with high probability. E.g. RP is the subclass of these that run in polynomial time.
2. Las Vegas algorithms always return the correct answer, but their running time is only probabilistically bound, e.g. ZPP.

Reduction of complexity

This technique transforms difficult problems into better-known problems solvable with (hopefully) asymptotically optimal algorithms. The goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithms. For example, one selection algorithm finds the median of an unsorted list by first sorting the list (the expensive portion), and then pulling out the middle element in the sorted list (the cheap portion). This technique is also known as *transform and conquer*.

Back tracking

In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution.

Optimization problems

For optimization problems there is a more specific classification of algorithms; an algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

Linear programming

When searching for optimal solutions to a linear function bound by linear equality and inequality constraints, the constraints can be used directly to produce optimal solutions. There are algorithms that can solve any problem in this category, such as the popular simplex algorithm.[56] Problems that can be solved with linear programming include the maximum flow problem for directed graphs. If a problem also requires that any of the unknowns be integers, then it is classified in integer programming. A linear programming algorithm can solve such a problem if it can be proved that all restrictions for integer values are superficial, i.e., the solutions satisfy these restrictions anyway. In the general case, a specialized algorithm or an algorithm that finds approximate solutions is used, depending on the difficulty of the problem.

Dynamic programming

When a problem shows optimal substructures—meaning the optimal solution can be constructed from optimal solutions to subproblems—and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called *dynamic programming* avoids recomputing solutions. For example, Floyd–Warshall algorithm, the shortest path between a start and goal vertex in a weighted graph can be found using the shortest path to the goal from all adjacent vertices. Dynamic programming and memoization go together. Unlike divide and conquer, dynamic programming subproblems often overlap. The difference between dynamic programming and simple recursion is the caching or memoization of recursive calls. When subproblems are independent and do not repeat, memoization does not help; hence dynamic programming is not applicable to all complex problems. Using memoization dynamic programming reduces the complexity of many problems from exponential to polynomial.

The greedy method

Greedy algorithms, similarly to a dynamic programming, work by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution and improve it by making small modifications. For some problems, they always find the optimal solution but for others they may stop at local optima. The most popular use of greedy algorithms is finding minimal spanning trees of graphs without negative cycles. Huffman Tree, Kruskal, Prim, Sollin are greedy algorithms that can solve this optimization problem.

The heuristic method

In optimization problems, heuristic algorithms find solutions close to the optimal solution when finding the optimal solution is impractical. These algorithms get closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. They can ideally find a solution very close to the optimal solution in a relatively short time. These algorithms include local search, tabu search, simulated annealing, and genetic algorithms. Some, like simulated annealing, are non-deterministic algorithms while others, like tabu search, are deterministic. When a bound on the error of the non-optimal solution is known, the algorithm is further categorized as an approximation algorithm.

Examples

One of the simplest algorithms finds the largest number in a list of numbers of random order. Finding the solution requires looking at every number in the list. From this follows a simple algorithm, which can be described in plain English as:

High-level description:

1. If a set of numbers is empty, then there is no highest number.
2. Assume the first number in the set is the largest.
3. For each remaining number in the set: if this number is greater than the current largest, it becomes the new largest.
4. When there are no unchecked numbers left in the set, consider the current largest number to be the largest in the set.

(Quasi-)formal description: Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

```
Algorithm LargestNumber
Input: A list of numbers L.
```

```
Output: The largest number in the list L.
```

```
if L.size = 0 return null
largest ← L[0]
for each item in L, do
    if item > largest, then
        largest ← item
return largest
```

- "←" denotes assignment. For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

See also

- Abstract machine
- ALGOL
- Algorithm = Logic + Control
- Algorithm aversion
- Algorithm engineering
- Algorithm characterizations
- Algorithmic bias
- Algorithmic composition
- Algorithmic entities
- Algorithmic synthesis
- Algorithmic technique
- Algorithmic topology
- Computational mathematics
- Garbage in, garbage out
- *Introduction to Algorithms* (textbook)
- Government by algorithm
- List of algorithms
- List of algorithm books
- List of algorithm general topics
- Medium is the message
- Regulation of algorithms
- Theory of computation
  - Computability theory
  - Computational complexity theory

Mathematics portal

Computer programming portal

Notes

1. "Definition of ALGORITHM" (https://www.merriam-webster.com/dictionary/algorithm). *Merriam-Webster Online Dictionary*. Archived (https://web.archive.org/web/20200214074446/https://www.merriam-webster.com/dictionary/algorithm) from the original on February 14, 2020. Retrieved November 14, 2019.
2. David A. Grossman, Ophir Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd

edition, 2004, ISBN 1402030045

3. "Any classical mathematical algorithm, for example, can be described in a finite number of English words" (Rogers 1987:2).
4. Well defined concerning the agent that executes the algorithm: "There is a computing agent, usually human, which can react to the instructions and carry out the computations" (Rogers 1987:2).
5. "an algorithm is a procedure for computing a *function* (concerning some chosen notation for integers) ... this limitation (to numerical functions) results in no loss of generality", (Rogers 1977:1).
6. "An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins" (Knuth 1973:5).
7. "A procedure which has all the characteristics of an algorithm except that it possibly lacks finiteness may be called a 'computational method'" (Knuth 1971:5).
8. "An algorithm has one or more outputs, i.e., quantities which have a specified relation to the inputs" (Knuth 1973:5).
9. Whether or not a process with random interior processes (not including the input) is an algorithm is debatable. Rogers opines that: "a computation is carried out in a discrete stepwise fashion, without the use of continuous methods or analog devices ... carried forward deterministically, without resort to random methods or devices, e.g., dice" (Rogers 1987:2).
10. Blair, Ann, Duguid, Paul, Goeing, Anja-Silvia and Grafton, Anthony. Information: A Historical Companion, Princeton: Princeton University Press, 2021. p. 247
11. "algorism" (https://www.oed.com/dictionary/algorism_n?tl=true). *Oxford English Dictionary*. Retrieved May 18, 2025.
12. Chaucer, Geoffrey. "The Miller's Tale" (https://chaucer.fas.harvard.edu/pages/millers-prologue-and-tale). Line 3210.
13. Skeat, Walter William (1914). "agrim, agrum" (https://books.google.com/books?id=z58YAAAAIAAJ&pg=PA5). In Mayhew, Anthony Lawson (ed.). *A Glossary of Tudor and Stuart Words: Especially from the Dramatists*. Clarendon Press. pp. 5–6.
14. Grabiner, Judith V. (December 2013). "The role of mathematics in liberal arts education". In Matthews, Michael R. (ed.). *International Handbook of Research in History, Philosophy and Science Teaching*. Springer. pp. 793–836. doi:10.1007/978-94-007-7654-8_25 (https://doi.org/10.1007%2F978-94-007-7654-8_25). ISBN 9789400776548.
15. "algorithm" (https://www.oed.com/dictionary/algorithm_n). *Oxford English Dictionary*. Retrieved May 18, 2025.
16. Stone (1971), p. 8.
17. Simanowski, Roberto (2018). *The Death Algorithm and Other Digital Dilemmas* (https://books.google.com/books?id=RJV5DwAAQBAJ). Untimely Meditations. Vol. 14. Translated by Chase, Jefferson. Cambridge, Massachusetts: MIT Press. p. 147. ISBN 9780262536370. Archived (https://web.archive.org/web/20191222120705/https://books.google.com/books?id=RJV5DwAAQBAJ) from the original on December 22, 2019. Retrieved May 27, 2019. "[...] the next level of abstraction of central bureaucracy: globally operating algorithms."
18. Dietrich, Eric (1999). "Algorithm". In Wilson, Robert Andrew; Keil, Frank C. (eds.). *The MIT Encyclopedia of the Cognitive Sciences* (https://books.google.com/books?id=-wt1aZrGXLYC). MIT Cognet library. Cambridge, Massachusetts: MIT Press (published 2001). p. 11. ISBN 9780262731447. Retrieved July 22, 2020. "An algorithm is a recipe, method, or technique for doing something."
19. Stone requires that "it must terminate in a finite number of steps" (Stone 1973:7–8).
20. Boolos and Jeffrey 1974, 1999:19
21. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*.

Springer Science & Business Media. pp. 7–8. ISBN 9783642181924.

22. Sriram, M. S. (2005). "Algorithms in Indian Mathematics" (https://books.google.com/books?id=qfJdDwAAQBAJ&pg=PA153). In Emch, Gerard G.; Sridharan, R.; Srinivas, M. D. (eds.). *Contributions to the History of Indian Mathematics*. Springer. p. 153. ISBN 978-93-86279-25-5.
23. Hayashi, T. (2023, January 1). Brahmagupta (https://www.britannica.com/biography/Brahmagupta). Encyclopedia Britannica.
24. Zaslavsky, Claudia (1970). "Mathematics of the Yoruba People and of Their Neighbors in Southern Nigeria" (https://www.jstor.org/stable/3027363). *The Two-Year College Mathematics Journal*. **1** (2): 76–99. doi:10.2307/3027363 (https://doi.org/10.2307%2F3027363). ISSN 0049-4925 (https://search.worldcat.org/issn/0049-4925). JSTOR 3027363 (https://www.jstor.org/stable/3027363).
25. Cooke, Roger L. (2005). *The History of Mathematics: A Brief Course*. John Wiley & Sons. ISBN 978-1-118-46029-0.
26. Chabert, Jean-Luc, ed. (1999). *A History of Algorithms* (https://link.springer.com/book/10.1007/978-3-642-18192-4). doi:10.1007/978-3-642-18192-4 (https://doi.org/10.1007%2F978-3-642-18192-4). ISBN 978-3-540-63369-3.
27. Dooley, John F. (2013). *A Brief History of Cryptology and Cryptographic Algorithms*. Springer Science & Business Media. pp. 12–3. ISBN 9783319016283.
28. Knuth, Donald E. (1972). "Ancient Babylonian Algorithms" (https://web.archive.org/web/20121224100137/http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF). *Commun. ACM*. **15** (7): 671–677. doi:10.1145/361454.361514 (https://doi.org/10.1145%2F361454.361514). ISSN 0001-0782 (https://search.worldcat.org/issn/0001-0782). S2CID 7829945 (https://api.semanticscholar.org/CorpusID:7829945). Archived from the original (http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF) on December 24, 2012.
29. Aaboe, Asger (2001). *Episodes from the Early History of Astronomy*. New York: Springer. pp. 40–62. ISBN 978-0-387-95136-2.
30. Ast, Courtney. "Eratosthenes" (http://www.math.wichita.edu/history/men/eratosthenes.html). Wichita State University: Department of Mathematics and Statistics. Archived (https://web.archive.org/web/20150227150653/http://www.math.wichita.edu/history/men/eratosthenes.html) from the original on February 27, 2015. Retrieved February 27, 2015.
31. Bolter 1984:24
32. Bolter 1984:26
33. Bolter 1984:33–34, 204–206.
34. Bell and Newell diagram 1971:39, cf. Davis 2000
35. Melina Hill, Valley News Correspondent, *A Tinkerer Gets a Place in History*, Valley News West Lebanon NH, Thursday, March 31, 1983, p. 13.
36. Davis 2000:14
37. Kleene 1943 in Davis 1965:274
38. Rosser 1939 in Davis 1965:225
39. Sipser 2006:157
40. Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "The (black) art of run-time evaluation: Are we comparing algorithms or implementations?". *Knowledge and Information Systems*. **52** (2): 341–378. doi:10.1007/s10115-016-1004-2 (https://doi.org/10.1007%2Fs10115-016-1004-2). ISSN 0219-1377 (https://search.worldcat.org/issn/0219-1377). S2CID 40772241 (https://api.semanticscholar.org/CorpusID:40772241).
41. Gillian Conahan (January 2013). "Better Math Makes Faster Data Networks" (http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks). discovermagazine.com. Archived (https://web.archive.org/web/20140513212427/http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks) from the

original on May 13, 2014. Retrieved May 13, 2014.

42. Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, "ACM-SIAM Symposium On Discrete Algorithms (SODA) (http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) Archived (https://web.archive.org/web/20130704180806/http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) July 4, 2013, at the Wayback Machine, Kyoto, January 2012. See also the sFFT Web Page (http://groups.csail.mit.edu/netmit/sFFT/) Archived (https://web.archive.org/web/20120221145740/http://groups.csail.mit.edu/netmit/sFFT/) February 21, 2012, at the Wayback Machine.
43. "Best Case" (https://xlinux.nist.gov/dads/HTML/bestcase.html). *Dictionary of Algorithms and Data Structures*. National Institute of Standards and Technology (NIST). National Institute of Standards and Technology. Retrieved May 29, 2025.
44. "worst case" (https://xlinux.nist.gov/dads/HTML/worstcase.html). *Dictionary of Algorithms and Data Structures*. National Institute of Standards and Technology (NIST). National Institute of Standards and Technology (NIST). Retrieved May 29, 2025.
45. Goodrich, Michael T.; Tamassia, Roberto (2002). *Algorithm Design: Foundations, Analysis, and Internet Examples* (http://ww3.algorithmdesign.net/ch00-front.html). John Wiley & Sons, Inc. ISBN 978-0-471-38365-9. Archived (https://web.archive.org/web/20150428201622/http://ww3.algorithmdesign.net/ch00-front.html) from the original on April 28, 2015. Retrieved June 14, 2018.
46. "Big-O notation (article) | Algorithms" (https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation). *Khan Academy*. Retrieved June 3, 2024.
47. John G. Kemeny and Thomas E. Kurtz 1985 *Back to Basic: The History, Corruption, and Future of the Language*, Addison-Wesley Publishing Company, Inc. Reading, MA, ISBN 0-201-13433-0.
48. Tausworthe 1977:101
49. Tausworthe 1977:142
50. Knuth 1973 section 1.2.1, expanded by Tausworthe 1977 at pages 100ff and Chapter 9.1
51. "The Experts: Does the Patent System Encourage Innovation?" (https://www.wsj.com/articles/SB10001424127887323582904578487200821421958). *The Wall Street Journal*. May 16, 2013. ISSN 0099-9660 (https://search.worldcat.org/issn/0099-9660). Retrieved March 29, 2017.
52. Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). *Knapsack Problems | Hans Kellerer | Springer* (https://www.springer.com/us/book/9783540402862). Springer. doi:10.1007/978-3-540-24777-7 (https://doi.org/10.1007%2F978-3-540-24777-7). ISBN 978-3-540-40286-2. S2CID 28836720 (https://api.semanticscholar.org/CorpusID:28836720). Archived (https://web.archive.org/web/20171018181055/https://www.springer.com/us/book/9783540402862) from the original on October 18, 2017. Retrieved September 19, 2017.
53. Goodrich, Michael T.; Tamassia, Roberto (2001). "5.2 Divide and Conquer". *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons. p. 263. ISBN 9780471383659.
54. Goodrich & Tamassia (2001), p. 245, 4.7.1 Prune-and-search.
55. For instance, the volume of a convex polytope (described using a membership oracle) can be approximated to high accuracy by a randomized polynomial time algorithm, but not by a deterministic one: see Dyer, Martin; Frieze, Alan; Kannan, Ravi (January 1991). "A Random Polynomial-time Algorithm for Approximating the Volume of Convex Bodies". *J. ACM*. **38** (1): 1–17. CiteSeerX 10.1.1.145.4600 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.4600). doi:10.1145/102782.102783 (https://doi.org/10.1145%2F102782.102783). S2CID 13268711 (https://api.semanticscholar.org/CorpusID:13268711).
56. George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.

Bibliography

- Axt, P (1959). "On a Subrecursive Hierarchy and Primitive Recursive Degrees" (https://doi.org/10.2307%2F1993169). *Transactions of the American Mathematical Society*. **92** (1): 85–105. doi:10.2307/1993169 (https://doi.org/10.2307%2F1993169). JSTOR 1993169 (https://www.jstor.org/stable/1993169).
- Bell, C. Gordon and Newell, Allen (1971), *Computer Structures: Readings and Examples*, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.
- Blass, Andreas; Gurevich, Yuri (2003). "Algorithms: A Quest for Absolute Definitions" (http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF). *Bulletin of European Association for Theoretical Computer Science*. **81**. Archived (https://ghostarchive.org/archive/20221009/http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF) from the original on October 9, 2022. Includes a bibliography of 56 references.
- Bolter, David J. (1984). *Turing's Man: Western Culture in the Computer Age* (1984 ed.). Chapel Hill, NC: The University of North Carolina Press. ISBN 978-0-8078-1564-9., ISBN 0-8078-4108-0
- Boolos, George; Jeffrey, Richard (1999) [1974]. *Computability and Logic* (https://archive.org/details/computabilitylog0000bool_r8y9) (4th ed.). Cambridge University Press, London. ISBN 978-0-521-20402-6.: cf. Chapter 3 *Turing machines* where they discuss "certain enumerable sets not effectively (mechanically) enumerable".
- Burgin, Mark (2004). *Super-Recursive Algorithms*. Springer. ISBN 978-0-387-95569-8.
- Campagnolo, M.L., Moore, C., and Costa, J.F. (2000) An analog characterization of the subrecursive functions. In *Proc. of the 4th Conference on Real Numbers and Computers*, Odense University, pp. 91–109
- Church, Alonzo (1936). "An Unsolvable Problem of Elementary Number Theory" (https://archive.org/details/sim_american-journal-of-mathematics_1936-04_58_2/page/344). *American Journal of Mathematics*. **58** (2): 345–363. doi:10.2307/2371045 (https://doi.org/10.2307%2F2371045). JSTOR 2371045 (https://www.jstor.org/stable/2371045). Reprinted in *The Undecidable*, p. 89ff. The first expression of "Church's Thesis". See in particular page 100 (*The Undecidable*) where he defines the notion of "effective calculability" in terms of "an algorithm", and he uses the word "terminates", etc.
- Church, Alonzo (1936). "A Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (1): 40–41. doi:10.2307/2269326 (https://doi.org/10.2307%2F2269326). JSTOR 2269326 (https://www.jstor.org/stable/2269326). S2CID 42323521 (https://api.semanticscholar.org/CorpusID:42323521). Church, Alonzo (1936). "Correction to a Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (3): 101–102. doi:10.2307/2269030 (https://doi.org/10.2307%2F2269030). JSTOR 2269030 (https://www.jstor.org/stable/2269030). S2CID 5557237 (https://api.semanticscholar.org/CorpusID:5557237). Reprinted in *The Undecidable*, p. 110ff. Church shows that the Entscheidungsproblem is unsolvable in about 3 pages of text and 3 pages of footnotes.
- Daffa', Ali Abdullah al- (1977). *The Muslim contribution to mathematics*. London: Croom Helm. ISBN 978-0-85664-464-1.
- Davis, Martin (1965). *The Undecidable: Basic Papers On Undecidable Propositions, Unsolvable Problems and Computable Functions* (https://archive.org/details/undecidablebasic0000davi). New York: Raven Press. ISBN 978-0-486-43228-1. Davis gives commentary before each article. Papers of Gödel, Alonzo Church, Turing, Rosser, Kleene, and Emil Post are included; those cited in the article are listed here by author's name.
- Davis, Martin (2000). *Engines of Logic: Mathematicians and the Origin of the Computer*. New York: W.W. Nortion. ISBN 978-0-393-32229-3. Davis offers concise biographies of Leibniz, Boole, Frege, Cantor, Hilbert, Gödel and Turing with von Neumann as the show

Leibniz, Boole, Frege, Cantor, Hilbert, Gödel and Turing with von Neumann as the show-stealing villain. Very brief bios of Joseph-Marie Jacquard, Babbage, Ada Lovelace, Claude Shannon, Howard Aiken, etc.

- This article incorporates public domain material from Paul E. Black. "algorithm" (https://xlinux.nist.gov/dads/HTML/algorithm.html). *Dictionary of Algorithms and Data Structures*. NIST.
- Dean, Tim (2012). "Evolution and moral diversity" (https://doi.org/10.4148%2Fbiyclc.v7i0.1775). *Baltic International Yearbook of Cognition, Logic and Communication*. **7**. doi:10.4148/biyclc.v7i0.1775 (https://doi.org/10.4148%2Fbiyclc.v7i0.1775).
- Dennett, Daniel (1995). *Darwin's Dangerous Idea* (https://archive.org/details/darwinsdangerous0000denn). New York: Touchstone/Simon & Schuster. pp. 32 (https://archive.org/details/darwinsdangerous0000denn/page/32)–36. ISBN 978-0-684-80290-9.
- Dilson, Jesse (2007). *The Abacus* (https://archive.org/details/abacusworldsfirs0000dils) ((1968, 1994) ed.). St. Martin's Press, NY. ISBN 978-0-312-10409-2., ISBN 0-312-10409-X
- Yuri Gurevich, *Sequential Abstract State Machines Capture Sequential Algorithms* (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.3017&rep=rep1&type=pdf), ACM Transactions on Computational Logic, Vol 1, no 1 (July 2000), pp. 77–111. Includes bibliography of 33 sources.
- van Heijenoort, Jean (2001). *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931* ((1967) ed.). Harvard University Press, Cambridge. ISBN 978-0-674-32449-7., 3rd edition 1976[?], ISBN 0-674-32449-8 (pbk.)
- Hodges, Andrew (1983). *Alan Turing: The Enigma*. New York: Simon and Schuster. ISBN 978-0-671-49207-6., ISBN 0-671-49207-1. Cf. Chapter "The Spirit of Truth" for a history leading to, and a discussion of, his proof.
- Kleene, Stephen C. (1936). "General Recursive Functions of Natural Numbers" (https://web.archive.org/web/20140903092121/http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1). *Mathematische Annalen*. **112** (5): 727–742. doi:10.1007/BF01565439 (https://doi.org/10.1007%2FBF01565439). S2CID 120517999 (https://api.semanticscholar.org/CorpusID:120517999). Archived from the original (http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1) on September 3, 2014. Retrieved September 30, 2013. Presented to the American Mathematical Society, September 1935. Reprinted in *The Undecidable*, p. 237ff. Kleene's definition of "general recursion" (known now as mu-recursion) was used by Church in his 1935 paper *An Unsolvable Problem of Elementary Number Theory* that proved the "decision problem" to be "undecidable" (i.e., a negative result).
- Kleene, Stephen C. (1943). "Recursive Predicates and Quantifiers" (https://doi.org/10.2307%2F1990131). *Transactions of the American Mathematical Society*. **53** (1): 41–73. doi:10.2307/1990131 (https://doi.org/10.2307%2F1990131). JSTOR 1990131 (https://www.jstor.org/stable/1990131). Reprinted in *The Undecidable*, p. 255ff. Kleene refined his definition of "general recursion" and proceeded in his chapter "12. Algorithmic theories" to posit "Thesis I" (p. 274); he would later repeat this thesis (in Kleene 1952:300) and name it "Church's Thesis"(Kleene 1952:317) (i.e., the Church thesis).
- Kleene, Stephen C. (1991) [1952]. *Introduction to Metamathematics* (Tenth ed.). North-Holland Publishing Company. ISBN 978-0-7204-2103-3.
- Knuth, Donald (1997). *Fundamental Algorithms, Third Edition*. Reading, Massachusetts: Addison–Wesley. ISBN 978-0-201-89683-1.
- Knuth, Donald (1969). *Volume 2/Seminumerical Algorithms, The Art of Computer Programming First Edition*. Reading, Massachusetts: Addison–Wesley.
- Kosovsky, N.K. *Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms*, LSU Publ., Leningrad, 1981
- Kowalski, Robert (1979). "Algorithm=Logic+Control" (https://doi.org/10.1145%2F359131.359136). *Communications of the ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://do

136). *Communications of the ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://do
i.org/10.1145%2F359131.359136). S2CID 2509896 (https://api.semanticscholar.org/CorpusI
D:2509896).

- A.A. Markov (1954) *Theory of algorithms*. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954 [i.e., Jerusalem, Israel Program for Scientific Translations, 1961; available from the Office of Technical Services, U.S. Dept. of Commerce, Washington] Description 444 p. 28 cm. Added t.p. in Russian Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algerifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.]
- Minsky, Marvin (1967). *Computation: Finite and Infinite Machines* (https://archive.org/details/computationfinit0000mins) (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 978-0-13-165449-5. Minsky expands his "...idea of an algorithm – an effective procedure..." in chapter 5.1 *Computability, Effective Procedures and Algorithms. Infinite machines.*
- Post, Emil (1936). "Finite Combinatory Processes, Formulation I". *The Journal of Symbolic Logic*. **1** (3): 103–105. doi:10.2307/2269031 (https://doi.org/10.2307%2F2269031). JSTOR 2269031 (https://www.jstor.org/stable/2269031). S2CID 40284503 (https://api.semanticscholar.org/CorpusID:40284503). Reprinted in *The Undecidable*, pp. 289ff. Post defines a simple algorithmic-like process of a man writing marks or erasing marks and going from box to box and eventually halting, as he follows a list of simple instructions. This is cited by Kleene as one source of his "Thesis I", the so-called Church–Turing thesis.
- Rogers, Hartley Jr. (1987). *Theory of Recursive Functions and Effective Computability*. The MIT Press. ISBN 978-0-262-68052-3.
- Rosser, J.B. (1939). "An Informal Exposition of Proofs of Godel's Theorem and Church's Theorem". *Journal of Symbolic Logic*. **4** (2): 53–60. doi:10.2307/2269059 (https://doi.org/10.2307%2F2269059). JSTOR 2269059 (https://www.jstor.org/stable/2269059). S2CID 39499392 (https://api.semanticscholar.org/CorpusID:39499392). Reprinted in *The Undecidable*, p. 223ff. Herein is Rosser's famous definition of "effective method": "...a method each step of which is precisely predetermined and which is certain to produce the answer in a finite number of steps... a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer" (p. 225–226, *The Undecidable*)
- Santos-Lang, Christopher (2015). "Moral Ecology Approaches to Machine Ethics" (http://grinfree.com/MoralEcology.pdf) (PDF). In van Rysewyk, Simon; Pontier, Matthijs (eds.). *Machine Medical Ethics*. Intelligent Systems, Control and Automation: Science and Engineering. Vol. 74. Switzerland: Springer. pp. 111–127. doi:10.1007/978-3-319-08108-3_8 (https://doi.org/10.1007%2F978-3-319-08108-3_8). ISBN 978-3-319-08107-6. Archived (https://ghostarchive.org/archive/20221009/http://grinfree.com/MoralEcology.pdf) (PDF) from the original on October 9, 2022.
- Scott, Michael L. (2009). *Programming Language Pragmatics* (3rd ed.). Morgan Kaufmann Publishers/Elsevier. ISBN 978-0-12-374514-9.
- Sipser, Michael (2006). *Introduction to the Theory of Computation* (https://archive.org/details/introductiontoth00sips). PWS Publishing Company. ISBN 978-0-534-94728-6.
- Sober, Elliott; Wilson, David Sloan (1998). *Unto Others: The Evolution and Psychology of Unselfish Behavior* (https://archive.org/details/untoothersevolut00sobe). Cambridge: Harvard University Press. ISBN 9780674930469.
- Stone, Harold S. (1971). *Introduction to Computer Organization and Data Structures*. McGraw-Hill, New York. ISBN 9780070617261. Cf. in particular the first chapter titled: *Algorithms, Turing Machines, and Programs*. His succinct informal definition: "...any sequence of instructions that can be obeyed by a robot, is called an *algorithm*" (p. 4).
- Tausworthe, Robert C (1977). *Standardized Development of Computer Software Part 1 Methods*. Englewood Cliffs NJ: Prentice–Hall, Inc. ISBN 978-0-13-842195-3.

- Turing, Alan M. (1936–37). "On Computable Numbers, With An Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. Series 2. **42**: 230–265. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712).. Corrections, ibid, vol. 43(1937) pp. 544–546. Reprinted in *The Undecidable*, p. 116ff. Turing's famous paper completed as a Master's dissertation while at King's College Cambridge UK.
- Turing, Alan M. (1939). "Systems of Logic Based on Ordinals". *Proceedings of the London Mathematical Society*. **45**: 161–228. doi:10.1112/plms/s2-45.1.161 (https://doi.org/10.1112%2Fplms%2Fs2-45.1.161). hdl:21.11116/0000-0001-91CE-3 (https://hdl.handle.net/21.11116%2F0000-0001-91CE-3). Reprinted in *The Undecidable*, pp. 155ff. Turing's paper that defined "the oracle" was his PhD thesis while at Princeton.
- United States Patent and Trademark Office (2006), *2106.02 **>Mathematical Algorithms: 2100 Patentability* (http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2106_02.htm), Manual of Patent Examining Procedure (MPEP). Latest revision August 2006
- Zaslavsky, C. (1970). Mathematics of the Yoruba People and of Their Neighbors in Southern Nigeria. The Two-Year College Mathematics Journal, 1(2), 76–99. https://doi.org/10.2307/3027363
- NIST Releases First 3 Finalized Post-Quantum Encryption Standards. https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards

Further reading

- Bellah, Robert Neelly (1985). *Habits of the Heart: Individualism and Commitment in American Life* (https://books.google.com/books?id=XsUojihVZQcC). Berkeley: University of California Press. ISBN 978-0-520-25419-0.
- Berlinski, David (2001). *The Advent of the Algorithm: The 300-Year Journey from an Idea to the Computer* (https://archive.org/details/adventofalgorith0000berl). Harvest Books. ISBN 978-0-15-601391-8.
- Chabert, Jean-Luc (1999). *A History of Algorithms: From the Pebble to the Microchip*. Springer Verlag. ISBN 978-3-540-63369-3.
- Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2009). *Introduction To Algorithms* (3rd ed.). MIT Press. ISBN 978-0-262-03384-8.
- Harel, David; Feldman, Yishai (2004). *Algorithmics: The Spirit of Computing*. Addison-Wesley. ISBN 978-0-321-11784-7.
- Hertzke, Allen D.; McRorie, Chris (1998). "The Concept of Moral Ecology". In Lawler, Peter Augustine; McConkey, Dale (eds.). *Community and Political Thought Today*. Westport, CT: Praeger.
- Jon Kleinberg, Éva Tardos(2006): *Algorithm Design*, Pearson/Addison-Wesley, ISBN 978-0-32129535-4
- Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms (http://www-cs-faculty.stanford.edu/~uno/aa.html) Archived (https://web.archive.org/web/20170701190647/http://www-cs-faculty.stanford.edu/~uno/aa.html) July 1, 2017, at the Wayback Machine*. Stanford, California: Center for the Study of Language and Information.
- Knuth, Donald E. (2010). *Selected Papers on Design of Algorithms (http://www-cs-faculty.stanford.edu/~uno/da.html) Archived (https://web.archive.org/web/20170716225848/http://www-cs-faculty.stanford.edu/~uno/da.html) July 16, 2017, at the Wayback Machine*. Stanford, California: Center for the Study of Language and Information.
- Wallach, Wendell; Allen, Colin (November 2008). *Moral Machines: Teaching Robots Right*

from Wrong. US: Oxford University Press. ISBN 978-0-19-537404-9.

- Bleakley, Chris (2020). *Poems that Solve Puzzles: The History and Science of Algorithms* (https://books.google.com/books?id=3pr5DwAAQBAJ). Oxford University Press. ISBN 978-0-19-885373-2.

External links

- "Algorithm" (https://www.encyclopediaofmath.org/index.php?title=Algorithm). *Encyclopedia of Mathematics*. EMS Press. 2001 [1994].
- Weisstein, Eric W. "Algorithm" (https://mathworld.wolfram.com/Algorithm.html). *MathWorld*.
- Dictionary of Algorithms and Data Structures (https://www.nist.gov/dads/) – National Institute of Standards and Technology

Algorithm repositories

- The Stony Brook Algorithm Repository (http://www.cs.sunysb.edu/~algorith/) – State University of New York at Stony Brook
- Collected Algorithms of the ACM (http://calgo.acm.org/) – Associations for Computing Machinery
- The Stanford GraphBase (http://www-cs-staff.stanford.edu/~knuth/sgb.html) Archived (https://web.archive.org/web/20151206222112/http://www-cs-staff.stanford.edu/%7Eknuth/sgb.html) December 6, 2015, at the Wayback Machine – Stanford University