Algorithm

Introduction

- Stages in solving a discrete mathematics problem:

1. Building an appropriate mathematical model: to translate the problem into mathematical language
2. Finding an appropriate solution method
3. Ideally in the form of a series of steps that lead to solving the problem

- – This is called an ALGORITHM

Origin of the word algorithm

ABU JA'FAR MOHAMMED IBN MUSA AL-KHOWARIZMI (C. 780 – C. 850) al-Khowarizmi, an astronomer and mathematician, was a member of the House of Wisdom, an academy of scientists in Baghdad. The name al-Khowarizmi means "from the town of Kowarzizm," which was then part of Persia, but is now called *Khiva* and is part of Uzbekistan. al-Khowarizmi wrote books on mathematics, astronomy, and geography. Western Europeans first learned about algebra from his works. The word *algebra* comes from al-jabr, part of the title of his book *Kitab al-jabr w'al muquabala*. This book was translated into Latin and was a widely used textbook. His book on the use of Hindu numerals describes procedures for arithmetic operations using these numerals. European authors used a Latin corruption of his name, which later evolved to the word *algorithm*, to describe the subject of arithmetic with Hindu numerals.

Algorithm

What is an algorithm?

Definition. An algorithm is a finite set of precise instructions for performing calculations/computations or solving a problem.

Algorithm

Properties/attributes of the algorithm:

s

Input: the algorithm has input from a certain set,

Output: the algorithm produces output in the form of a certain set (solution),

Definiteness: each calculation step must be precise.

Correctness: produces the correct output for every possible input,

Finiteness: the algorithm must produce output in a finite number of steps, even though there are many calculations,

Effectiveness: each calculation step must be precise and executed in a finite time

Generality: is general for a group of problems.

Contoh algoritma

We will use pseudocode to write the algorithm, which is similar to Pascal language.

Example: algorithm to find the maximum value of a finite series

procedure max(a_1, a_2, …, a_n: integers)
max := a_1
for i := 2 **to** n
    if max < a_i **then** max := a_i
{max is the biggest element}

Algorithm example

Another example: a linear search algorithm, which is an algorithm that searches for a particular element of a finite sequence linearly.

procedure linear_search(x: integer; a_1, a_2, …, a_n:integers)
i := 1
while ($i \leq n$ and $x \neq a_i$)
    i := i + 1

if $i \leq n$ **then** location := i
else location := 0

{the position of the element being searched for is the subscript of the same term as x, or zero if not found.}

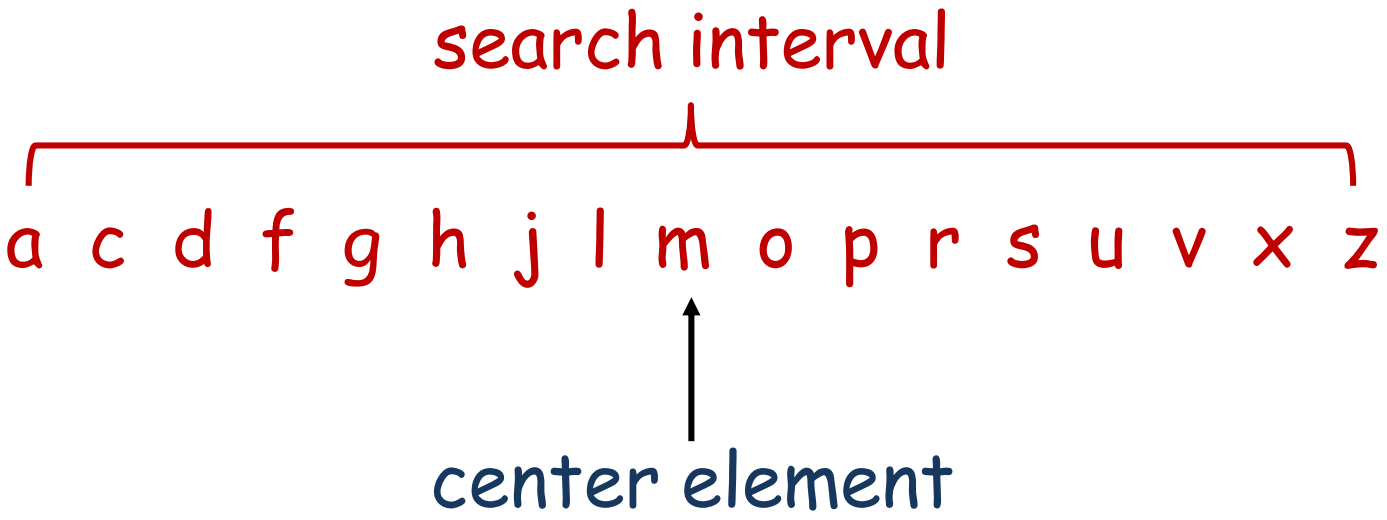
Algorithm Example

If the elements of the sequence are sorted in a specific order (ascending or descending), a binary search algorithm is more efficient than a linear one.

In binary search, the algorithm iteratively limits the relevant search interval to the position of the element being searched.

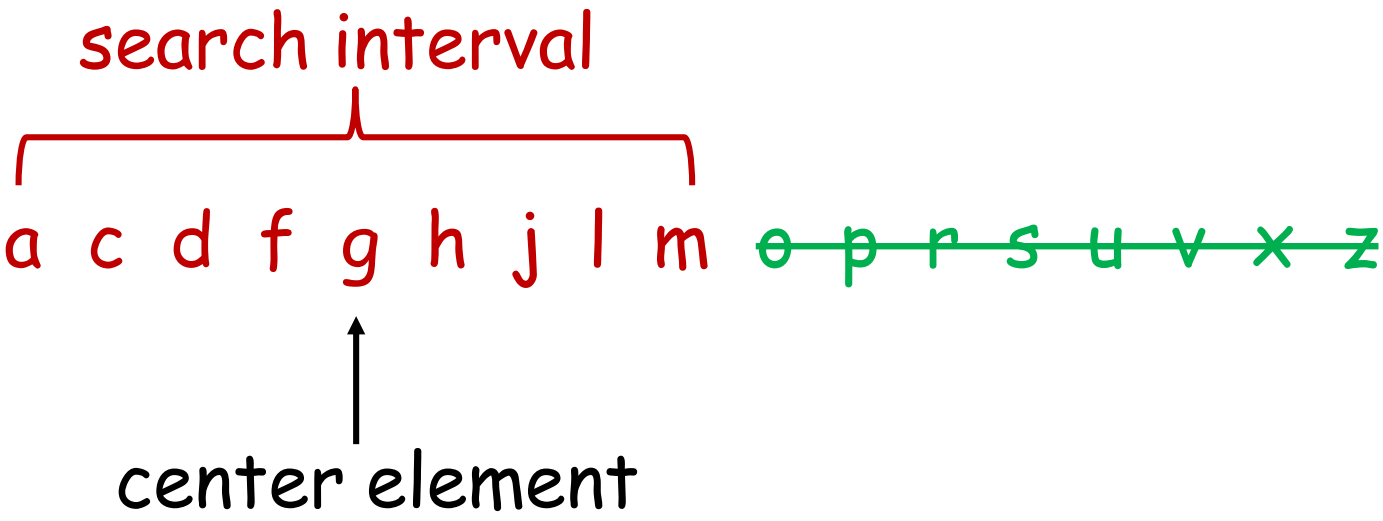
Algorithm Example

binary search of the letter 'j'



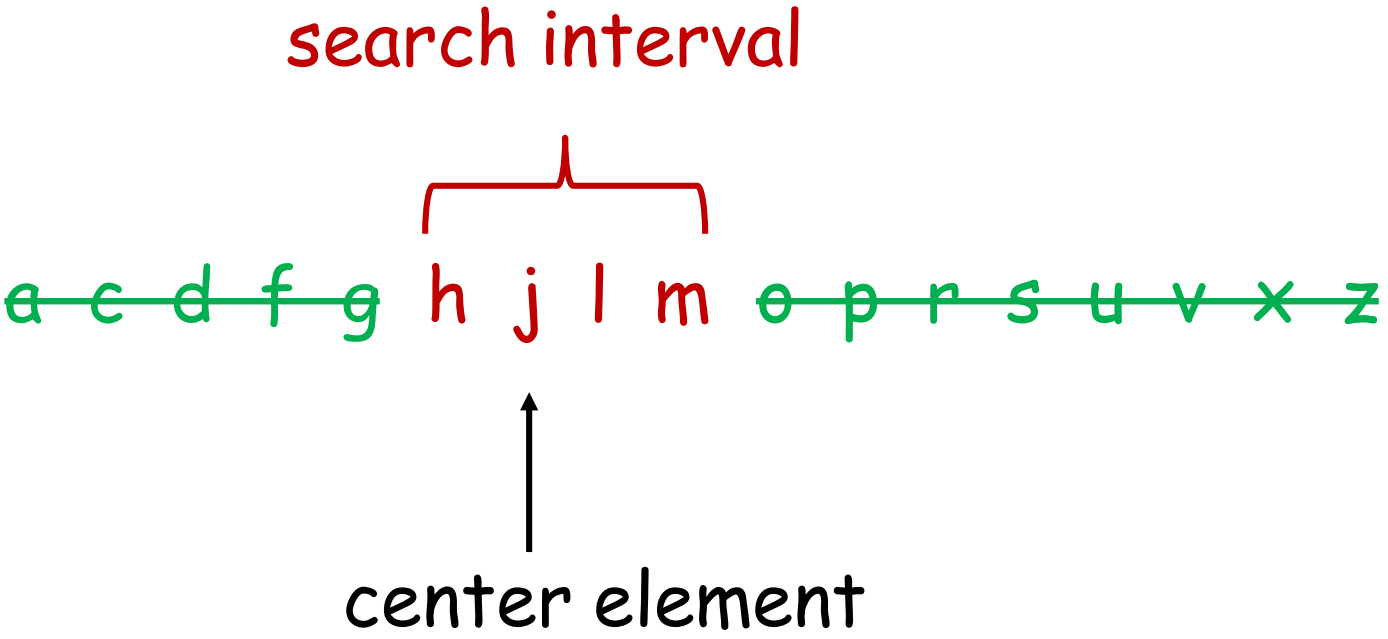
Algorithm Example

binary search of the letter 'j'



Algorithm Example

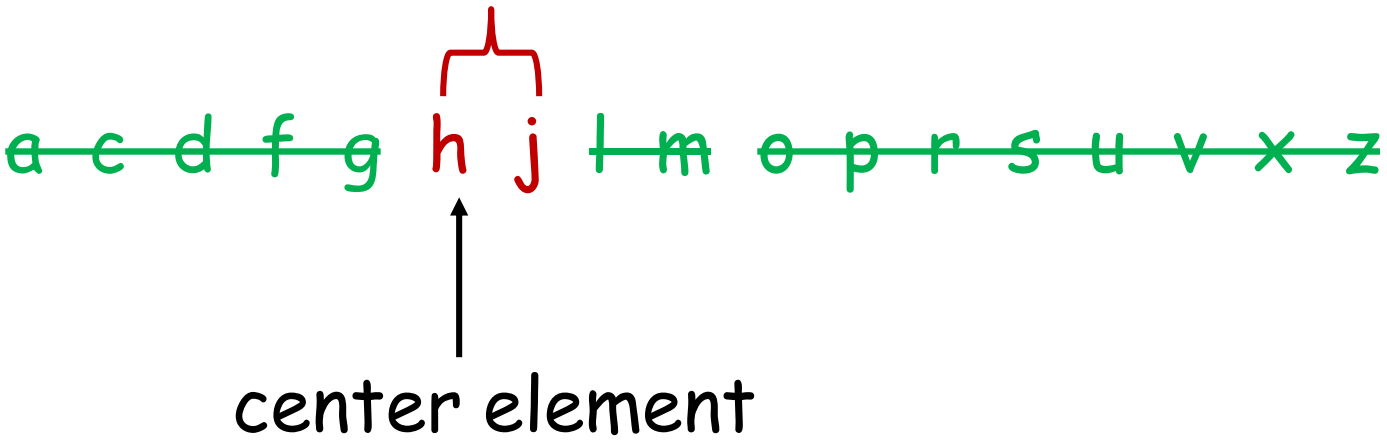
binary search of the letter 'j'



Algorithm Example

binary search of the letter 'j'

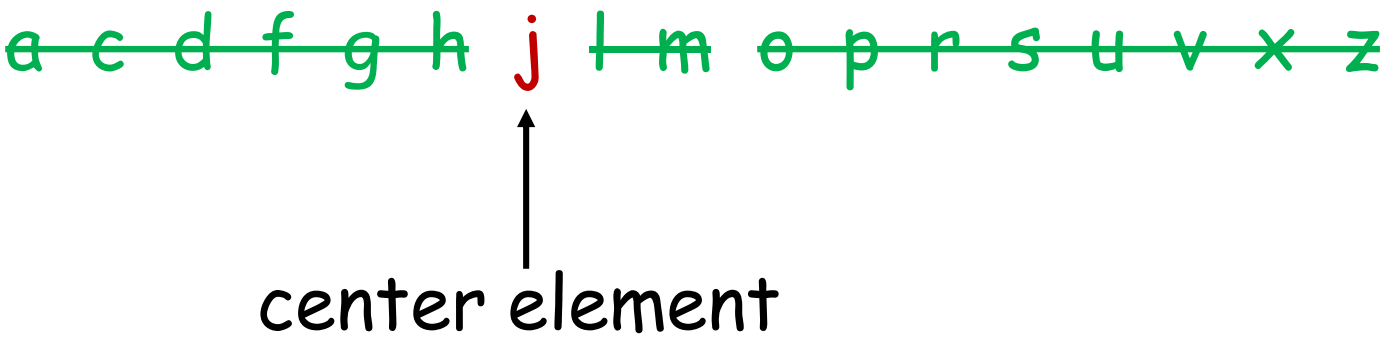
search interval



Algorithm Example

binary search for the letter 'j'

search interval



Found it !

Algorithm Example

procedure binary_search(x: integer;
$a_1, a_2, \ldots, a_n$: integers)
i := 1 {i is left endpoint of search interval}
j := n {j is right endpoint of search interval}
while (i < j)
begin
m := $\lfloor (i + j)/2 \rfloor$
if x > a_m **then** i := m + 1
else j := m
end
if x = a_i **then** location := I
else location := 0
{the location of the searched element is the subscript of the same term as x, or zero if not found.}

Growth functions and measures of complexity

Complexity

It is clear that, in sorted sequences, binary search is more efficient than linear search.

How to analyze the efficiency of an algorithm?

We can measure the

- **time** (number of elementary computations)
- **space** (number of memory cells) required by an algorithm.

These measures are called **computational complexity** and **space complexity**, respectively.

Complexity

What is the time complexity of the linear search algorithm?
We will define the number of worst-case comparisons as a function of the length of the sequence n.
The ***worst case*** of the linear algorithm occurs when the searched element is not in the sequence.
In that case, each item in the sequence is compared with the searched element.

Complexity

For n elements, the loop

while ($i \leq n$ and $x \neq a_i$)

$i=i + 1$

is executed n times, thus requiring 2n comparison processes.

When entering the (n+1)th time, only the comparison $i \leq n$ is executed and the loop is terminated.

Finally, the comparison

if $i \leq n$ then location := i

is executed, thus in the *worst case*, the time complexity is of 2n + 2.

Complexity

What is the time complexity of the binary search algorithm?

Again we will define the number of comparisons in the worst case as a function of the number of terms in the sequence n.

We assume that there are $n = 2^k$ elements in the sequence, which means $k = \log n$.

If n is not a power of 2, the series can be considered as a sub-series (part series) of a larger series, where $2^k < n < 2^{k+1}$.

Complexity

First cycle of the loop

```
while (i < j)
begin
        m := ⌊(i + j)/2⌋
        if x > a_m then i := m + 1
                else j := m
end
```

The search interval is limited to 2^{k-1} elements, using two comparison processes.

Complexity

In the second cycle, the search interval is limited to a number of 2^{k-2} elements, once again with two comparisons.

This process is repeated until there is only one element remaining (2^0) in the search interval.

In this condition, a total of 2k comparisons are performed.

Complexity

And then

while (i < j)

we exit the loop, and the final comparison

if $x = a_i$ **then** location := i

determines whether the searched element has been found.

Thus, the total time complexity for the binary search algorithm is

$$2k + 2 = 2\lceil \log n \rceil + 2.$$

Complexity

In general, for small inputs, we are not interested in either space or time complexity.

The difference in time complexity for linear search versus binary search is not significant for n = 10, but very significant for n = 2^{30}.

Complexity

Suppose, there are two algorithms A and B that can solve a class of problems.

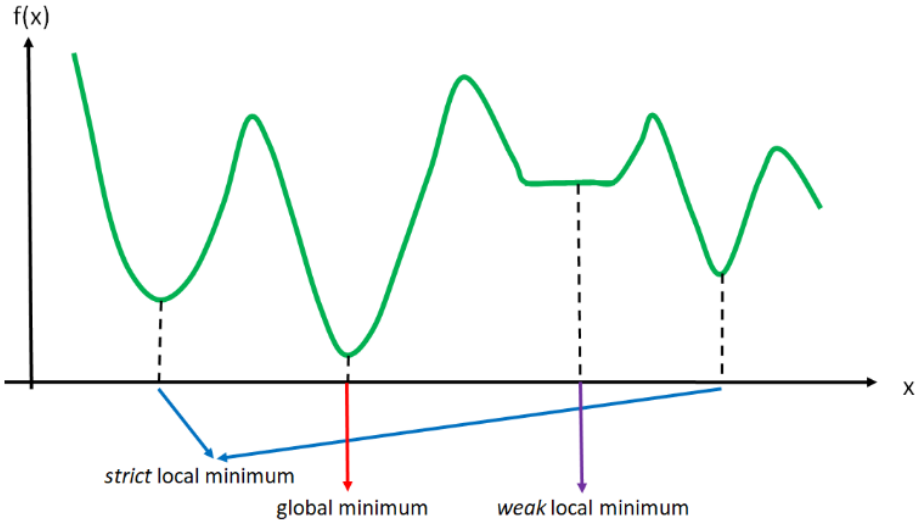
The time complexity of A is 5,000n, while for B it is $\lceil 1.1^n \rceil$ for input with n elements.

Complexity

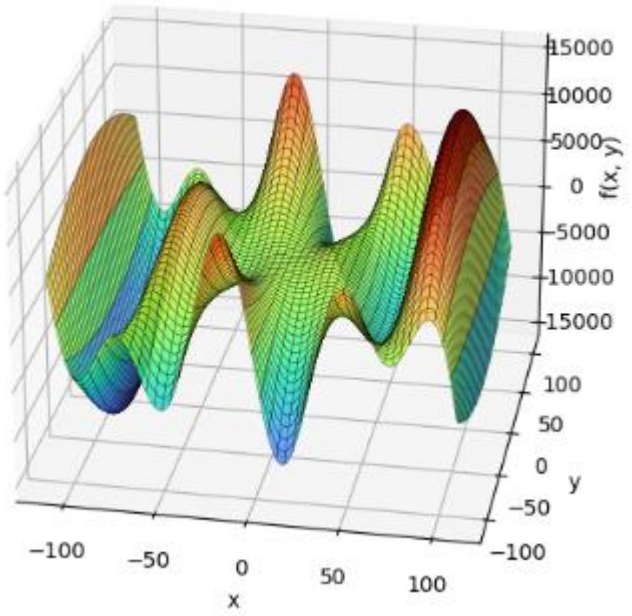
Comparison: Time complexity of algorithms A and B

Numb. of variables	Algorithm A	Algorithm B
0	5,000n	$\lceil 1.1^n \rceil$
10	50,000	3
100	500,000	13,781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

Elective course:
ET4244 : Optimization for Telecommunications



$$f(x, y) = (2x^2 - y^2)\sin(0.09x)$$



only 2 unknown variables

Algorithm	Gradient-descent	Branch and bound
Initial point	(-10, 10)	-
Optimal point	$(-26.3, 5.4 \times 10^{-5})$	(50, 0)
Function value	−967.56	−4887.65
Time complexity	0.0010 sec	almost 2.5 hours

Gradient descent very common in deep learning applications

Complexity

This means that algorithm B cannot be used for input with large elements, while algorithm A still can.

So what is important is the **growth** of the complexity function.

The growth of complexity with increasing input size, n, is a suitable measure for comparing algorithms.

Growth of a function

The growth of a function is denoted by the notation O **(big-O).**

Definition. Let f and g be two functions from integers to real numbers. We say that f(x) is O(g(x)) if there are constants C and k such that

$$|f(x)| \leq C|g(x)|$$

for some x > k. [read as: “f(x) is the big-O of g(x)”]

Growth of a function

When analyzing the complexity of functions, f(x) and g(x) are always positive.

Therefore, we can simplify the big-O requirements to

$f(x) \leq C \cdot g(x)$ for some $x > k$.

If we want to show that **f(x) is O(g(x)),** we only need to determine one pair (C, k) (which is never unique).

Growth of a function

The idea behind big-O notation is to find an **upper bound** on the growth of a function f(x) for large x.

This bound is given by a function g(x), which is usually much simpler than f(x).

We accept the constant C in the condition that $f(x) \leq C.g(x)$ when $x > k$, since C never grows with x.

We are only interested in large x, so it does not matter if $f(x) > C.g(x)$ for $x \leq k$.

Growth of a function

Example:

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

For $x > 1$:

$$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$$
$$\Rightarrow x^2 + 2x + 1 \leq 4x^2$$

Hence, for $C = 4$ and $k = 1$:

$$f(x) \leq Cx^2 \text{ when } x > k.$$

$\Rightarrow f(x)$ is $O(x^2)$.

Growth of a function

Question: If f(x) is $O(x^2)$, is f(x) also $O(x^3)$?

Yes. x^3 grows faster than x^2, hence x^3 also grows faster than f(x).

Therefore, we always have to find the smallest simple function g(x) for which f(x) is O(g(x)).

Growth of a function

Popular functions for g(n) are: n log n, 1, 2^n, n^2, n!, n, n^3, log n

Ordered from the slowest to fastest growth:

- 1
- log n
- n
- n log n
- n^2
- n^3
- 2^n
- n!

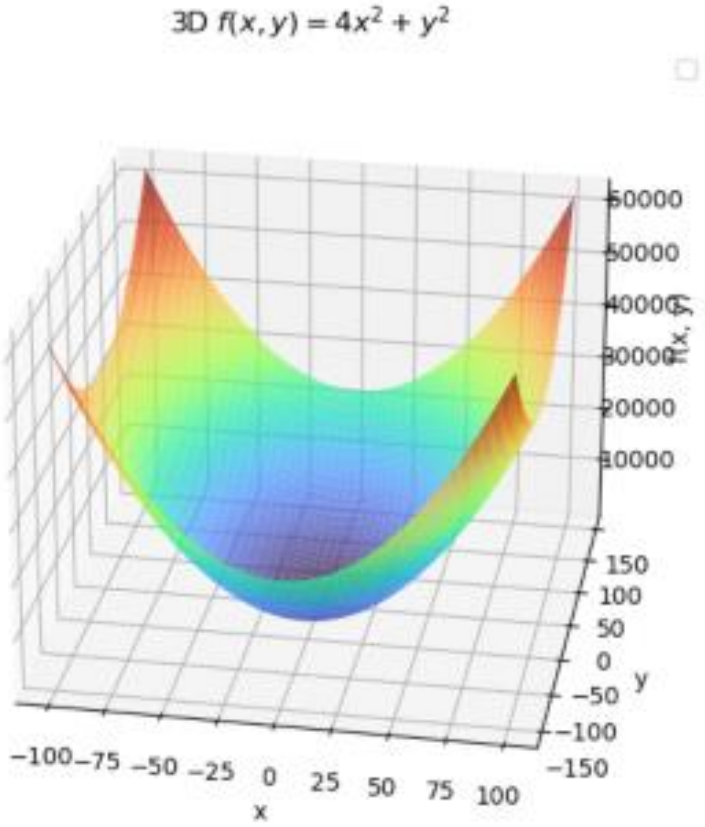
Growth of a function

Problems that can be solved with polynomial worst-case complexity are called **tractable.**

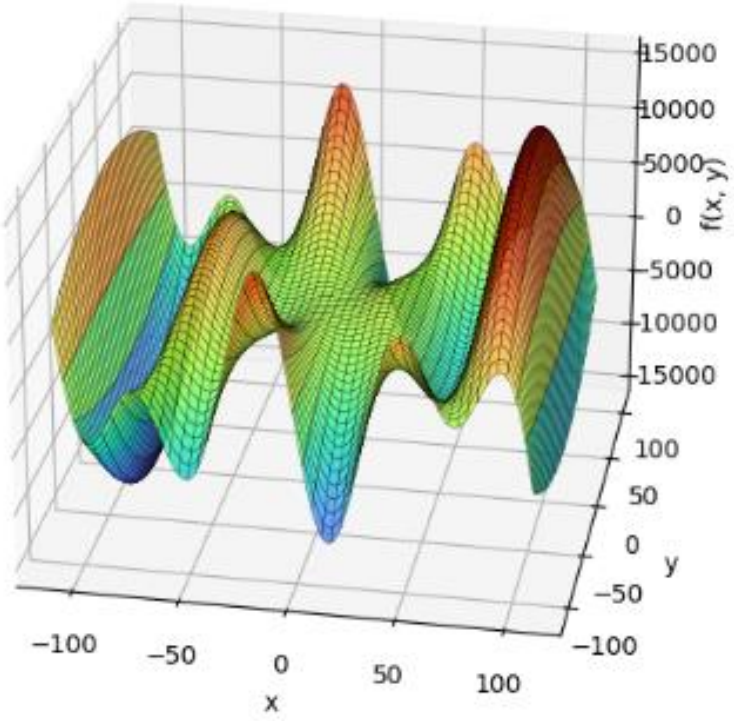
Problems with higher complexity are called **intractable**.

Problems that cannot be solved by any algorithm are called **unsolvable.**

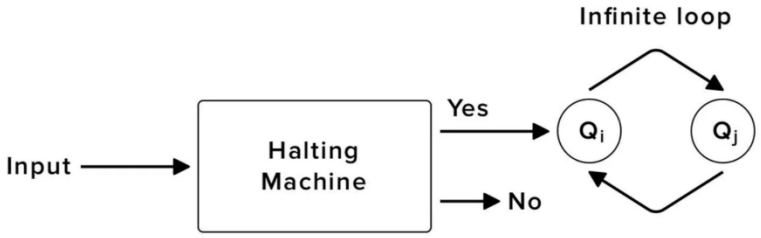
Polynomial time complexity,
e.g. $O(n^3)$ → tractable



Exponential complexity
$O(2^n)$ → intractable



Turing halt machine → unsolvable



Some useful rules for Big-O

Theorem 1. For any **polynomial** $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$, where $a_0, a_1, \ldots, a_n$ real numbers, $f(x)$ is $O(x^n)$.

Theorem 2. If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$

Theorem 3. If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$, then $(f_1 + f_2)(x)$ is $O(g(x))$.

Theorem 4. If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 f_2)(x)$ is $O(g_1(x)\, g_2(x))$.

Example of complexity problems

Analyze the following algorithm and determine its complexity?

procedure who_knows($a_1, a_2, \ldots, a_n$: integers)
m := 0
for i := 1 to n-1
    for j := i + 1 to n
        if $|a_i - a_j| > m$ **then** m := $|a_i - a_j|$
{m is the maximum difference between two numbers from input sequence}

Amount of comparisons: n-1 + n-2 + n-3 + … + 1

$= (n - 1)n/2 = 0.5n^2 - 0.5n$

Complexity is $O(n^2)$.

Example of complexity problems

This algorithm also solves the same problem:

procedure max_diff(a_1, a_2, …, a_n: integers)
min := a_1
max := a_1
for i := 2 to n
    if a_i < min **then** min := a_i
        else if a_i > max **then** max := a_i
    m := max - min

Number of comparisons: 2(n – 1)=2(n - 2)

Time complexity O(n).

Epilog

Mathematician of the day

DONALD E. KNUTH (BORN 1938) Knuth grew up in Milwaukee, where his father taught bookkeeping at a Lutheran high school and owned a small printing business. He was an excellent student, earning academic achievement awards. He applied his intelligence in unconventional ways, winning a contest when he was in the eighth grade by finding over 4500 words that could be formed from the letters in "Ziegler's Giant Bar." This won a television set for his school and a candy bar for everyone in his class.

Knuth had a difficult time choosing physics over music as his major at the Case Institute of Technology. He then switched from physics to mathematics, and in 1960 he received his bachelor of science degree, simultaneously receiving a master of science degree by a special award of the faculty who considered his work outstanding. At Case, he managed the basketball team and applied his talents by constructing a formula for the value of each player. This novel approach was covered by *Newsweek* and by Walter Cronkite on the CBS television network. Knuth began graduate work at the California Institute of Technology in 1960 and received his Ph.D. there in 1963. During this time he worked as a consultant, writing compilers for different computers.

Knuth joined the staff of the California Institute of Technology in 1963, where he remained until 1968, when he took a job as a full professor at Stanford University. He retired as Professor Emeritus in 1992 to concentrate on writing. He is especially interested in updating and completing new volumes of his series *The Art of Computer Programming*, a work that has had a profound influence on the development of computer science, which he began writing as a graduate student in 1962, focusing on compilers. In common jargon, "Knuth," referring to *The Art of Computer Programming*, has come to mean the reference that answers all questions about such topics as data structures and algorithms.

Knuth is the founder of the modern study of computational complexity. He has made fundamental contributions to the subject of compilers. His dissatisfaction with mathematics typography sparked him to invent the now widely used TeX and Metafont systems. TeX has become a standard language for computer typography. Two of the many awards Knuth has received are the 1974 Turing Award and the 1979 National Medal of Technology, awarded to him by President Carter.

Knuth has written for a wide range of professional journals in computer science and in mathematics. However, his first publication, in 1957, when he was a college freshman, was a parody of the metric system called "The Potrzebie Systems of Weights and Measures," which appeared in *MAD Magazine* and has been in reprint several times. He is a church organist, as his father was. He is also a composer of music for the organ. Knuth believes that writing computer programs can be an aesthetic experience, much like writing poetry or composing music.

Knuth pays $2.56 for the first person to find each error in his books and $0.32 for significant suggestions. If you send him a letter with an error (you will need to use regular mail, because he has given up reading e-mail), he will eventually inform you whether you were the first person to tell him about this error. Be prepared for a long wait, because he receives an overwhelming amount of mail. (The author received a letter years after sending an error report to Knuth, noting that this report arrived several months after the first report of this error.)