

What is an Algorithm?

Before we start talking about algorithms for an entire semester, we should try to figure out what an algorithm actually *is*. We could, for example, have a short look at how others define an algorithm:

Several definitions found in the WWW:

Browsing the web, we can find quite number definitions of the term "algorithm". Here's a small collection of them.

Definition 1:

An algorithm is a computable set of steps to achieve a desired result.

(found on numerous websites)

Definition 2:

An algorithm is a set of rules that specify the order and kind of arithmetic operations that are used on specified set of data.

Definition 3:

An algorithm is a sequence of finite number of steps arranged in a specific logical order which, when executed, will produce a correct solution for a specific problem.

(taken from an undergraduate course at [UMBC](#)).

Definition 4:

An algorithm is a set of instructions for solving a problem. When the instructions are followed, it must eventually stop with an answer.

(given by Prof. [Carol Wolf](#) in a [course on algorithms](#)).

Definition 5:

An algorithm is a finite, definite, effective procedure, with some output.

(given by Donald Knuth)

Essential properties of an algorithm

Though the given definitions differ slightly, we can clearly extract the essential properties an algorithm should have:

- an algorithm is finite (w.r.t.: set of instructions, use of resources, time of computation)
- instructions are precise and computable.
- instructions have a specified logical order, however, we can discriminate between
 - deterministic algorithms (every step has a well-defined successor), and
 - non-deterministic algorithms (randomized algorithms, but also parallel algorithms!)
- produce a result.

Basic questions about algorithms:

For each algorithm, especially a new one, we should ask the following basic questions:

- does it terminate?
- is it correct?
- is the result of the algorithm determined?
- how much memory will it use?

Throughout this course, we will be mainly concerned with another important question:

How long will an algorithm have to work to achieve the desired result?

1.1. Example: Fibonacci Numbers

Our first example of an algorithm will be one that computes the Fibonacci series. For repetition:

The sequence $f_j, j \in \mathbb{N}_0$ of the Fibonacci numbers is defined *recursively* as:

- $f_0 := 1$
- $f_1 := 1$
- $f_j := f_{j-1} + f_{j-2}$ for each $j \geq 2$

1.1.1 A recursive algorithm

The definition of the fibonacci numbers can be translated almost directly into a recursive algorithm:

```
Fibo (n:Integer) : Integer {  
    if n=0 then return 1;  
    if n=1 then return 1;  
    if n>1 then return Fibo(n-1) + Fibo(n-2);  
}
```

We take a little time to explain our notation:

the algorithm **Fibo** takes a parameter n of type Integer as its single argument, and returns a result of type Integer. The *recursive calls* $Fibo(n-1)$ and $Fibo(n-2)$ require the execution of our algorithm using the values $n-1$ and $n-2$, respectively, as parameters. In our algorithm, the two results are then added and returned as the result of the call $Fibo(n)$.

Question:

How many arithmetic operations does it take to compute f_j using the algorithm **Fibo**?

That means, we neglect the costs of all operations except the arithmetic ones. Thus, we will basically count the number of additions.

Define:

$TFibo(n)$ = number of arithmetic operations (+, -, *, /) that **Fibo** will perform with n as input parameter.

Examining the function **Fibo**, we see that:

- $TFibo(0) = TFibo(1) = 0$, as there are no additions performed, if the parameter n is 0 or 1.
- $TFibo(n) = TFibo(n-1) + TFibo(n-2)$, if the parameter n is larger than 1. In that case, the number of additions is the sum of the additions performed by the two recursive calls.

Such a recursive characterization of the number of operations is very often found for recursive algorithms. We will soon examine techniques to solve such *recurrence equations*. In the meantime, we try to stick to more basic techniques.

Operation count for Fibo:

We set up a table of the number of additions that are performed by a call to **Fibo**(n):

n	0	1	2	3	4	5	6	7	8
fn	1	1	2	3	5	8	13	21	34
TFibon	0	0	3	6	12	21	36	60	99

Proposition:

After a (very) close look at the values given in this table, we propose that

$$TFibo(n) = 3fn - 3$$

Proof:

We prove this proposition by induction:

- case $n=0$:
from the recurrence equation, we know that $TFibo(0) = 0$,
and the right hand side also evaluates to $3f_0 - 3 = 3 - 3 = 0$

- case $n=1$:
 $TFibo \square 1=0$, and $3f1-3=3-3=0$
- case $n \geq 2$ (induction step):
we assume that $TFibo \square m = 3fm-3$ is correct for all $m < n$ (induction assumption), especially for $m=n-1$, and $m=n-2$. Then:
 $TFibo \square n=3+TFibo \square n-1+TFibo \square n-2=3+(3fn-1-3)+(3fn-2-3)=3(fn-1+fn-2)-3=3fn-3$
(q.e.d.!)

An estimate of the size of $TFibo$ $n=3fn-3$

We will use the following, rather "famous", inequality for the fibonacci numbers,

$$2 \lfloor n/2 \rfloor \leq f_n \leq 2^n$$

to get a more precise estimate of **Fibo's** operation count.

We will prove the two inequalities separately, by induction. In either case, we use that $f_{j+1} \geq f_j$ for all $j \geq 0$ (proof left to the reader....)

Proof for $f_n \leq 2^n$:

By induction over n :

- case $n=0$: then $f_0=1$, which is equal to $2^0=1$
- case $n=1$: then $f_1=1$, which is smaller than $2^1=2$
- case $n \geq 2$ (induction step):
 $f_n=f_{n-1}+f_{n-2} \leq f_{n-1}+f_{n-1}=2f_{n-1} \leq 2 \cdot 2^{n-1}=2^n$

Proof for $f_n \geq 2^{\lfloor n/2 \rfloor}$:

- **case 1:** $n=2k \Rightarrow \lfloor n/2 \rfloor = k$
again, we prove this by induction (over k):
 - for $k=0$, which means that $n=2k=0$, we have $f_0=1$, which is equal to $2^0=1$.
 - for $k \geq 1$ (i.e. $n \geq 2$), we compute that $f_{2k}=f_{2k-1}+f_{2k-2} \geq 2f_{2k-2}=2f_{2(k-1)}$.
By induction assumption $f_{2(k-1)} \geq 2^{k-1}$, and therefore $f_{2k} \geq 2 \cdot 2^{k-1}=2^k$.

- **case 2:** $n=2k+1 \Rightarrow \lfloor n/2 \rfloor = k$
we can reduce this to case 1 by $f_n = f_{2k+1} \geq f_{2k} \geq 2^k$

Result:

Using the above estimate for the fibonacci numbers, we derive the following operation count for the algorithm **Fibo**:

$$3 \cdot 2^{\lfloor n/2 \rfloor} - 3 \leq T_{\text{Fibo}} \leq 3 \cdot 2^{n-3}$$

Hence, the operation count of Fibo *increases exponentially* with the size of the input parameter.

Example

How long would a typical computer take to compute the 100-th fibonacci number using the algorithm **Fibo**?

Answer: $n=100 \Rightarrow T_{\text{Fibo}} \geq 3 \cdot 2^{50} - 3 \geq 10^{15}$, i.e. the computer has to perform more than 10¹⁵ additions.

If we assume that our computer is able to perform one arithmetic operation per nanosecond (compare this to the typical GHz clock rate of current processors), the execution time would be *at least 39 days*!

Exercise:

How large is the respective upper bound for the computing time?

Remark:

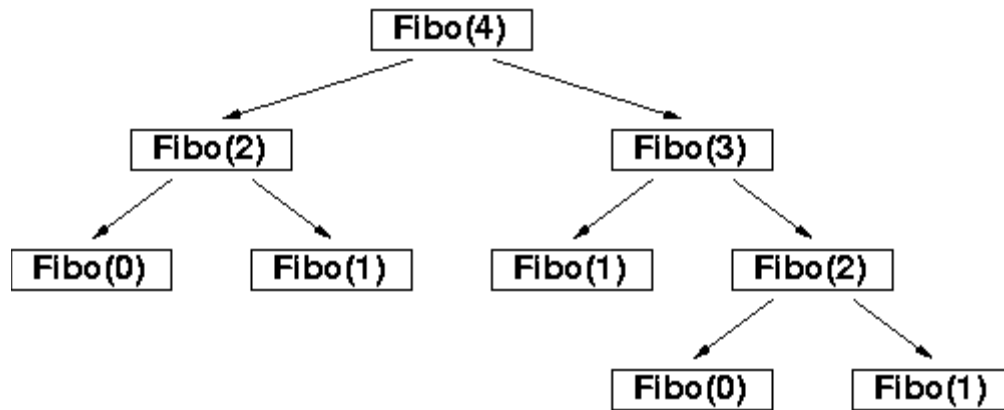
In our estimate of **Fibo**'s operation count, we have used a rather weak lower bound of the size of the fibonacci numbers. A well known algebraic formulation of the fibonacci numbers claims that $f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$.

This leads to a quite accurate estimate of **Fibo**'s operation count:
 $T_{\text{Fibo}} \approx 1021$.

This means, that under the conditions above (1 operation per nanosecond), the computation of the 100-th fibonacci number will take *approximately 31 500 years*, which by far exceeds our initial estimate of "at least 39 days" . . .

1.1.2. An iterative algorithm

To analyse how the recursive algorithm works, we can draw a "tree" of the function calls, for example invoked by a call to Fibo(4):



We can easily see that intermediate results like Fibo(2) or Fibo(1) are computed again (and again and again..). Hence, we should try to store these intermediate results, and reuse them.

Strategy

- Use two variables, `last2`, and `last1`, to store the last two fibonacci numbers f_{n-2} and f_{n-1} , respectively.
- Use one additional variable, `f`, to compute f_n .

Iterative Algorithm

```
Fibit (x : Integer) : Integer {  
  if x < 1 then return 1;  
  else {  
    last2 := 1;  
    last1 := 1;  
    for i from 2 to x do {  
      f := last2 + last1;  
      last2 := last1;  
      last1 := f;  
    }  
  }  
}
```

Question:

How many arithmetic operations does it take to compute the n -th Fibonacci number using algorithm **Fibit**?

Answer

- For $n \leq 1$: 0 operations (return 1 as the result)
- for $n \geq 2$: There is 1 operation per cycle of the for-loop.
The loop is executed $n-1$ times, thus we need $n-1$ operations.

Therefore,

$$TFibit \square n = \begin{cases} 0 & \text{for } n \leq 1 \\ n-1 & \text{for } n \geq 2 \end{cases}$$

We can see that the operation count of Fibit *increases linearly* with the size of the input parameter.

Example

Again, we imagine a computer that performs 1 arithmetic operation per nanosecond. The number of arithmetic operations to compute the 100-th fibonacci number using algorithm **Fibit** is now $TFibit \square 100 \approx 99$. Hence, the computing time will be *only 99 nanoseconds!*

Remark

Please, do not take this example as one to show that recursive programming, on itself, is slow. It's not recursion that makes **Fibo** slow, it's excessive recomputation of intermediate results.

1.2. Random Access Machines

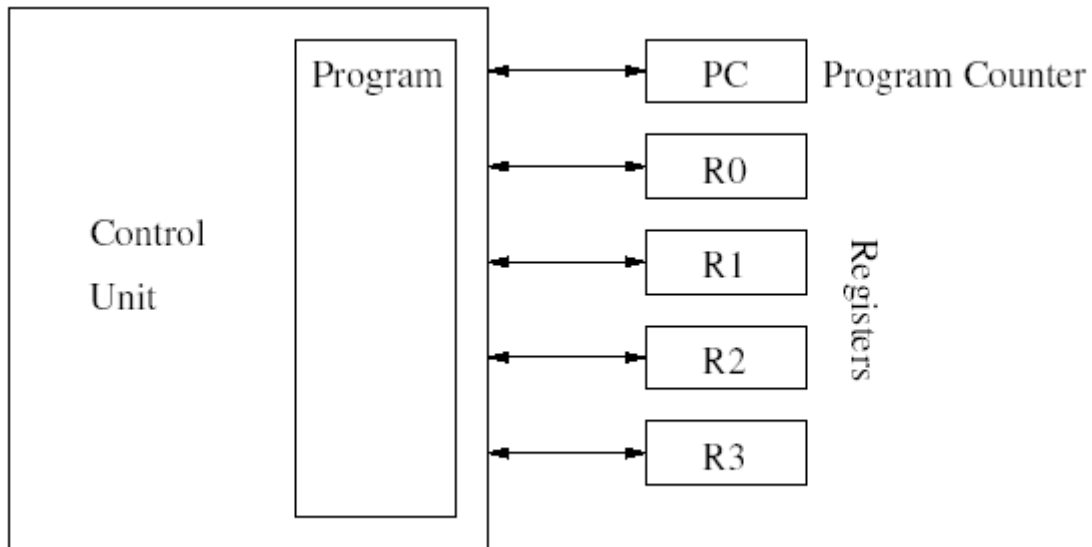
In this chapter, we will try to show a more systematic way of computing the running time of an algorithm than just adding up its arithmetic operations. After all, the time to copy data, execute loops, etc. cannot always be neglected. As the execution time is usually machine dependent, we will try to establish a *reference machine* to objectively compute the working time for algorithms.

1.2.1. Definition of a Random Access Machine

A random access machine (RAM) is a simple **model of computation**. Its memory consists of an unbounded sequence of registers. Each register may hold an integer value.

The **control unit** of a RAM holds a **program**, which consists of a numbered list of statements.

The **program counter** determines which statement is to be executed next.



Rules for executing a RAM-program:

- in each work cycle the RAM executes one statement of the program.
- the program counter specifies the number of the statement that is to be executed.
- the program ends when the program counter takes an invalid value (i.e. when there is no statement in the program that has the specified number).

Execution of a RAM-program:

To "run" a program in the RAM, we therefore need to:

- define the program, i.e. the exact list of statements.
- define starting values for the registers (**the input**).
- define starting values for the program counter (usually, we'll start with the first statement).

Statements of a RAM

Notation:

$\langle Ri \rangle$ $=^{\wedge}$ the integer stored in the i -th register
 $\langle Ri \rangle := x$ $=^{\wedge}$ set the content of the i -th register to the value x

List of Statements:

Statement	Effect on registers	Program Counter
$Ri \bullet Rj$	$\langle Ri \rangle := \langle Rj \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$Ri \bullet RRj$	$\langle Ri \rangle := \langle R\langle Rj \rangle \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$RRi \bullet Rj$	$\langle R\langle Ri \rangle \rangle := \langle Rj \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$Ri \bullet k$	$\langle Ri \rangle := k$	$\langle PC \rangle := \langle PC \rangle + 1$
$Ri \bullet Rj +$ Rk	$\langle Ri \rangle := \langle Rj \rangle +$ $\langle Rk \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$Ri \bullet Rj -$ Rk	$\langle Ri \rangle := \max\{0, \langle Rj \rangle$ $- \langle Rk \rangle\}$	$\langle PC \rangle := \langle PC \rangle + 1$
GOTO m		$\langle PC \rangle := m$
IF $Ri=0$ GOTO m		$\langle PC \rangle := \{ \text{mif } \langle Ri \rangle = 0 \langle PC \rangle$ $+ \text{lotherwise.}$
IF $Ri > 0$ GOTO m		$\langle PC \rangle := \{ \text{mif } \langle Ri \rangle > 0 \langle PC \rangle$ $+ \text{lotherwise.}$

Example: (Multiplication on a RAM)

Basic idea

Using the well known identity $xy = \sum_{i=1}^y x$, we will attempt to multiply x by y by adding up x exactly y times.

A respective algorithm in our previous notation could, for example, look like this:

```

mult (x: Integer, y; Integer) : Integer {
    sum := 0;
    while y > 0 do {
        sum := sum + x;
        y := y - 1;
    }
    return sum;
}

```

The Multiplication RAM

starting configuration of the RAM:

$\langle R0 \rangle := x$, $\langle R1 \rangle := y$, and $\langle Ri \rangle := 0$ for all $i \geq 2$
 $\langle PC \rangle := 0$

desired result: $\langle R0 \rangle := xy$

RAM program:

We will use register $\langle R2 \rangle$ for the variable `sum`. Further, we will need register $\langle R3 \rangle$ to hold the value 1, because we have to subtract 1 from the variable `y` in each loop cycle, and the RAM does not provide operations to subtract numbers, only contents of registers.

- (0) R3 • 1
- (1) IF R1 = 0 GOTO 5
- (2) R2 • R2 + R0
- (3) R1 • R1 - R3
- (4) GOTO 1
- (5) R0 • R2
- (6) STOP

Example:

Let's examine the working steps of our RAM on the special input $x=4$, and $y=3$:

after step	$\langle PC \rangle$	$\langle R0 \rangle$	$\langle R1 \rangle$	$\langle R2 \rangle$	$\langle R3 \rangle$
0 (at start)	0	4	3	0	0
1	1	4	3	0	1

2	2	4	3	0	1
3	3	4	3	4	1
4	4	4	2	4	1
5	1	4	2	4	1

6	2	4	2	4	1
7	3	4	2	8	1
8	4	4	1	8	1
9	1	4	1	8	1

10	2	4	1	8	1
11	3	4	1	12	1
12	4	4	0	12	1
13	1	4	0	12	1

14	5	4	0	12	1
15	6	12	0	12	1
16	stop				

We can see that our RAM performs three (for $y=3$) iterations of its basic loop (statements 1 to 4). Each of the loop iterations requires 4 work cycles of the RAM. The remaining statements require 3 work steps (one before, and two after the loop iterations). Hence, for $y=3$, our RAM performs 15 work steps. For general y , it is easy to see that the number of work cycles is $1+4y+2=3+4y$. Note, that the number of work cycles is independent of x !

1.2.2. Measuring complexity

We will consider vectors of integers as input of a RAM, i.e. $x=(x_1, \dots, x_m)$, where $x_i \in \mathbb{N}$ for $i=1, \dots, m$

The starting configuration of the RAM is given by $\langle R_i \rangle = x_i + 1$ for $i=0, \dots, m-1$, and $\langle R_j \rangle = 0$ for $i \geq m$

Definition: "Uniform size of x "

Let $x=(x_1, \dots, x_m)$ be the input of a RAM, then the **uniform size** of the input is defined as $\|x\|_{\text{uni}} = \text{defm}$.

Definition: "Uniform time complexity"

Let M be a RAM and x be its input. The **uniform time complexity** $T_{\text{Muni}}(x)$ of M on x is then defined as the number of work cycles M performs on x .

Example

The Multiplication RAM has a uniform time complexity of $T_{\text{Muni}}(xy) = 3 + 4y$, independent of x . The uniform size of its input is $\|x\|_{\text{uni}} = 2$, independent of the size of x , and y .

Exercise

Discuss, whether the uniform complexity of a RAM is a good model for measuring complexity of real algorithms in real computers.

Definition: "Logarithmic size of x "

Let $x=(x_1,\dots,x_m)$ be the input of a RAM. Then, the **logarithmic size** of the input x shall be defined as $\|x\|_{\log} = \sum_{i=1}^m \lceil \log_2 x_i \rceil$, where $\lceil z \rceil$ shall be the number of (binary) digits required to represent $z \in \mathbb{N}$.

Remarks:

- for $x \in \mathbb{N}: \lceil \log_2 x \rceil + 1 = \lceil \log_2 (x+1) \rceil$
 idea for proof: with k binary digits 2^k different integers can be represented, which implies that, if $\lceil \log_2 x \rceil = k$, then x is between 2^{k-1} , and 2^k .
- if we use the decimal system (instead of the binary system), its \log_{10} instead of \log_2 (similar for all other systems ...). We will always use the binary system, and write \log instead of \log_2 .

Definition: Logarithmic time complexity

Again, M is a RAM and x is its input. Then, the **logarithmic costs** of a RAM's work cycles are defined as

Statement	Logarithmic Costs
$R_i \bullet R_j$	$\lceil \log_2 R_j \rceil + 1$
$R_i \bullet R R_j$	$\lceil \log_2 R_j \rceil + 1 + \lceil \log_2 \lceil \log_2 R_j \rceil \rceil + 1$
$R R_i \bullet R_j$	$\lceil \log_2 R_j \rceil + 1 + \lceil \log_2 R_i \rceil + 1$
$R_i \bullet k$	$\lceil \log_2 k \rceil + 1$
$R_i \bullet R_j + R_k$	$\lceil \log_2 R_j \rceil + 1 + \lceil \log_2 R_k \rceil + 1$
$R_i \bullet R_j - R_k$	$\lceil \log_2 R_j \rceil + 1 + \lceil \log_2 R_k \rceil + 1$
GOTO m	1
IF $R_i=0$ GOTO m IF $R_i>0$ GOTO m	$\lceil \log_2 R_i \rceil + 1$

The **logarithmic time complexity** $T_{M \log} x$ of M on x is defined as the sum of the logarithmic costs of all working steps M performed on x .

Example:

We compute the logarithmic costs for an execution of the Multiplication-RAM on input (x,y) . We can compute the logarithmic costs by summing up the costs for each loop cycle separately:

for statement (0): 2 for the 0-th loop cycle: $+ly+1+l_0+lx+1+ly+l_1+1+1$ for the 1st loop cycle: $+ly-1+1+lx+lx+1+ly-1+l_1+1+1$ for the i-th loop cycle: $+ly-i+1+l_i \square lx+lx+1+ly-i+l_1+1+1$ for the (y-1)-st (final) loop cycle: $+l_1+1+l(y-1) \square lx+lx+1+l_1+1+1$ for the final IF-statement: $+l_0+1$ for the statement (4): $+ly \square lx+1$

This leads us to the following expression for the logarithmic costs:

$$TM_{\log} \square xy = \sum_{i=0}^{y-1} ly-i+y \square lx + \sum_{i=0}^{y-1} li \square lx + \sum_{i=0}^{y-1} ly-1ly-i+(1+1+l_1+1+1) \square y+4+l_0+lx \square y=5+(4+l_1+lx) \square y+2 \square \sum_{i=1}^y li + \sum_{i=0}^{y-1} 1li \square lx+lx \square y \leq 5+(5+lx) \square y+2 \square \sum_{i=1}^y ly + \sum_{i=0}^{y-1} 1(li+lx)+lx+ly \leq 5+y \square (5+lx)+2 \square y \square ly+y \square ly+y \square lx+lx+ly \leq 5+y \square (5+2 \square lx+3 \square ly)+lx+ly$$

As the logarithmic size of the input, $\| xy \| \log = lx+ly$, we may conclude that $TM_{\log} \square xy \leq 5+y \square (5+3 \square \| xy \| \log) + \| xy \| \log$

Hence, we can get a relation between the size of the input, and the logarithmic costs.

Definition: uniformly, and logarithmically time-bounded

Let $t: \mathbb{N} \rightarrow \mathbb{N}$ be a function, and M be RAM.

1. M is called **uniformly t_n -time-bounded**, if $TM_{uni} \leq t_n$ for all $x: \| x \|_{uni} = n$, i.e. for all inputs of uniform size n, the uniform time complexity has to be bounded by t_n .
2. M is called **logarithmically t_n -time-bounded**, if $TM_{\log} \leq t_n$ for all $x: \| x \|_{\log} = n$, i.e. for all inputs of logarithmic size n, the logarithmic time complexity has to be bounded by t_n .

Remarks:

- $TM_{uni} = \sup \{ TM_{uni} : \| x \|_{uni} = n \}$
- $TM_{\log} = \sup \{ TM_{\log} : \| x \|_{\log} = n \}$

Example: (Multiplication-RAM)

The Multiplication RAM's uniform time complexity, $TM_{uni} \square xy = 4 \square y+3$, is independent of the uniform size of its input, $\| (x,y) \|_{uni} = 2$, which is independent

of x , and y . There is no function t_n such that the size of the input $T_{M_{\text{unix}}, y} = 4 \cdot y + 3 \leq t_n$, because y can become arbitrarily large. Thus, **M is not uniformly t_n -time-bounded** for any function t .

The logarithmic time complexity of the Multiplication RAM is bounded by $T_{M_{\log}}(xy) \leq 5 + y \cdot (5 + 3 \cdot \lceil \log xy \rceil) + \lceil \log xy \rceil$.

If the logarithmic size of the input is $n := \lceil \log xy \rceil = \lceil \log x + \log y \rceil$, then

$$T_{M_{\log}}(xy) \leq 5 + y \cdot (5 + 3 \cdot n) + n \leq 5 + 2 \cdot y \cdot (5 + 3 \cdot n) + n \leq 5 + 2n \cdot (5 + 3 \cdot n) + n$$

which means exactly that M is logarithmically time bounded w.r.t. the function $t_n = 5 + 2n \cdot (5 + 3 \cdot n) + n$. We can therefore say that

"M has an exponential time-complexity w.r.t. the logarithmic complexity measure".

Exercise

Discuss, in which situations the logarithmic time complexity can be a better model for the characterisation of computing time than the uniform time complexity. Consider the following examples:

- sorting a large set of numbers (phone numbers, for example), the size of the numbers is within a fixed range;
- computing the prime factorization of a single, very large integer
- computing a matrix-vector product, or solving a large linear system of equations

Discuss how the word length (i.e. the number of bits a CPU can process in a single step) affects whether uniform or logarithmic complexity is more appropriate.

1.3. Asymptotic Behaviour of Functions

Definition:

Given functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$

$\text{MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2}$

CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaacYcacaWGNbGaaiOoiablwriLkabgkziUkabl2riHoaaCaaaleqabaGaey4kaScaaaaa@3F0A@, then we define

1. $f \in O(g) \Leftrightarrow \text{def } \exists c > 0 \exists n_0 \forall n \geq n_0: f(n) \leq c \cdot g(n)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIGiolaad+eacaGGOaGaam4zaiaacMcacqGHuhY2daWgaaWcbaGaamizaiaadwgac aWGMbaabeaakiabgoGiKiaadogacqGH+aGpcaalWaGaey4alqlaamOBamaaBaaaleaacaalWaaabeaakiabgcGiliaad6gacqGHLjYScaWGUbWaaSbaaSqaaiaaicdaaeqaaOGaaiOoiaadAgacaGGOaGaamOBaiaacMcacqGHKjYOcaWGJbGaeyyXICTaam4zaiaacIcacaWGUbGaaiykaaaa@587C@

2. $f \in \Omega(g) \Leftrightarrow \text{def } \exists c > 0 \exists n_0 \forall n \geq n_0: f(n) \geq c \cdot g(n)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIGiolabfM6axjaacIcacaWGNbGaaiykaiabgsDiBpaaBaaaleaacaWGKbGaamyzaiaadAgaaeqaaOGaey4alqlaam4yaiabg6da+iaaicdacqGHdicjcaWGUbWaaSbaaSqaaiaaicdaaeqaaOGaeyialilaamOBaiabgwMiZkaad6gadaWgaaWcbaGaaGimaaqabaGccaGG6aGaamOzaiaacIcacaWGUbGaaiykaiabgwMiZkaadogacqGHflY1caWGNbGaaiikaiaad6gacaGGPaaaaa@5947@

3. $f \in \Theta(g) \Leftrightarrow \text{def } f \in O(g) \text{ and } f \in \Omega(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIGiolabfI5arjaacIcacaWGNbGaaiykaiabgsDiBpaaBaaaleaacaWGKbGaamyzaiaadAgaaeqaaOGaamOzaiaabgIGiolaad+eacaGGOaGaam4zaiaacMcacaaMb8UaaeiaiaabggacaqGUbGaaeiaiaabccacaWGMbGaeyicI4SaeuyQdCLaaik aiaadEgacaGGPaaaaa@52BE@

4. $f \in o(g) \Leftrightarrow \text{def } \forall c > 0 \exists n_0 \forall n \geq n_0: f(n) \leq c \cdot g(n)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+gacaGGOaGaam4zaiaacMcacqGHuhY2daWgaaWcbaGaamizaiaadwgac aWGMbaabeaakiabgcGiliaadogacqGH+aGpcaalWaGaey4alqlaamOBam aaBaaaleaacaalWaaabeaakiabgcGiliaad6gacqGHLjYScaWGUbWaaSbaa SqaaiaaicdaaeqaaOGaaiOoiaaadAgacaGGOaGaamOBaiaacMcacqGHKj YOcaWGJbGaeyyXICTaam4zaiaacIcacaWGUbGaaiykaaaa@5897@

5. $f \in \omega(g) \Leftrightarrow \text{def } \forall c > 0 \exists n_0 \forall n \geq n_0: f(n) \geq c \cdot g(n)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabeM 8a3jaacIcacaWGNbGaaiykaiabgsDiBpaaBaaaleaacaWGKbGaamyzaiaad AgaaeqaaOGaeyialilaam4yaiabg6da+iaaicdacqGHdicjaWGUbWaaSbaa SqaaiaaicdaaeqaaOGaeyialilaamOBaiabgwMiZkaad6gadaWgaaWcbaGa aGimaaqabaGccaGG6aGaamOzaiaacIcacaWGUbGaaiykaiabgwMiZkaad ogacqGHfIY1caWGNbGaaikaiaad6gacaGGPaaaaa@5981@

Remarks:

1. if $f \in O(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+eacaGGOaGaam4zaiaacMcaaaa@3B75@, then g is called an **asymptotic upper bound** of f ,

2. if $f \in \Omega(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabfM

$6axjaaclcacaWGNbGaaiykaaaa@3C2F@$, then g is called an **asymptotic lower bound** of f ,

3. if $f \in \Theta(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabfI5arjaaclcacaWGNbGaaiykaaaa@3C18@, then g is called an

asymptotically tight bound of f ,

4. if $f \in o(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+gacaGGOaGaam4zaiaacMcaaaa@3B95@, then f is called an

asymptotically smaller than g ;

5. if $f \in \omega(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabeM8a3jaaclcacaWGNbGaaiykaaaa@3C6E@, then f is called an

asymptotically larger than g ;

Further Remarks:

1. in the definition for $f \in o(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+gacaGGOaGaam4zaiaacMcaaaa@3B95@, we can replace the condition $f(n) \leq c \cdot g(n)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzL

$\forall c > 0 \exists n_0 \forall n \geq n_0: f(n)g(n) \leq c$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaWaaSaaaeaacaWGMbGaaiikaiaad6gacaGGPaaabaGaam4zaiaaclcacaWGUbGaaiykaaacqGHKjYOcaWGbbaaaa@3F09@, thus the definition can also be written as

$\forall c > 0 \exists n_0 \forall n \geq n_0: f(n)g(n) \leq 1/c$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaeyialilaam4yaiabg6da+iaaicdacqGHdicjcaWGUbWaaSbaaSqaaiaaicdaaeqaaOGaeyialilaamOBaiabgwMiZkaad6gadaWgaaWcbaGaaGimaaqabaGccaGG6aWaaSaaaeaacaWGMbGaaiikaiaad6gacaGGPaaabaGaam4zaiaaclcacaWGUbGaaiykaaacqGHKjYodaWcaaqaiaaigdaaeaaacaWGbbaaaaaa@4C30@, which is equivalent to $\lim_{n \rightarrow \infty} f(n)g(n) = 0$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaWaaCbeaeaaciGGSbGaiyAaiaac2gaaSqaaiaad6gacqGHsgIRcqGHEisPaeqaaOWaaSaaaeaacaWGMbGaaiikaiaad6gacaGGPaaabaGaam4zaiaaclcacaWGUbGaaiykaaacqGH9aqpcaalWaaaaa@4590@.

2. in the same way, $f \in \omega(g)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaEgaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabglGiolabeM8a3jaaclcacaWGNbGaaiykaaaa@3C6E@ is equivalent to

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaWaaCbeaeaaciGGSbGaaiyAaiaac2gaaSqaiaad6gacqGHsgIRcqGHEisPaeqaaOWaaSaaaeaacaWGNbGaaikaiaad6gacaGGPaaabaGaamOzaiaaclcacaWGUbGaaikykaa acqGH9aqpcaalWaaaaa@4590@

3. from these two observation, we may conclude that $f \in o(g) \Leftrightarrow g \in \omega(f)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIGiolaad+gacaGGOaGaam4zaiaacMcacqGHuhY2caWGNbGaeyicl4SaeqyYdCNaaiikaiaadAgacaGGPaaaaa@4472@

In literature, you will also often find notations like $f = O(g)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabg2da9iaad+eacaGGOaGaam4zaiaacMcaaaa@3AF7@, instead of $f \in O(g)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIGiolaad+eacaGGOaGaam4zaiaacMcaaaa@3B75@.

Example: (Multiplication RAM)

- for the uniform time complexity of the multiplication RAM, we get $TM_{uni}(x,y) = 4y + 3 \Rightarrow TM_{uni}(x,y) \in O(y)$ (use f.e. $c=5, y \geq 2$) and $TM_{uni}(x,y) \in \Omega(y)$ (use f.e. $c=3, y \geq 0$) $\Rightarrow TM_{uni}(x,y) \in \Theta(y)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt

LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaqbaeqabmWaaaqaiaadsfadaqhaaWcbaGaaeytaaqaaiaabwhacaqGUbGaaeyAaaaakiaaclcacaWG4bGaaillaiaadMhacaGGPaGaeyypa0JaaGinaiaadMhacqGHRaWkcaalZaaabaGaeyO0H4TaamivamaaDaaaleaacqGnbaabaGaaeyDaiaab6gacaqGPbaaaOGaaiikaiaadlhacaGGSaGaamyEaiaacMcacqGHilZcaWGpbGaaikaaiaadMhacaGGPaaabaGaaikaiaabwhacaqGZbGaaeyzaiaabccacaqGMbGaaeOlaiabwgacaqGUaGaaeiaiaadogacqGH9aqpcaal1aGaaillaiaadMhacqGHLjYScaalYaGaaikyaaqaaaqaaiaabggacaqGUbGaaeizaiaabccacaWgubWaa0baaSqaaiaab2eaaeaacaqG1bGaaeOBaiaabMgaaaGccaGGOaGaamiEaiaacYcacaWG5bGaaikyaiabglGiolaabfM6axjaaclcacaWG5bGaaikyaaqaaiaaclcacaqG1bGaae4CaiaabwgacaqGgaGaaeOzaiaab6cacaqGLbGaaeOlaiabccacaWGJbGaeyypa0JaaG4maiaacYcacaWG5bGaaeyzlmRaaGimaiaacMcaaeaaeaacaqGHshl3caWGubWaa0baaSqaaiaab2eaaeaacaqG1bGaaeOBaiaabMgaaaGccaGGOaGaamiEaiaacYcacaWG5bGaaikyaiabglGiolaabf15arjaaclcacaWG5bGaaikyaaqaaaaaaaa@93D2@

- for the logarithmic time complexity, we get

$TM \log(n) \leq 5 + (5+3n) \cdot 2n \Rightarrow TM \log(n) \in O(n \cdot 2n)$ or $TM \log(n) \in O(cn)$ for $c > 2$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaqbaeqabiGaaaqaiaadsfadaqhaaWcbaGaaeytaaqaaiaabYgacaqGVbGaae4zaaaakiaaclcacaWGUbGaaikyaiabgsmiJkaaiwdacqGHRaWkcaGGOaGaaGynaiabgUcaRiaaioda caWGUbGaaikyaiabgwSixlaaikdadaahaaWcbeqaaiaad6gaaaaakeaacqGHshl3caWGubWaa0baaSqaaiaab2eaaeaacaqGSbGaae4BaiaabEgaaaGccaGGOaGaamOBaiaacMcacqGHilZcaWGpbGaaikaiaad6gacqGHflY1caalYaWaaWbaaSqabeaacaWGUbbaaaOGaaiykaaqaaaqaaiaab+gacaqGYbGaaeiaiaadsfadaqhaaWcbaGaaeytaaqaaiaabYgacaqGVbGaae4zaaaakiaaclcacaWGUbGaaikyaiabglGiolaad+eacaGGOaGaam4yamaaCaaaleqabaGaamOBaaaakiaacMcacaqGgaGaaeOzaiaab+gacaqGYbGaaeiaiaadoga cqGH+aGpcaalYaaaaaaaa@7152@

Remark:

The definitions for the asymptotical bounds may be used for multidimensional functions $f, g: \mathbb{N}^k \rightarrow \mathbb{R}^+$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbb9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaacYcacaWGNbGaaiOoaiablwriloaaCaaaleqabaGaam4AaaaakiabgkziUkabl2riHoaaCaaaleqabaGaey4kaScaaaaa@4031@, too, if we replace the terms $\forall n > n_0$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbb9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaeyialilaamOBaiabg6da+iaad6gadaWgaaWcbaGaaGimaqabaaaaa@3A91@ by $\forall n = (n_1, \dots, n_k): n_i \geq n_0$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbb9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaeyialilaamOBaiabg2da9iaaclcacaWGUbWaaSbaaSqaai aaigdaaeqaaOGaaiilaiaac6cacaGGUaGaaiOlaiiaacYcacaWGUbWaaSbaaSqaai adUgaaeqaaOGaaiykaiaacQdacaWGUbWaaSbaaSqaaiadMgaaeqaaOGaeyyzl mRaamOBamaaBaaaleaacaalWaaabeaaaaa@47F6@.

Common phrases that denote the growth of a function f :

- f “constant”, if $f \in \Theta(1)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIgiolabf15 arjaaclcacaalXaGaaiykaaaaa@3BE7@

- f “logarithmic”, if $f \in O(\log n)$

MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabgIgiolaad+ eacaGGOaGaciiBaiaac+gacaGGNbGaamOBaiiaacMcaaaa@3E4C@

- f “polylogarithmic”, if $f \in O(\log^k n)$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+eacaGGOaGaciiBaiaac+gacaGGNbWaaWbaaSqabeaacaWGRbaaaOGaamOBaiaacMcaaaa@3F73@
- f “linear”, if $f \in O(n)$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabl5arjaacIcacaWGUbGaaiykaaaa@3C1F@
- f “quadratic”, if $f \in O(n^2)$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+eacaGGOaGaamOBamaaCaaaleqabaGaaGOMaaaakiaacMcaaaa@3C6F@, especially if $f \in O(n^2)$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolabl5arjaacIcacaWGUbWaaWbaaSqabeaacaalYaaaaOGaaiykaaaa@3D12@
- f “polynomial”, if $f \in O(n^k)$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabgIGiolaad+eacaGGOaGaamOBamaaCaaaleqabaGaam4AaaaakiaacMcaaaa@3CA3@ for some $k \in \mathbb{N}$
MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzL

bvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt
LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L
q=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=
xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaam4AaiabGIgiolablwri
Lcaa@39CD@

- f “exponential”, if $f \in O(cn)$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzL
bvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt
LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L
q=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=
xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiabGIgiolaad+
eacaGGOaGaam4yamaaCaaaleqabaGaamOBaaaakiaacMcaaaa@3C9B
@ for some $c \in \mathbb{N}$, or $c \in \mathbb{R}^+$

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzL
bvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt
LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L
q=Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=
xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaam4yaiabGIgiolablwri
LkaabYcacaqGgaGaae4BaiaabkhacaqGgaGaam4yaiabGIgiolabl2riHoaa
CaaaleqabaGaey4kaScaaaaa@428C@

Comparison of functions

The O -, Ω -, Θ -, o -, and ω -

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzLbvyNv2
CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbItLDharqqtubsr4
rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9Lq=Jc9vqaqpepm0xbba
a9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=xb9adbaqaaeGaciGaaiaabe
qaamaabaabaaGcbaGaam4taiabgkHiTiaacYcacqHPoWvcqGHsislcaGGSaGae
uiMdeLaeyOel0laailaiaad+gacqGHsislcaGGSaGaaeiiaiaabggacaqGUbGaaeizai
aabccacqaHjpWDcqGHsislaaa@47EA@ notations define **relations** between
functions. We can therefore ask whether these relations are transitive, reflexive,
symmetric, etc.:

- **transitive:** all of the five relations are transitive!
- **reflexive:** only O , Ω , and Θ

MathType@MTEF@5@5@+=feaafart1ev1aqatCvAUfeBSjuyZL2yd9gzL
bvyNv2CaerbuLwBLnhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt
LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L

$q = Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=$
 $xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaam4taiaacYcacqqHP$
 $oWvcaGGSaGaaeiaabggacaqGUbGaaeiaabccacqqHyoquaaa@3F2$
 $8@$ are reflexive

- **symmetric:** $f \in \Theta(g)$ if and only if $g \in \Theta(f)$

$MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzL$
 $bvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt$
 $LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L$
 $q = Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=$
 $xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabGIGiolabf15$
 $arjaaclcacaWGNbGaaikaiaabccacaqGPbGaaeOzaiaabccacaqGHbGaae$
 $OBaiaabsgacaqGGAaae4Baiaab6gacaqGSbGaaeyEaiaabccacaqGPbG$
 $aaeOzaiaabccacaWGNbGaeyicl4SaeuiMdeLaaiikaiaadAgacaGGPaaaa$
 $@4FA6@$

- **transpose symmetry:**

$f \in O(g)$ if and only if $g \in \Omega(f)$

$MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzL$
 $bvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt$
 $LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L$
 $q = Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=$
 $xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabGIGiolaad+$
 $eacaGGOaGaam4zaiaacMcacaqGGAaaeyAaiaabAgacaqGGAaaeyyai$
 $aab6gacaqGKbGaaeiaaab+gacaqGUbGaaeiBaiaabMhacaqGGAaaeyA$
 $aiaabAgacaqGGAaaam4zaiaabGIGiolabfM6axjaaclcacaWGMbGaaikaiaa$
 $@4F1A@$

$f \in \Omega(g)$ if and only if $g \in O(f)$

$MathType@MTEF@5@5@+=feaafear1ev1aqatCvAUfeBSjuyZL2yd9gzL$
 $bvyNv2CaerbuLwBLNhiov2DGi1BTfMBaeXatLxBI9gBaerbd9wDYLwzYbIt$
 $LDharqqtubsr4rNCHbGeaGqiVu0Je9sqqrpepC0xbbL8F4rqqrFfpeea0xe9L$
 $q = Jc9vqaqpepm0xbba9pwe9Q8fs0=yqaqpepae9pg0FirpepeKkFr0xfr=xfr=$
 $xb9adbaqaaeGaciGaaiaabeqaamaabaabaaGcbaGaamOzaiaabGIGiolabfM$
 $6axjaaclcacaWGNbGaaikaiaabccacaqGPbGaaeOzaiaabccacaqGHbGaa$
 $eOBaiaabsgacaqGGAaae4Baiaab6gacaqGSbGaaeyEaiaabccacaqGPb$
 $GaaeOzaiaabccacaWGNbGaeyicl4Saam4taiaaclcacaWGMbGaaikaiaa$
 $@4F1A@$