Rozdział 1

Mathematical Preliminaries

This chapter is briefly introduction to basic mathematical notations which are very useful for algorithm design, verification and analysis.

1.1 Basic Concepts

1.1.1 Floors and Ceilings

Let $x \in \mathbb{R}^+$. The floor of x ($\lfloor x \rfloor$) is defined to be the largest integer that is no larger than x. The ceiling x ($\lceil x \rceil$) is defined to be the smallest integer that is no smaller than x.

For example $\lceil 1,25 \rceil = 2$ and $\lfloor 1,25 \rfloor = 1$.

The following inequalities are true:

$$\lfloor x \rfloor \leqslant x \leqslant \lceil x \rceil, \tag{1.1}$$

$$\lceil x \rceil - 1 < x < \lfloor x \rfloor + 1. \tag{1.2}$$

1.1.2 Fibonacci Numbers

Fibonacci numbers F_n $(n \geqslant 0)$ are defined as follows:

$$\begin{cases} F_0 = 0 & for \quad n = 0, \\ F_1 = 1 & for \quad n = 1, \\ F_n = F_{n-1} + F_{n-2} & for \quad n \geqslant 2. \end{cases} \tag{1.3}$$

Example of Fibonacci numbers are numbers like that $0, 1, 1, 2, 3, 5, 8, \ldots$.

1.1.3 Binomial Coefficients

The binomial coefficients are defined as follows:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1.4}$$

where $n, k \in \mathbb{N}$ and $0 \leqslant k \leqslant n$.

The binomial coefficient $\binom{n}{k}$ in combinatorial sense means the number of choosing k things form n without replacement.

The following equality is true:

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \tag{1.5}$$

In are shown the values of some of the binomial coefficients The rows of this triangle are labeled ”$n = 0$”, ”$n = 1$”, etc.. This triangle is called ”Pascal’s triangle”.

Looking at the Pascal’s triangle it is easily find some facts about binomial coefficients:

1. $\binom{n}{k} = \binom{n}{n-k}$,

2. $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ $for\ n, k \neq 0$,

3. Sum of elements from each rows is equal to 2^n.

					1					
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1

Tabela 1.1: Pascal's triangle

1.1.4 Manipulations with Sum

The following equality for finite geometrical series is true

$$S_n = 1 + x + x^2 + \ldots + x^n = \frac{1 - x^{n+1}}{1 - x}, \ x \neq 1. \tag{1.6}$$

For $x = 1$ is easy that $S_n = n + 1$. How we can proof that above equality

$$S_n + x^{n+1} = 1 + x + x^2 + \ldots + x^n + x^{n+1} = 1 + x(1 + x + x^2 + \ldots + x^n) = 1 + xS_n$$

$$\Downarrow$$

$$S_n - xS_n = 1 - x^{n+1}$$

$$\Downarrow$$

$$S_n = \frac{1 - x^{n+1}}{1 - x}.$$

For infinite geometrical series is true

$$S = 1 + x + x^2 + \ldots + x^n + \ldots = \frac{1}{1 - x}, \tag{1.7}$$

if only $|x| < 1$. For $|x| \geqslant 1$ this sum not exist.

The following equality for finite arithmetical series is true

$$S_n = x_1 + \ldots + x_n = \frac{x_1 + x_n}{2} * n. \tag{1.8}$$

1.2 Mathematical Induction

Mathematical induction is old(Gauss 1796), simply and very useful proof technic. It is use when we want proof a property for every natural numbers ($n \in \mathbb{N}$). Below is simply scheme of mathematical induction:

1. We should check if our property holds for $n = 1$,
2. Next we should take assumption that property holds for $n, n \geqslant 2$,
3. Finally we should proof that our assumption holds for $n + 1$.

Example of induction:
Property: For all $n \in \mathbb{N}$, if $a + 1 > 1$, then $(a + 1)^n \geqslant 1 + na$.
Check for $n = 1$: It is true that $a + 1 = 1 + a$.
Induction assume: It is true that $(a + 1)^n \geqslant 1 + na$.
Check for $n + 1$: We should proof that $(a + 1)^{n+1} \geqslant 1 + (n + 1)a$.

$$L = (a+1)^{n+1} = (a+1)^n(a+1) \geqslant (1+na)(a+1) = a+1+na^2+na \geqslant$$

$$\geqslant a + 1 + na = P \ \ \textit{because } na^2 > 0.$$

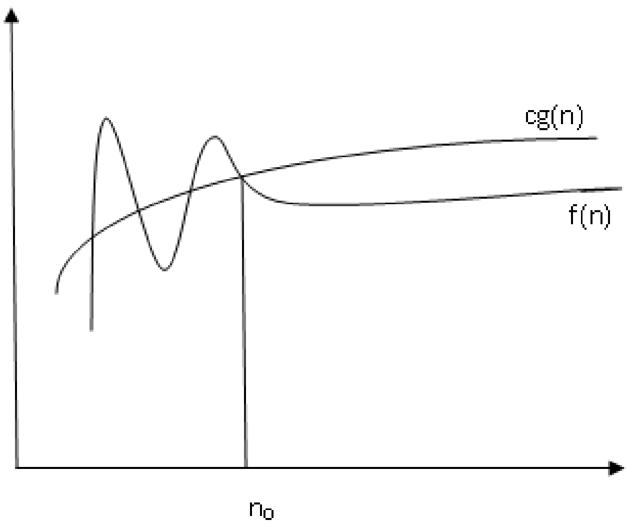
1.3 $O(Oh)$, Θ, Ω - notations

Asymptotic notations (O, Θ, Ω) are very useful for the analysis of algorithms since it gives us possibility to compare algorithms.

$$f(n) = O(g(n)) \Leftrightarrow \underset{c \in \mathbb{R}^+}{\exists} \ \underset{n_0 \in \mathbb{R}^+}{\exists} \ \underset{n \geqslant n_0}{\forall} \ 0 \leqslant f(n) \leqslant cg(n) \tag{1.9}$$

The notation $f(n) = O(g(n))$ means that $f(n)$ is asymptotically smaller than or equal to $g(n)$ (Figure 1.1). Therefore, in an asymptotic sense $g(n)$ is an upper bound for $f(n)$.

$$f(n) = \Omega(g(n)) \Leftrightarrow \underset{c \in \mathbb{R}^+}{\exists} \ \underset{n_0 \in \mathbb{R}^+}{\exists} \ \underset{n \geqslant n_0}{\forall} \ f(n) \geqslant cg(n) \geqslant 0 \tag{1.10}$$



Rysunek 1.1: O–notation

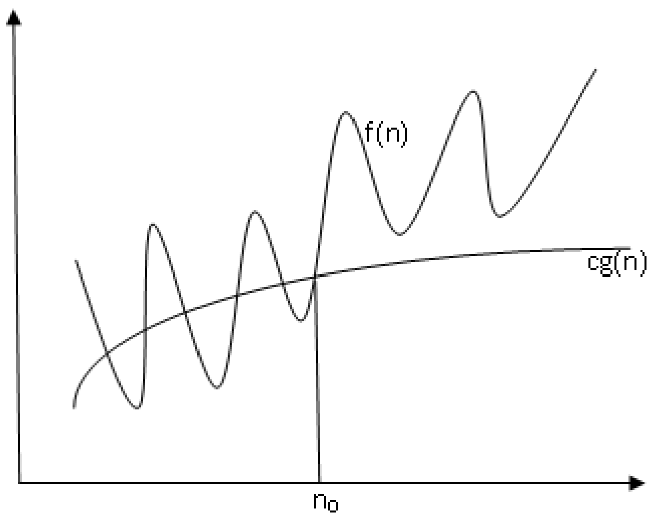
The notation $f(n) = \Omega(g(n))$ means that $f(n)$ is asymptotically bigger than or equal to $g(n)$ (Figure 1.2). Therefore, in an asymptotic sense $g(n)$ is an lower bound for $f(n)$.

$$f(n) = \Theta(g(n)) \Leftrightarrow \underset{c_1,c_2\in\mathbb{R}^+}{\exists}\ \underset{n_0\in\mathbb{R}^+}{\exists}\ \underset{n\geqslant n_0}{\forall}\ c_2g(n) \geqslant f(n) \geqslant c_1g(n) \geqslant 0 \quad (1.11)$$

The notation $f(n) = \Theta(g(n))$ means that $f(n)$ is asymptotically equal to $g(n)$ (Figure 1.3). We can say that $f(n)$ is essentially the same as g(n), to within a constant multiple.

The O-notation has the following property:

1. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(max\{g_1(n), g_2(n)\})$;
2. If $f(n) = ag(n) + b$, then $f(n) = O(g(n))$;
3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$;



Rysunek 1.2: Ω–notation

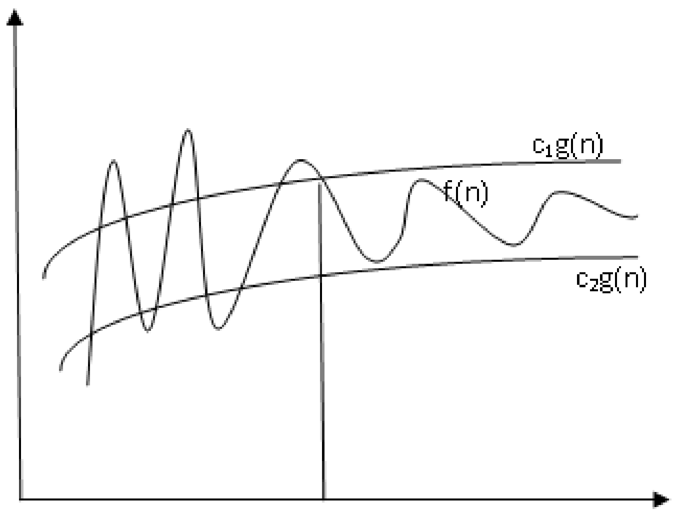
4. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

Example of O–notation:
Proof: If $n > 1$, then $f(n) = 2\sqrt{n} + 1 = O(n + n^2)$.
Claim: It is simply that $n > 1 \Rightarrow n^2 > n \Rightarrow n > \sqrt{n} \Rightarrow 2n > 2\sqrt{n}$.
Then, $2\sqrt{n}+1 < 2n+2 < 2n+2n^2$ because $1 < n^2$. We get that $2\sqrt{n}+1 < 2n + 2n^2 < 2(n + n^2)$. So we find that $c = 2$ and $n_0 = 2$ for O–notation definition.

1.4 Recurrence Equations

Recurrence equations arise frequently in the analysis of algorithms, particularly in the analysis of recursive as well as divide-and-conquer algorithms Shape of recurrence equation:

$$T(n) = \begin{cases} c & for \quad n = n_0, \\ aT(f(n)) + g(n) & for \quad otherwise, \end{cases} \tag{1.12}$$



Rysunek 1.3: Θ–notation

where parameters are mean:
c – running time for base,
n_0 – base of recursion,
a – number of times recursive call is made,
$f(n)$ – size of problem solved by recursive call,
$g(n)$ – all other processing not counting recursive calls.

Techniques for the solution of recurrence equation:

1. The substitution method – it is method for solving recurrences entails two steps: 1) guess the form of the solution, 2) use mathematical induction to find the constants and show that the solution work.

2. The repeated substitution – it is method for solving recurrences entails four steps: 1) substitute a few times until you see a pattern, 2) write a formula in terms of n and the number of substitutions i, 3) choose i so that all references to the base case, 4) solve the resulting summation.

3. The recursion tree – it is kind of substitution method which gives graphical help to guess the solution. In recursion tree, each node represents the cost of single subproblem somewhere in the set of recursive function invocations. We should sum the costs within each level of the tree to obtain a set per–level costs, and then we sum all the pre-level costs to determine the total cost of all levels of the recursion.

4. The master theorem – this theorem gives us recipe how to solve recurrence equation in the specific form and looks as follows:
   if n is a power of c, the solution to the recurrence

$$T(n) = \begin{cases} d & for \quad n \leqslant 1, \\ aT(n/c) + bn & for \quad otherwise, \end{cases} \tag{1.13}$$

   is

$$T(n) = \begin{cases} O(n) & if \quad a < c, \\ O(nlogn) & if \quad a = c, \\ O(n^{log_c a}) & if \quad a > c. \end{cases} \tag{1.14}$$

1.5 Solutions of Recurrence Equations – Examples

Example of substitution method:

$$T(n) = \begin{cases} 0 & for \quad n = 0, \\ 2T(n-1) + 1 & for \quad n \leqslant 1, \end{cases}$$

We guess that $T(n) = 2^n - 1$ is solution. By substitution method we should proof this result by induction on n. Now suppose that the hypothesis is true for n. We are required to proof that $T(n+1) = 2^{n+1} - 1$.
$T(n+1) = 2T(n) + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 1$.

Another solution:

$$\begin{cases} T(0) = 0 & | * 1/2^0 \\ T(n) = 2T(n-1) + 1 & | * 1/2^n \end{cases}$$

$$\begin{cases} T(0) = 0 \\ \frac{T(n)}{2^n} = \frac{T(n-1)}{2^{n-1}} + \frac{1}{2^n} \end{cases}$$

Let $S(n) := \frac{T(n)}{2^n}$. Then, we get

$$\begin{cases} S(0) = 0 \\ S(n) = S(n-1) + \frac{1}{2^n} \end{cases}$$

It is simply now to find solution $S(n) = 1 - \frac{1}{2^n}$. Therefor $T(n) = 2^n - 1$.

Example of repeated substitution:

$$T(n) = \begin{cases} 1 & for \quad n = 1, \\ 2T(n/2) + n & for \quad n > 1, \end{cases}$$

$T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n = 4T(n/4) + n + n =$
$= 4(2T(n/8) + n/4) + n + n = 8T(n/8) + n + n + n.$
After i–substitutions,

$$T(n) = 2^i T(n/2^i) + ni.$$

Now, this identity should be proofed by induction on i.
It is trivially true for $i = 1$.
Now, suppose that $i > 1$ and that

$$T(n) = 2^{i-1} T(n/2^{i-1}) + n(i-1)$$

Then,

$$T(n) = 2^{i-1}(2T(n/2^i) + n/2^{i-1}) + n(i-1) = 2^i T(n/2^i) + n + n(i-1) =$$

$= 2^i T(n/2^i) + ni$ as required.
Now, $i = log_2 n$ could be taken. Then,

$$T(n) = nT(1) + nlog_2 n = n + nlog_2 n.$$

Example of recursion tree:

$$T(n) = \begin{cases} 1 & for \quad n = 1, \\ 2T(n/2) + 1 & for \quad n > 1, \end{cases}$$

The recursion tree for this identity (see Table 1.2).
The solution is

$$S = 1 + 2 + 2^2 + \ldots + 2^{log_2 n - 1} + n = \frac{1 - 2^{log_2 n}}{1 - 2} + n = 2n + 1.$$

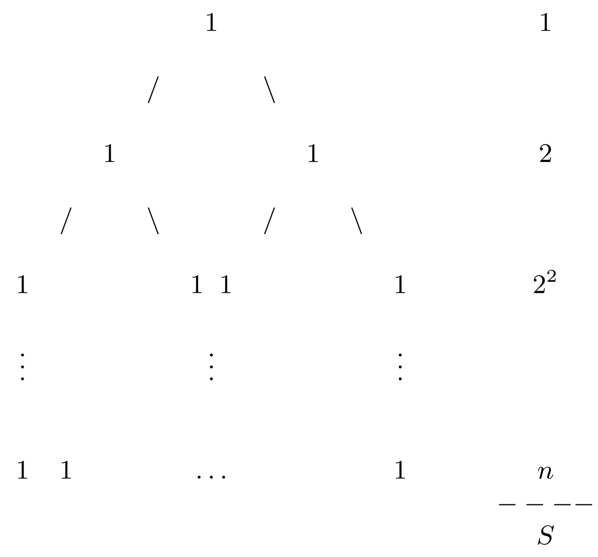


Tabela 1.2: Decision tree

Example of master theorem:

If

$$T(n) = \begin{cases} 1 & for \quad n = 1 \\ 3T(n/2) + 9n & for \quad n > 1, \end{cases}$$

then $T(n) = O(n^{log_2 3})$.

If

$$T(n) = \begin{cases} 1 & for \quad n = 1 \\ 3T(n/3) + 6n - 9 & for \quad n > 1, \end{cases}$$

then $T(n) = O(nlog_2 n)$.

Zadania

Zadanie 1.1 *Udowodnij, że dla wszystkich liczb naturalnych $n \in \mathcal{N}$ poniższa równość jest prawdziwa*

$$1 + 2 + \ldots + n = \frac{n(n+1)}{2}.$$

Zadanie 1.2 *Udowodnij indukcyjnie, że liczba naturalna $n^2 - n$ gdzie $n \in \mathcal{N}$ jest podzielna przez* 2.

Zadanie 1.3 *Udowodnij, że dla wszystkich liczb naturalnych $n \in \mathcal{N}$ poniższa równość jest prawdziwa*

$$1^3 + 2^3 + \ldots + n^3 = (1 + 2 + \ldots + n)^2.$$

Zadanie 1.4 *Udowodnij indukcyjnie, że liczba naturalna $8^n - 1$ gdzie $n \in \mathcal{N}$ jest podzielna przez* 7.

Zadanie 1.5 *Udowodnij indukcyjnie, że następująca nierówność $n! > 2^n$ jest prawdziwa dla $n \geqslant 4$.*

Zadanie 1.6 *Udowodnij indukcyjnie, że następująca nierówność $3^n > n^2 + 5$ jest prawdziwa dla wszystkich liczb naturalnych $n \geqslant 3$.*

Zadanie 1.7 *Udowodnij, że dla wszystkich liczb naturalnych $n \in \mathcal{N}$ poniższa równość jest prawdziwa*

$$1^2 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}.$$

Zadanie 1.8 *Udowodnij indukcyjnie, że następująca nierówność* $13^n - 7$ *gdzie* $n \in \mathcal{N}$ *jest podzielna przez* 6.

Zadanie 1.9 *Udowodnij, że dla wszystkich liczb naturalnych* $n \in \mathcal{N}$, *jeśli* $1 + x > 0$ *to*

$$(1 + x)^n \geqslant 1 + nx.$$

Zadanie 1.10 *Udowodnij, że dla wszystkich liczb naturalnych* $n \in \mathcal{N}$ *następujące równości są prawdziwe:*

a) $\sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n}$ *for* $n \geqslant 1$;

b) $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ *for* $n \geqslant 0$;

c) $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$ *for* $n \geqslant 1$.

Zadanie 1.11 *Udowodnij, że dla* $n \in \mathcal{N}$ *następująca równość jest prawdziwa*

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}.$$

Zadanie 1.12 *Udowodnij indukcyjnie, że dla* $n \geqslant 7$ *prawdziwa jest następująca nierówność* $n! > 3^n$.

Zadanie 1.13 *Udowodnij indukcyjnie, że dla* $n \geqslant 5$ *prawdziwe jest następująca nierówność* $2^n > n^2$.

Zadanie 1.14 *Udowodnij, że dla wszystkich liczb naturalnych* $n \in \mathcal{N}$ *prawdziwa jest następująca nierówność*

$$\sum_{i=1}^{n} i2^i = (n - 1)2^{n+1} + 2.$$

Zadanie 1.15 *Rozwiąż następujące równania rekurencyjne:*

1. $\begin{cases} T(0) = 0 \\ T(n) = 2T(n-1) + 1 \end{cases}$

2. $\begin{cases} T(1) = 8 \\ T(n) = 3T(n-1) - 15 \end{cases}$

3. $\begin{cases} T(1) = 1 \\ T(n) = nT(n-1) + n \end{cases}$

4. $\begin{cases} T(0) = 1 \\ T(n) = 2T(n-1) - 9 \end{cases}$

5. $\begin{cases} T(0) = 0 \\ nT(n) = (n+1)T(n-1) + 2n(n+1) \textit{ dla } n \geqslant 1 \end{cases}$

6. $\begin{cases} T(0) = 0 \\ \frac{1}{2^n}T(n) = 2^{-n+1}T(n-1) + 4^{-n} \textit{ dla } n \geqslant 1 \end{cases}$

7. $\begin{cases} T(0) = 0 \\ \frac{1}{3^n}T(n) = 3^nT(n-1) + 9^{-n} \textit{ dla } n \geqslant 1 \end{cases}$

8. $\begin{cases} T(0) = 0 \\ (n+2)^2T(n) = n^2T(n-1) + \frac{9^n}{n+1} \textit{ dla } n \geqslant 1 \end{cases}$

9. $\begin{cases} T(1) = 3 \\ T(n) = 6T(n/6) + 3n - 1 \end{cases}$

10. $\begin{cases} T(1) = 1 \\ T(n) = 3T(n-1) + 2 \end{cases}$

11. $\begin{cases} T(1) = 1 \\ T(n) = 4T(n/3) + n^2 \end{cases}$

12. $\begin{cases} T(1) = 1 \\ T(n) = 3T(n/2) + n - 2 \end{cases}$

13. $\begin{cases} T(1) = 1 \\ T(n) = 6T(n/6) + 2n + 3 \end{cases}$

14. $\begin{cases} T(1) = 1 \\ T(n) = 2T(n/2) + 6n - 1 \end{cases}$

15. $\begin{cases} T(1) = 2 \\ T(n) = 4T(n/3) + 3n - 5 \end{cases}$

16. $\begin{cases} T(1) = 1 \\ T(n) = 3T(n/2) + n - 2 \end{cases}$

17. $\begin{cases} T(0) = 5 \\ 2T(n) = nT(n-1) + 3n! \end{cases}$

Zadanie 1.16 *Udowodnij następujące tożsamości:*

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$$

Zadanie 1.17 *Oblicz następujące sumy:*

$$\sum_{k=0}^{n} x^k = ?$$

$$\sum_{k=0}^{n} k2^k = ?$$

$$\sum_{k=0}^{n} k^2 = ?$$

$$\sum_{k=0}^{n} k^3 = ?$$

Rozdział 2

Analysis of Algorithms

When we are solving a problem we have to transform this problem to algorithm. It is mean that we should decide if our solution is effective or others propositions are better. So we need some methods which give us possibility to choice among algorithms.

As efficient algorithm is mean algorithm which runs as fast as possible and doesn't use too much internal storage. The cost of algorithm should be optimize.

We can find in [] that the running time of a program depends on such factors as:

- the input data to the program,
- the quality of code generated by the compiler used to create the object program,
- the nature and speed of the instructions on the machine used to execute the program,
- the time complexity of the algorithm underlying the program.

The fact that running time depends on the input data means that the running time of a program should be defined as a function of the input data. Often, the running time depends on the "size" of the input data but sometimes the quality of data is important. A good example is a sorting problem. In this kind of problem is true that the number of objects that will be sort is a main factor in analysis algorithm but the quality like sorted input data may decrease the running time of program.

The functions $T(n)$ are putted to precise the running time of a program on inputs of size n. This time is measured by number of elementary operations done by algorithm. For making this process easier at the begging we can make an assumption that all elementary operations going on one time unit (homogeneous criterion). Sometimes is easier to measure only dominance operations. It is possible if the number of all elementary operations in algorithm is proportional to the number of all dominance operations which was done by algorithm.

The Table 2.1 shows fact that we ought to remember about computational complexity during creation the algorithm. If we have bad (with high complexity) algorithm we will not be able to processing large sized input data.

$T(n)$	Name	Problems
$O(1)$	constant	
$O(logn)$	logarithmic	
$O(n)$	linear	easy-solved
$O(nlogn)$	linear-logarithmic	
$O(n^2)$	quadratic	
$O(n^3)$	cubic	
$O(2^n)$	exponential	
$O(n!)$	factorial	hard-solved

Tabela 2.1: Computational Complexity Functions

Different kind of functions which are determine computational complexity are shows on Table 2.2. The line presented at this table assign strictly partition between algorithmic problems for easily solved problems and hardly solved problems.

How was told that the algorithm could in different way response to quality of input data. In that case we define $T_{pes}(n)$ to be the worst case running time, that is, the maximum, over all inputs of size n, of the running time on that input. The pessimistic running time can be define as follows:

$$T_{pes}(n) = max\{T(d) : d \in D_n\}$$

, where d means input data and D_n means every acceptable inputs of size n for the program.

The average case running time (T_{avg}) is defined as follows:

$$T_{avg}(n) = \sum_{d \in D_n} Pr(d)\dot{T}(d)$$

, where $Pr(d)$ means the probability of appearance input data d among all acceptable inputs.

$T_{avg}(n)$ is the average time taken over all inputs of size n and could be interpret as the expected running time, given some probability distribution on the inputs.

In practice, the average running time is often much harder to determine than the worst-case running time, both because the analysis becomes mathematically difficult as in example[].

The optimistic case running time ($T_{opt}(n)$) is defined as follows:

$$T_{opt}(n) = min\{T(d) : d \in D_n\}$$

.

Example 2.1

Analyze two algorithms: POLYNOMIAL and HORNER for the sake of elementary operations such that " $+$ " and " $*$ ".

INPUT: The polynomial coefficients are stored in an array $A[0 \ldots n]$, with $A[i] = a_i$ for all $0 \leqslant i \leqslant n$.

OUTPUT: $P(x) = a_n x^n + \ldots + a_1 x + a_0$.

Polynomial(A, n, x)

	$T(n)$						
n	n	$n\,log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01 μs	.03 μs	.1 μs	1 μs	10 μs	10 s	1 μs
20	.02 μs	.09 μs	.4 μs	8 μs	160 μs	2.84 h	1 ms
30	.03 μs	.15 μs	.9 μs	27 μs	810 μs	6.83 d	1 s
40	.04 μs	.21 μs	1.6 μs	64 μs	2.56 ms	121 d	18 m
50	.05 μs	.28 μs	2.5 μs	125 μs	6.25 ms	3.1 y	13 d
100	.10 μs	.66 μs	10 μs	1 ms	100 ms	3171 y	$4 * 10^{13}$ y
10^3	1 μs	9.96 μs	1 ms	1 s	16.67 m	3.17*10^{13}y	32*10^{283}y
10^4	10 μs	130 μs	.1 s	16.7 m	115.7 d	3.17*10^{23}y	
10^5	.1 ms	1.66 ms	10 s	11.6 d	3171 y	3.17*10^{33}y	
10^6	1 ms	19.9 ms	16.7 m	31.7 y	3.17*10^7y	3.17*10^{43}y	

μs = microsecond = 10^{-6}seconds; ms = milliseconds = 10^{-3}seconds
s = second; m = minutes; h = hours; d = days; y = years

Tabela 2.2: Comparison of time complexity for different functions and different size of input data

```
$P \leftarrow A[0]$
for $i \leftarrow 1$ to $n$
    do $W \leftarrow x$
        for $j \leftarrow 1$ to $i - 1$
            do $W \leftarrow Wx$
        $P \leftarrow A[i]W + P$
return $P$
```

```
HORNER($A, n, x$)
$V \leftarrow 0$
for $i \leftarrow n$ downto $0$
    do $V \leftarrow A[i] + Vx$
return $V$
```

For POLYNOMIAL algorithm we get the following results:
$L^+ = n$ where L^+ means number of addition operation realized by this

algorithm.
$L^* = n + (1 + 2 + \ldots + n - 1) = n + \frac{n}{2}(n-1) = \frac{2n+n^2-n}{2} = \frac{n+n^2}{2}$, where L^* means number of multiplication operation realized by this algorithm.
In that case $T(n) = \frac{n^2+3n}{2} = O(n^2)$.

For HORNER algorithm we get the following results:
$L^+ = n + 1$ and $L^* = n + 1$. In that case $T(n) = 2n + 2 = O(n)$.

It was shown that schema in second algorithm proposed by Wiliam G. Horner is cost-effective method for polynomial.

2.1 Recursive Algorithms

The analysis of recursive algorithms is a little harder than that of non recursive ones. First, you have to derive a recurrence relation for the running time, and then you have to solve it. Techniques for the solution of recurrence relations was explained in Chapter 1.

The most popular example of recursive algorithm is factorial function because this function has recursive shape like that:

```
FACTORIAL($n$)
1  if $n \leqslant 1$
2     then return 1
3     else  return $n * $FACTORIAL($n-1$)
```

Zadania

Zadanie 2.1 *Przeanalizuj poniższy algorytm*

```
SEQUENTIALSEARCH($A, n, x$)
1  for ($i \leftarrow 0; i < n$ && $x! = A[i]); i = i + 1$)
2      do if $i = n$
3            then return $-1$
4  return 1
```

Zadanie 2.2 *Dokonaj analizy czasu działania poniższego algorytmu, gdzie z jest n-bitową liczbą naturalną:*

```
MULTIPLY($y, z$)
1  if $z = 0$
2     then return 0
3     else  if  $z$ is odd
4              then return MULTIPLY($2y, \lfloor z/2 \rfloor$) $+ y$
5              else  return MULTIPLY($2y, \lfloor z/2 \rfloor$)
```

Zadanie 2.3 *Przeanalizuj poniższy algorytm sortujący, gdzie L jest n-elementową listą (zakładamy, że liczba n jest potęgą 2).*

```
MERGESORT($L, n$)
1  if $n \leqslant 1$
2     then return $L$
3     else  return $Merge$(MERGESORT($L_1, n/2$), MERGESORT($L_2, n/2$))
```

Procedura Merge łączy dwie posortowane listy L_1 i L_2 w jedną posortowaną listę w czasie liniowym.

Zadanie 2.4 *Przeanalizuj algorytm wyznaczający liczby Fibonacciego:*

```
FIBONACCI($n$)
1  if $n \leqslant 1$
2     then return $n$
3     else  return FIBONACCI($n-1$) + FIBONACCI($n-2$)
```

Dodatek: *A recurrence relation of the linear form $a_n = Aa_{n-1} + Ba_{n-2}$. It could be guess that the solution has the form $a_n = r^n$. In that case $r^n = Ar^{n-1} + Br^{n-2}$ must be true for all $n > 1$. Dividing through by r^{n-2}, we get that all these equations reduce to the same thing:*

$$r^2 = Ar + B,$$

$$r^2 - Ar - B = 0.$$

Solve for r to obtain the two roots λ_1, λ_2. Different solutions are obtained depending on the nature of the roots:

- *if these roots are distinct, we have the general solution*

$$a_n = C\lambda_1^n + D\lambda_2^n,$$

- *if these roots are identical (when $A^2 + 4B = 0$), we have*

$$a_n = C\lambda^n + Dn\lambda^n.$$

This is the most general solution, the two constants C and D can be chosen freely to produce a solution. If "initial conditions" $a_0 = a$, $a_1 = b$ have been given then we can solve (uniquely) for C and D.

Zadanie 2.5 *Przeanalizuj następujący algorytm ze względu na operację dodawania i mnożenia (linia 6 i linia 8)*

```
MULTIPLICATION($m, n$)
1  $MULT \leftarrow 0$
2  $N \leftarrow n$
3  $s \leftarrow m$
4  while $N > 0$
5      do if $odd(N)$
6             then $MULT \leftarrow MULT + s$
7          $N \leftarrow N div 2$
8          $s \leftarrow 2 * s$
9  return $MULT$ {*MULT=nm*}
```

Uwaga: *Zamiast analizowania wszystkich kombinacji* 0 *i* 1 *proponujemy przeprowadzenie obliczeń na następującym wzorcu:*

m	...	*k+1*	*k*	*k-1*	...	*1*
0	...	*0*	*1*	*0*	...	*1*

Wzorzec ten to wektor binarny, który zawiera 1 *na* k*-tej pozycji,* 0 *na pozycjach powyżej pozycji* k *oraz wszystkie możliwe kombinacje* 0 *i* 1 *na pozycjach od* 1 *do* $k-1$.

Zadanie 2.6 *Jaką wartość zwraca funkcja MYSTERY. Wyraź odpowiedź jako funkcję zależną od* n*. Podaj używając* O*-notacji, pesymistyczny czas działania.*

```
MYSTERY($n$)
r ← 0
for $i \leftarrow 1$ to $n-1$
    do for $j \leftarrow i+1$ to $n$
            do for $k \leftarrow 1$ to $j$
                    do $r \leftarrow r+1$
return $r$
```

Zadanie 2.7 *Podaj pesymistyczny i średni czas działania ponizszego algorytmu, gdzie* $A[1..n]$ *to tablica* n*-elementowa.*

```
MINMAX($A$)
$j \leftarrow 1$
$k \leftarrow 1$
for $i \leftarrow 2$ to $n$
    do if $A[i] > A[j]$
          then $j \leftarrow i$
          else if $A[i] < A[k]$
                  then $k \leftarrow i$
return $(j, k)$
```

Dodatek: *Dla ułatwienia analizy algorytmu można wykorzystać pojęcie tablicy inwersji.*

Dla każdej permutacji elementów $A[1], A[2], \ldots, A[n]$ *można wskazać w sposób jednoznaczny tablicę inwersji* $B[1], B[2], \ldots, B[n]$ *postaci:*

$$B[i] = |\{j : 1 \leqslant j < iandA[j] > A[i]\}|.$$

Na przykład dla tablicy $A[1..6] = [3\ 6\ 8\ 2\ 10\ 9]$ *tablica inwersji* $B[1..6]$ *wygląda w następujący sposób* $B[1..6] = [0\ 0\ 0\ 3\ 0\ 1]$.

Zadanie 2.8 *Wskaż pesymistyczny i średni czas działania algorytmu BubbleSort(A) ze względu na operację dominującą jaką jest porównanie kluczy znajdujących się w tablicy* $A[1..n]$.

```
BubbleSort($A$)
1 for $i \leftarrow 1$ to $n$
2    do for $j \leftarrow 2$ to $n$
3          do if $A[j-1] > A[j]$
4               then $swap(A[j-1], A[j])$
5 return $A$
```

Zadanie 2.9 *Przeanalizuj poniższy algorytm BubbleSort(n) ze względu na operację dominującą jaką jest porównanie kluczy znajdujących się w tablicy* $A[1..n]$.

```
BubbleSort($n$)
1 if $n > 1$
2   then for $i \leftarrow 1$ to $n-1$
3             do if $A[i] > A[i+1]$
4                  then $swap(A[i], A[i+1])$
5          BubbleSort($n-1$)
```

Zadanie 2.10 *Udowodnij indukcyjnie, że* $3^n - 2^n$ *dla wszystkich* $n \geqslant 0$ *reprezentuje czas działania algorytmu* $T(n)$.

```
T($n$)
1 if $n \leqslant 1$
2   then return $n$
3   else return $5T(n-1) - 6T(n-2)$
```

Zadanie 2.11 *Przeanalizuj algorytm dzielenia całkowitego ze względu na elementarne operacje "$+$" oraz "$-$". Ten algorytm zwraca $q, r \in \mathbf{N}$ takie, że $x = qy + r$ i $r < y$, gdzie $x, y \in \mathbf{N}$.*

```
DIVIDE($x, y$)
$r \leftarrow x$
$q \leftarrow 0$
$w \leftarrow y$
while $w \leqslant x$
    do $w \leftarrow 2 * w$
while $w > y$
    do $q \leftarrow 2 * q$
       $w \leftarrow w div 2$
       if $w \leqslant r$
          then $r \leftarrow r - w$
               $q \leftarrow q + 1$
return $(q, r)$
```

Zadanie 2.12 *Zapisz równanie rekurencyjne dla czasu działania algorytmu sortowania przez prosty wybór. Rozwiąż zaproponowane równanie.*

Zadanie 2.13 *Udowodnij, że algorytm rozwiązujący problem wież Hanoi musi wykonać $2^n - 1$ kroków.*