Analysis of algorithms. Complexity of Algorithms.

Complexity of algorithm measures how „fast" is the algorithm (time complexity) and what „amount" of memory it uses (space complexity). Time and memory are 2 basic resources in computations. In this classes we will focus only on time complexity of algorithms.

Time Complexity of Algorithm is the number of dominating operations executed by the algorithm as the function of data size.

The number dominating operation can be define in various way. It's strictly depends on algorithm. For example it could be the number of compares of two values in sorting algorithm, the number of multiplies in algorithm that calculate the factorial, the number of divisions in algorithm that check whether given number is prime, sum of all arithmetic operations performes by particular algorithm and so one..

We will focus on pessimistic Time Complexity of algorithm which is the greatest number of dominating operations of all possible inputs datasets of size n. So in the other words it is the worst time in which algorithm can run.

For example calculating factorial takes n operation. So its time complexity is equal to n.

Here are the time complexities in order of growth from least to greatest that you're most likely to see:

— 1 - constant function,
— $\log_2 n$ - logarithmic,
— n - linear,
— $n \log_2 n$ - linear-logarithmic,
— n^2 - square,
— n^k - polynomial ($k > 2$),
— 2^n - exponential,
— $n!$ - factorial

There's very good summary in `http://www.cc.gatech.edu/~lebanon/notes/algs.pdf` (You can omit the part (at the ending) abour recurrence functions because we will not analyse complexity of recurrence functions).

Big Omega and Big Theta Notations

We will use three notations of complexity functions:

The function $g(n)$ is **the upper bound of rank of order** of the function $f(n)$:

$$f(n) = O(g(n)) \Leftrightarrow \exists_{c>0} \exists_{n_0} \forall_{n \geqslant n_0} \quad |f(n)| \leqslant c \cdot |g(n)|$$

We can also say that $|f(n)| \leqslant c \cdot |g(n)|$ holds for almost every $n \in \mathbb{N}$. Set of function for which $g(n)$ is the upper bound of rank of order is denoted by $O(g(n))$.

The function $g(n)$ is **the lower bound of rank of order** of the function $f(n)$:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists_{c>0} \exists_{n_0} \forall_{n \geqslant n_0} \quad c \cdot |g(n)| \leqslant |f(n)|$$

We can also say that $c \cdot |g(n)| \leqslant |f(n)|$ holds for almost every $n \in \mathbb{N}$. Set of function for which $g(n)$ is the lower bound of rank of order is denoted by $\Omega(g(n))$.

The function $f(n)$ has **the same rank of order** as the function $g(n)$:

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge g(n) = O(f(n))$$

In the other words $\exists c_0, c_1 > 0$, $\exists n_0 \in \mathbb{N}$, such that $c_0 \cdot |g(n)| \leqslant |f(n)| \leqslant c_1 \cdot |g(n)|$ for all $n \geqslant n_0$.

Set of function for which $g(n)$ has the same rank of order is denoted by $\Theta(g(n))$.

In respect to early mention functions, we say that algorithm has the complexity (or runs in time equal to) $O(log(n))$, $O(n)$, $O(n\log(n))$, and so one...

Examples: $\log n \in O(n)$, $n^2 \in \Omega(n)$, $n \in O(n)$, $n \in \Omega(n)$, $n \in \Theta(n)$, $20n \in \Theta(n)$.

Some other examples `http://lcm.csa.iisc.ernet.in/dsa/node6.html`.

Exercise

Task 1

Sort given functions in ascending order according to their ranks of order: $2^{n!}$, n^2 , $(2009!)n^2$, $n^3 - n^2$, n^3 , $n^3 + n^2$, 2^n , $n^{2009!}$, $n!$, $n\log n$, $\log n$. Are there any functions of the same rank ?

Find and prove relation between functions:

— $2n$ compare with $(20!)n$,
— $n^2 - n$ compare with $n^2 + n$,
— $\log_2 n$ compare with n,
— 3^n compare with 2^n
— $n!$ compare with 2^n

Example: $2n \in \Theta((20!)n)$. Proof: For $c_1 = 1$ and $c_2 = \frac{1}{20!}$, it holds that for every $n \geqslant 1$, $c_2 \cdot (20!)n = n \leqslant 2n \leqslant (20!)n = c_1 \cdot (20!)n$.

Task 2

For given number $n \in \mathbb{N}$ generate all prime numbers which are less or equal n. Compare the complexity of two given algorithms:

— for all numbers $2 \leqslant k \leqslant n$ sequentially we check whether it is a prime number
— we use method called Sieve of Eratosthenes `http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes` from the list $2...n$ we cross out consecutively: numbers which are divisible by 2, numbers which are divisible by 3,..and so one up to $\sqrt{n}$.

Can both of this algorithm be implemented such that theirs time complexity would be equal/different?

Task 3

There is a polynomial $W(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} + a_nx^n$. We want to find the value of this polynomial in point x_0, we can do it in two ways:

— calculate direct from the formula $W(x_0) = a_0 + a_1x_0 + a_2x^2 + \ldots + a_nx^n$
— use *Horner scheme* transforming the polynomial into form

$$\begin{aligned} W(x) &= a_0 + x(a_1 + a_2x + \ldots + a_{n-1}x^{n-2} + a_nx^{n-1}) = \\ &= a_0 + x(a_1 + x(a_2 + \ldots + a_{n-1}x^{n-3} + a_nx^{n-2})) = \\ &= a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1}x + a_n)\ldots)) \end{aligned}$$

Find the time complexity of both algorithms when we choose the dominating operation to be a)Adding b) Multiplying c)Any arithmetic operation.

Task 4

Find complexity of algorithms from previous class (task 2-5).

Additional information

There would be some taks in the final test to sort given functions in respect to their ranks of order, compare some functions and prove the relation between them and to calculate time complexity of some algorithm. So there is no homework from this class.