ITMO UNIVERSITY

Saint Petersburg, Russia

Analysis and Development of Algorithms

Lecture 1 (introductory and informal)

About me

ITMO UNIVERSITY

Dr Petr Chunaev

- Candidate of Physical and Mathematical Sciences, PhD
- Researcher at eScience Research Institute of ITMO University
- Lecturer at School of Translational Information Technologies of ITMO University

Main scientific interests:

- Pure mathematics (Approximation Theory, Functional Analysis, Operator Theory)
- Numerical methods, mathematical modelling
- Social networks analysis
- FinTech

ITMOre than a UNIVERSITY

Contact information

- chunaev@itmo.ru (send reports)
- telegram @yamatico (ask questions)
- telegram channel for lectures and tasks https://t.me/joinchat/AAAAAFkAsNgh3sFBfzeubQ
- 4 Birzhevaya Line, room 306B (send hand-writing letters)

Course structure (very flexible)

ITMO UNIVERSITY

- Introduction to Algorithms
- Algorithm complexity
- Basic algorithms
- Basic structures

To get the course credits you should

- Write several programs on **preferable** programming language and perform a number of experiments
- Work on a project and defend it with a presentation

ITMOre than a UNIVERSITY

The flexibility of the course

ITMO UNIVERSITY

Let us take into account

- your current knowledge of Theory of Algorithms,
- your general interests in Computer Science, Big Data and Machine Learning,
- algorithms necessary for you current or future work

and find a reasonable balance between theory and practice

Short survey

Your experience in algorithms

ITMO more than a UNIVERSITY

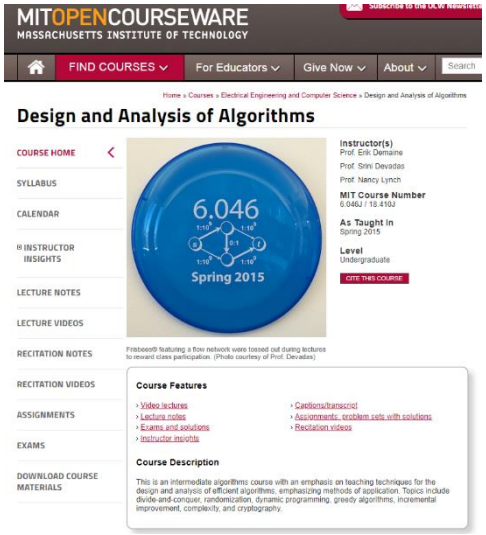
References

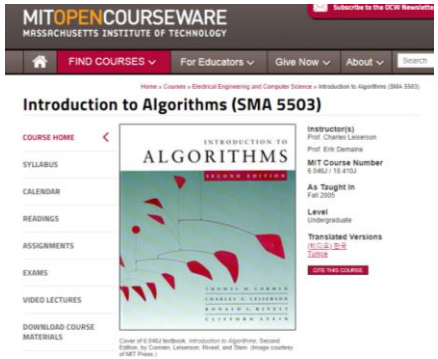
ITMO UNIVERSITY

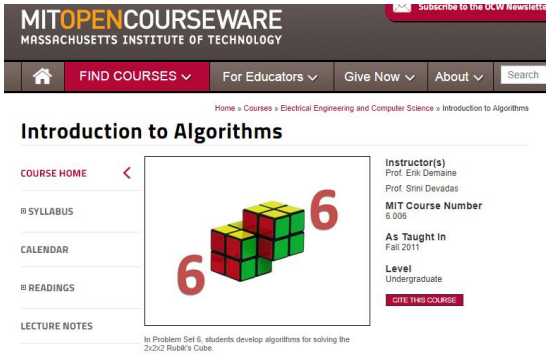
Algorithms and data structures

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest; Clifford Stein. Introduction to Algorithms. 3rd edition. MIT Press, 2009.
2. Ananiy Levitin. Introduction to the Design and Analysis of Algorithms. Addison Wesley, 2011.

Other textbooks, lecture notes and YouTube tutorials







ITMOre than a UNIVERSITY

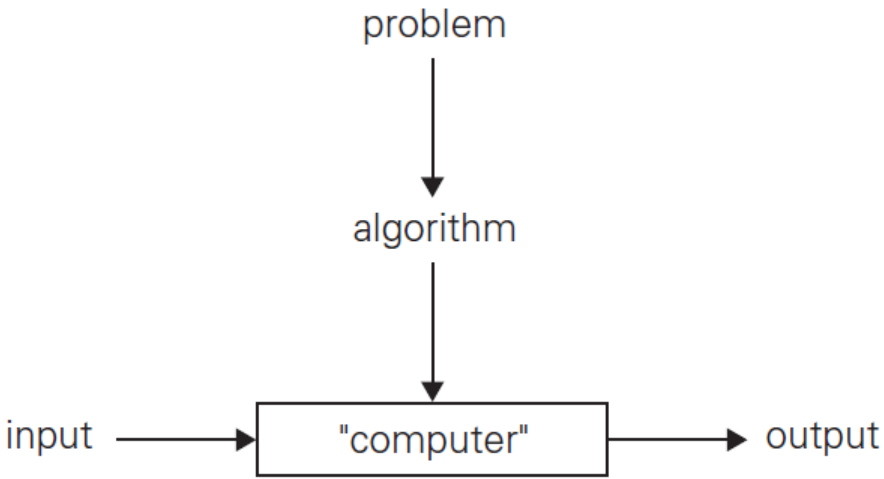
Introduction to algorithms and their complexity analysis (refreshment)

Definition of an algorithm

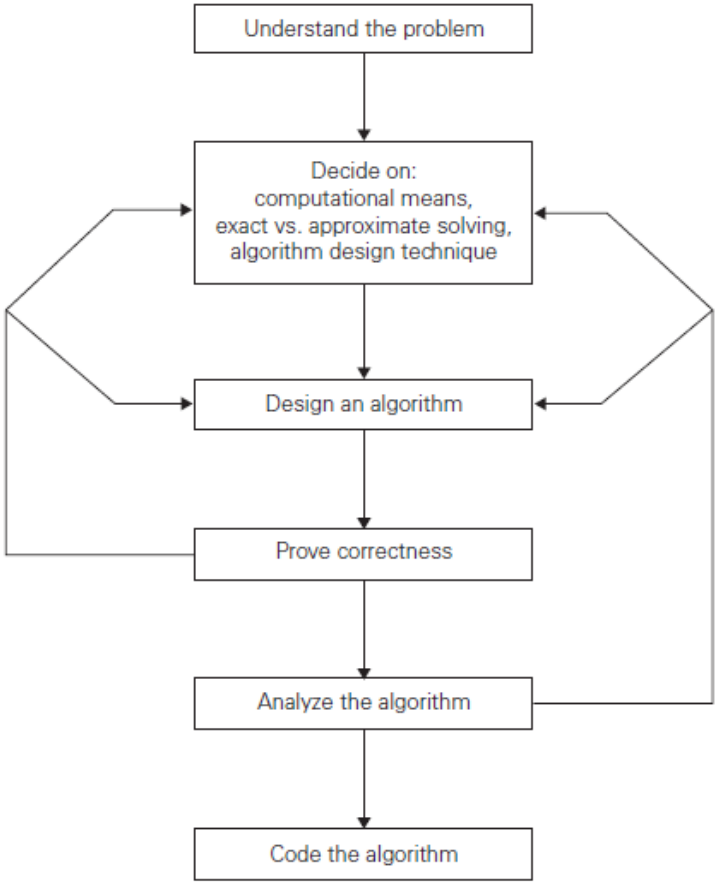
An ***algorithm*** is a sequence of clear instructions for solving a problem, i.e., for obtaining a required output for any acceptable input in a finite amount of time.

- The class of inputs for which an algorithm works has to be specified carefully.
- The same algorithm can be represented in several different ways.
- There may exist several algorithms for solving the same problem.
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speed.

Definition of algorithm



Understanding the Problem



Why do we study algorithms?

ITMO UNIVERSITY

"Algorithms + Data Structures = Programs." — *Niklaus Wirth*

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

— *Linus Torvalds (creator of Linux)*

Why do we study algorithms?

ITMO UNIVERSITY

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮

Design and Analysis of Algorithms

ITMO UNIVERSITY

Aspects of studying algorithms:

1. *Designing algorithms:*
   - putting the pieces of the puzzles together,
   - choosing data structures,
   - selecting the basic approaches to obtain a solution to the problem,
   - The most popular design strategies are:
     - ***divide&conquer,***
     - ***greedy,***
     - ***dynamic programming***
     - ***linear programming***
     - ***backtracking,***
     - ***branch&bound.***
     - ***heuristics***
2. *Expressing and implementing the algorithm*

Concerns are:

- Clearness, conciseness, effectiveness, etc.

Algorithm Performance

ITMO UNIVERSITY

There are *two aspects* of algorithm performance:

Time & space

- Time
  - Instructions take time.
  - How fast does the algorithm perform?
  - What affects its runtime?
- Space
  - Data structures take space
  - What kind of data structures can be used?
  - How does choice of data structure affect the runtime?

➢ **We will focus on time:**

- How to estimate the time required for an algorithm
- How to reduce the time required

- ***Analysis of Algorithms***
  - Analyze the efficiency of different methods to obtain a solution.
- How do we compare the time efficiency of two algorithms that solve the same problem?

Naive Approach: implement these algorithms in a programming language and run them to compare their time requirements (actually, your Task 1).

Analysis of Algorithms

Comparing the programs (instead of algorithms) has difficulties.

- *What data should the program use?*
  - Any analysis must be independent of specific data. Execution time is sensitive to the amount of data manipulated, grows as the amount of data increases.
- *What computer should we use?*
  - We should compare the efficiency of the algorithms independently of a particular computer. Because of the execution speed of the processors, the execution times for an algorithm on the same data set on two different computers may differ.
- *How are the algorithms coded?*
  - Comparing running times means comparing the implementations.
  - We should not compare implementations, because they are sensitive to programming style that may cloud the issue of which algorithm is inherently more efficient.

Analysis of Algorithms

ITMO UNIVERSITY

- Analyze algorithms independently of
  - *specific implementations, computers,* or *data.*

- To analyze algorithms:
  - First, we start to count the number of significant operations in a particular solution to assess its efficiency.
  - Then, we will express the efficiency of algorithms using growth functions.

What is Important?

- If the problem size is always very small, we can probably ignore the algorithm's efficiency.
- We have to weight the trade-offs between algorithm's time requirement and its memory requirements.

ITMO more than a UNIVERSITY

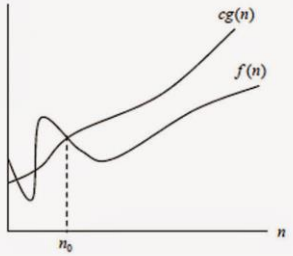
Analysis of Algorithms

ITMO UNIVERSITY

- A running time function, **T(n)**, yields the time required to execute the algorithm of a problem of size n.
  - T(n) may contain unspecified constants, which depend on the characteristics of the ideal machine.
  - We cannot determine this function exactly.
  - Example: $T(n) = an^2 + bn + c$,
    - where a, b and c are unspecified constants.

The (Big) O Notation

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.



$g(n)$ is an asymptotic upper bound for $f(n)$.

Examples:

$n^2 = O(n^2)$
$n^2 + n = O(n^2)$
$n^2 + 1000n = O(n^2)$
$5230n^2 + 1000n = O(n^2)$

$n = O(n^2)$
$\frac{n}{1200} = O(n^2)$
$n^{1.99999} = O(n^2)$
$\frac{n^2}{\log n} = O(n^2)$

Note: Since changing the base of a log only changes the function by a constant factor, we usually don't worry about log bases in asymptotic notation.

ITMO more than a UNIVERSITY

General Rules for Estimation

- **Loops**
  - The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.
- **Nested Loops**
  - Running time of a nested loop containing a statement in the inner most loop is the running time of statement multiplied by the product of the size of all loops.
- **Consecutive Statements**
  - Just add the running times of those consecutive statements.
- **If/Else**
  - Never more than the running time of the test plus the larger of running times of the options.

Analysis of Algorithms

ITMO UNIVERSITY

We measure the complexity of an algorithm by identifying a basic operation and then counting how many times the algorithm performs that basic operation for an input size n.

Each operation has its own real running time depending on many factors (programming language, particular computer, etc.)

problem	input of size *n*	*Basic operation*
searching in a list	lists with n elements	Comparison
sorting a list	lists with n elements	Comparison
multiplying two matrices	two n-by-n matrices	Multiplication
traversing a tree	tree with n nodes	Accessing a node

The Execution Time of Algorithms

Example: Simple If-Statement

The input is n

	Cost	Times
`if (n < 0)`	c1	1
`  av = -n;`	c2	1
`else`		
`  av = n;`	c3	1

Total Cost = c1 + max(c2,c3) = O(1)

The Execution Time of Algorithms

Example: Simple Loop

The input is n

	Cost	Times
`i = 1;`	c1	1
`sum = 0;`	c2	1
`while (i <= n) {`	c3	n+1
`  i = i + 1;`	c4	n
`  sum = sum + i;`	c5	n
`}`		

Total Cost = c1 + c2 + (n+1)c3 + n(c4 + c5)=O(n)

The Execution Time of Algorithms

ITMO UNIVERSITY

Example: Nested Loop

The input is n

```
                                Cost        Times
i=1;                            c1            1
sum = 0;                        c2            1
while (i <= n) {                c3            n+1
 j=1;                           c4            n
 while (j <= n) {               c5            n*(n+1)
     sum = sum + i;             c6            n*n
     j = j + 1;                 c7            n*n
   }
   i = i +1;                    c8            n
}
```

Total Cost = $c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+n*n*c6+n*n*c7+n*c8=O(n^2)$

Algorithm Growth Rates

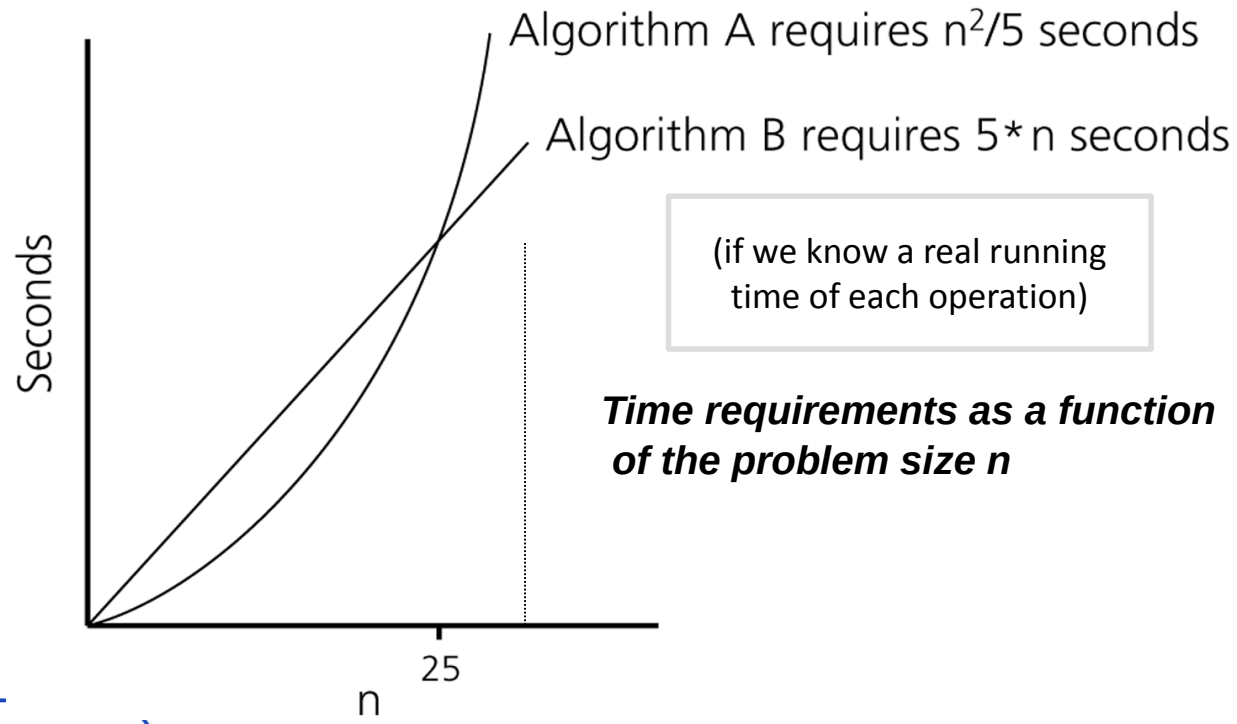
- Algorithm's proportional time requirement is known as *growth rate*.
  - The *growth rate* of ***T(n)*** is referred to the *computational complexity* of the algorithm (the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform).

- The *computational complexity* gives a concise way of saying how the running time, *T(n)*, varies with *n* and is independent of any particular implementation.

Algorithm Growth Rates

- We can compare the efficiency of two algorithms by comparing their growth rates.
  - The lower the growth rate, the faster the algorithm, at least for large values of n
- *For example,* $T(n) = an^2 + bn + c$, the growth rate is $O(n^2)$
- The *goal* of the *algorithm designer:* an algorithm with as low a growth rate of the running function, T(n), of that algorithm as possible

Algorithm Growth Rates

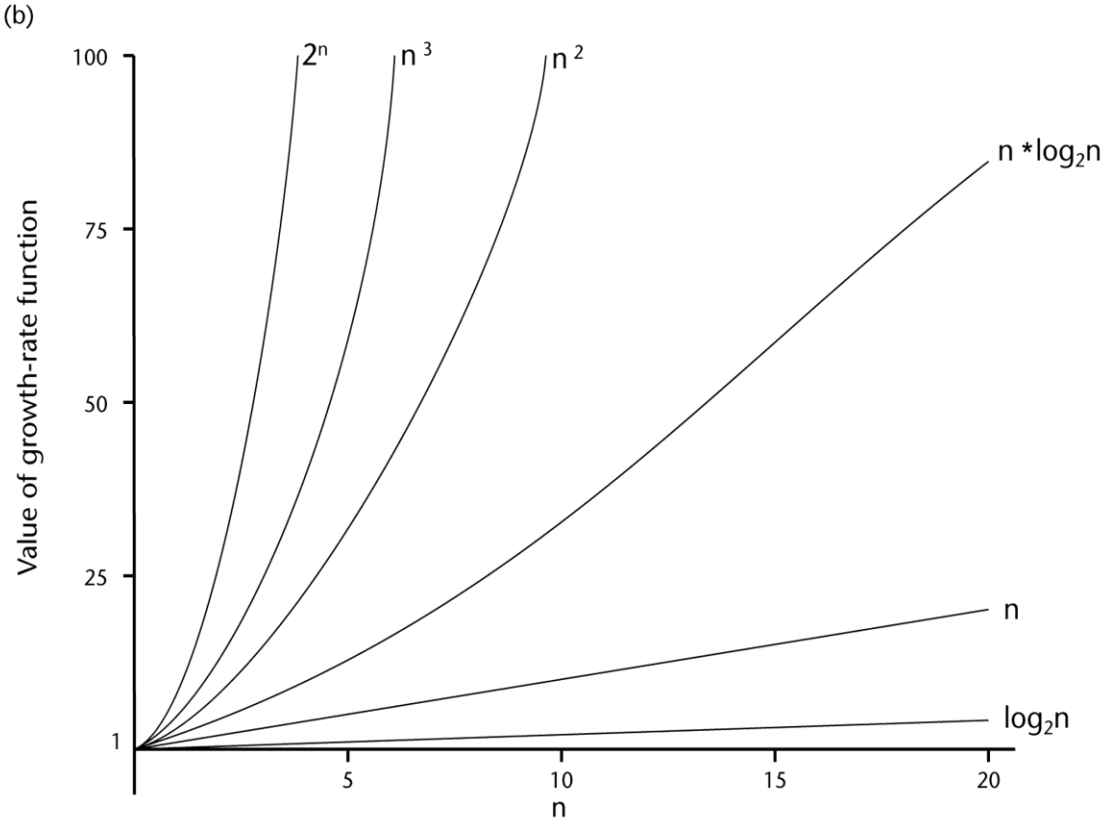


(if we know a real running time of each operation)

Time requirements as a function of the problem size n

order of growth	name	typical code framework	description	example
1	constant	`a = b + c;`	statement	add two numbers
log N	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum
N^2	quadratic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs
N^3	cubic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples
2^N	exponential	Combinatorial algorithms	exhaustive search	check all subsets

A Comparison of Growth-Rate Functions



Thank you for your attention!

ITMOre than a UNIVERSITY