Структуры и алгоритмы обработки данных

Лекция 5

Анализ алгоритмов и их сложности

Сложность алгоритма

Для практики недостаточно знать, что задача алгоритмически разрешима

Т. к. ресурсы ЭВМ (оперативная память и время процессора) ограничены, следует выбирать из эквивалентных алгоритмов наиболее **эффективный**

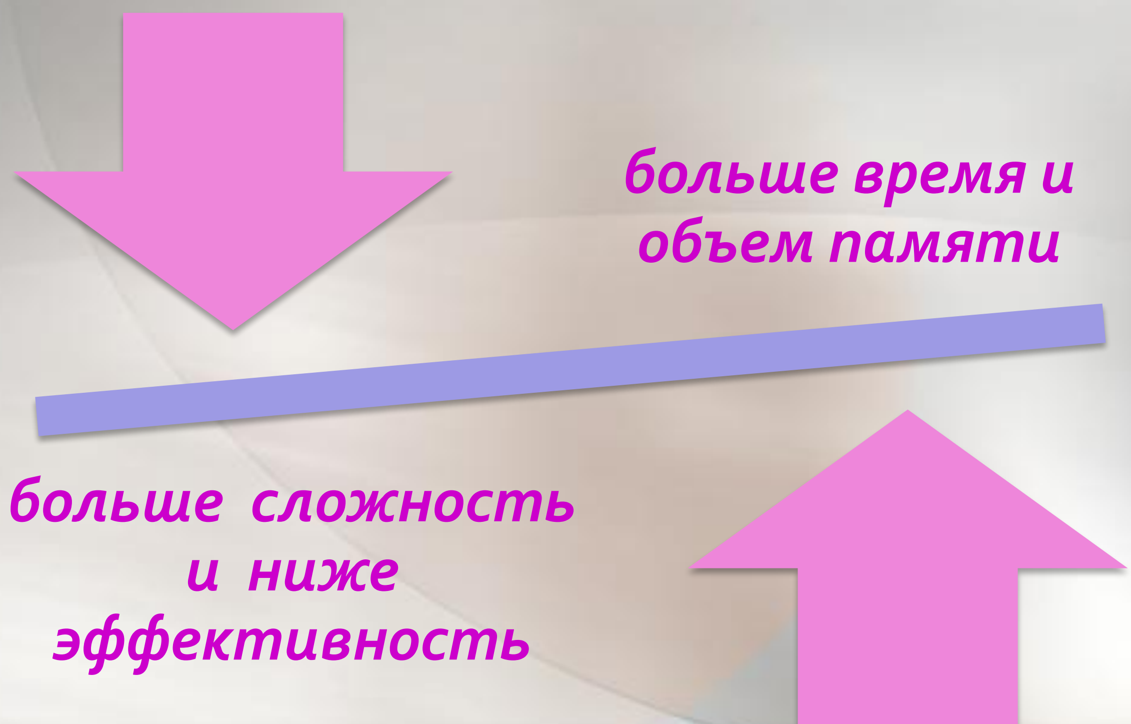
Для оценки качества введено понятие **сложности** или обратное понятие — **эффективность алгоритма**

Сложность алгоритма

Оценка сложности зависит от:

- ***времени*, *затраченного на выполнение алгоритма***
- ***объема памяти*, *требуемой для хранения исходных данных задачи***

Оценка качества алгоритма



Оценка качества алгоритма

Сложность алгоритма

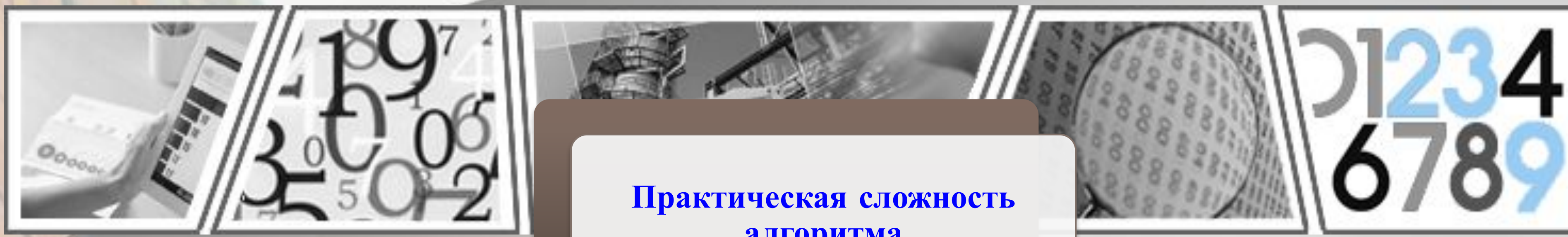
- **Временная**
  - *- характеризует временные затраты на реализацию алгоритма*
- **Емкостная**
  - *- характеризует затраты памяти на реализацию алгоритма*

Оценка качества алгоритма

Сложность алгоритма

- Практическая
- Теоретическая

Оценка качества алгоритма



Практическая сложность алгоритма

Практическая временная сложность

- оценивается во временных единицах (секунды, миллисекунды, количество временных тактов процессора, количество выполнения циклов и т.п.)

Практическая емкостная сложность

- выражается в битах, байтах, словах и т.п.

Сложность алгоритма

выбранный язык программирования

выбранный математический метод формулирования задачи

быстродействие компьютера и его емкостные ресурсы

искусство и опыт программиста

Сложность

Выбор алгоритма

Какой из множества алгоритмов выбрать для решения конкретной задачи?

Алгоритм

Как оценить эффективность программы?

Эффективность программы

Лучший способ сравнения эффективностей алгоритмов - сопоставление их порядков сложности; применим к временной и пространственной сложности

память

время

Задачи и многообразие алгоритмов их решения

Для большинства задач существует более одного способа их решения

Можно сформулировать несколько алгоритмов, приводящих к одному и тому же результату

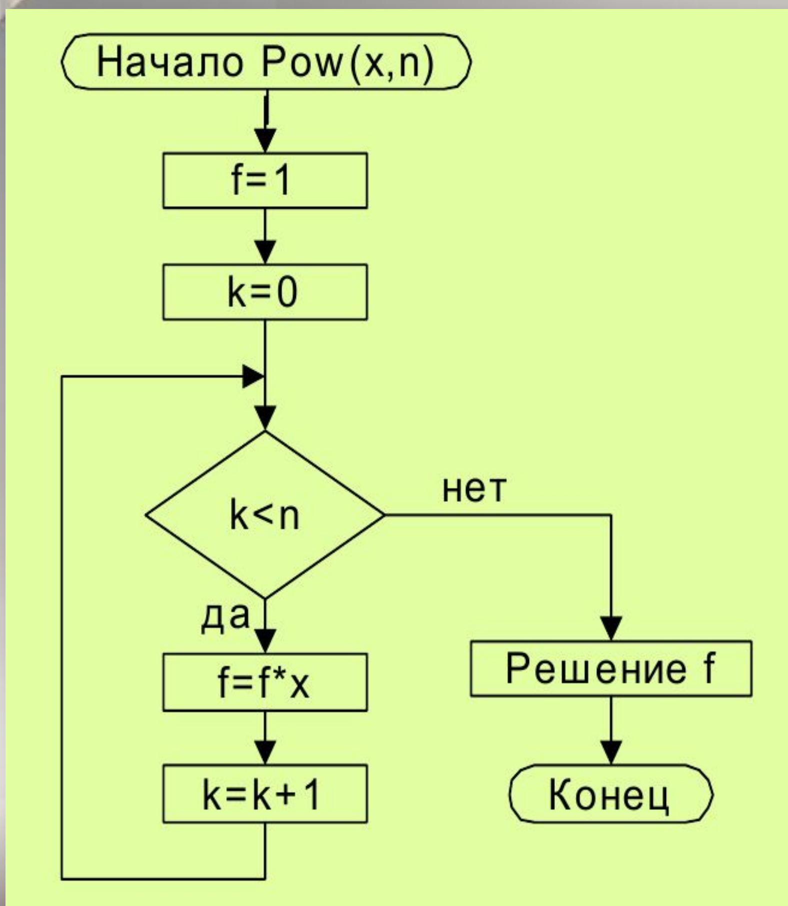
***<u>Пример</u>*: задача возведения в степень**

Задачи и многообразие алгоритмов их решения

Задача возведения в степень

Дано число ***x*** и натуральное (целое неотрицательное) число **n ≥ 0**.

Вычислить значение функции ***f(x) = x^n***



Очевидный способ решения:

$$x^n = \underbrace{x \cdot x \cdot \ldots \cdot x}_{n\ раз}$$

$$f(x) = x^n$$

Задачи и многообразие алгоритмов их решения

Задача возведения в степень

Дано число ***x*** и натуральное (целое неотрицательное) число **n ≥ 0**.

Вычислить значение функции ***f(x) = x^n***

$$x^n = \underbrace{x \cdot x \cdot \ldots \cdot x}_{n\ раз} = \begin{cases} \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз} \cdot \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз} \cdot x, & если\ n - нечетное \\ \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз} \cdot \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз}, & если\ n - четное \end{cases}$$

div **- операция целочисленного деления**

$$x^n = \begin{cases} 1, & если\ n = 0 \\ x \cdot \left(x^{n\,div\,2}\right)^2, & если\ n - нечетное \\ \left(x^{n\,div\,2}\right)^2, & если\ n - четное \end{cases}$$

Задачи и многообразие алгоритмов их решения

Задача возведения в степень

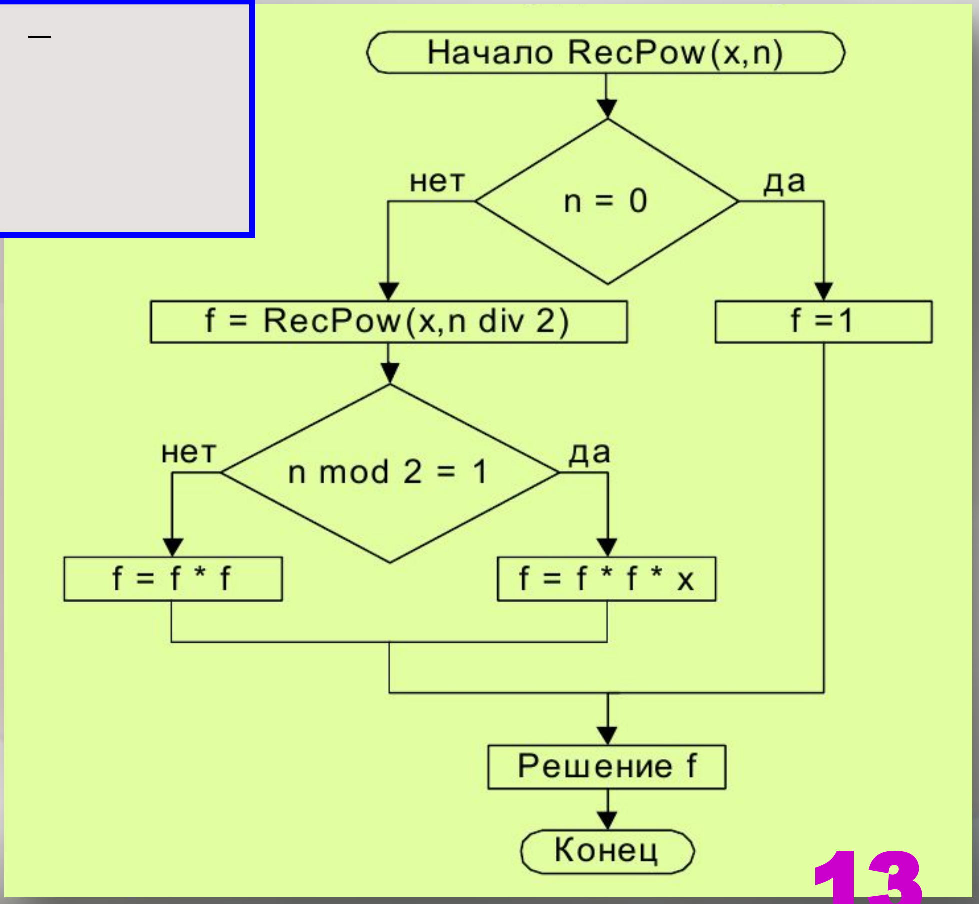
Дано число x и натуральное (целое неотрицательное) число $n \geq 0$.

Вычислить значение функции $f(x) = x^n$

$$x^n = \underbrace{x \cdot x \cdot \ldots \cdot x}_{n\ раз} = \begin{cases} x \cdot \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз} \cdot \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз}, & если\ n - нечетное \\ \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз} \cdot \underbrace{x \cdot \ldots \cdot x}_{n\,div\,2\ раз}, & если\ n - четное \end{cases}$$

$$x^n = \begin{cases} 1, & если\ n = 0 \\ x \cdot \left(x^{n\,div\,2}\right)^2, & если\ n - нечетное \\ \left(x^{n\,div\,2}\right)^2, & если\ n - четное \end{cases}$$

Рекурсивный алгоритм возведения в степень



Задачи и многообразие алгоритмов их решения

Задача возведения в степень

Дано число x и натуральное (целое неотрицательное) число $n \geq 0$.

Вычислить значение функции $f(x) = x^n$

$$n = d_m \cdot 2^m + d_{m-1} \cdot 2^{m-1} + \ldots + d_1 \cdot 2^1 + d_0 \cdot 2^0,$$

$d_i \in \{0, 1\}, \quad 0 \leq i \leq m$ – двоичная цифра

$$x^n = x^{d_m \cdot 2^m} \cdot x^{d_{m-1} \cdot 2^{m-1}} \cdot \ldots \cdot x^{d_1 \cdot 2^1} \cdot x^{d_0 \cdot 2^0}.$$

Обозначим: $c_i = x^{2^i}, \quad 0 \leq i \leq m$

$$x^n = \prod_{i=0}^{m} c_i^{d_i}.$$

$c_0 = x$, и $c_i = c_{i-1}^2$ для всех $1 \leq i \leq m$

$$c_i^{d_i} = \begin{cases} c_i, \text{ если } d_i = 1 \\ 1, \text{ если } d_i = 0 \end{cases}.$$

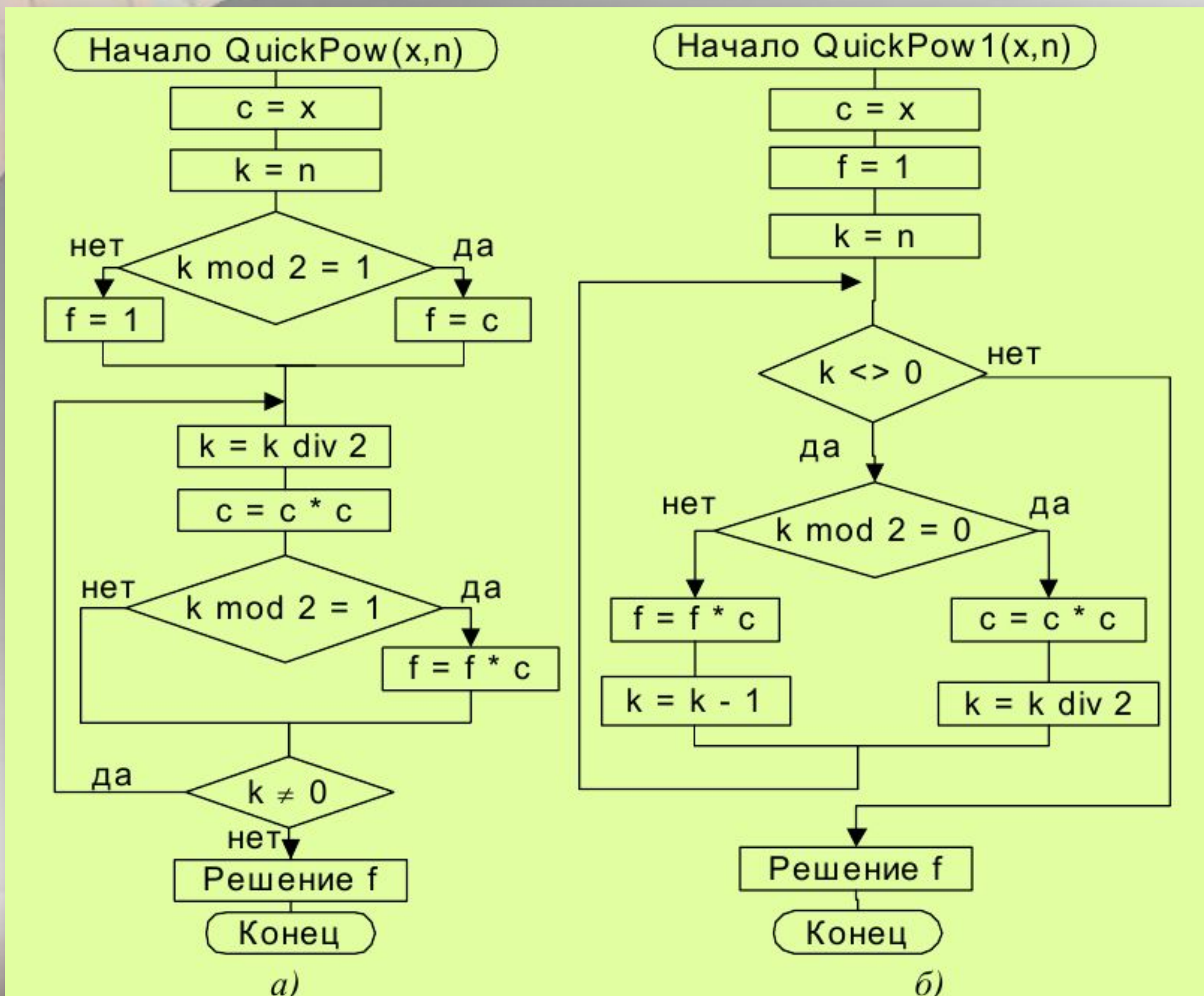
$$c_2 = x^{2^2} = x^4$$

$$c_3 = x^{2^3} = x^8$$

$$c_3 = c_2^{\,2}$$

Задачи и многообразие алгоритмов их решения

Задача возведения в степень – быстрые алгоритмы



Для такой относительно простой задачи, как возведение в степень существуют, по крайней мере, **4 алгоритма**

Проблема выбора алгоритма. Понятие временной сложности

Алгоритм должен

- быть простым для понимания, перевода в программный код и отладки

- эффективно использовать компьютерные ресурсы и выполняться по возможности быстро

Противоречащие друг другу требования

Программа должна выполняться только несколько раз: стоимость программы оптимизируется по стоимости написания (а не выполнения) программы

Решение задачи требует значительных вычислительных затрат: стоимость выполнения программы может превысить стоимость написания программы, особенно при многократном выполнении

Программисты должны быть осведомлены не только о методах построения быстрых программ, но и знать, когда их следует применить, желательно с минимальными программистскими усилиями

Проблема выбора алгоритма. Понятие временной сложности

качество скомпилированного кода исполняемой программы

машинные инструкции, используемые для выполнения программы

ввод исходной информации в программу

временная сложность алгоритма программы

Время выполнения программы

Время выполнения является функцией

- ***размера данных***
- ***самих исходных данных***

Функцию T(n) можно найти экспериментально, но это сложно…

Проблема выбора алгоритма. Понятие временной сложности

Временна́я сложность алгоритма

- это время T, необходимое для его выполнения в зависимости от исходных данных

$$T = k \cdot t,$$

где k – количество вычислительных действий
t – время выполнения одного действия

В практике оценки времени выполнения программ, T(n) понимают как количество элементарных шагов – инструкций, выполняемых <u>на идеализированном компьютере</u>

Проблема выбора алгоритма. Понятие временной сложности

T(n) – ***зависимость времени от объема входных данных***

n – ***это размерность задачи***
(для линейного массива – размер массива)

Поведение **T** ***при увеличении*** **n** ***называется теоретической сложностью*** – **O(**f(n)**)**

Асимптотические соотношения оценки временной сложности

Оценка временной сложности алгоритма – **математически оценить время исполнения подсчетом операций**

1. Записать алгоритм в виде кода одного из развитых языков программирования (например, Java или C++).
2. Перевести программу в последовательность машинных команд (например, байт-коды, используемые в виртуальной машине Java).
3. Определить для каждой машинной команды i время t_i, необходимое для ее выполнения.
4. Определить для каждой машинной команды i количество повторений n_i команды i за время выполнения алгоритма.
5. Определить произведение $t_i * n_i$ всех машинных команд, что и будет составлять <u>время выполнения алгоритма</u>.

Асимптотические соотношения оценки временной сложности

Оценка временной сложности –
математически оценить время исполнения подсчетом операций

Основные (простейшие) операции, аналоги машинных команд:

- ***присваивание переменной значения***
- ***вызов функции***
- ***выполнение арифметической операции***
- ***сравнение двух чисел***
- ***индексация массива***
- ***переход по ссылке на объект***
- ***возвращение из функции***

Асимптотические соотношения оценки временной сложности

Оценка временной сложности –
математически оценить время исполнения подсчетом операций

Пример. Задача вычисления значения многочлена

$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_i \cdot x^i + \ldots + a_1 \cdot x^1 + a_0,$$

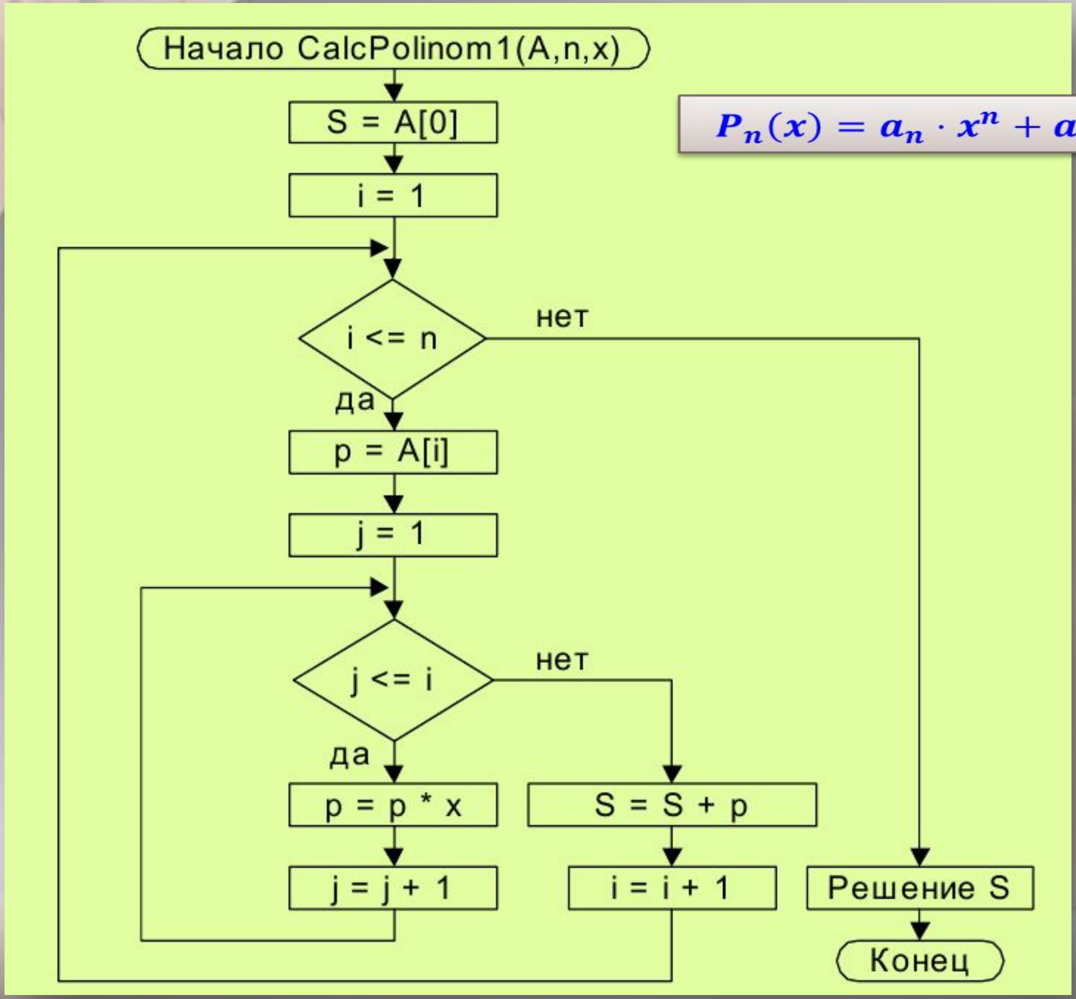
Задано:

- **массив коэффициентов A={A[0], A[1],..., A[n]}**
- **значение переменной x**

Вычислить значение многочлена $S = P_n(x)$

Асимптотические соотношения оценки временной сложности

Пример. Задача вычисления значения многочлена



$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_i \cdot x^i + \ldots + a_1 \cdot x^1 + a_0,$$

Алгоритм **1:**

для каждого слагаемого, кроме a_o возвести x в заданную степень последовательным умножением и затем помножить на коэффициент

Асимптотические соотношения оценки временной сложности

Пример. Задача вычисления значения многочлена

$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_i \cdot x^i + \dots + a_1 \cdot x^1 + a_0,$$

Алгоритм **1,** оценка временной сложности

- Вычисление *i*-го слагаемого(*i=1…n*) требует *i* умножений
  всего 1 + 2 + 3 + … + n = n(n+1)/2 умножений
- Требуется *n* сложений и одна операция начального присваивания значения a_0

Временная сложность алгоритма:

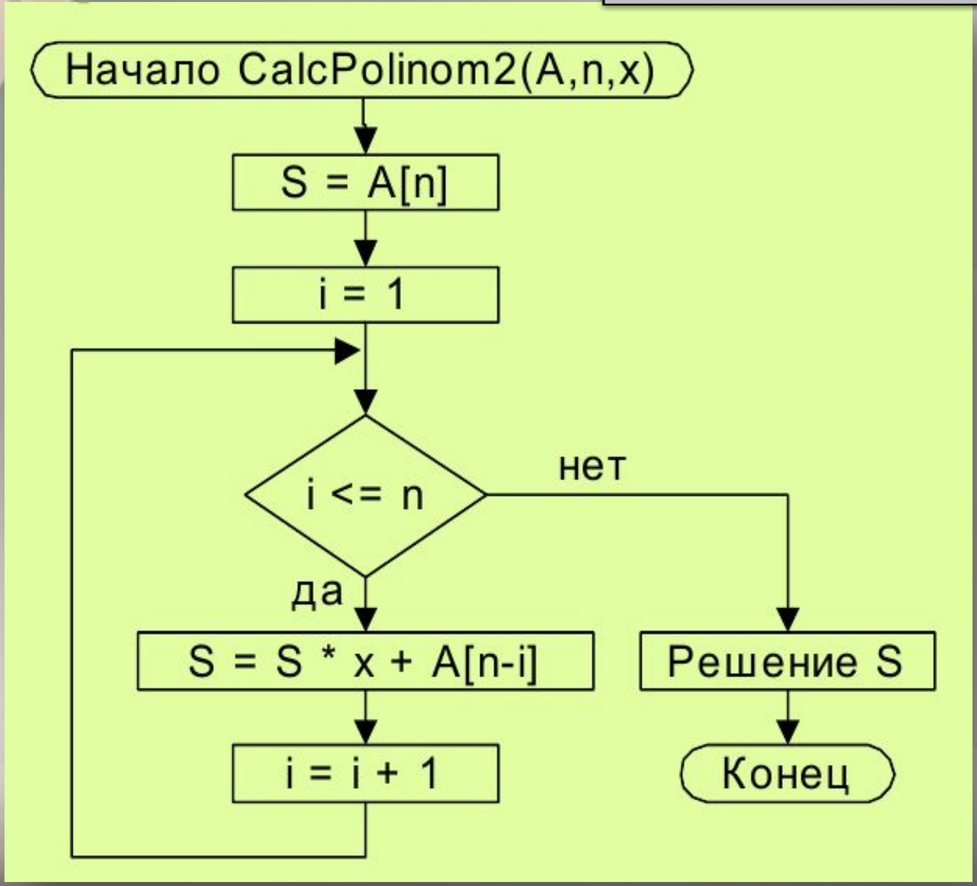
$T(n) = n(n+1)/2 + n + 1 = n^2/2 + 3n/2 + 1$ операций

Асимптотические соотношения оценки временной сложности

Пример. Задача вычисления значения многочлена

$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_i \cdot x^i + \dots + a_1 \cdot x^1 + a_0,$$

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \dots (a_i + x(a_{n-1} + a_n \cdot x))\dots),$$



Алгоритм **2:**

$$S_0 = a_n,$$
$$S_1 = S_0\, x + a_{n-1},$$
$$S_2 = S_1\, x + a_{n-2},$$
$$\dots,$$
$$S_i = S_{i-1}\, x + a_{n-i},$$
$$\dots,$$
$$S_n = S_{n-1}\, x + a_0,\ P_n(x) = S_n$$

Схема Горнера

Асимптотические соотношения оценки временной сложности

Пример. Задача вычисления значения многочлена

$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_i \cdot x^i + \ldots + a_1 \cdot x^1 + a_0,$$

Алгоритм **2,** оценка временной сложности

- **для вычисления S_i требуется 1 умножение и 1 сложение**
  всего такая итерация осуществляется n раз

Временная сложность алгоритма:

T(n) = n умножений + n сложений = 2n операций

Асимптотические соотношения оценки временной сложности

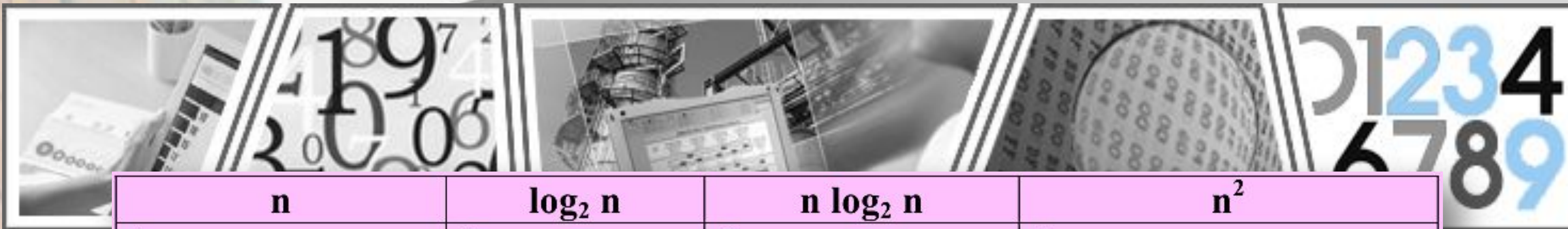
- Обычно подробная оценка временной сложности не требуется
- Достаточно указать асимптотическую скорость возрастания количества операций при увеличении ***n***

Например, функция $T(n) = n^2/2 + 3n/2 + 1$ возрастает приблизительно как $n^2/2$

Это – верхняя оценка, т.е. количество операций (а значит, и время работы) растет не быстрее, чем квадрат количества элементов

Говорят, что *T(n) есть O(n²)*, или *T(n) имеет порядок O(n²)*

Асимптотические соотношения оценки временной сложности



n	$\log_2 n$	$n \log_2 n$	n^2
1	0	0	1
16	4	64	256
256	8	2 048	65 536
4 096	12	49 152	16 777 216
65 536	16	1 048 565	4 294 967 296
1 048 576	20	20 969 520	1 099 301 922 576
16 775 616	24	402 614 784	281 421 292 179 456

Если числа в таблице - микросекунды, то

- **для задачи с *n=1048576* элементами алгоритму с временем работы порядка *O(log n)* потребуется 20 микросекунд**
- **алгоритму с временем работы порядка *O(n²)* – более 12 дней**

Время выполнения алгоритмов

Сложность алгоритма	n=10	$n=10^3$	$N=10^6$
Log n	0.2сек	0.6сек	1.2сек
n	0.6сек	1час	16.6час
n^2	6сек	16.6час	1902года
2^n	1час	10^{295} лет	10^{300000} лет

Асимптотические соотношения оценки временной сложности

Для описания скорости роста функций используется О-символика (Paul Bachman 1894г.)

- "урезанная" верхняя оценка временной сложности алгоритма, отражающая поведение этой сложности *в пределе при увеличении размера задачи до бесконечности*
- называется асимптотической временной сложностью или верхним порядком роста временной сложности

Асимптотические соотношения оценки временной сложности

- Когда мы говорим, что T(n) имеет степень роста O(f(n)), то подразумевается, что f(n) является верхней границей скорости роста T(n)

Когда мы говорим, что время выполнения *T(n)* некоторой программы имеет порядок $O(n^2)$, то подразумевается, что существуют положительные константы *c* и n_0 такие, что для всех $n \geq n_0$, выполняется неравенство $T(n) < c\,n^2$.

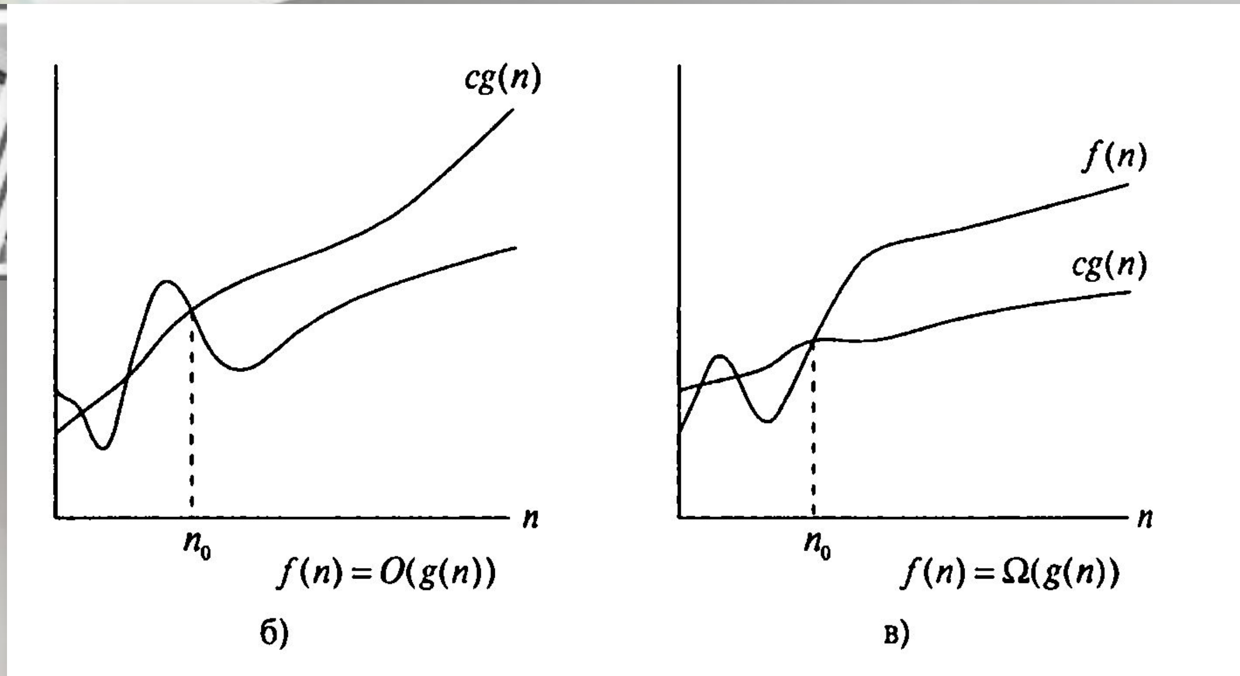
- Чтобы указать нижнюю границу скорости роста T(n), используется обозначение: Ω(g(n)) это подразумевает существование такой константы c, что для бесконечного числа значений n выполняется неравенство T(n) > cg(n).

Асимптотические соотношения оценки временной сложности

Временная сложность алгоритма в худшем случае — функция размера входных данных, которая показывает <u>максимальное количество элементарных операций</u>, которые могут быть затрачены алгоритмом для решения экземпляра задачи указанного размера - O(f(n))

Аналогично определяется понятие временная сложность алгоритма в наилучшем случае - *Ω(g(n))*

Асимптотические соотношения оценки временной сложности



$$\Omega(g(n)) = \left\{ \begin{array}{l} f(n): \text{ существуют положительные константы } c \text{ и } n_0 \\ \text{такие что } 0 \leqslant cg(n) \leqslant f(n) \text{ для всех } n \geqslant n_0 \end{array} \right\}.$$

$$O(g(n)) = \left\{ \begin{array}{l} f(n): \text{ существуют положительные константы } c \text{ и } n_0 \\ \text{такие что } 0 \leqslant f(n) \leqslant cg(n) \text{ для всех } n \geqslant n_0 \end{array} \right\}.$$

Асимптотические соотношения оценки временной сложности

Также используется оценка $\Theta(n)$, которая является комбинаций $O(n)$ и $\Omega(n)$

$\Theta(n)$ - точная оценка асимптотики

$\Theta(g(n))$ – множество функций $T(n)$, для которых существуют такие константы c_1, c_2 и n_0, что $c_1\,g(n) \le T(n) \le c_2\,g(n)$ для всех $n \ge n_0$.
Оценка $\Theta(g(n))$ существует только тогда, когда $O(g(n))$ и $\Omega(g(n))$ совпадают и равна им

Асимптотические соотношения оценки временной сложности

O( ) – асимптотическая оценка алгоритма на худших входных данных,
Ω( ) – на лучших входных данных,
Θ( ) – сокращенная запись одинаковых **O( )** и **Ω( )**.

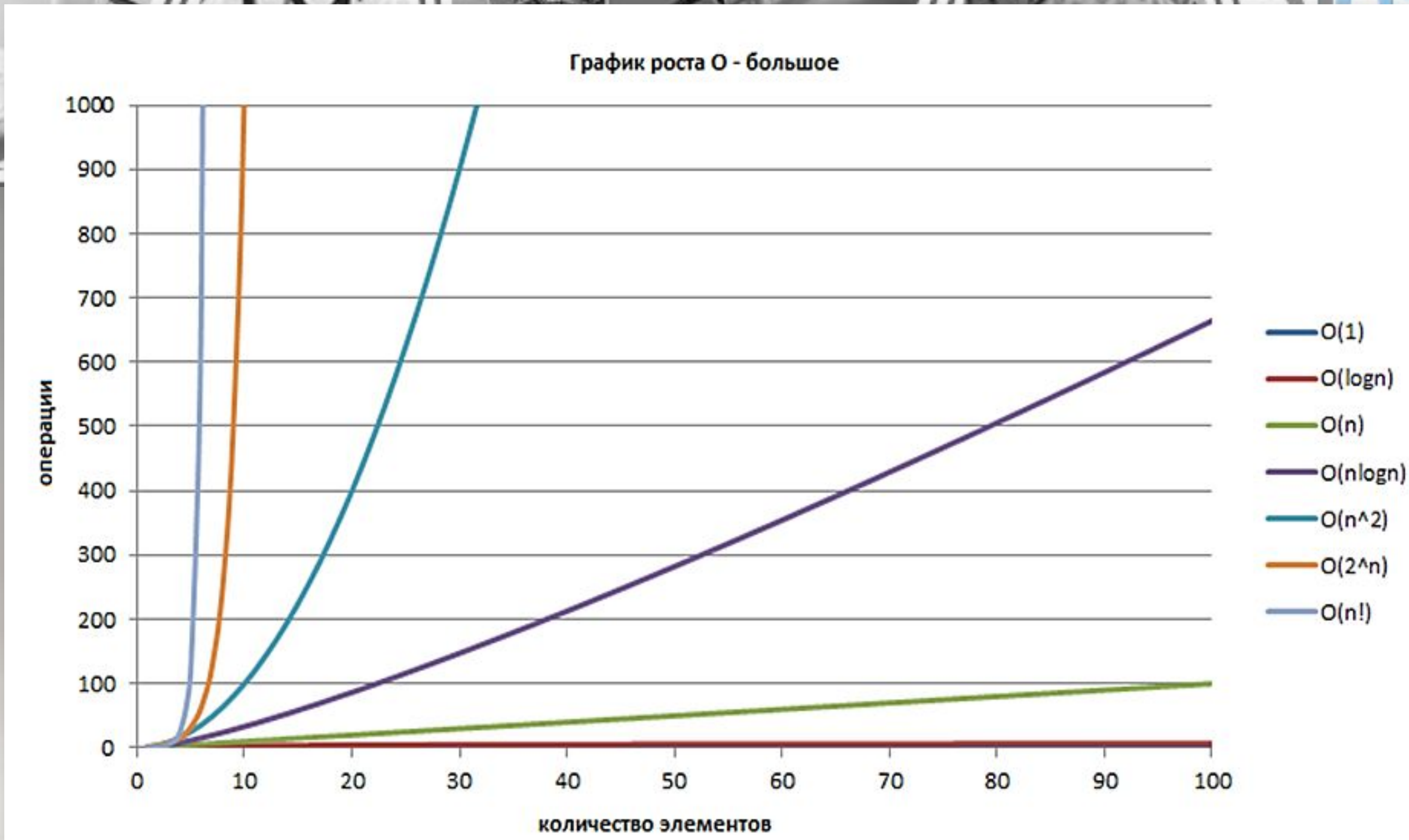
Интуитивный смысл этих оценок:

$$\text{При } n \to \infty\ T(n) = \begin{cases} O(g(n)) : T(n) \text{ растет не быстрее } g(n); \\ \Omega(g(n)) : T(n) \text{ растет не медленне } g(n); \\ \Theta(g(n)) : T(n) \text{ растет так же, как и } g(n). \end{cases}$$

Сравнительная оценка сложности алгоритмов

Сложность $O(f(n))$	Тип зависимости	Значение при $n=2^{10}=$ $=1024$
$O(n)$	*Линейная*	1024
$O(n \cdot \log_2 n)$	*Логарифмическая*	10240
$O(n^2)$ $O(n^3)$	*Полиномиальная*	$\approx 10^6$ $\approx 10^9$
$O(2^n)$	*Экспоненциальная*	$\approx 10^{300}$

Асимптотические соотношения оценки временной сложности



Вычисление временной сложности

Базовые принципы определения времени выполнения программ:

- При оценке за функцию берется количество операций, возрастающее быстрее всего

$$\tfrac{1}{2} n^2 + n = O(n^2)$$

- При оценке O( ) константы не учитываются
- Основание логарифма внутри символа O( ) не пишется

Некоторые операции с символом О

$$f(n) = O(f(n)),$$

$$c \cdot O(f(n)) = O(f(n)), \qquad \text{если } c \text{ — константа},$$

$$O(f(n)) + O(f(n)) = O(f(n)),$$

$$O(O(f(n))) = O(f(n)),$$

$$O(f(n))O(g(n)) = O(f(n)g(n)),$$

Сравнительная оценка сложности алгоритмов

Задача

Дано: *два алгоритма A_1 и A_2, решающих одну и ту же задачу размерности $n=10^6$*

A_1 имеет сложность $O_1(n^2)$ и выполняется на суперкомпьютере с быстродействием 10^8 оп/с;
A_2 имеет сложность $O_2(n \cdot \log_2 n)$ и выполняется на обычном компьютере с быстродействием 10^6 оп/с

Требуется:

найти время решения задачи t_1, t_2 - ?

Сравнительная оценка сложности алгоритмов

Решение

$t_1 = 10^{12} / 10^{8} = 10^{4}$ с ≈ 2,8 ч

$t_2 = 10^{6} \cdot \log_2 10^{6} / 10^{6} = 6 \cdot \log_2 10 \approx 20$ с

Вывод: *Разработка эффективных алгоритмов не менее важна, чем разработка быстрой электроники!*

Временная сложность алгоритмов

n	$\log n$	$\sqrt{n}$	n	$n \log n$	n^2	n^3	2^n
2	1	1,4	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2,8	8	24	64	512	256
16	4	4	16	64	256	4 096	65 536
32	5	5,7	32	160	1 024	32 768	4 294 967 296
64	6	8	64	384	4 096	262 144	$1{,}83 \times 10^{19}$
128	7	11	128	896	16 384	2 097 152	$3{,}40 \times 10^{38}$
256	8	16	256	2 048	65 536	16 777 216	$1{,}15 \times 10^{77}$
512	9	23	512	4 608	262 144	134 217 728	$1{,}34 \times 10^{154}$
1 024	10	32	1 024	10 240	1 048 576	1 073 741 824	$1{,}79 \times 10^{308}$

Рост некоторых функций

Консультации

- понедельник – 5 пара
- пятница – 4 пара