NATIONAL RESEARCH
UNIVERSITY

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Асимптотика

Рамон Антонио Родригес Залепинос
arodriges@hse.ru

Асимптотика – краткое название

- **Скорость роста функций**
- **Asymptotic efficiency/complexity**
- **Сложность алгоритма**

Асимптотически оценивают

- Время работы алгоритма (CPU, GPU, FPGA, …)
- Требующийся объем памяти алгоритму (оперативной, дисковой, …)
- Количество операций ввода/вывода (дисковых, сетевых сообщений, …)
- …

Например

- Время работы алгоритма $O(n^2)$ *– как это интерпретировать?*
- Требующийся объем памяти алгоритму $O(n^3)$ *– как это интерпретировать?*
- Количество операций ввода/вывода $O(n \log n)$ *– как это интерпретировать?*

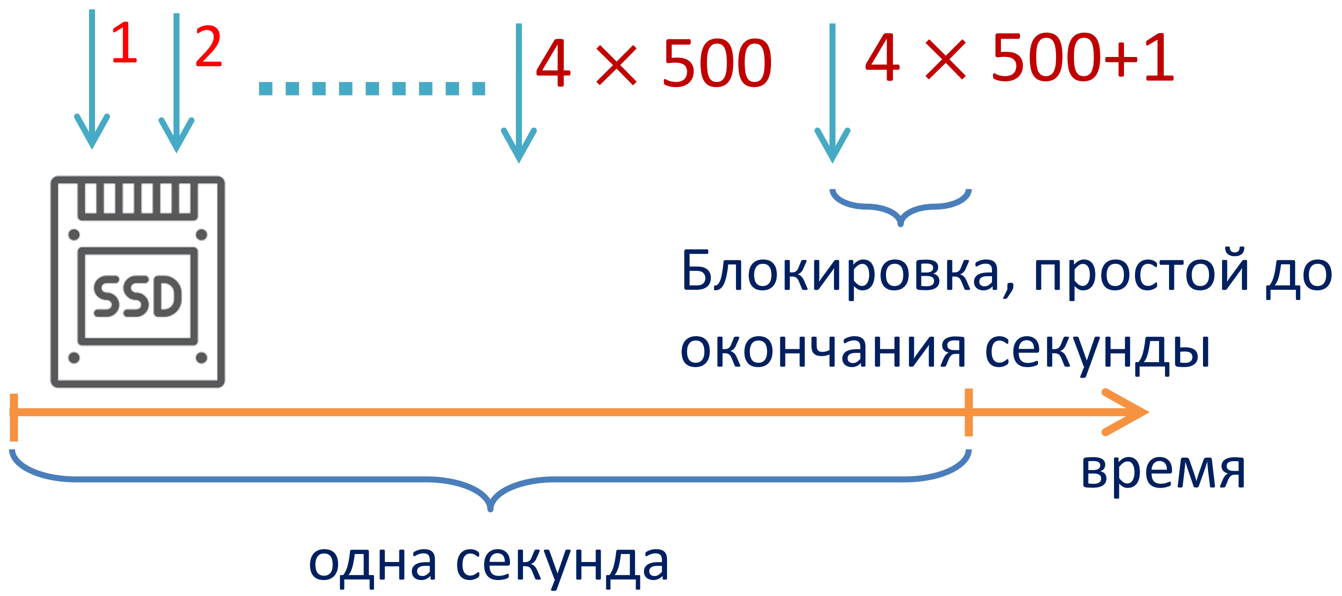
Зачем учитывать объем памяти и число операций ввода/вывода?

Современный пример

Стоимость (цена, руб.) работы алгоритма:

IOPS & объем памяти

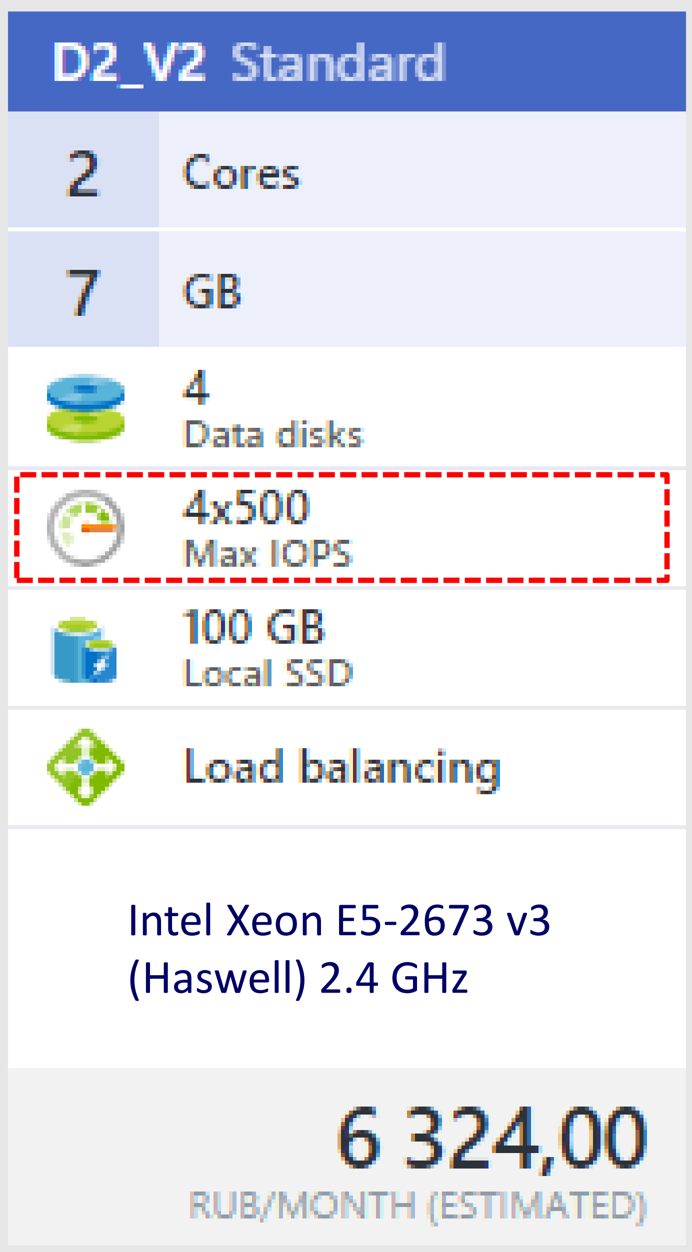
чтение/запись



Microsoft Azure

4 × 500 = max IOPS

Input/**O**utput **O**perations **p**er **S**econd



Мы вернемся к этому при изучении структур данных для вторичной памяти

Для чего нужно знать сложность алгоритма?

Один и тот же алгоритм запускают на

- Разных языках программирования
- Разном оборудовании
- Входных данных разного размера

Нужен метод, который позволит сравнивать характеристики алгоритмов
(время работы, используемую память, количество операций в/в, …)

Зачем?

- Использование алгоритмов на одном и том же оборудовании
- Выбор алгоритмов для реализации в программной системе
- Стоимость работы системы, напр. в «облаке» (cost of ownership)
- …

Асимптотическая оценка и численная оценка (benchmark)

стоимость операций

размер входа, напр. n

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

Алгоритм (псевдокод)

Количество повторов операций в зависимости от n

$T(n)$

$T(n)$

Время работы в зависимости от n: формула

Порядок роста $T(n)$

$O(n^2)$

конкретные входные данные

Конкретная реализация алгоритма

запуск

Время работы в физических единицах, напр. миллисекунды

особенности hardware

Как сравнить алгоритмы, если $T_1(n) = O(g(n)) = T_2(n)$

напр., стр. данных выполняют поиск за $O(\log n)$

Пример: задача сортировки

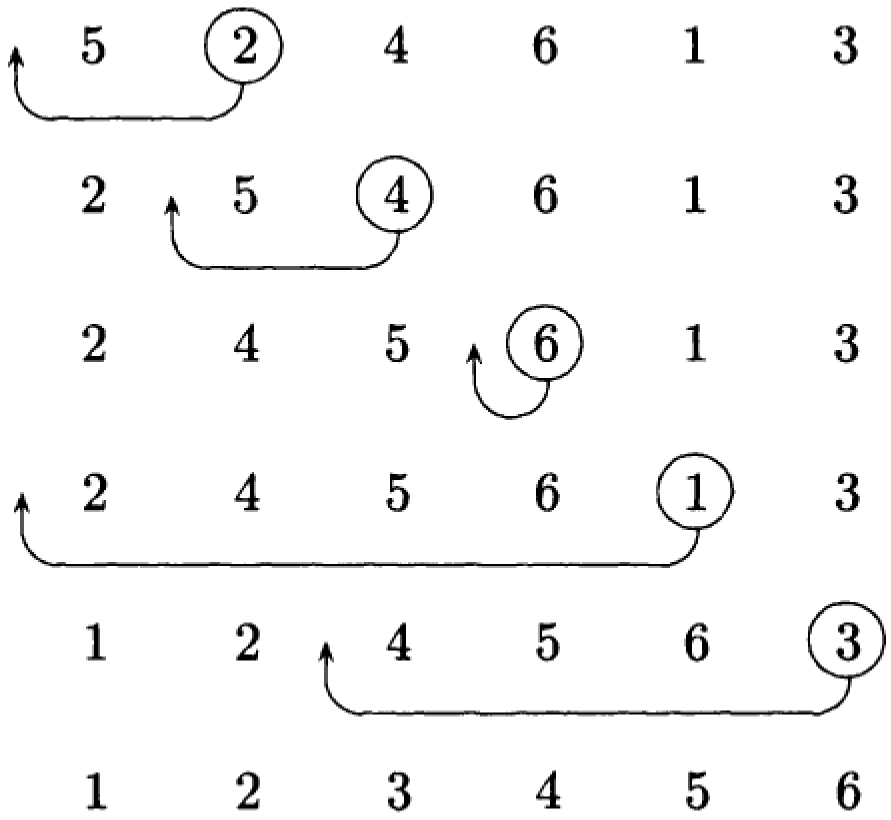
Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.
Выход: Перестановка $\langle a'_1, a'_2, \dots, a'_n \rangle$ исходной последовательности, для которой $a'_1 \le a'_2 \le \dots \le a'_n$.

Алгоритм считают **правильным** (correct), если на любом допустимом (для данной задачи) входе он заканчивает работу и выдает результат, удовлетворяющий требованиям задачи. В этом случае говорят, что алгоритм **решает** (solves) данную вычислительную задачу.

Пример: задача сортировки

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.
Выход: Перестановка $\langle a'_1, a'_2, \dots, a'_n \rangle$ исходной последовательности, для которой $a'_1 \le a'_2 \le \cdots \le a'_n$.

Сортировка вставками



Слева – все элементы уже отсортированы

Справа – элементы, которые нужно вставить в уже отсортированную последовательность

Идем слева направо и вставляем по одному элементу

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Пример: сортировка вставками

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

```
INSERTION-SORT(A)
for j ← 2 to length[A]
      do key ← A[j]
         i ← j − 1
         while i > 0 and A[i] > key
             do A[i + 1] ← A[i]
                i ← i − 1
         A[i + 1] ← key
```

Псевдокод
(всегда должен сопровождаться словесным описанием в письменных работах)

Кормен, Лейзерсон, Ривест:
Алгоритмы. Построение и анализ

Пример: сортировка вставками – анализ

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

	INSERTION-SORT(A)	*стоимость*	*число раз*	
1	**for** $j \leftarrow 2$ **to** $length[A]$	c_1	n	– *почему не* $(n-1)$?
2	**do** $key \leftarrow A[j]$	c_2	$n-1$	
3	$i \leftarrow j-1$	c_4	$n-1$	
4	**while** $i > 0$ **and** $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$	– t_j кол-во выполнений строки 4
5	**do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$	
6	$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$	
7	$A[i+1] \leftarrow key$	c_8	$n-1$	

- Обозначим $\boldsymbol{T(n)}$ **– время работы** (running time) алгоритма
- Время работы зависит от **размера входа** (input size) и от **состояния входных данных**
- Размер входа в данном случае можно положить равным n (длина последовательности)
- c_i - стоимости присваивания, сравнения элементарных типов, …

Пример: сортировка вставками – анализ в лучшем случае

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

```
Insertion-Sort(A)
for j ← 2 to length[A]
    do key ← A[j]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

стоимость	*число раз*
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n}(t_j-1)$
c_7	$\sum_{j=2}^{n}(t_j-1)$
c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) =$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

Алгоритм линейный в лучшем случае
(входные данные уже отсортированы)

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Пример: сортировка вставками – анализ в худшем случае

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

стоимость	*число раз*
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Нас почти всегда интересует время работы в **худшем случае** (почти всегда зависит от состояния входных данных), **массив отсортирован в обратном порядке** → $t_j = j$ (сравниваем key со всеми элементами слева)

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Арифметическая прогрессия

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n.$$

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Докажем по индукции

- для $n = 1$
- Положим, что равенство верно для n и проверим это для $n + 1$

$$\sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1) = n(n+1)/2 + (n+1) = (n+1)(n+2)/2.$$

$$(n+1)\left(\frac{n}{2} + 1\right) = (n+1)\left(\frac{n+2}{2}\right) = (n+1)(n+2)/2$$

Пример: сортировка вставками – анализ

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

INSERTION-SORT(A)

```
for j ← 2 to length[A]
        do key ← A[j]
           i ← j − 1
           while i > 0 and A[i] > key
                  do A[i + 1] ← A[i]
                     i ← i − 1
           A[i + 1] ← key
```

стоимость	*число раз*
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

$$t_j = j$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \boxed{\sum_{j=2}^{n} t_j} + c_6 \boxed{\sum_{j=2}^{n} (t_j - 1)} + c_7 \boxed{\sum_{j=2}^{n} (t_j - 1)} + c_8(n-1)$$

$$\boxed{\sum_{j=2}^{n} j = 2 + \dots + n = \frac{n(n+1)}{2} - 1}$$

$$\boxed{\sum_{j=2}^{n} (j-1) = 1 + \dots + n - 1 = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}}$$

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Пример: сортировка вставками – анализ

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

стоимость	*число раз*
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

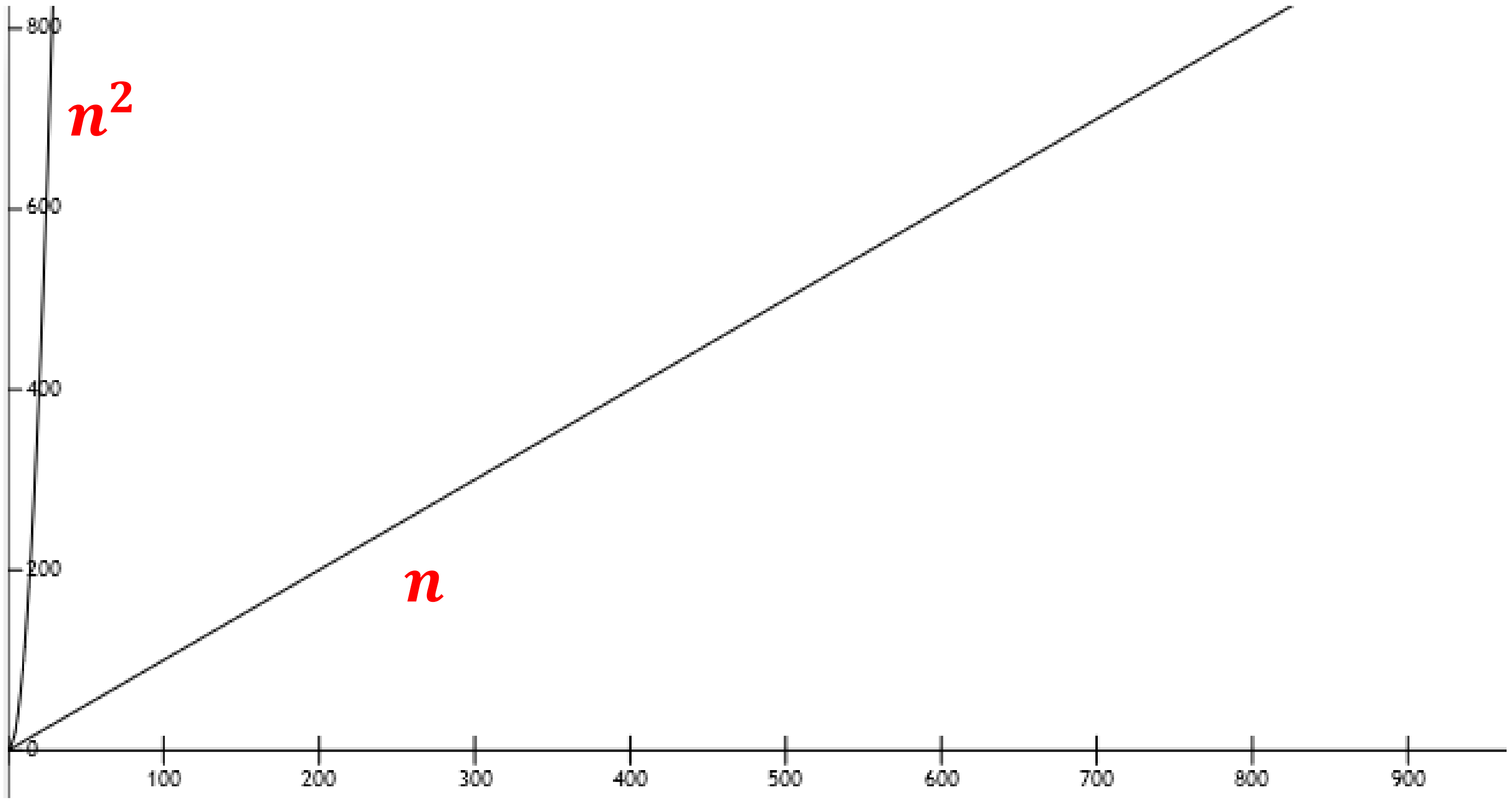
$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) =$$

$$= \boxed{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)} n^2 + \boxed{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)} n - \boxed{(c_2 + c_4 + c_5 + c_8)}$$

$T(n) = \boldsymbol{a} n^2 + \boldsymbol{b} n + \boldsymbol{c}$ (квадратичная функция)

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Скорость роста функций



$T(n) = an^2 + bn + c$ (квадратичная функция)

При достаточно больших n, слагаемое n по сравнению с n^2 не имеет существенного значения, напр. $n = 300$, 90000 + 300

Пример: сортировка вставками – анализ

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
      do key ← A[j]
         i ← j − 1
         while i > 0 and A[i] > key
              do A[i + 1] ← A[i]
                 i ← i − 1
         A[i + 1] ← key
```

стоимость	*число раз*
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

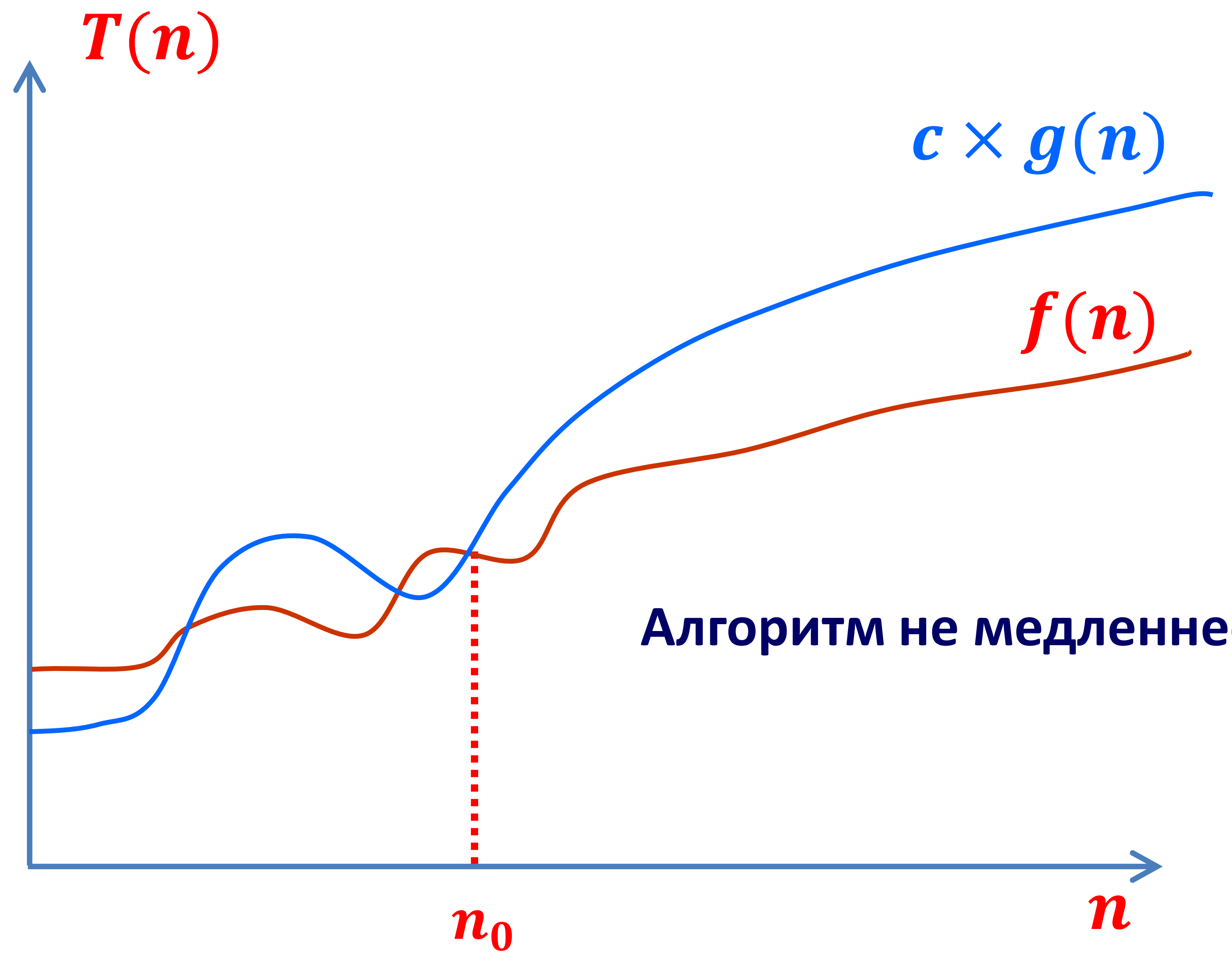
$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) =$$

$$= \boxed{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)} n^2 + \boxed{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)} n - \boxed{(c_2 + c_4 + c_5 + c_8)}$$

$T(n) = \boldsymbol{a} n^2 + \boldsymbol{b} n + \boldsymbol{c} = O(n^2)$ («о большое от n квадрат» – почти сленг)

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

O большое (оценка сверху)



Алгоритм не медленнее $g(n)$

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

$$\exists c, n_0 > 0:$$
$$0 \le f(n) \le c \times g(n)$$
$$\text{для } \forall n \ge n_0$$

т.е. при достаточно большом n

$f(n)$ и $g(n)$ асимптотически положительны:

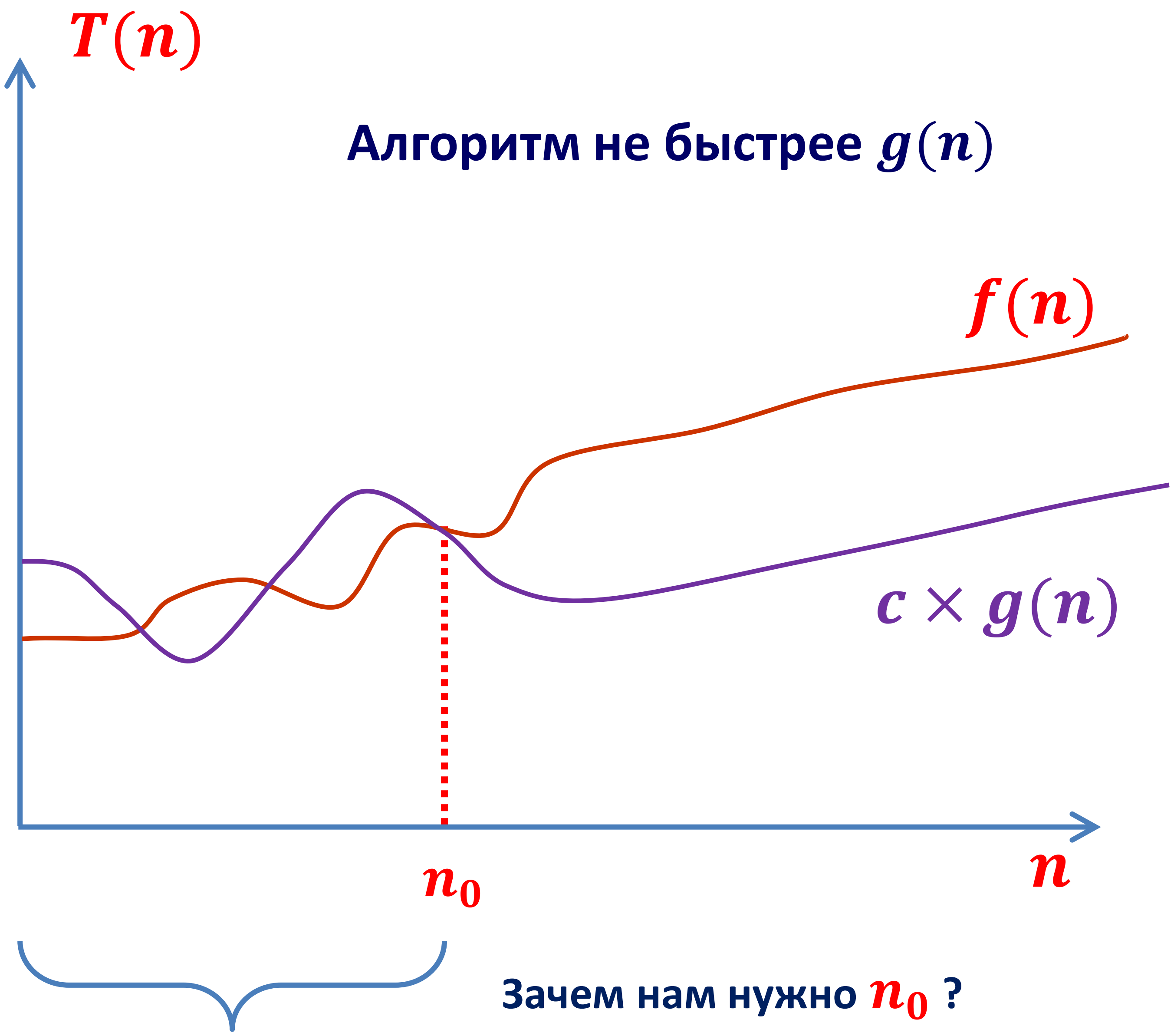
$f(n) \ge 0$ и $g(n) \ge 0$ **при достаточно большом** n

Ω омега большое (оценка снизу)

$$f(n) = \Omega(g(n))$$

$$\exists c, n_0 > 0:$$
$$0 \le c \times g(n) \le f(n)$$
для $\forall n \ge n_0$

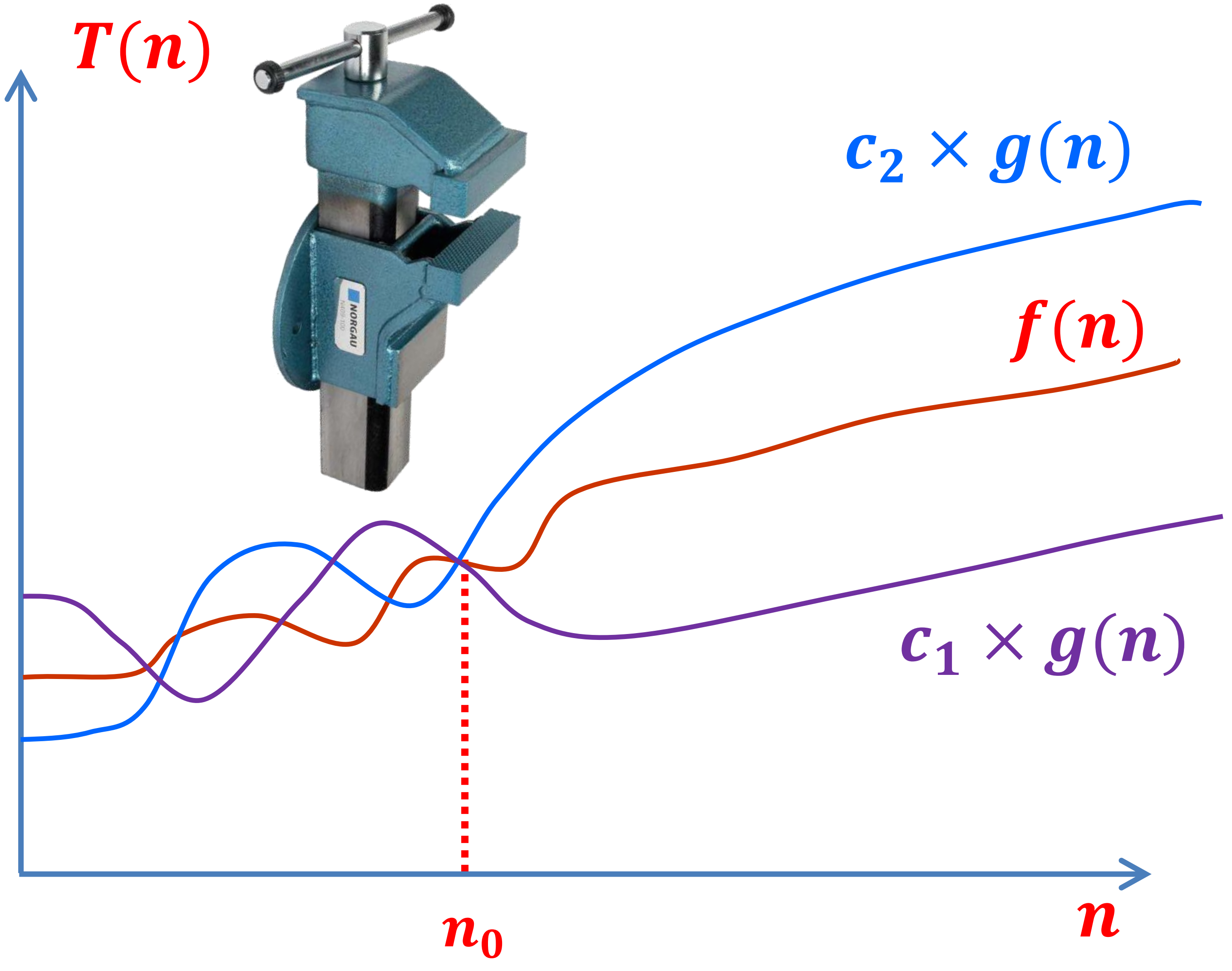
т.е. при достаточно большом n



Зачем нам нужно n_0 ?

При $n < n_0$ доминирует не время работы алгоритма, а фоновые события: время запуска, системные вызовы, другие события OS (значения функций при $n < n_0$ часто можно рассматривать как случайные)

$f(n)$ и $g(n)$ асимптотически положительны



Θ – тета (оценка в среднем)

$$f(n) = \Theta(g(n))$$

$$\exists c_1, c_2, n_0 > 0:$$
$$c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$$
для $\forall n \geq n_0$

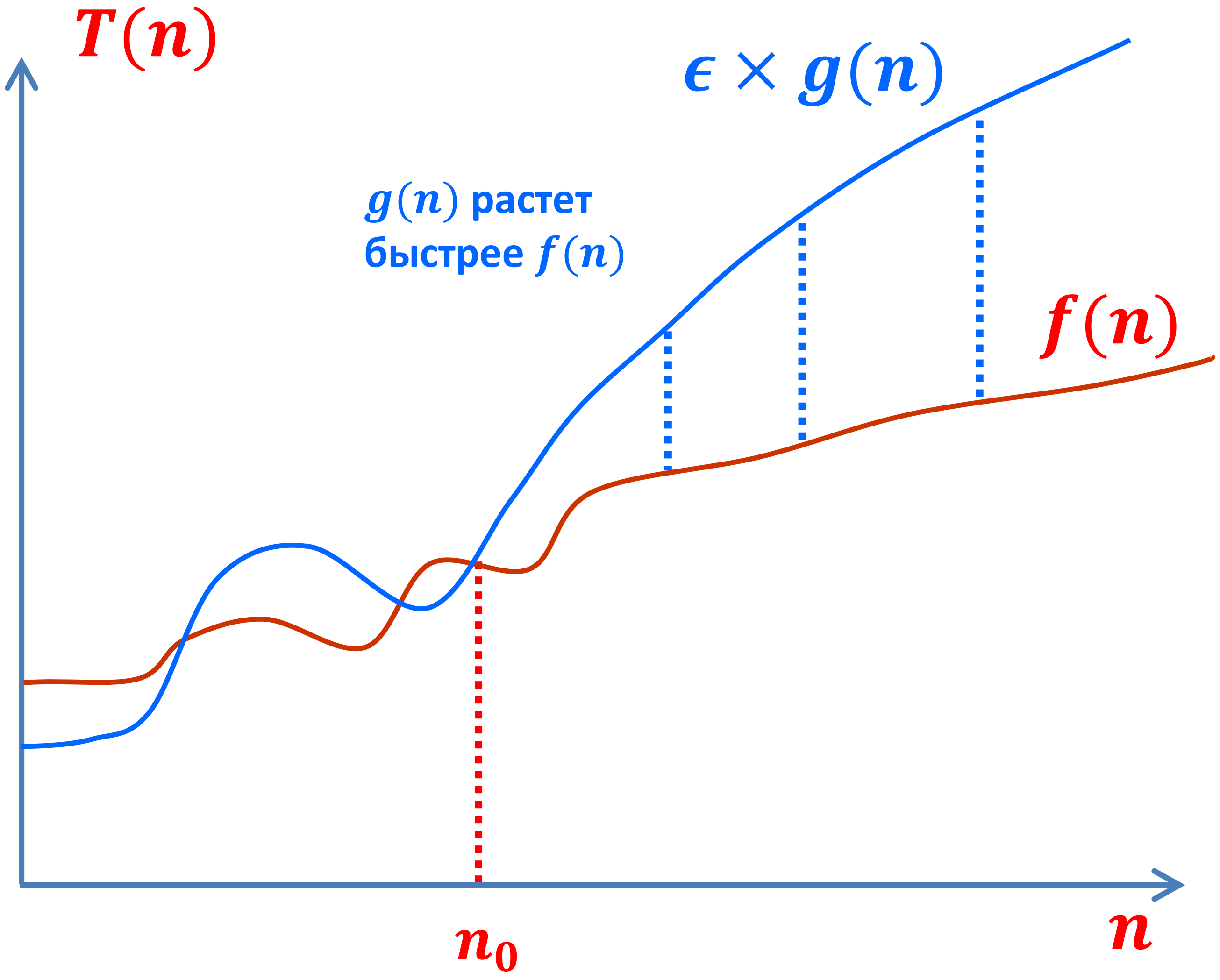
т.е. при достаточно большом n

$f(n)$ и $g(n)$ асимптотически положительны:
$f(n) \geq 0$ и $g(n) \geq 0$
при достаточно большом n

$f(n) = \Theta(g(n))$ – **асимптотически точная оценка**

o малое (грубая оценка сверху)

$$f(n) = o(g(n))$$



1. Если для $\forall \epsilon > 0\ \exists n_0$:
$$0 \le f(n) \le \epsilon \times g(n)$$
для $\forall n \ge n_0$

т.е. при достаточно большом n

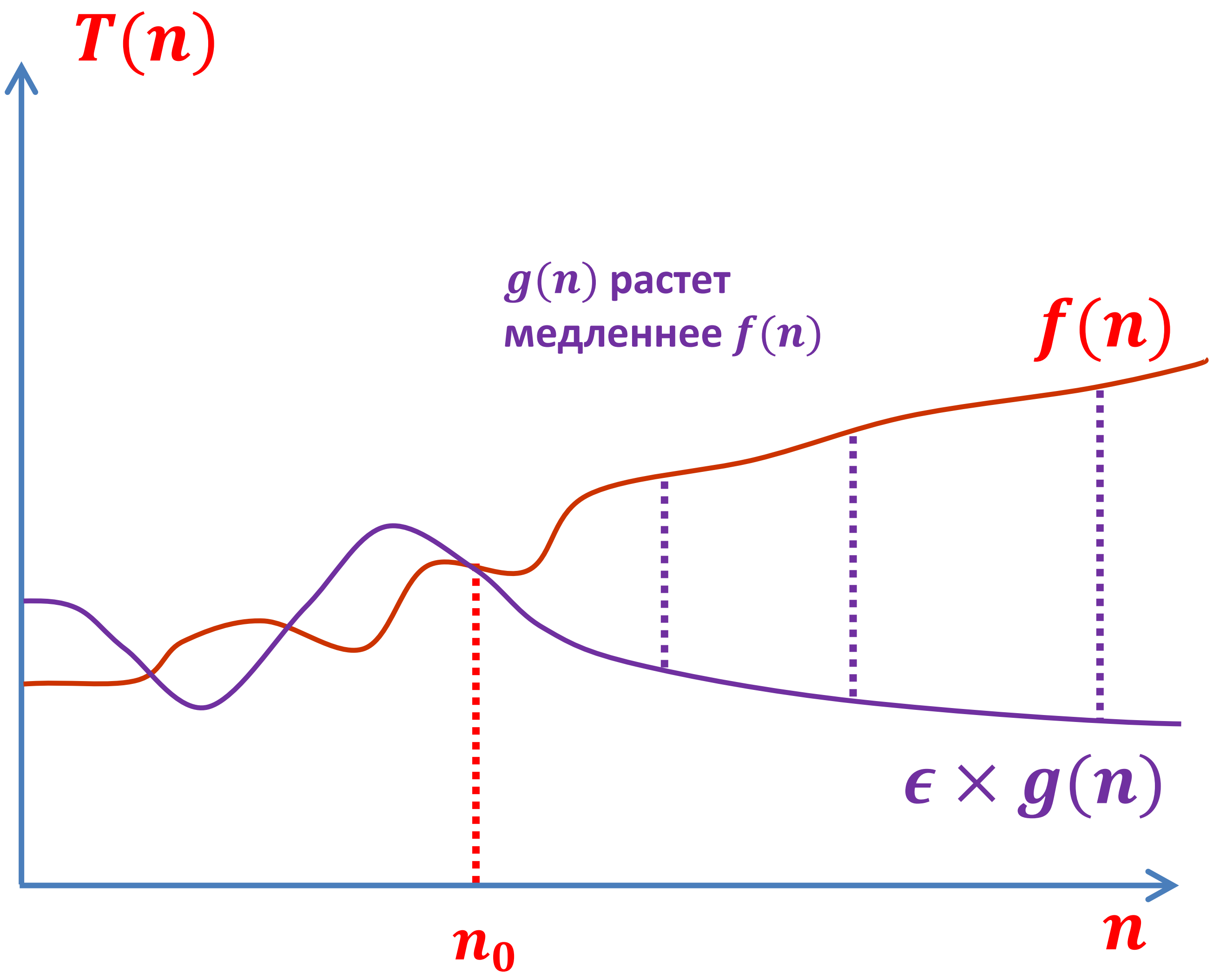
2. $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = O(g(n))$$
$$\frac{f(n)}{g(n)} \approx c$$

$$2n = o(n^2)$$
$$2n^2 \ne o(n^2)$$

ω малое (грубая оценка снизу)

$$f(n) = \omega(g(n))$$



1 **Если для $\forall \epsilon > 0\ \exists n_0$:**

$$0 \le \epsilon \times g(n) \le f(n)$$

для $\forall n \ge n_0$

т.е. при достаточно большом n

2
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

$$f(n) = \Omega(g(n))$$
$$\frac{f(n)}{g(n)} \approx c$$

$$n^2/2 = \omega(n)$$
$$n^2/2 \ne \omega(n^2)$$

Асимптотическая оценка (два крайних случая)

в лучшем случае

в худшем случае

<u>Для обоих случаев можно получить:</u>

обычно

$T(n) = \cdots$

всегда для худшего случая

O – оценка сверху

Θ – оценка в среднем (точная оценка)

Ω – оценка снизу

o – грубая оценка сверху (получить O тяжело)

ω – грубая оценка снизу (получить Ω тяжело)

Обычно никогда на практике не интересна

- это время, за которое алгоритм гарантировано завершится
- мы не знаем заранее какие входные данные поступят – худшие случаи довольно частые, напр. поиск отсутствующего элемента в структуре данных
- часто время работы в худшем почти такое же, как и среднем

«Виды» сложности

- **Константная (постоянная)**
  $$T(n) = O(1)$$
- **Логарифмическая**
  $$T(n) = O(log\ n)$$
- **Линейная**
  $$T(n) = O(n)$$
- **Полиномиальная (квадратичная, кубическая, полином 4-ой степени, …)**
  $$T(n) = O(n^2)$$
- **Экспоненциальная**
  $$T(n) = O(2^n)$$

обычно для структур данных

Алгоритмов больше, чем типичных разновидностей порядков роста => можем readily сравнивать большинство алгоритмов между собой

Свойства

Транзитивность:

$$f(n) = \Theta(g(n)) \text{ и } g(n) = \Theta(h(n)) \text{ влечёт } f(n) = \Theta(h(n)),$$
$$f(n) = O(g(n)) \text{ и } g(n) = O(h(n)) \text{ влечёт } f(n) = O(h(n)),$$
$$f(n) = \Omega(g(n)) \text{ и } g(n) = \Omega(h(n)) \text{ влечёт } f(n) = \Omega(h(n)),$$
$$f(n) = o(g(n)) \text{ и } g(n) = o(h(n)) \text{ влечёт } f(n) = o(h(n)),$$
$$f(n) = \omega(g(n)) \text{ и } g(n) = \omega(h(n)) \text{ влечёт } f(n) = \omega(h(n)).$$

Рефлексивность:

$$f(n) = \Theta(f(n)), \quad f(n) = O(f(n)), \quad f(n) = \Omega(f(n)).$$

Симметричность:

$$f(n) = \Theta(g(n)) \text{ если и только если } g(n) = \Theta(f(n)).$$

Свойства

Обращение:

$f(n) = O(g(n))$ если и только если $g(n) = \Omega(f(n))$,

$f(n) = o(g(n))$ если и только если $g(n) = \omega(f(n))$.

Можно провести такую параллель: отношения между функциями f и g подобны отношениям между числами a и b:

$$\begin{array}{lcl} f(n) = O(g(n)) & \approx & a \leqslant b \\ f(n) = \Omega(g(n)) & \approx & a \geqslant b \\ f(n) = \Theta(g(n)) & \approx & a = b \\ f(n) = o(g(n)) & \approx & a < b \\ f(n) = \omega(g(n)) & \approx & a > b \end{array}$$

Теорема

Теорема 2.1. *Для любых двух функций* $f(n)$ *и* $g(n)$ *свойство* $f(n) = \Theta(g(n))$ *выполнено тогда и только тогда, когда* $f(n) = O(g(n))$ *и* $f(n) = \Omega(g(n))$.

Для любых двух функций свойства $f(n) = O(g(n))$ *и* $g(n) = \Omega(f(n))$ *равносильны.*

Стратегия решений задач по асимптотике

0. Постановка задачи: доказать, что $f(n) = \Theta(g(n))$

1. Вначале написать, что мы хотим доказать (т.е. написать неравенство)

2. Найти $\boldsymbol{n_0}$, $\boldsymbol{c_1}$ и $\boldsymbol{c_2}$

Пример: докажите, что

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

интуитивно ясно, но нужно доказать формально

Мы хотим доказать, что:

$$c_1 \times n^2 \le \frac{1}{2}n^2 - 3n \le c_2 \times n^2$$

для $\forall n \ge n_0$

Разделим все части на n^2:

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

Решение:

$$c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$$

Стратегия решений задач по асимптотике

0. Постановка задачи: доказать, что $f(n) = \Theta(g(n))$

1. Вначале написать, что мы хотим доказать (т.е. написать неравенство)

2. Найти $\boldsymbol{n_0},\ \boldsymbol{c_1}$ и $\boldsymbol{c_2}$

Пример: докажите, что

$$\boldsymbol{6n^3 \neq \Theta\left(n^2\right)}$$

интуитивно ясно, но нужно доказать формально – «от обратного»

Если утверждение истинно, то $\exists c_2, n_0$:

$$\boldsymbol{6n^3 \leq c_2 \times n^2}$$

для $\forall \boldsymbol{n} \geq \boldsymbol{n_0}$

Из этого следует, что (разделив на n^2):

$$\boldsymbol{n \leq \frac{c_2}{6}}$$

для $\forall \boldsymbol{n} \geq \boldsymbol{n_0}$

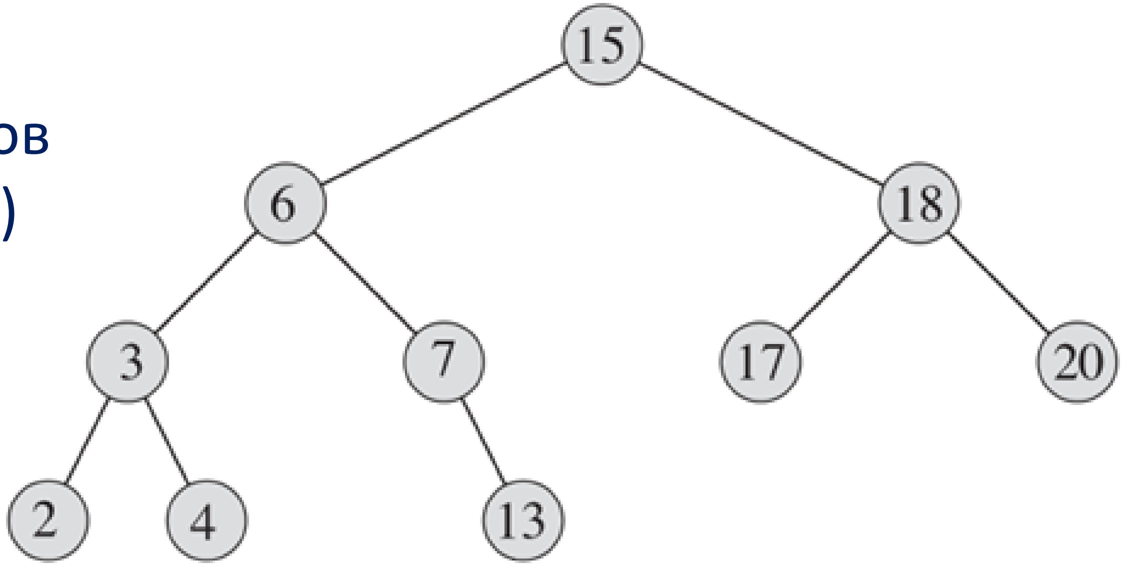
Невозможно подобрать такое c_2

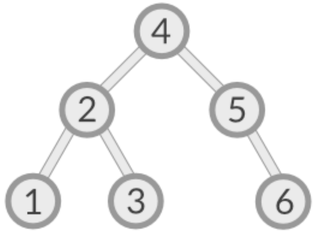
Задача решена.

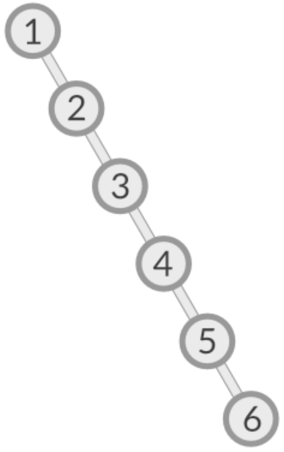
Асимптотическая оценка: двоичное дерево поиска

- Дерево состоит из узлов и ребер
- Корень дерева – один из узлов
- Ребра соединяют родителя и потомков
- У родителя может быть не более двух потомков
- В каждом узле – находится ключ (напр., число)
- Ключ левого потомка < ключа родителя
- Ключ правого потомка > ключа родителя

Корень дерева







$$T(n) = O(n)$$
$$T(n) = \Omega(\log n)$$

```
ITERATIVE-TREE-SEARCH(x, k)
  while x ≠ NIL and k ≠ x.key
      if k < x.key
          x = x.left
      else x = x.right
  return x
```

Подробнее о двоичном дереве поиска мы узнаем чуть позже

Пример: сортировка слиянием

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$, n является степенью двойки для простоты (2, 4, 8, 16, …)

Выход: Перестановка $\langle a'_1, a'_2, \dots, a'_n \rangle$ исходной последовательности, для которой $a'_1 \le a'_2 \le \dots \le a'_n$.

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Пример: сортировка слиянием

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$, n является степенью двойки для простоты (2, 4, 8, 16, …)

Выход: Перестановка $\langle a'_1, a'_2, \dots, a'_n \rangle$ исходной последовательности, для которой $a'_1 \le a'_2 \le \cdots \le a'_n$.

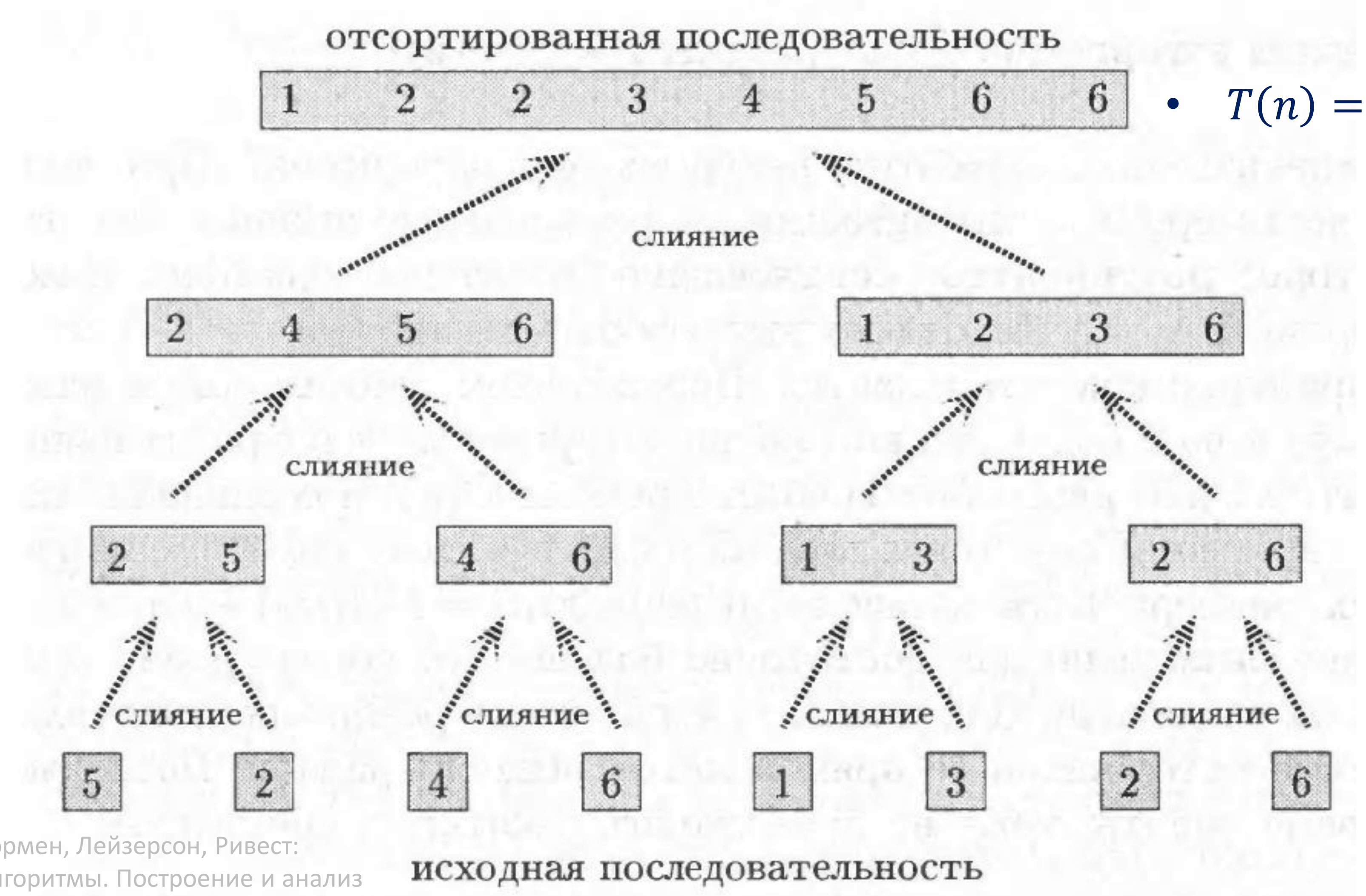
- $\log n$ уровней
- на каждом уровне $O(n)$ сравнений

Кормен, Лейзерсон, Ривест: Алгоритмы. Построение и анализ

Пример: сортировка слиянием

Вход: Последовательность n чисел $\langle a_1, a_2, \dots, a_n \rangle$,
n является степенью двойки для простоты (2, 4, 8, 16, …)

Выход: Перестановка $\langle a'_1, a'_2, \dots, a'_n \rangle$ исходной последовательности, для которой $a'_1 \le a'_2 \le \cdots \le a'_n$.



- $T(n) = O(n \log n)$

Кормен, Лейзерсон, Ривест:
Алгоритмы. Построение и анализ

Домашнее задание (не на оценку)

$O(n^2)$ – верхняя асимптотическая оценка времени работы алгоритма сортировки вставками в худшем случае для массива длины n. Докажите это.

Указание: используйте описанную ранее стратегию решений задач по асимптотике.

$$T(n) = \boxed{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)} n^2 + \boxed{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)} n - \boxed{(c_2 + c_4 + c_5 + c_8)}$$

В худшем случае

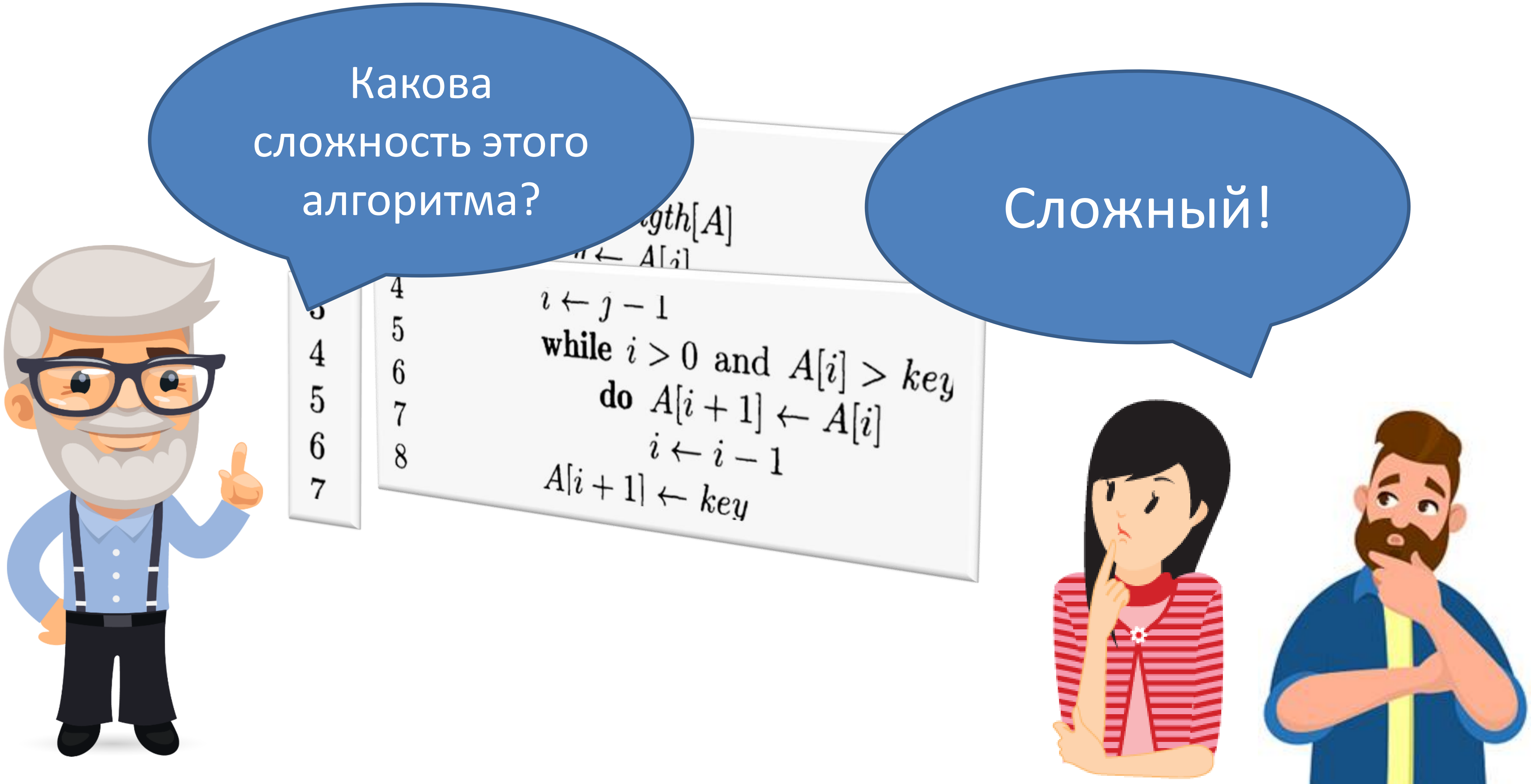
$$T(n) = \boldsymbol{a}n^2 + \boldsymbol{b}n + \boldsymbol{c} = O(n^2)$$

Проверим это

$$\mathbf{0} \leq \boldsymbol{a}n^2 + \boldsymbol{b}n + \boldsymbol{c} \leq \boldsymbol{q} \times \boldsymbol{n}^{\mathbf{2}}$$

Цель – найти $\boldsymbol{q}, \mathbf{n_0}$, при которых неравенство всегда справедливо

Сложность алгоритма



Благодарю за внимание!

Рамон Антонио Родригес Залепинос
arodriges@hse.ru

NATIONAL RESEARCH UNIVERSITY