

# Лабораторная работа 1

## Введение в анализ алгоритмов

**Курносов Михаил Георгиевич**

к.т.н. доцент Кафедры вычислительных систем  
Сибирский государственный университет  
телекоммуникаций и информатики

<http://mkurnosov.net/teaching>

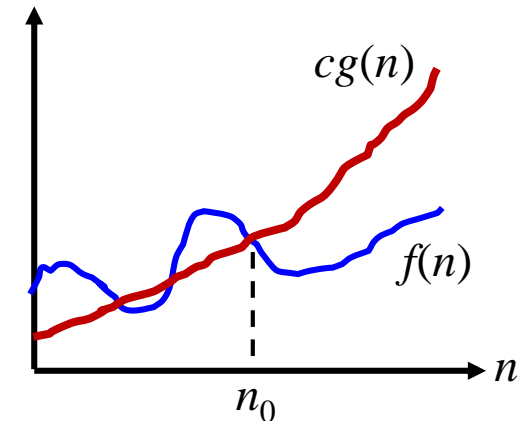
# O-обозначения

---

- Пусть  $f(n)$  – это количество операций выполняемых алгоритмом
- O-обозначение  $f(n) = O(g(n))$

$$f(n) \in O(g(n)) = \left\{ \begin{array}{l} \exists c > 0, n_0 > 0: \\ 0 \leq f(n) \leq cg(n), \forall n \geq n_0 \end{array} \right\}$$

- Существуют положительные константы  $c$  и  $n_0$  такие, что  $f(n) \leq c \cdot g(n)$  для всех  $n \geq n_0$  ( $c > 0, n_0 > 0$ )
- Функция  $f(n)$  **ограничена сверху** функцией  $g(n)$  с точность до постоянного множителя
- Используется чтобы показать, что функция (время работы алгоритма) растет не быстрее чем функция  $g(n)$



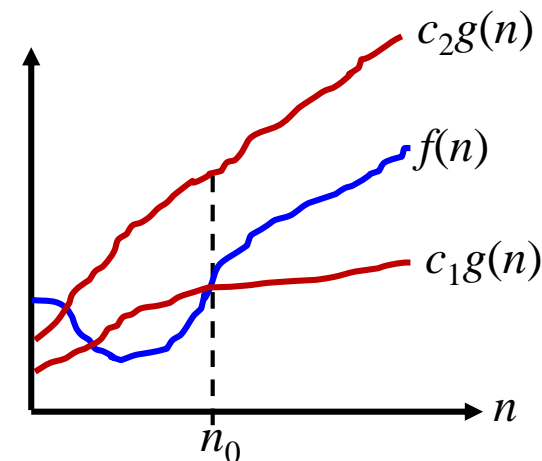
# Θ-обозначения

---

- Пусть  $f(n)$  – это количество операций выполняемых алгоритмом
- Θ-обозначение  $f(n) = \Theta(g(n))$

$$f(n) \in \Theta(g(n)) = \left\{ \begin{array}{l} \exists c_1 > 0, c_2 > 0, n_0 > 0: \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \\ \forall n \geq n_0 \end{array} \right\}$$

- Функция  $f(n)$  **ограничена снизу и сверху** функцией  $g(n)$  с точность до постоянного множителя



# Анализ вычислительной сложности алгоритма

---

1. Определяем *параметры* алгоритма, от которых зависит время его выполнения

**$n, m$**

2. Выражаем количество операций, выполняемых алгоритмом, как функцию от его *параметров* (для худшего, среднего, лучшего случаев)

$$T(n, m) = 2n^2 + 4\log_2 m$$

3. Строим *асимптотическую оценку* вычислительной сложности алгоритма – переходим к асимптотическим обозначениям  $O$ ,  $\Theta$ ,  $\Omega$

$$T(n, m) = O(2n^2 + 4\log_2 m) \\ = O(n^2)$$

# Вычисление факториала числа

---

```
function Factorial(x)
    fact = 1
    for i = 2 to x do
        fact = fact * i
    end for
    return fact
end function
```

1. Параметр, определяющий время выполнения алгоритма: **x**

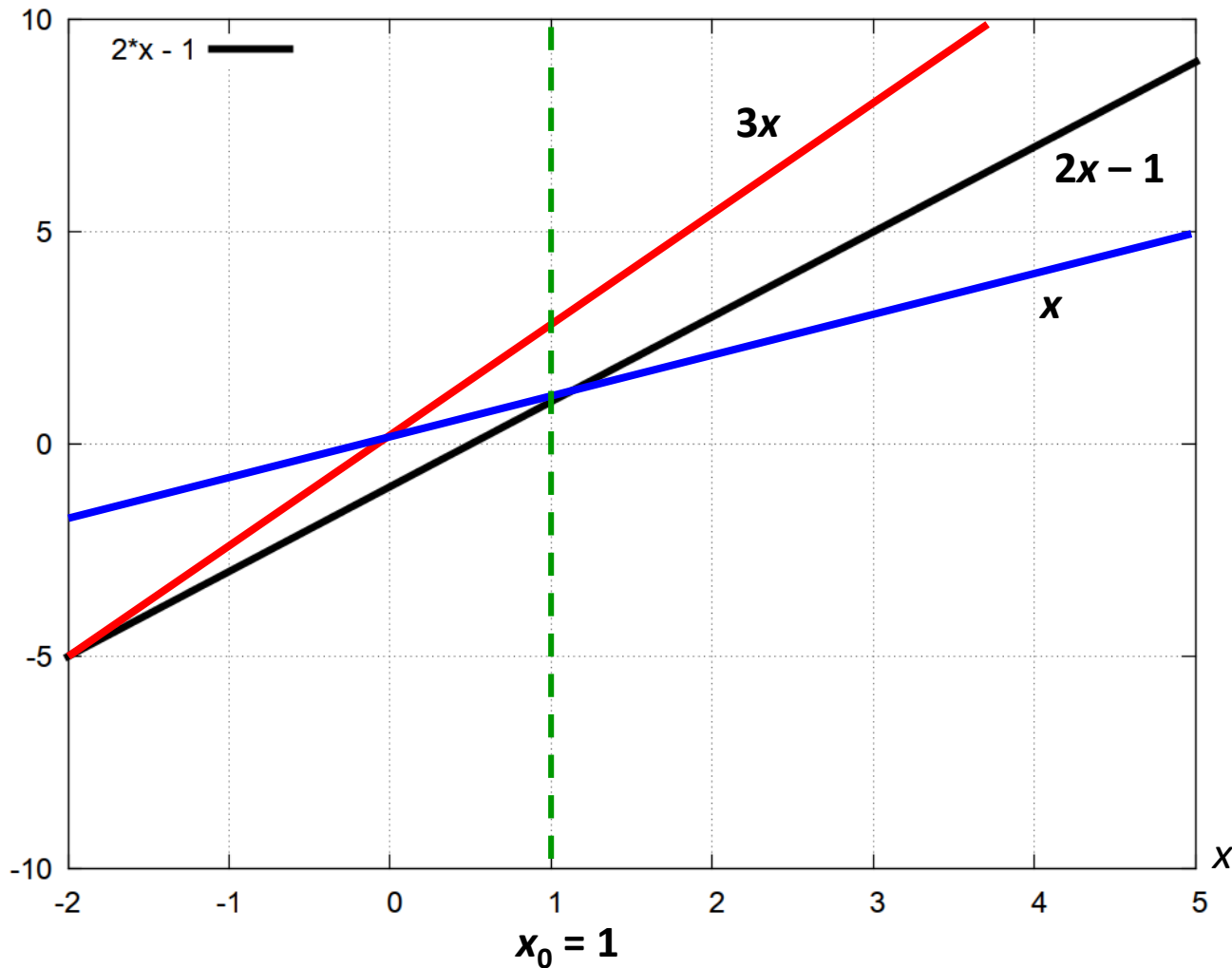
2. Количество  $T(x)$  операций выполняемых алгоритмом:

$$T(x) = 1 + 2(x - 1) = 2x - 1$$

3. Асимптотическая оценка вычислительной сложности алгоритма:

- $T(x) = O(2x - 1) = O(x)$  – асимптотическая верхняя граница;
- $T(x) = \Omega(2x - 1) = \Omega(x)$  – асимптотическая нижняя граница;
- $T(x) = \Theta(2x - 1) = \Theta(x)$  – асимптотическая точная оценка функции  $T(x)$ .

# Вычисление факториала числа



Оценка сверху:

$$2x - 1 \leq cx$$

$$c = ?$$

$$2 - 1/x \leq c$$

Начиная с  $x \geq x_0 = 1$  функция  $T(x) = 2x - 1$  ограничена сверху функцией  $3x$ , следовательно  $T(x) = O(x)$ ,  $c_0 = 3$ ,  $x_0 = 1$

# Свойства $O$ , $\Theta$ , $\Omega$

---

1.  $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$

Пример:  $n^3 + n^2 + n + 1 = O(n^3)$

2.  $O(c \cdot f(n)) = O(f(n))$

Пример:  $O(4n^3) = O(n^3)$

3.  $O(f(n)) * O(g(n)) = O(f(n) * g(n))$

Пример:  $O(n^3) * O(n) = O(n^4)$

# Сортировка выбором (Selection Sort)

```
function SelectionSort(v[1:n], n)
```

```
  for i = 1 to n do
```

```
    min = i
```

— 1 операция

```
    for j = i + 1 to n do
```

```
      if v[j] < v[min] then
```

```
        min = j
```

```
      end if
```

— 3 операции

(худший случай – worst case)

```
    end for
```

```
    if min != i then
```

```
      temp = v[i]
```

```
      v[i] = v[min]
```

```
      v[min] = temp
```

```
    end if
```

— 5 операций

```
  end for
```

*Сумма членов  
арифметической  
прогрессии*

$$T(n) = n + 5n + 3((n-1) + (n-2) + \dots + 1) = ?$$

$$T(n) = 6n + \frac{3n^2 - 3n}{2}$$



# Сортировка выбором (Selection Sort)

Асимптотическая *верхняя* граница  
вычислительной сложности:

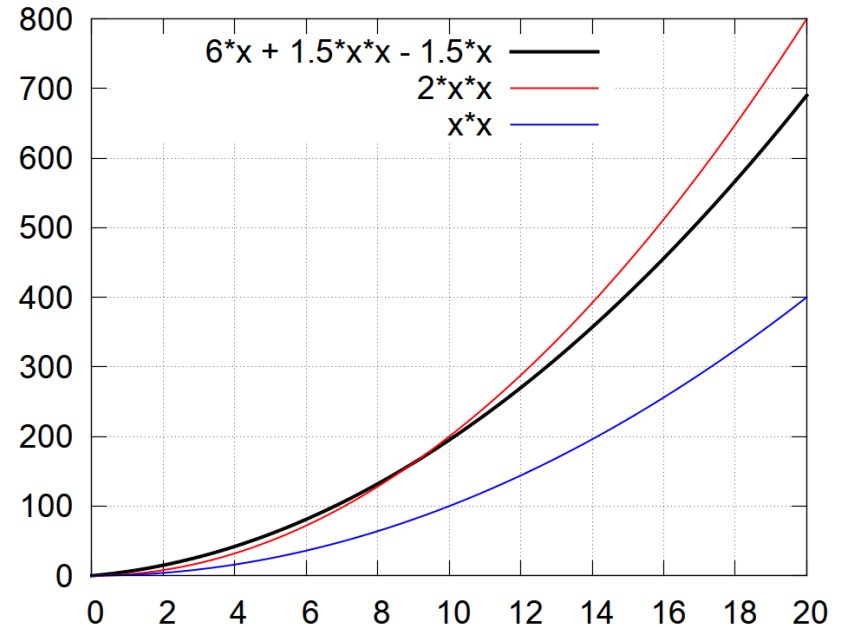
$$T(n) = O(n^2), \quad c = 2, \quad \text{при } n \geq n_0 = 9$$

Асимптотическая *нижняя* граница  
вычислительной сложности:

$$T(n) = \Omega(n^2), \quad c = 1, \quad \text{при } n \geq n_0 = ?$$

Асимптотическая *точная* оценка  
функции  $T(n)$  :

$$T(n) = \Theta(n^2), \quad c_1 = 2, \quad c_2 = 1, \quad \text{при } n \geq n_0 = ?$$



$$T(n) = 6n + \frac{3n^2 - 3n}{2}$$

# Умножение матриц

---

```
function MatrixMultiply(a[1:m][1:n], b[1:n][1:q],  
                        c[1:m][1:q])  
  
for i = 1 to m do  
    for j = 1 to q do  
        c[i][j] = 0  
        for k = 1 to n do  
            c[i][j] = c[i][j] + a[i][k] * b[k][j]  
        end for  
    end for  
end for
```

$$T(m, n, q) = mq(1 + 3n) = mq + 3mqn$$

$$T(m, n, q) = O(mq + 3mqn) = O(\max(mq, mqn)) = O(mqn)$$

# Пример с 3-я функциями

```
function first()  
    for i = 1 to n do  
        for j = 1 to n do  
            second()  
        end for  
    end for  
end function  
  
function second()  
    for i = 1 to n do  
        y = y * i - 2  
    end for  
  
    for i = 1 to n do  
        for j = 1 to 2 * n do  
            z = z + i * j  
        end for  
    end for  
end function  
  
function main()  
    for i = 1 to n do  
        x = x + i  
    end for  
    first()  
end function
```

$$T(n) = 2n + T_{first}$$

$$T_{first}(n) = n^2 \cdot T_{second}$$

$$T_{second}(n) = 3n + 6n^2$$

$$T(n) = 2n + n^2(3n + 6n^2) = 6n^4 + 3n^3 + 2n$$

$$T(n) = O(n^4)$$

# Бинарный поиск (Binary Search)

```
function BinarySearch(v[1:n], key)
  l = 1
  h = n
  while l <= h do
    mid = (l + h) / 2
    if v[mid] = key then
      return mid
    else if key > v[mid] then
      l = mid + 1
    else
      h = mid - 1
    end if
  end while
  return -1
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	3	3	5	7	9	13	23	25	29	31	33	37	41	42	46	49	50	52	67	73	81	94

# Бинарный поиск (Binary Search)

```
function BinarySearch(v[1:n], key)
  l = 1
  h = n
  while l <= h do
    mid = (l + h) / 2
    if v[mid] = key then
      return mid
    else if key > v[mid] then
      l = mid + 1
    else
      h = mid - 1
    end if
  end while
  return -1
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	3	3	5	7	9	13	23	25	29	31	33	37	41	42	46	49	50	52	67	73	81	94

key = 100

# Бинарный поиск (Binary Search)

```
function BinarySearch(v[1:n], key)
  l = 1
  h = n
  while l <= h do
    mid = (l + h) / 2
    if v[mid] = key then
      return mid
    else if key > v[mid] then
      l = mid + 1
    else
      h = mid - 1
    end if
  end while
  return -1
```

$$\frac{n}{2^1}, \frac{n}{2^2}, \dots, \frac{n}{2^k}$$

- $n = 1$  – 1 итерация
- $n = 23$  – 5 итераций
- $n = 32$  – 6 итераций

$$\frac{n}{2^k} = 1, \quad n = 2^k,$$

$$\log_2 n = \log_2 2^k = k$$

$$T(n) = 7\lceil \log_2 n \rceil + 1$$

$$T(n) = O(\log_2 n)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	3	3	5	7	9	13	23	25	29	31	33	37	41	42	46	49	50	52	67	73	81	94

# Линейный конгруэнтный генератор

---

```
function rand()  
    i = i + 1  
    r[i] = (69069 * r[i - 1] + 5) mod 232  
    return r[i]  
end function
```

$$T = 2 + 5 = 7$$

$$T = O(1)$$

# Сортировка подсчетом (Counting Sort)

---

```
function CountingSort(a[1:n], n)
  for i = 0 to k - 1 do
    c[i] = 0
  end for
  for i = 0 to n - 1 do
    c[a[i]] = c[a[i]] + 1
  end for
  for j = 1 to k - 1 do
    c[j] = c[j] + c[j - 1]
  end for
  for i = n - 1 to 0 do
    c[a[i]] = c[a[i]] - 1
    b[c[a[i]]] = a[i]
  end for
end function
```

Вычислительная сложность алгоритма?

Сложность алгоритма по памяти?