ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Анализ алгоритмов. Сортировка данных

Преподаватель:

Доцент Кафедры ВС, к.т.н.

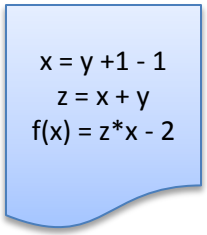
Поляков Артем Юрьевич

План лекции

1. Эффективность алгоритмов
2. Задача сортировки
3. Алгоритм сортировки вставками
4. Анализ алгоритма сортировки вставками
5. Алгоритм сортировки слиянием
6. Анализ алгоритма сортировки слиянием
7. Сравнение рассмотренных алгоритмов сортировки данных.

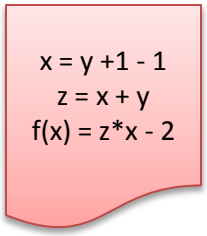
Эффективность алгоритмов

A_1
ops. $\approx c_1 \cdot n^2$
($c_1 = 2$)



C_1
ops./sec $\approx 10^4$

A_2
ops. $\approx c_2 \cdot n \cdot \log_2 n$
($c_2 = 50$)



C_2
ops./sec $\approx 10^6$

n	$t[C_1(A_2)]$, с	$t[C_2(A_1)]$, с
10^6	100	2000

Задача сортировки (sorting problem)

Дано: последовательность из n чисел $\langle a_1, a_2, a_3, \ldots, a_n \rangle$

Необходимо: переставить элементы последовательности так, чтобы для любых элементов новой последовательности $\langle a'_1, a'_2, a'_3, \ldots, a'_n \rangle$ выполнялось соотношение:

$$a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n$$

Пример:

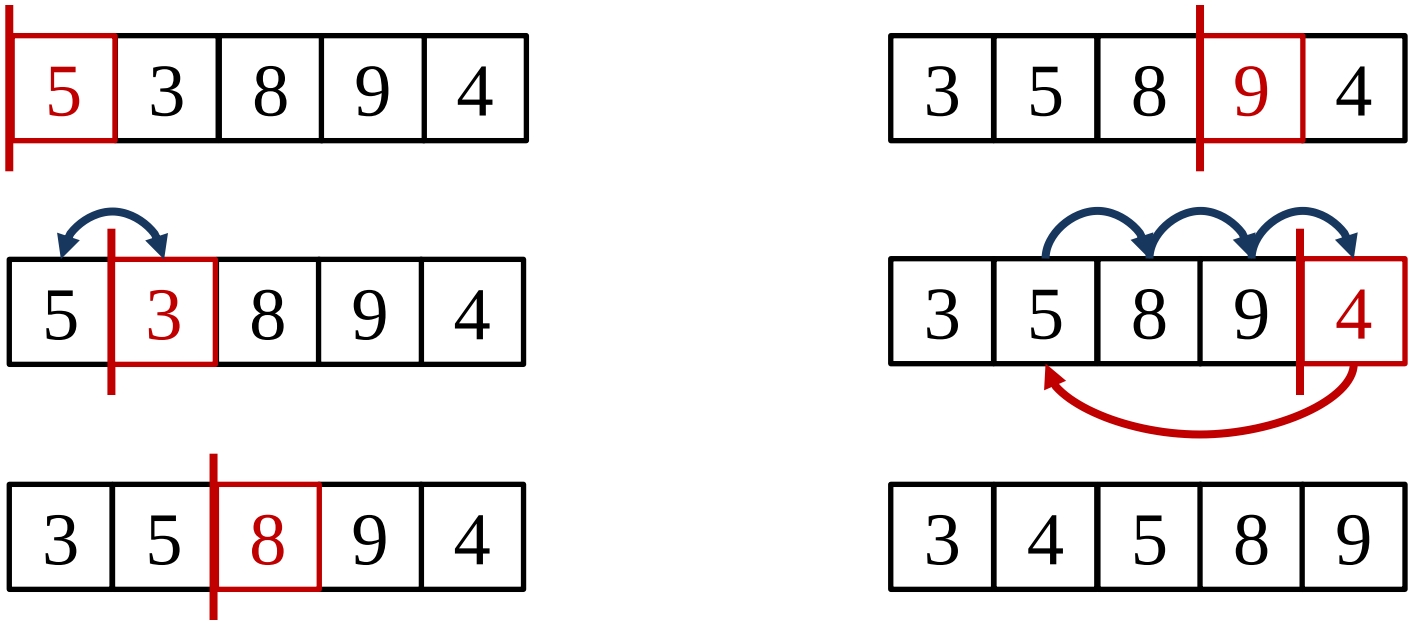
Входная последовательность: ‹5, 3, 8, 9, 4›

Выходная последовательность: ‹3, 4, 5, 8, 9›

Входные данные (последовательность), удовлетворяющие всем заданным ограничениям задачи, называется **экземпляром задачи**.

Метод сортировки вставками

- Обработка элементов последовательности производится слева направо
- Для очередного элемента выполняется поиск его места в отсортированной части массива.
- Поиск осуществляется путем поэлементного сравнения справа налево.



Правила оформления псевдокода

1. Циклические конструкции обозначаются английскими словами (аналогично языку программирования СИ): **while**, **do-while**, **for.**

2. Конструкция ветвления обозначается ключевыми словами **if-then-else**.

3. Структура блоков указывается с помощью отступов.

4. Для описания комментариев используется '//'

5. Присваивание обозначается как '←': $x \leftarrow y$.

6. Переменные являются локальными для той процедуры, в которой они используются. Глобальные переменные указываются явным образом.

Алгоритм сортировки вставками

```
INSERTION_SORT(A)
1    for j ← 2 to len(A) do
2        k ← A[j]
3        // Сдвиг всех элементов A справа от j и  больше A[j]
4        i ← j – 1
5        while i > 0  и A[i] > k do
6            A[i+1] = A[i]
7            i ← i – 1
8        A[i+1] = k
```

Инвариант цикла

Инвариант цикла – это логическое выражение, зависящее от задачи, решаемой при помощи рассматриваемого цикла.

Инварианты циклов используются в теории верификации (проверки) алгоритмов для доказательства правильности выполнения цикла.

Инварианты корректных алгоритмов обладают следующими свойствами:

Инициализация. Инвариант справедлив перед инициализацией цикла

Сохранение. Если инвариант выполняется перед очередной итерацией цикла, то он также выполняется после нее.

Завершение. По завершении цикла инварианты позволяют убедиться в корректности алгоритма.

Корректность алгоритма сортировки вставками

Инвариант: на j-й итерации цикла массив A[1…(*j*–1)] состоит из исходных элементов, расположенных в порядке возрастания.

Инициализация. На первой итерации $j = 2$, массив A[1…1] состоит из одного исходного элемента, расположенного по возрастанию.

Сохранение. на *j*-й итерации элементы массива *A*[*j*–1], *A*[j–2] … *A*[*j*–*m*] сдвигаются до тех пор, пока не будет найдено место для *A*[*j*], куда он и будет помещен. Инвариант – истинный.

```
INSERTION_SORT(A)
1  for j ← 2 to len(A) do
2      k ← A[j]
3      // comment
4      i ← j – 1
5      while i > 0  и A[i] > k do
6          A[i+1] = A[i]
7          i ← i – 1
8      A[i+1] = k
```

Завершение. На *j*+1 итерации массив A[1…j] состоит из исходных элементов, расположенных по возрастанию.

Временная сложность алгоритма сортировки вставками

Время работы алгоритма – сумма промежутков времени, необходимых для выполнения каждой инструкции.

INSERTION_SORT(*A*)		Время	Кол-во
1	**for** $j \leftarrow 2$ to *len*(*A*) **do**	c_1	n
2	$k \leftarrow A[j]$	c_2	$n-1$
3	// Сдвиг	0	
4	$i \leftarrow j-1$	c_3	$n-1$
5	**while** $i > 0$ и $A[i] > k$ **do**	c_4	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i-1$	c_6	
8	$A[i+1] = k$	c_7	$n-1$

t_j – количество элементов, которое потребовалось передвинуть для того, чтобы найти место для $A[j]$.

Временная сложность алгоритма сортировки вставками (благоприятный случай)

Входная последовательность упорядочена по возрастанию. Тогда количество (t_j) шагов, которое потребовалось для того, чтобы найти место для $A[j]$, во все случаях равно 1: $t_j = 1$.

	INSERTION_SORT(A)	Время	Кол-во
1	**for** $j \leftarrow 2$ to $len(A)$ **do**	c_1	n
2	$k \leftarrow A[j]$	c_2	$n-1$
3	// Сдвиг	0	
4	$i \leftarrow j-1$	c_3	$n-1$
5	**while** $i > 0$ и $A[i] > k$ **do**	c_4	$\sum_{j=2}^{n} t_j$
6	$A[j+1] = A[j]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i-1$	c_6	
8	$A[i+1] = k$	c_7	$n-1$

$$T(n) = c_1 n + (c_2+c_3+c_7)(n-1) + c_4(n-1)$$

Временная сложность алгоритма сортировки вставками (худший случай)

A упорядочена по убыванию, $t_j = j$.

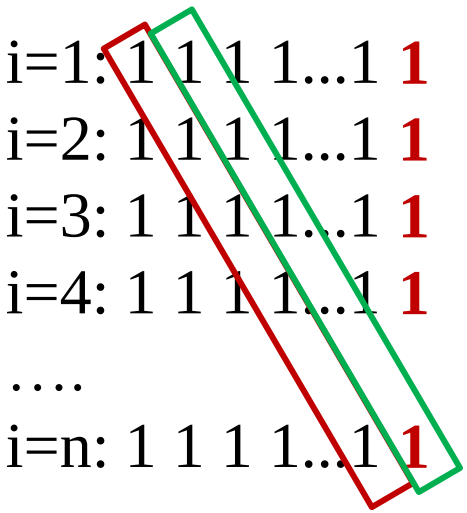
INSERTION_SORT(*A*)	Время	Кол-во
1 **for** $j \leftarrow 2$ to *len*(*A*) **do**	c_1	n
2 $k \leftarrow A[j]$	c_2	$n-1$
3 // Сдвиг	0	
4 $i \leftarrow j-1$	c_3	$n-1$
5 **while** $i > 0$ и $A[i] > k$ **do**	c_4	$\sum_{j=2}^{n} t_j$
6 $A[j+1] = A[j]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_6	
8 $A[i+1] = k$	c_7	$n-1$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j-1)$$

Сумма ряда $\sum_j j$ (графическое док-во)

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = (n^2 + n)/2$$

1) Графически



$S_{кв}$: n^2,
$S_{тр}$: $n^2/2$

ДИАГОНАЛЬ!

1) $n^2/2$ включает лишь половину диагонали
2) длина диагонали: n
3) необходимо включить диагональ дважды
4) получаем прямоугольник со сторонами: $(n+1)$ и n.

Искомая площадь равна половине его площади: $(n+1)\cdot n/2$

Метод математической индукции

Суть метода:

1. Доказать, что утверждение P справедливо для некоторого тривиального случая. Например в рассматриваемой задаче при n=1 => P(1).

2) Доказать, что если справедливо $P(n\text{-}1)$, то справедливо $P(n)$

Если доказано, что пп. 1 и 2 верны, то:

P(1) – спр. => P(2) – спр. => P(3) - спр. => … => P(n-1) – спр. => P(n) - спр.

Сумма ряда $\sum_{j=1\ldots n} j$ (метод мат. индукции)

$$P(n): \sum_{i=1}^{n} i = 1 + 2 + 3 \ldots + n = (n^2 + n)/2$$

Доказательство:

1) $P(1)$:

$n=1$, $(n+1)\cdot n/2 = 1$

2) $P(n)$:

Пусть справедливо $P(n-1)$, тогда:

$$\sum_{i=1}^{n-1} i = (n-1+1)\cdot(n-1)/2 = (n^2 - n)/2$$

3) Учитывая, что:

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$$

4) Получим: $$P(n) = P(n-1) + n = \frac{(n^2 - n)}{2} + n = \frac{(n^2 + n)}{2}$$

Временная сложность алгоритма сортировки вставками (худший случай, повтор)

A упорядочена по возрастанию, $t_j = j$.

INSERTION_SORT(*A*)	Время	Кол-во
1 **for** $j \leftarrow 2$ to *len*(*A*) **do**	c_1	n
2 $k \leftarrow A[j]$	c_2	$n-1$
3 // Сдвиг	0	
4 $i \leftarrow j-1$	c_3	$n-1$
5 **while** $i > 0$ и $A[i] > k$ **do**	c_4	$\sum_{j=2}^{n} t_j$
6 $A[j+1] = A[j]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_6	
8 $A[i+1] = k$	c_7	$n-1$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j-1)$$

Временная сложность алгоритма сортировки вставками (худший случай) (2)

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j-1) =$$

$$\left| \begin{array}{l} \sum_{j=2}^{n} j = \left(\sum_{j=1}^{n} j \right) - 1 = \frac{n+1}{2} n - 1 \\ \sum_{j=2}^{n} (j-1) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} \end{array} \right| =$$

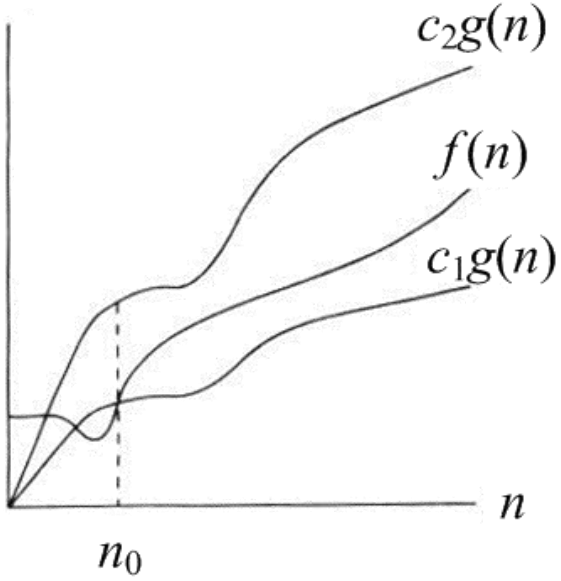
$$(c_1 + c_2 + c_3 + c_7) n - (c_2 + c_3 + c_7) + c_4 \left(\frac{1}{2} n^2 + \frac{1}{2} n - 1 \right) +$$

$$(c_5 + c_6) \left(\frac{1}{2} n^2 - \frac{1}{2} n \right) =$$

$$\left(\frac{c_4 + c_5 + c_6}{2} \right) n^2 + \left(c_1 + c_2 + c_3 + c_7 + \frac{c_4 - (c_5 + c_6)}{2} \right) n - \left(c_2 + c_3 + c_7 + c_4 \right) =$$

$$T_{IS}(n) = an^2 + bn + c$$

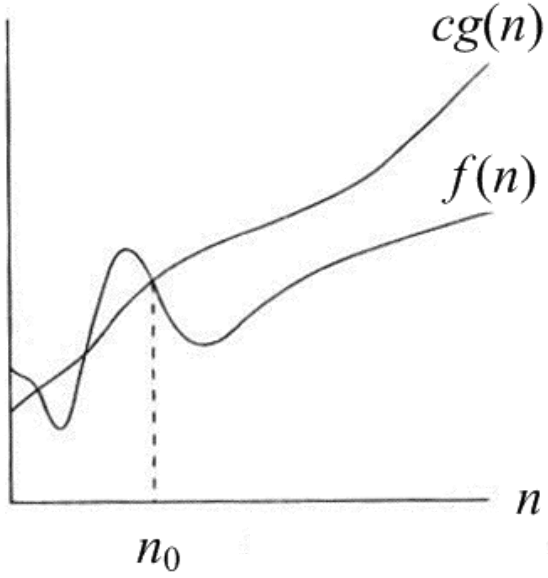
Асимптотические обозначения



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

или

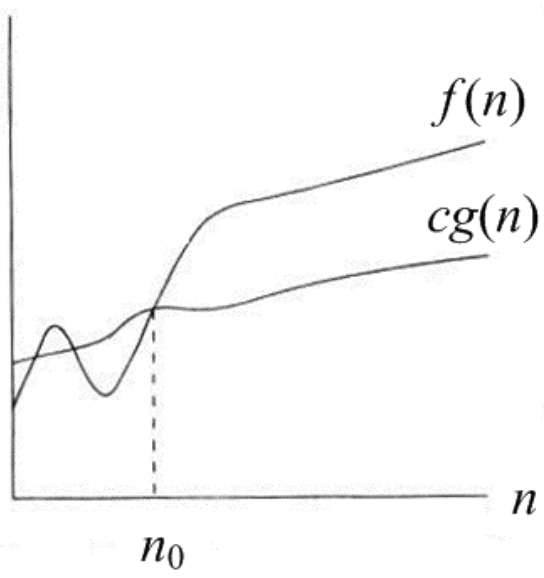
$f(n) = \Theta(g(n))$



$f(n) \in O(g(n))$

или

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

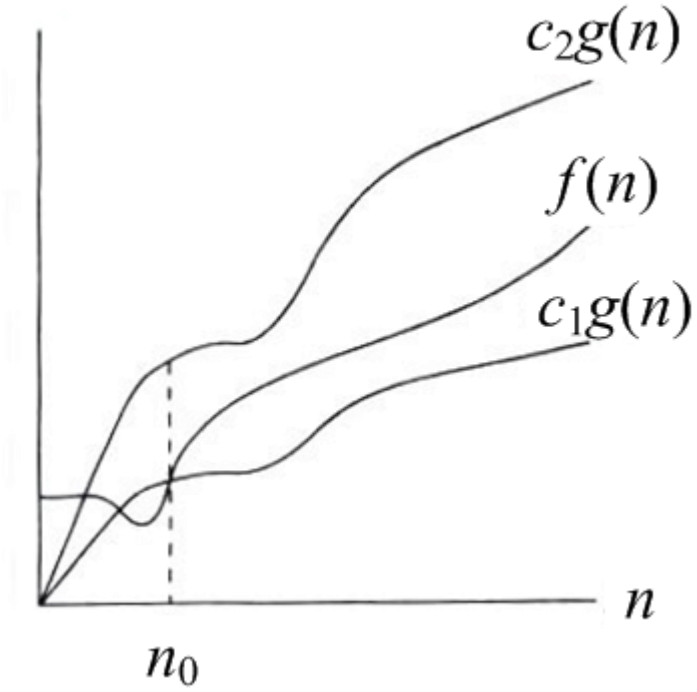


$f(n) \in \Omega(g(n))$

или

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

Θ – обозначения



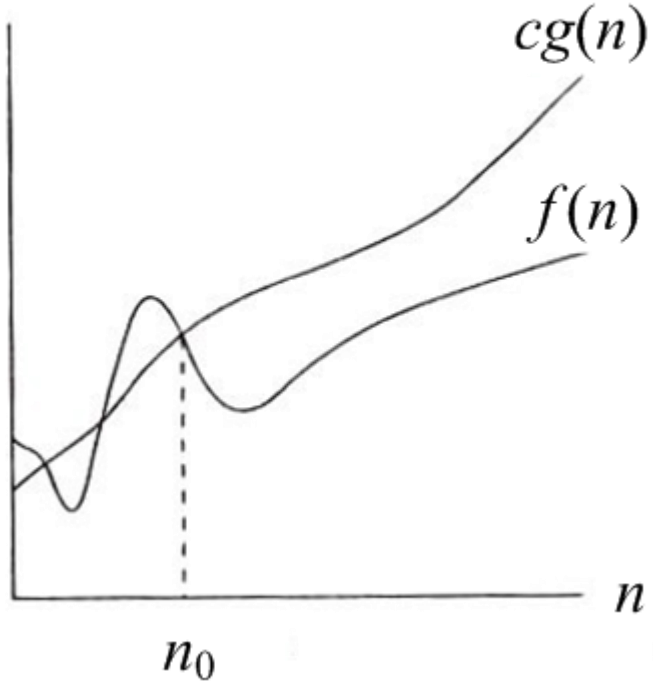
$f(n) \in \Theta(\, g(n)\,)$

или

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

Существуют такие константы c_1, c_2 и значение n_0, что $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

O – обозначения (O большое)



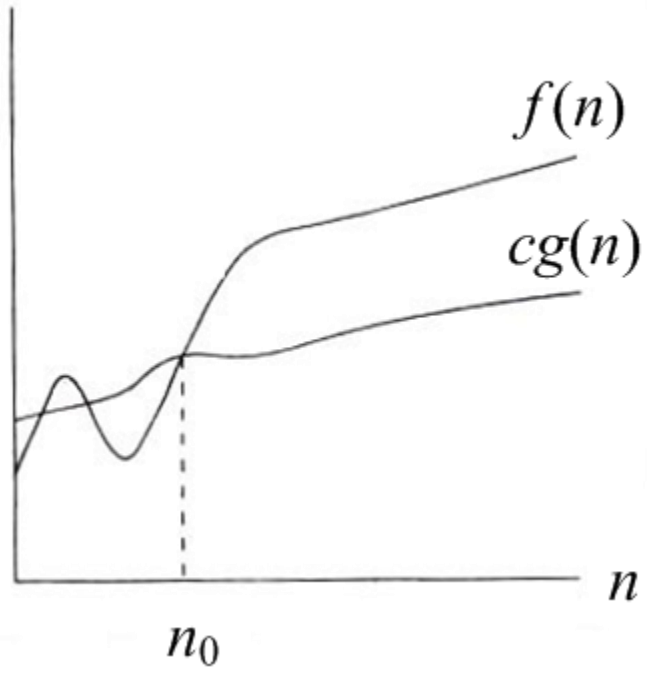
$f(n) \in O(\ g(n)\ )$

или

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

Существует такая константа c и значение n_0,
что $f(n) \leq c \cdot g(n)$

Ω – обозначения



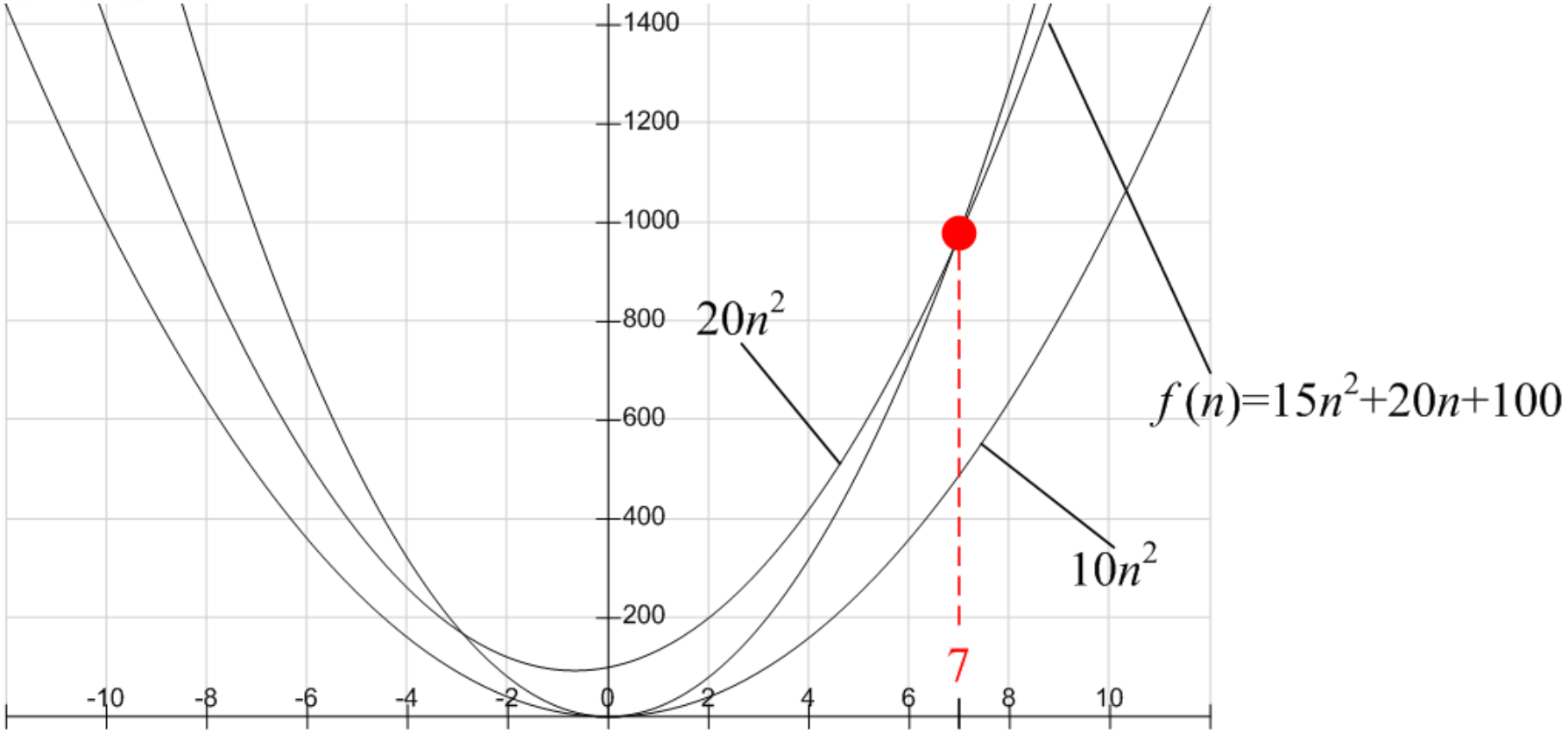
$f(n) \in \Omega(g(n))$

или

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

Существует такая константа c и значение n_0,
что $c \cdot g(n) \leq f(n)$

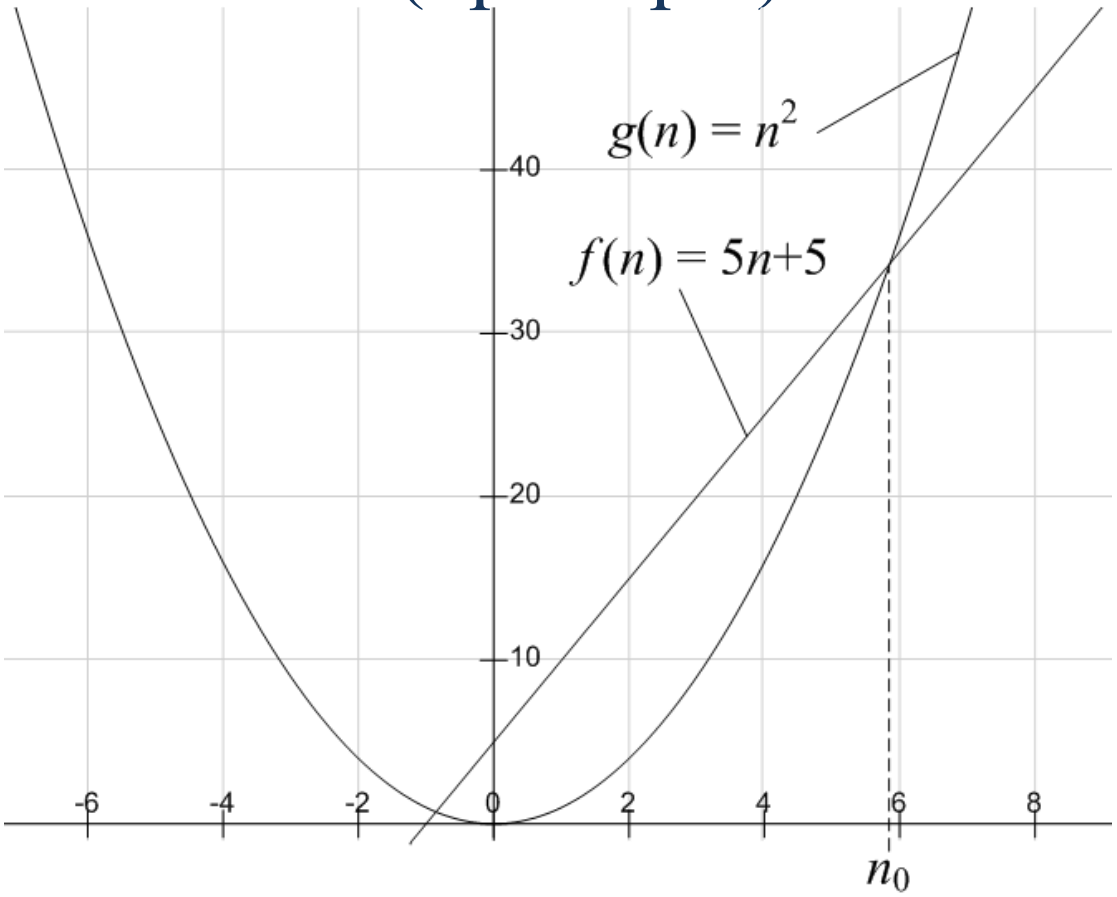
Асимптотические обозначения (пример 1)



$f(n) = \Theta(n^2)$, т.к. существуют константы $c_1 = 10$ и $c_2 = 20$, для которых начиная с $n = 7$ справедливо, что $10n^2 \le f(n) \le 20n^2$

Если $f(n) = \Theta(n^2)$ => $f(n) = O(n^2)$ и $f(n) = \Omega(n^2)$. **Обратное не верно**.

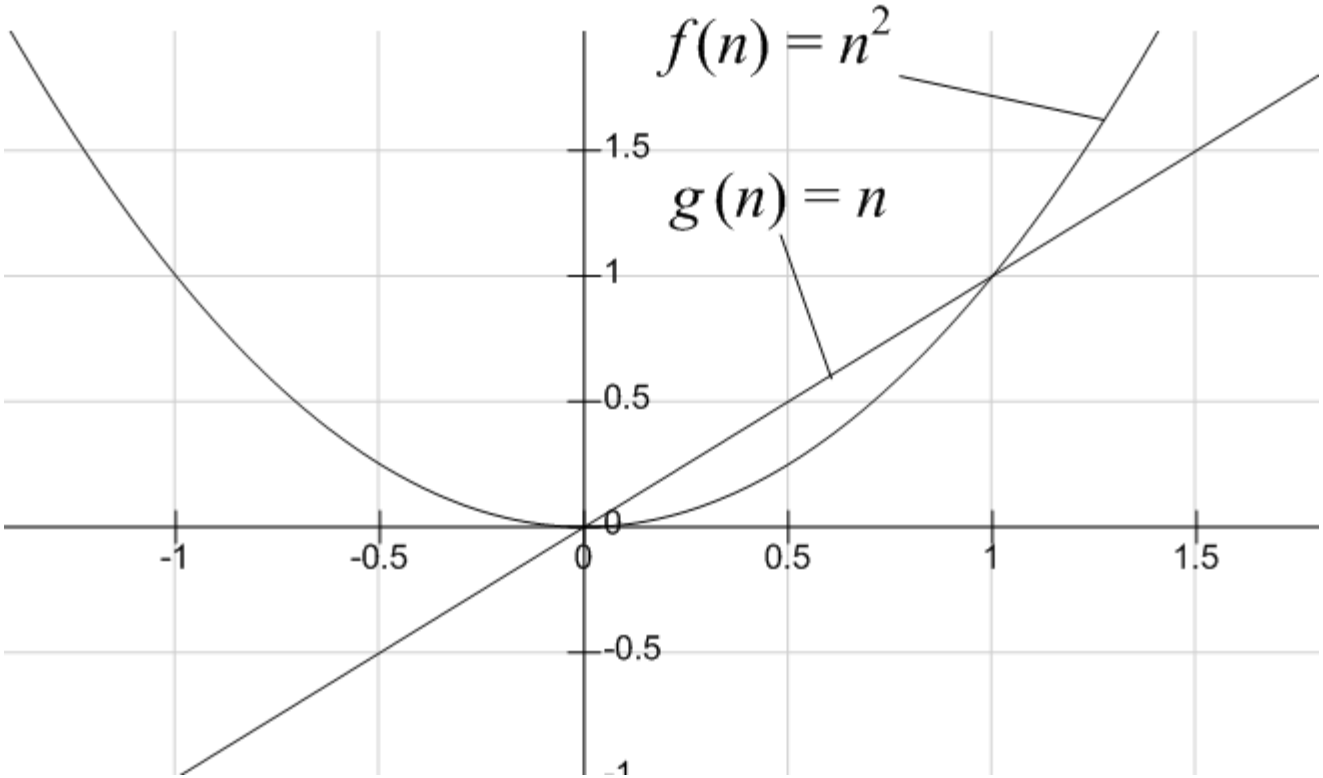
Асимптотические обозначения (пример 2)



$f(n) = O(n^2)$, т.к. существует константа $c = 1$, для которой начиная с $n \approx 6$ справедливо, что при $n > 6 \Rightarrow 5n + 5 \leq n^2$

Однако нет такой константы c' и n_0, что при $n > n_0 \Rightarrow 5n + 5 \geq c'n^2$

Асимптотические обозначения (пример 3)



$f(n) = n^2 = \Omega(n)$, т.к. существует константа $c = 1$, для которой начиная с $n = 1$ справедливо, что при $n \geq 1$ => $n^2 \geq n$

Однако нет такой константы c' и n_0, что при $n > n_0$ => $n^2 \leq c'n$

поэтому $\boldsymbol{f(n) = n^2 \neq O(n)}$ => $\boldsymbol{f(n) = n^2 \neq \Theta(n^2)}$

Определить асимптотические отношения между функциями $f_1(n)$ и $f_2(n)$

$f_1(n)$	$f_2(n)$	O	Ω	Θ
n	n^2	$n = O(n^2)$	$n^2 = \Omega(n)$	–
$\lg n$	n			
2^n	$2^{n/2}$			
$n^{\lg c}$	$n^{\lg n}$			
$n!$	n^n			
$\sqrt{n}$	$n^{\sin n}$			
$\lg(n!)$	$\lg(n^n)$			
ax^2+bx	x^2			

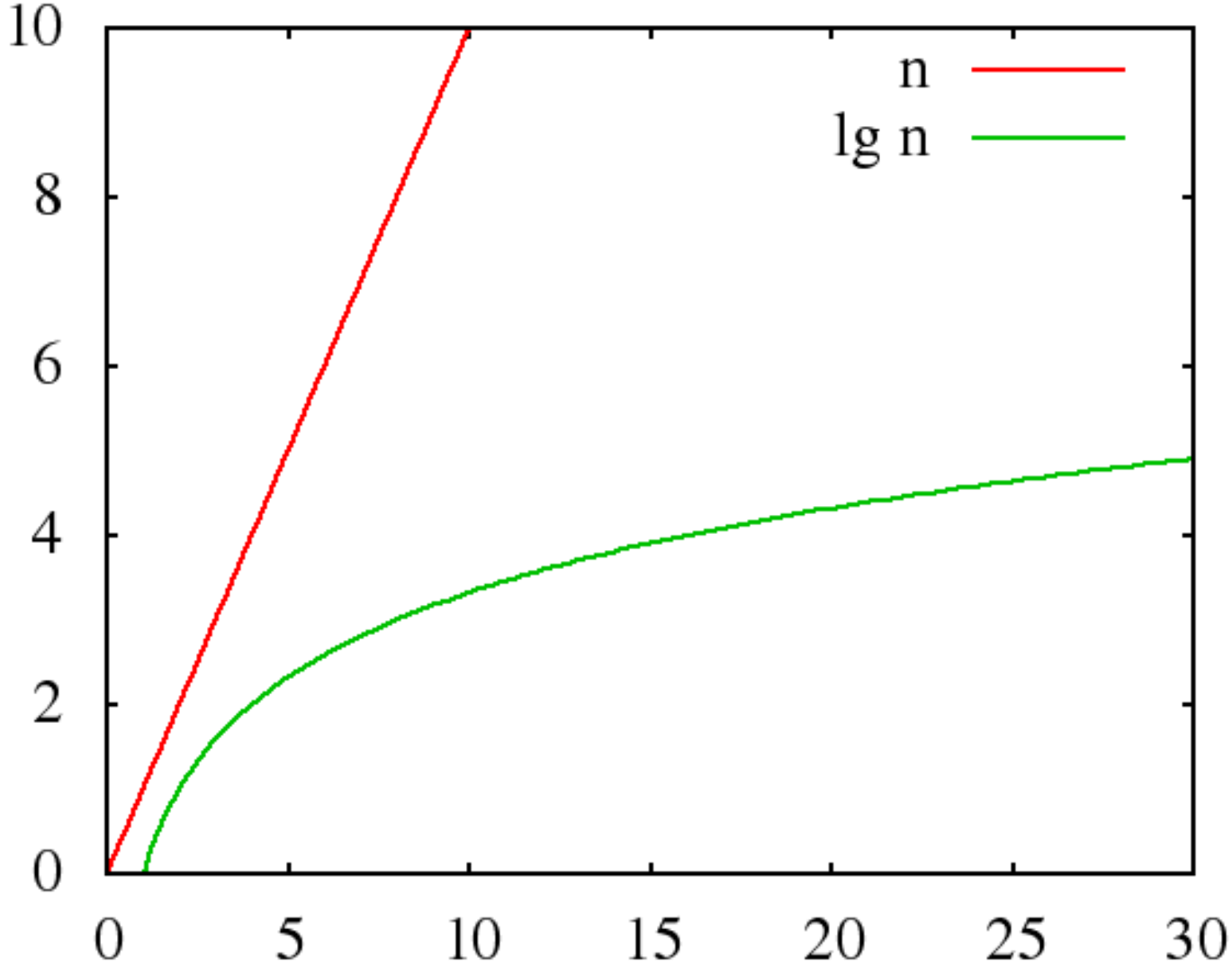
$$\lg n = \log_2 n$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

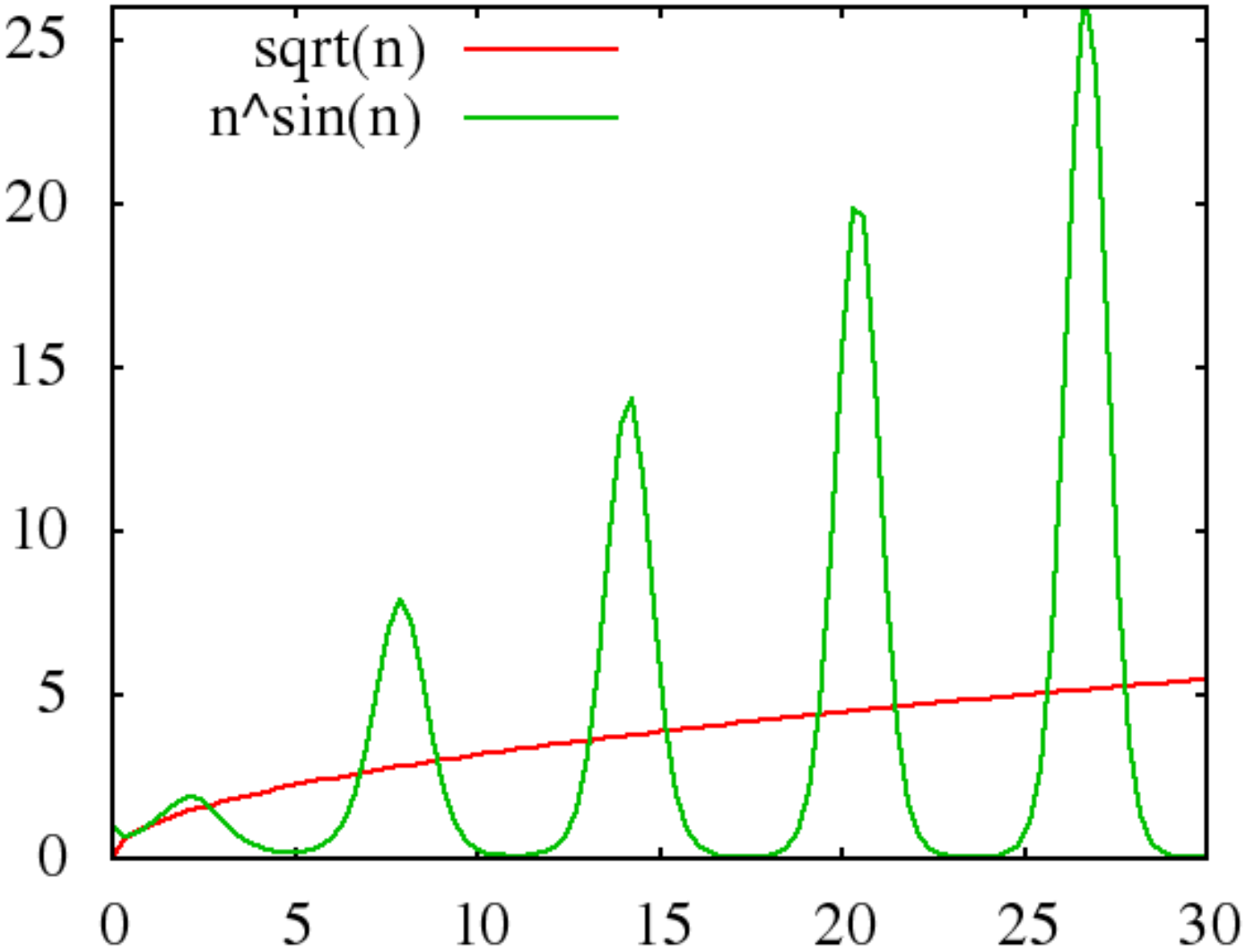
Определить асимптотические отношения между функциями $f_1(n)$ и $f_2(n)$

$f_1(n)$	$f_2(n)$	O	Ω	Θ
n	n^2	$n = O(n^2)$	$n^2 = \Omega(n)$	–
$\lg n$	n	$\ln n = O(n)$	$n = \Omega(\ln n)$	–
2^n	$2^{n/2}$	$2^{n/2} = O(2^n)$	$2^n = \Omega(2^{n/2})$	–
$n^{\lg c}$	$n^{\lg n}$	$n^{\lg n} = O(n^{\lg c})$	$n^{\lg c} = \Omega(n^{\lg n})$	–
$n!$	n^n	$n! = O(n^n)$	$n^n = \Omega(n!)$	–
$\sqrt{n}$	$n^{\sin n}$	–	–	–
$\lg(n!)$	$\lg(n^n)$	$\lg n! = O(\lg n^n)$	$\lg n^n = O(\lg n!)$	–
an^2+bn	n^2	$an^2+bn = O(n^2)$	$an^2+bn = \Omega(n^2)$	$an^2+bn = \Theta(n^2)$

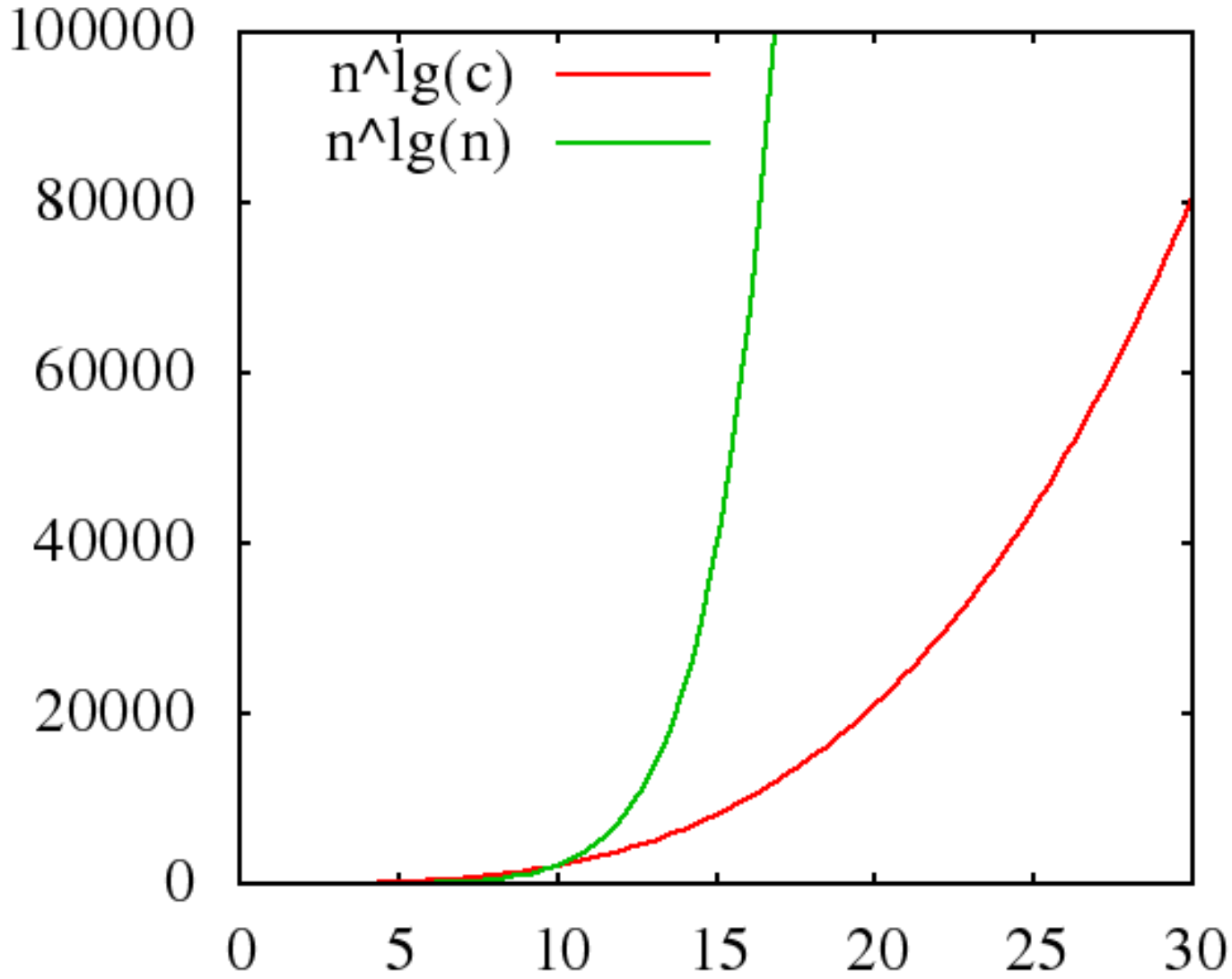
Определить асимптотические отношения между функциями $\lg n$ и n



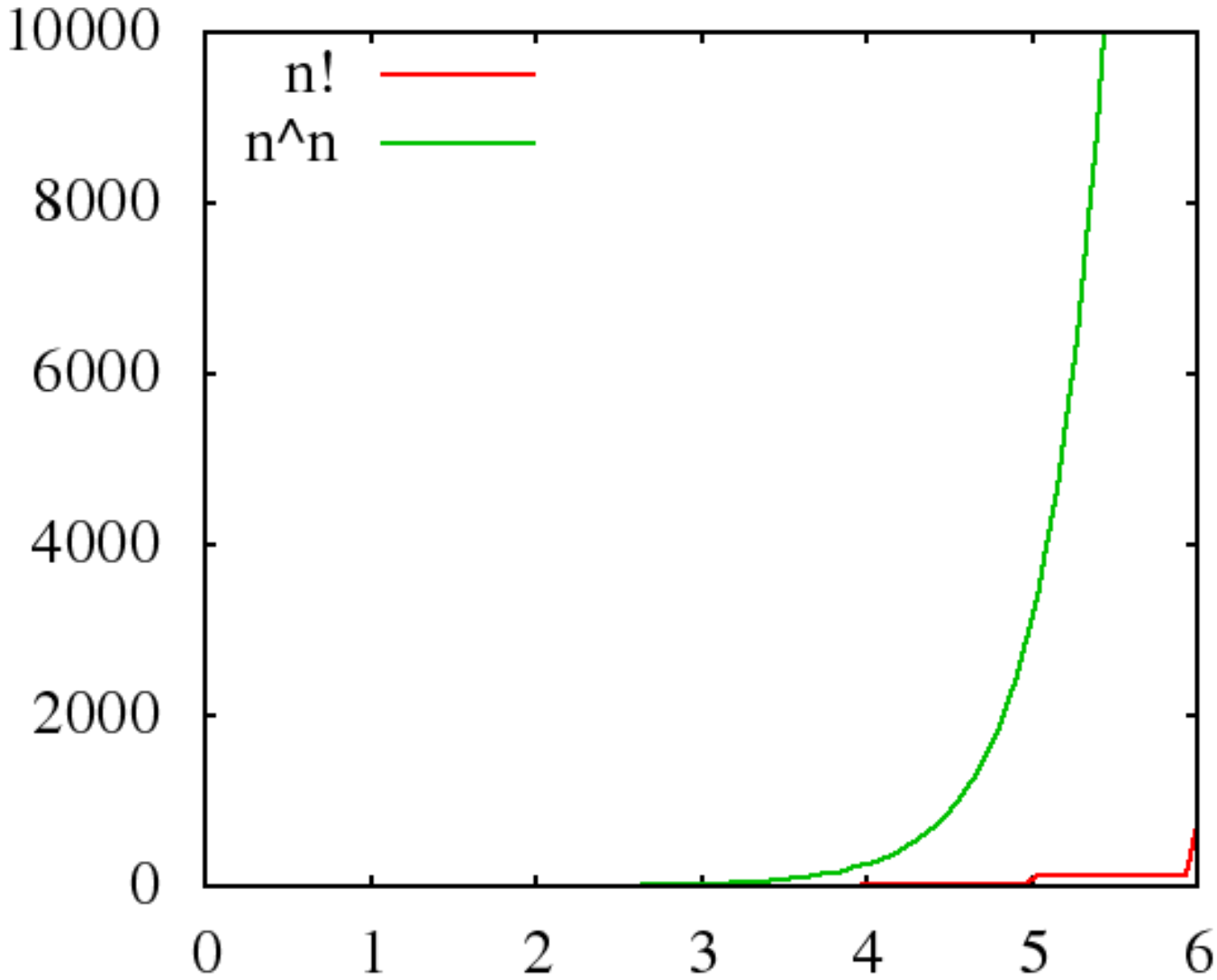
Определить асимптотические отношения между функциями $\sqrt{n}$ и $n^{\sin n}$



Определить асимптотические отношения между функциями $n^{\lg c}$ и $n^{\lg n}$



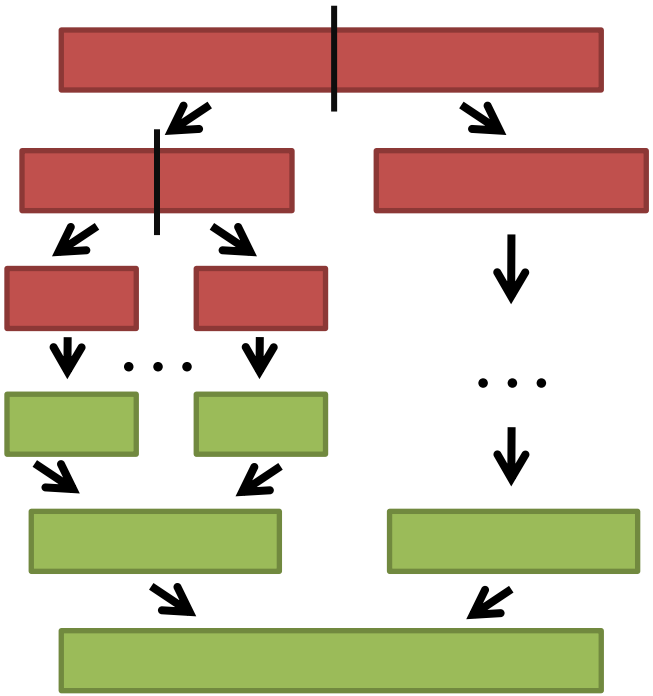
Определить асимптотические отношения между функциями $n!$ и n^n



Разработка алгоритмов

Инкрементный подход

Известно, что часть массива отсортирована (зел.). Вставка нового элемента (голуб.) осуществляется так, чтобы массива остался отсортированным.



Метод декомпозиции (разбиения)

Исходная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем.

Далее каждая из подзадач рекурсивно решается тем же алгоритмом.

После этого полученные решения подзадач комбинируются в решение исходной задачи.

Метод декомпозиции (метод "разделяй и властвуй")

На каждом уровне рекурсии выполняются следующие шаги:

Разделение (декомпозиция) задачи на несколько подзадач.

Покорение – рекурсивное решение подзадач. Если подзадачи достаточно малы каждая из них решается непосредственно.

Комбинирование – решение исходной задачи из решений вспомогательных задач.

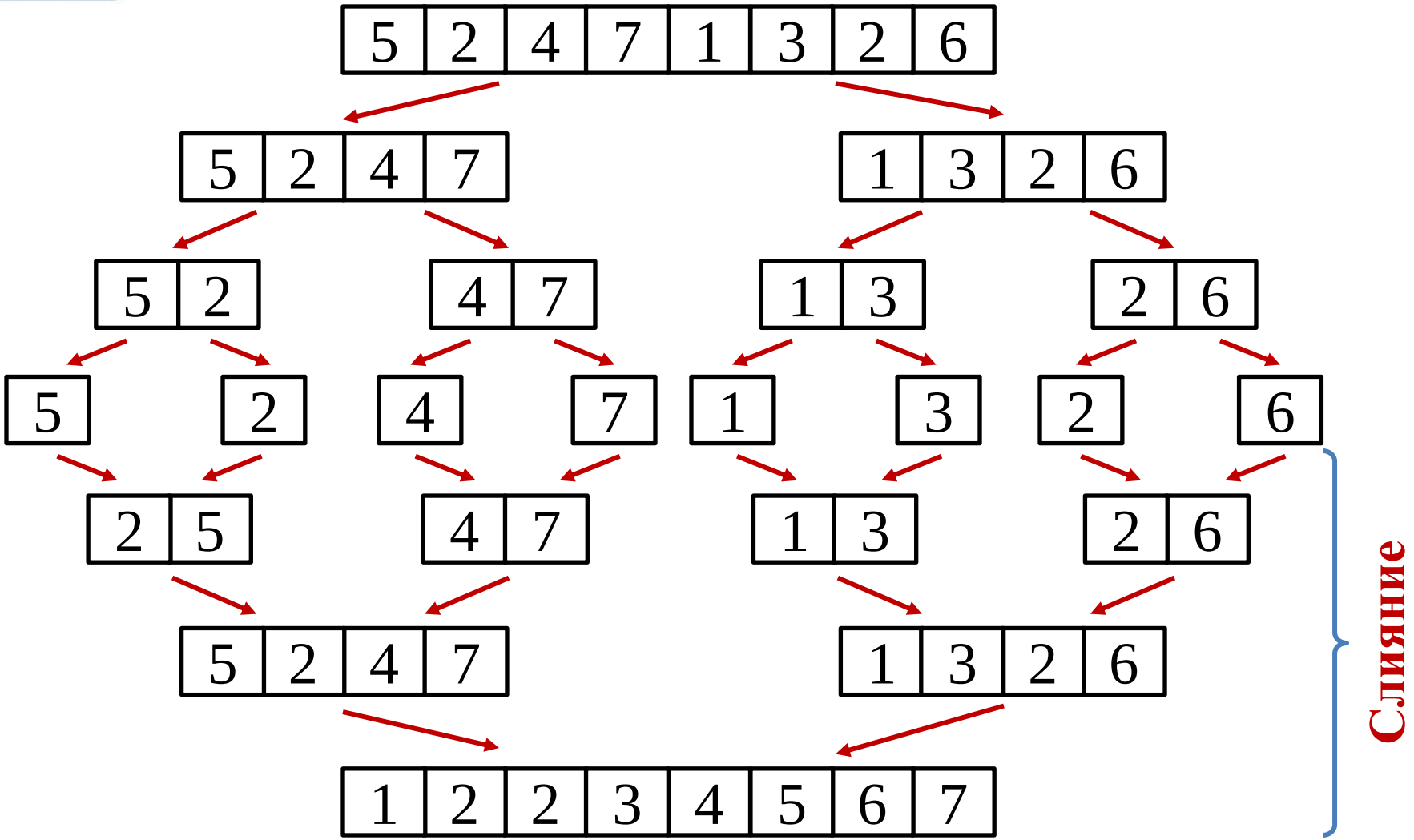
Алгоритм сортировки слиянием

Разделение входная последовательность из n элементов разбивается на две (под)последовательности по $n/2$ элементов.

Покорение – каждая из (под)последовательностей методом слияния.

Комбинирование – слияние отсортированных (под)последовательностей для получения окончательного результата.

Пример


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
2 5
4 7
1 3
2 6
5 2 4 7
1 3 2 6
1 2 2 3 4 5 6 7
Слияние

Алгоритм сортировки слиянием

Разделение входная последовательность из n элементов разбивается на две (под)последовательности по $n/2$ элементов.

Покорение – каждая из (под)последовательностей методом слияния.

Комбинирование – слияние отсортированных (под)последовательностей для получения окончательного результата.

```
MERGE_SORT(A, l, r)
1   if l < r then
2       k ← [(r + l)/2]
3       MERGE_SORT(A, r, k)
4       MERGE_SORT(A, k, r)
5       MERGE(A, l, k, r)
```

Слияние последовательностей

MERGE(A, l, k, r)

1 $n_1 \leftarrow k - l + 1$, $n_2 \leftarrow r - k$

2 **for** $i \leftarrow 1$ **to** n_1 **do** $L[i] \leftarrow A[l + i - 1]$ // $L = A[l \ldots k]$

3 **for** $j \leftarrow 1$ **to** n_2 **do** $R[j] \leftarrow A[k + j]$ // $R = A[(k + 1) \ldots r]$

4 $i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$

5 **while** $i \leq n_1$ и $j \leq n_2$ **do** // Слияние пока L и R не пусты

6 **if** $L[i] \leq R[j]$ **then** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$

7 **else** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$

8 $k \leftarrow k + 1$

9 // Дописываем в А массив, оставшийся не пустым

10 **while** $i \leq n_1$ **do** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$, $k \leftarrow k + 1$

11 **while** $j \leq n_2$ **do** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$, $k \leftarrow k + 1$

Вычислительная сложность метода декомпозиции

$$T_{DEC}(n) = \begin{cases} c, & n = 1 \\ aT_{DEC}(n/a) + T_D(n) + T_C(n), & n > 1 \end{cases}$$

где $T_{DEC}(n)$ – вычислительная сложность метода декомпозиции, $T_D(n)$ – вычислительная сложность **разбиения**,

$a \cdot T_{DEC}(n/a)$ – вычислительная сложность **покорения** (в общем случае входные данные разбиваются на a частей, каждая из которых обрабатывается **тем же** алгоритмом)

$T_C(n)$ – вычислительная сложность **комбинирования**.

Анализ алгоритма сортировки слиянием

Разделение Разбиение входной последовательности длины n пополам требует фиксированного времени, на зависит от n.

Покорение – $2 \cdot T_{MS}(n/2)$ – решение двух подзадач алгоритмом MS.

Комбинирование – вычислительную сложность данного шага обозначим через $T_M(n)$, ее анализ будет выполнен отдельно.

MERGE_SORT(A, l, r)			$T_{MS}(r - l + 1)$
1	**if** $l < r$ **then**	c_1	$O(1)$
2	$k \leftarrow [(r + l)/2]$	c_2	$O(1)$
3	MERGE_SORT(A, r, k)	c_3	$T_{MS}(k - l + 1)$
4	MERGE_SORT(A, k+1, r)	c_4	$T_{MS}(r - k)$
5	MERGE(A, l, k, r)	c_5	$T_M(r - l + 1)$

Вычислительная сложность процедуры слияния

	MERGE(A, l, k, r)	$O(f(n))$
1	$n_1 \leftarrow k - l + 1$, $n_2 \leftarrow r - k$	?
2	**for** $i \leftarrow 1$ **to** n_1 **do** $L[i] \leftarrow A[l + i - 1]$ // L=$A[l \ldots k]$	?
3	**for** $j \leftarrow 1$ **to** n_2 **do** $R[j] \leftarrow A[k + j]$ // $R = A[(k + 1)\ldots r]$	?
4	$i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$	?
5	**while** $i \le n_1$ и $j \le n_2$ **do** // Слияние пока L и R не пусты	?
6	**if** $L[i] \le R[j]$ **then** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$	?
7	**else** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$	?
8	$k \leftarrow k + 1$	?
9	// Дописываем в А массив, оставшийся не пустым	?
10	**while** $i \le n_1$ **do** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$, $k \leftarrow k + 1$	?
11	**while** $j \le n_2$ **do** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$, $k \leftarrow k + 1$	?

Вычислительная сложность процедуры слияния

	MERGE(A, l, k, r)	$O(\ f(n)\ )$
1	$n_1 \leftarrow k - l + 1$, $n_2 \leftarrow r - k$	$O(1)$
2	**for** $i \leftarrow 1$ **to** n_1 **do** $L[i] \leftarrow A[l + i - 1]$ // $L = A[l \ldots k]$	$O(n_1)=O(n/2)$
3	**for** $j \leftarrow 1$ **to** n_2 **do** $R[j] \leftarrow A[k + j]$ // $R = A[(k + 1)\ldots r]$	$O(n_2)=O(n/2)$
4	$i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$	$O(1)$
5	**while** $i \le n_1$ и $j \le n_2$ **do** // Пока L и R не пусты	
6	**if** $L[i] \le R[j]$ **then** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$	$O(n_1+h_2)$ или $O(n_2+h_1)$
7	**else** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$	
8	$k \leftarrow k + 1$	
9	// Дописываем в А массив, оставшийся не пустым	
10	**while** $i \le n_1$ **do** $A[k] \leftarrow L[i]$, $i \leftarrow i + 1$, $k \leftarrow k + 1$	$O(n_2-h_2)$ или $O(n_2-h_1)$
11	**while** $j \le n_2$ **do** $A[k] \leftarrow R[j]$, $j \leftarrow j + 1$, $k \leftarrow k + 1$	

Вычислительная сложность процедуры слияния

$$T_M(n) = O(1) + O(n/2) + O(n/2) + O(1) + \begin{cases} O(n_1 + h_2) + O(n_2 - h_2) \\ O(n_2 + h_1) + O(n_1 - h_1) \end{cases}$$

$$T_M(n) = 2O(1) + O(n/2 + n/2) + \begin{cases} O(n_1 + h_2 + n_2 - h_2) \\ O(n_2 + h_1 + n_1 - h_1) \end{cases}$$

$$T_M(n) = 2O(1) + O(n) + \begin{cases} O(n_1 + n_2) \\ O(n_2 + n_1) \end{cases}$$

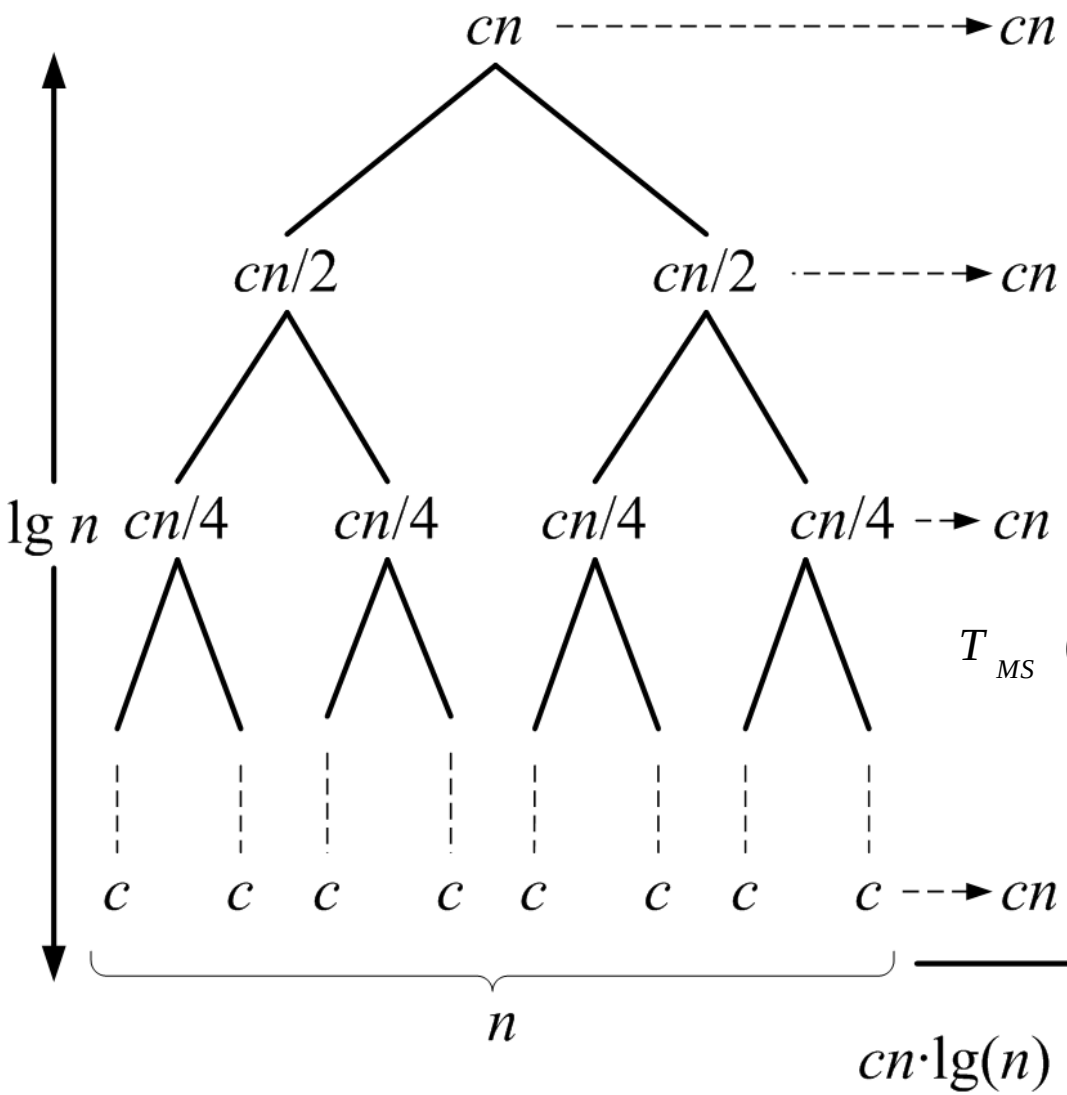
$$T_M(n) = 2O(1) + O(n) + O(n) = 2O(1) + 2O(n) = O(n)$$

Анализ вычислительной сложности алгоритма сортировки слиянием

$$T_{DEC}(n) = \begin{cases} c, & n = 1 \\ aT_{DEC}(n/a) + T_D(n) + T_C(n), & n > 1 \end{cases}$$

$$T_{MS}(n) = \begin{cases} \Theta(1) \\ 2 \cdot T_{MS}(n/2) + \Theta(1) + \Theta(n) \end{cases}$$

Распределение операций по шагам



На каждом шаге алгоритма производится:

1. Разбиение: $\Theta(1)$
2. Покорение: $2T_{MS}(n/2)$
3. Комбинирование: $\Theta(n)$

$$T_{MS}(n) = \begin{cases} \Theta(1) \\ 2 \cdot T_{MS}(n/2) + \Theta(n) \end{cases}$$

$$T_{MS}(n) = cn \cdot (\lg n + 1) = \Theta(n \lg n)$$

Литература

1. (CLRS) Кормен Т., Лейзерсон Ч., Ривест Р. Штайн К. Алгоритмы: построение и анализ, 2-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2012. – 1296 с.: ил. – Парал. тит. англ. ISBN 978–5–8459–0857–5.

СПАСИБО ЗА ВНИМАНИЕ!