Разделяй и властвуй

Divide and Conquer

Разделяй и властвуй

  - Divide and conquer (Algorithmic paradigm)
- T(n) > 2 T(n/2)
- Разбить задачу на подзадачи меньшего размера
- Рекурсия / цикл
- Базовый случай
  - Элементарная / тривиальная задача
  - Не требует обработки
- Корректность работы алгоритма

Сортировка слиянием

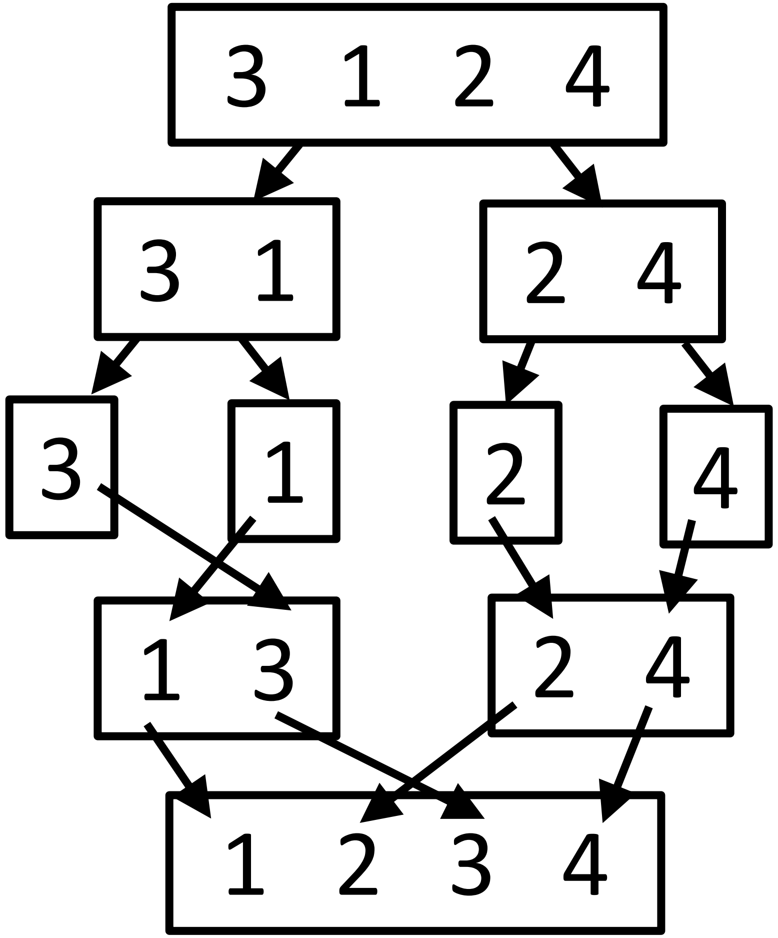
Merge Sort

Ссылки

- Кнут 2018 Искусство программирования том 3 Сортировка и поиск
  - 5.2.4 Сортировка методом слияния с. 174-185
- Кормен 2013 Алгоритмы Построение и анализ 3е
  - 2.3 Разработка алгоритмов. с.52-63
- Кормен 2014 Алгоритмы Вводный курс
  - Гл. 3. Сортировка слиянием. С.50-58
- Лафоре 2013 Структуры данных и алгоритмы в Java
  - Гл. 6 Рекурсия Сортировка слиянием с.267-289
- Хайнеман 2017 Алгоритмы C C++ Java Python
  - Гл. 4 Алгоритмы сортировки. Сортировка слиянием с.109-118
- Хайнеман 2023 Алгоритмы Python
  - Гл. 5 Сортировка без магии. Сортировка слиянием с.170-174
- Скиена 2022 Алгоритмы 3е
  - Гл. 4 Сортировка и поиск. 4.5 Сортировка слиянием. С.158-160

Пример сортировки слиянием

- Рекурсия



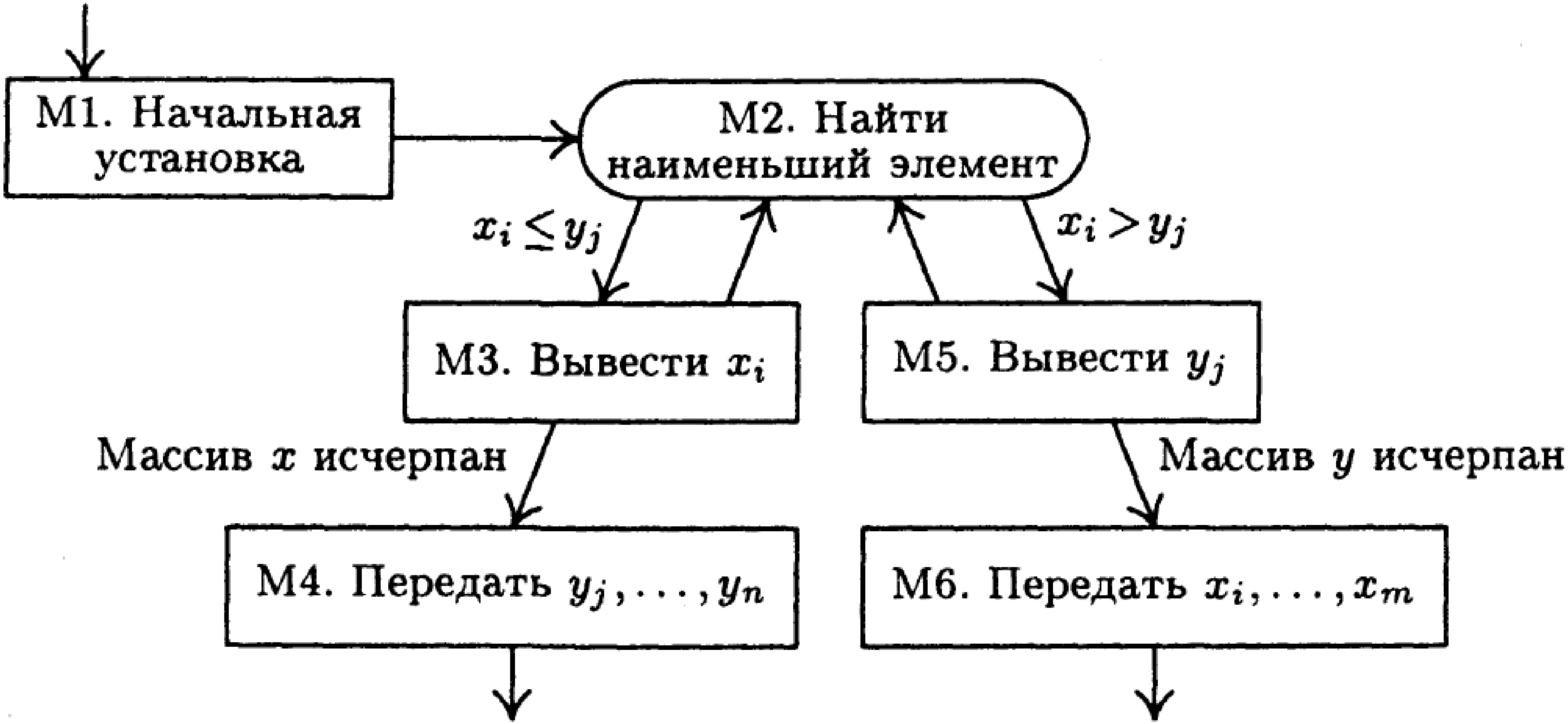


Рис. 29. Слияние $x_1 \leq \cdots \leq x_m$ с $y_1 \leq \cdots \leq y_n$.

```
sort (A)
    if A имеет меньше 2 элементов then
        return A
    else if A имеет два элемента then
        Обменять элементы A, если они не в порядке
        return A

    sub1 = sort(left half of A)
    sub2 = sort(right half of A)

    Объединить sub1 и sub2 в новый массив B
    return B
end
```

MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
Пусть L[1..n1 + 1] и R[1..n2 + 1] — новые массивы
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Procedure MERGE-SORT(A, p, r)

Inputs:

- A: an array.
- p, r: starting and ending indices of a subarray of A.

Result: The elements of the subarray $A[p \mathinner{..} r]$ are sorted into nondecreasing order.

1. If $p \geq r$, then the subarray $A[p \mathinner{..} r]$ has at most one element, and so it is already sorted. Just return without doing anything.
2. Otherwise, do the following:

    A. Set q to $\lfloor (p + r)/2 \rfloor$.
    B. Recursively call MERGE-SORT(A, p, q).
    C. Recursively call MERGE-SORT$(A, q + 1, r)$.
    D. Call MERGE(A, p, q, r).

Procedure MERGE(A, p, q, r)

Inputs:

- A: an array.
- p, q, r: indices into A. Each of the subarrays $A[p\,..\,q]$ and $A[q+1\,..\,r]$ is assumed to be already sorted.

Result: The subarray $A[p\,..\,r]$ contains the elements originally in $A[p\,..\,q]$ and $A[q+1\,..\,r]$, but now the entire subarray $A[p\,..\,r]$ is sorted.

1. Set n_1 to $q - p + 1$, and set n_2 to $r - q$.
2. Let $B[1\,..\,n_1+1]$ and $C[1\,..\,n_2+1]$ be new arrays.
3. Copy $A[p\,..\,q]$ into $B[1\,..\,n_1]$, and copy $A[q+1\,..\,r]$ into $C[1\,..\,n_2]$.
4. Set both $B[n_1+1]$ and $C[n_2+1]$ to ∞.
5. Set both i and j to 1.
6. For $k = p$ to r:

   A. If $B[i] \le C[j]$, then set $A[k]$ to $B[i]$ and increment i.

   B. Otherwise ($B[i] > C[j]$), set $A[k]$ to $C[j]$ and increment j.

```python
def sort(A):

    def rsort(lo, hi):
        if hi > lo:
            mid = (lo+hi) // 2
            rsort(lo, mid)
            rsort(mid+1, hi)
            merge(lo, mid, hi)

    rsort(0, len(A)-1)
```

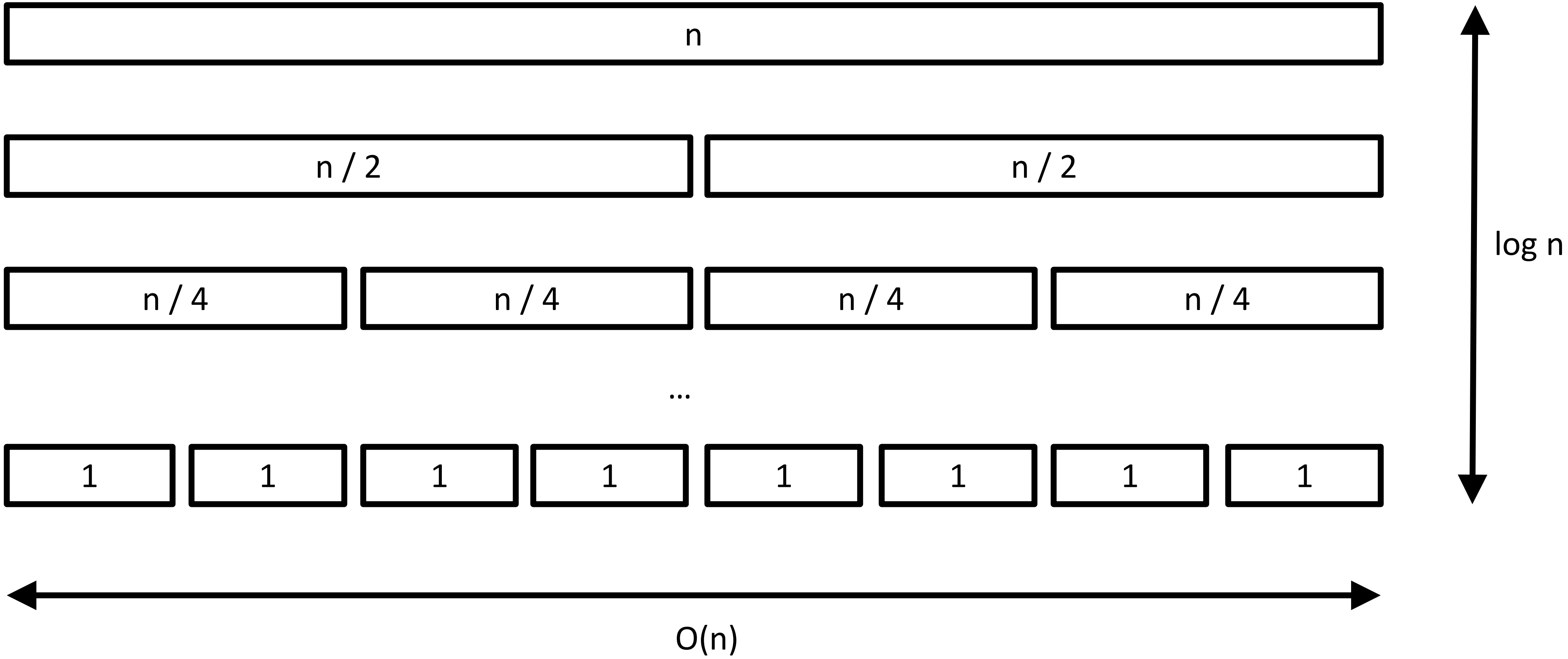
```python
def merge_sort(A):
    aux = [None] * len(A)

    def rsort(lo, hi):
        if hi > lo:
          mid = (lo+hi) // 2
          rsort(lo, mid)
          rsort(mid+1, hi)
          merge(lo, mid, hi)

    def merge(lo, mid, hi):
        aux[lo:hi+1] = A[lo:hi+1]
        left, right = lo, mid+1
        for i in range(lo, hi+1):
            if left > mid or right <= hi and aux[right] < aux[left]:
                A[i] = aux[right]
                right += 1
            else:
                A[i] = aux[left]
                left += 1

    rsort(0, len(A)-1)
```

Split-Sort-Merge



Функция sort()

```
def sort(x):
  n = len(x)
  if n < 2:
    return x
  else:
    mid = n//2
    left = sort(x[:mid])
    right = sort(x[mid:])
    return merge(left, right)
```

Функция merge()

```
def merge(left, right):
  i = j = 0
  s = []
  len_left = len(left)
  len_right = len(right)
  while i < len_left and j < len_right:
    if left[i] <= right[j]:
      s.append(left[i])
      i += 1
    else:
      s.append(right[j])
      j += 1
  s += left[i:] + right[j:]
  return s
```

Временна́я сложность

- Рекурсия

$T(n) = 2\ T(n/2) + C\ n$

$T(n/2) = 2\ T(n/4) + C\ n/2$

$T(n) = 2\ (2\ T(n/4) + C\ n/2) + C\ n = 4\ T(n/4) + 2\ C\ n$

$T(n/4) = 2\ T(n/8) + C\ n/4$

$T(n) = 4\ (2\ T(n/8) + C\ n/4) + 2\ C\ n = 8\ T(n/8) + 3\ C\ n$

$T(n) = 2^K\ T(n/2^K) + K\ C\ n$

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

$T(n) = 2^K\, T(n/2^K) + K\, C\, n$

$n = 2^K$

$K = \log_2 n$

$n/2^K = 1$

$T(n) = n\, T(1) + C\, n \log n = O(n) + O(n \log n) = O(n \log n)$

Пространственная сложность

- M(n) - ?
- Учитываем "новые" массивы
- Не учитываем входной массив

- Сортировка «на месте»
  - in place

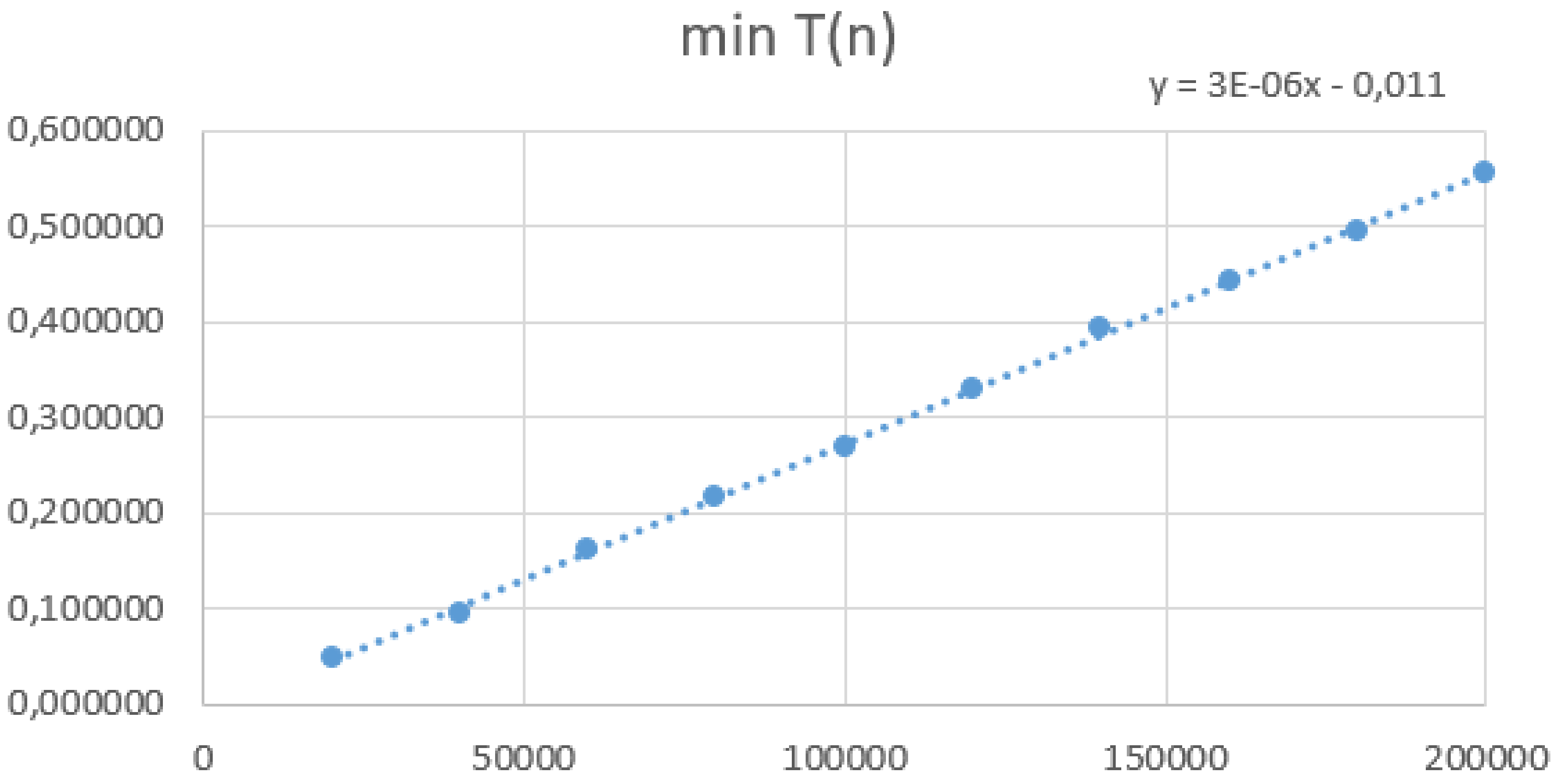
- Распараллеливание вычислений

Наихудший и наилучший случаи

- Количество сравнений и слияний
- T(n) - ?

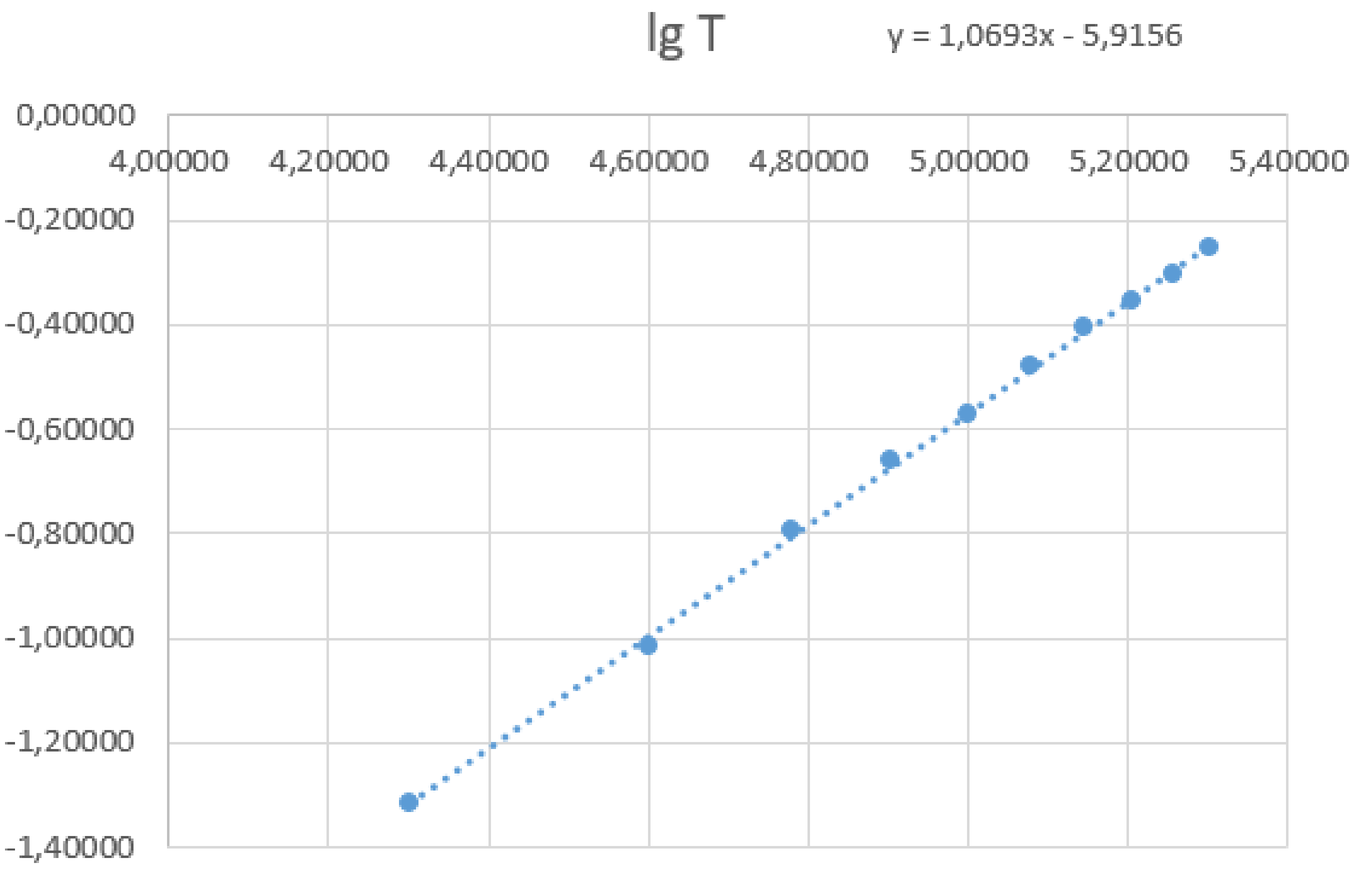
- Массив упорядочен по возрастанию
- Массив упорядочен по убыванию
- Массив заполнен случайным образом

min T(n)



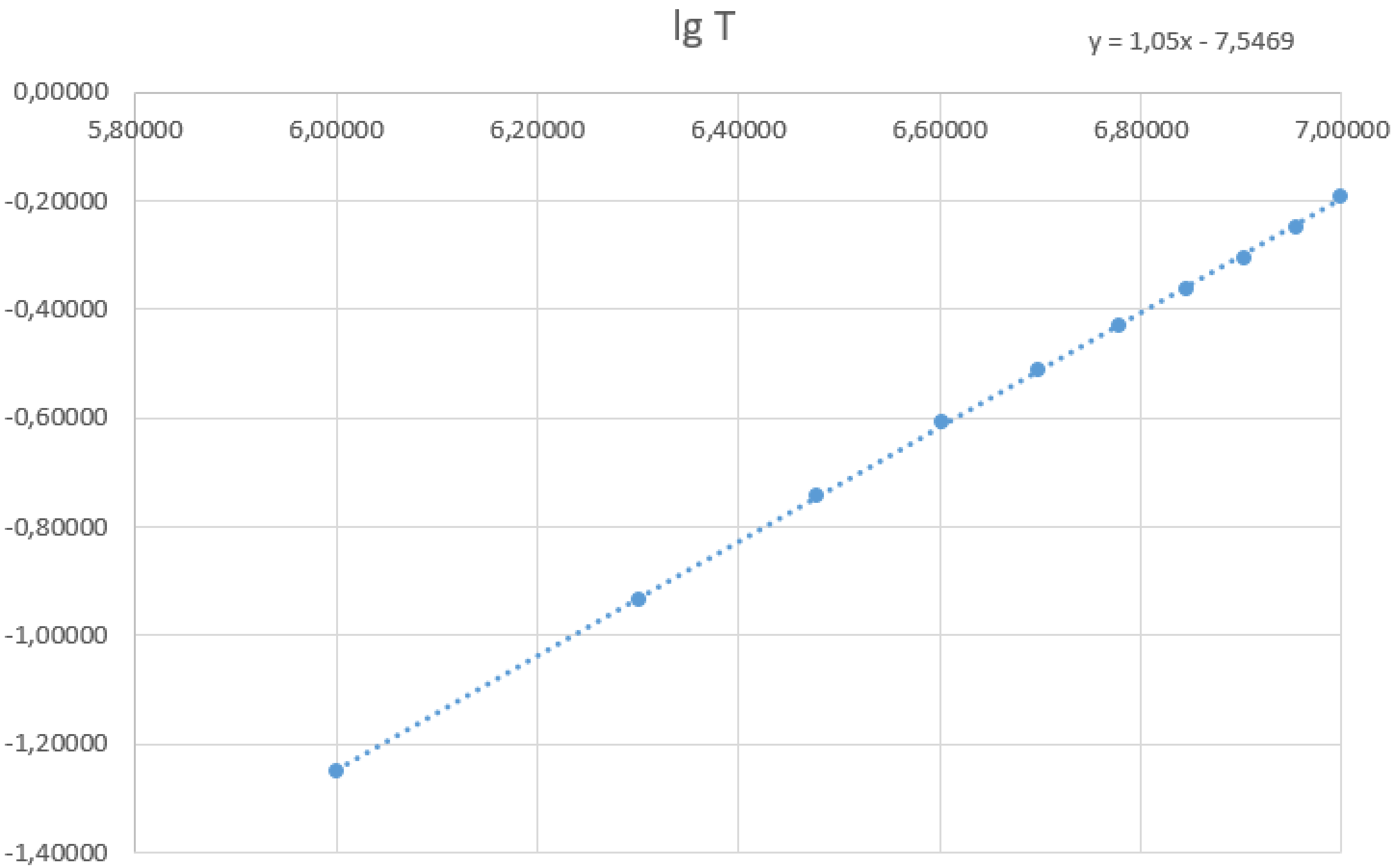
min T(n)
y = 3E-06x - 0,011
0,600000
0,500000
0,400000
0,300000
0,200000
0,100000
0,000000
0
50000
100000
150000
200000

lg T (lg n) - ру



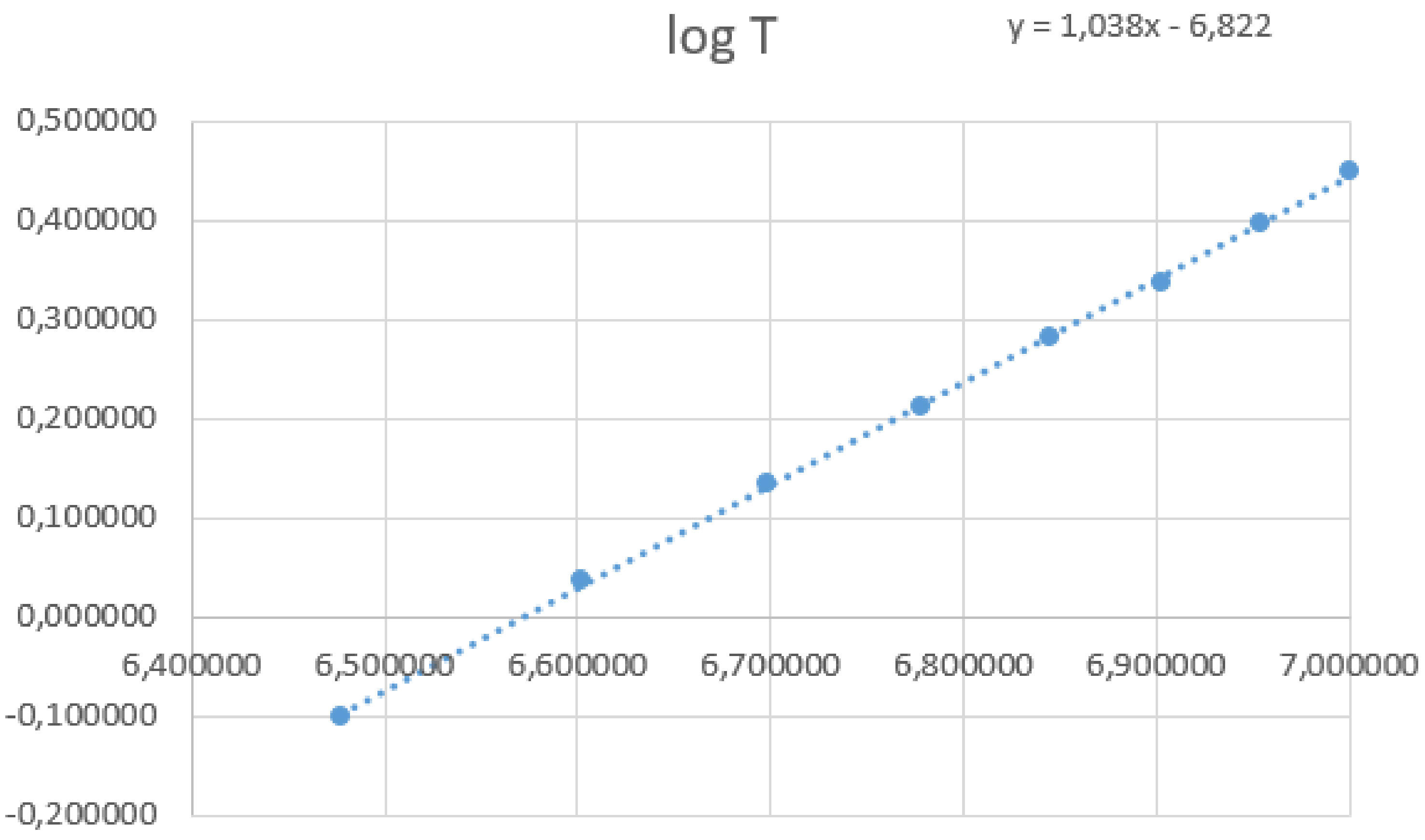
lg T
y = 1,0693x - 5,9156
0,00000
-0,20000
-0,40000
-0,60000
-0,80000
-1,00000
-1,20000
-1,40000
4,00000
4,20000
4,40000
4,60000
4,80000
5,00000
5,20000
5,40000

lg T (lg n) - java



lg T
y = 1,05x - 7,5469
0,00000
5,80000
6,00000
6,20000
6,40000
6,60000
6,80000
7,00000
-0,20000
-0,40000
-0,60000
-0,80000
-1,00000
-1,20000
-1,40000

lg T (lg n) - cs

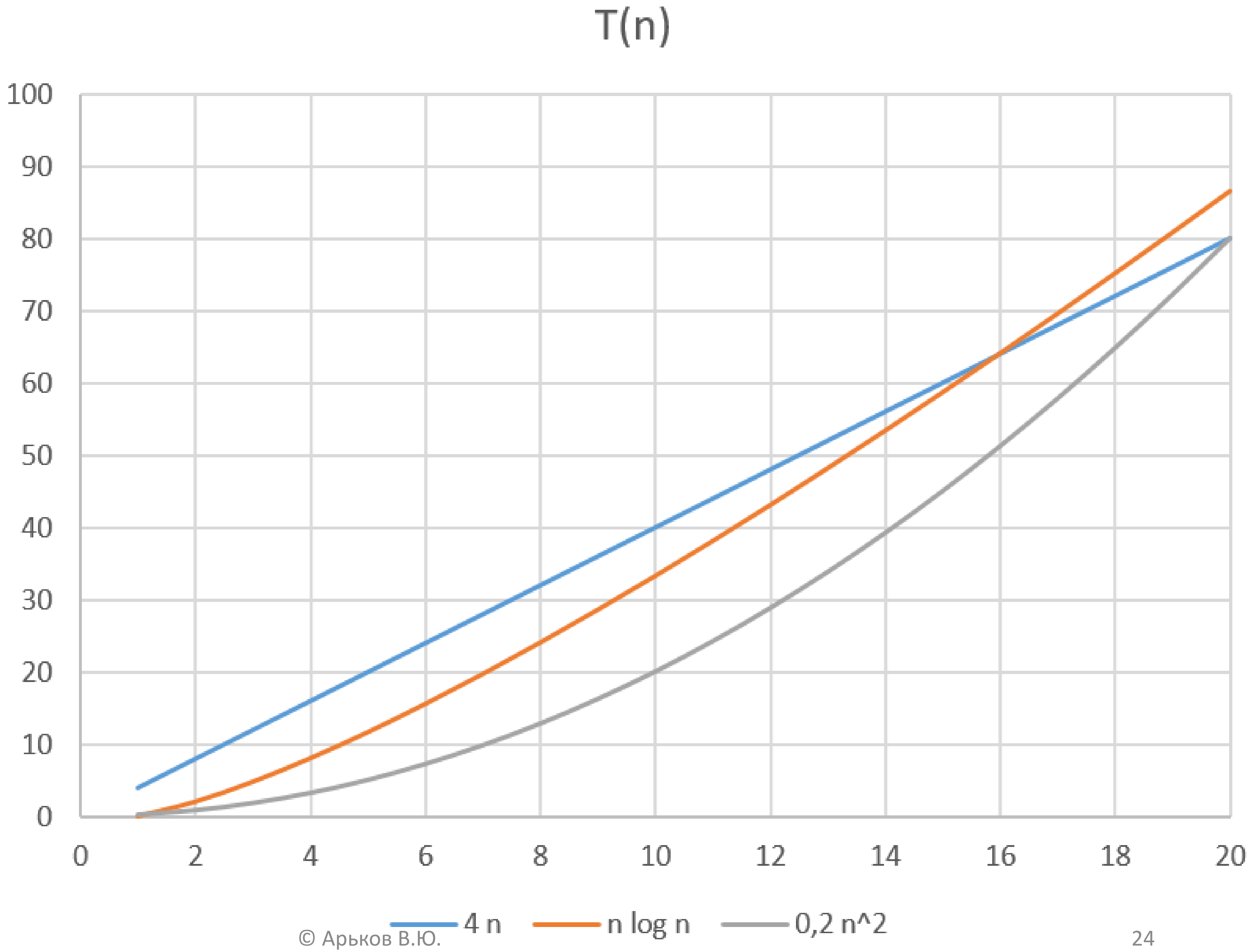

log T
y = 1,038x - 6,822
0,500000
0,400000
0,300000
0,200000
0,100000
0,000000
-0,100000
-0,200000
6,400000
6,500000
6,600000
6,700000
6,800000
6,900000
7,000000

Теория:

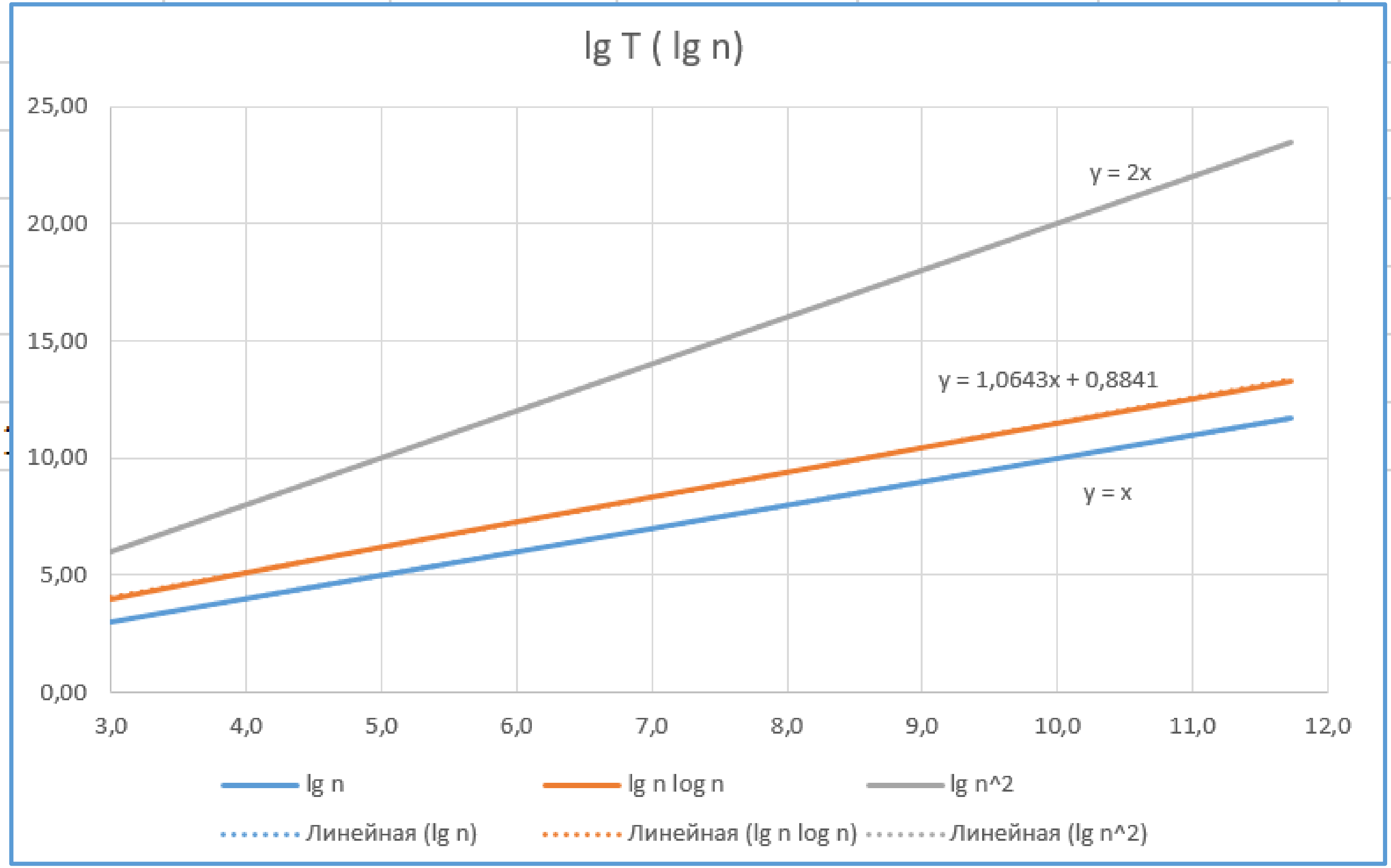
T(n) = n

T(n) = n^2

T(n) = n log n



n	n	n log n	n^2	lg n	lg n	lg n log n	lg n^2
1000	1000	9965,784	1,0E+06	3,0	3,00	4,00	6,00
2000							6,60
4000							7,20
8000							7,81
16000							8,41
32000							9,01
64000							9,61
128000							10,21



Улучшения

- Гибридный алгоритм
- Адаптивный алгоритм

Обсуждение

- Алгоритмы и структуры данных
  - Массивы
  - Списки
- Реализация алгоритма
  - Сверху вниз
  - Снизу вверх
  - Параллельные вычисления
- Внешняя память
  - Произвольный доступ
  - Последовательный доступ

Сравнение алгоритмов сортировки

- Time and Space Complexities of Sorting Algorithms
- https://www.interviewkickstart.com/learn/time-complexities-of-all-sorting-algorithms