

# Анализ алгоритмов

---

«О большое»

# Задача и решение

- Задача
  - Поиск максимального значения в массиве
- Решение
  - Перебор всех элементов в цикле
- Алгоритм
  - Последовательность шагов
- Вычислительная сложность
  - Линейная
  - $O(n)$

# Псевдокод [Нетология-01-с.24]

## Алгоритм поиска максимума в массиве

Решение:

1	<code>arr = [1, 22, 3, 45, 7]</code>	} Заводим или получаем на вход массив
2	<code>n = длина(arr)</code>	
3		
4	<code>max = arr[0]</code>	} Первым кандидатом на максимум возьмём первый элемент массива
5		
6	<code>for i от 0 до n-1</code>	} Пробежимся циклом по массиву
7		
8	<code>if arr[i] &gt; max</code>	} Если очередной элемент больше текущего максимума, то обновим
9	<code>max = arr[i]</code>	
10		
11	<code>print "Максимум: " max</code>	



# Псевдокод [Cormen 2009, p.214]

MINIMUM(*A*)

```
1  min = A[1]
2  for i = 2 to A.length
3      if min > A[i]
4          min = A[i]
5  return min
```

# Генерируем псевдокод

- составь простой алгоритм поиска максимума в массиве, представь алгоритм в виде псевдокода с комментариями
- напиши псевдокод для предыдущего решения русскими словами с короткими комментариями
- объясни простыми словами какая сложность этого алгоритма и почему

# Псевдокод [perplexity.ai]

Функция НайтиМаксимум(массив):

// Проверяем, что массив не пустой

Если длина(массив) == 0:

    Вернуть "Массив пустой"

// Инициализируем переменную максимум первым элементом  
максимум = массив[0]

// Проходим по всем элементам массива

Для каждого элемент в массиве:

    // Если текущий элемент больше текущего максимума

    Если элемент > максимум:

        // Обновляем максимум

        максимум = элемент

// Возвращаем найденный максимум

Вернуть максимум

# Псевдокод [ГигаЧат]

# Функция для поиска максимального значения в массиве

Функция НайтиМаксимум(массив)

# Проверяем, есть ли элементы в массиве

Если Длина(массив) == 0

Вернуть "Массив пуст"

# Предполагаем, что первый элемент является максимальным

МаксимальноеЗначение = массив[0]

# Перебираем все остальные элементы массива

Для каждого элемента в массив[1:]

# Сравниваем текущий элемент с предполагаемым максимумом

Если элемент > МаксимальноеЗначение

# Обновляем значение максимума

МаксимальноеЗначение = элемент

# Возвращаем найденное максимальное значение

Вернуть МаксимальноеЗначение

# Генерируем код

Составь программу на Python для псевдокода

```
arr = [1, 22, 3, 45, 7]
n = длина(arr)
max = arr[0]
for i от 0 до n-1
    if arr[i] > max
        max = arr[i]
print "Максимум: " max
```

Оформи код в виде функции



# Генерируем код [ГигаЧат]

```
def find_max(arr):  
    # Проверяем, что массив не пустой  
    if not arr:  
        print("Массив пуст")  
        return None  
  
    # Предполагаем, что первый элемент является максимальным  
    max_val = arr[0]  
    # Проходим по всем элементам массива, начиная со второго  
    for i in range(1, len(arr)):  
        # Если текущий элемент больше максимального, обновляем максимум  
        if arr[i] > max_val:  
            max_val = arr[i]  
  
    # Возвращаем найденное максимальное значение  
    return max_val
```

# Генерируем код [Perplexity]

```
def find_maximum(array):  
    # Check if the array is empty  
    if len(array) == 0:  
        return "The array is empty"  
    # Initialize the maximum variable with the first element of the  
    array  
    maximum = array[0]  
    # Iterate through all elements in the array  
    for i in range(len(array)):  
        # If the current element is greater than the current maximum  
        if array[i] > maximum:  
            # Update the maximum  
            maximum = array[i]  
    # Return the found maximum  
    return maximum
```

# Тестируем программу

✓  
0s

```
▶ # Пример использования функции  
arr = [1, 22, -3, 45, 0]  
result = find_max(arr)  
if result is not None:  
    print(f"Максимум: {result}")
```

⇒ Максимум: 45

✓  
0s

```
[4] print(find_max([]))
```

⇒ Массив пуст  
None

# Измеряем время

```
import time
import numpy as np
n = 1000000
x = np.random.randint(1, n*100, size=n)
start = time.time()
m = find_max(x)
end = time.time()
t = end - start
print(f"T({len(x)}) = {t:.6f}    max(x) = {m}")
```

T(1000000) = 0.280949 max(x) = 99999885

# Подбираем размер задачи n

```
n = 1_000_000
```

```
T(1000000) = 0.128893  max(x) = 99999894
```

```
n = 5_000_000
```

```
T(5000000) = 0.674824  max(x) = 499999994
```

```
n = 10_000_000
```

```
T(10000000) = 2.499915  max(x) = 999999973
```

```
n = 20_000_000
```

```
T(20000000) = 2.675346  max(x) = 1999999820
```

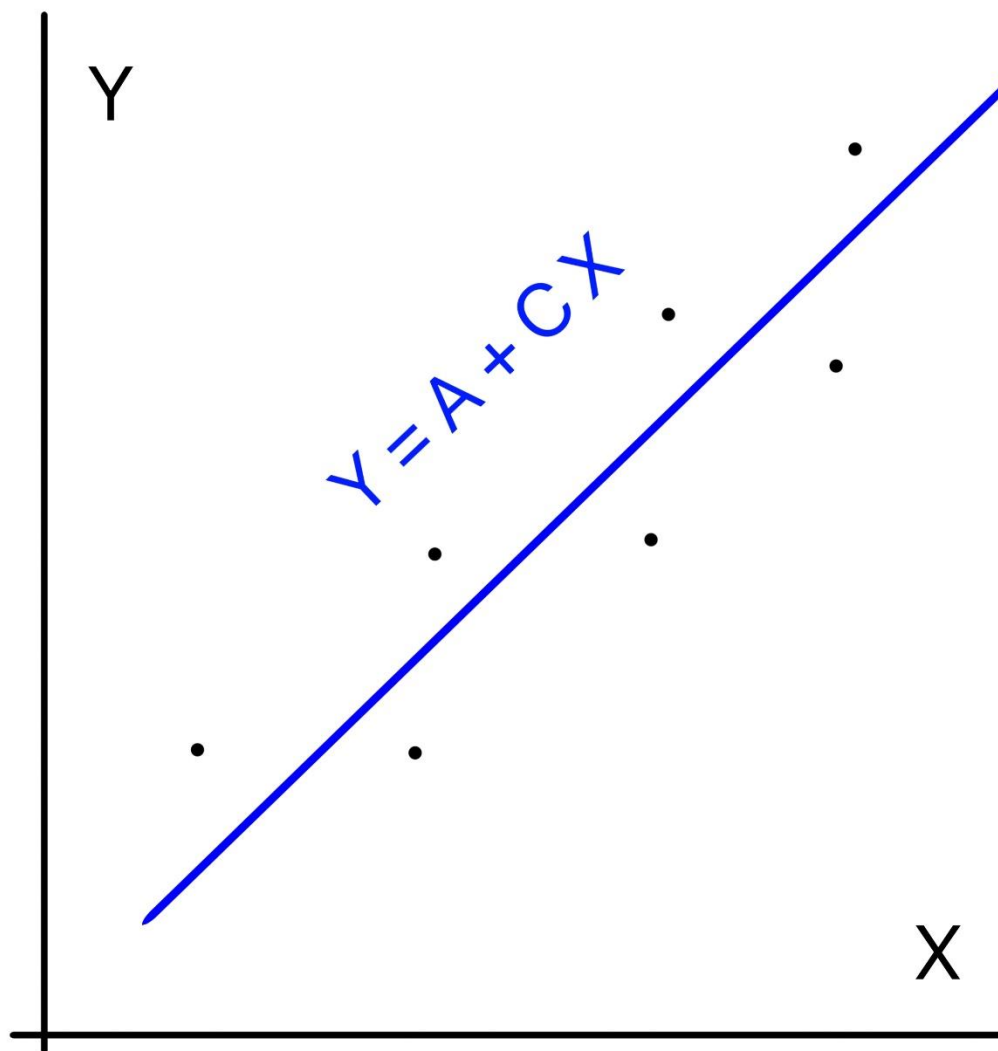
```
n = 50_000_000
```

```
T(50000000) = 7.826333  max(x) = 4999999998
```

# Организуем эксперимент

- Вызываем функцию в цикле
  - Не менее 10 значений  $n$
  - Не менее 10 прогонов
- Выполняем программу
  - В облаке
    - Colab
  - На локальном компьютере
    - CMD
- Перенаправляем результаты в файл \*.csv

# Линейная зависимость



# Оцениваем «порядок»

- Ищем линейную зависимость  $Y(X)$ 
  - Используем логарифмирование
- Выбираем значения  $n$ 
  - Равномерные значения по горизонтальной оси
  - Разумные пределы значений по времени
- Получаем уравнение  $Y(X)$ 
  - Оцениваем «линейность» визуально
- Делаем выводы о сложности  $T(n) = O(?)$ 
  - «О большое»



# Проводим отладку

✓  
10s



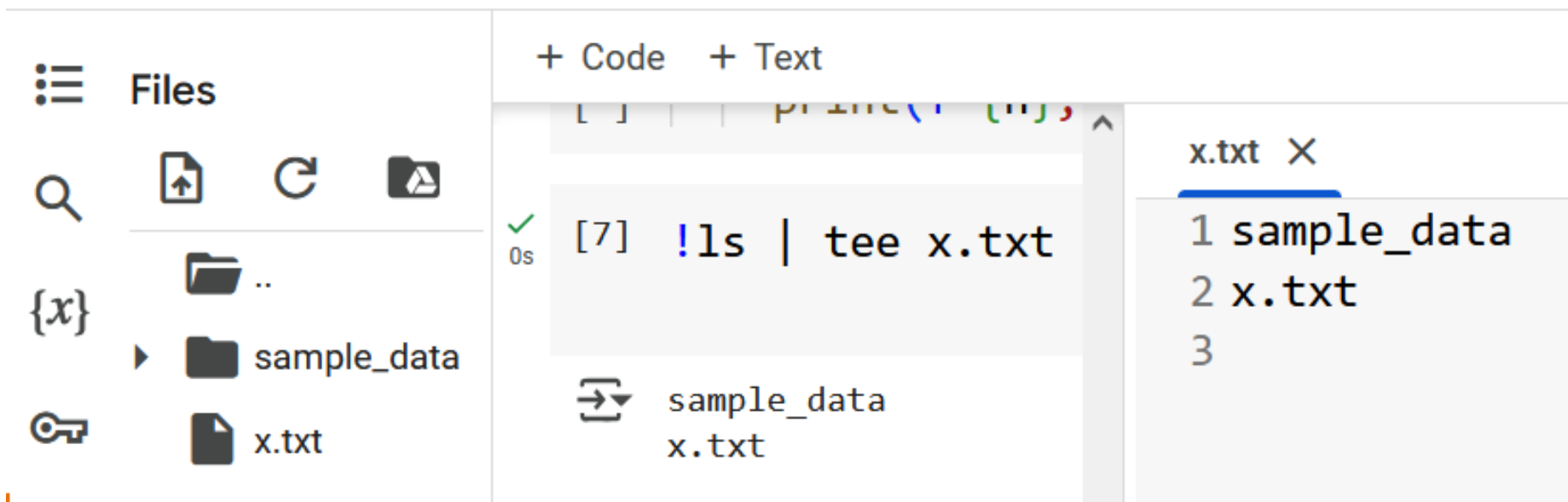
```
print("n;T")
for n in range(5_000_000, 50_000_000, 25_000_000):
    for _ in range(3):
        x = np.random.randint(1, n*100, size=n)
        start = time.time()
        m = find_max(x)
        end = time.time()
        t = end - start
        print(f"{n};{t:.6f}".replace(".", ","))
```



```
n;T
5000000;0,411240
5000000;0,416581
5000000;0,414855
30000000;2,462676
30000000;2,530023
30000000;3,047002
```

# Команда tee

- Английская буква «Т»
  - Т-образное соединение (труб, кабелей и т.п.)
- Одновременный вывод на экран и в файл
  - Конвейер в командной строке



# Вывод без буферизации

- `print(f"{n};{t}", flush=True)`
  - Выводим результаты немедленно
- Lavatory Flush Button



# Cell Magic

- Записываем содержимое кодовой ячейки в файл
- `%%writefile f.py`
- Запускаем программу в следующей ячейке
- `!python f.py`



%%writefile f.py

```
def find_max(arr):
    max_val = arr[0]
    for i in range(1, len(arr)):
        if arr[i] > max_val:
            max_val = arr[i]
    return max_val

import time
import numpy as np

print("\n;T")
for n in range(5_000_000, 50_000_001, 5_000_000):
    for _ in range(10):
        x = np.random.randint(1, n*100, size=n)
        start = time.time()
        m = find_max(x)
        end = time.time()
        t = end - start
        print(f"{n};{t:.6f}".replace(".", ","), flush=True)
```



Overwriting f.py

# Следим за выполнением

- Запускаем программу
- Наблюдаем за результатами опытов



```
!python f.py | tee f.csv
```

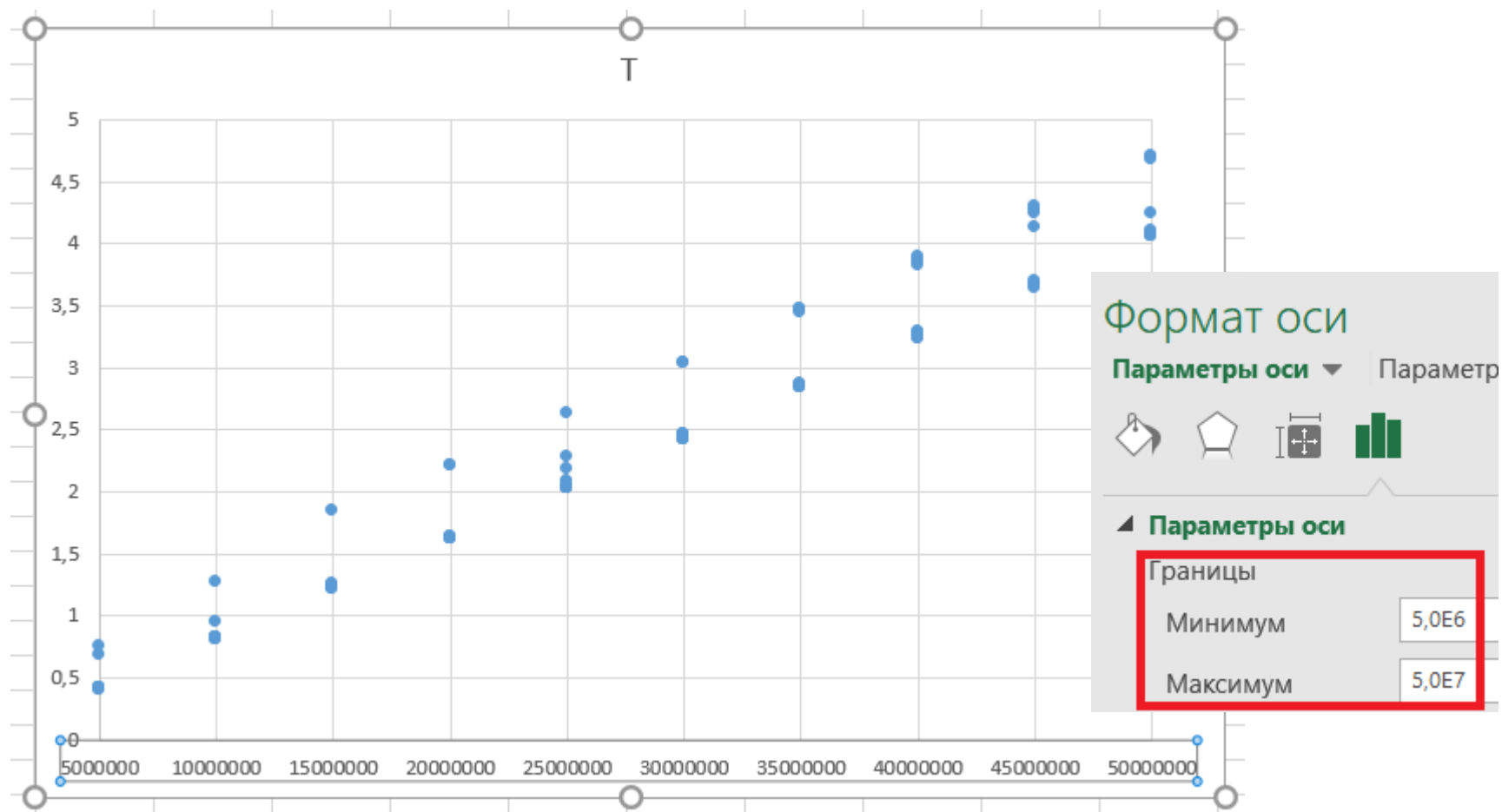
```
... n;T  
5000000;0,421167  
5000000;0,422872  
5000000;0,423601  
5000000;0,415334
```

# Загружаем данные

- Строки «прижаты влево»
- Числа «прижаты вправо»

B2				0,682869			
	A	B	C				
1	n	T					
2	5000000	0,682869					
3	5000000	0,756302					
4	5000000	0,409466					
5	5000000	0,41261					
6	5000000	0,410397					
7	5000000	0,414943					

# Диаграмма разброса





# Сводная таблица min T(n)

Названия строк ▾	Минимум по полю T	n	min T
5000000	0,40541	5000000	0,40541
10000000	0,807517	10000000	0,807517
15000000	1,217452	15000000	1,217452
20000000	1,621333	20000000	1,621333
25000000	2,023448	25000000	2,023448
30000000	2,419896	30000000	2,419896
35000000	2,836357	35000000	2,836357
40000000	3,229203	40000000	3,229203
45000000	3,641051	45000000	3,641051
50000000	4,054813	50000000	4,054813
<b>Общий итог</b>	<b>0,40541</b>		

## Поля сводной таблицы

Выберите поля для добавления в отчет:

☒ n☒ T

Перетащите поля в нужную область:

🔍 ФИЛЬТРЫ

📊 СТОЛБЦЫ

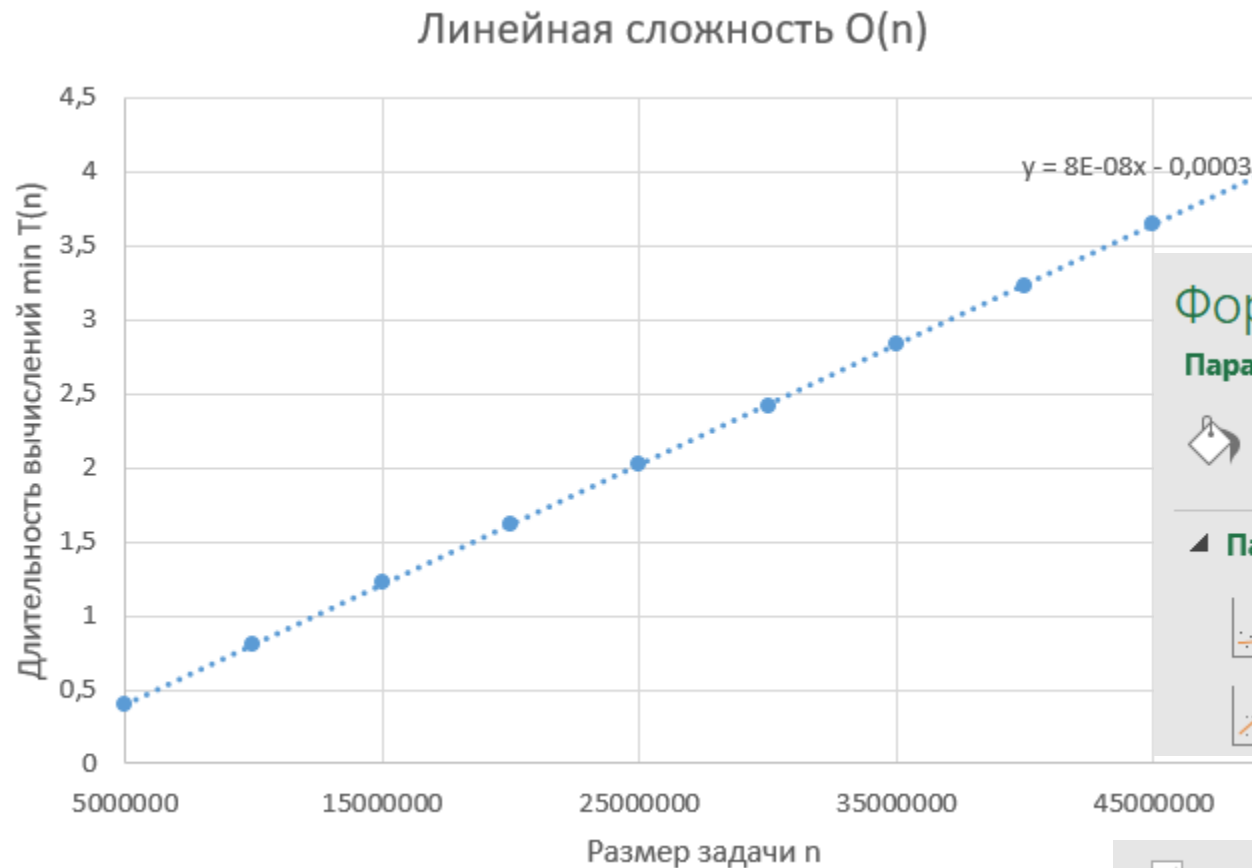
☰ СТРОКИ

Σ ЗНАЧЕНИЯ

n ▾

Минимум по полю T ▾

# Демонстрируем $O(n)$



## ЭЛЕМЕНТЫ ДИАГРАММЫ

- ☒ Оси
- ☒ Названия осей
- ☒ Название диаграммы
- ☐ Подписи данных
- ☐ Предел погрешностей
- ☒ Сетка
- ☐ Легенда
- ☒ Линия тренда

## Формат линии тренда

### Параметры линии тренда ▾



### ▲ Параметры линии тренда



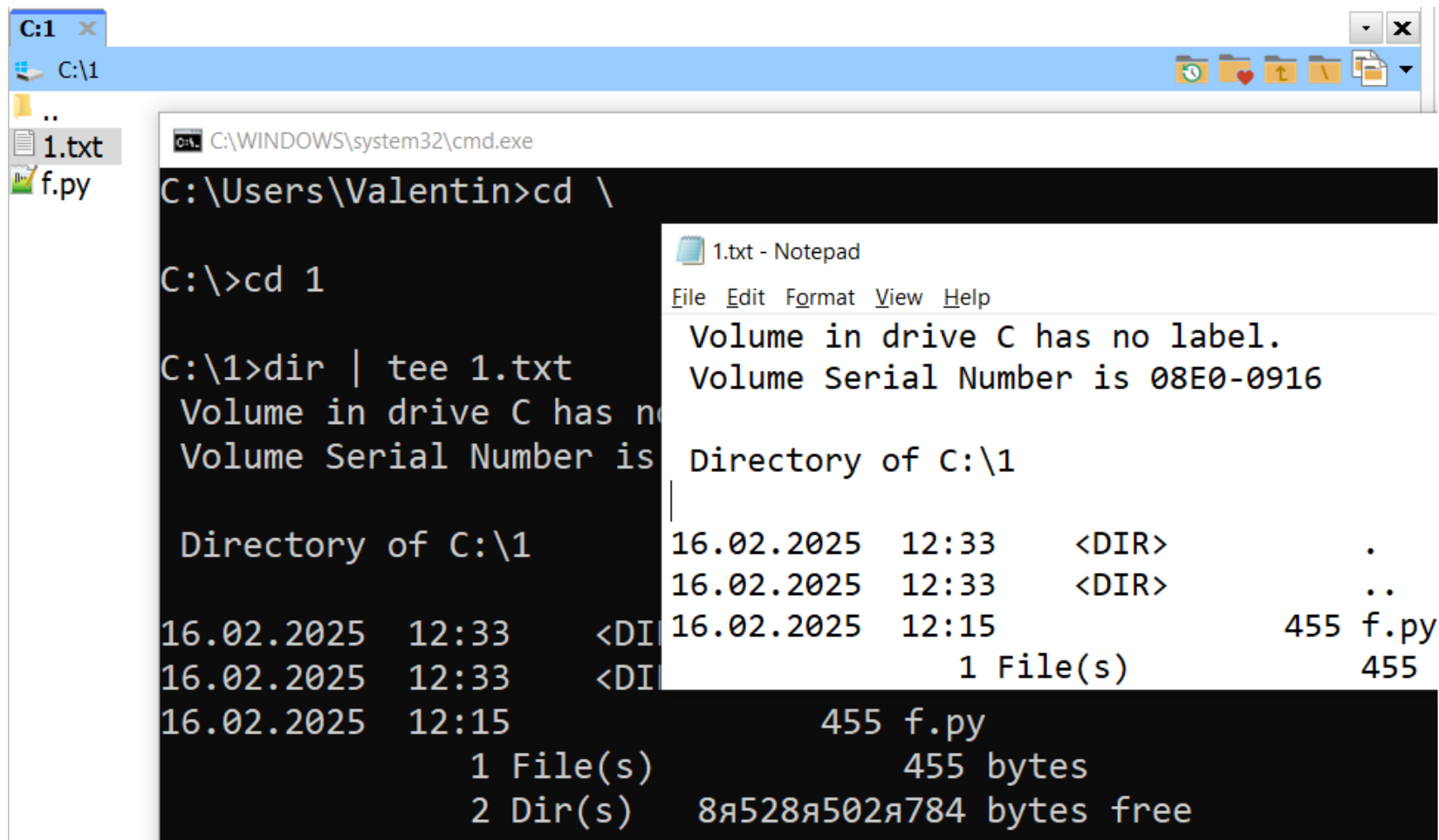
☐ Экспоненциальная



☒ Линейная

☒ показывать уравнение на диаграмме

# Команда TEE для CMD



```
C:\Users\Valentin>cd \

C:\>cd 1

C:\1>dir | tee 1.txt
Volume in drive C has no label.
Volume Serial Number is 08E0-0916

Directory of C:\1

16.02.2025  12:33    <DIR>          .
16.02.2025  12:33    <DIR>          ..
16.02.2025  12:15                455 f.py
                                1 File(s)        455
                                455 f.py
                                1 File(s)        455 bytes
                                2 Dir(s)      8я528я502я784 bytes free
```

# Локальный запуск

```
C:\1>pip install numpy
```

```
C:\1>python f.py | tee x.csv
```

```
n;T
```

```
5000000;0,729054
```

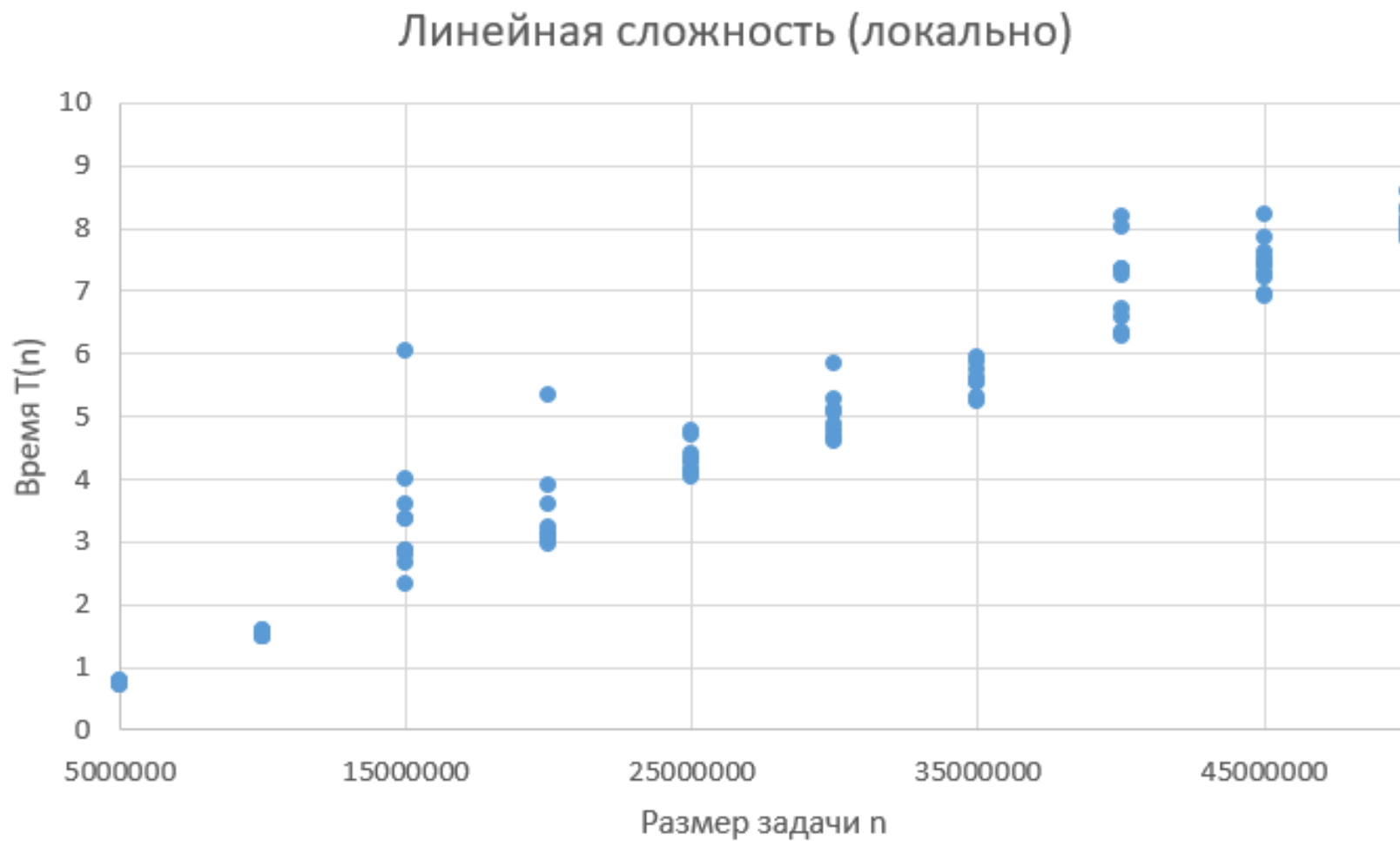
```
5000000;0,695180
```

```
5000000;0,693057
```

```
5000000;0,683624
```

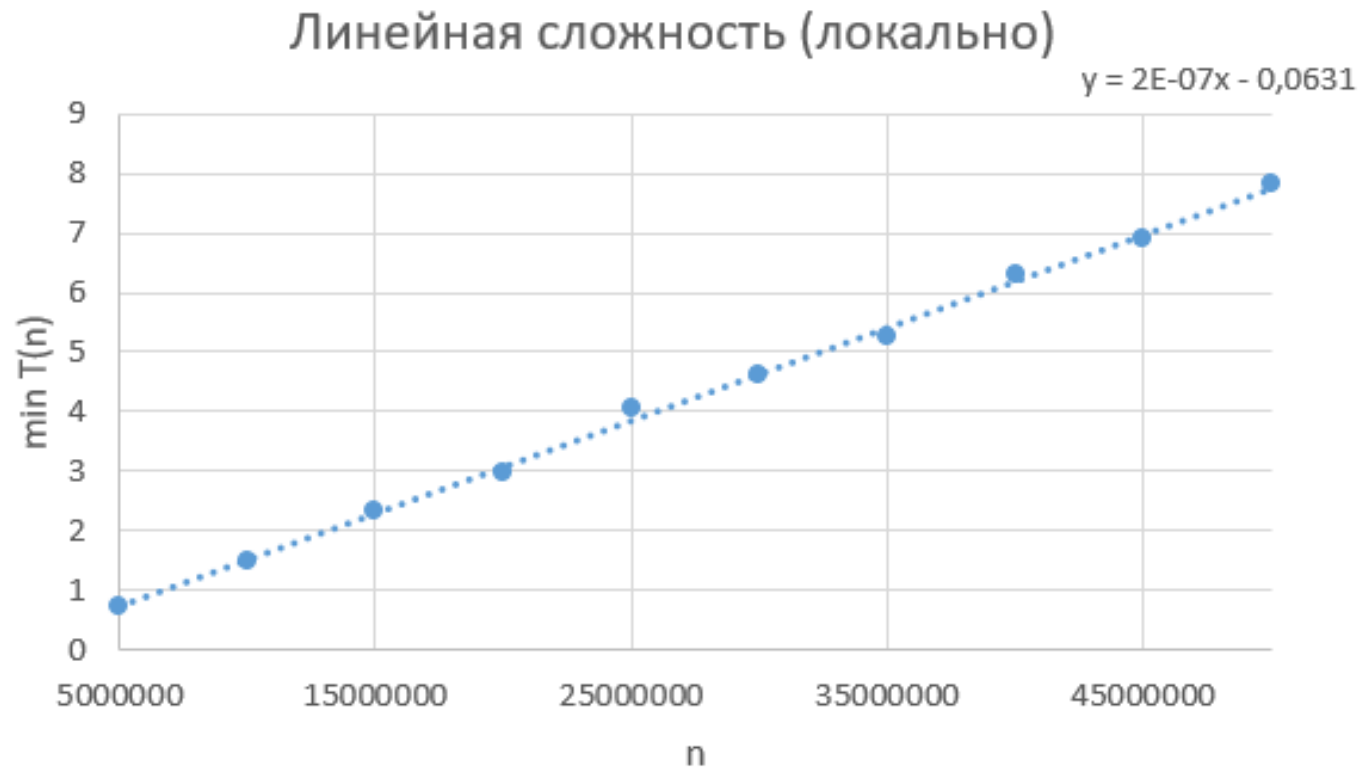
```
5000000;0,821838
```

# Исходные данные



# Сводная таблица и график

n	min T(n)
5000000	0,719248
10000000	1,489998
15000000	2,310825
20000000	2,968997
25000000	4,044994
30000000	4,609996
35000000	5,247003
40000000	6,286897
45000000	6,922062
50000000	7,810177



# О большое

- Ограничение сверху
  - Наихудший случай
  - Максимальная трудоемкость
  - Наибольшее число операций
- Полный перебор элементов массива
  - Обновление максимума на каждом элементе
- Значения по возрастанию

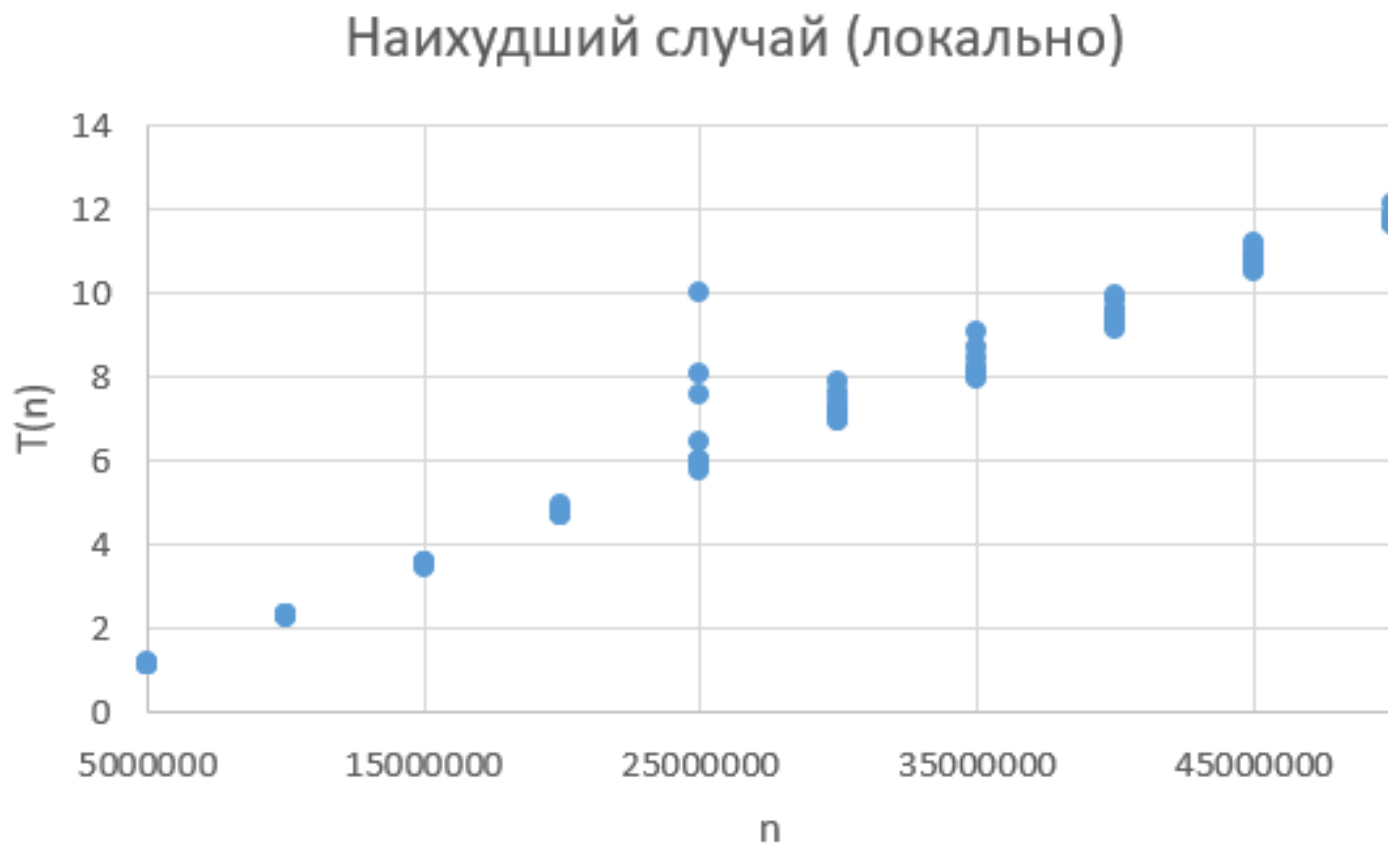
# Мин – макс - среднее

- Минимальное, максимальное, среднее время
- О-большое ( $O$ ) - наихудший случай
  - Верхняя граница роста времени выполнения
- Омега-большое ( $\Omega$ ) - наилучший случай
  - Нижняя граница роста времени выполнения
- Тэта-большое ( $\Theta$ ) - средний случай
  - Верхняя и нижняя границы сложности совпадают



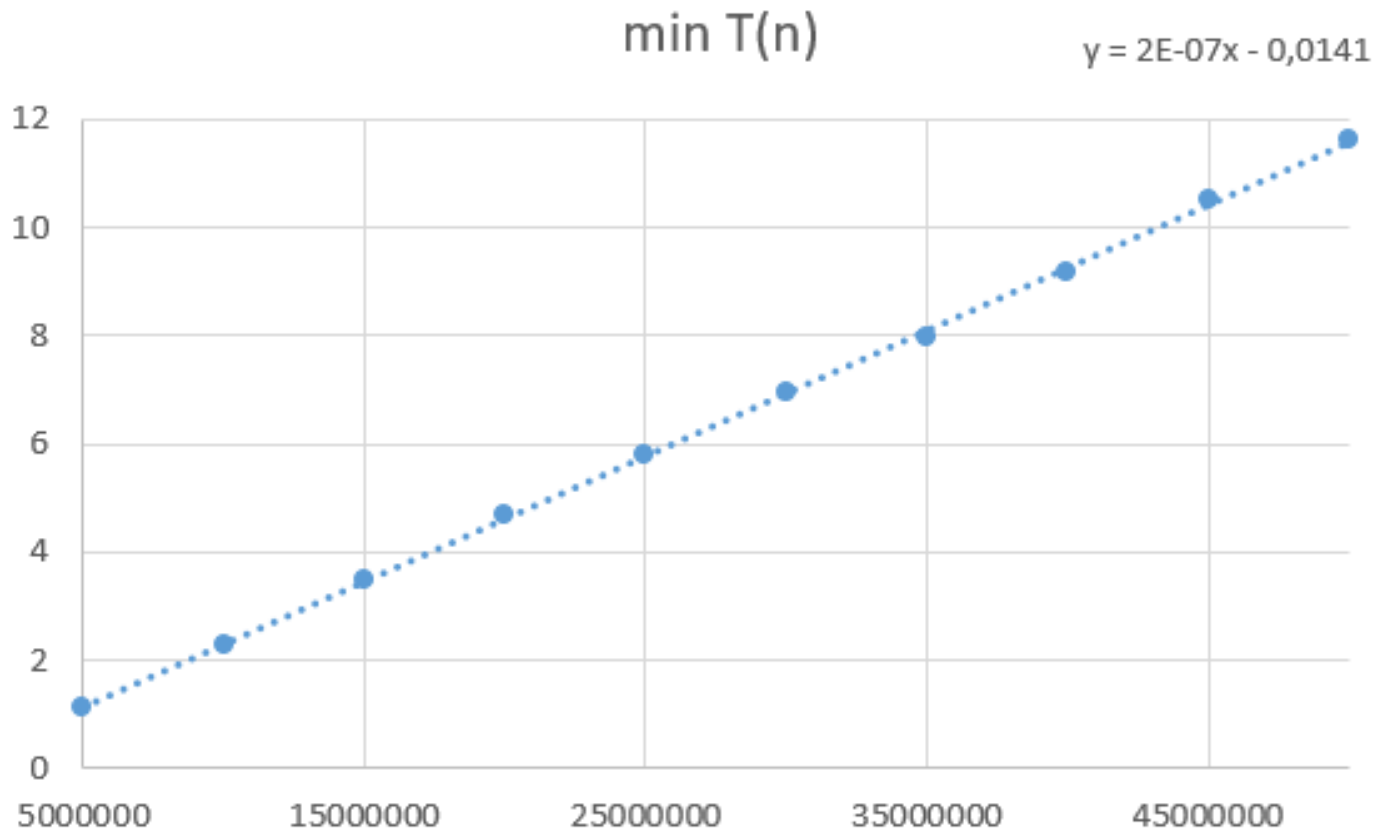
# Сырые данные

n	T
5000000	1,156646
5000000	1,166718
5000000	1,15206
5000000	1,169089
5000000	1,160728
5000000	1,167743
5000000	1,160377
5000000	1,138617
5000000	1,194712
5000000	1,162534
10000000	2,337568
10000000	2,358345
10000000	2,321273
10000000	2,300161



# Окончательный вывод: $O(n)$

n	min T(n)
5000000	1,138617
10000000	2,298232
15000000	3,489654
20000000	4,686069
25000000	5,787944
30000000	6,967161
35000000	7,98491
40000000	9,166447
45000000	10,50826
50000000	11,63717



# Логарифм – «линеаризация»

- Линейный «тренд» на графике
  - Легко увидеть на диаграмме разброса
  - Автоматическое уравнение и линия
- Линеаризация через логарифмирование
  - Нелинейная зависимость превращается в линейную
  - Равномерные значения аргумента в масштабе
- Примеры линеаризации
  - $O(\log n)$
  - $O(n^2)$
  - $O(2^n)$