Сортировка

Общие вопросы программирования

- *Д.Кнут Искусство программирования т.3 Сортировка и поиск*
- Как улучшать алгоритмы и программы?
- Как исследовать эффективность алгоритмов?
- Как выбрать алгоритм для конкретной задачи?
- В каком смысле алгоритм «наилучший»?
- Как теория вычислений согласуется с практикой?
- Как эффективно использовать различные виды памяти?

* Как алгоритм взаимодействует со структурой данных?

Сортировка

- Технологии построения и анализа алгоритмов
- Иллюстрация базовых концепций – каких?
- Подготовка данных для других алгоритмов – каких?
- Значительная доля машинного времени – сколько?

Определение

- Сортировка – Sorting
- Расположение объектов в заданном «порядке»
  - Возрастание
  - Невозрастание
  - Убывание
  - Неубывание
- Данные
  - Значения
  - Значение и ключ
  - Значение и ключи

Алгоритмы сортировки

- ИИ
  - Сколько существует алгоритмов сортировки?
  - How many sorting algorithms are available?

- Wiki
  - Алгоритм сортировки
  - Sorting algorithm

Метод пузырька

Bubble Sort

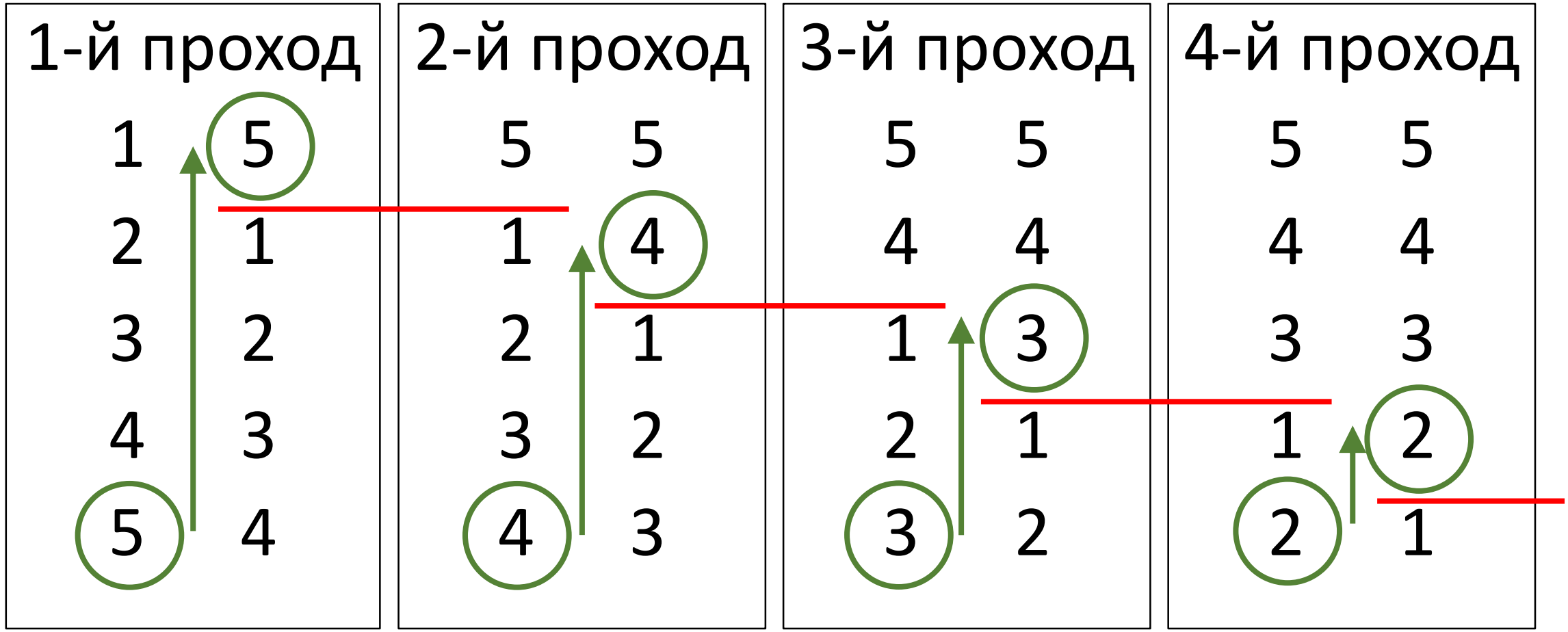
Сортировка методом пузырька

- Пузырьковая сортировка
  - Bubble Sort
- Сортировка по возрастанию
  - Каждый элемент массива/списка «всплывает» на свое место
  - «Обменная сортировка» / обмен значениями
  - Сортировка «на месте» (in place) без дополнительной памяти
- Попарное сравнение
  - Бинарные инструкции
  - Двухоперандовые команды
  - Инструкции с двумя операндами

Ссылки

- Википедия
  - https://ru.wikipedia.org/wiki/Сортировка_пузырьком
- Тимофей Хирьянов: Алгоритмы на Python 3. Лекция №6
  - https://youtu.be/NLq7nB9bV0M
- Bubble Sort - CS50 Shorts
  - https://cs50.harvard.edu/x/2025/shorts/bubble_sort/

Имитационное моделирование



Оценим сложность

- Худший случай
- Число обменов
- Сложность
- T(n) = O(?)

Вычислительная сложность

Сумма арифметической прогрессии

$1 + 2 + \ldots + (n - 1) =$

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

$= n \cdot (n - 1) / 2 =$

$= O(n^2)$

Квадратичная сложность

Практика

- Псевдокод на русском языке для пузырьковой сортировки
- Программная реализация псевдокода
- Лучший и худший случаи для пузырьковой сортировки
- Улучшение алгоритма: контроль числа обменов
  - Swap Counter

```
def bubble(x):
    n = len(x)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if x[j] > x[j+1]:
                x[j], x[j+1] = x[j+1], x[j]
    return x

x = [3, 2, 1, 4, 5]
print(x)
print(bubble(x))
```

```
[3, 2, 1, 4, 5]
[1, 2, 3, 4, 5]
```

```
using System;
class Program {
    static void BubbleSort(int[] x) {
        int n = x.Length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (x[j] > x[j + 1])
                    (x[j], x[j + 1]) = (x[j + 1], x[j]);
    }
    static void Main() {
        int[] x = { 3, 2, 1, 4, 5 };
        Console.WriteLine(string.Join(" ", x));
        BubbleSort(x);
        Console.WriteLine(string.Join(" ", x));
    }
}
```

```
3 2 1 4 5
1 2 3 4 5
```

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] x = {3, 2, 1, 4, 5};
        System.out.println(Arrays.toString(x));
        bubble(x);
        System.out.println(Arrays.toString(x));
    }
    private static void bubble(int[] x) {
        int n = x.length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (x[j] > x[j + 1]) {
                    int temp = x[j];
                    x[j] = x[j + 1];
                    x[j + 1] = temp;
                }
    }
}
```

```
[3, 2, 1, 4, 5]
[1, 2, 3, 4, 5]
```

«Трассировка»

```python
def bubble_sort(x):
    """Bubble sort of list x, returns sorted x."""

    n = len(x)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if x[j] > x[j + 1]:
                x[j], x[j + 1] = x[j + 1], x[j]
                print(x)
    return x

x = [5, 4, 3, 2, 1]
print(x, "<--")
print(bubble_sort(x), "-->")
```

```
C:\1>python bubble2.py
[5, 4, 3, 2, 1] <--
[4, 5, 3, 2, 1]
[4, 3, 5, 2, 1]
[4, 3, 2, 5, 1]
[4, 3, 2, 1, 5]
[3, 4, 2, 1, 5]
[3, 2, 4, 1, 5]
[3, 2, 1, 4, 5]
[2, 3, 1, 4, 5]
[2, 1, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5] -->
```

Подбираем размер n: T(n) = 0,1 … 1 сек

```
11  import time
12  n = 3000
13  x = list(range(n, 0, -1))
14  t0 = time.time()
15  y = bubble_sort(x)
16  t1 = time.time()
17  t = t1 - t0
18  print(f"n;T")
19  print(f"{n};{t:.10f}")
```

```
C:\1>python bubble_time.py
n;T
1000;0.1040003300

C:\1>python bubble_time.py
n;T
2000;0.4490010738

C:\1>python bubble_time.py
n;T
3000;1.2900097370
```

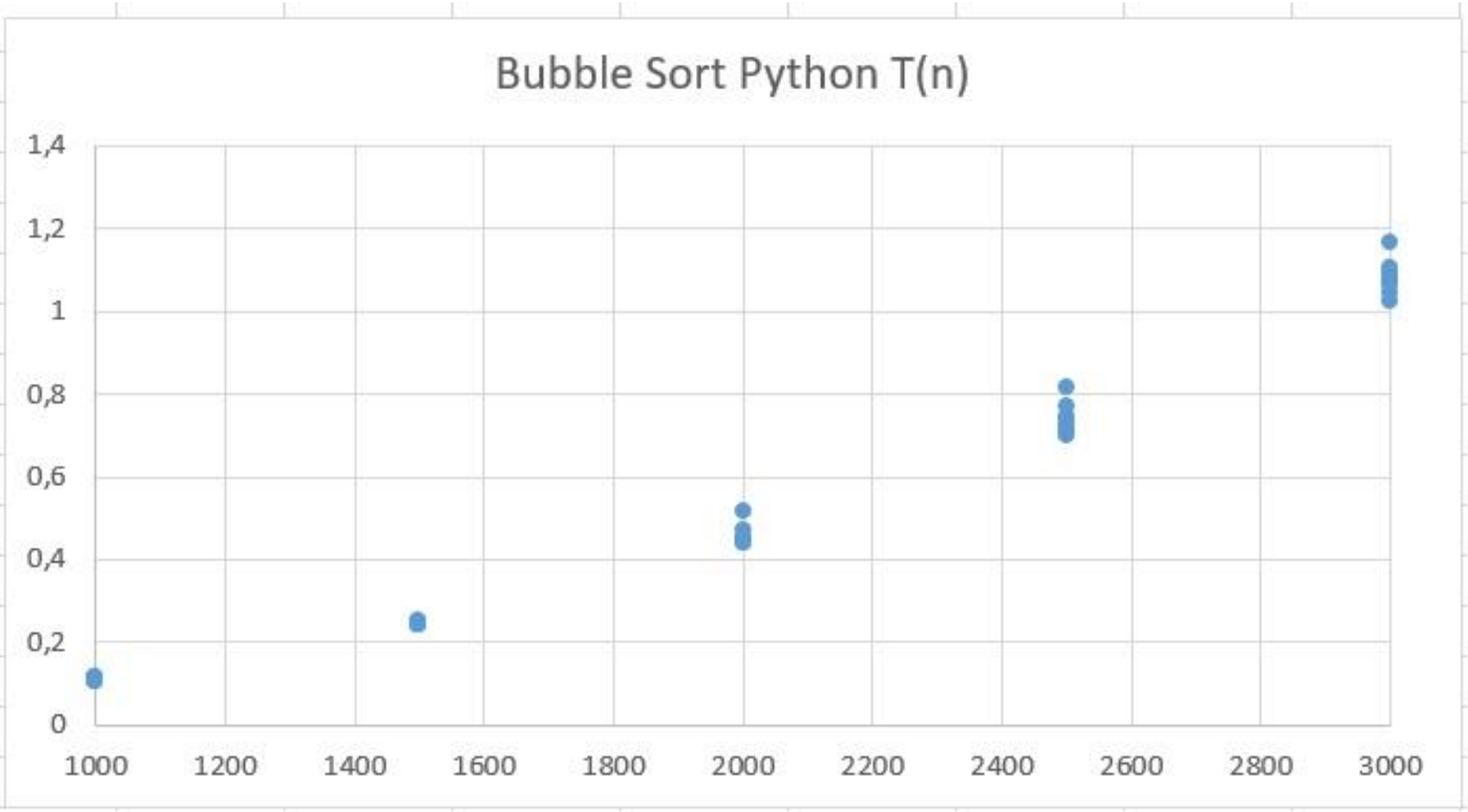
Измеряем время

```
12 import time
13 print("n;T(n)")
14 for n in range(1000, 3001, 500):
15     for _ in range(10):
16         x = list(range(n, 0, -1))
17         t0 = time.time()
18         bubble_sort(x)
19         t1 = time.time()
20         t = t1 - t0
21         print(f"{n};{t:.10f}".replace(".",","))
```

```
C:\1>python bubble_csv.py
n;T(n)
1000;0,1059956551
1000;0,1029930115
1000;0,1080000401
1000;0,1100053787
1000;0,1069927216
1000;0,1090002060
1000;0,1060047150
```

График T(n)

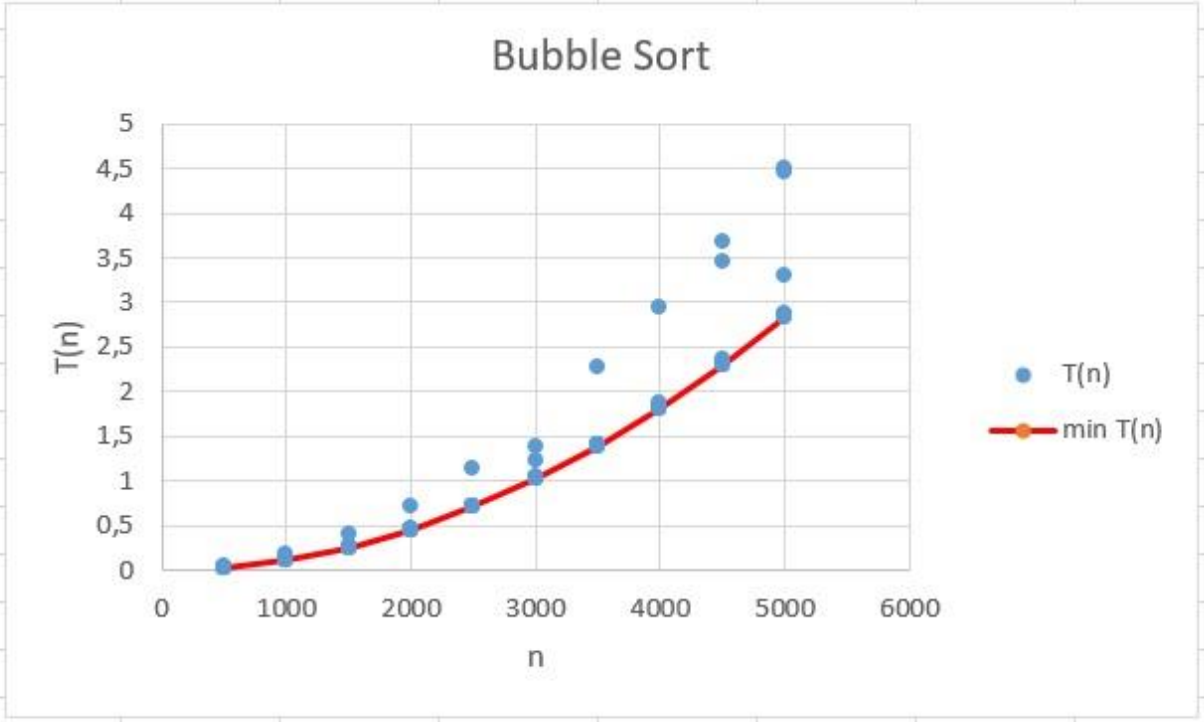
n	T(n)
1000	0,106
1000	0,111999
1000	0,114995
1000	0,111999
1000	0,116015
1000	0,11399
1000	0,108002
1000	0,107996
1000	0,111004
1000	0,106
1500	0,245995
1500	0,253006
1500	0,243999
1500	0,242997
1500	0,243004



Сводная таблица: Минимальные T(n)

```
>python bubble5.py > bubble10.csv
```

n	T(n)
500	0,031248
1000	0,109004
1500	0,26397
2000	0,454315
2500	0,703497
3000	1,019984
3500	1,405499
4000	1,827406
4500	2,30425
5000	2,855259
500	0,0204
1000	0,112232
1500	0,254936
2000	0,446522
2500	0,703553

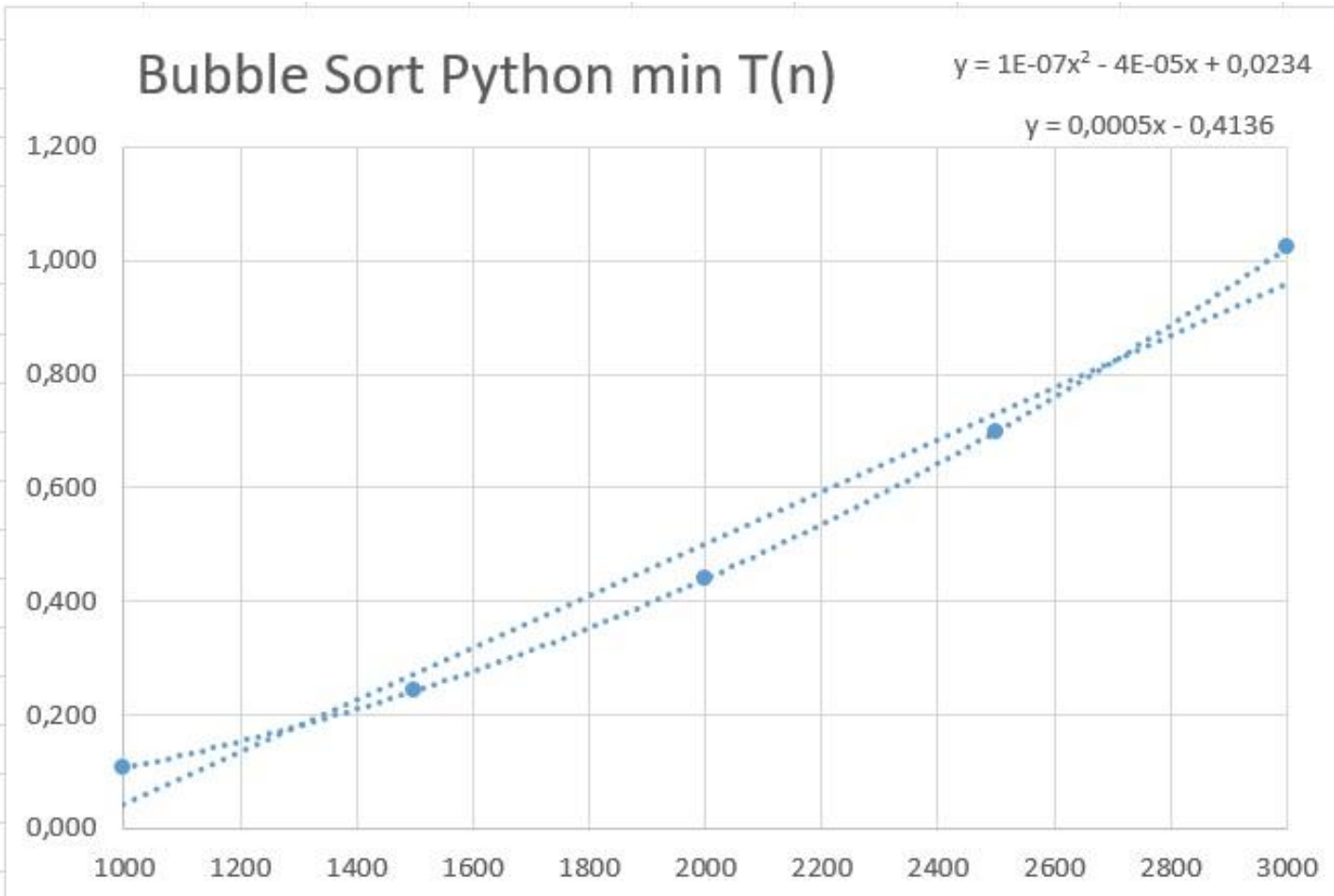


n	min T(n)
500	0,0203912258
1000	0,1041204929
1500	0,2469537258
2000	0,4406352043
2500	0,7034971714
3000	1,0169024467
3500	1,3789484501
4000	1,8149890900
4500	2,3042502403
5000	2,8357009888
Общий итог	**0,020391226**

n	min T(n)
500	0,020391
1000	0,10412
1500	0,246954
2000	0,440635
2500	0,703497
3000	1,016902
3500	1,378948
4000	1,814989
4500	2,30425
5000	2,835701

Квадратичная сложность

n	Мин T(n)
1000	0,106
1500	0,243
2000	0,440
2500	0,698
3000	1,023



Порядок полинома по графику

- $T = a x + b$
- $T = a x^2 + b x + c$
- $T = a x^3 + b x^2 + c x + d$
- $O(n^k)$ – не учитываем младшие степени и коэффициенты

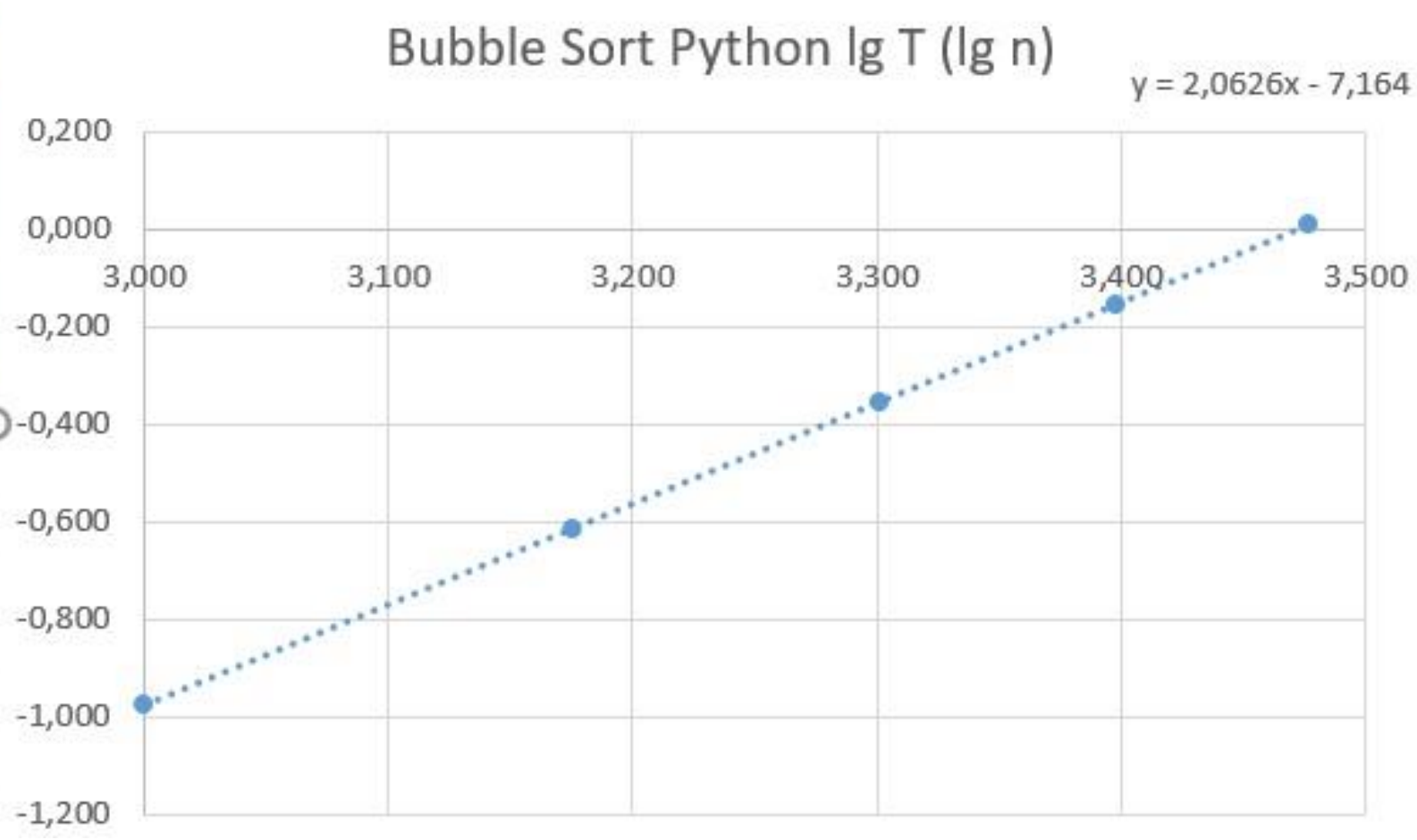
- Логарифмируем уравнение T(n)
- Строим график $\lg T = f(\lg n)$
- Определяем наклон линии

Порядок в логарифмических координатах

- $T = a\ n^2 + b\ n + c \approx a\ n^2$
- $\lg T = \lg (a\ n^2) = \lg a + 2 \lg n$
- $\lg T = A + 2 \lg n$
- Коэффициент наклона линии на графике = 2
- Порядок полинома = 2

Порядок полинома

n	Мин T(n)	lg n	lg T
1000	0,106	3,000	-0,975
1500	0,243	3,176	-0,614
2000	0,440	3,301	-0,357
2500	0,698	3,398	-0,156
3000	1,023	3,477	0,010



Оценим сложность алгоритма O(n)

Строка программы	Число операций
`for i in range(n - 1):`	
`    for j in range(0, n - i - 1):`	
`        if x[j] > x[j + 1]:`	
`            x[j], x[j + 1] = x[j + 1], x[j]`	

Сложность алгоритма

- Наихудший случай O(n)
- Наилучший случай Ω(n)

- O-микрон и O-мега

Улучшение алгоритма

- Остановиться, если больше не требуются перестановки
- Отслеживать «флаг» – число перестановок за последний проход
- Если в конце прохода флаг = 0, выход
- Оценить число дополнительных операций
- Измерить сложность в наилучшем и наихудшем случае