



OTUS

ОНЛАЙН ОБРАЗОВАНИЕ

# Онлайн-образование





# Меня хорошо видно && слышно?

Ставьте +, если все хорошо  
Напишите в чат, если есть проблемы

Проверить, идет ли запись!







# Анализ сложности алгоритмов и сортировка

---

Панкрашов Дмитрий

# Маршрут вебинара

Алгоритмы



Сложность (Big O)



Сортировки



Практика

# Цели и смысл вебинара | На занятии вы сможете

1

Понять концепцию Big O

2

Анализировать сложность алгоритма

3

Реализовать сортировку коллекции



The background of the slide is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the entire image. In the center, there is a network diagram consisting of a grid of points connected by thin lines, with some points highlighted in a lighter blue. The title text is centered over this network diagram.

# Алгоритмы и их сравнения

## Что такое “алгоритм”?



**Последовательность действий,  
для решения некоторой задачи**

**Пример: вычисление площади  
(треугольника, квадрата)**

## Сравнение алгоритмов

**Для чего нам может понадобиться  
сравнивать алгоритмы между  
собой?**

**Наводящий пример: обмен  
значений переменных**



# Сравнение алгоритмов

```
int a = 5;  
int b = 7;  
int temp;  
  
temp = b;  
b = a;  
a = temp;
```

```
int a = 5;  
int b = 7;  
  
a = a + b;  
b = a - b;  
a = a - b;
```

**По каким критериям мы можем  
сравнить алгоритмы?**



# Сравнение алгоритмов - критерии

- **Память**
- **Время выполнения (скорость)**

# Сравнение алгоритмов - способы

- **Аналитический**
- **Экспериментальный**





Big O

# O - нотация

**Сложность алгоритма** - это количественная характеристика, которая говорит о том, сколько времени \ памяти потребуется для выполнения алгоритма.

**Big O** показывает, как **сложность** алгоритма растёт с увеличением входных данных. Она всегда показывает худший вариант развития событий - верхнюю границу.

Зачем изучать Big O:

- Чтобы уметь видеть и исправлять неоптимальный код.
- Спрашивают на собеседованиях.
- Потеря производительности от непонимания Big O.

# Попробуем разобраться...

Допустим, у нас есть массив:

```
var array = new int[] {1, 2, 3, 4, 5};
```

Какова сложность доступа к элементу?

```
array[0]
```

```
array[1]
```

```
array[2]
```



# Сложность $O(1)$ - константная

```
var array = new int[] {1, 2, 3, 4, 5};  
array[0]  
array[1]  
array[2]
```

Для доступа к элементу массива время, затраченное на его получение - константа.

Сложность  $O(1)$  означает, что время, затрачиваемое на выполнение не зависит от входных данных.

# Ищем значение в массиве

```
var array = new int[] {7, 18, 158, 16, 23};

public void FindElement(int[] array, int element)
{
    for(int i = 0; i < array.Length; i++)
    {
        if(array[i] == element)
            // do something
    }
}
```

Сколько элементов надо перебрать, чтобы найти нужный?

# Сложность $O(n)$ - линейная

Сложность алгоритма увеличивается линейно с увеличением входных данных.

FindElement на массиве из 10 элементов отработает за  $X$  микросекунд.

А на массиве из 100..10 000 элементов уже за  $X*10 \dots X*100$  микросекунд



# Ищем повторяющееся значение в массиве

```
var array = new int[] {7, 18, 158, 16, 23};

public void FindElement(int[] array, int element)
{
    for(int i = 0; i < array.Length; i++)
    {
        for(int j = 0; j < array.Length; j++)
        {
            if(array[i] == array[j] && i != j)
                // do something
        }
    }
}
```

# Сложность $O(n^2)$ - квадратичная

Удвоение размера входных данных увеличивает время выполнения в 4 раза.

Например, при увеличении данных в 10 раз, количество операций (и время выполнения) увеличится примерно в 100 раз.

Если алгоритм имеет квадратичную сложность, то это повод пересмотреть необходимость использования данного алгоритма. Но иногда этого не избежать.

# Ищем значение в массиве

```
var array = new int[] {7, 16, 18, 23, 158};
```

```
public void FindElement(int[] array, int element)
{
    int median = array.Length / 2;
    int startIndex = 0;
    if (array[median] > element)
    {
        startIndex = median;
    }
    for(int i = 0; i < array.Length; i++)
    {
        if(array[i] == element)
            // do something
    }
}
```



# Сложность $O(\log n)$ - логарифмическая

Сложность алгоритма растёт логарифмически с увеличением входных данных.

Другими словами это такой алгоритм, где на каждой итерации берётся половина элементов.

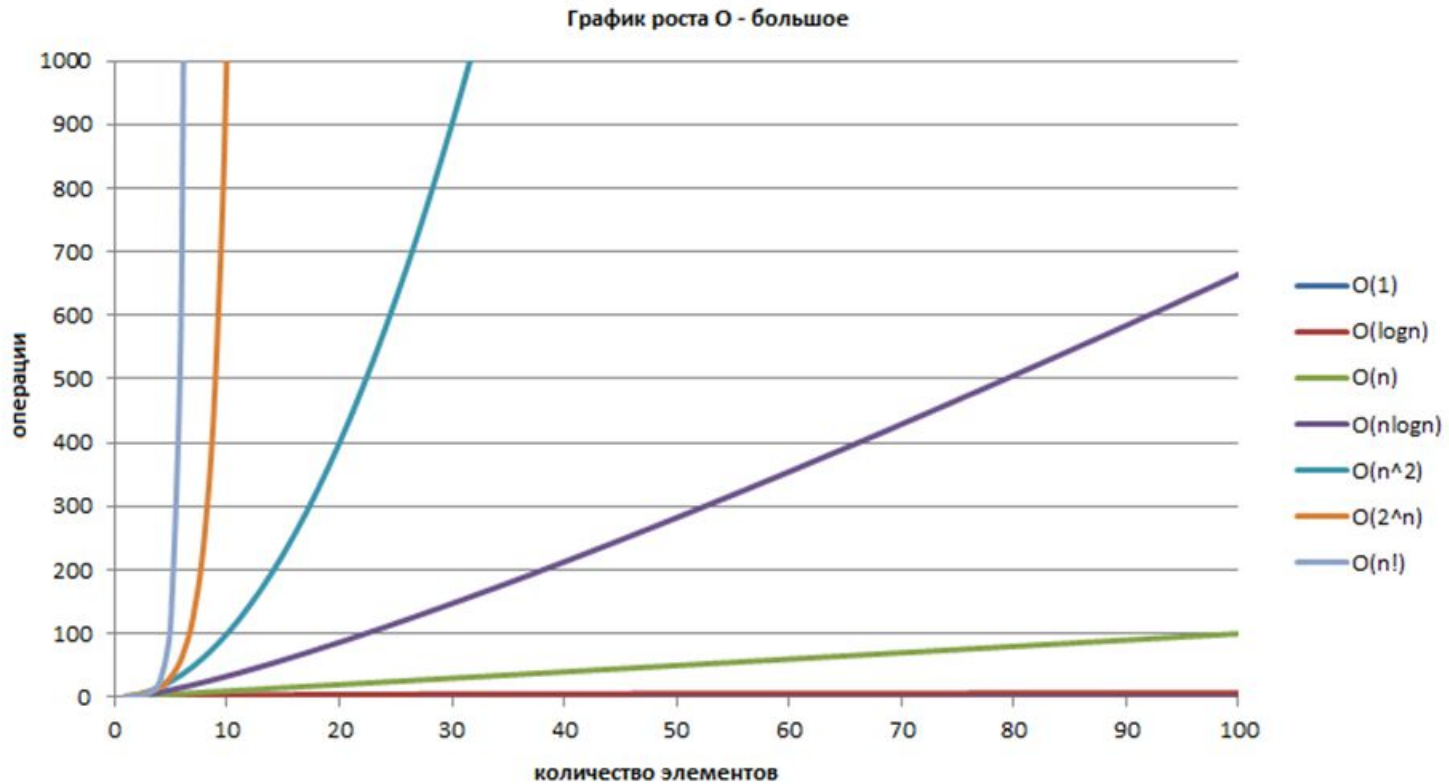
# Сложность $O(n * \log n)$ - линейризованная

$$O(n) * O(\log n)$$

Удвоение размера входных данных увеличит время выполнения чуть более, чем вдвое.

**Можно ли улучшить пример с поиском повторяющихся значений с  $O(n^2)$  до  $O(n * \log n)$  ?**

# Big O - визуализация





The image features a background of a dense city skyline, likely New York City, with numerous skyscrapers. The entire image is overlaid with a semi-transparent blue and teal gradient. A network of white lines and dots is superimposed on the background, creating a digital or technological feel. The word "Сортировки" is centered in the middle of the image in a large, white, sans-serif font.

# Сортировки

# Алгоритмы сортировок

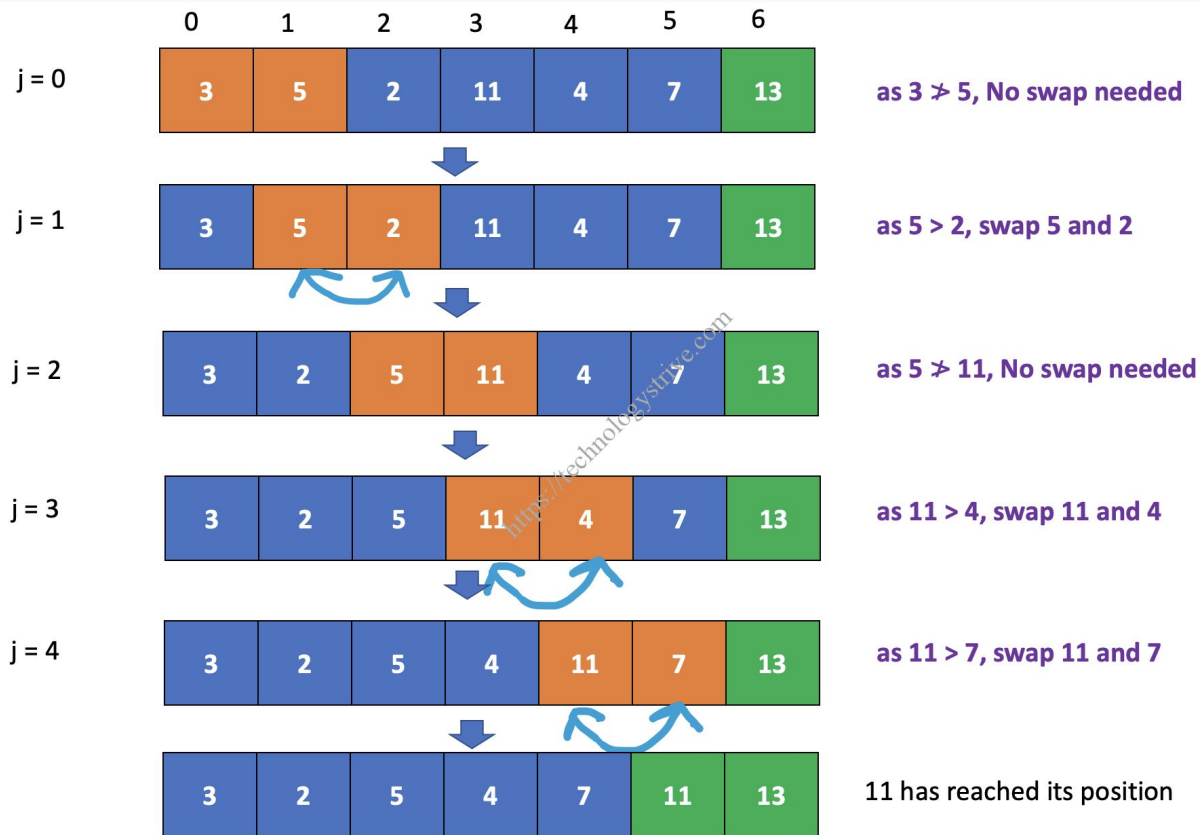
- Пузырьком
- Быстрая
- Вставками
- Перемешиванием
- Слиянием
- Расческой
- ...
- их слишком много

# Сложности алгоритмов сортировок

| Алгоритм                 | Структура данных | Временная сложность |                |                | Вспомогательные данные |
|--------------------------|------------------|---------------------|----------------|----------------|------------------------|
|                          |                  | Лучшее              | В среднем      | В худшем       | В худшем               |
| Быстрая сортировка       | Массив           | $O(n \log(n))$      | $O(n \log(n))$ | $O(n^2)$       | $O(n)$                 |
| Сортировка слиянием      | Массив           | $O(n \log(n))$      | $O(n \log(n))$ | $O(n \log(n))$ | $O(n)$                 |
| Пирамидальная сортировка | Массив           | $O(n \log(n))$      | $O(n \log(n))$ | $O(n \log(n))$ | $O(1)$                 |
| Пузырьковая сортировка   | Массив           | $O(n)$              | $O(n^2)$       | $O(n^2)$       | $O(1)$                 |
| Сортировка вставками     | Массив           | $O(n)$              | $O(n^2)$       | $O(n^2)$       | $O(1)$                 |
| Сортировка выбором       | Массив           | $O(n^2)$            | $O(n^2)$       | $O(n^2)$       | $O(1)$                 |
| Блочная сортировка       | Массив           | $O(n+k)$            | $O(n+k)$       | $O(n^2)$       | $O(nk)$                |
| Поразрядная сортировка   | Массив           | $O(nk)$             | $O(nk)$        | $O(nk)$        | $O(n+k)$               |

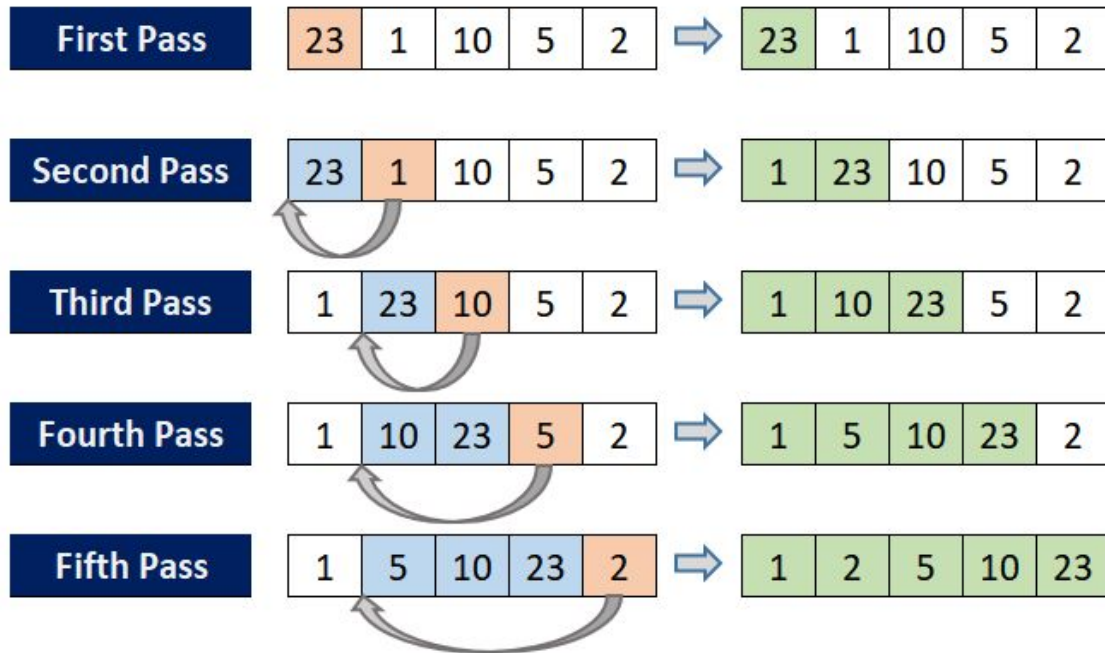


# Сортировка пузырьком

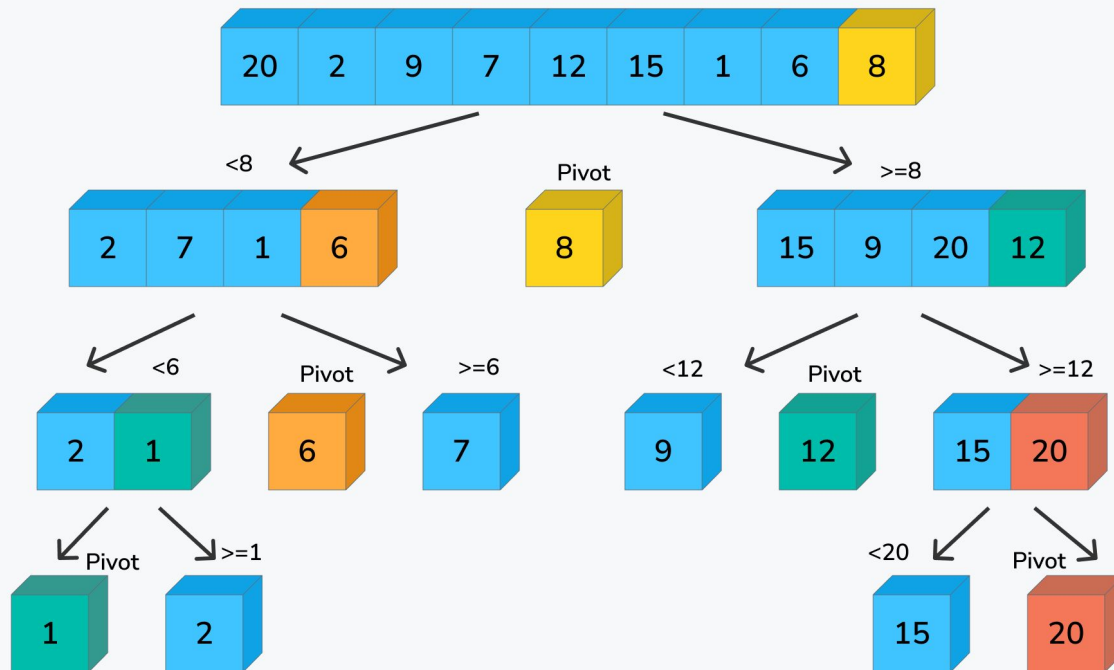




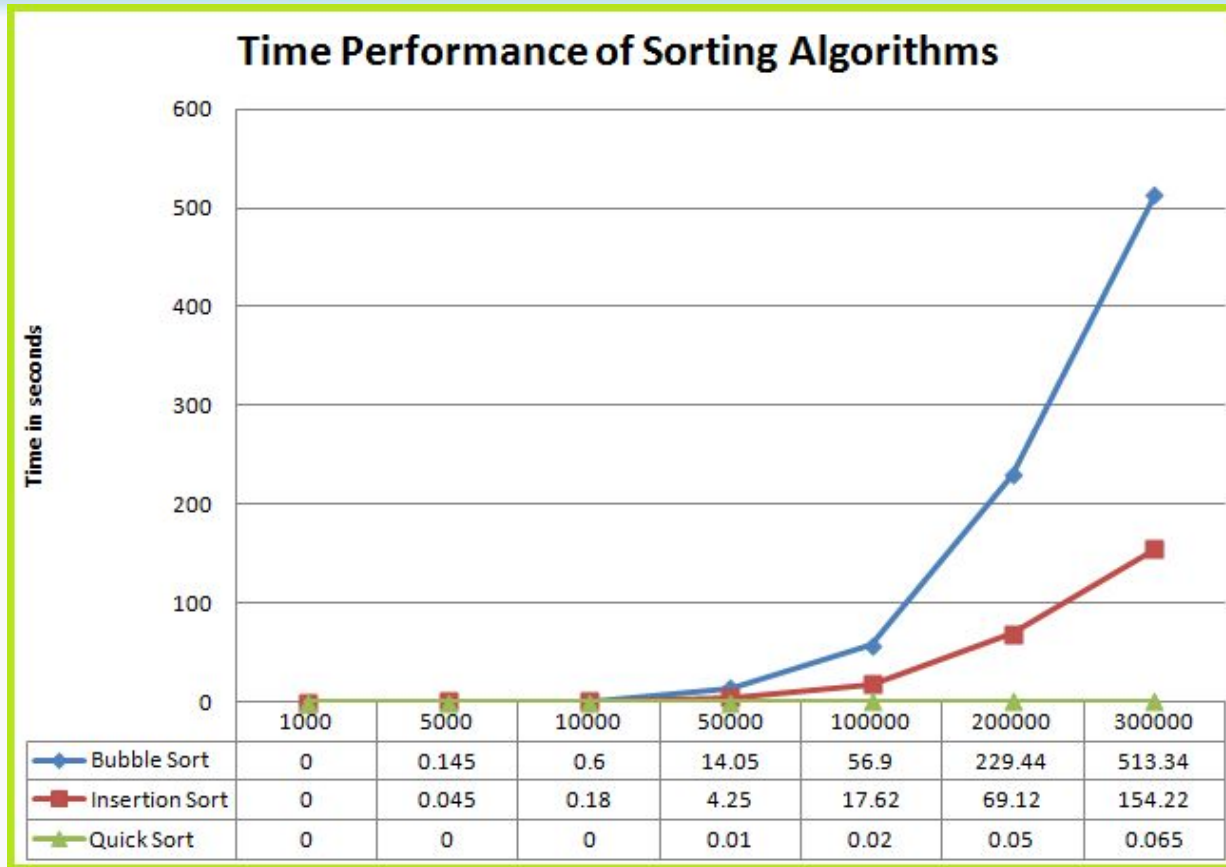
# Сортировка вставками



# Сортировка быстрая



# Сложность сортировок

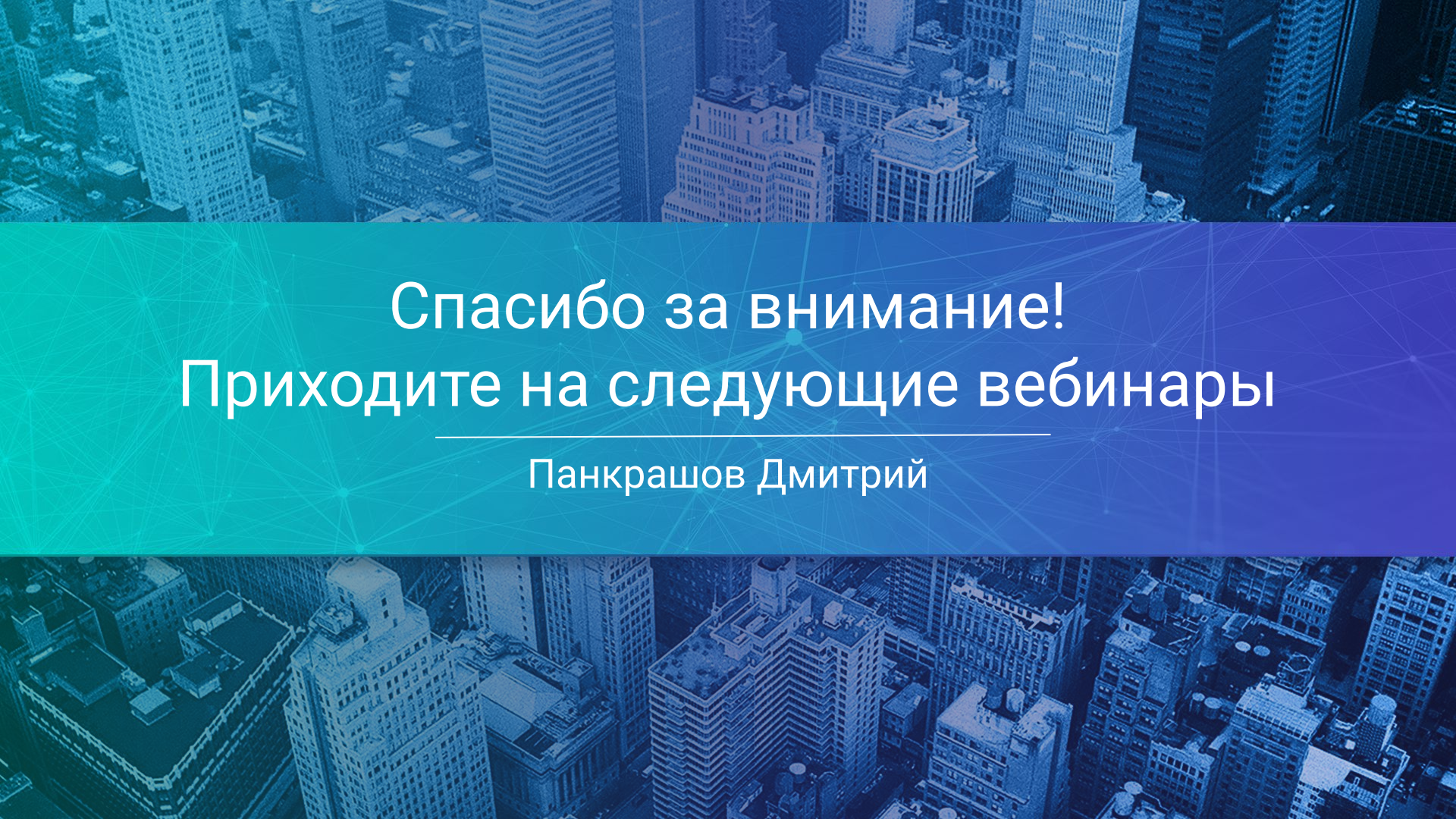




The image features a high-angle, aerial view of a dense urban skyline, likely New York City, with numerous skyscrapers and buildings. The entire image is overlaid with a semi-transparent blue layer. A network of thin, light blue lines connects various points across the blue area, creating a digital or technological aesthetic. The word "ПРАКТИКА" is centered in the middle of the image in a bold, white, sans-serif font.

# ПРАКТИКА





Спасибо за внимание!  
Приходите на следующие вебинары

---

Панкрашов Дмитрий