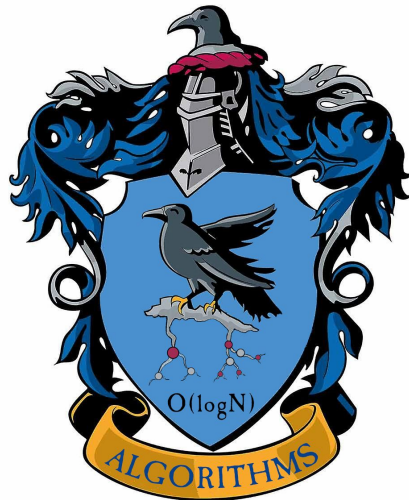


Алгоритмы и структуры данных

Элементарные сортировки



Задача

В **отсортированном** по возрастанию массиве из уникальных элементов найти **индекс** элемент со **значением** x . Если такого нет, то вернуть -1 .

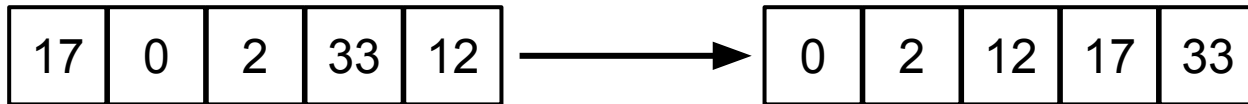
1	2	7	13	35	37	90	93	99
---	---	---	----	----	----	----	----	----

Задача

Задан массив из **уникальных** элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.

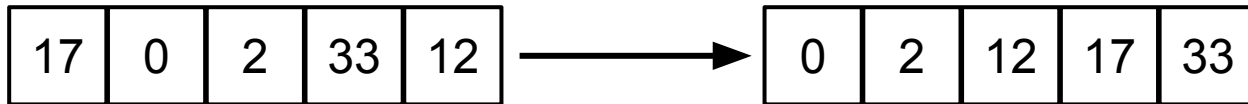
Задача

Задан массив из **уникальных** элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.



Задача

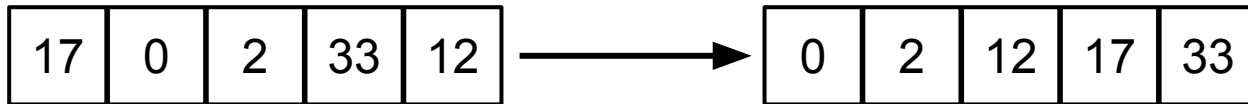
Задан массив из **уникальных** элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.



Алгоритм?

Задача

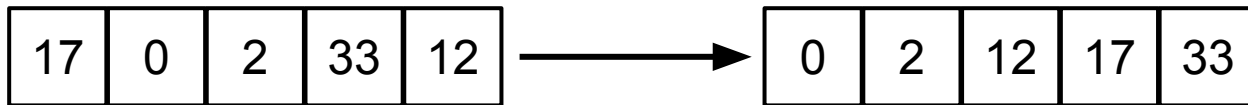
Задан массив из **уникальных** элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.



Алгоритм? Пузырёк

Задача

Задан массив из **уникальных** элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.



Алгоритм? Пузырёк Сортировка выбором!

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

17	0	2	33	12
----	---	---	----	----

ищем элемент, который должен стоять на позиции 0

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

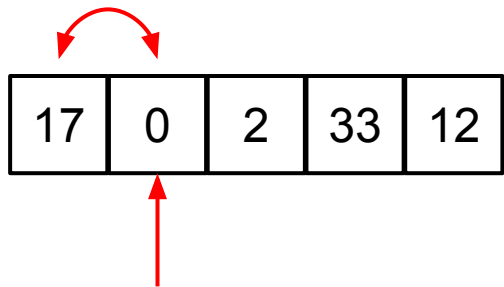
17	0	2	33	12
----	---	---	----	----



ищем элемент, который должен стоять на позиции 0

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.



ищем элемент, который должен стоять на позиции 0
меняем с нулевым элементом

Сортировка выбором

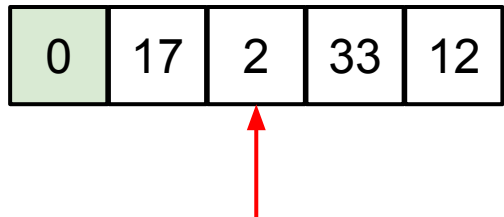
Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

0	17	2	33	12
---	----	---	----	----

ищем элемент, который должен стоять на позиции 0
меняем с нулевым элементом
повторяем для хвоста массива

Сортировка выбором

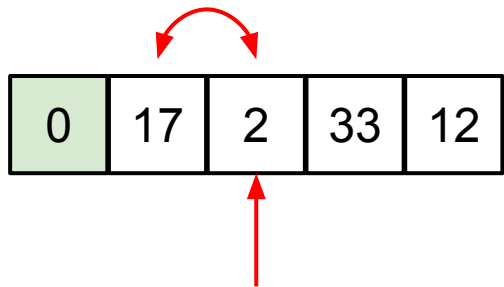
Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.



ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.



ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

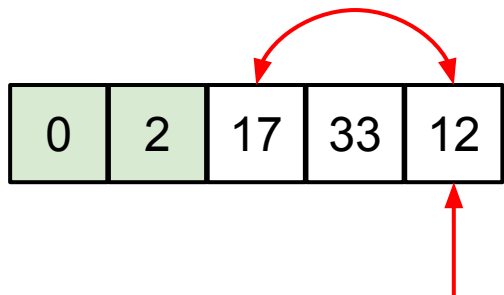
Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

0	2	17	33	12
---	---	----	----	----

ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.



ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

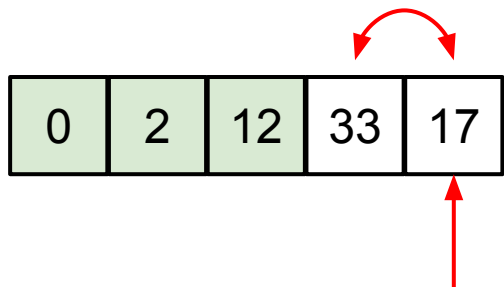
Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

0	2	12	33	17
---	---	----	----	----

ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.



ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

Идея: найти для каждой позиции в массиве подходящий элемент из оставшихся.

0	2	12	17	33
---	---	----	----	----



ищем элемент, который должен стоять на позиции i
меняем с i -ым элементом
повторяем для хвоста массива

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

} Поиск в
хвосте

→ Меняем элементы с индексами i, min_index местами

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

} Поиск в хвосте



Меняем элементы с индексами i, min_index местами

Сложность?

Сортировка выбором

Какой **худший** случай?

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

} Поиск в
хвосте



Меняем элементы с индексами i, min_index местами

Сложность?

Сортировка выбором

Какой **худший** случай?

Любой!

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

} Поиск в хвосте



Меняем элементы с индексами i, min_index местами

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

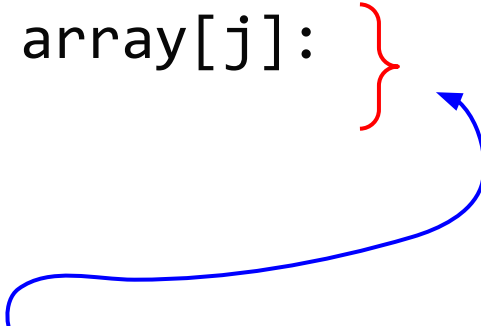
Сколько раз
выполнится
тело цикла?

} ?

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

Сколько раз
выполнится
тело цикла?


$$(n - 1) + (n - 2) + \dots + 1$$

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1)$$

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1) = C * \left(\frac{1 + (n - 1)}{2}\right) * (n - 1)$$

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1) = C * \underbrace{\left(\frac{1 + (n - 1)}{2}\right) * (n - 1)}$$

сумма арифм. прогрессии

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1) = C * \left(\frac{1 + (n - 1)}{2} * (n - 1)\right) = C * \left(\frac{n^2}{2} - \frac{n}{2}\right)$$

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1) = C * \left(\frac{1 + (n - 1)}{2} * (n - 1)\right) = C * \left(\frac{n^2}{2} - \frac{n}{2}\right)$$

$$T(n) = O(n^2)$$

Сортировка выбором

Оценка сложности

$$T(n) \leq C * ((n - 1) + (n - 2) + \dots + 1) = C * \left(\left(\frac{1 + (n - 1)}{2}\right) * (n - 1)\right) = C * \left(\frac{n^2}{2} - \frac{n}{2}\right)$$

$$T(n) = O(n^2)$$

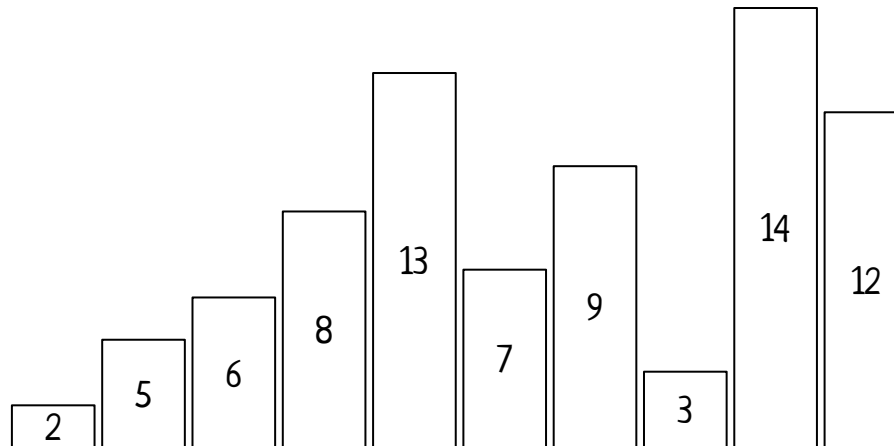


Можем ли
мы лучше?

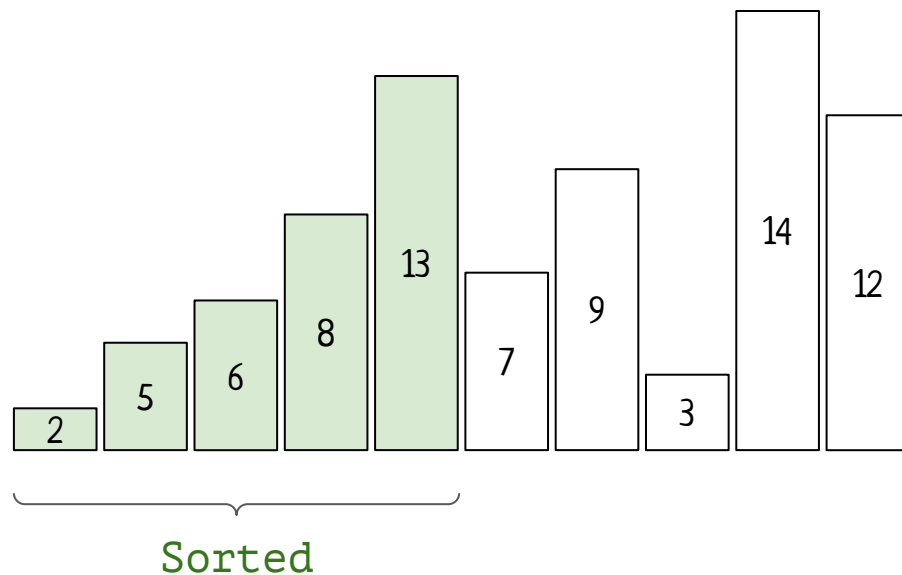
Сортировка вставками

Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.

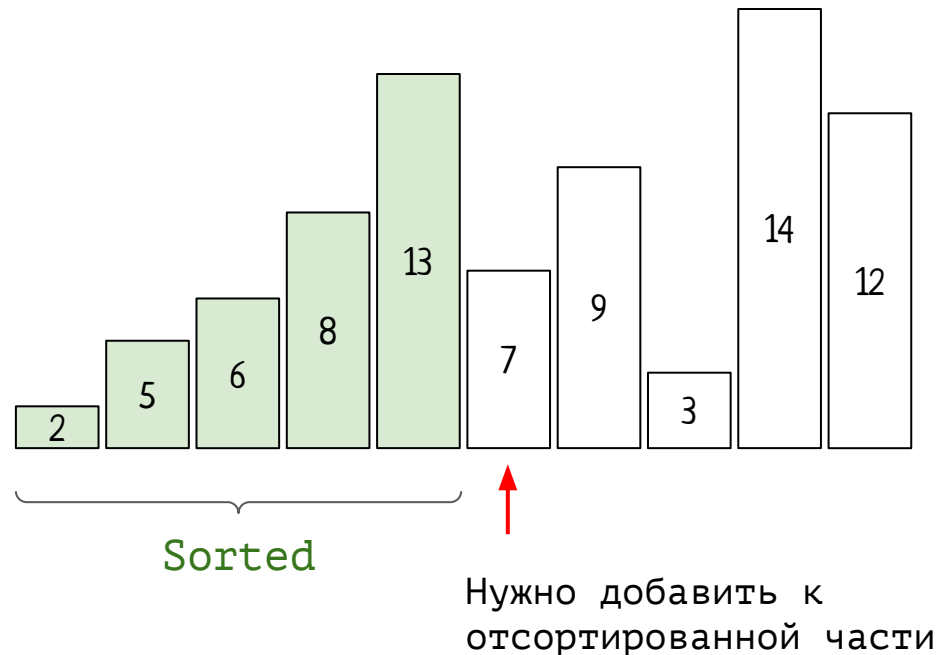
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



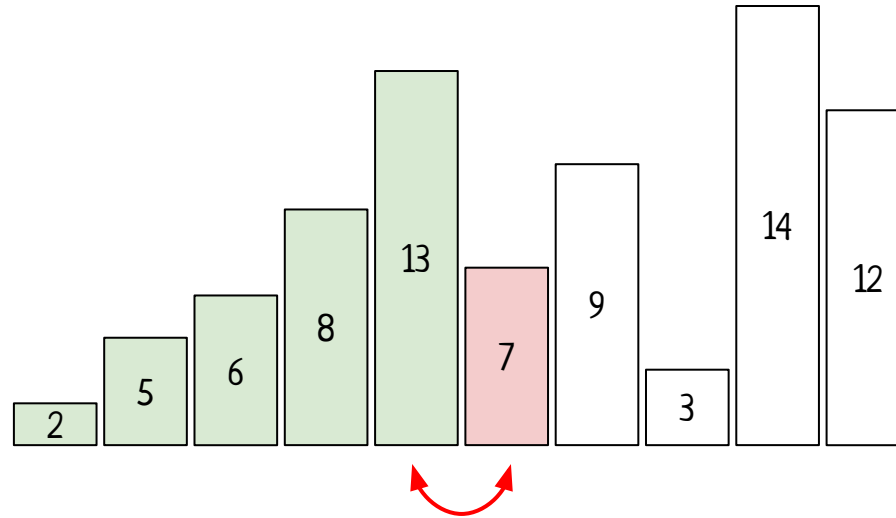
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



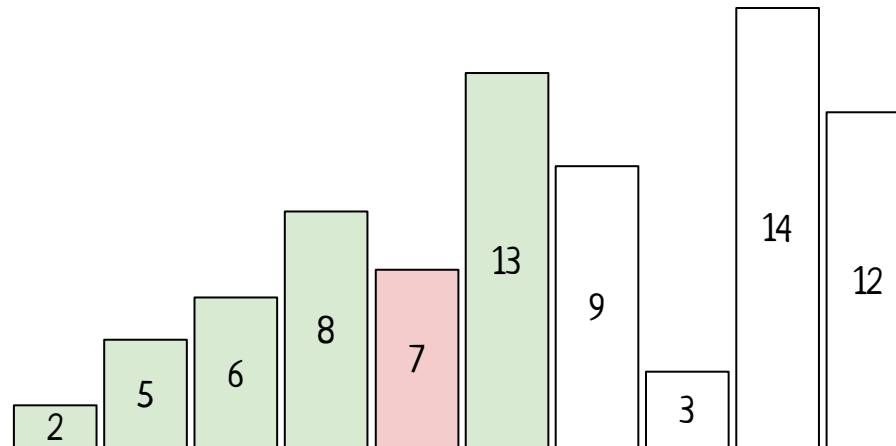
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.

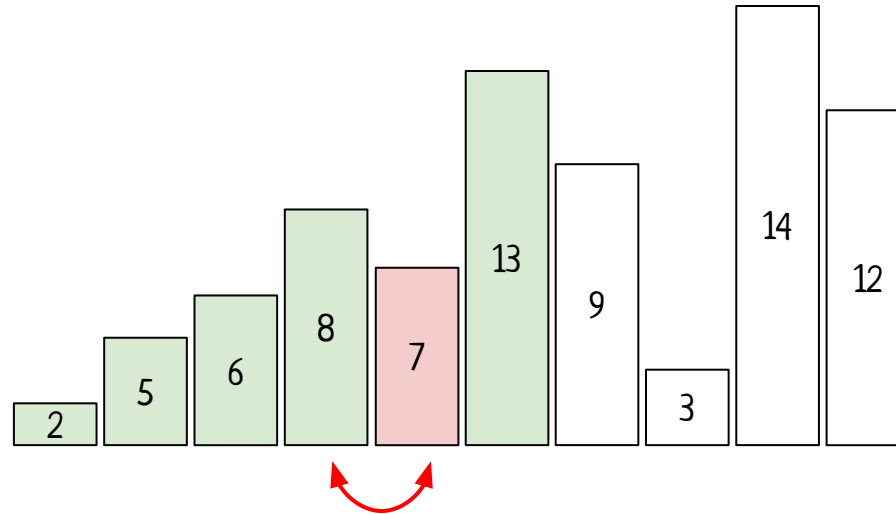


Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.

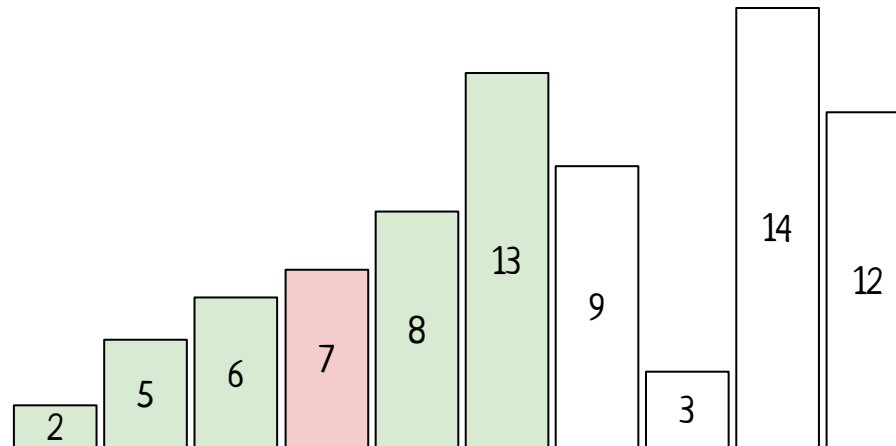


Продолжаем двигать элемент влево, пока он не встанет на свое место

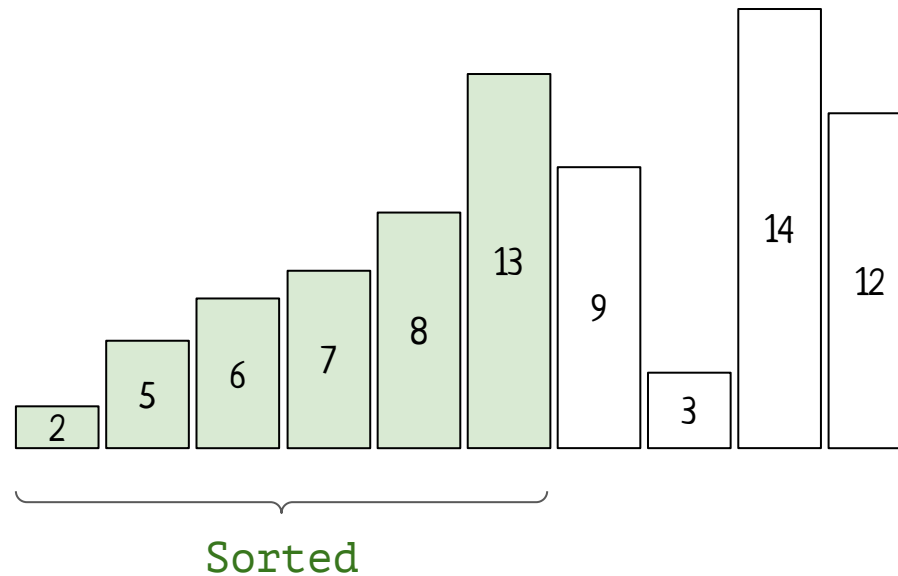
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



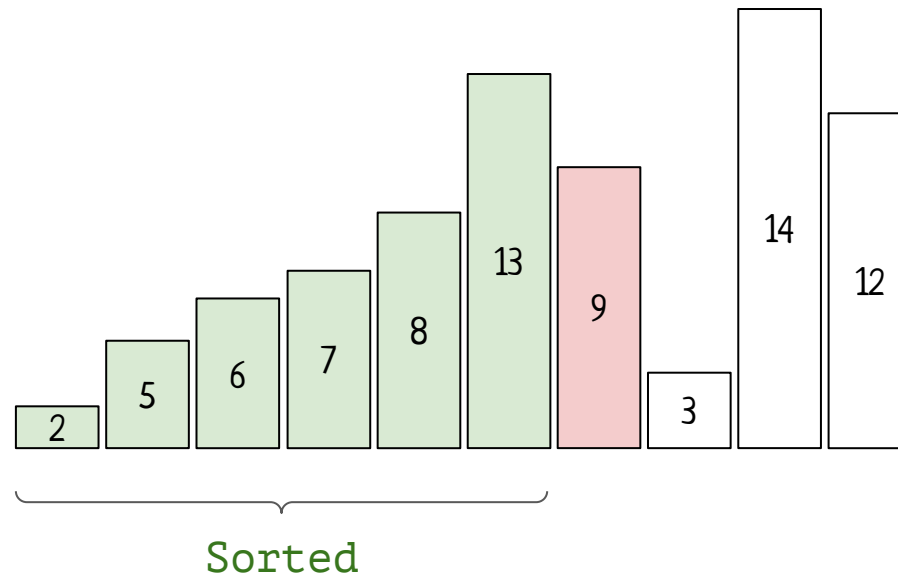
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



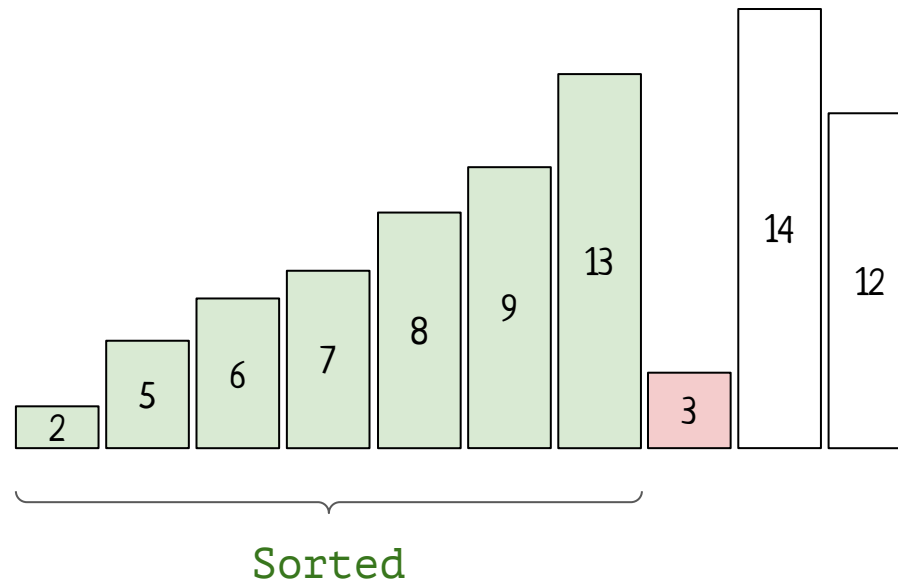
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



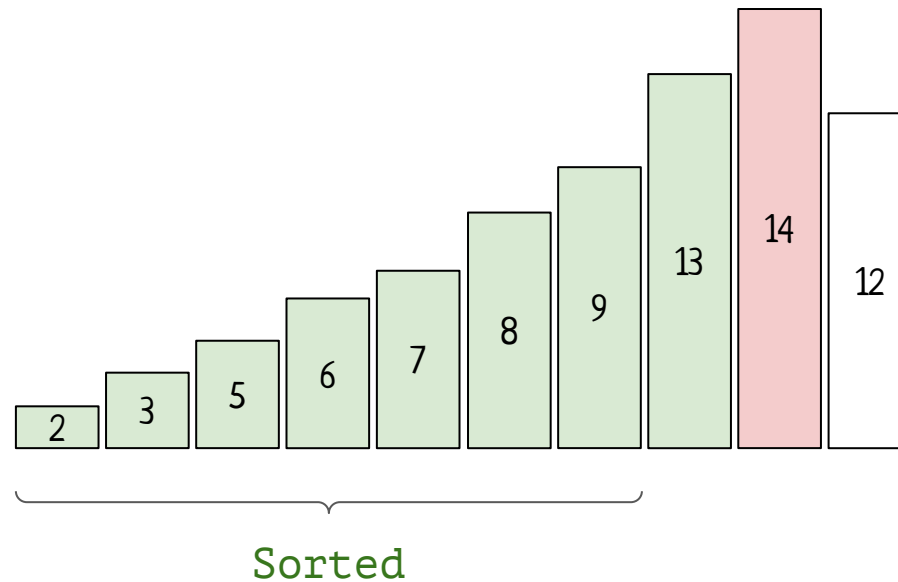
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



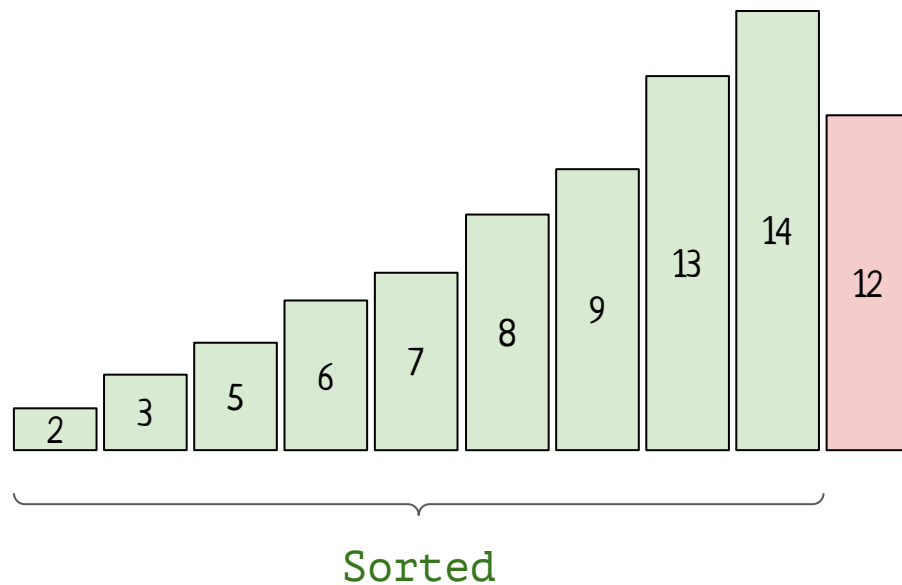
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



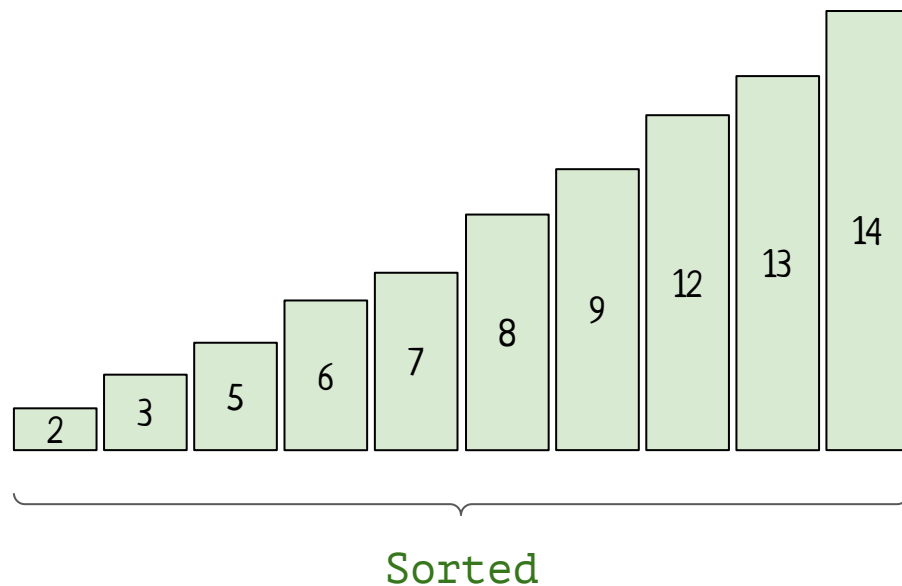
Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



Идея: поддерживаем инвариант: в левой части массива отсортированные элементы, в правой - еще нет.



Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Сортировка вставками


```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Сложность?

В этот раз худший случай - это **важно**, т.к.
количество операций зависит от того, насколько
далеко "ходим назад"!

Худший случай?




Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):   $c_1 * N$  операций  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):  ←  $c_1 * N$  операций  
         $c_2 * (N - 1)$  → j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):   $c_1 * N$  операций  
         $c_2 * (N - 1)$    $j = i$   
         $c_3 * \sum_{j=1}^{N-1} t_j$    $\text{while } j > 0 \text{ and array}[j - 1] > \text{array}[j]:$   
            swap(array, j - 1, j)  
            j -= 1
```

где t_j - количество проверок в цикле для j -ого элемента

Сортировка вставками

```
def insertion_sort(array: int[]):
```

```
    for i in [1, len(array)): ←  $c_1 * N$  операций
```

```
         $c_2 * (N - 1)$  → j = i
```

```
         $c_3 * \sum_{j=1}^{N-1} t_j$  → while j > 0 and array[j - 1] > array[j]:
```

```
             $c_4 * \sum_{j=1}^{N-1} (t_j - 1)$  → swap(array, j - 1, j)
```

```
             $c_5 * \sum_{j=1}^{N-1} (t_j - 1)$  → j -= 1
```

где t_j - количество проверок в цикле для j -ого элемента

Сортировка вставками

```
def insertion_sort(array: int[]):
```

```
    for i in [1, len(array)): ←  $c_1 * N$  операций
```

```
         $c_2 * (N - 1)$  → j = i
```

```
         $c_3 * \sum_{j=1}^{N-1} t_j$  → while j > 0 and array[j - 1] > array[j]:
```

```
             $c_4 * \sum_{j=1}^{N-1} (t_j - 1)$  → swap(array, j - 1, j)
```

```
             $c_5 * \sum_{j=1}^{N-1} (t_j - 1)$  → j -= 1
```

где t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Лучший случай: ?

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Лучший случай: $t_j = 1$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Лучший случай: $t_j = 1$

$$\text{Тогда: } T(N) = c_1 * N + c_2 * (N - 1) + c_3 * (N - 1) = O(N)$$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Лучший случай: $t_j = 1$

$$\text{Тогда: } T(N) = c_1 * N + c_2 * (N - 1) + c_3 * (N - 1) = O(N)$$

Что это за массив?

Сортировка вставками

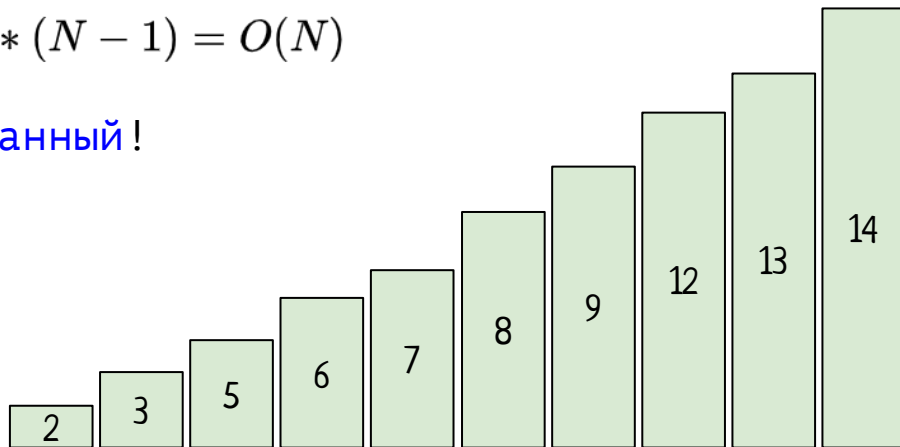
Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Лучший случай: $t_j = 1$

Тогда: $T(N) = c_1 * N + c_2 * (N - 1) + c_3 * (N - 1) = O(N)$

Что это за массив? Уже **отсортированный**!



Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай?

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай? Каждый раз сравниваемся со всеми!

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай? Каждый раз сравниваемся со всеми!
Что это за массив?

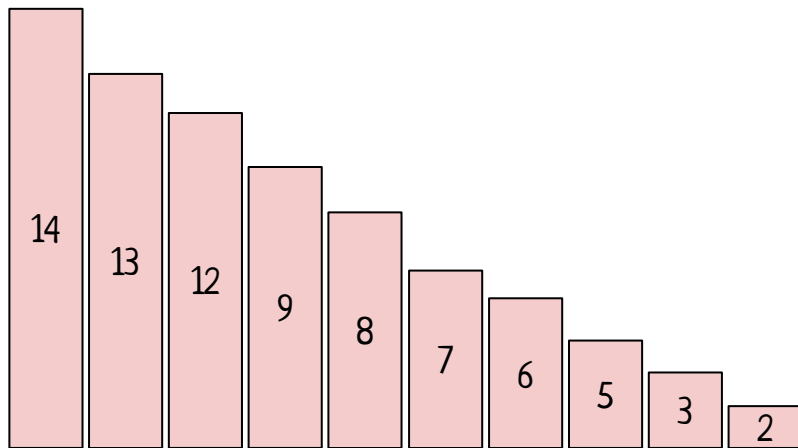
Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай? Каждый раз сравниваемся со всеми!

Что это за массив? Отсортированный в обратном порядке.



Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай: каждый раз сравниваемся со всеми, т.е. $t_j = j$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай: каждый раз сравниваемся со всеми, т.е. $t_j = j$

Тогда: $\sum_{j=1}^{N-1} j = \frac{(N-1)*N}{2}$

$$\sum_{j=1}^{N-1} (j - 1) = \frac{(N-1)*(N-2)}{2}$$

И соответственно: $T(N) = a * N^2 + b * N + c = O(N^2)$

Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай: каждый раз сравниваемся со всеми, т.е. $t_j = j$

Тогда: $\sum_{j=1}^{N-1} j = \frac{(N-1)*N}{2}$

$$\sum_{j=1}^{N-1} (j - 1) = \frac{(N-1)*(N-2)}{2}$$

И соответственно: $T(N) = a * N^2 + b * N + c = O(N^2)$



Сортировка вставками

Если t_j - количество проверок в цикле для j -ого элемента

$$T(N) = c_1 * N + c_2 * (N - 1) + c_3 * \sum_{j=1}^{N-1} t_j + (c_4 + c_5) * \sum_{j=1}^{N-1} (t_j - 1)$$

Худший случай: каждый раз сравниваемся со всеми, т.е. $t_j = j$

$$\begin{aligned} \text{Тогда: } \sum_{j=1}^{N-1} j &= \frac{(N-1)*N}{2} \\ \sum_{j=1}^{N-1} (j - 1) &= \frac{(N-1)*(N-2)}{2} \end{aligned}$$

И соответственно: $T(N) = a * N^2 + b * N + c = O(N^2)$

Но это уже **лучше**, чем сортировка выбором!
На **некоторых** данных ведет себя хорошо.

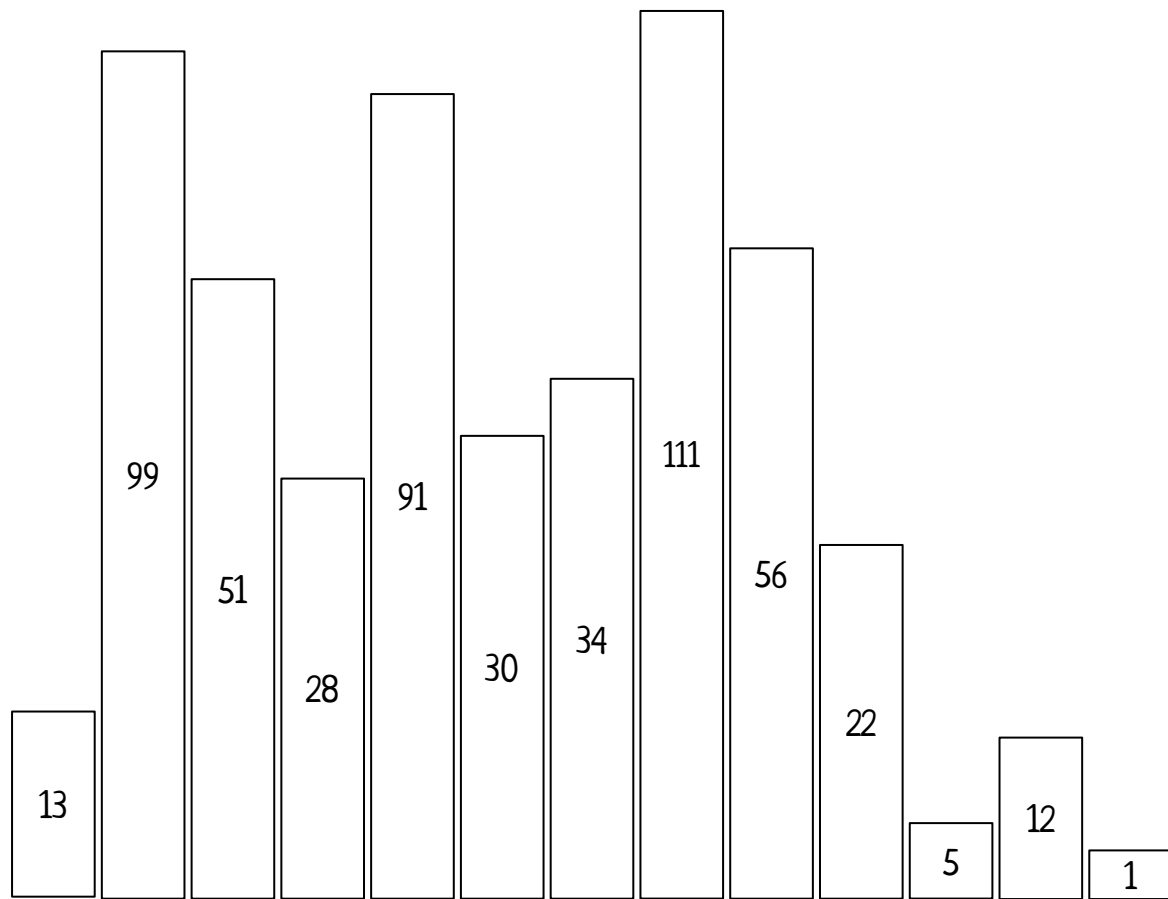


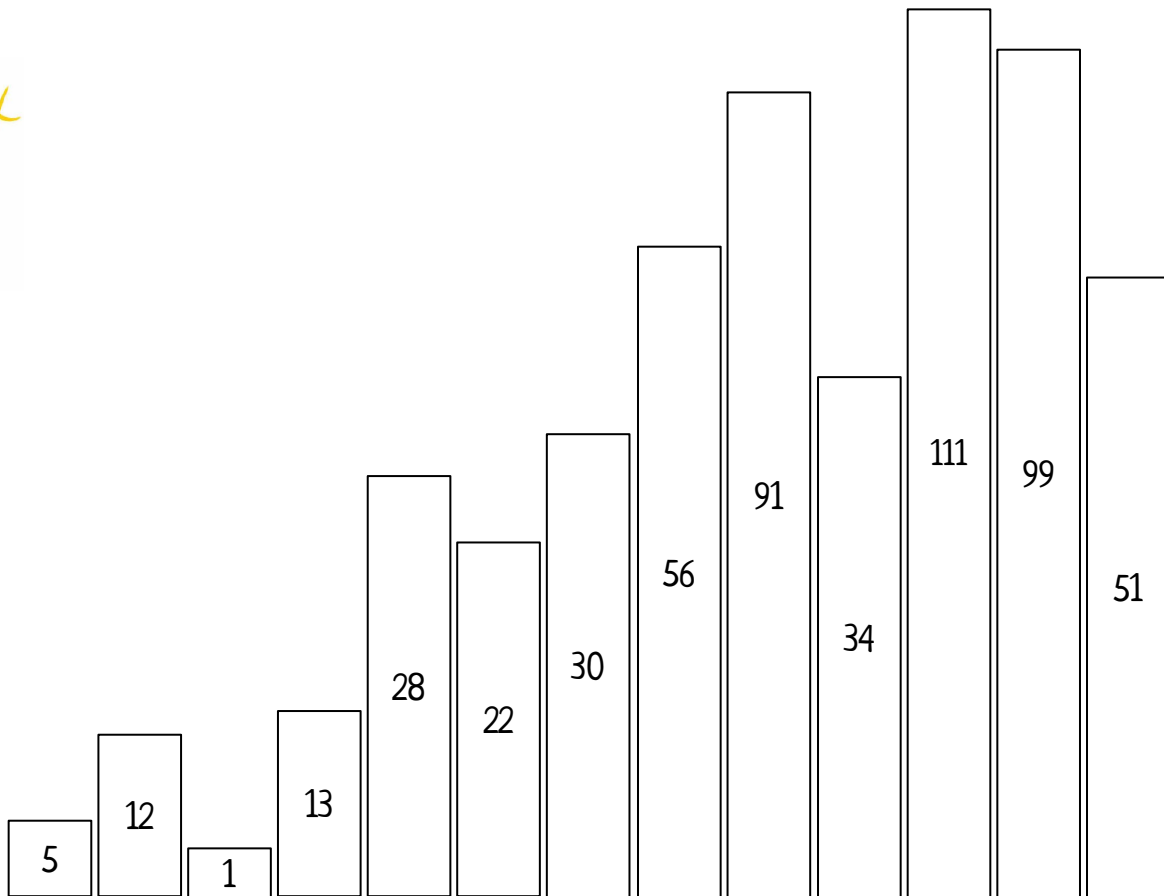
Сортировка Шелла

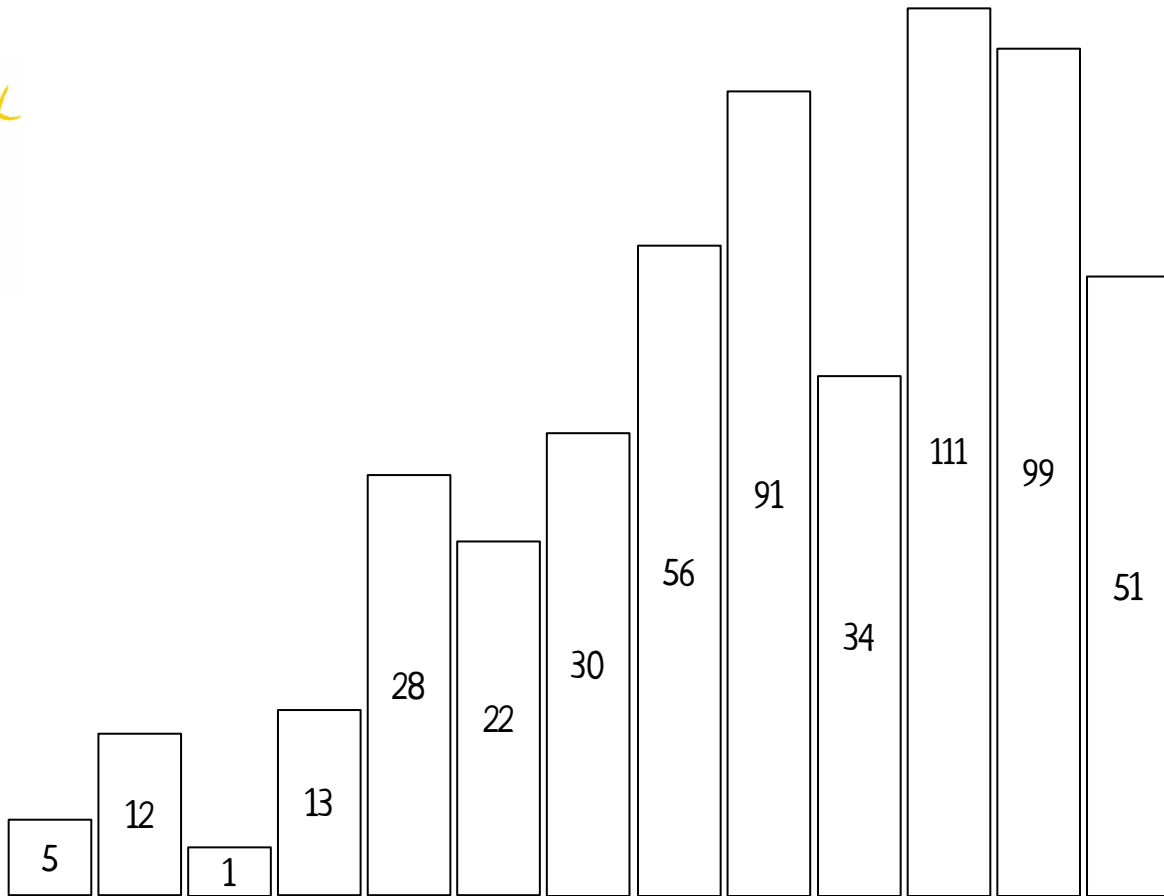
Идея: как исправить недостатки сортировки вставками?

Сортировка Шелла

Идея: как исправить недостатки сортировки вставками? А что, если предварительно "подготовить" массив?

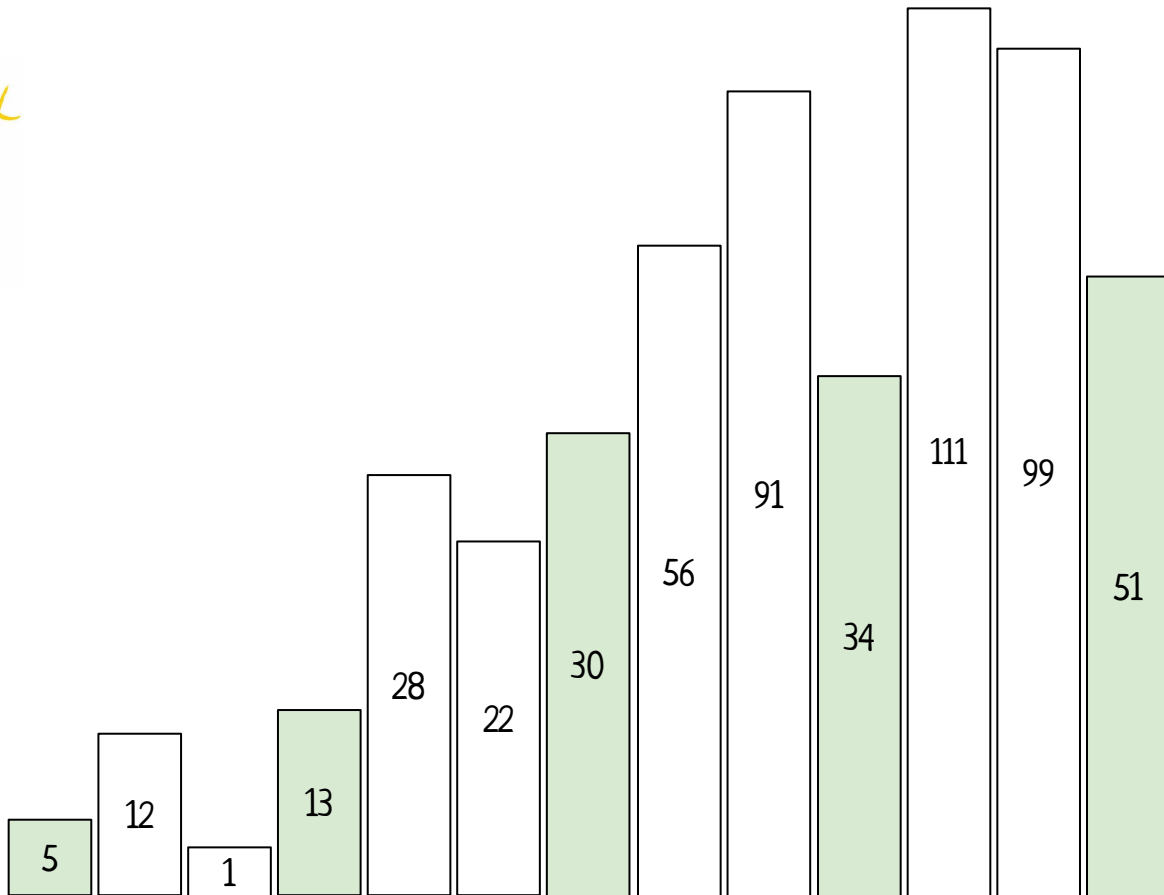






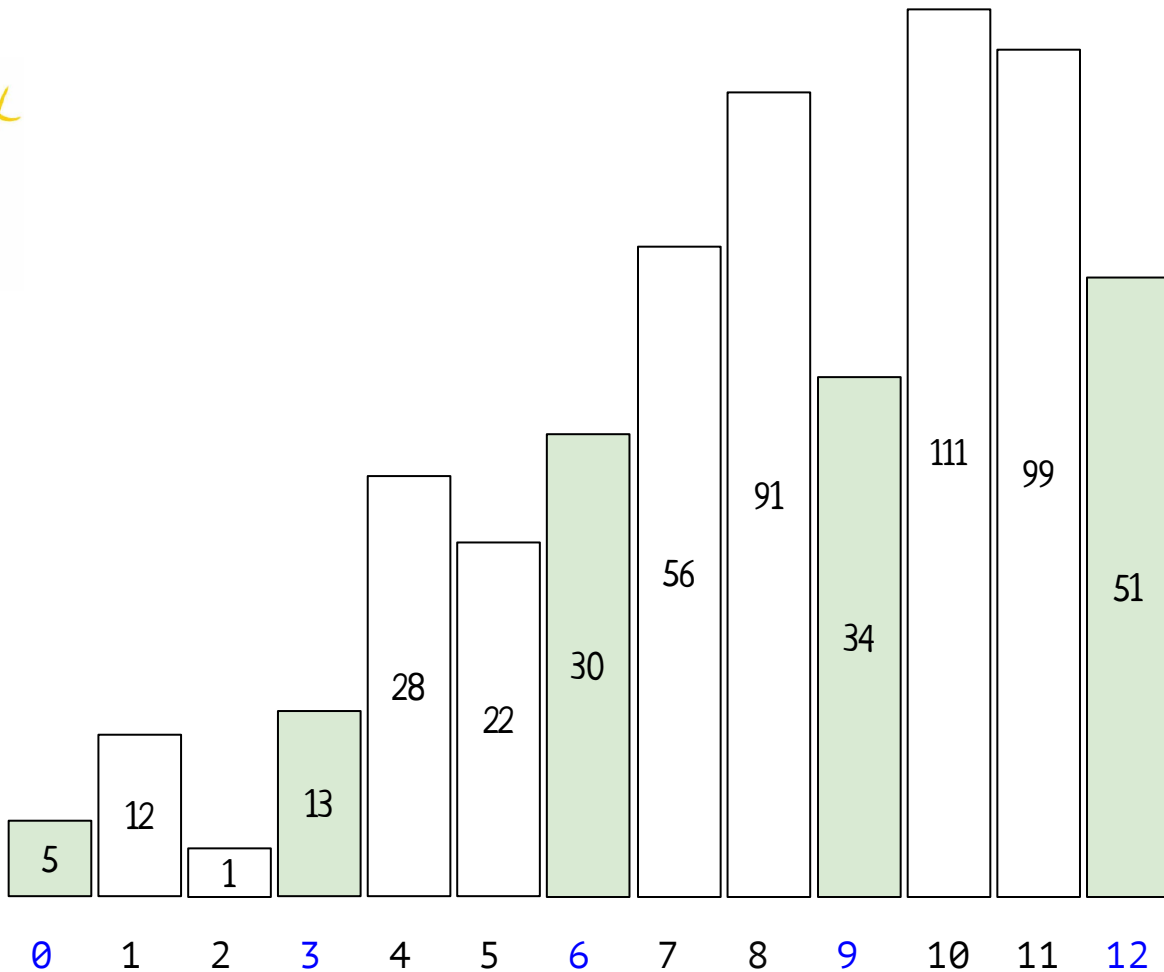
Стало ли лучше?

Что можете
сказать про
такой массив?



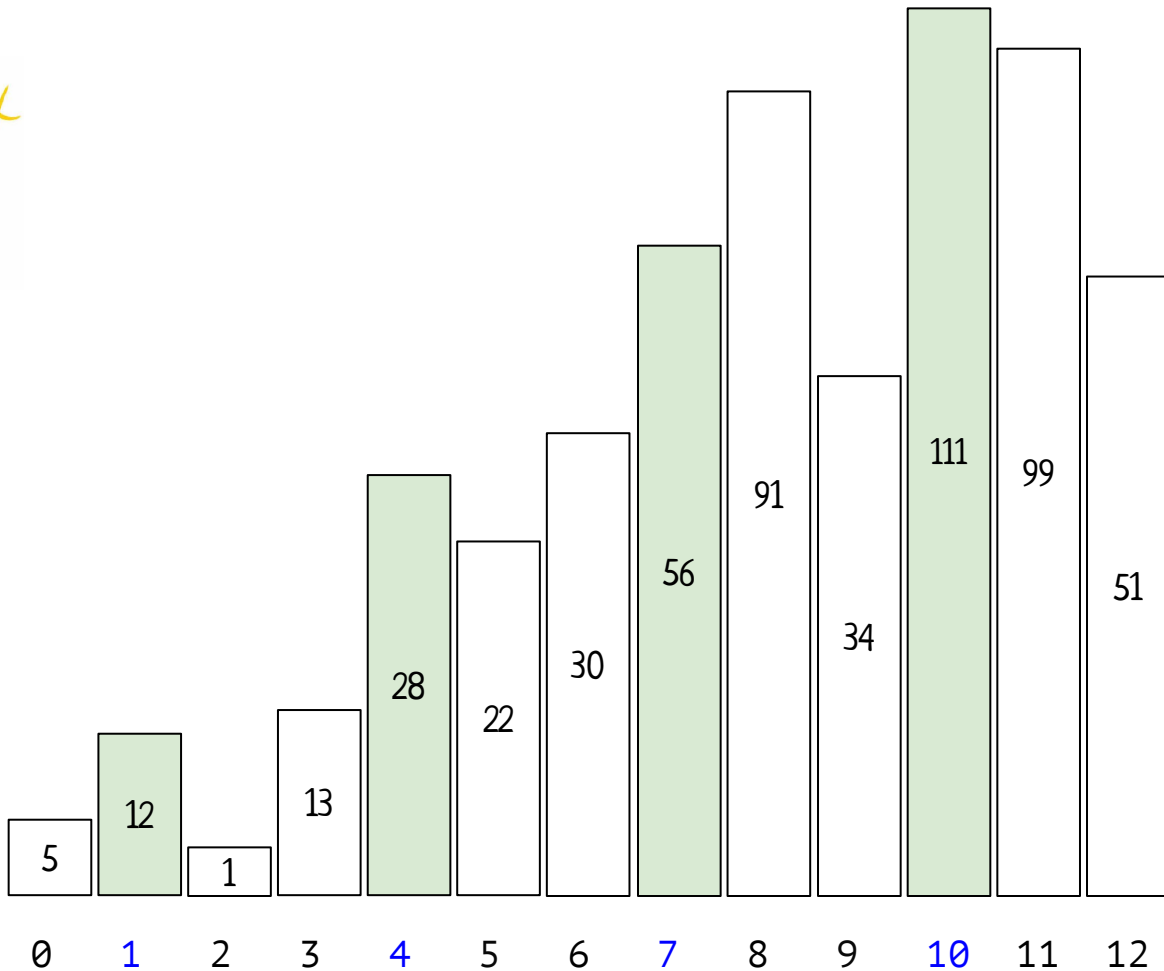
Стало ли лучше?

Что можете
сказать про
такой массив?



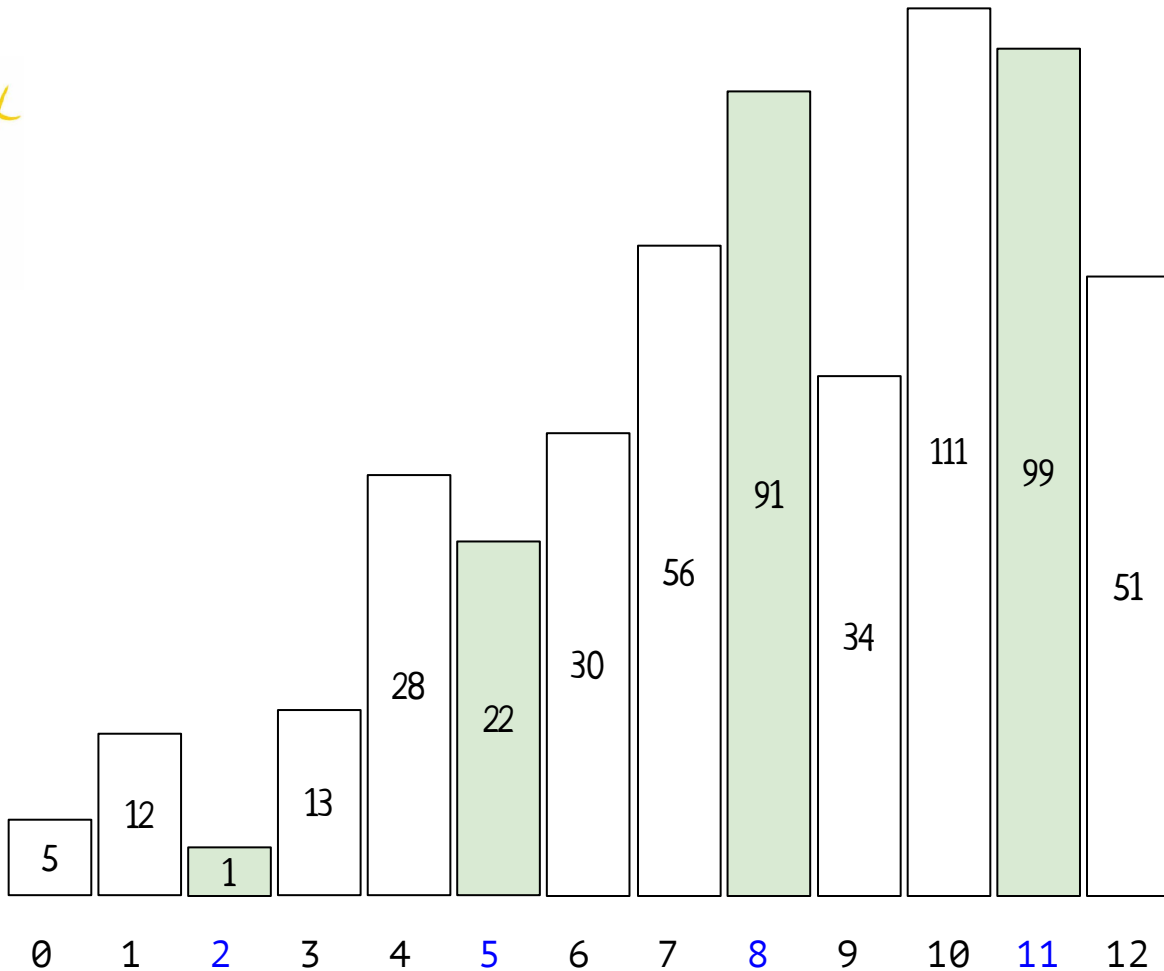
Стало ли лучше?

Что можете
сказать про
такой массив?



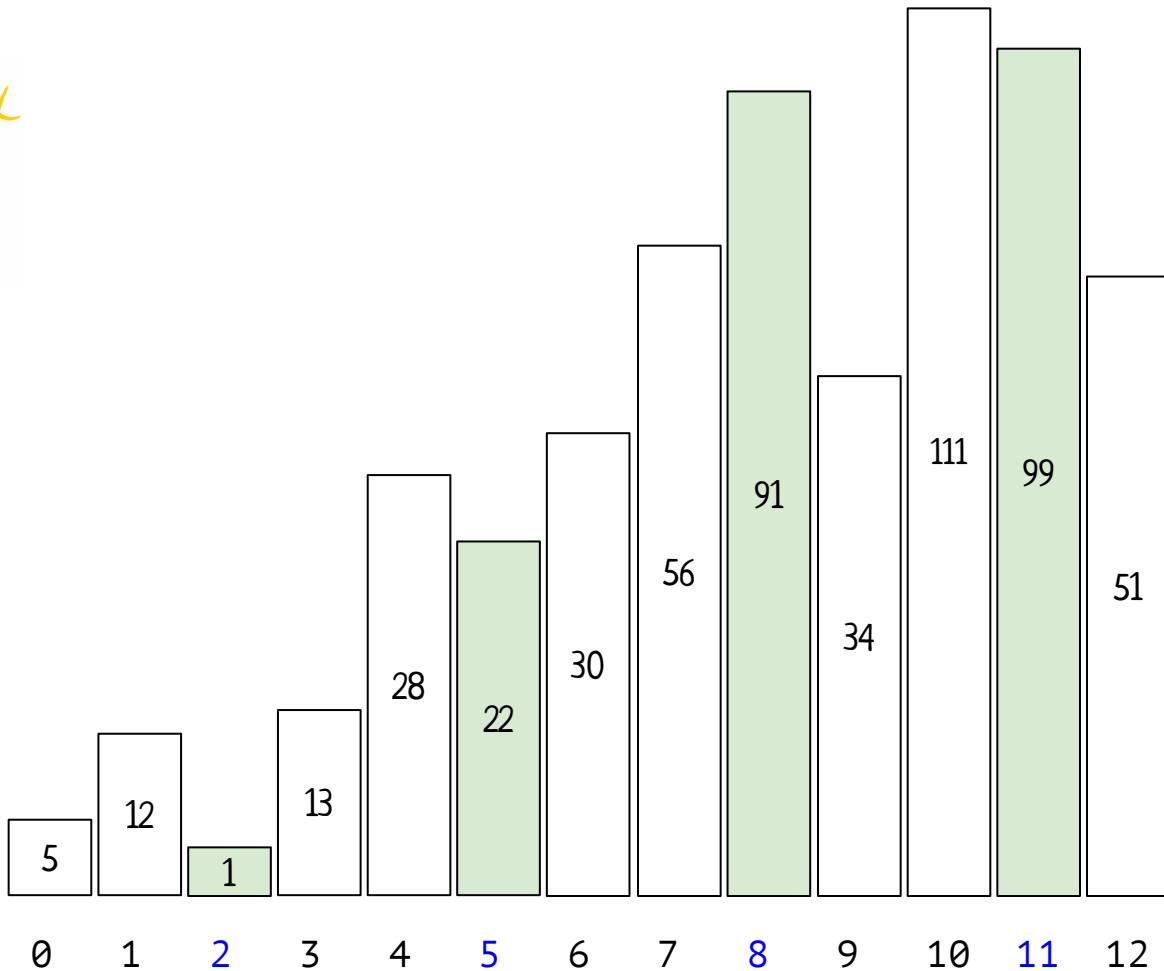
Стало ли лучше?

Что можете
сказать про
такой массив?



Стало ли лучше?

Что можете
сказать про
такой массив?

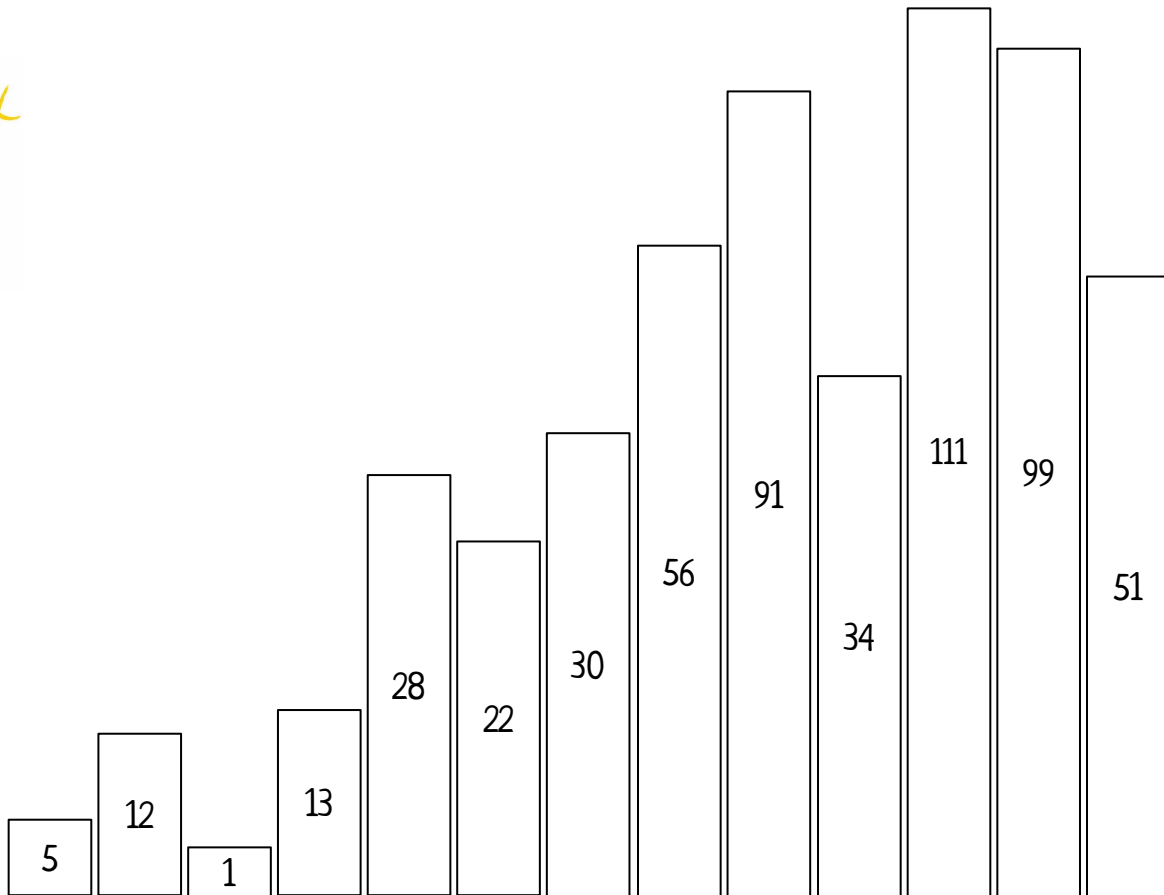


Стало ли лучше?

Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

(такой массив
называется
3-sorted)



Стало ли лучше?

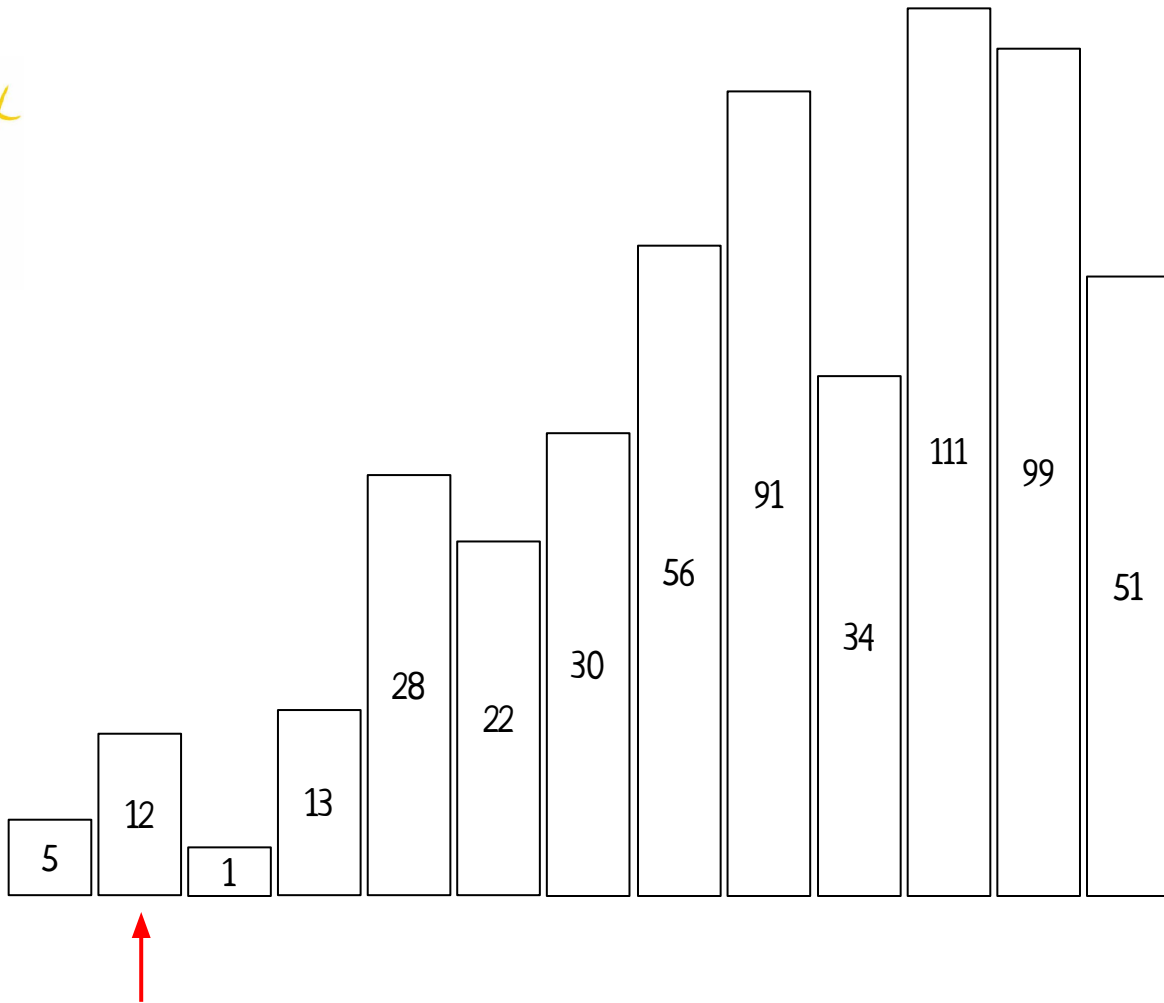
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-**
его элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 0



Стало ли лучше?

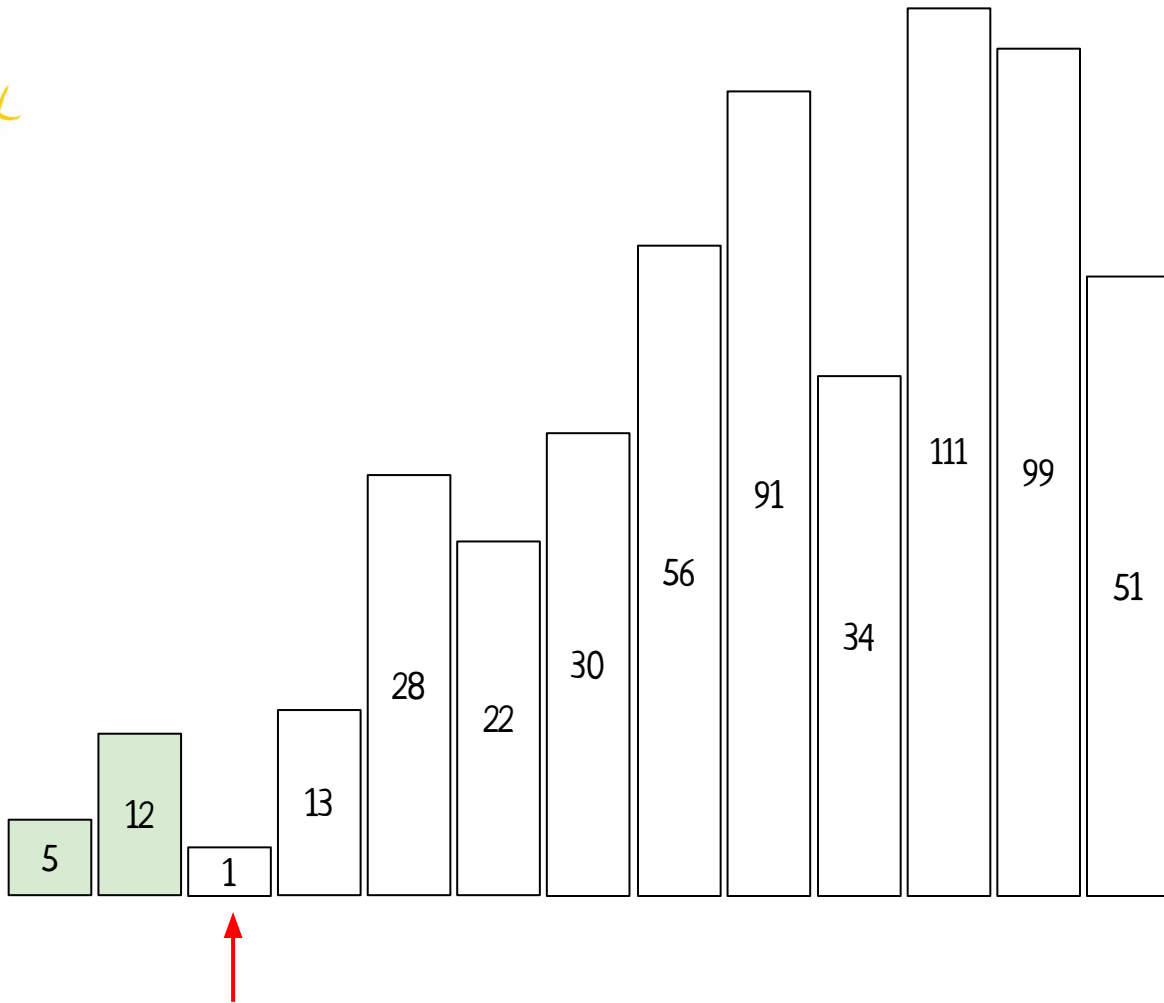
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 0



Стало ли лучше?

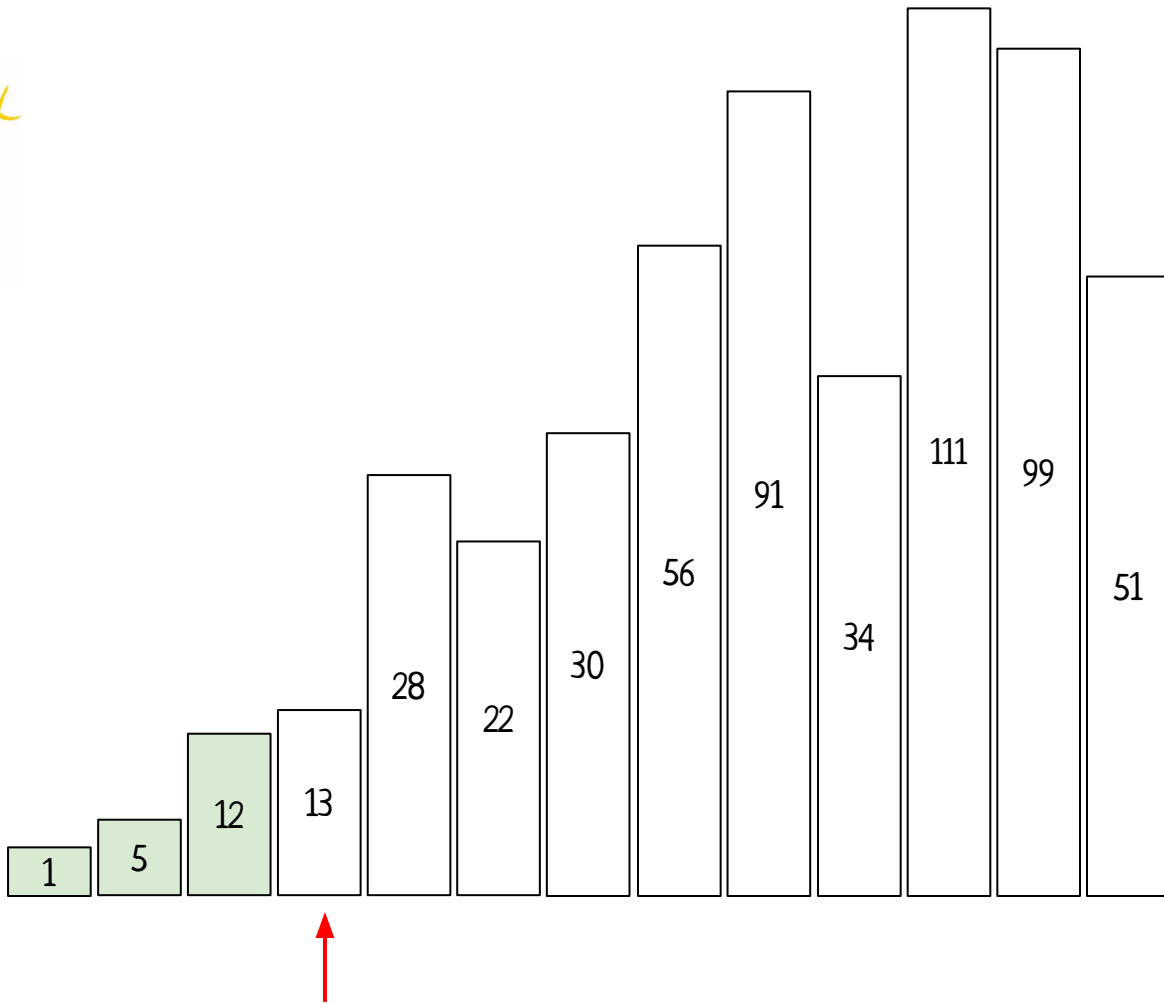
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 2



Стало ли лучше?

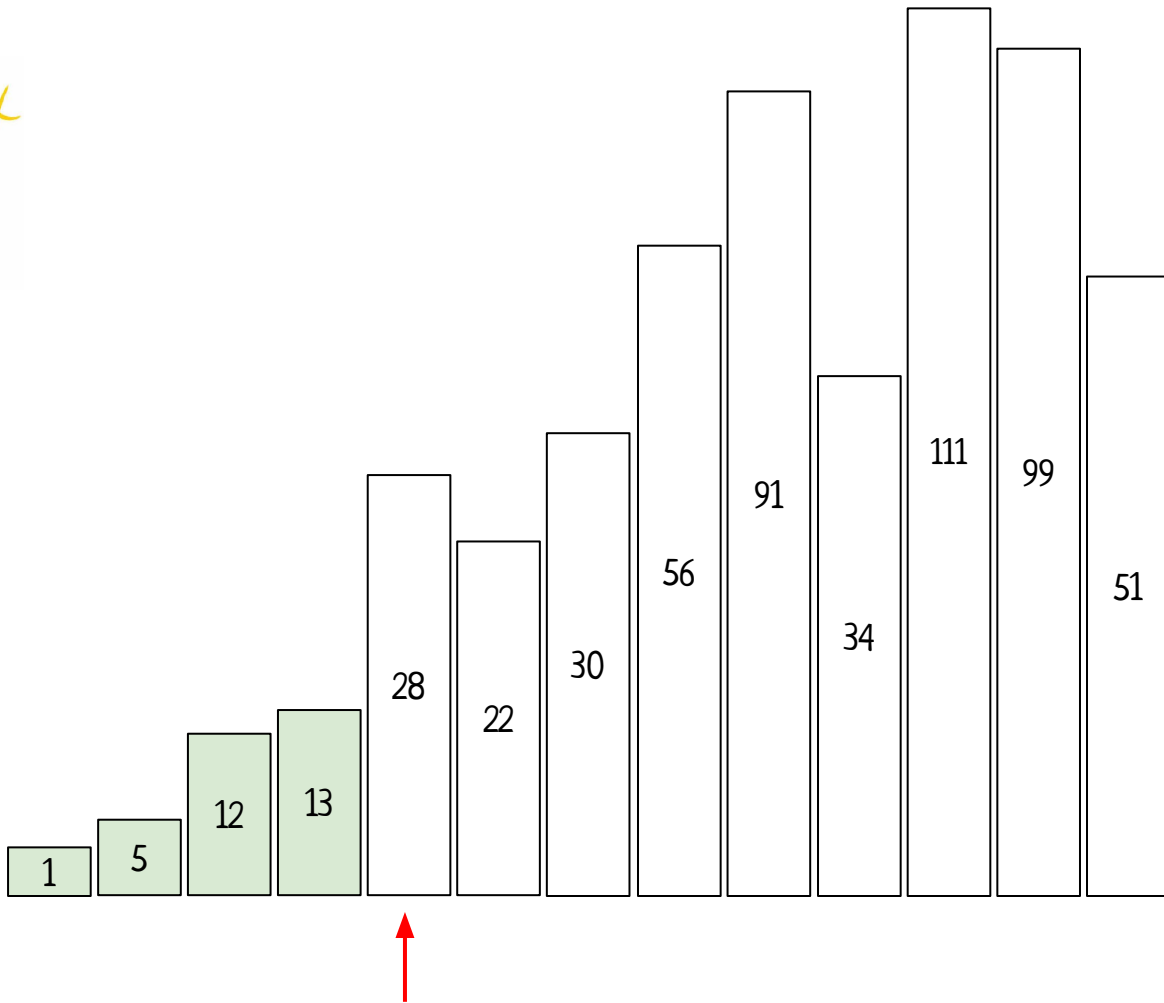
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 2



Стало ли лучше?

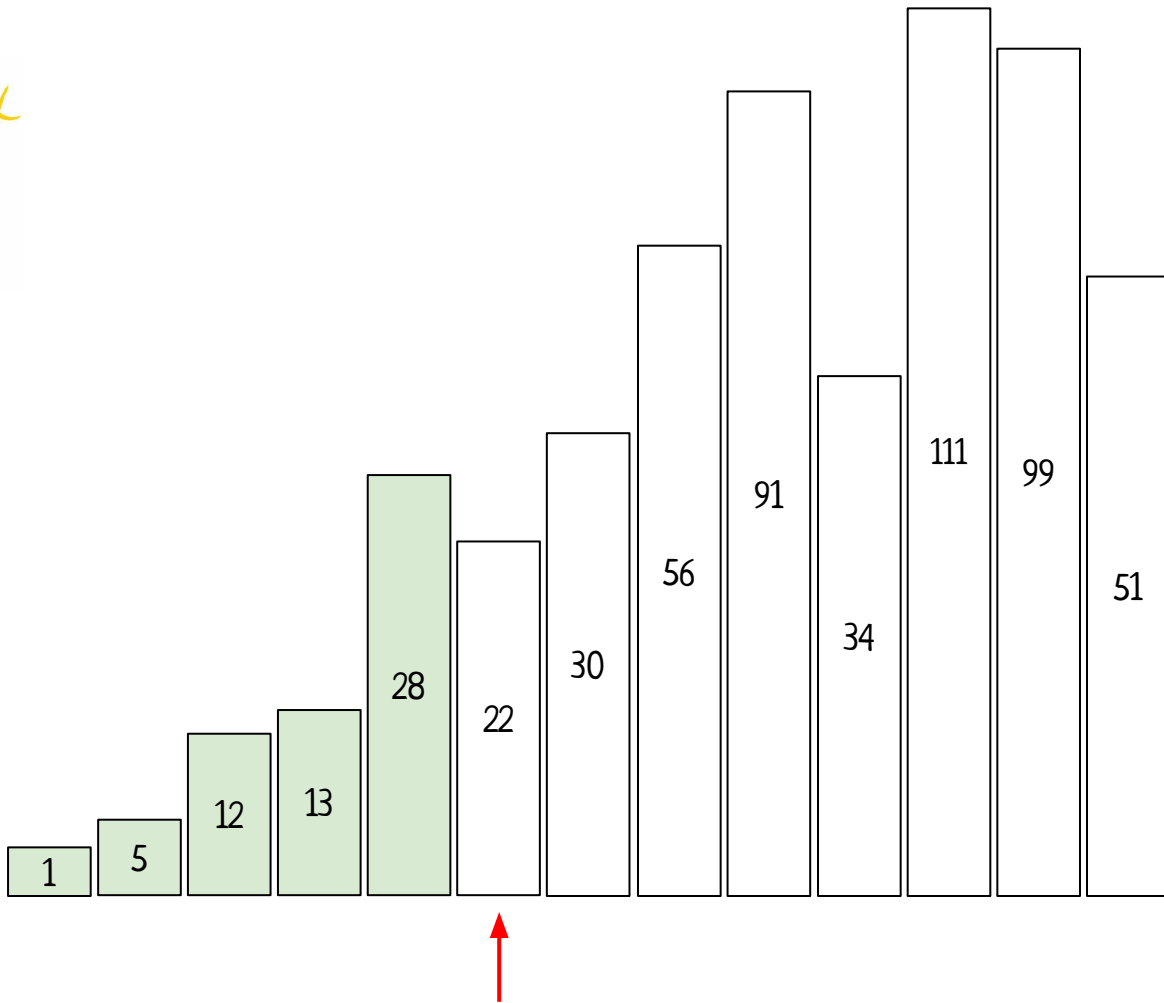
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 2



Стало ли лучше?

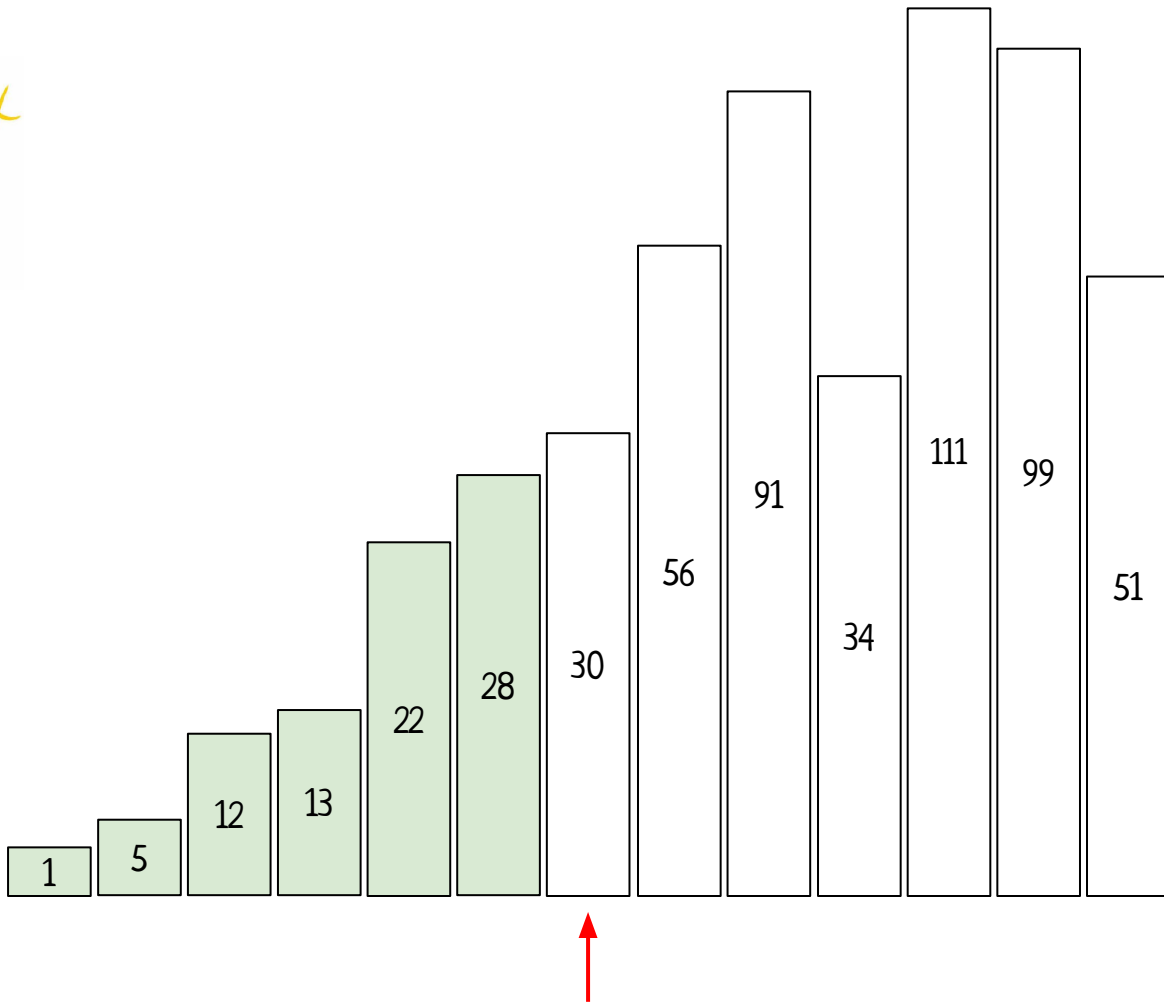
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 3



Стало ли лучше?

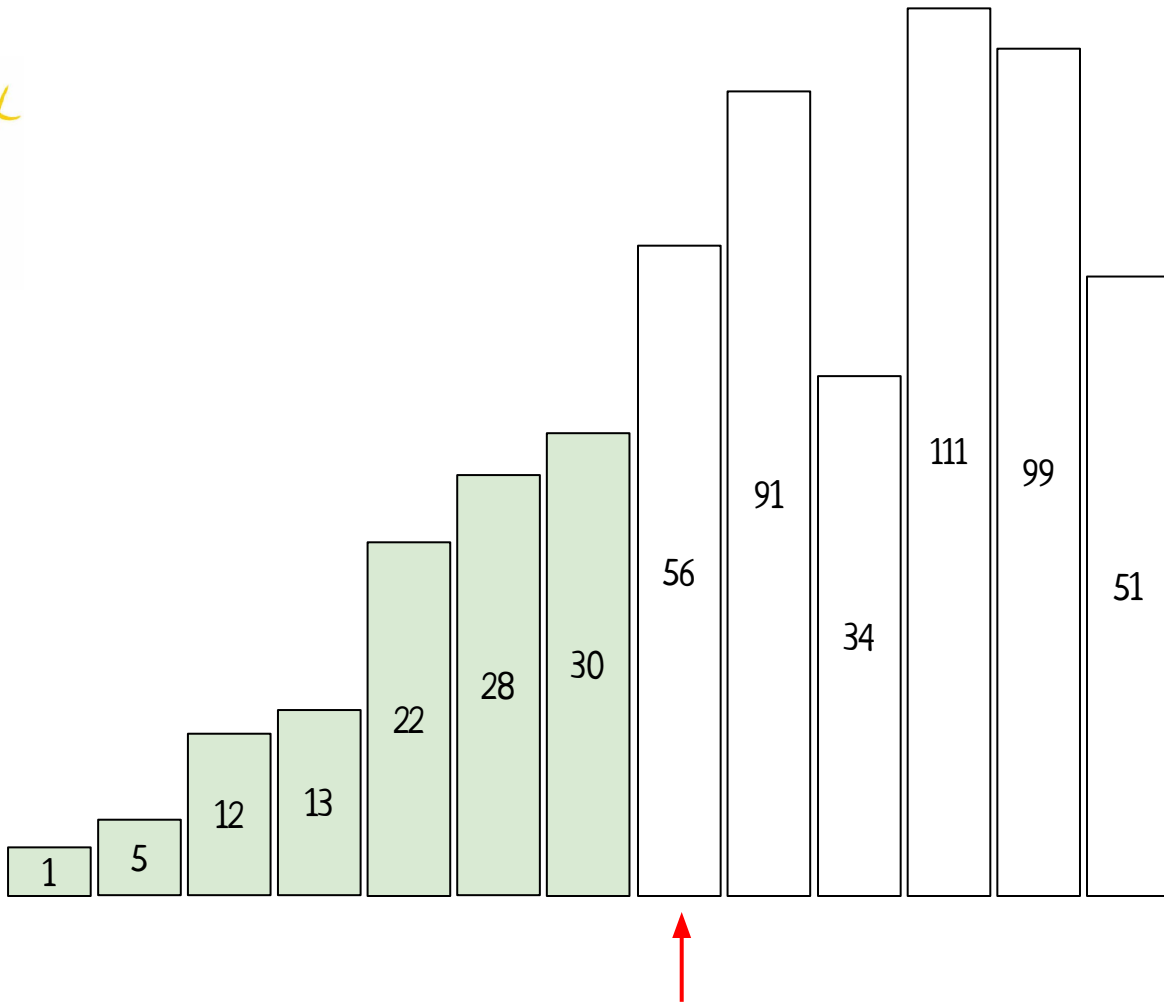
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 3



Стало ли лучше?

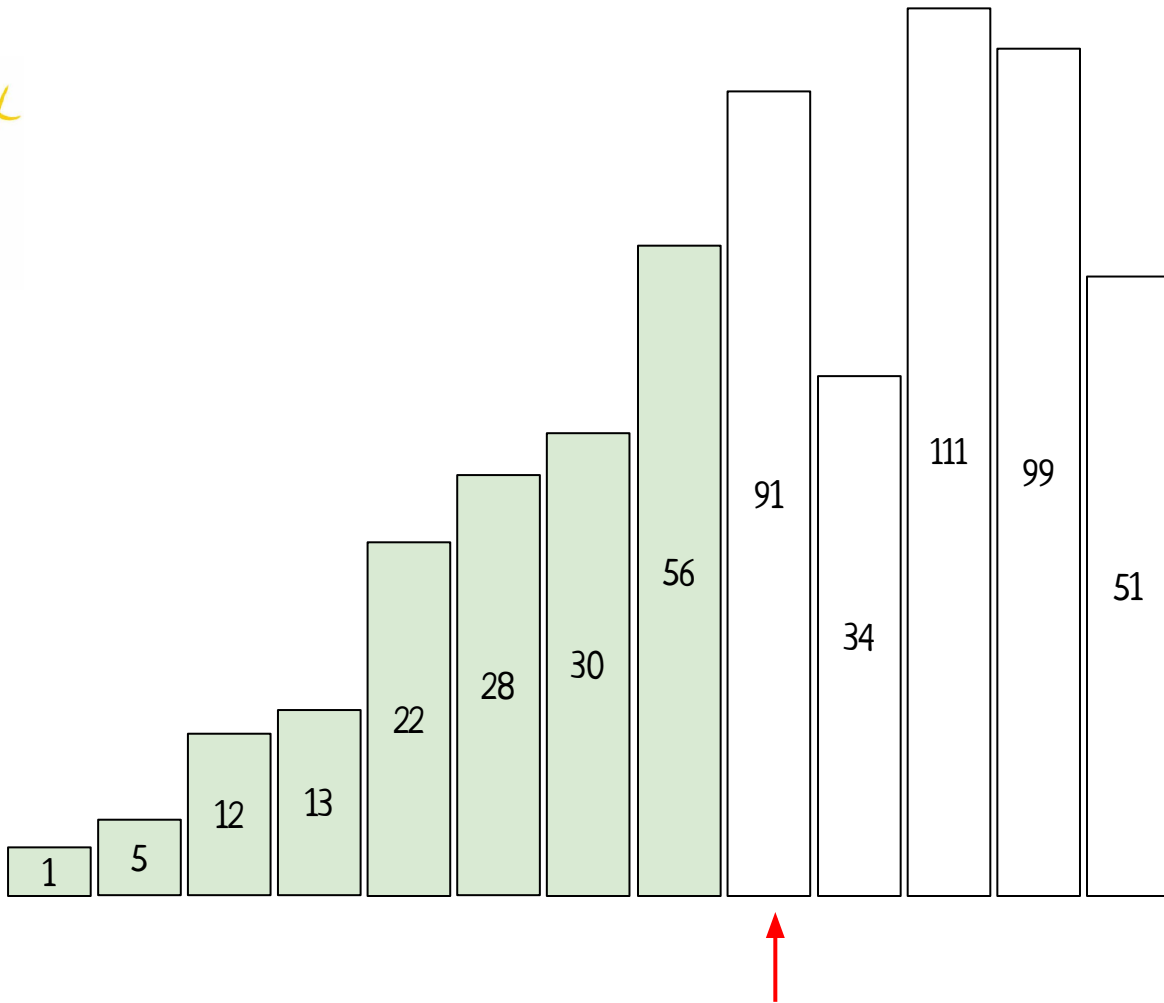
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 3



Стало ли лучше?

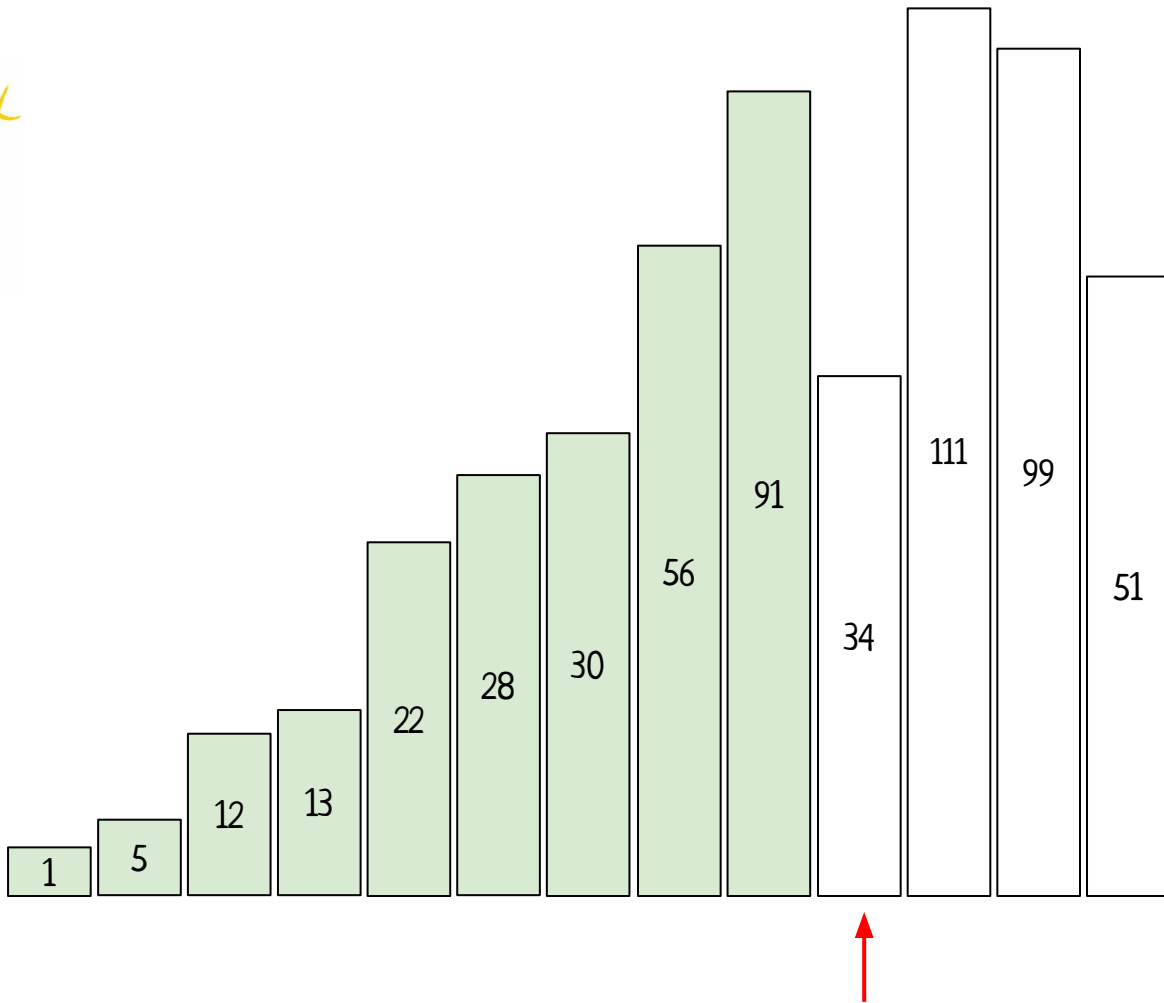
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 3



Стало ли лучше?

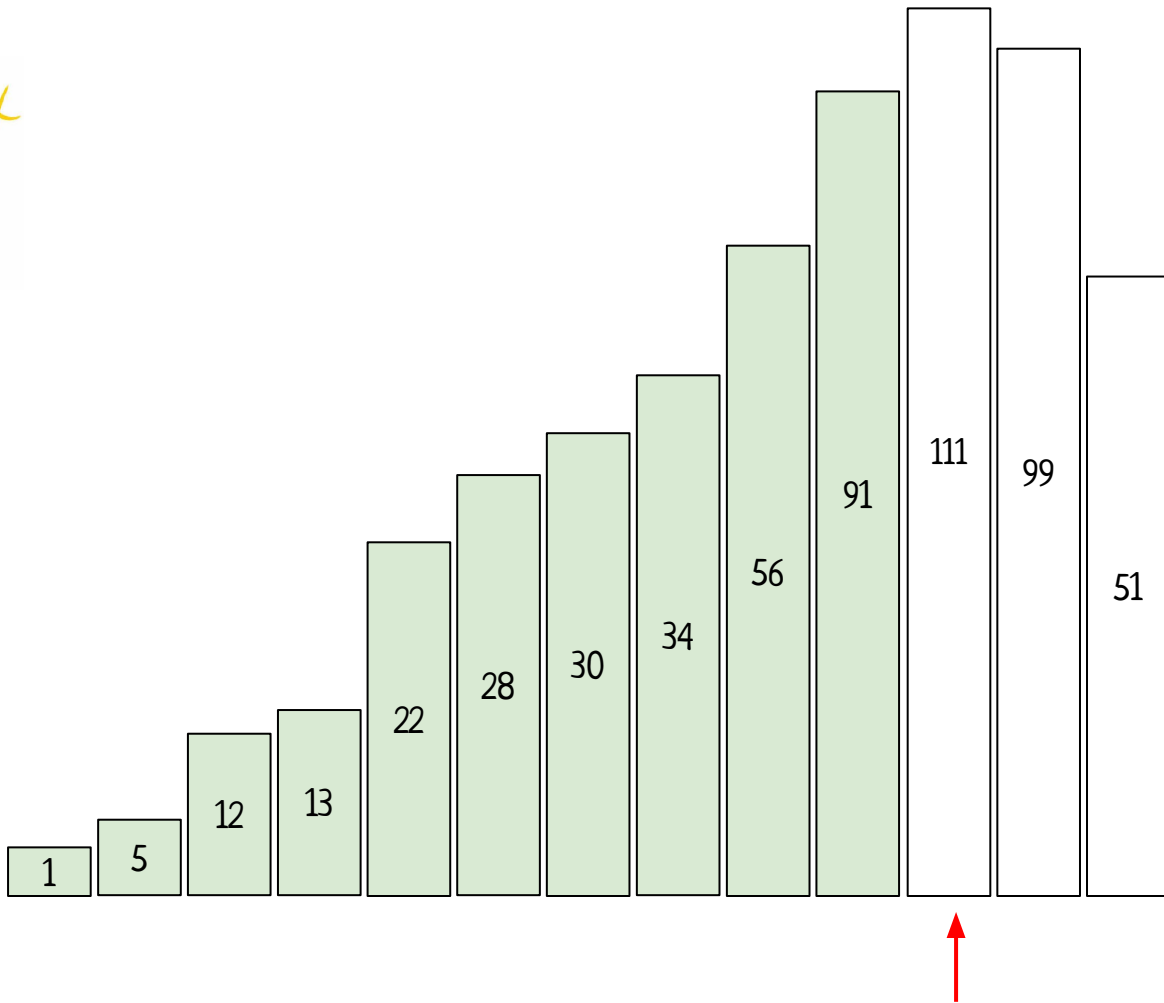
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 5



Стало ли лучше?

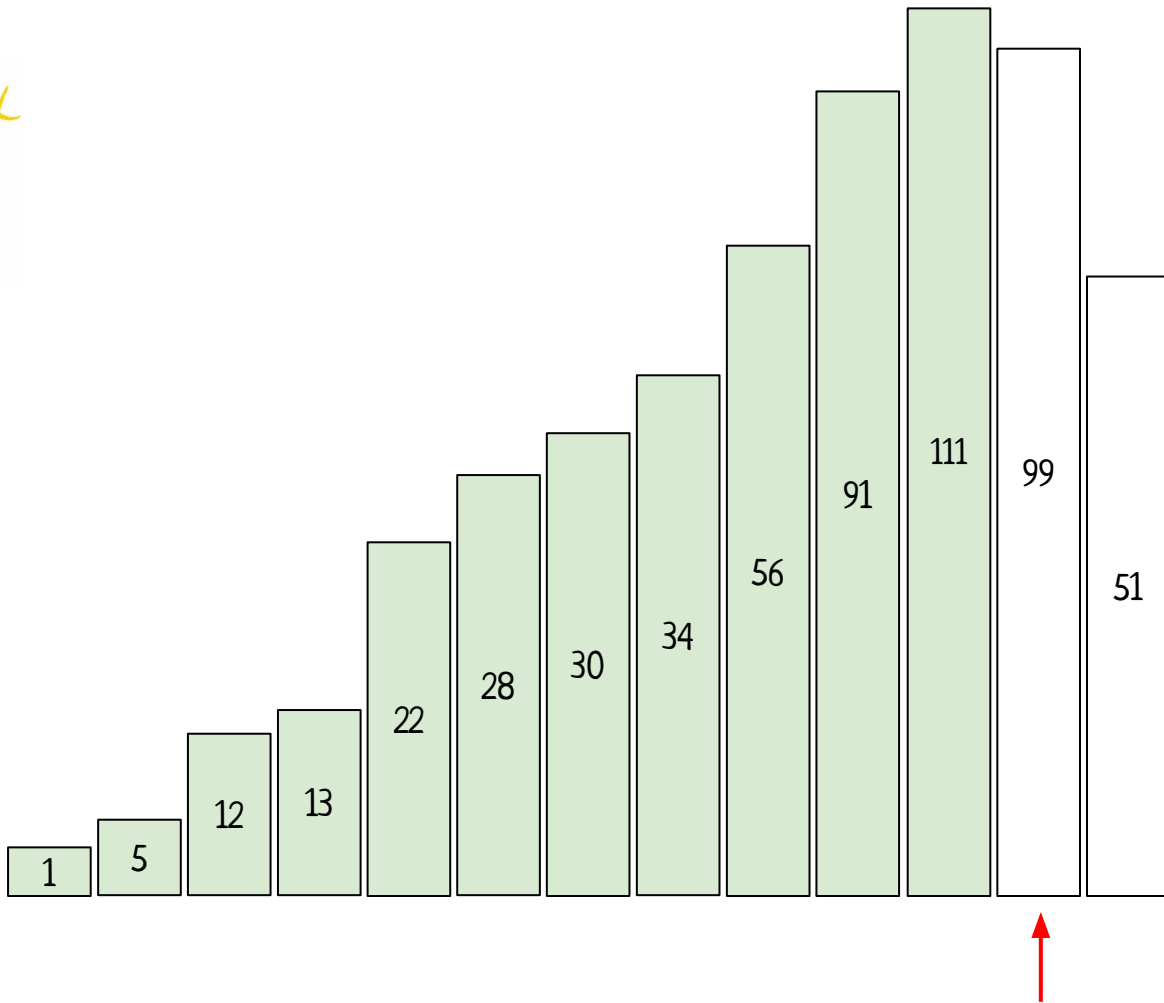
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 5



Стало ли лучше?

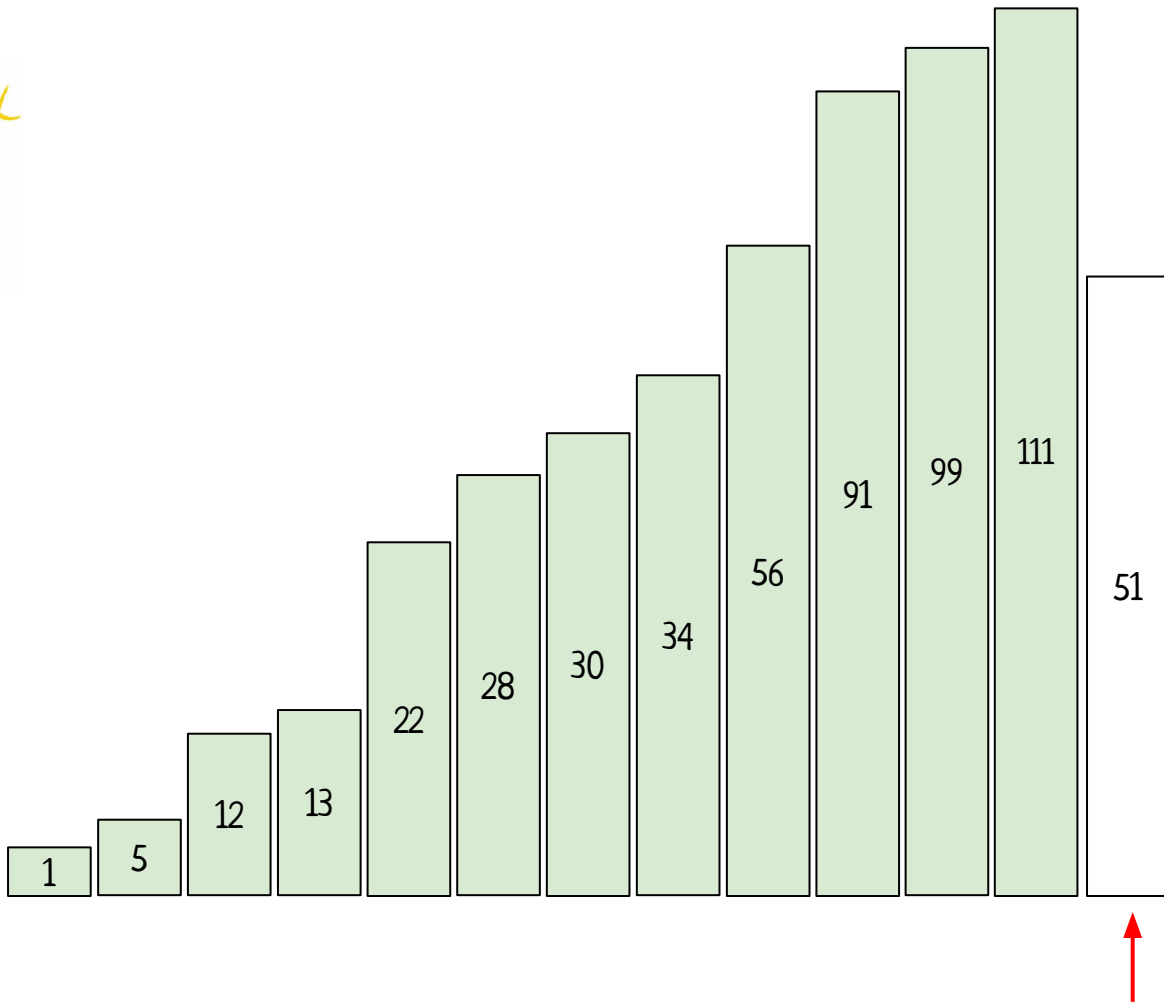
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 6



Стало ли лучше?

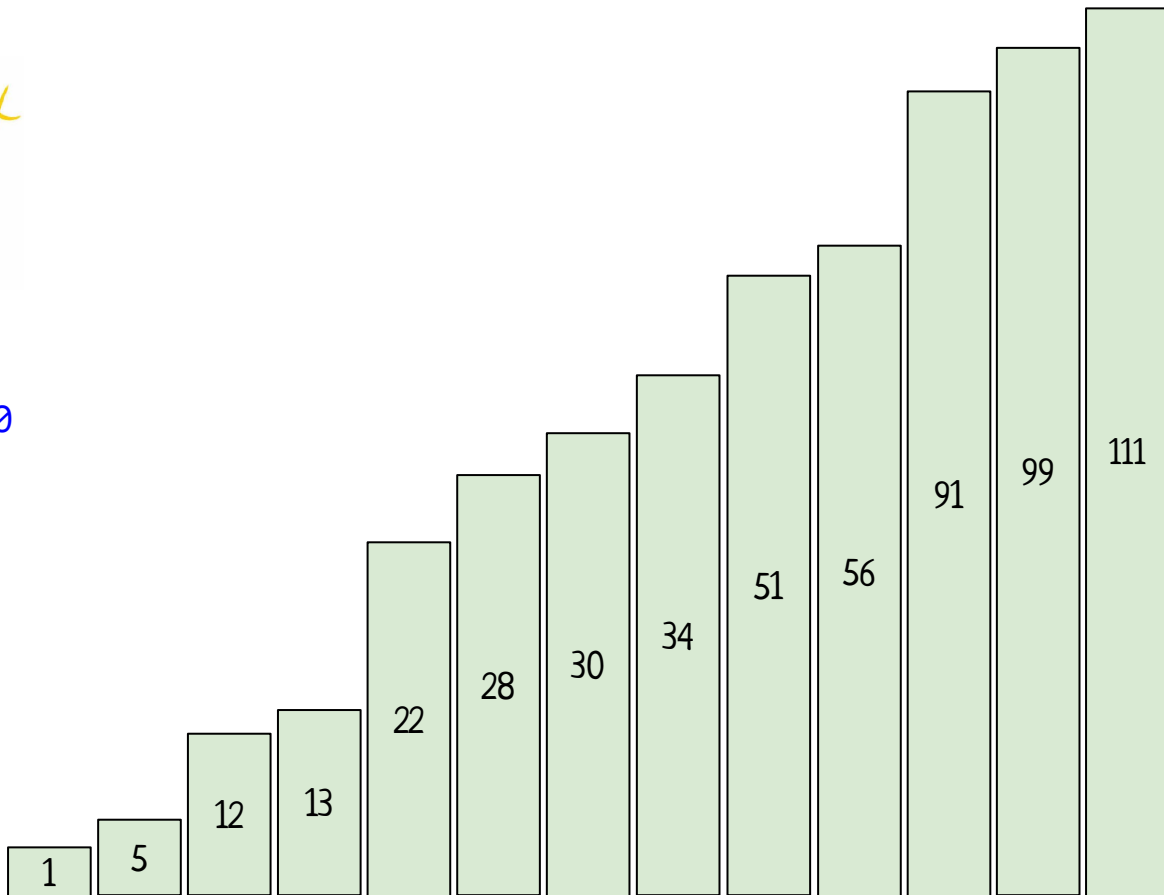
Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 10



Стало ли лучше?

Что можете
сказать про
такой массив?

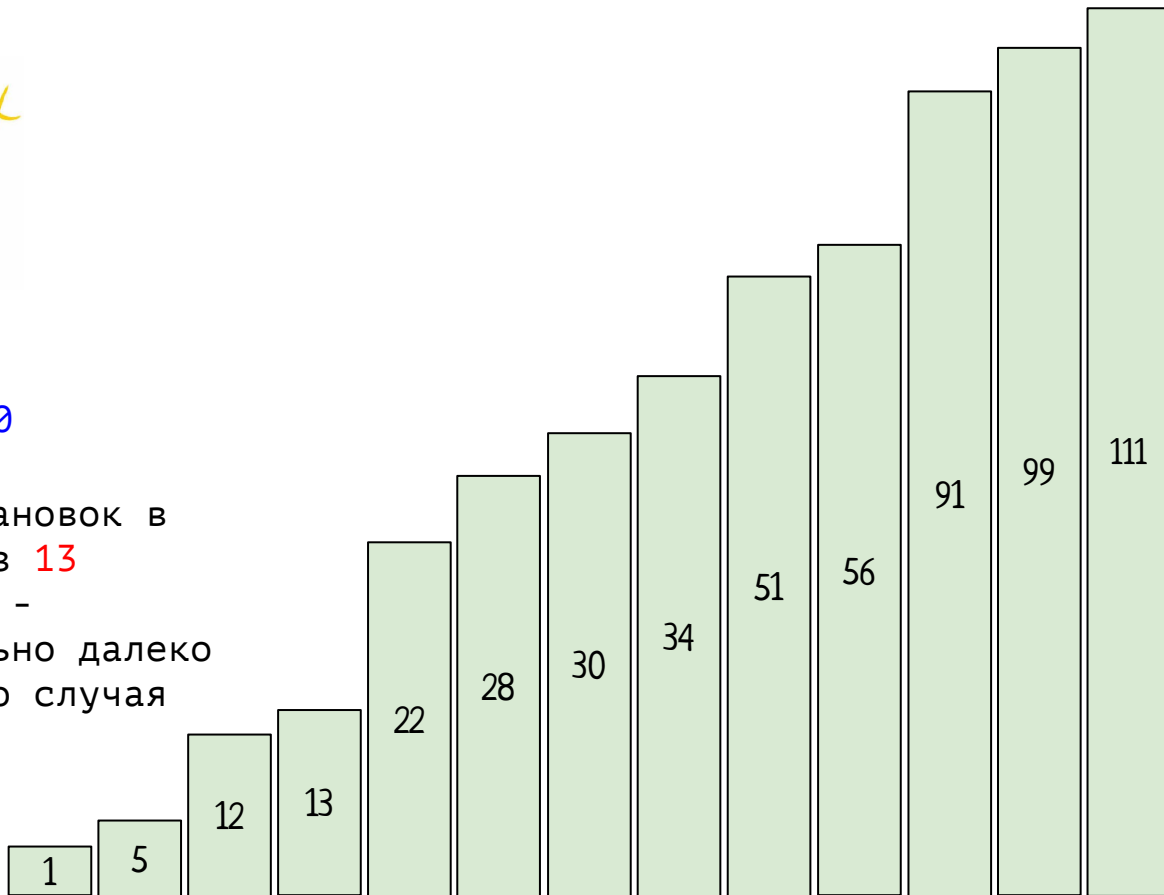
Все подпоследовательности
из каждого **3-его** элемента
упорядочены!

А теперь
запускаем
**сортировку
вставками.**



счетчик
swap-ов: 10

10 перестановок в
массиве из 13
элементов -
это довольно далеко
от худшего случая



Стало ли лучше?

Что можете
сказать про
такой массив?

Все подпоследовательности
из каждого 3-
его элемента
упорядочены!

А теперь
запускаем
сортировку
вставками.

Сортировка Шелла

1. Пусть есть некоторая убывающая последовательность целых чисел:
 k_1, k_2, \dots, k_p ; где $k_1 < N; k_p = 1$.

Сортировка Шелла

1. Пусть есть некоторая убывающая последовательность целых чисел:
 k_1, k_2, \dots, k_p ; где $k_1 < N; k_p = 1$.
2. Последовательно k_i -сортируем массив

Сортировка Шелла

1. Пусть есть некоторая убывающая последовательность целых чисел:
 k_1, k_2, \dots, k_p ; где $k_1 < N; k_p = 1$.
2. Последовательно k_i -сортируем массив
3. На последнем шаге ($k_p = 1$) запускаем сортировку **вставками**.

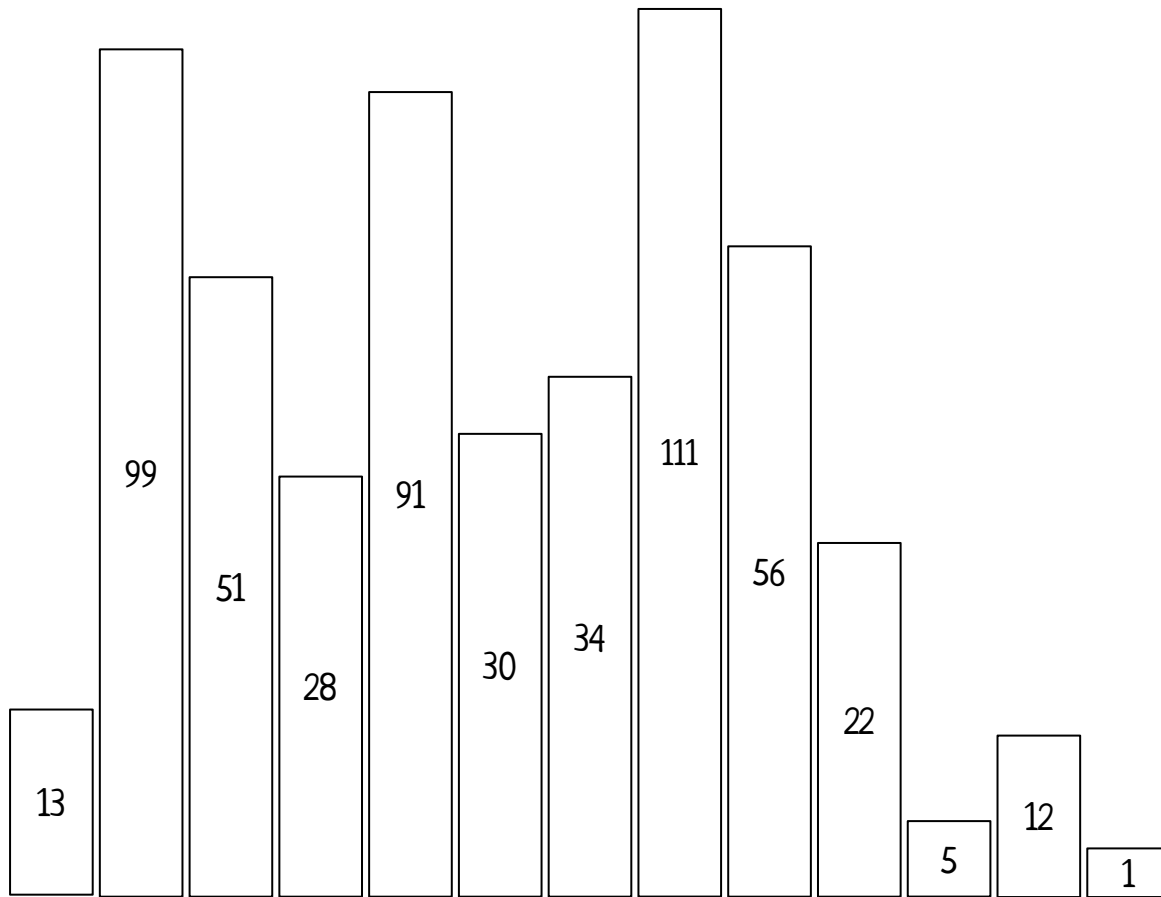
Пример

Пусть $k_1 = 7; k_2 = 3; k_3 = 1$

$$k_1 = 7;$$

$$k_2 = 3;$$

$$k_3 = 1$$

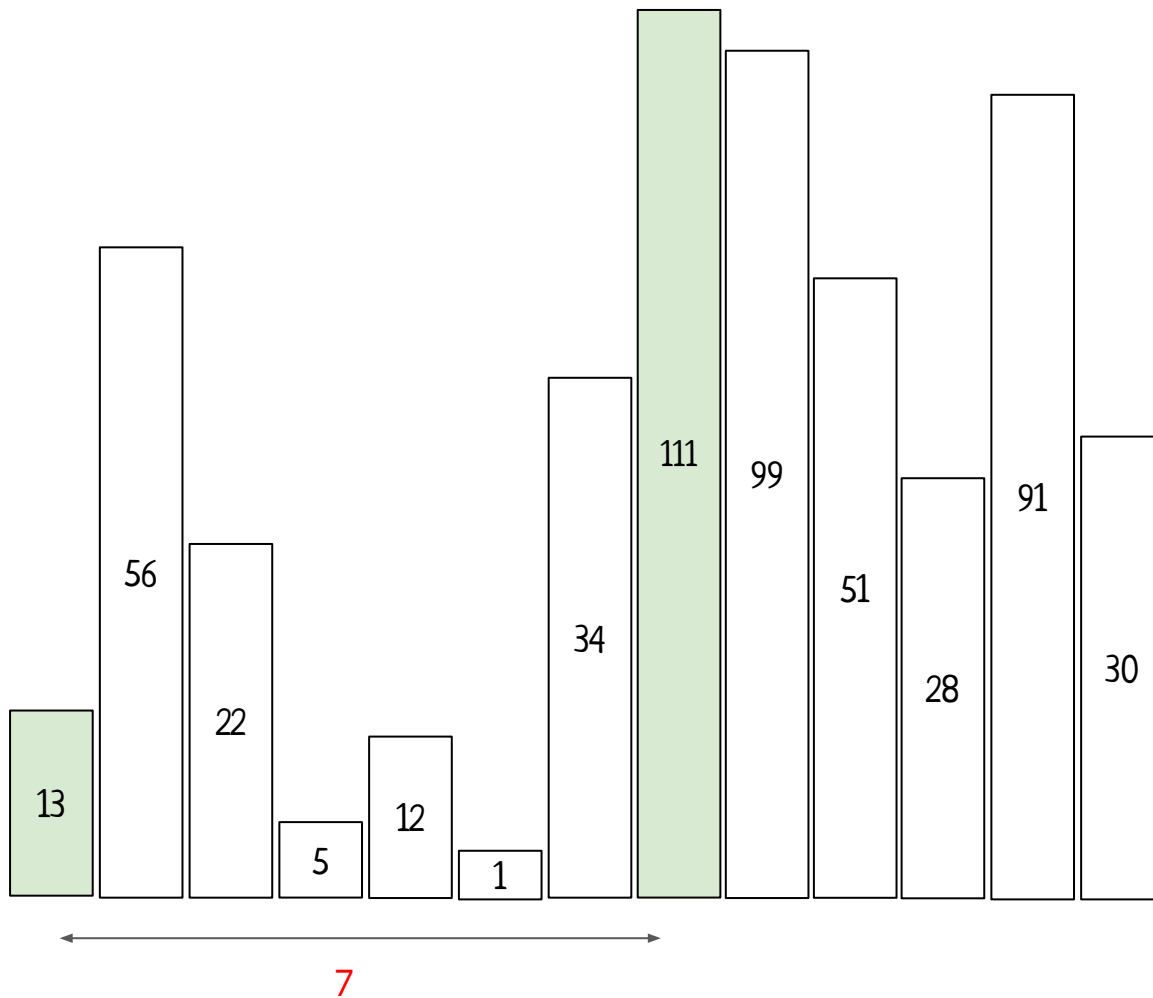


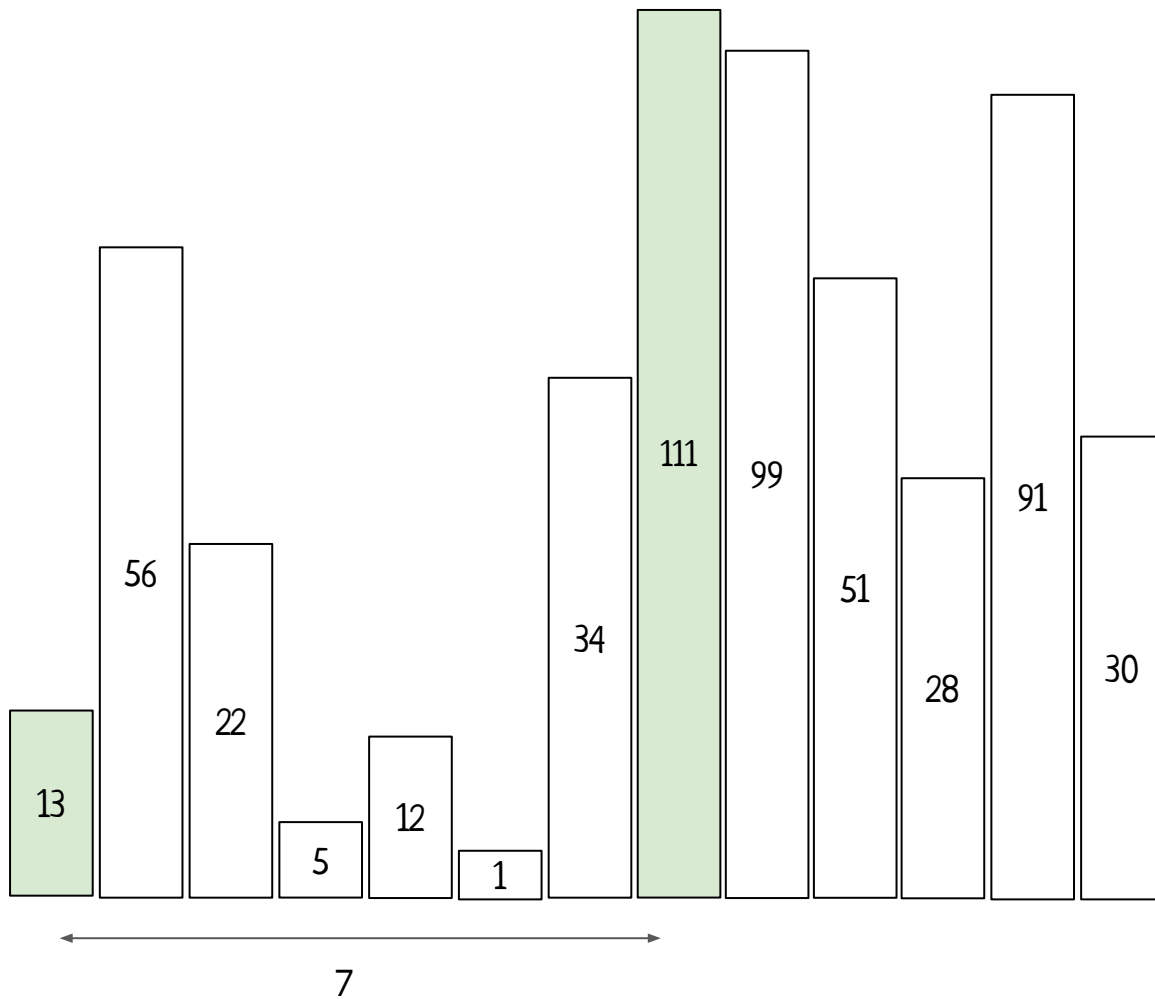
$$k_1 = 7;$$

$$k_2 = 3;$$

$$k_3 = 1$$

1) превращаем в
7-sorted





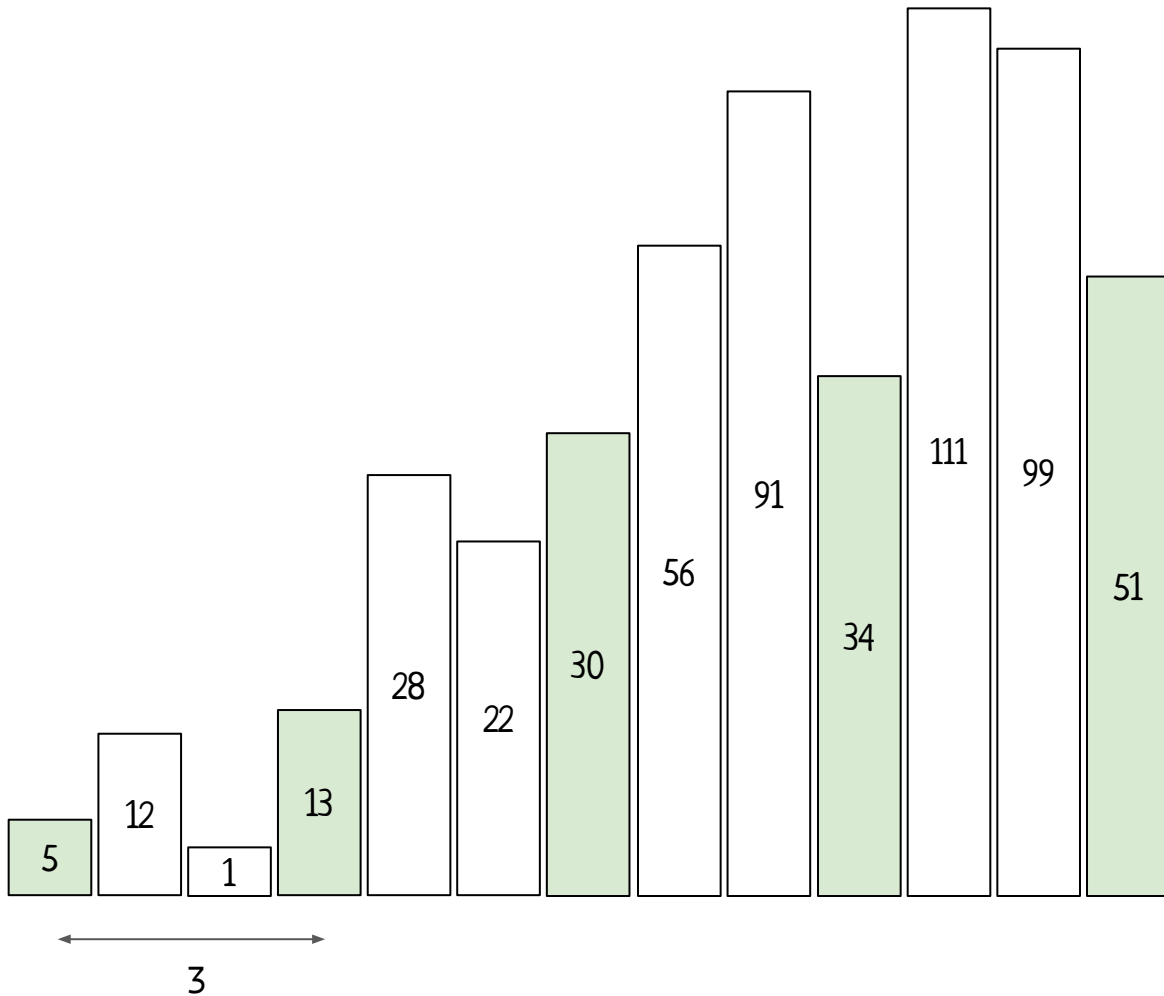
$$k_1 = 7;$$

$$k_2 = 3;$$

$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили 5
swap-ов



$$k_1 = 7;$$

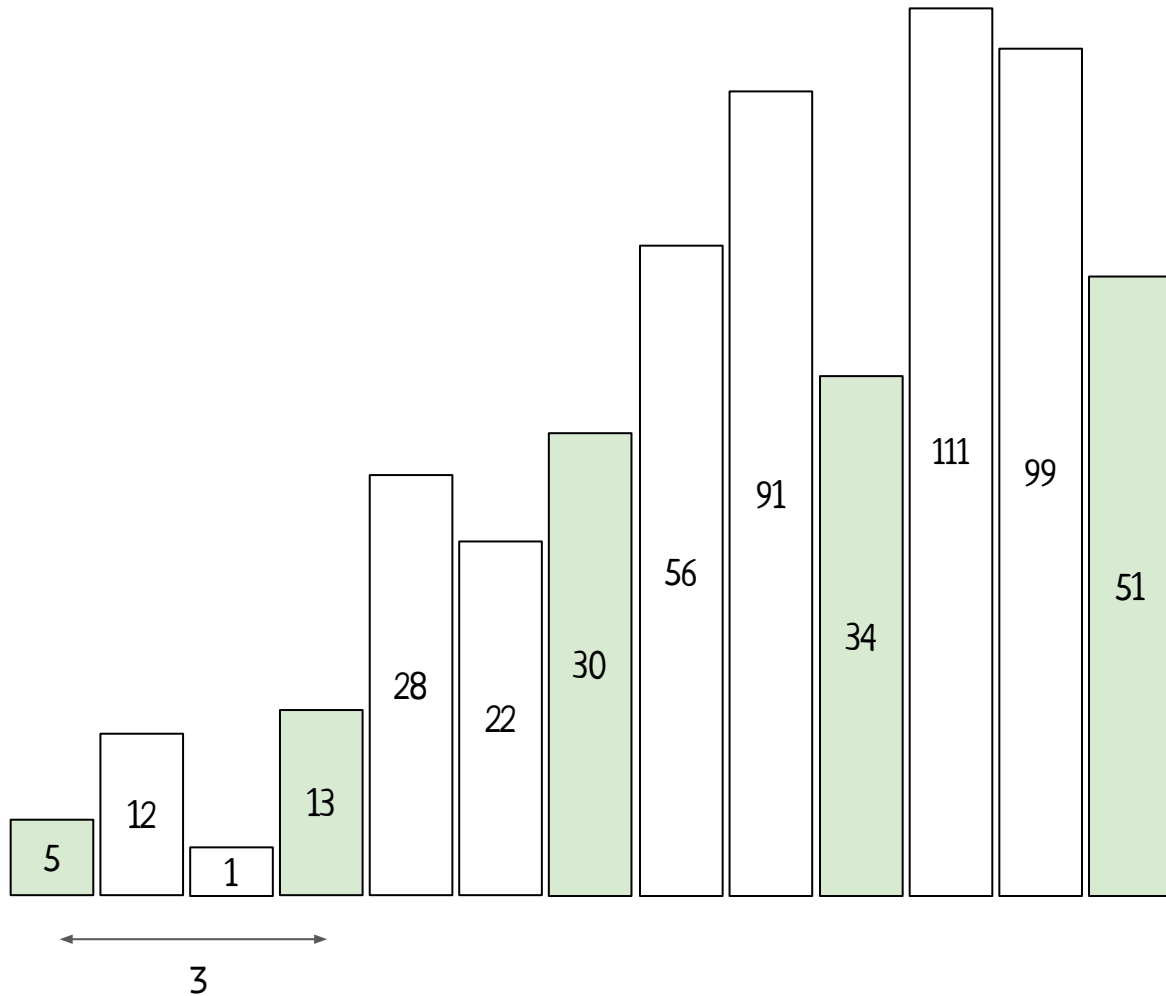
$$k_2 = 3;$$

$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили **5**
swap-ов

2) превращаем в
3-sorted



$$k_1 = 7;$$

$$k_2 = 3;$$

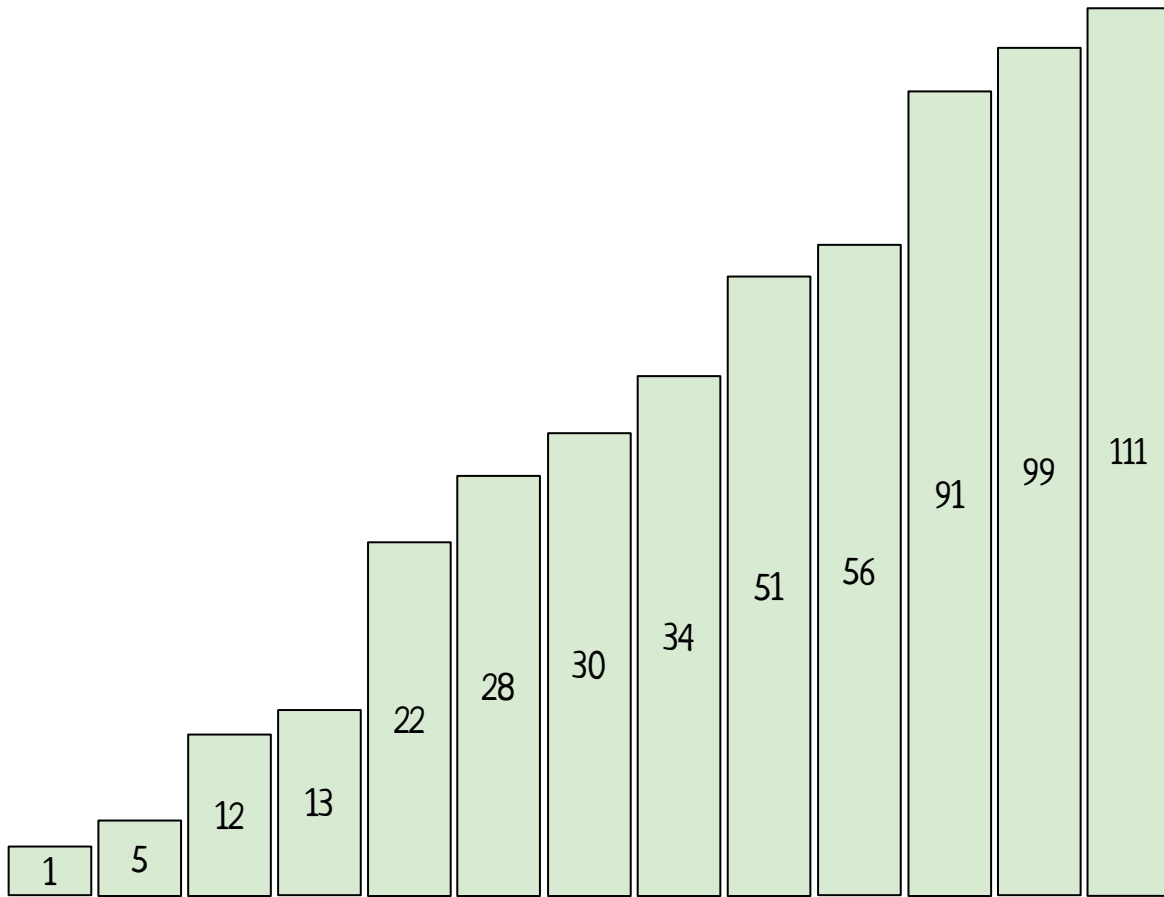
$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили 5
swap-ов

2) превращаем в
3-sorted

Потратили еще
8 swap-ов



$$k_1 = 7;$$

$$k_2 = 3;$$

$$k_3 = 1$$

1) превращаем в
7-sorted

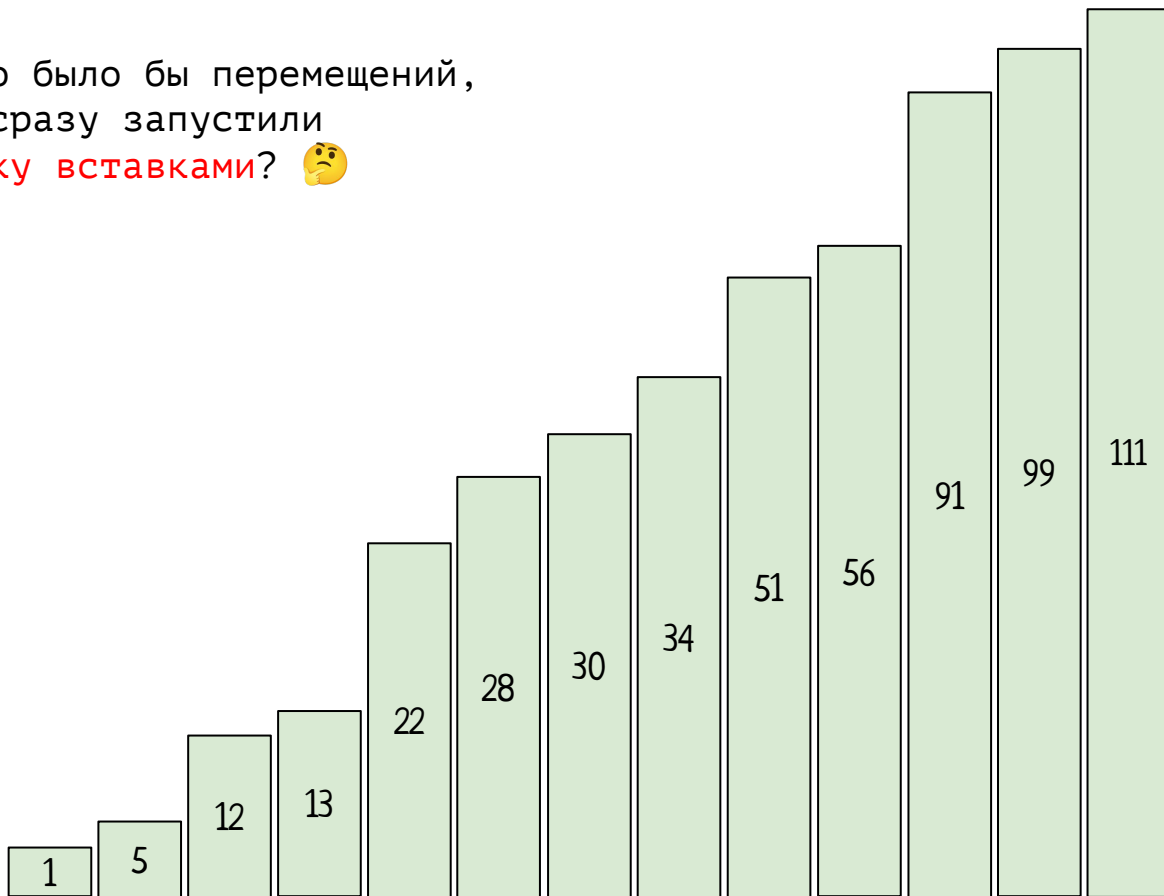
Потратили 5
swap-ов

2) превращаем в
3-sorted

Потратили еще
8 swap-ов

3) сортировка
вставками
(10 swap)

А сколько было бы перемещений,
если бы сразу запустили
сортировку вставками? 🤔



$$k_1 = 7;$$
$$k_2 = 3;$$
$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили **5**
swap-ов

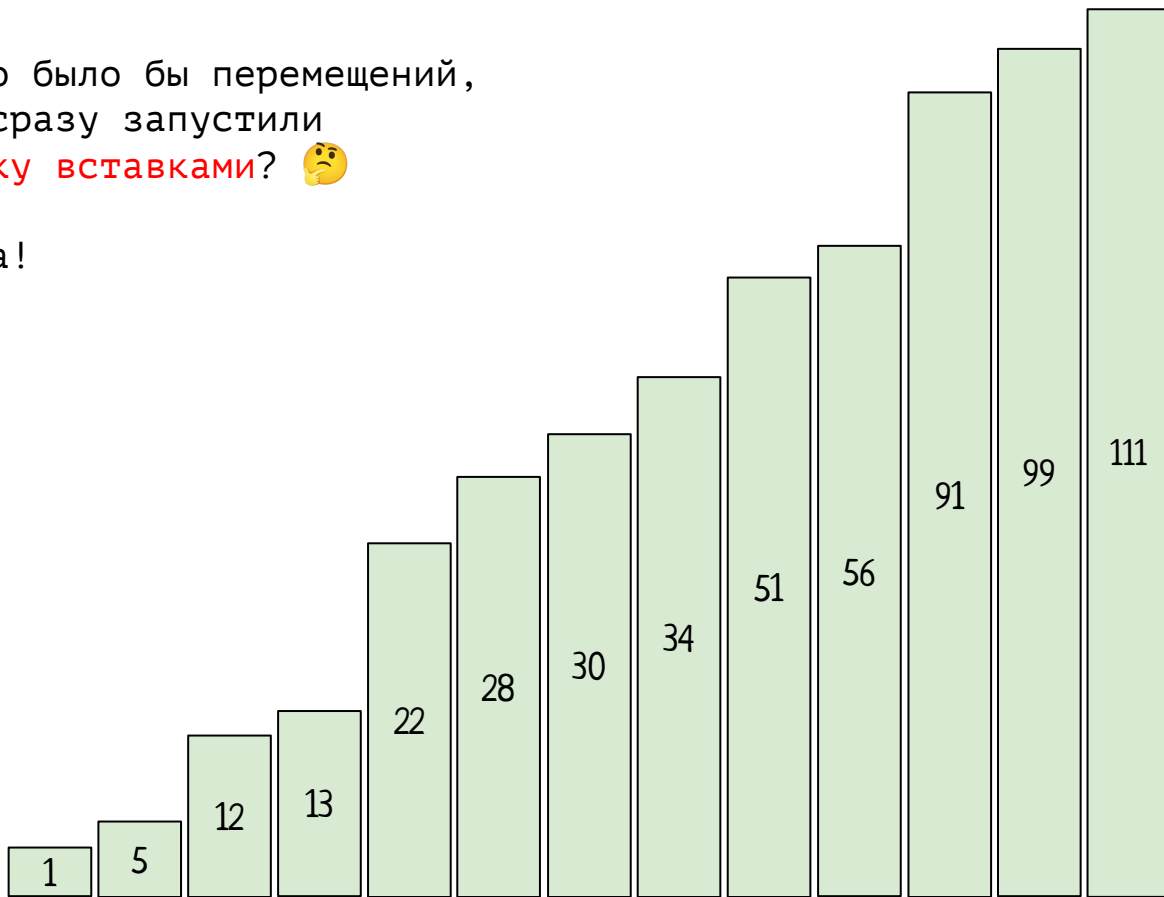
2) превращаем в
3-sorted

Потратили еще
8 swap-ов

3) сортировка
вставками
(**10** swap)

А сколько было бы перемещений,
если бы сразу запустили
сортировку вставками? 🤔

53 swap-a!



$$k_1 = 7;$$
$$k_2 = 3;$$
$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили 5
swap-ов

2) превращаем в
3-sorted

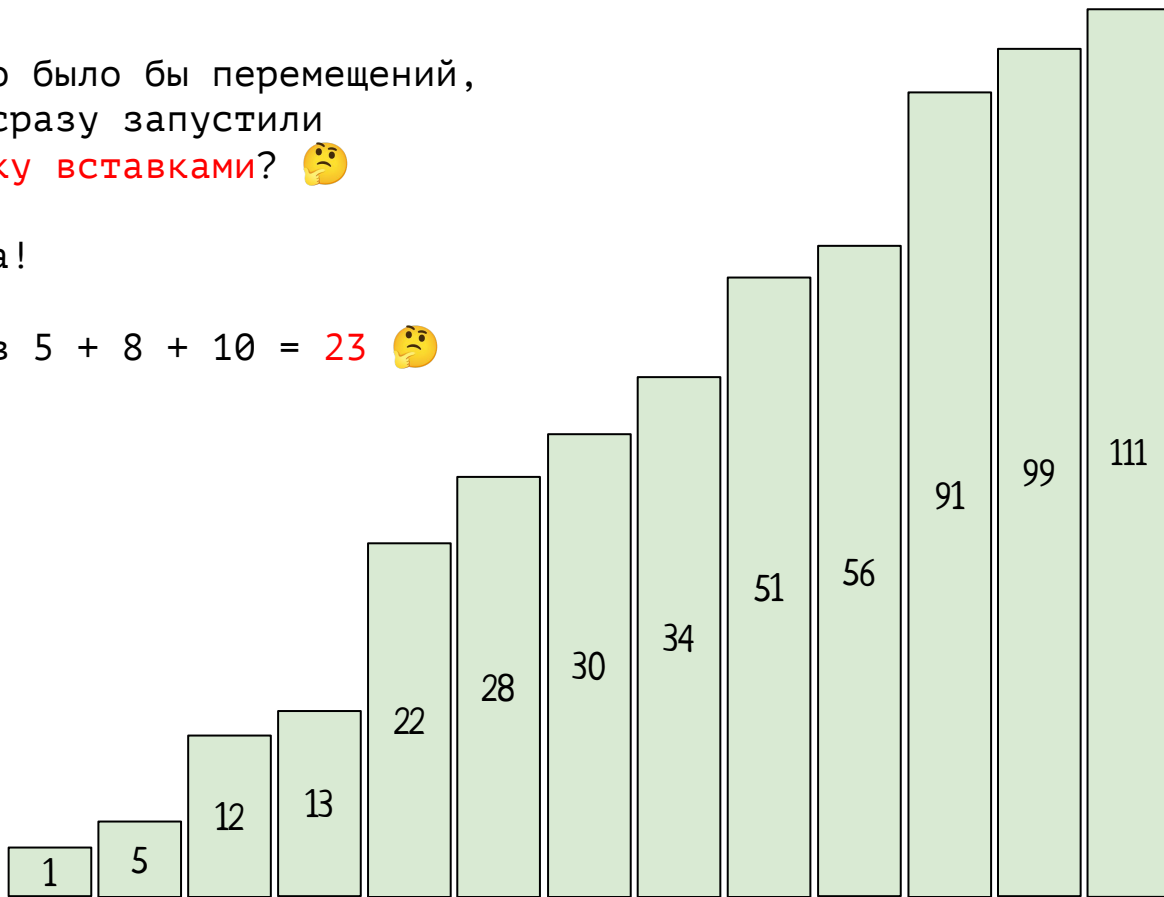
Потратили еще
8 swap-ов

3) сортировка
вставками
(10 swap)

А сколько было бы перемещений,
если бы сразу запустили
сортировку вставками? 🤔

53 swap-a!

53 против $5 + 8 + 10 = 23$ 🤔



$$k_1 = 7;$$

$$k_2 = 3;$$

$$k_3 = 1$$

1) превращаем в
7-sorted

Потратили 5
swap-ов

2) превращаем в
3-sorted

Потратили еще
8 swap-ов

3) сортировка
вставками
(10 swap)

Сортировка Шелла

Вопросы:

1. Как превратить массив в k -sorted?

Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Сортировка вставками

```
def insertion_sort_k(array: int[], k: int):  
    for i in [k, len(array)):  
        j = i  
        while j - k >= 0 and  
            array[j - k] > array[j]:  
            swap(array, j - k, j)  
            j -= k
```

Сортировка Шелла

Вопросы:

1. Как превратить массив в k -sorted?

Упражнение: доказать, что если $k > p$, то после приведения k -sorted массива к p -sorted виду, он остается k -sorted.

Сортировка Шелла

Вопросы:

1. Как превратить массив в k -sorted?
2. Какие последовательности чисел брать?
3. Сложность алгоритма?

Сортировка Шелла

Вопросы:

1. Как превратить массив в k -sorted?
2. Какие последовательности чисел брать?
3. Сложность алгоритма?

Есть варианты

Сортировка Шелла

Варианты последовательностей:

Сортировка Шелла

Варианты последовательностей:

$$1. \lfloor \frac{N}{2} \rfloor, \lfloor \frac{N}{4} \rfloor, \dots, 2, 1$$

by Donald Shell, 1959

Сортировка Шелла

Варианты последовательностей:

1. $\lfloor \frac{N}{2} \rfloor, \lfloor \frac{N}{4} \rfloor, \dots, 2, 1$

by Donald Shell, 1959

Сложность: $\Theta(N^2)$



Сортировка Шелла

Варианты (обратных) последовательностей:

$$1. 2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$$

by Donald Shell, 1959

Сложность: $\Theta(N^2)$



Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$

Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$

Сложность: ?

by Thomas N. Hibbard, 1963

Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$

Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$

Сложность: $\Theta(N^{\frac{3}{2}})$

by Thomas N. Hibbard, 1963



Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$

Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$

Сложность: $\Theta(N^{\frac{3}{2}})$

3. $\frac{3^k - 1}{2} : 1, 4, 13, \dots$

Сложность: $\Theta(N^{\frac{3}{2}})$

by Donald Knuth, 1973



Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$

Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$

Сложность: $\Theta(N^{\frac{3}{2}})$

3. $\frac{3^k - 1}{2} : 1, 4, 13, \dots$

Сложность: $\Theta(N^{\frac{3}{2}})$

4. $4^k + 3 * 2^{k-1} + 1 : 1, 8, 23, \dots$

Сложность: $O(N^{\frac{4}{3}})$

by Robert Sedgwick, 1982



Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$ Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$ Сложность: $\Theta(N^{\frac{3}{2}})$

3. $\frac{3^k - 1}{2} : 1, 4, 13, \dots$ Сложность: $\Theta(N^{\frac{3}{2}})$

4. $4^k + 3 * 2^{k-1} + 1 : 1, 8, 23, \dots$ Сложность: $O(N^{\frac{4}{3}})$

5. 1, 4, 10, 23, 57, 132, 301, 701

by Marcin Ciura, 2001

Сортировка Шелла

Варианты (обратных) последовательностей:

1. $2^k : 1, 2, 4, \dots \lfloor \frac{N}{2} \rfloor$ Сложность: $\Theta(N^2)$

2. $2^k - 1 : 1, 3, 7, 15, \dots$ Сложность: $\Theta(N^{\frac{3}{2}})$

3. $\frac{3^k - 1}{2} : 1, 4, 13, \dots$ Сложность: $\Theta(N^{\frac{3}{2}})$

4. $4^k + 3 * 2^{k-1} + 1 : 1, 8, 23, \dots$ Сложность: $O(N^{\frac{4}{3}})$

5. $1, 4, 10, 23, 57, 132, 301, 701$ Сложность: ???

Сортировка Шелла

Статьи:

<https://sedgewick.io/wp-content/themes/sedgewick/papers/1996Shellsort.pdf>

https://www.researchgate.net/publication/2481703_Best_Increments_for_the_Average_Case_of_Shellsort

Мини-задача #5 (1 или 2 балла)

По заданному массиву количества цитирований статей рассчитать индекс Хирша исследователя.

Решить задачу необходимо на литкоде:

<https://leetcode.com/problems/h-index/>

Встроенные сортировки использовать **нельзя**.

За реализацию сортировки Shell-а со сложностью лучше **квадратичной** начисляется **дополнительный балл**.

Стабильность сортировок

Стабильность сортировок

Задан массив из **уникальных** элементов.

Преобразовать его таким образом, чтобы все элементы шли **по возрастанию**. Т.е.

отсортировать по возрастанию.

Стабильность сортировок

Задан массив из ~~уникальных~~ элементов.

Преобразовать его таким образом, чтобы все элементы шли по возрастанию. Т.е.

отсортировать по возрастанию.

Стабильность сортировок

Задан массив из ~~уникальных~~ элементов.

Преобразовать его таким образом, чтобы все элементы шли по возрастанию. Т.е.

отсортировать по возрастанию.

Изменится ли что-нибудь в алгоритмах?

Стабильность сортировок

Задан массив из ~~уникальных~~ элементов.
Преобразовать его таким образом, чтобы все
элементы шли **по возрастанию**. Т.е.
отсортировать по возрастанию.

Изменится ли что-нибудь в алгоритмах? Скорее
нет, но в такой задаче нужно думать о
стабильности.

Стабильность сортировок

Фамилия	Дата рождения
Петров	12.02.1999
Иванова	27.05.2002
Анисов	01.03.2000
Кузнецова	27.05.2002
Авдеев	12.02.1999

Стабильность сортировок

Фамилия	Дата рождения
Петров	12.02.1999
Иванова	27.05.2002
Анисов	01.03.2000
Кузнецова	27.05.2002
Авдеев	12.02.1999

сортируем
по фамилии



Фамилия	Дата рождения
Авдеев	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002
Петров	12.02.1999

Стабильность сортировок

Фамилия	Дата рождения
Авдеев	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002
Петров	12.02.1999

сортируем
по дате
рождения



Стабильность сортировок

Фамилия	Дата рождения
Авдеев	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002
Петров	12.02.1999

сортируем
по дате
рождения




Хорошо бы
при этом
сохранить
порядок для
совпадающих
элементов

Стабильность сортировок

Фамилия	Дата рождения
Авдеев	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002
Петров	12.02.1999

сортируем
по дате
рождения




Хорошо бы
при этом
сохранить
порядок для
совпадающих
элементов

Фамилия	Дата рождения
Авдеев	12.02.1999
Петров	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002

Стабильность сортировок

Фамилия	Дата рождения
Авдеев	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002
Петров	12.02.1999

сортируем
по дате
рождения



Хорошо бы
при этом
сохранить
порядок для
совпадающих
элементов

Фамилия	Дата рождения
Авдеев	12.02.1999
Петров	12.02.1999
Анисов	01.03.2000
Иванова	27.05.2002
Кузнецова	27.05.2002

Стабильность сортировок

Сортировка называется **стабильной** (устойчивой), если во время своей работы она сохраняет исходный порядок на **совпадающих** элементах.

Стабильность сортировок

Сортировка называется **стабильной** (устойчивой), если во время своей работы она сохраняет исходный порядок на **совпадающих** элементах.

Какие из пройденных нами сортировок являются стабильными? 🤔

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

Сортировка выбором

```
def selection_sort(array: int[]):  
    for i in [0, len(array)):  
        min_index = i  
        for j in [i + 1, len(array)):  
            if array[min_index] > array[j]:  
                min_index = j  
        swap(array, i, min_index)
```

Является ли стабильной сортировкой?

Сортировка выбором

2	15	53	15	7	6
---	----	----	----	---	---

Сортировка выбором

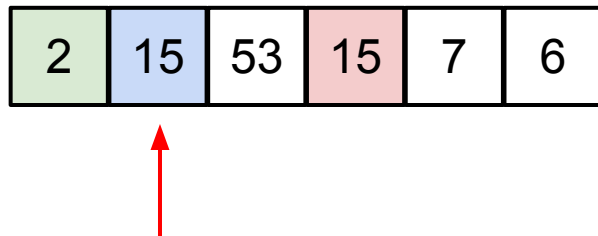
2	15	53	15	7	6
---	----	----	----	---	---

Сортировка выбором

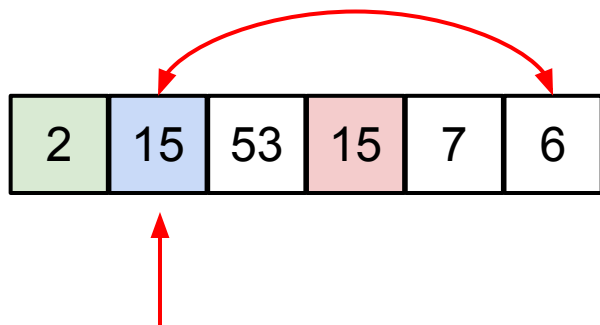
2	15	53	15	7	6
---	----	----	----	---	---



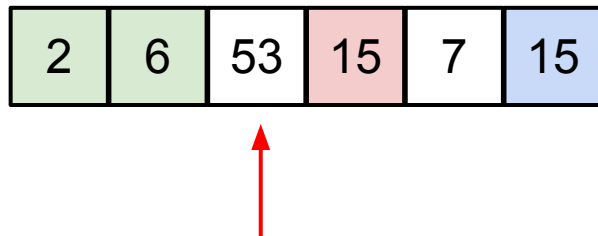
Сортировка выбором



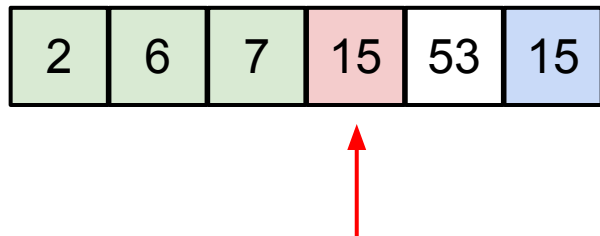
Сортировка выбором



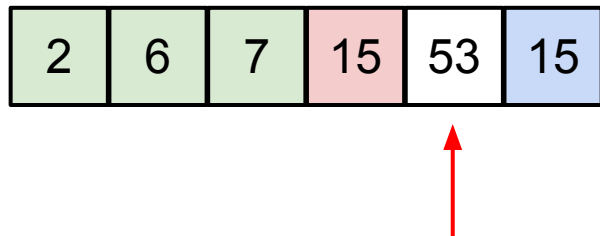
Сортировка выбором



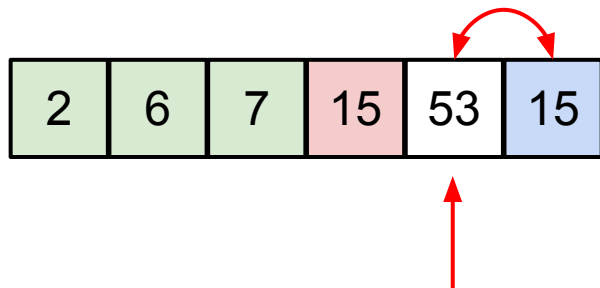
Сортировка выбором



Сортировка выбором



Сортировка выбором



Сортировка выбором

2	6	7	15	15	53
---	---	---	----	----	----

Сортировка выбором

2	6	7	15	15	53
---	---	---	----	----	----

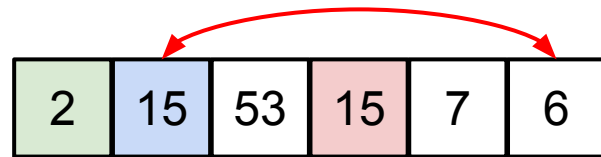
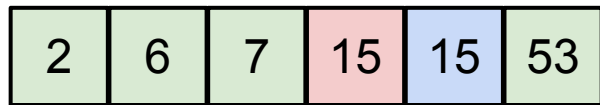


пятнашки поменялись местами =>
сортировка **не является** стабильной

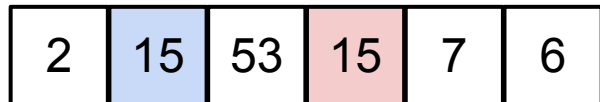
2	15	53	15	7	6
---	----	----	----	---	---

Сортировка выбором

Эмпирический признак нестабильности:
дальний swap, когда вы отправляете
элемент куда-то в конец

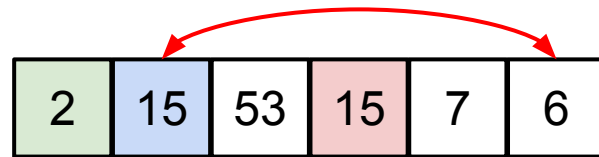
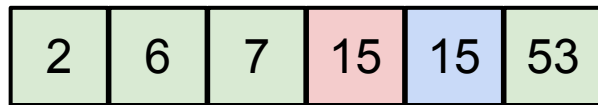


пятнашки поменялись местами =>
сортировка **не является** стабильной

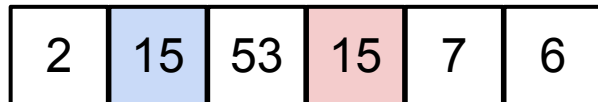


Сортировка выбором

Эмпирический признак нестабильности:
дальний swap, когда вы отправляете
элемент куда-то в конец



пятнашки поменялись местами =>
сортировка **не является** стабильной



Чем хороша сортировка **пузырьком**,
так это тем, что она **стабильна**.

И еще лучшим случаем за $O(N)$

Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array]):  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Является ли стабильной сортировкой?

Сортировка вставками

```
def insertion_sort(array: int[]):  
    for i in [1, len(array)):  
        j = i  
        while j > 0 and array[j - 1] > array[j]:  
            swap(array, j - 1, j)  
            j -= 1
```

Является ли стабильной сортировкой?

Да! Движение элемента влево как и раз
и заканчивается **на совпадающем** => их
порядок не меняется

Сортировка Шелла

1. Пусть есть некоторая убывающая последовательность целых чисел:
 k_1, k_2, \dots, k_p ; где $k_1 < N; k_p = 1$.
2. Последовательно k_i -сортируем массив
3. На последнем шаге ($k_p = 1$) запускаем сортировку **вставками**.

Сортировка Шелла

1. ...
2. Последовательно k_i -сортируем массив
3. На последнем шаге ($k_p = 1$) запускаем сортировку **вставками**.

Стабильная сортировка?

Сортировка Шелла

1. ...
2. Последовательно k_i -сортируем массив
3. На последнем шаге ($k_p = 1$) запускаем сортировку **вставками**.

Стабильная сортировка? **Нет**.

Упражнение: построить контрпример.

Стабильность сортировок

Стабильность - важная характеристика сортировок, в будущем при анализе сортировок будем за этим **следить**.



Бонус

Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:  
    for i in [1, len(array)):  
        if array[i] < array[i - 1]:  
            return False  
    return True
```

Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```


Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:  
    for i in [1, len(array)):  
        if array[i] < array[i - 1]:  
            return False  
    return True
```

```
def bogosort(array: int[]):  
    while not is_sorted(array):  
        shuffle(array)
```



Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

Сложность
сортировки?

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```

Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

Сложность
сортировки?

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```

Если shuffle
просто перебирает
все комбинации?

Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

Сложность
сортировки?

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```

Если shuffle
просто перебирает
все комбинации?

$O(N * N!)$

Bogosort a.k.a. MonkeySort

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

Сложность
сортировки?

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```

Если shuffle
просто перебирает
все комбинации?

$O(N * N!)$

Bogosort a.k.a. MonkeySort

Стабильность?
Конечно нет :)

```
def is_sorted(array: int[]) -> bool:
    for i in [1, len(array)):
        if array[i] < array[i - 1]:
            return False
    return True
```

Сложность
сортировки?

```
def bogosort(array: int[]):
    while not is_sorted(array):
        shuffle(array)
```

Если shuffle
просто перебирает
все комбинации?

$O(N * N!)$

Takeaways

- Задача сортировки элементов в массиве

Takeaways

- Задача сортировки элементов в массиве
- Элементарные сортировки: выбором, ~~пузырьком~~, вставками и т.д. Сложность **квадратичная**, но есть нюансы.

Takeaways

- Задача сортировки элементов в массиве
- Элементарные сортировки: выбором, пузырьком, вставками и т.д. Сложность **квадратичная**, но есть нюансы.
- Сортировка Шелла, как выход за границу квадратичной сложности (и открытая научная задача)

Takeaways

- Задача сортировки элементов в массиве
- Элементарные сортировки: выбором, ~~пузырьком~~, вставками и т.д. Сложность **квадратичная**, но есть нюансы.
- Сортировка Шелла, как выход за границу квадратичной сложности (и открытая научная задача)
- **Стабильность** сортировок

Мини-задача #5 (1 или 2 балла)

По заданному массиву количества цитирований статей рассчитать индекс Хирша исследователя.

Решить задачу необходимо на литкоде:

<https://leetcode.com/problems/h-index/>

Встроенные сортировки использовать **нельзя**.

За реализацию сортировки Shell-а со сложностью лучше **квадратичной** начисляется **дополнительный балл**.

Мини-задача #6 (1 балл)

Отсортировать массив `array` так, чтобы выполнялось:

$$\text{array}[0] < \text{array}[1] > \text{array}[2] < \text{array}[3] > \dots$$

Гарантируется, что такой порядок существует.

Решить задачу необходимо на литкоде:

<https://leetcode.com/problems/wiggle-sort-ii/>

Во время решения можно использовать **любую** сортировку (кроме встроенных)