

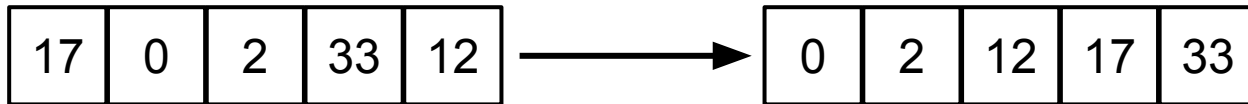
# Алгоритмы и структуры данных

Сортировка слиянием, количество инверсий



# Задача сортировки

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



# Сортировка слиянием

Переходим в следующую **лигу**!



# Задача

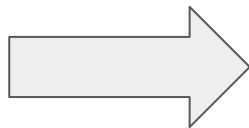
Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.

# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.

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


5	13	28	29	99
---	----	----	----	----




0	2	5	12	13	17	28	29	33	99
---	---	---	----	----	----	----	----	----	----

# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



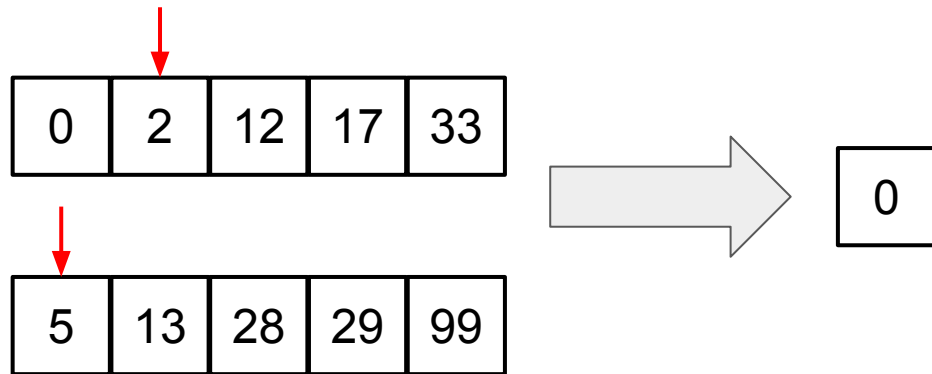
0	2	12	17	33
---	---	----	----	----



5	13	28	29	99
---	----	----	----	----

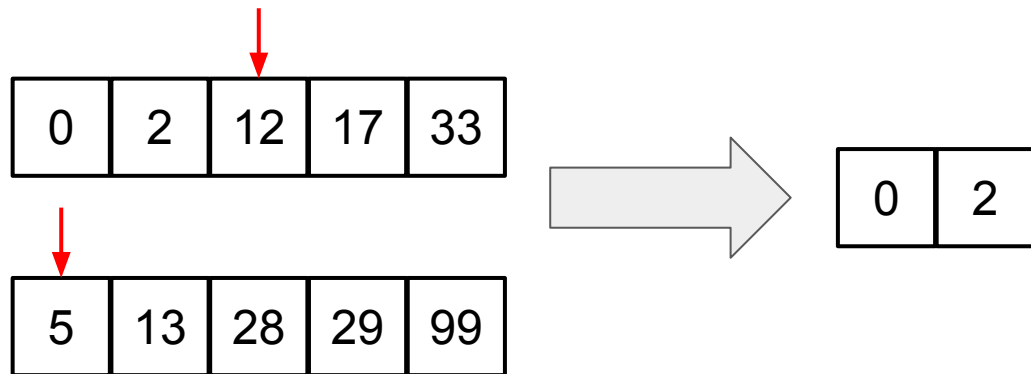
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



# Задача

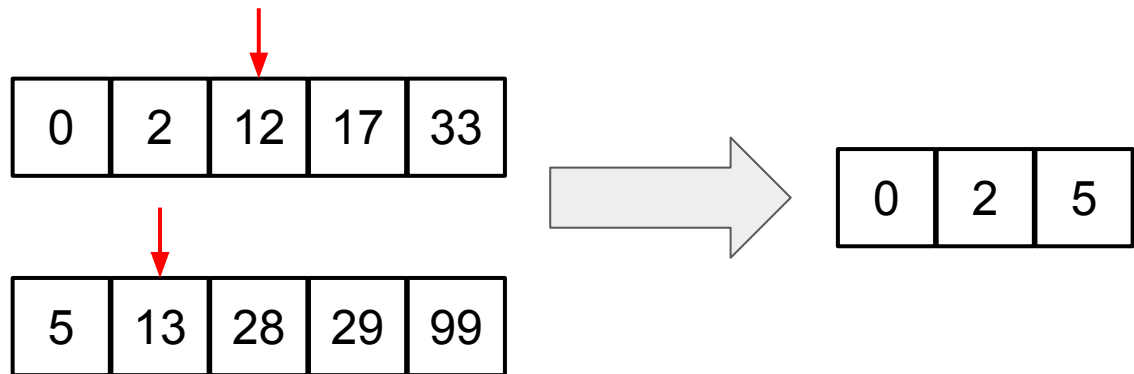
Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.





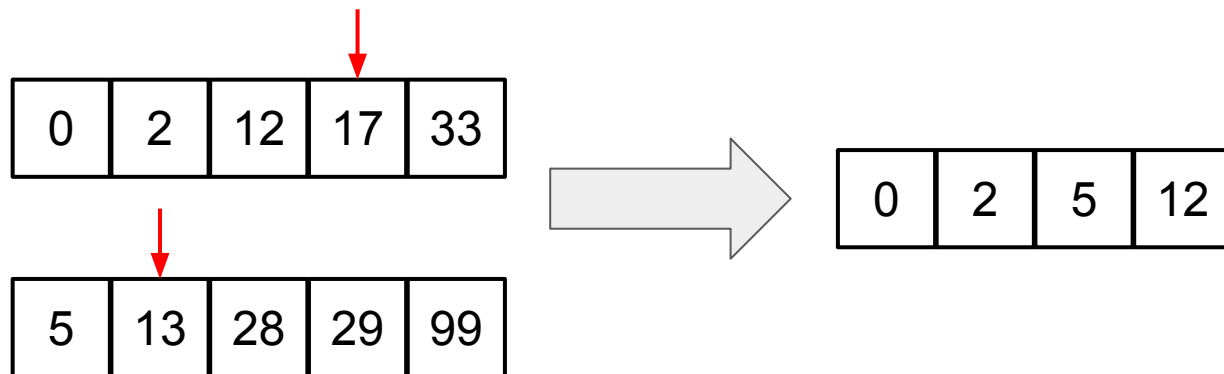
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



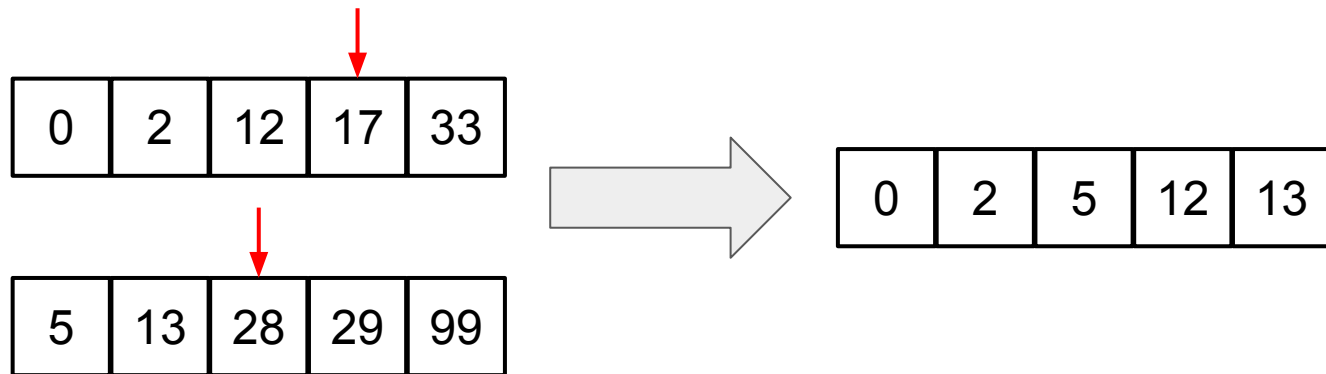
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



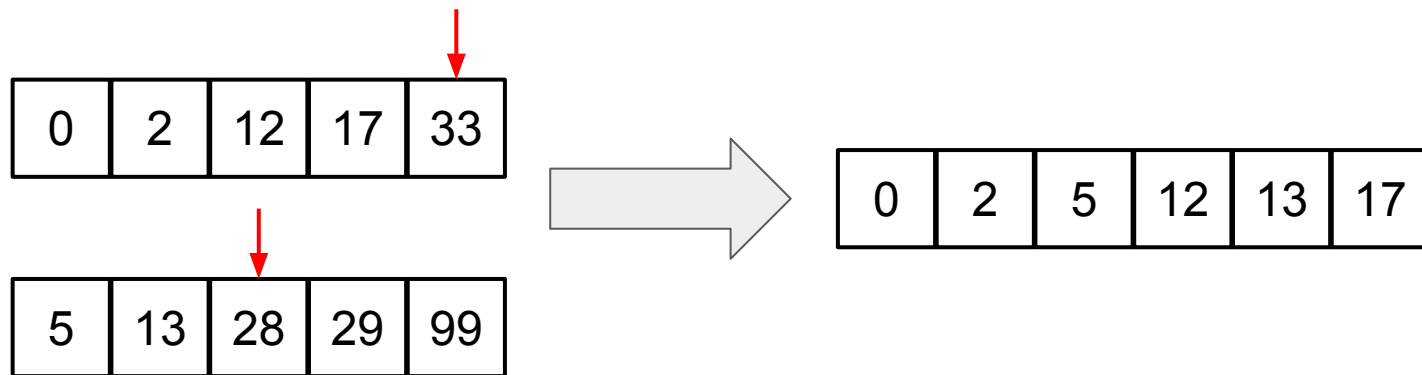
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



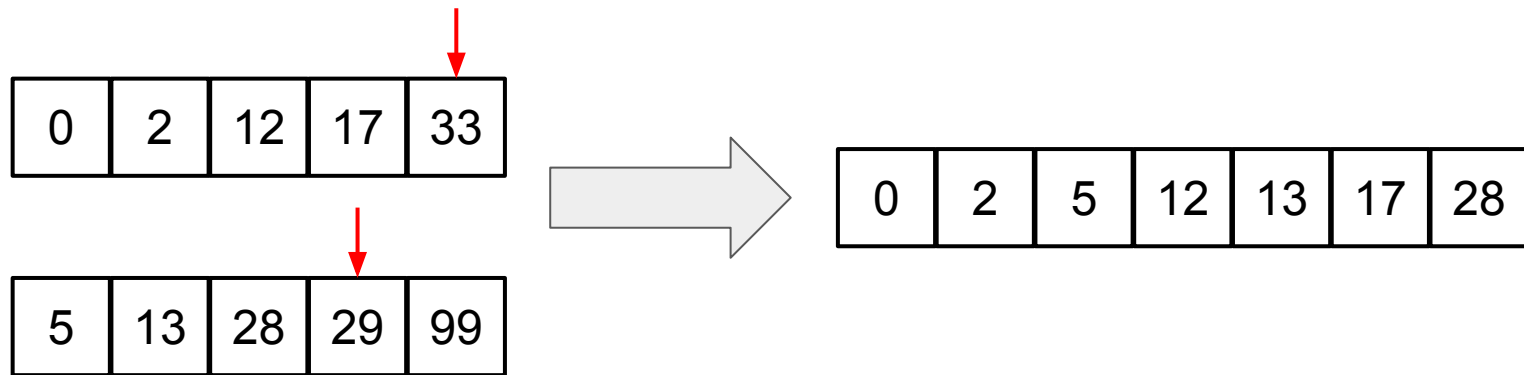
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



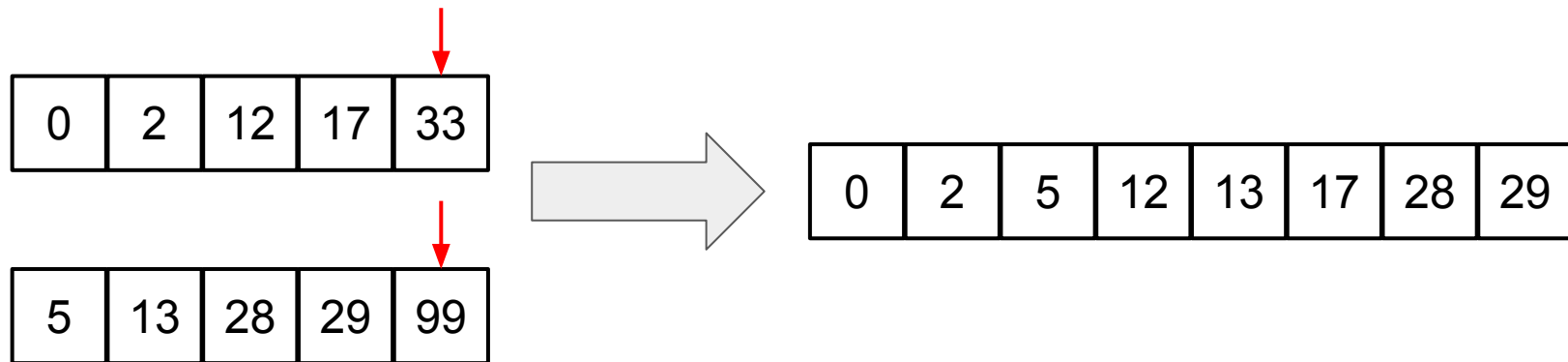
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



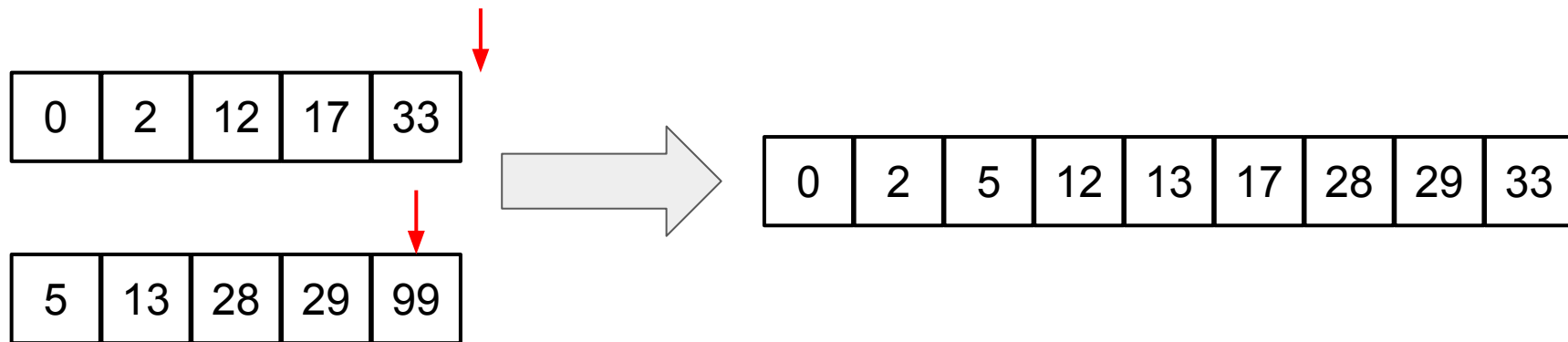
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



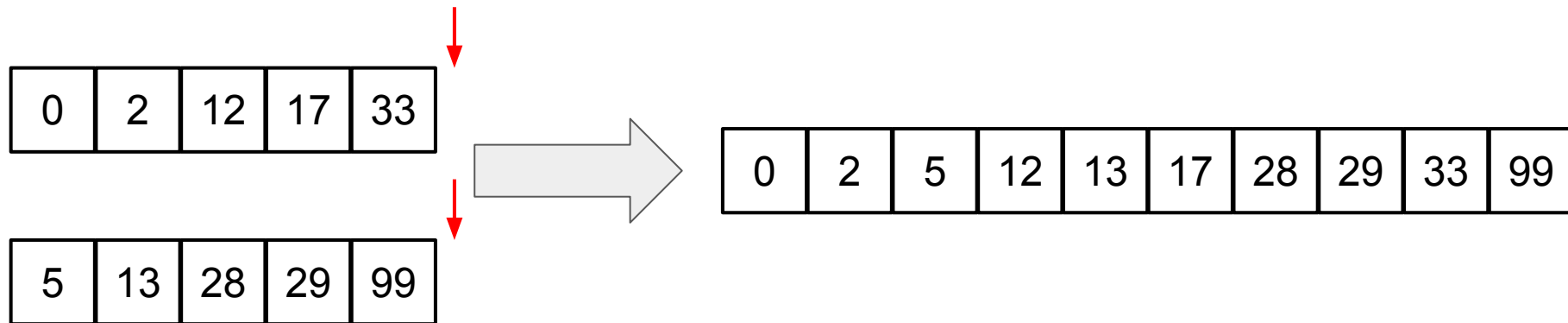
# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.



# Задача

Пусть есть **два** отсортированных массива размерности  $N/2$ . Построить **третий** массив, состоящий из элементов первых двух, но при этом **отсортированный**.





## Процедура слияния

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize
```

## Процедура слияния

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:
```

## Процедура слияния

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]
```

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]
```

???

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]  
  
    while i < lsize: res[k++] = left[i++]  
    while j < rsize: res[k++] = right[j++]
```

Сложность?

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]  
  
    while i < lsize: res[k++] = left[i++]  
    while j < rsize: res[k++] = right[j++]
```

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]
```

Сложность?

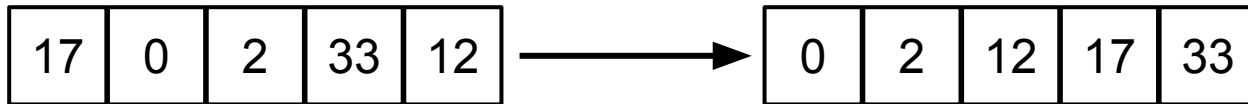
$O(N)$

где N – длина res

```
while i < lsize: res[k++] = left[i++]  
while j < rsize: res[k++] = right[j++]
```

# Задача сортировки

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





# Задача сортировки

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

Вот бы он бы состоял из двух уже  
отсортированных массивов...

# Задача сортировки

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

Вот бы он бы состоял из двух уже  
отсортированных массивов...

Значит пора применять рекурсию!



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

```
def merge_sort(array: int[]):  
    if len(array) == 1: return
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
  
    if len(array) == 2:  
        if array[0] > array[1]:  
            swap_elements(array, 0, 1)  
        return
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]
```



```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle], array[middle:], buffer)
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle], array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...
```

Сложность?

```
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])
```

```
    buffer = new int[len(array)]  
    merge(array[:middle], array[middle:], buffer)
```

```
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...
```

Сложность?

Сколько раз будет  
вызываться функция merge?

```
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])
```

```
    buffer = new int[len(array)]  
    merge(array[:middle], array[middle:], buffer)
```

```
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...
```

Сложность?

Сколько раз будет  
вызываться функция merge?

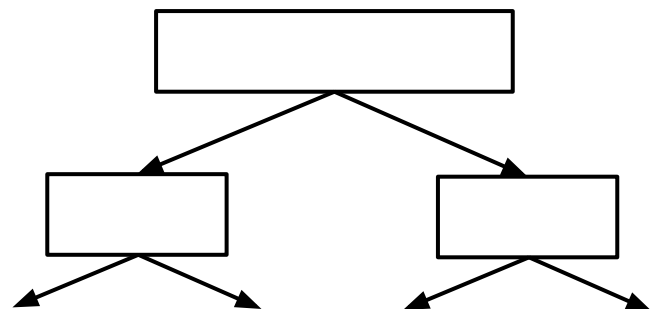
```
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])
```

$\log_2 N$  (как бин. поиск)

```
    buffer = new int[len(array)]  
    merge(array[:middle], array[middle:], buffer)
```

```
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

# Сортировка слиянием

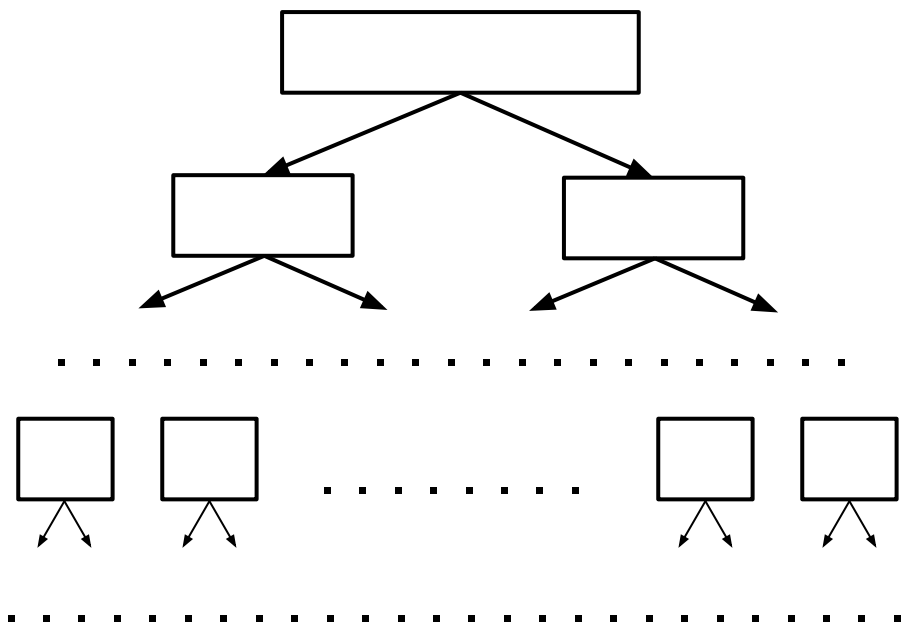


уровень 0

уровень 1

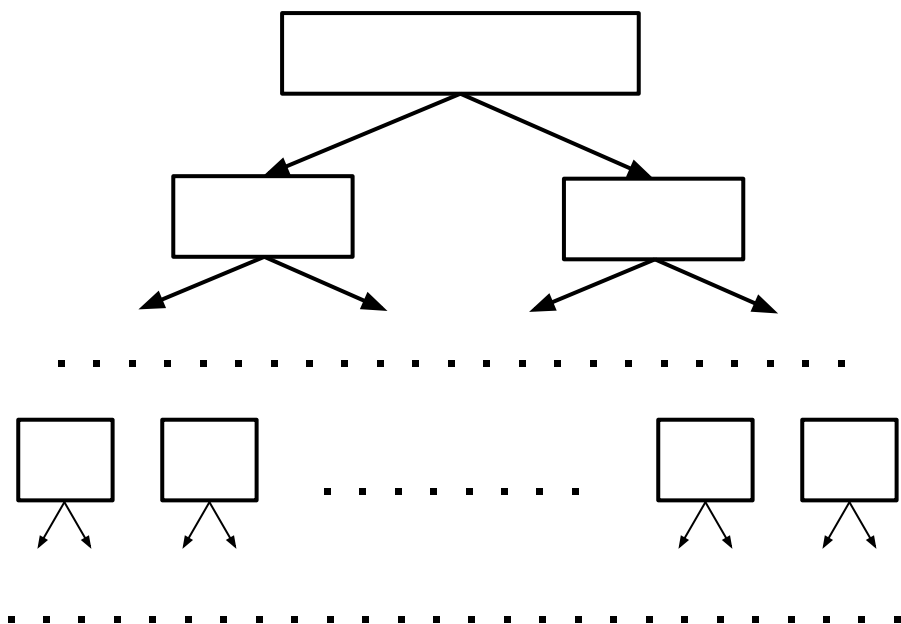
уровень  $j$

# Сортировка слиянием



Сколько разных  
массивов на уровне j?

# Сортировка слиянием



уровень 0

уровень 1

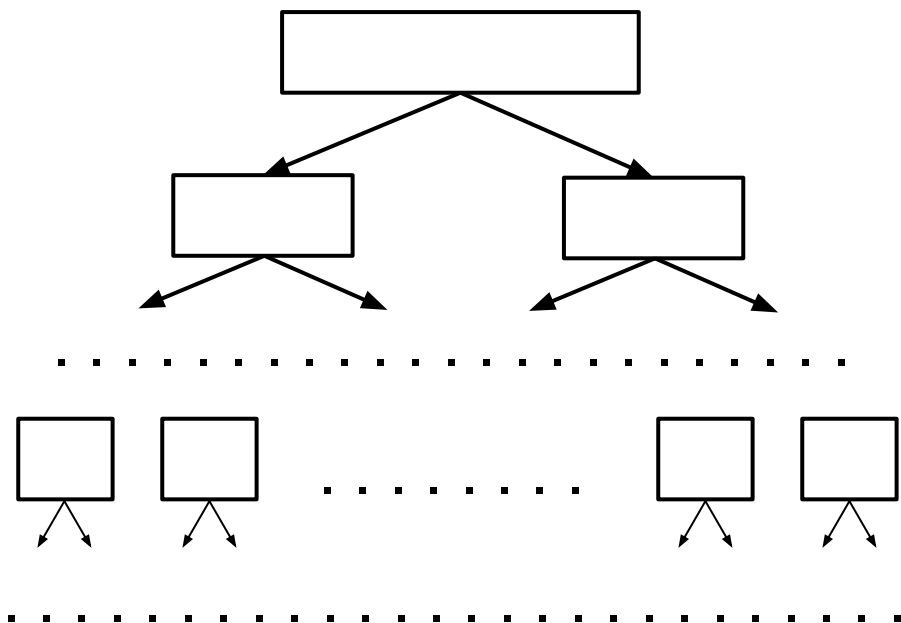
уровень j

Сколько разных  
массивов на уровне j?

$$2^j$$



# Сортировка слиянием



уровень 0

уровень 1

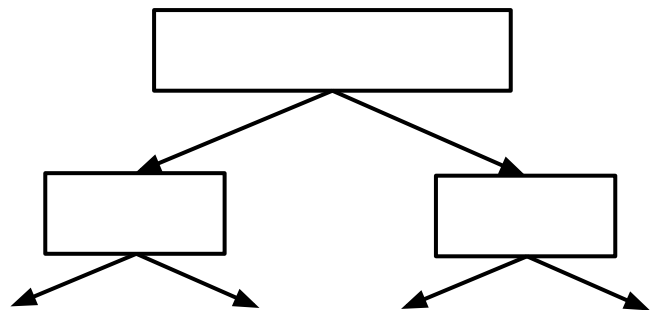
уровень j

Сколько разных  
массивов на уровне j?

$$2^j$$

Размер массива на уровне j?

# Сортировка слиянием



уровень 0

уровень 1

уровень j

Сколько разных  
массивов на уровне j?

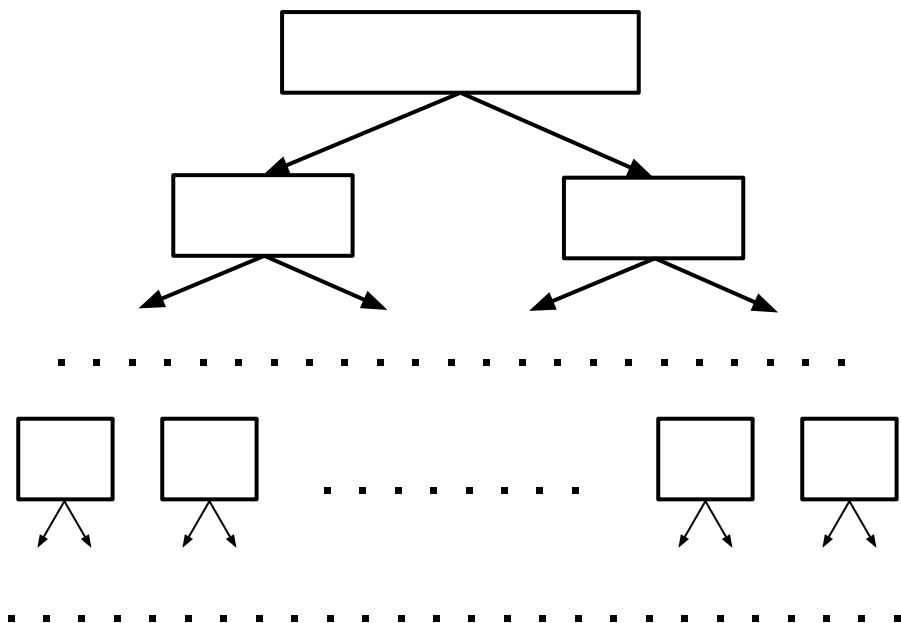
$$2^j$$

Размер массива на уровне j?

$$\frac{n}{2^j}$$



# Сортировка слиянием



уровень 0

уровень 1

уровень j

Сколько разных  
массивов на уровне j?

$$2^j$$

Размер массива на уровне j?  $\frac{n}{2^j}$

Количество операций для  
слияния на уровне j?

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]
```

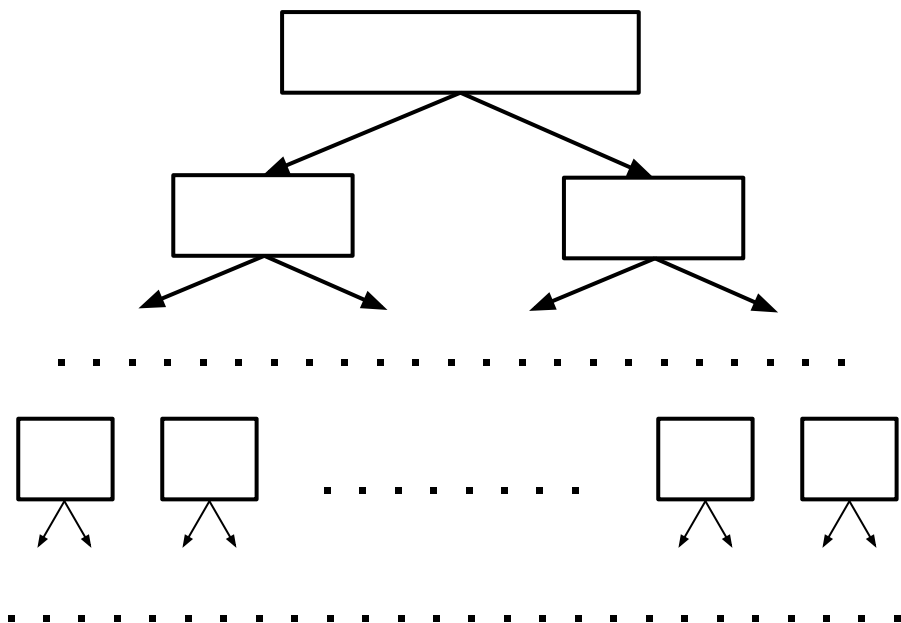
Сложность?

$O(N)$

где N – длина res

```
while i < lsize: res[k++] = left[i++]  
while j < rsize: res[k++] = right[j++]
```

# Сортировка слиянием



уровень 0

уровень 1

уровень j

Сколько разных  
массивов на уровне j?

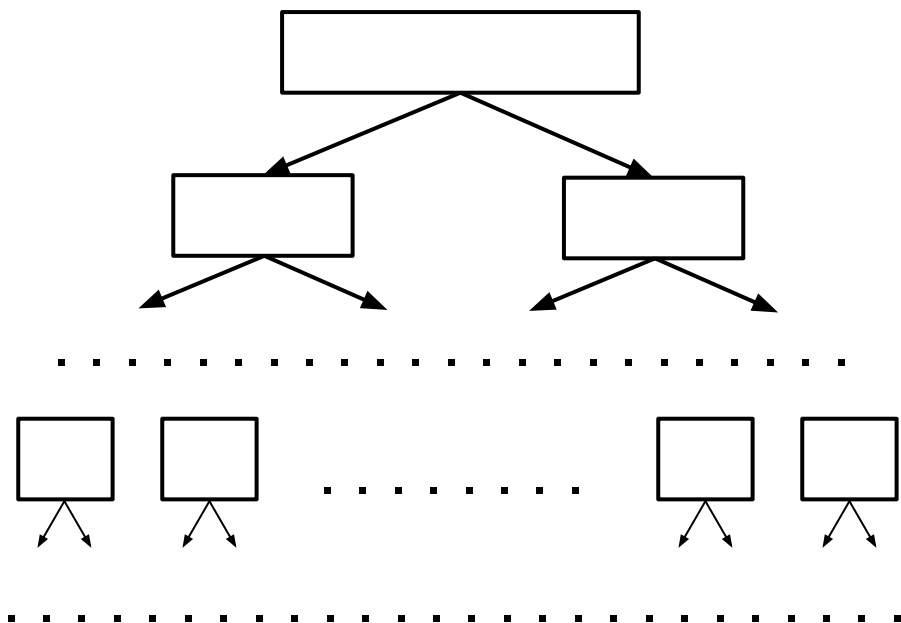
$$2^j$$

Размер массива на уровне j?  $\frac{n}{2^j}$

Количество операций для  
слияния на уровне j?

$$T_j(n) \leq 2^j * (C * (\frac{n}{2^j}))$$

# Сортировка слиянием



уровень 0

уровень 1

уровень j

Сколько разных  
массивов на уровне j?

$$2^j$$

Размер массива на уровне j?  $\frac{n}{2^j}$

Количество операций для  
слияния на уровне j?

$$T_j(n) \leq 2^j * (C * (\frac{n}{2^j}))$$

$$T_j(n) \leq C * n$$

Не зависит от j!

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сложность?

Сколько раз будет  
вызываться функция merge?

$\log_2 N$  (как бин. поиск)

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сложность?

Сколько раз будет  
вызываться функция merge?

$\log_2 N$  (как бин. поиск)

$$T(N) \leq \log_2 N * T_j(N) \leq$$



```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сложность?

Сколько раз будет  
вызываться функция merge?

$\log_2 N$  (как бин. поиск)

$$\begin{aligned} T(N) &\leq \log_2 N * T_j(N) \leq \\ &\leq \log_2 N * C * N \end{aligned}$$

```

def merge_sort(array: int[]):
    if len(array) == 1: return
    if len(array) == 2: ...
    middle = len(array) / 2
    merge_sort(array[:middle])
    merge_sort(array[middle:])

    buffer = new int[len(array)]
    merge(array[:middle],
          array[middle:], buffer)

    for i in [0; len(array)):
        array[i] = buffer[i]

```

Сложность?

Сколько раз будет  
вызываться функция merge?

$\log_2 N$  (как бин. поиск)

$$T(N) \leq \log_2 N * T_j(N) \leq \log_2 N * C * N$$

$$T(N) = O(N * \log_2 N)$$



## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!

```
def merge(left, right, res: int[]):  
    lsize, rsize = len(left), len(right)  
    n = len(res)  
    assert n == lsize + rsize  
    i, j, k = 0, 0, 0  
    while k < n and i < lsize and j < rsize:  
        if left[i] < right[j]:  
            res[k++] = left[i++]  
        else:  
            res[k++] = right[j++]  
  
    while i < lsize: res[k++] = left[i++]  
    while j < rsize: res[k++] = right[j++]
```

```
def merge(left, right, res: int[]):
    lsize, rsize = len(left), len(right)
    n = len(res)
    assert n == lsize + rsize
    i, j, k = 0, 0, 0
    while k < n and i < lsize and j < rsize:
        if left[i] <= right[j]:
            res[k++] = left[i++]
        else:
            res[k++] = right[j++]
    while i < lsize: res[k++] = left[i++]
    while j < rsize: res[k++] = right[j++]
```

Вот теперь - стабильная!

## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

А сколько потребляет **памяти**?



## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

А сколько потребляет **памяти**?

(обычно говорят про "дополнительную" память,  
т.е. не учитывают размер исходных данных)

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$$K * len(array) + C$$

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$O(\text{len}(\text{array}))$

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$O(\text{len}(\text{array}))$

А сколько тратится памяти за  
**всю** работу алгоритма на  
массиве длины N?

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$O(\text{len}(\text{array}))$

А сколько тратится памяти за  
**всю** работу алгоритма на  
массиве длины N?

Заметим, что память нам  
нужна только **на время** merge

```
def merge_sort(array: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort(array[:middle])  
    merge_sort(array[middle:])  
  
    buffer = new int[len(array)]  
    merge(array[:middle],  
          array[middle:], buffer)  
  
    for i in [0; len(array)):  
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$O(\text{len}(\text{array}))$

А сколько тратится памяти за  
**всю** работу алгоритма на  
массиве длины N?

Заметим, что память нам  
нужна только **на время** merge

Поэтому самое **большое**  
потребление памяти – buffer  
размера N

```
def merge_sort(array: int[]):
    if len(array) == 1: return
    if len(array) == 2: ...
    middle = len(array) / 2
    merge_sort(array[:middle])
    merge_sort(array[middle:])

    buffer = new int[len(array)]
    merge(array[:middle],
          array[middle:], buffer)

    for i in [0; len(array)):
        array[i] = buffer[i]
```

Сколько тратится памяти за  
каждый вызов merge\_sort?

$O(\text{len}(\text{array}))$

А сколько тратится памяти за  
**всю** работу алгоритма на  
массиве длины N?

$O(N)$



```
def merge_sort_impl(array, buffer: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort_impl(array[:middle], buffer)  
    merge_sort_impl(array[middle:], buffer)  
  
    assert (len(array) <= len(buffer))  
    merge(array[:middle], array[middle:], buffer)  
  
    for i in [0; len(array)): array[i] = buffer[i]
```

```
def merge_sort_impl(array, buffer: int[]):  
    if len(array) == 1: return  
    if len(array) == 2: ...  
    middle = len(array) / 2  
    merge_sort_impl(array[:middle], buffer)  
    merge_sort_impl(array[middle:], buffer)  
  
    assert (len(array) <= len(buffer))  
    merge(array[:middle], array[middle:], buffer)  
    for i in [0; len(array)): array[i] = buffer[i]  
  
def merge_sort(array: int[]):  
    merge_sort_impl(array, new int[len(array)])
```

## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

А сколько потребляет **памяти**?  
 $O(N)$

## Сортировка слиянием: детали

Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

А сколько потребляет **памяти**?

$O(N)$  и это **хуже**, чем любая сортировка,  
пройденная ранее.

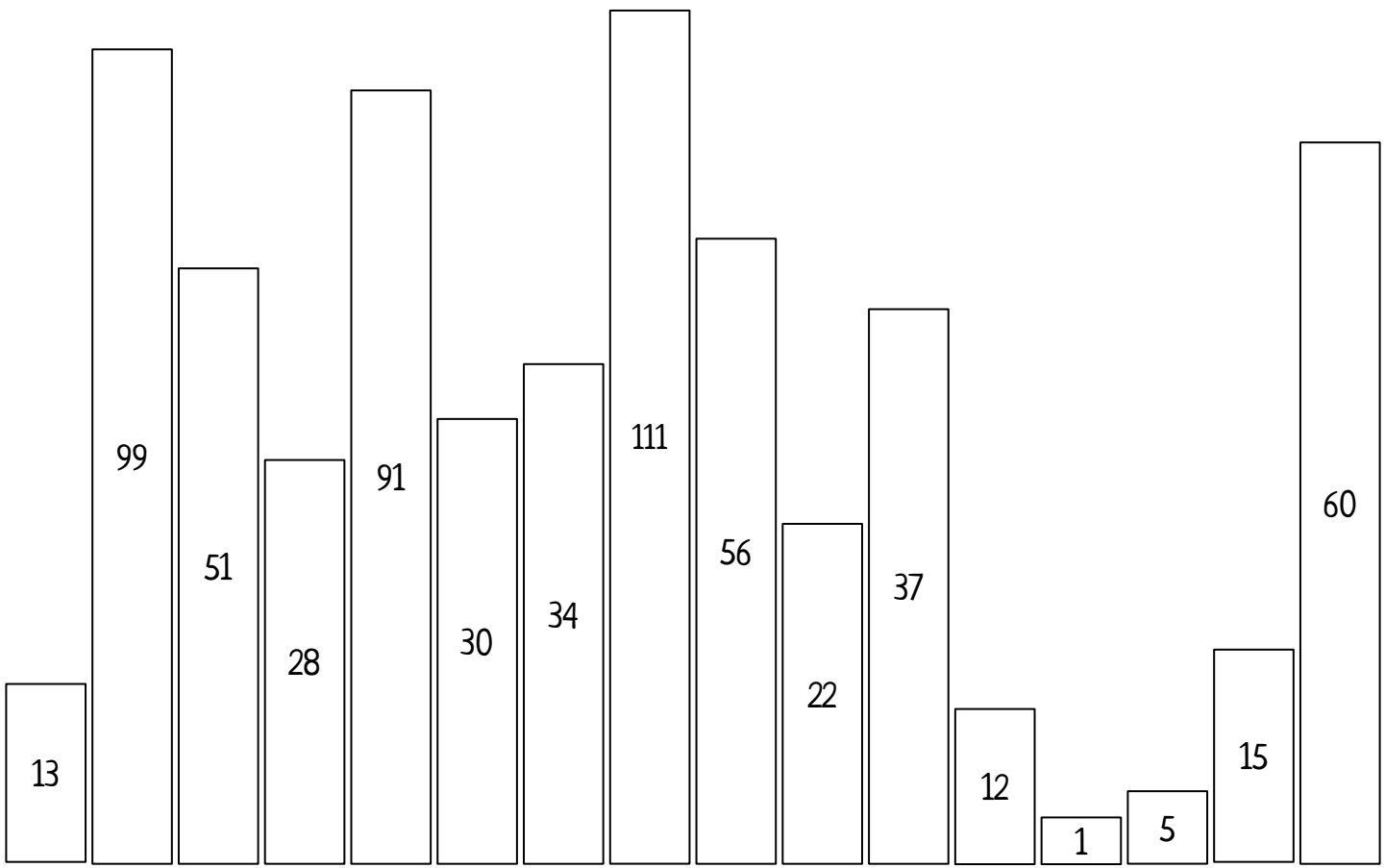
## Сортировка слиянием: детали

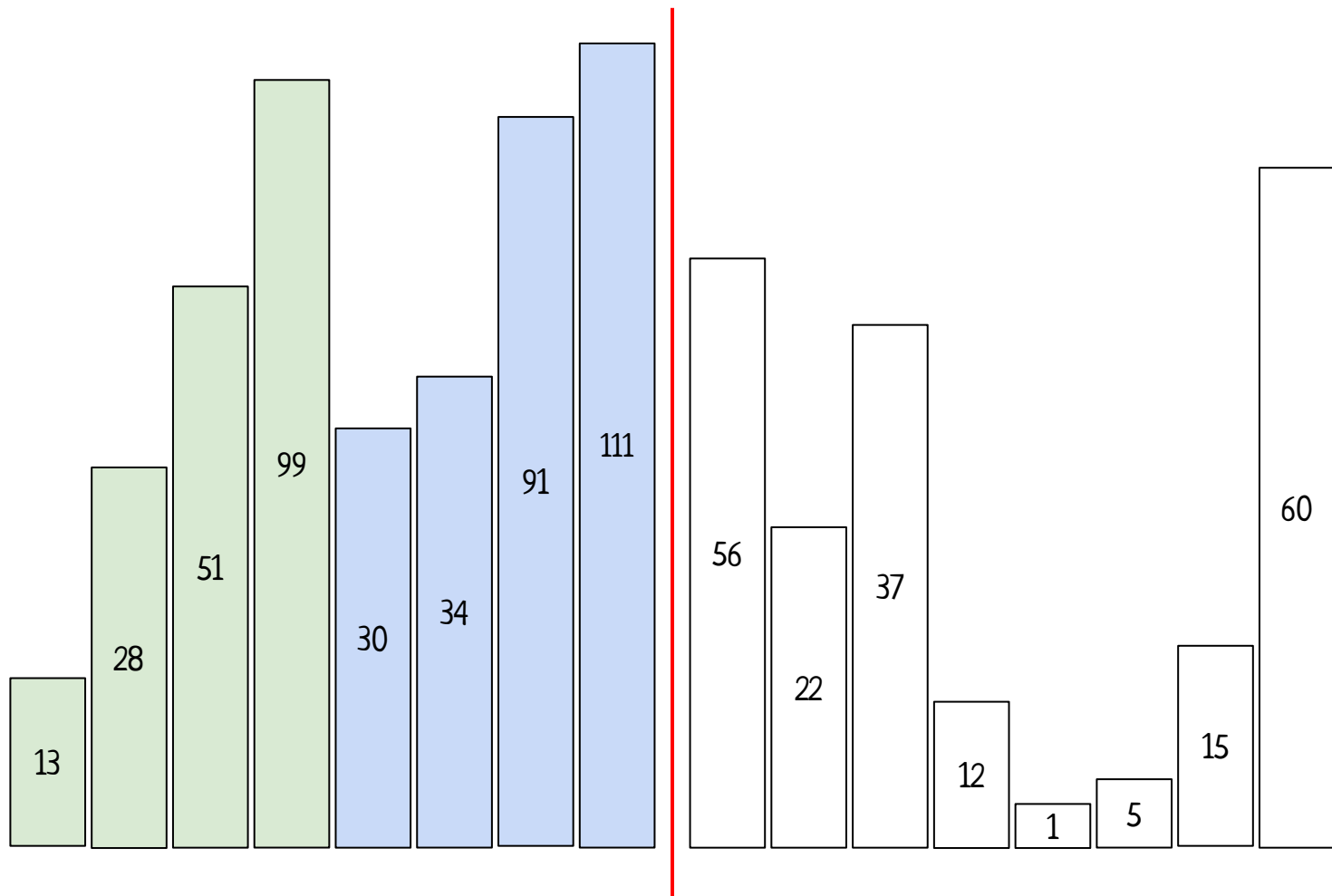
Является ли сортировка слиянием **стабильной**?

Сильно зависит от процедуры слияния!  
(обычно стабильная)

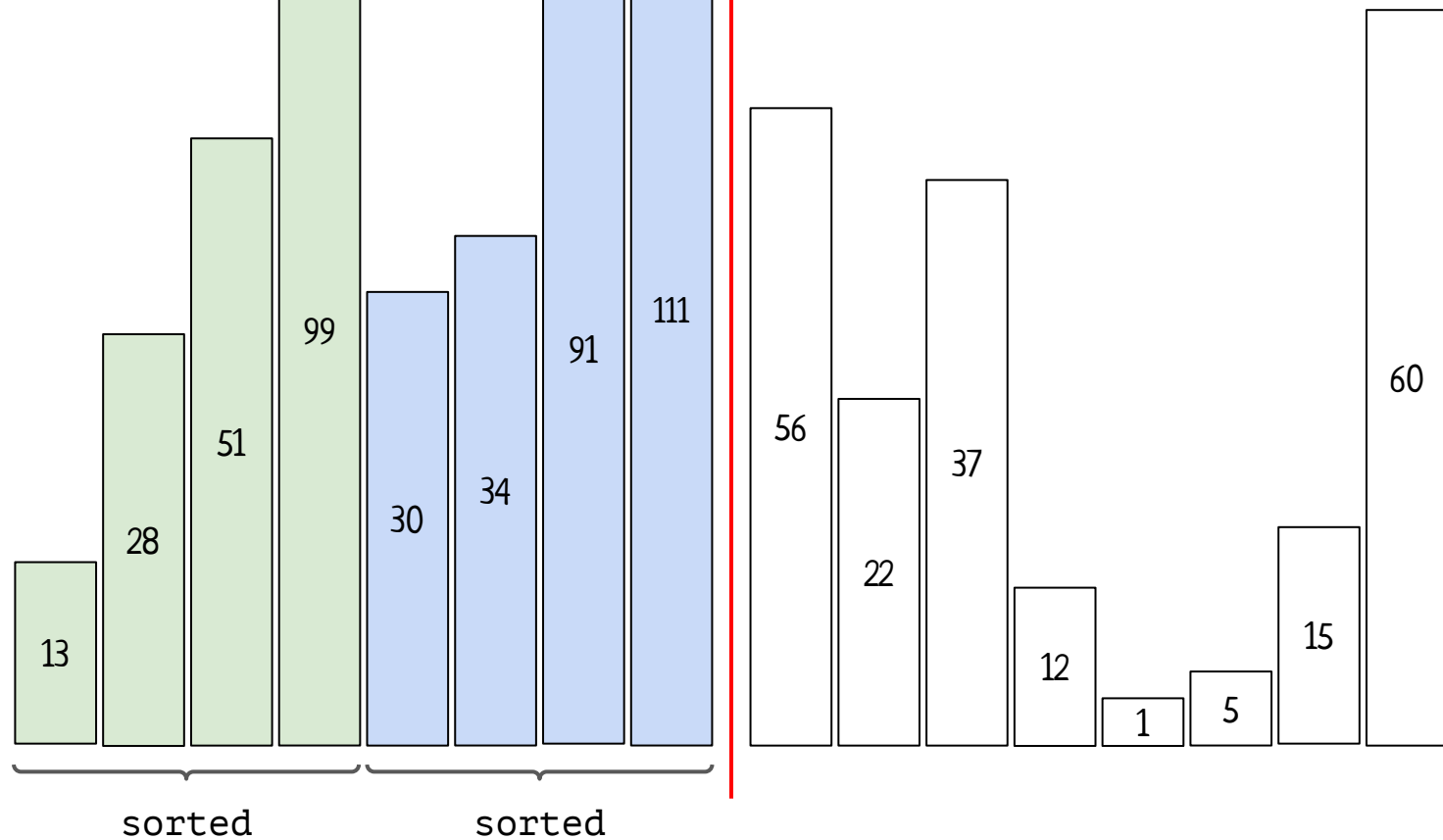
А сколько потребляет **памяти**?

$O(N)$  и это **хуже**, чем любая сортировка, пройденная ранее. Как не тратить память?





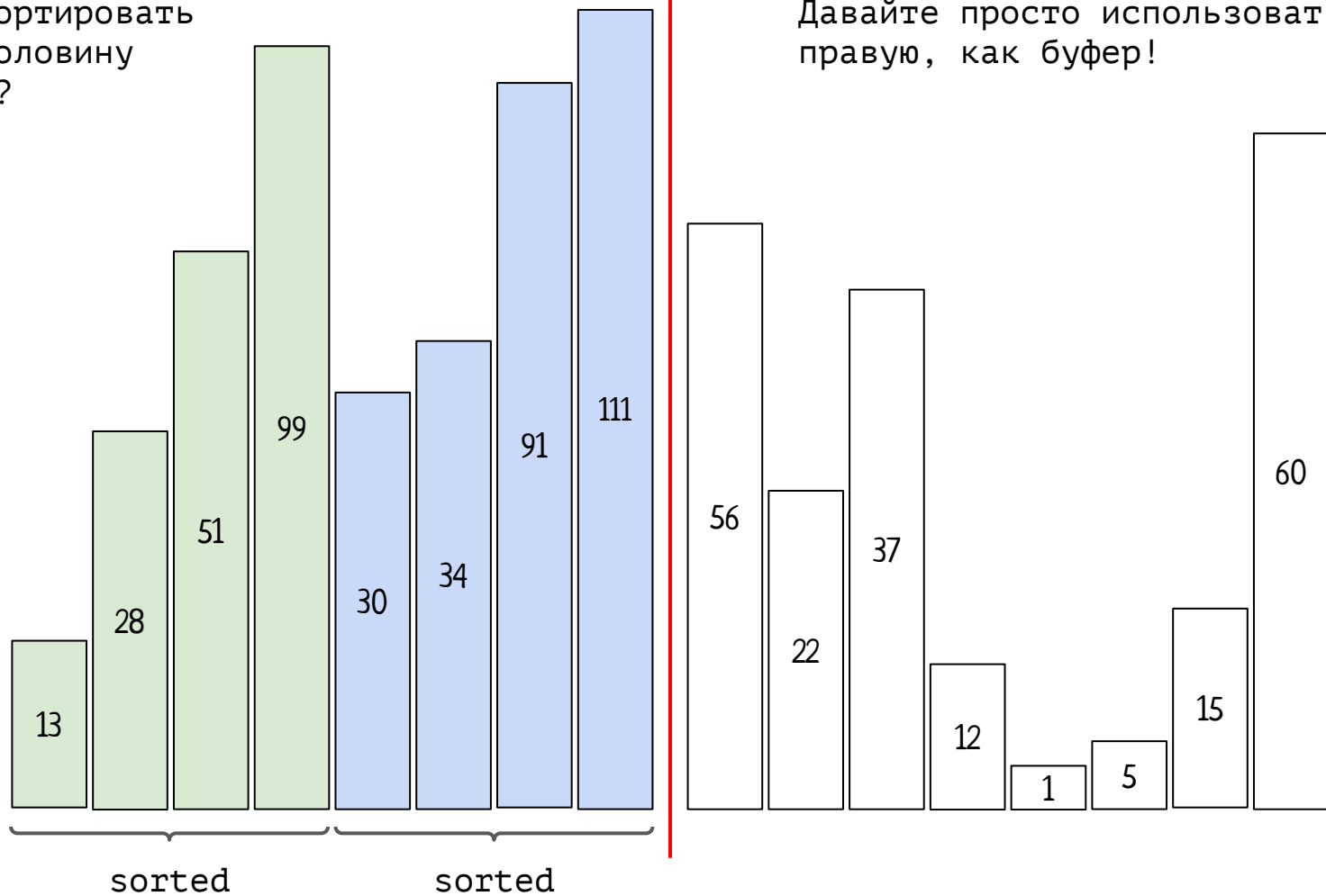
Как отсортировать  
левую половину  
массива?



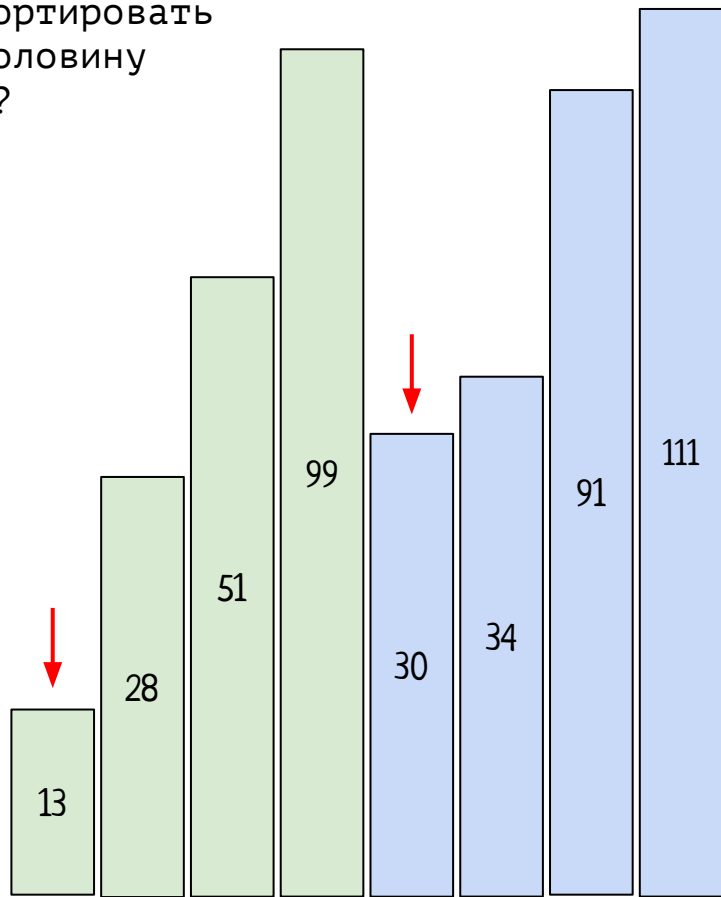


Как отсортировать  
левую половину  
массива?

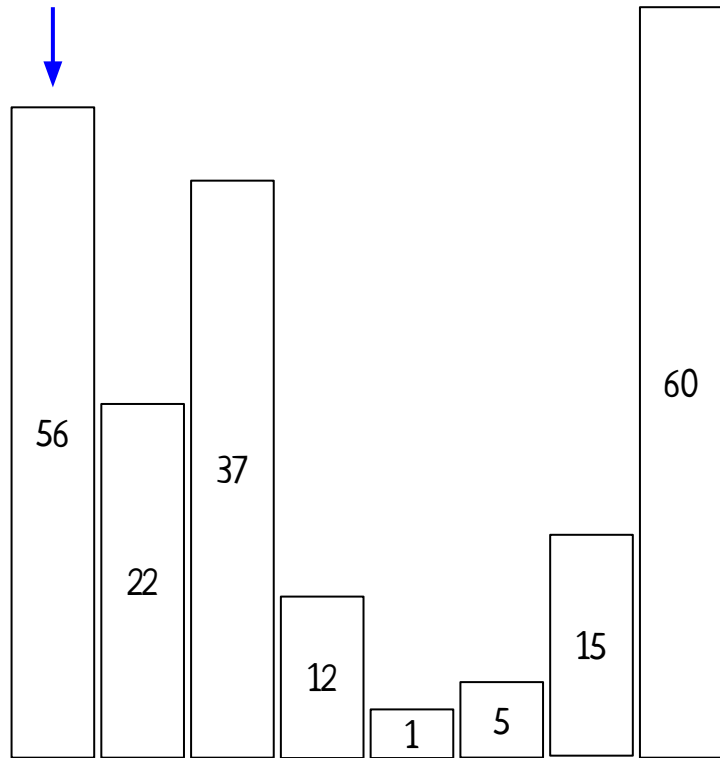
Давайте просто использовать  
правую, как буфер!



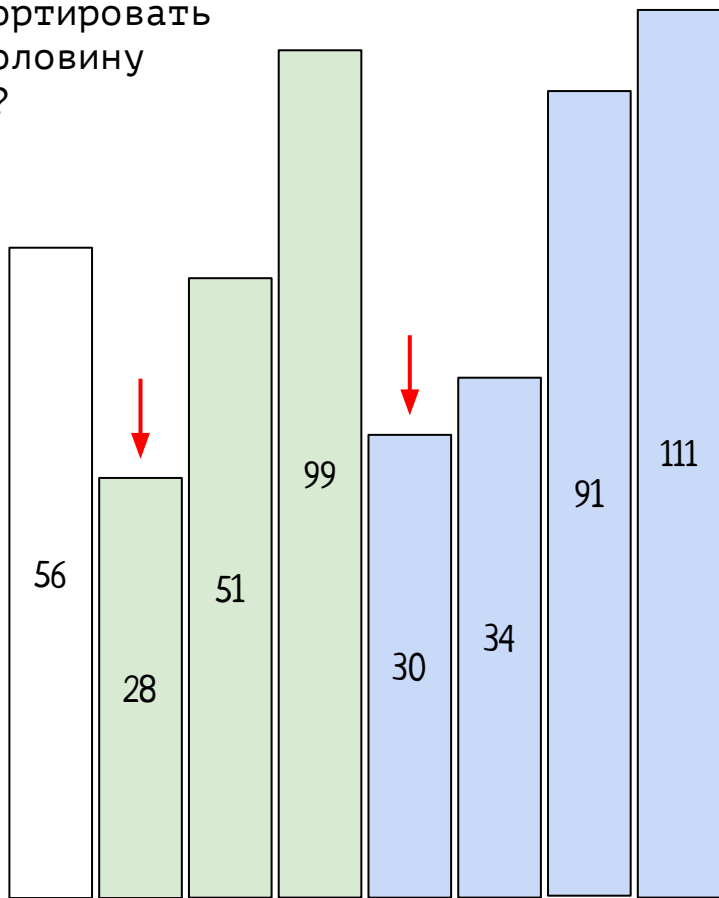
Как отсортировать  
левую половину  
массива?



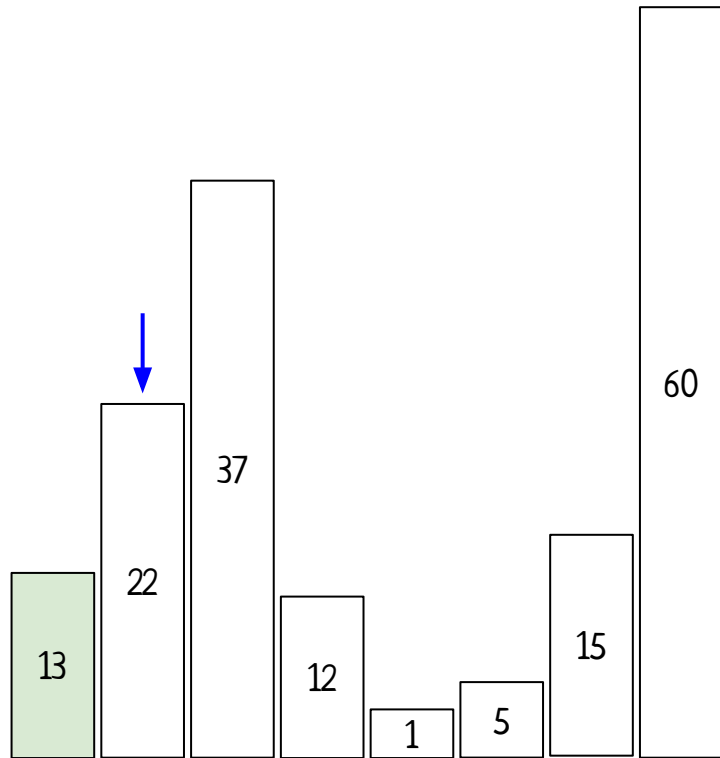
Давайте просто использовать  
правую, как буфер!



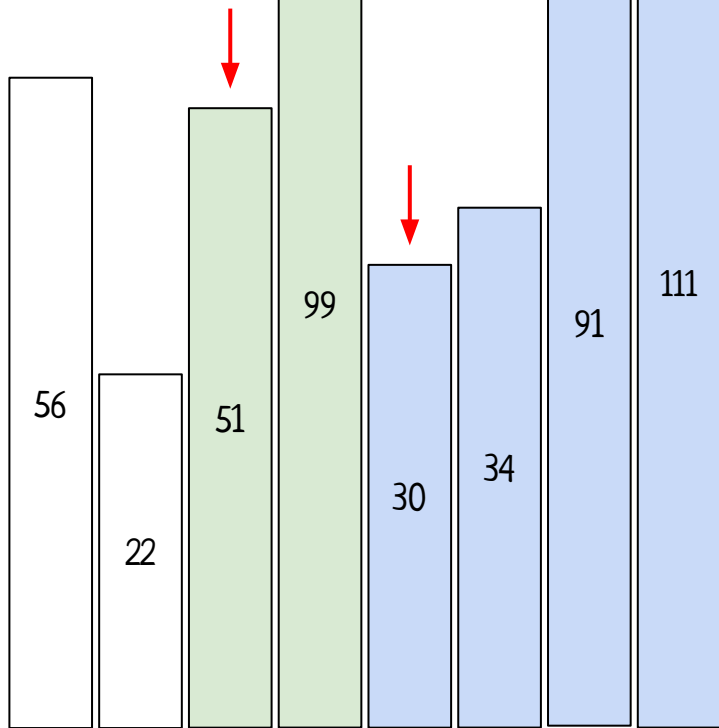
Как отсортировать  
левую половину  
массива?



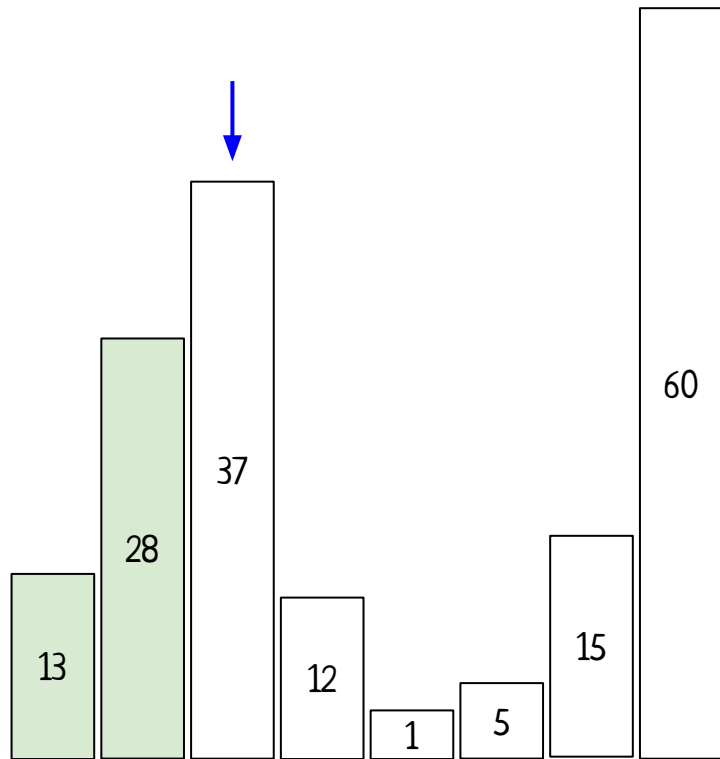
Давайте просто использовать  
правую, как буфер!



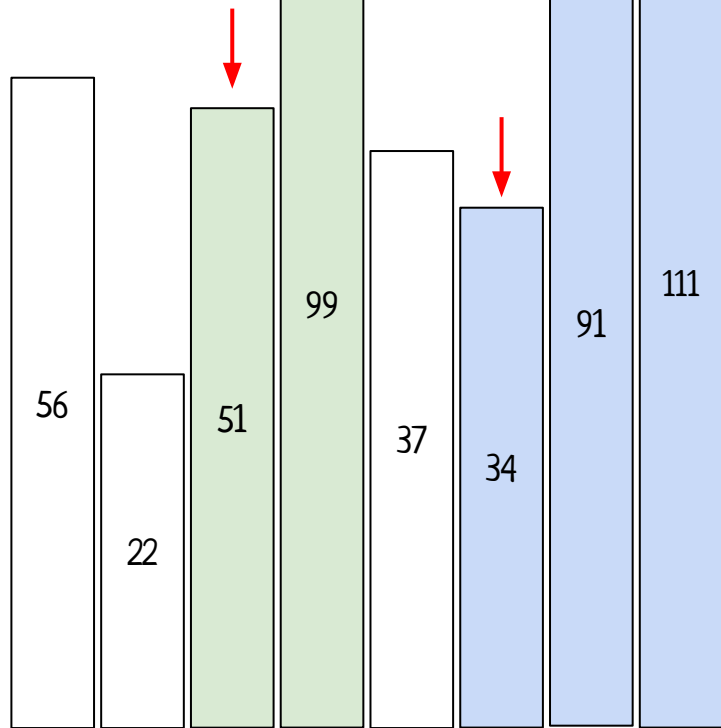
Как отсортировать  
левую половину  
массива?



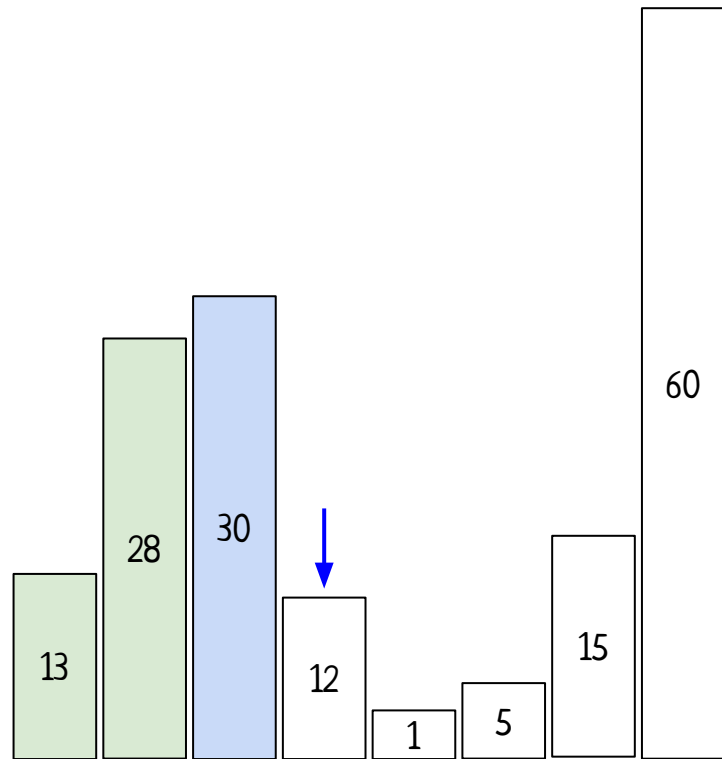
Давайте просто использовать  
правую, как буфер!



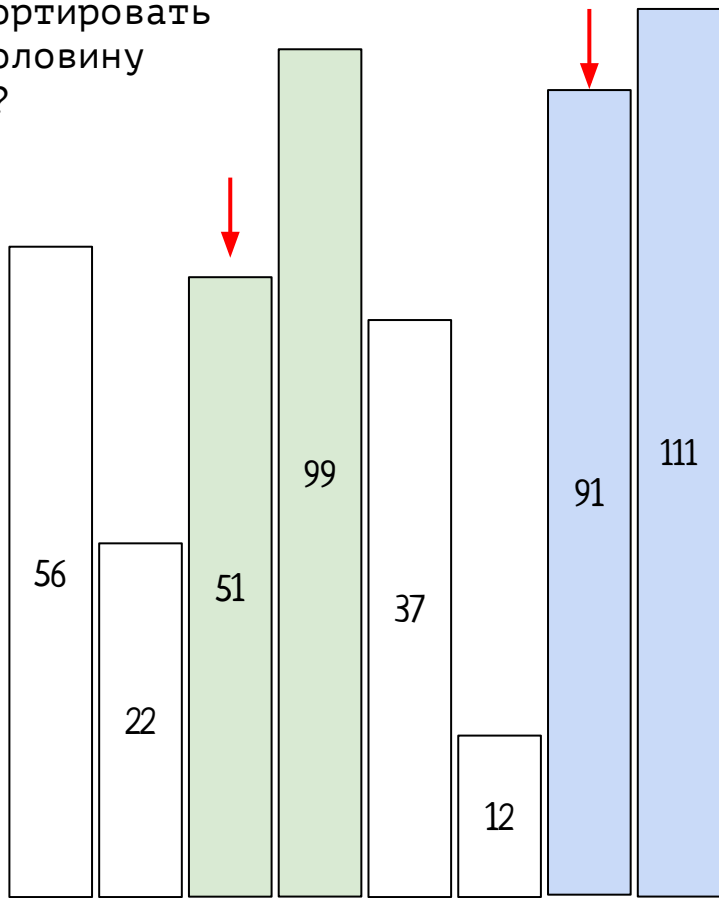
Как отсортировать  
левую половину  
массива?



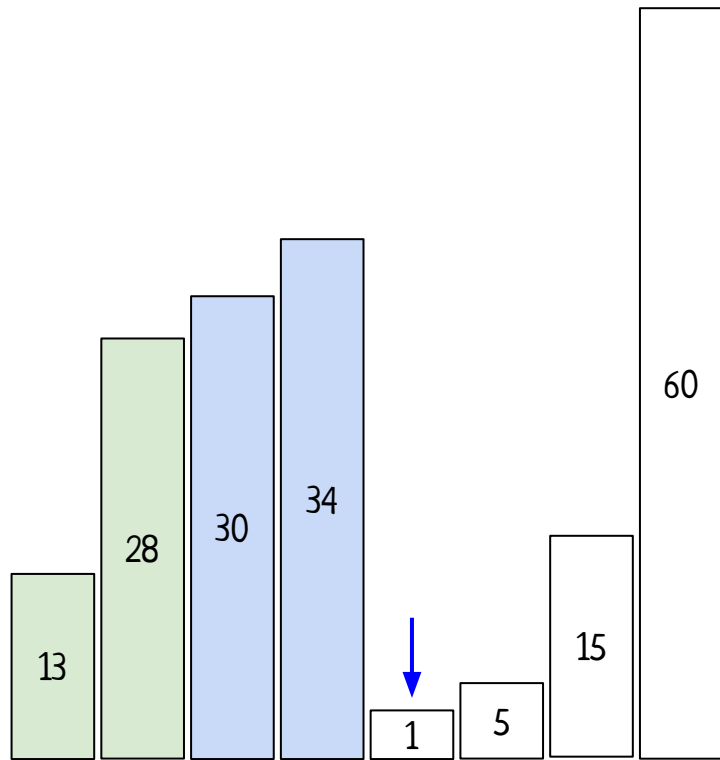
Давайте просто использовать  
правую, как буфер!



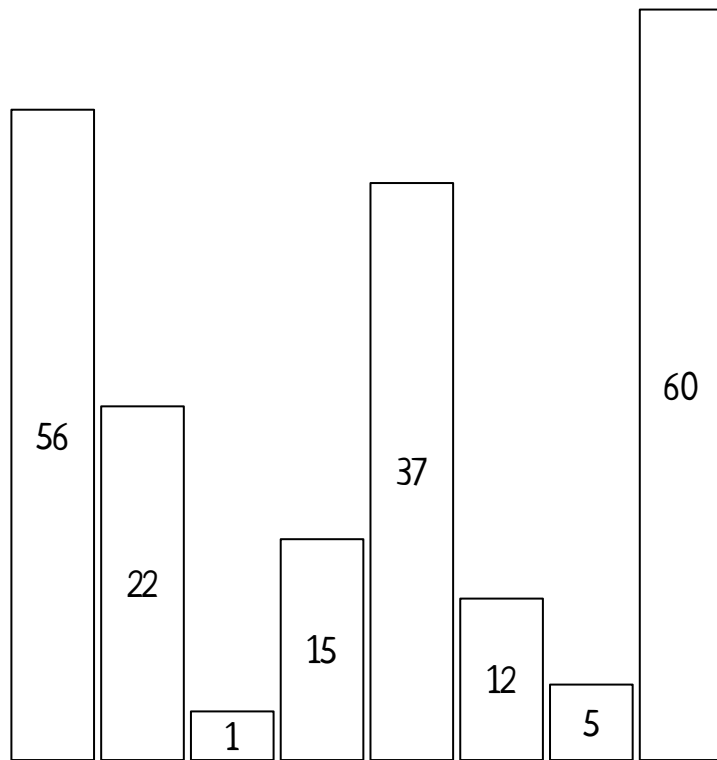
Как отсортировать  
левую половину  
массива?



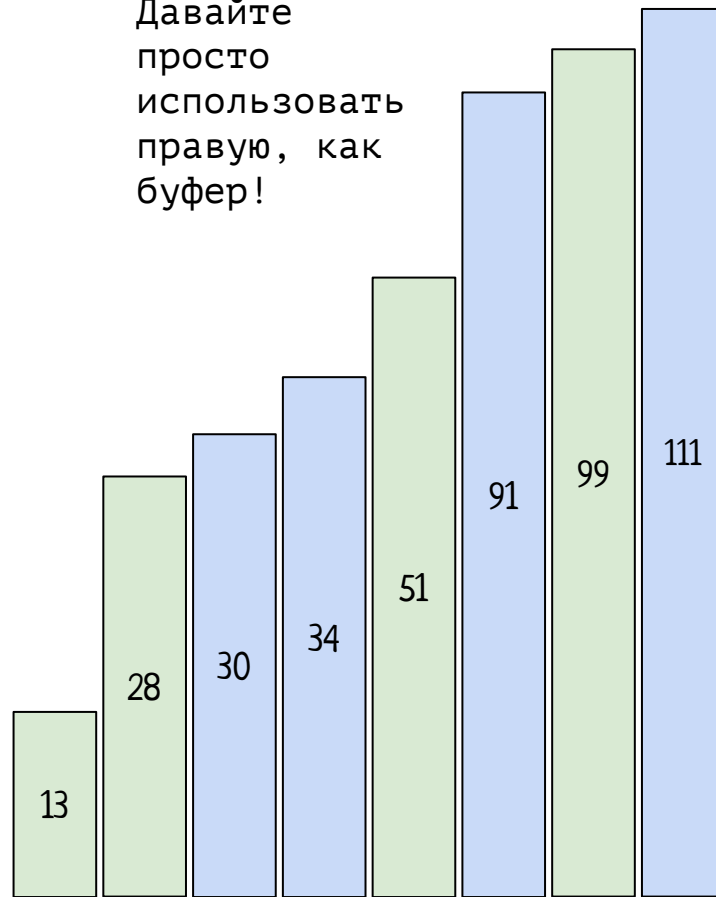
Давайте просто использовать  
правую, как буфер!



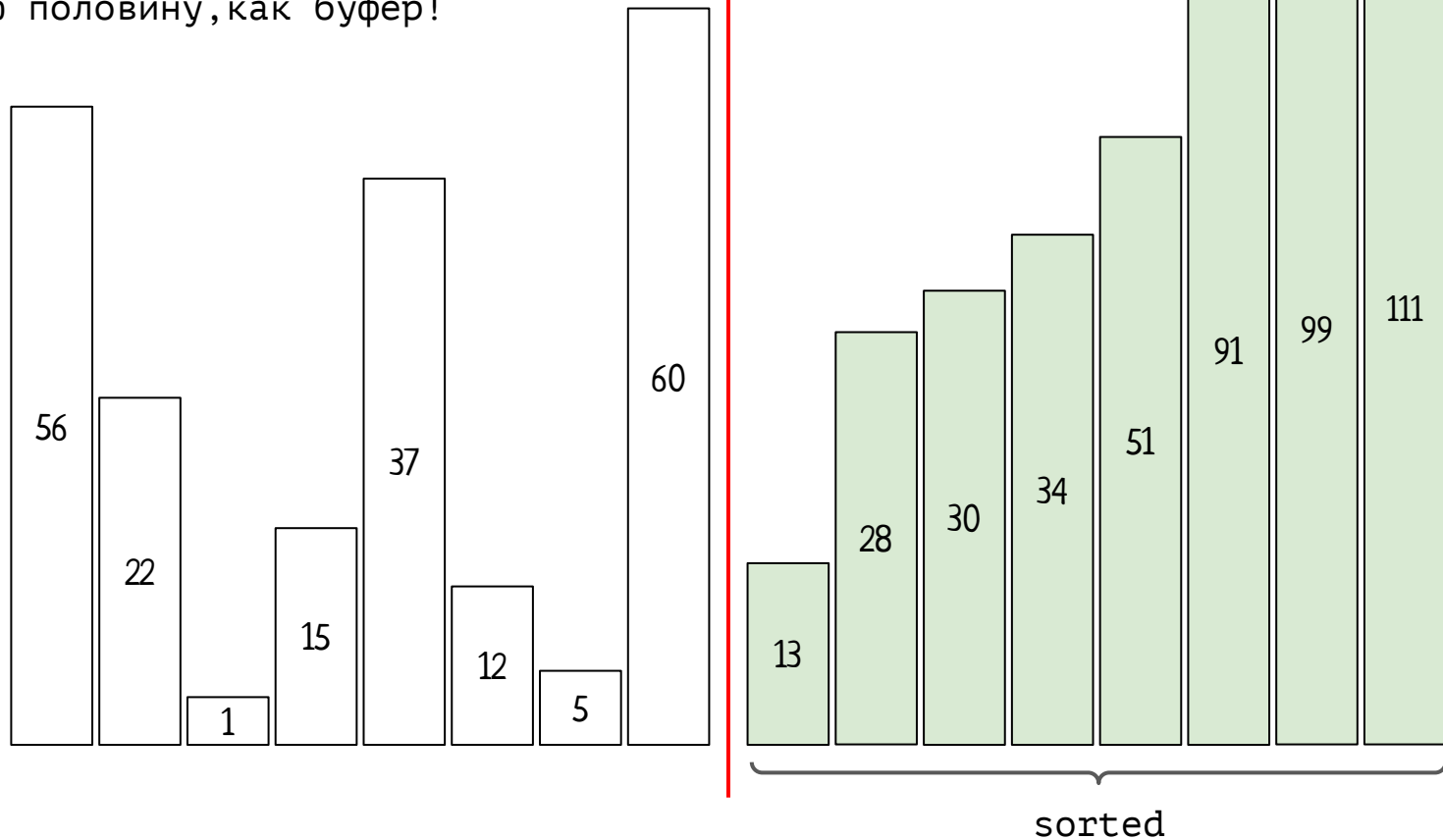
Как отсортировать  
левую половину  
массива?



Давайте  
просто  
использовать  
правую, как  
буфер!

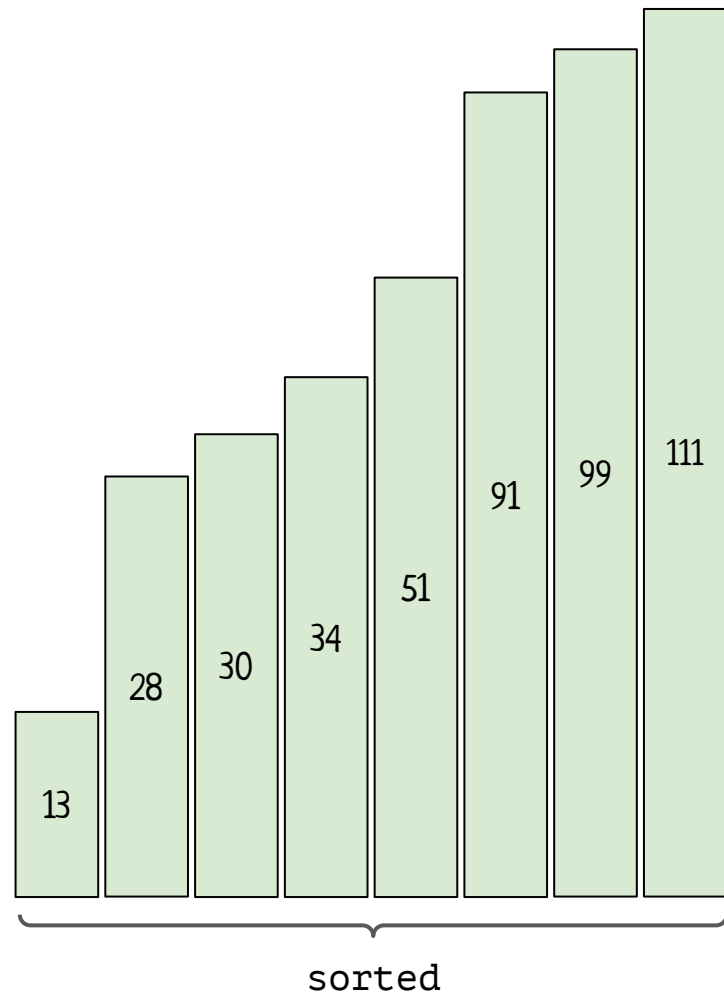
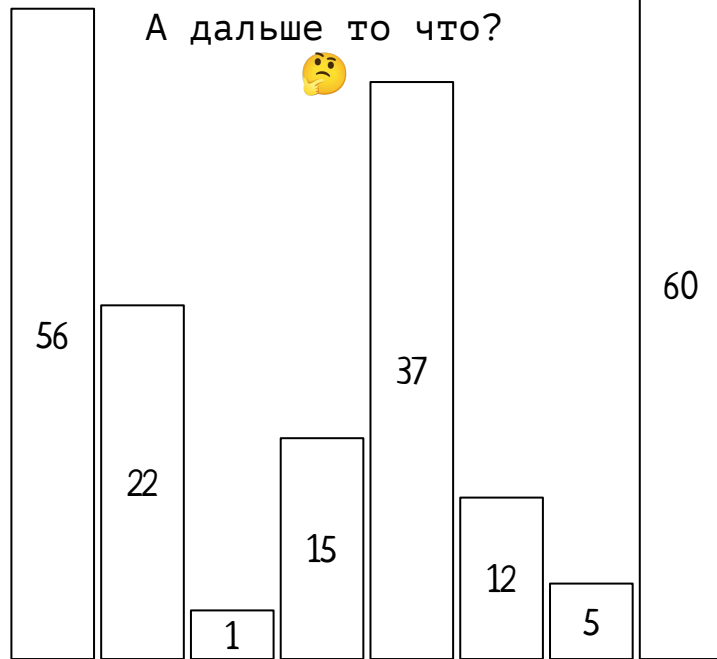


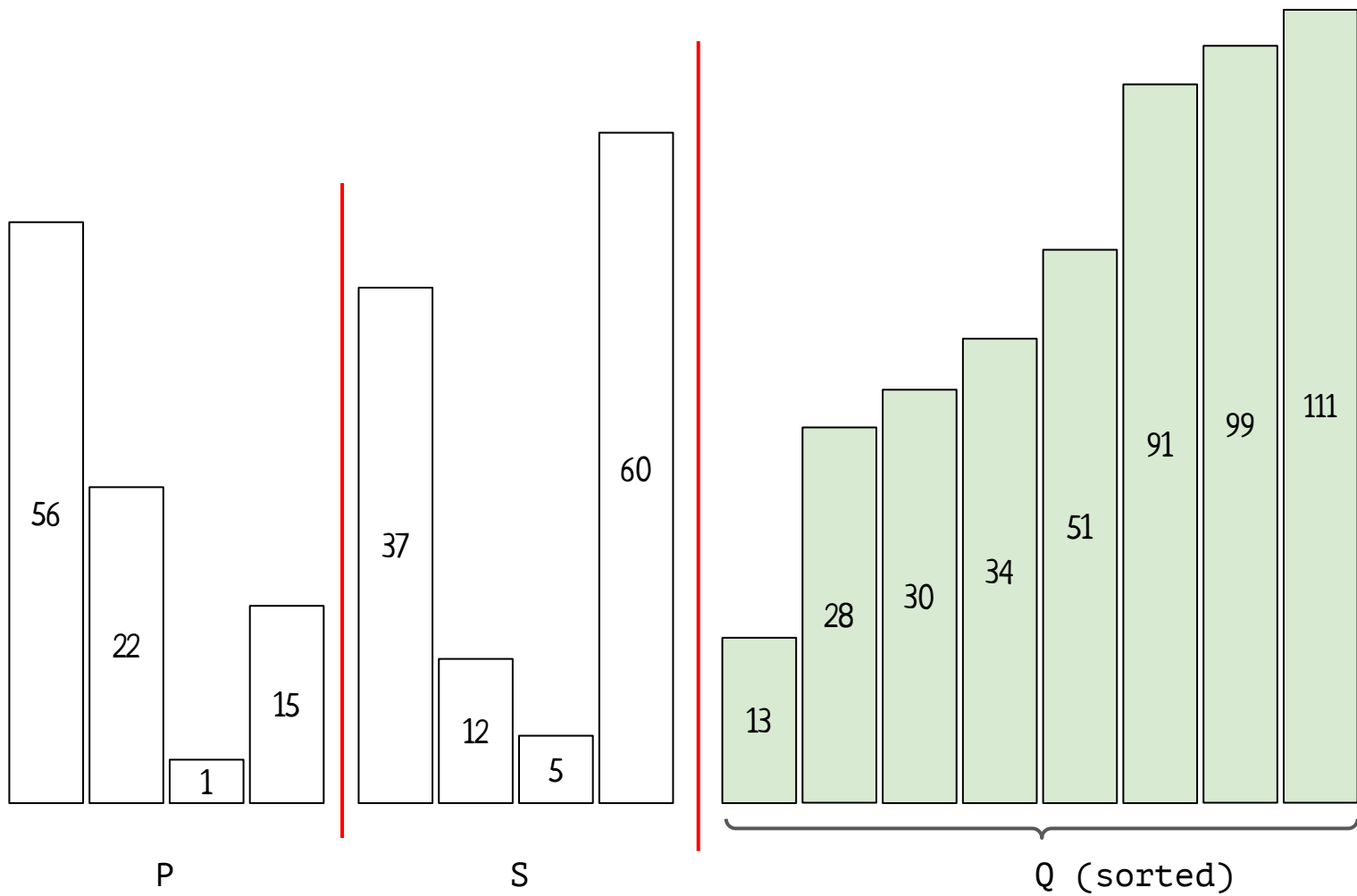
Т.е. первую половину массива  
можно отсортировать обычной  
сортировкой слияния, используя  
вторую половину, как буфер!



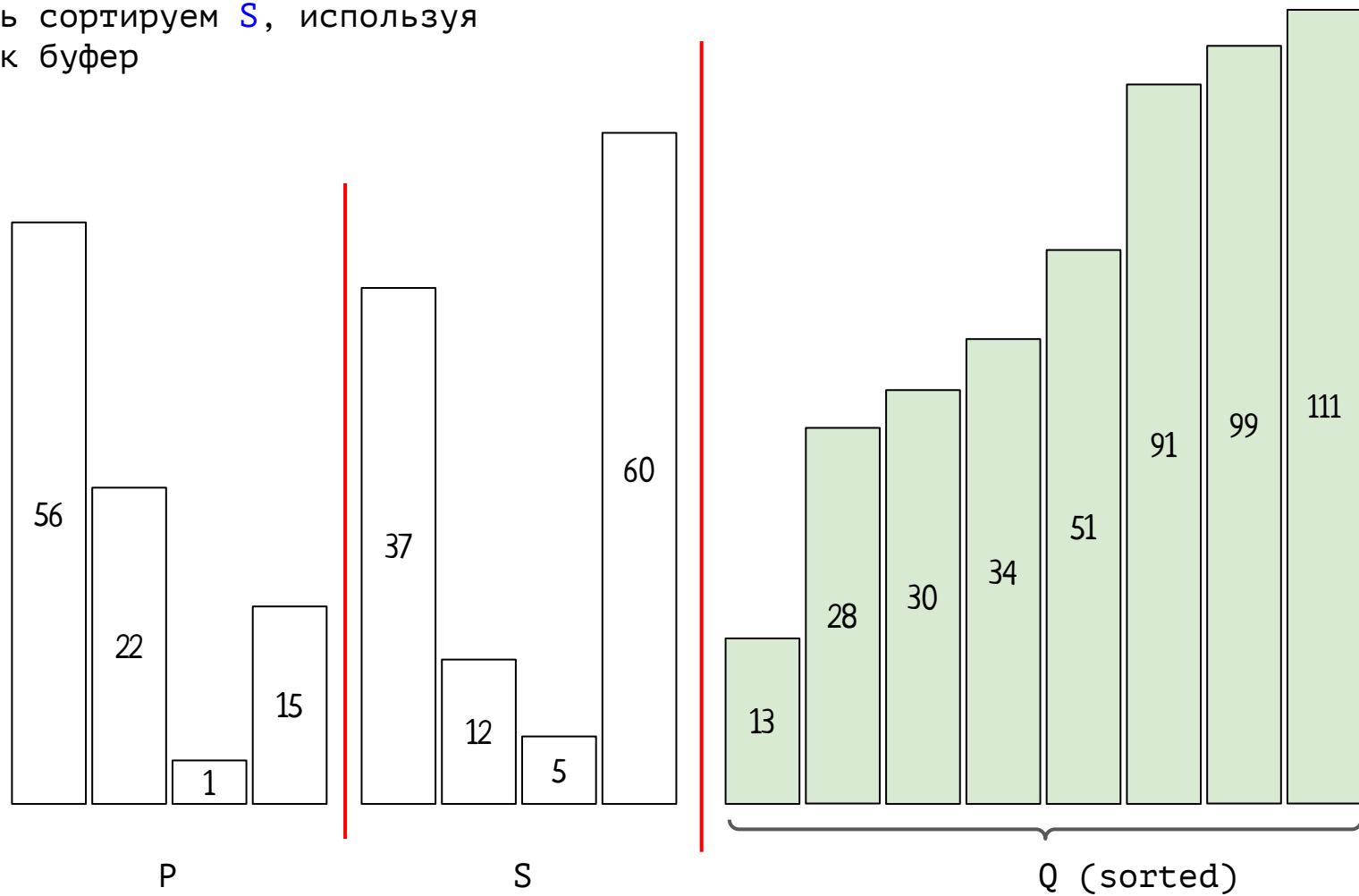


Т.е. первую половину массива  
можно отсортировать обычной  
сортировкой слияния, используя  
вторую половину, как буфер!

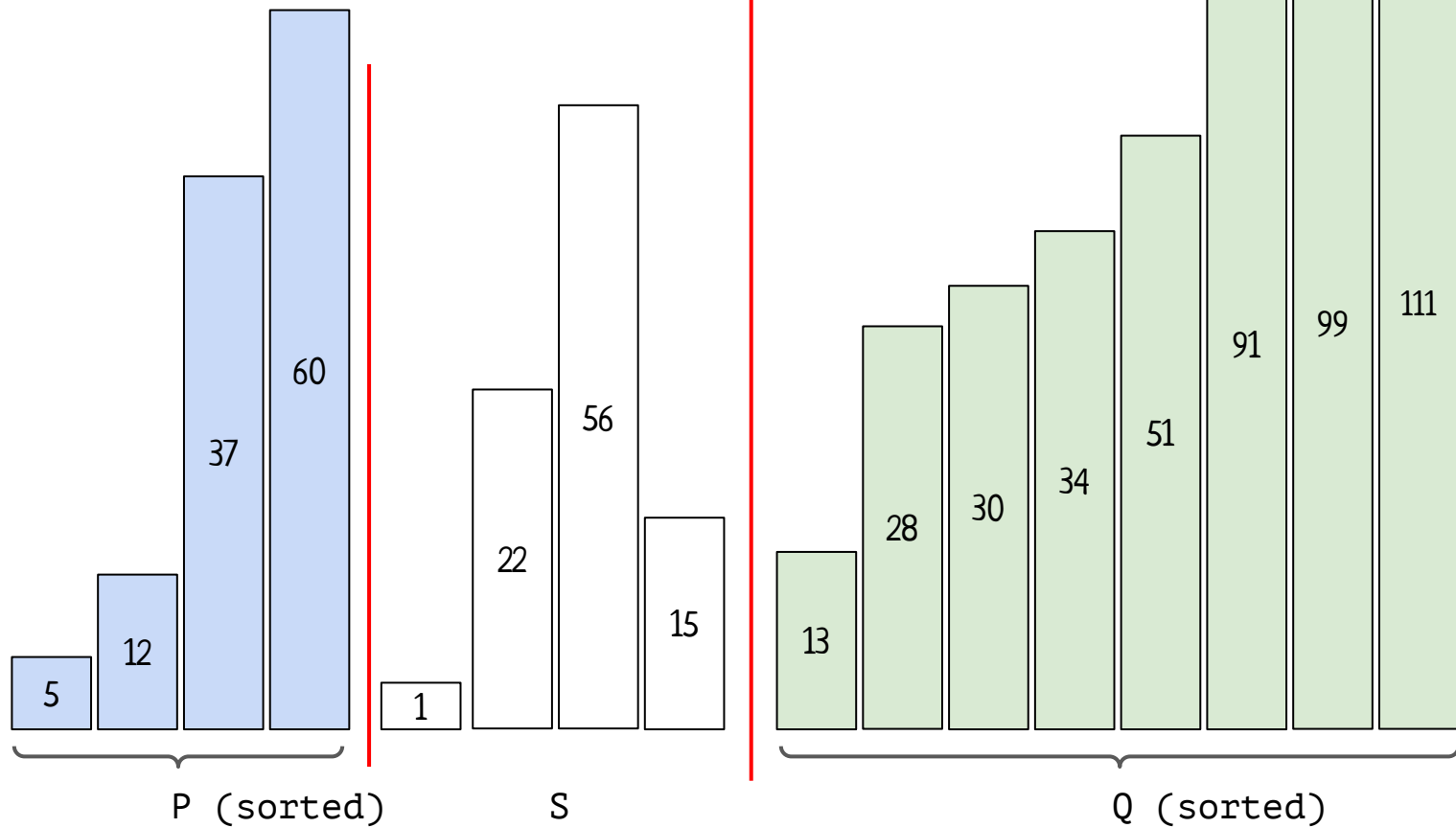




Теперь сортируем **S**, используя  
P, как буфер

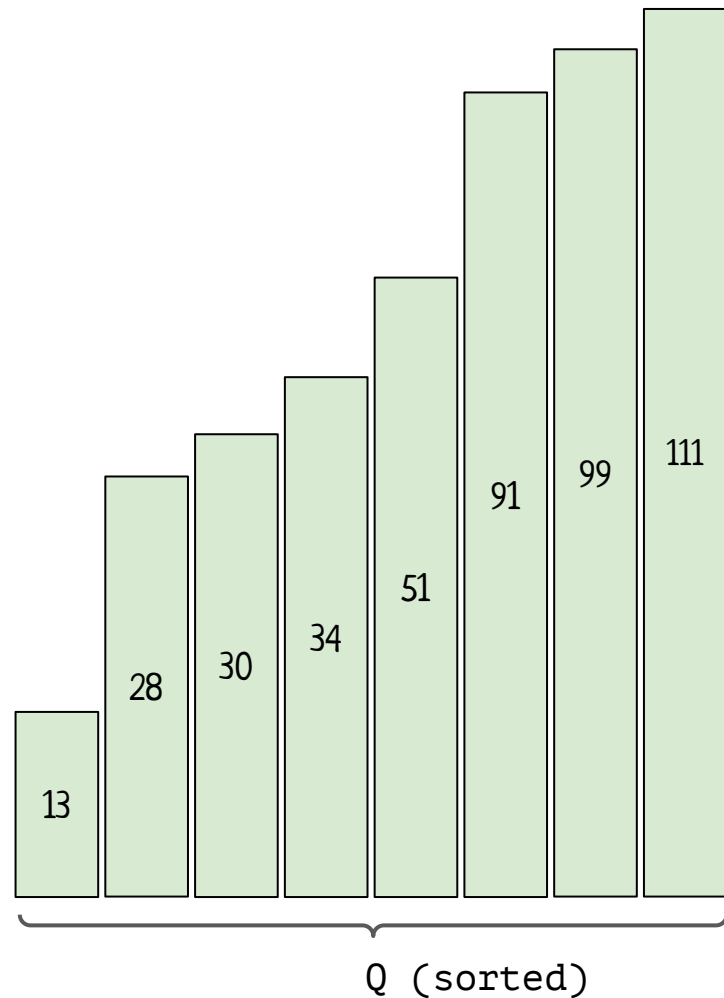
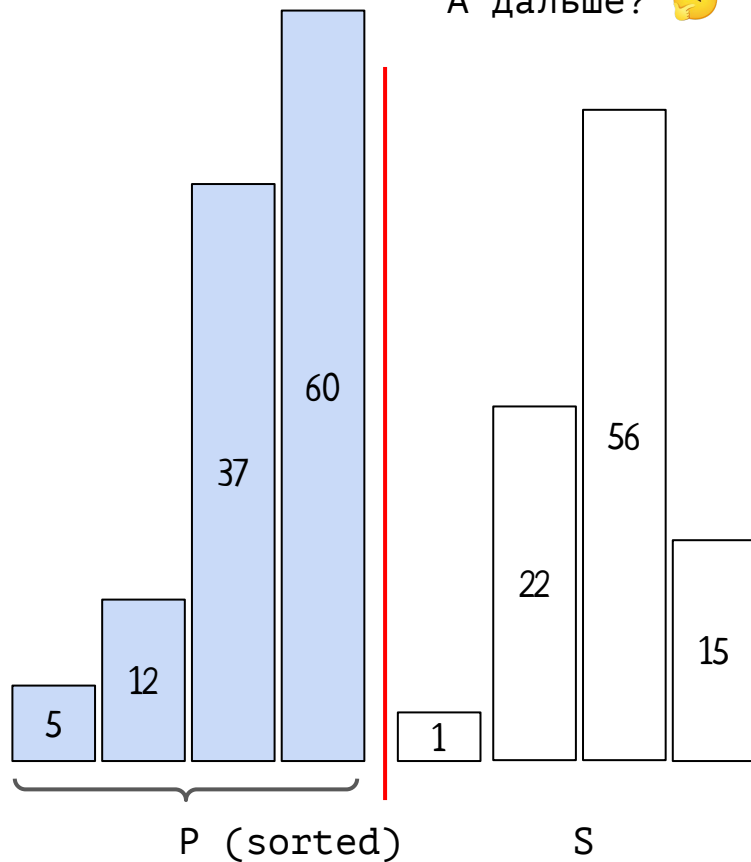


Теперь сортируем S, используя  
P, как буфер



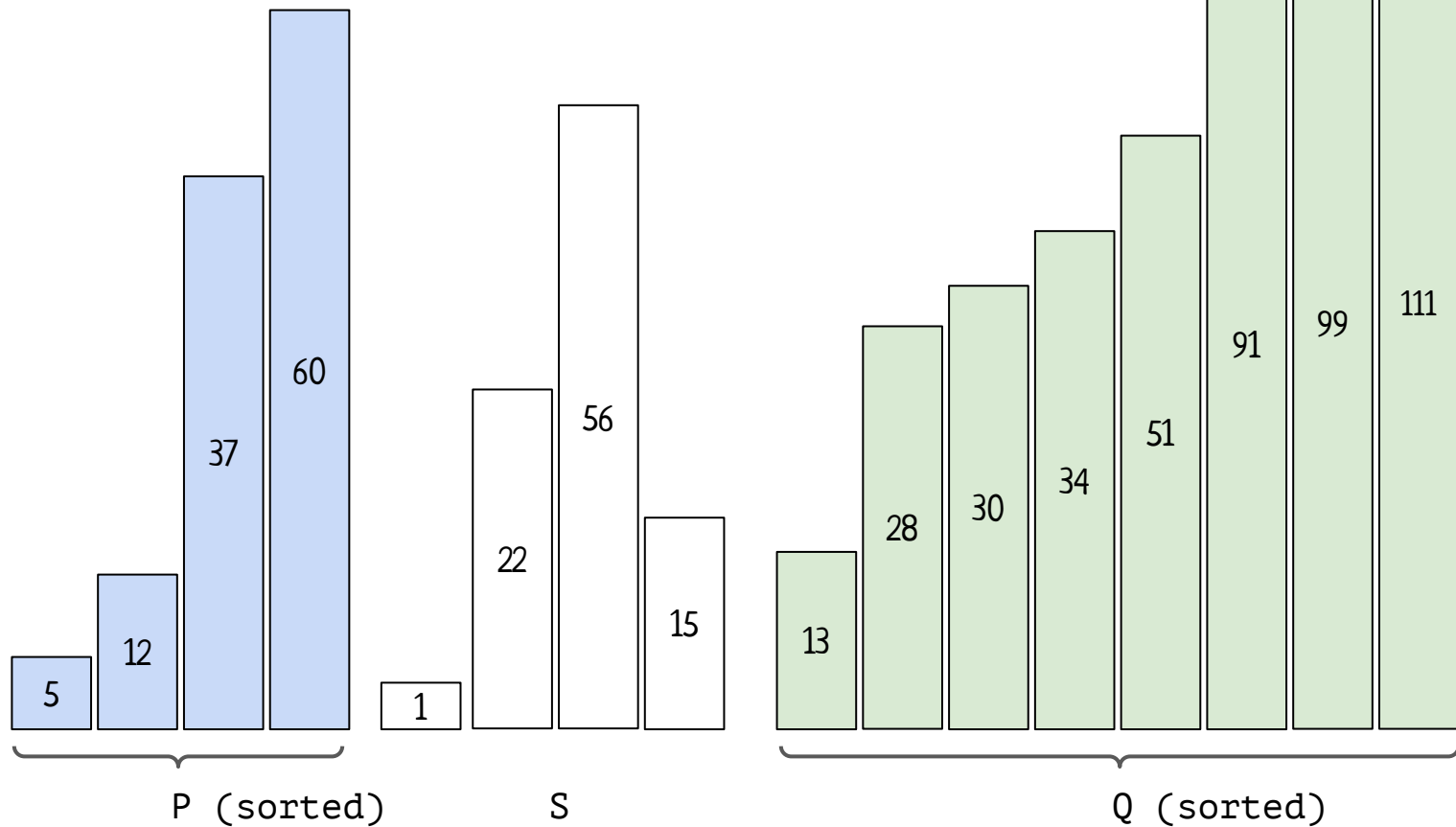
Теперь сортируем **S**, используя  
P, как буфер

А дальше? 🤔

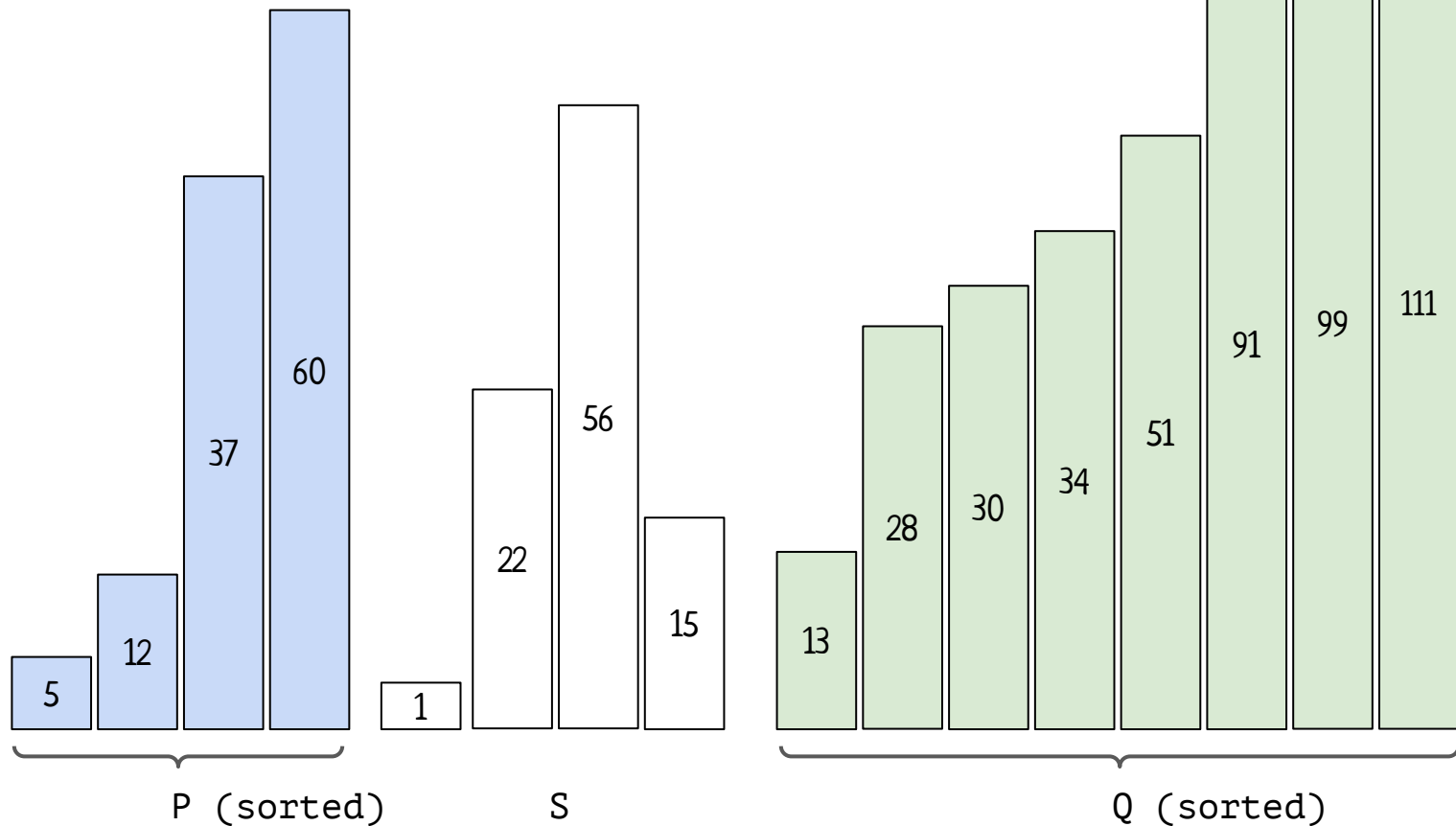


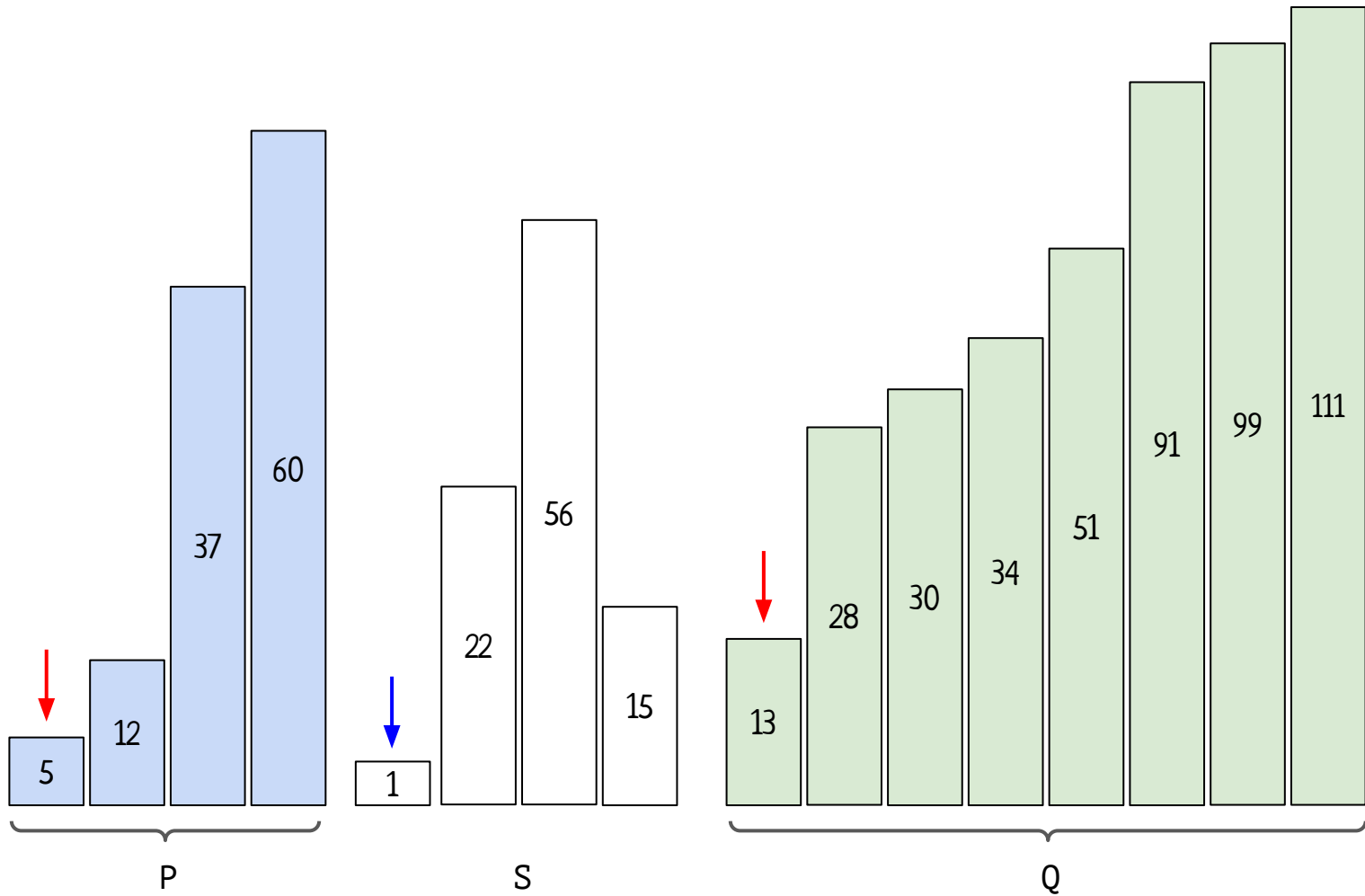
Теперь сортируем **S**, используя **P** и **Q**,  
**P**, как буфер

Дальше нужно сжать **P** и **Q**,  
используя в качестве буфера...

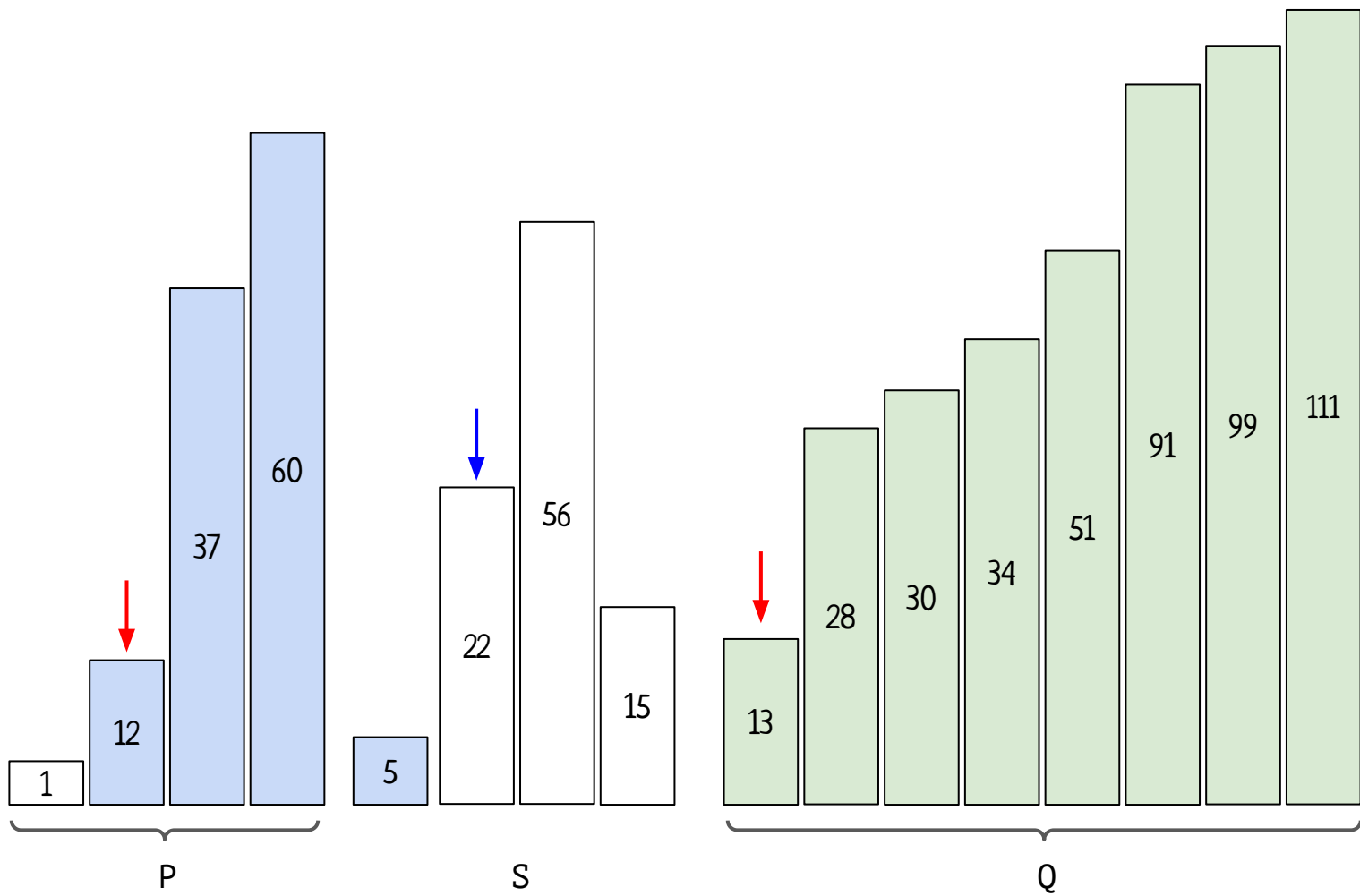


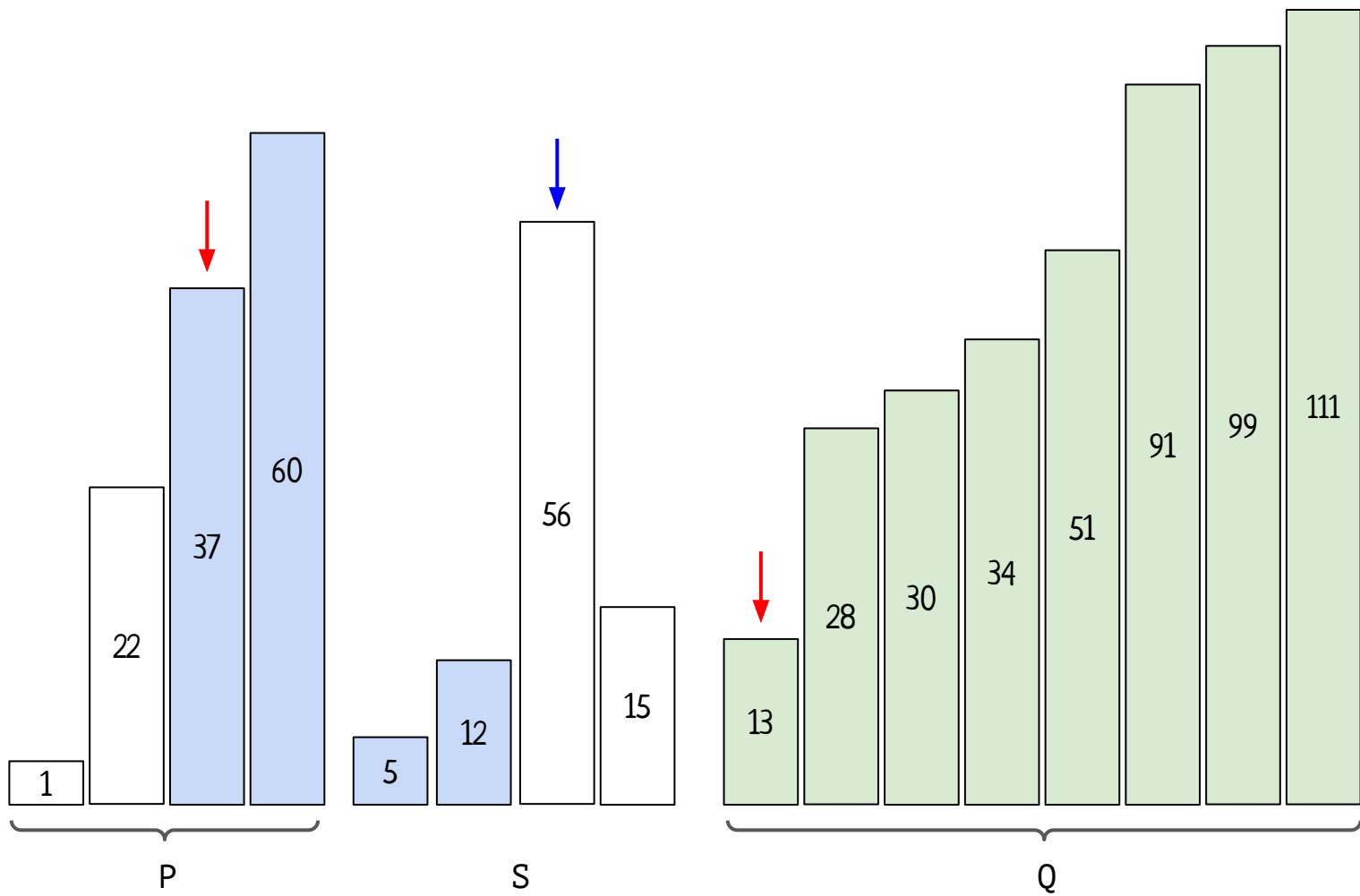
Теперь сортируем **S**, используя **P** и **Q**,  
**P**, как буфер  
Дальше нужно сжать **P** и **Q**,  
используя в качестве буфера...  
**S + Q!**

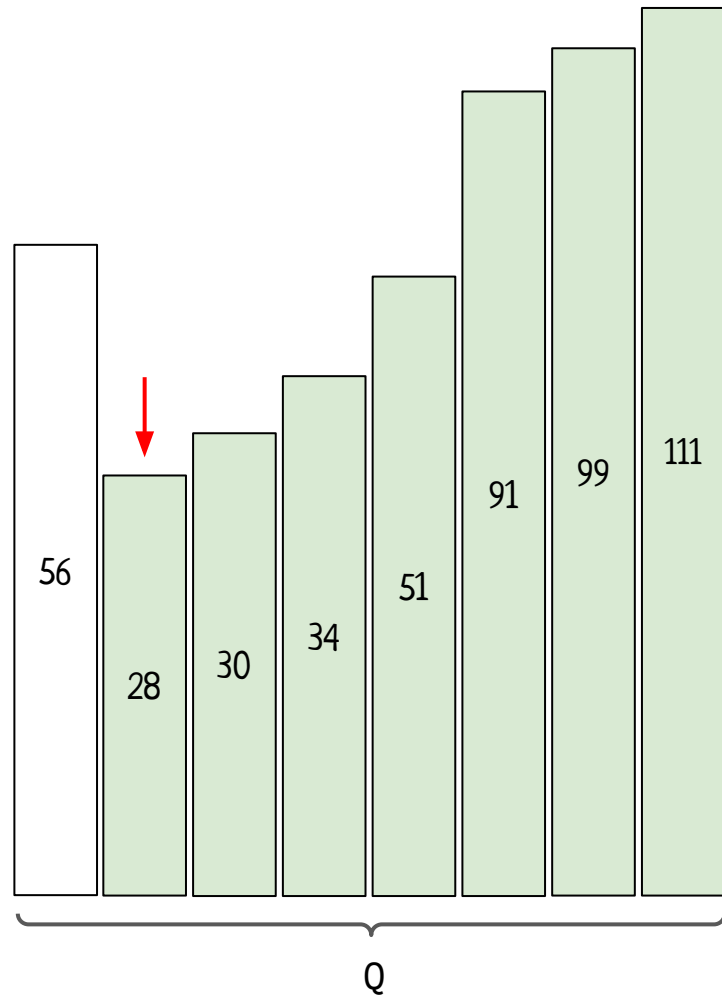
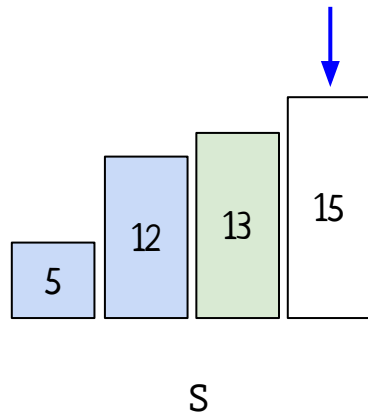
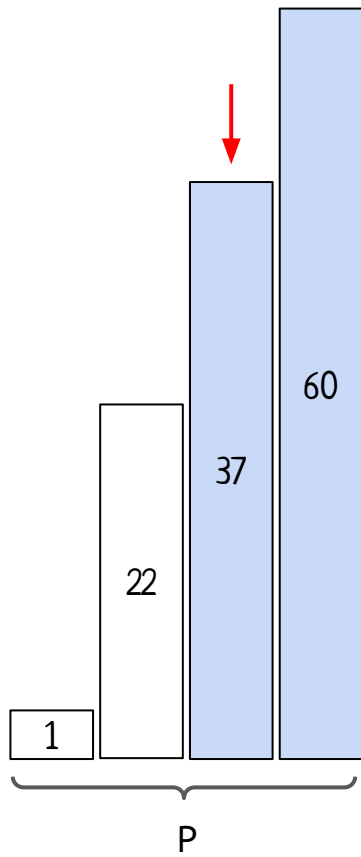


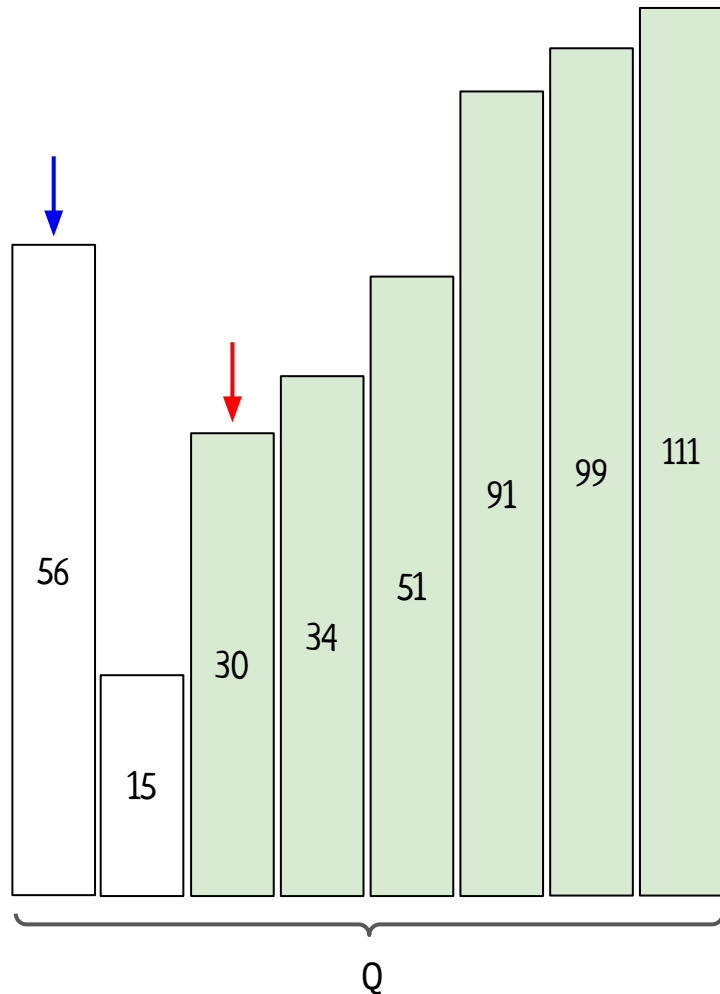
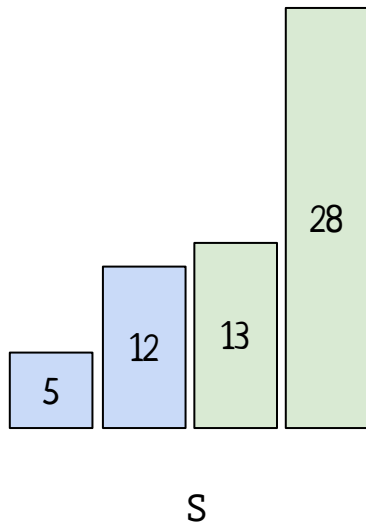
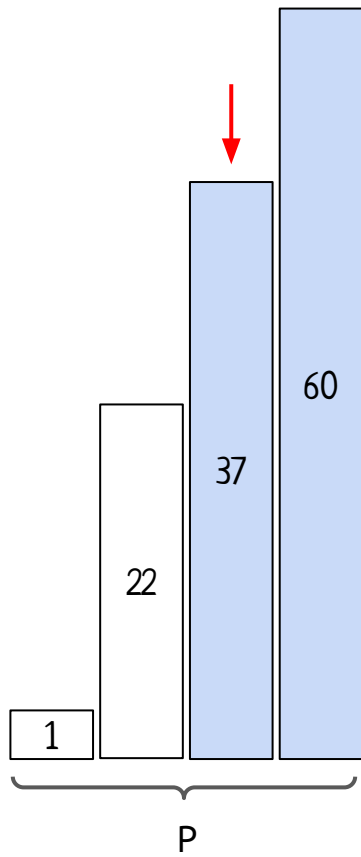


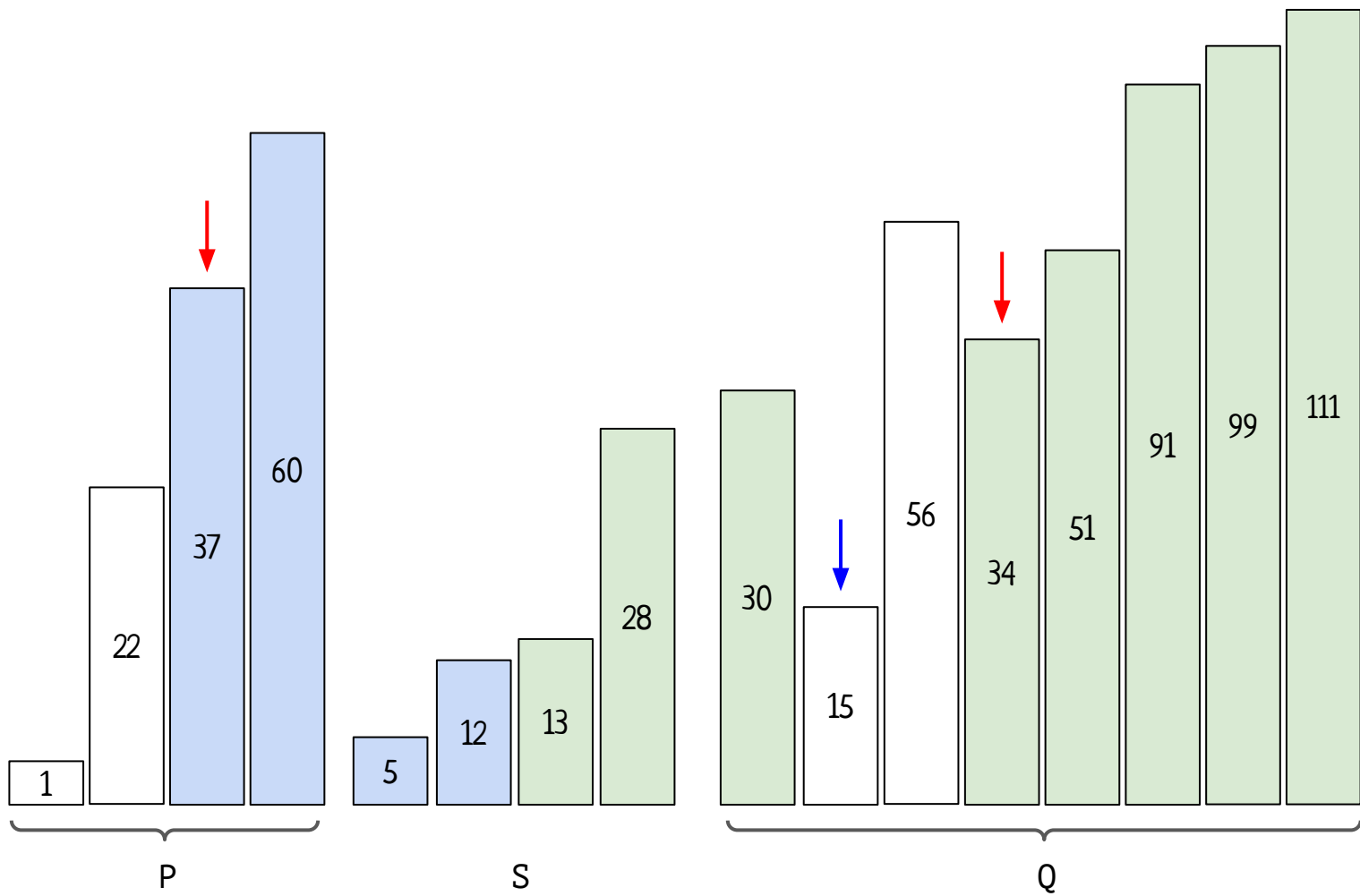


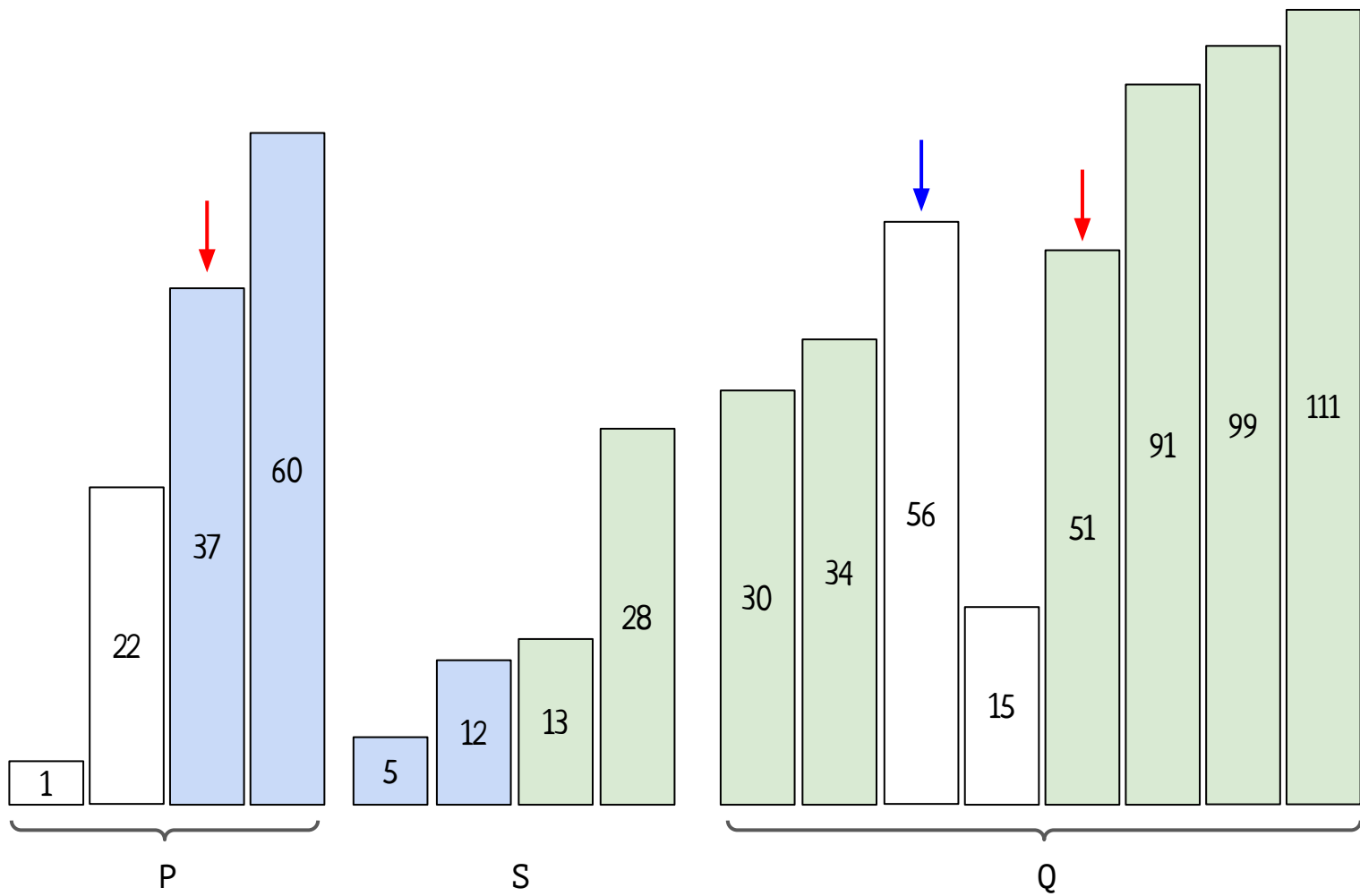


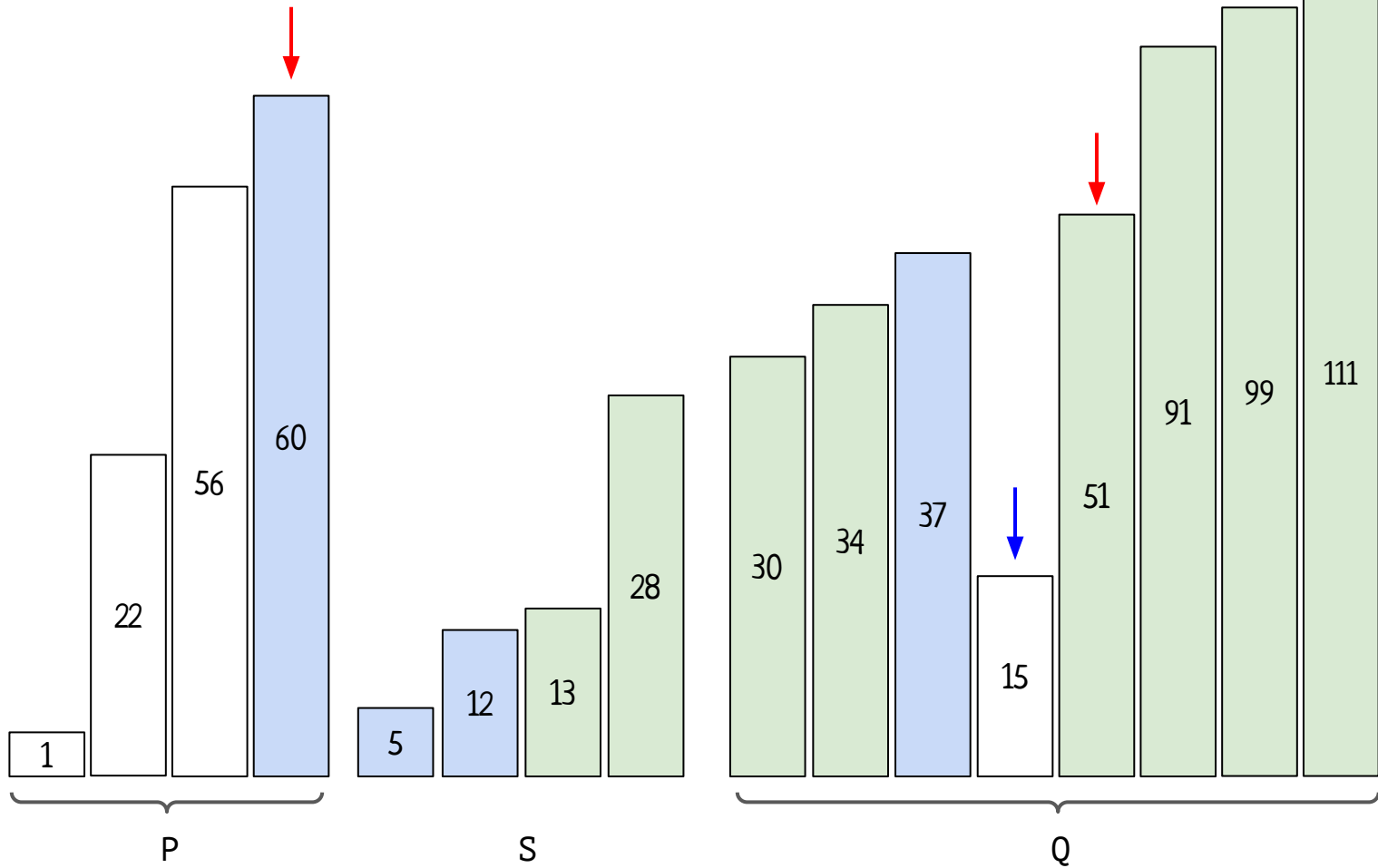


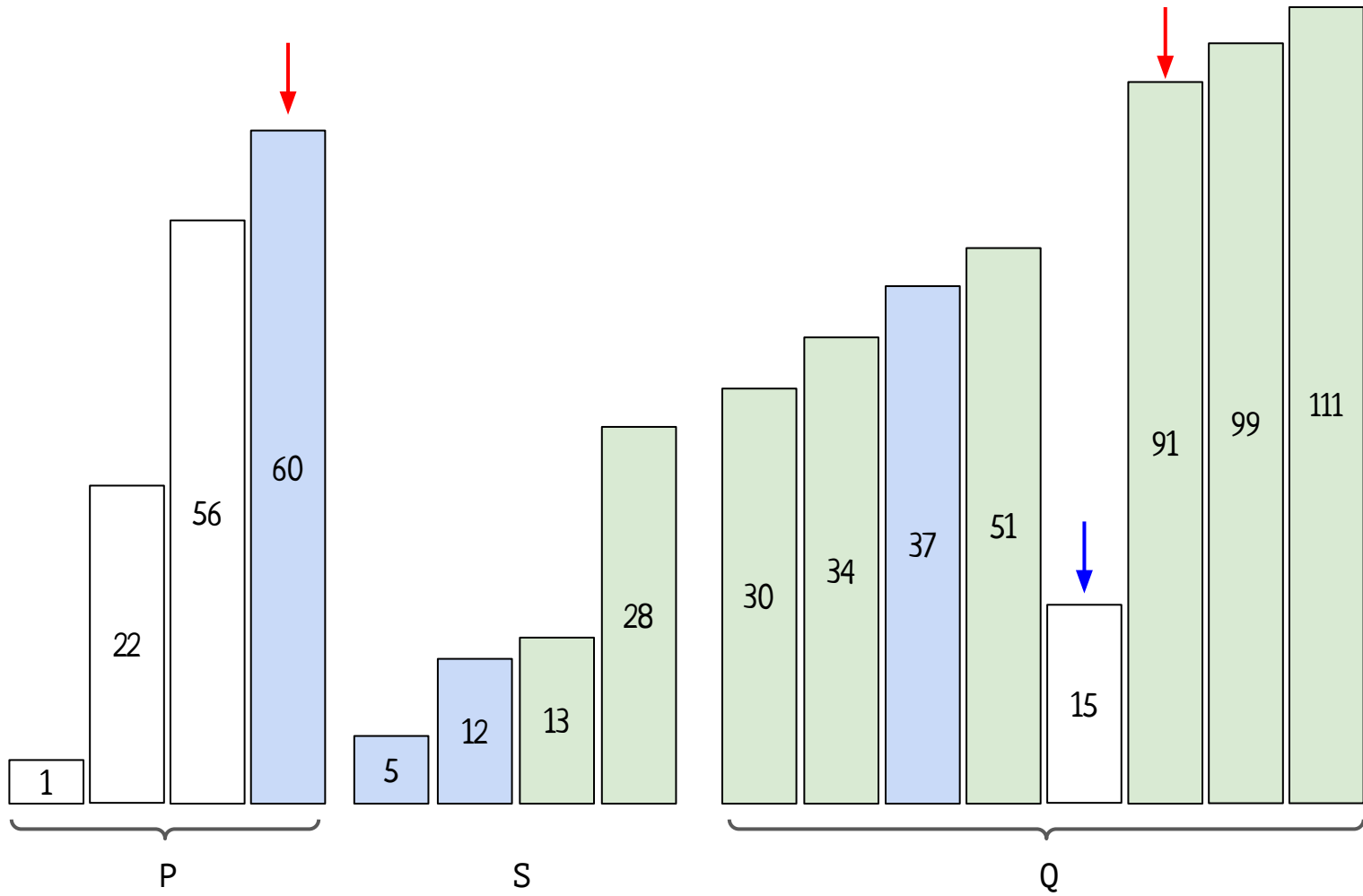




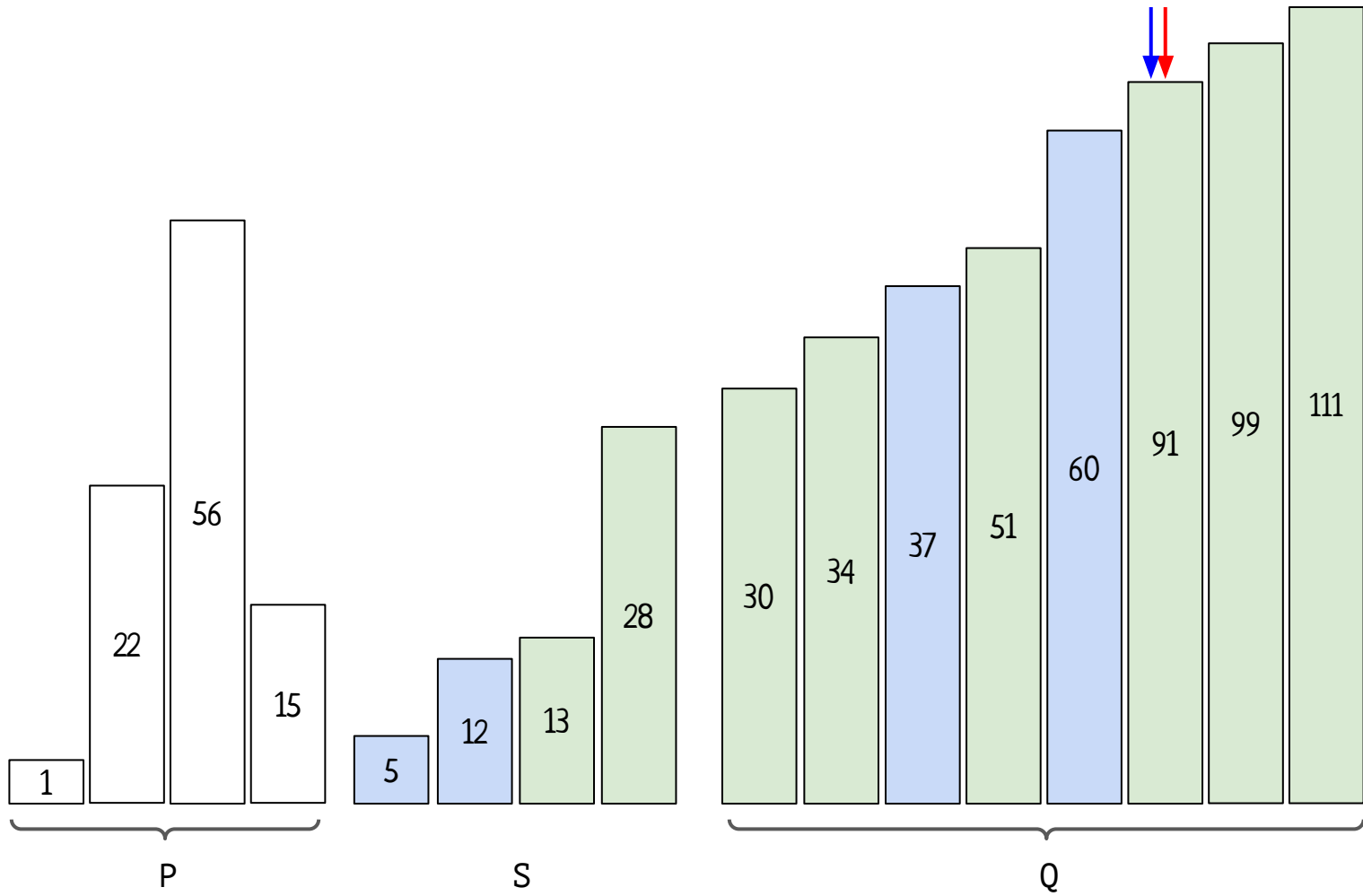


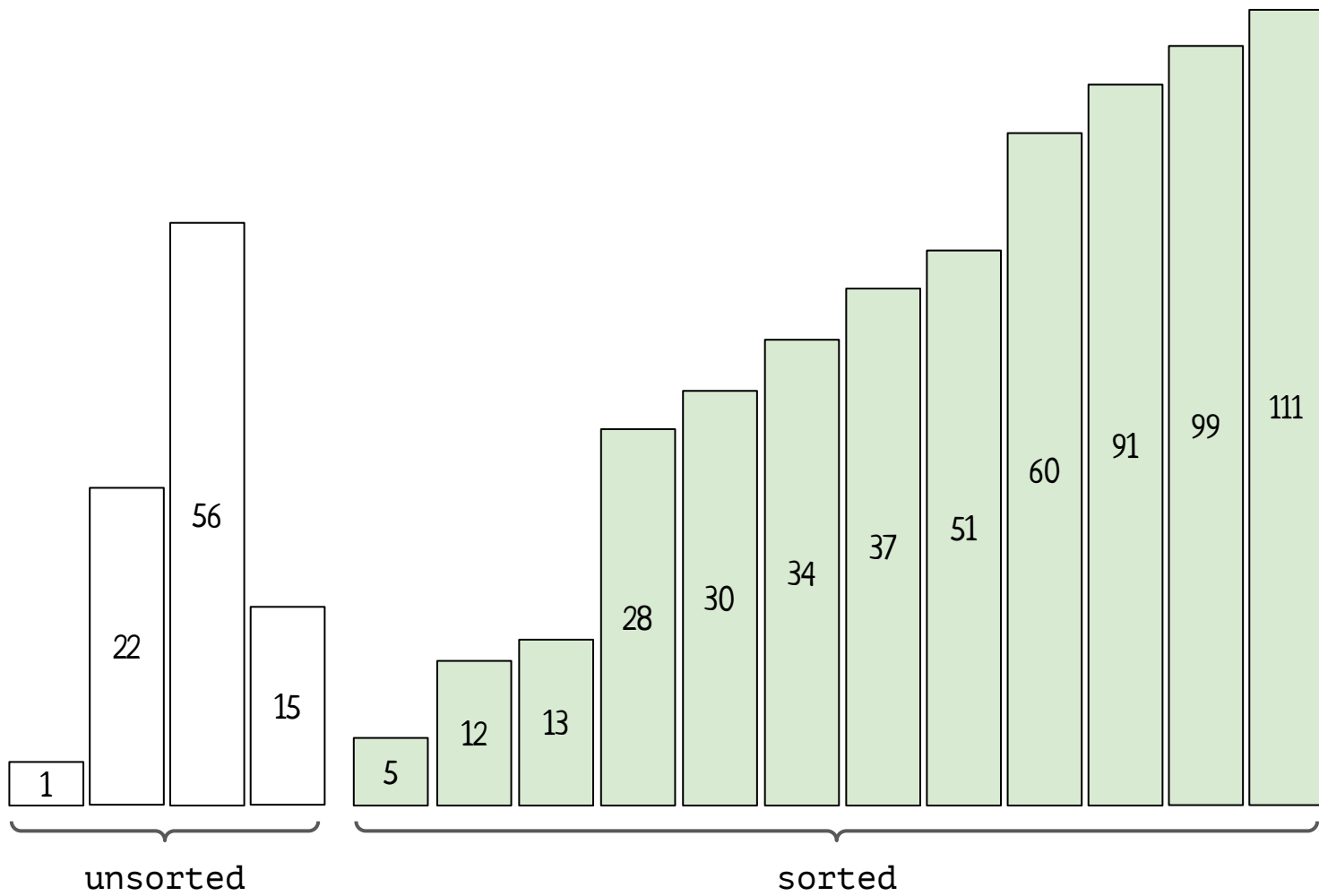






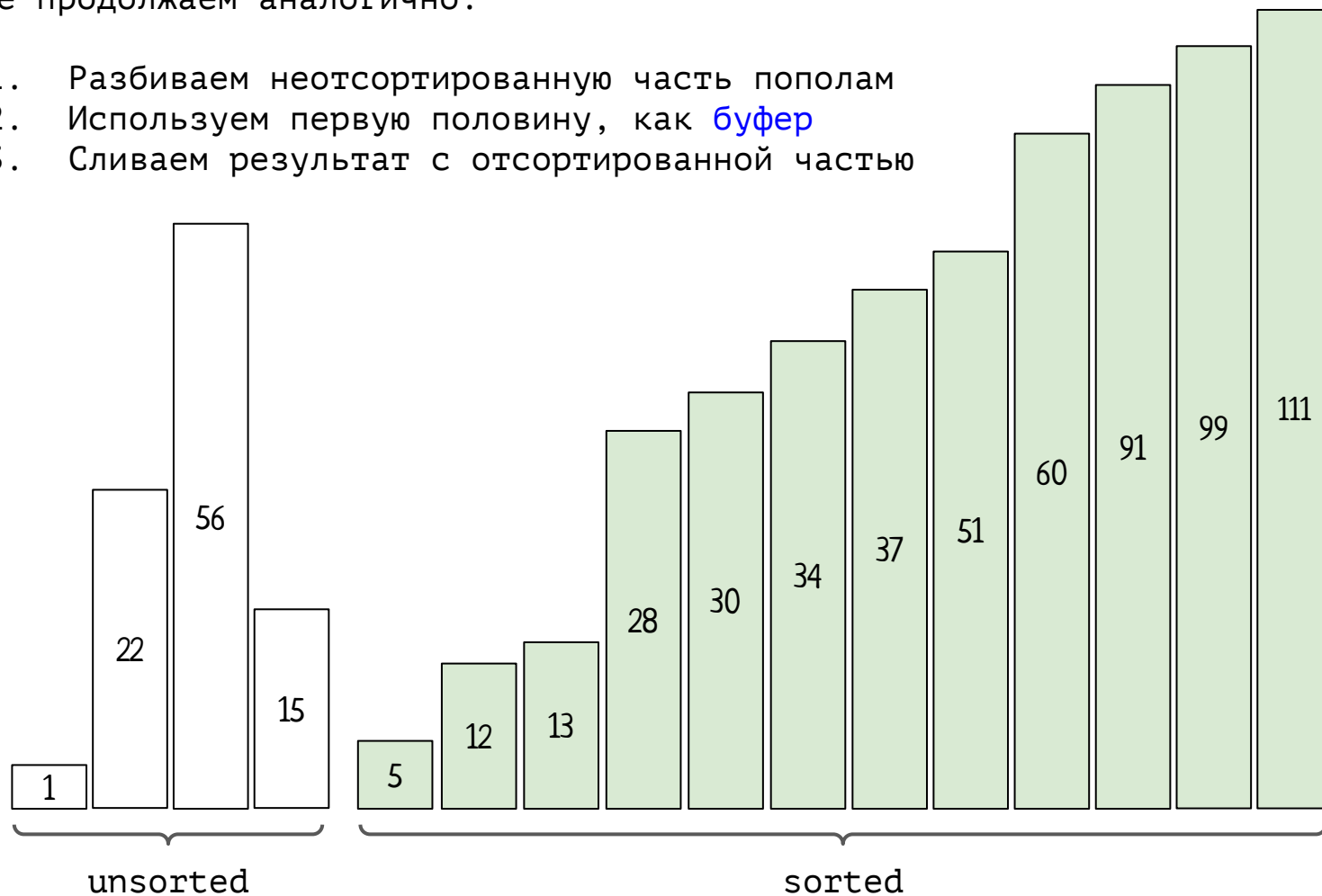






Дальше продолжаем аналогично:

1. Разбиваем неотсортированную часть пополам
2. Используем первую половину, как **буфер**
3. Сливаем результат с отсортированной частью



# In-place merge sort

**Упражнение:** доказать корректность описанного алгоритма сортировки слиянием без дополнительной памяти.

Главное доказать, что элементы из  $Q$  не выходят за пределы рабочей области  $S + Q$

# In-place merge sort

Временная сложность:  $O(N * \log N)$

# In-place merge sort

Временная сложность:  $O(N * \log N)$

Но сравнений и перемещений элементов  
**больше**, чем в обычной сортировке слиянием

# In-place merge sort

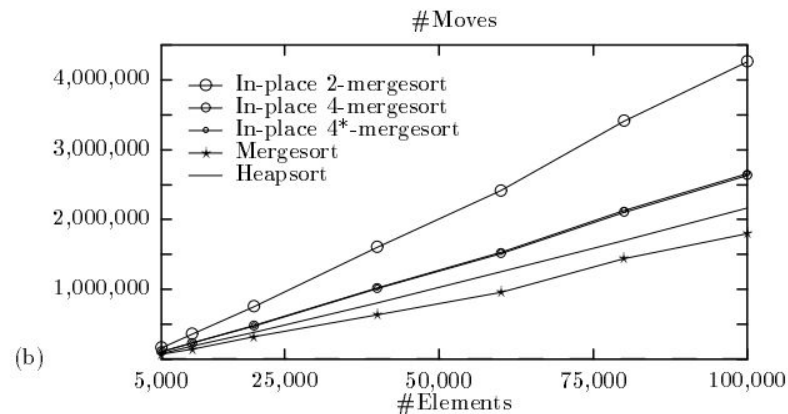
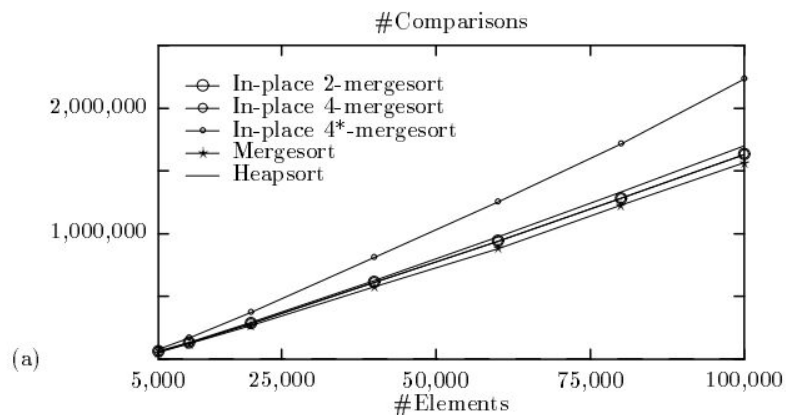
Временная сложность:  $O(N * \log N)$

Но сравнений и перемещений элементов **больше**, чем в обычной сортировке слиянием

При этом есть разные стратегии и оптимизации, повышающие **производительность**

# In-place merge sort

Временная сложность:  $O(N * \log N)$





# In-place merge sort

Временная сложность:  $O(N * \log N)$

Ёмкостная сложность:  $O(1)$

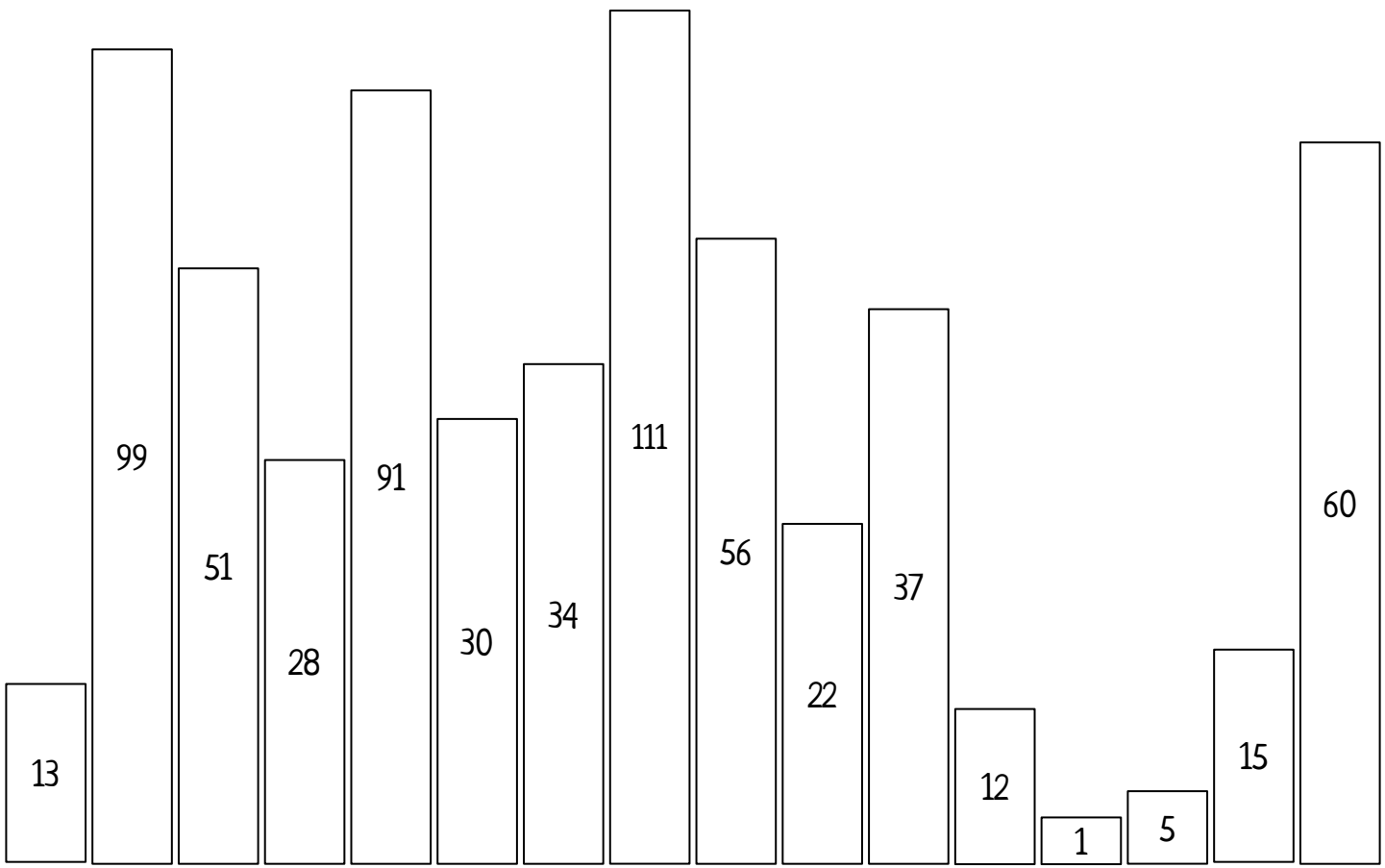


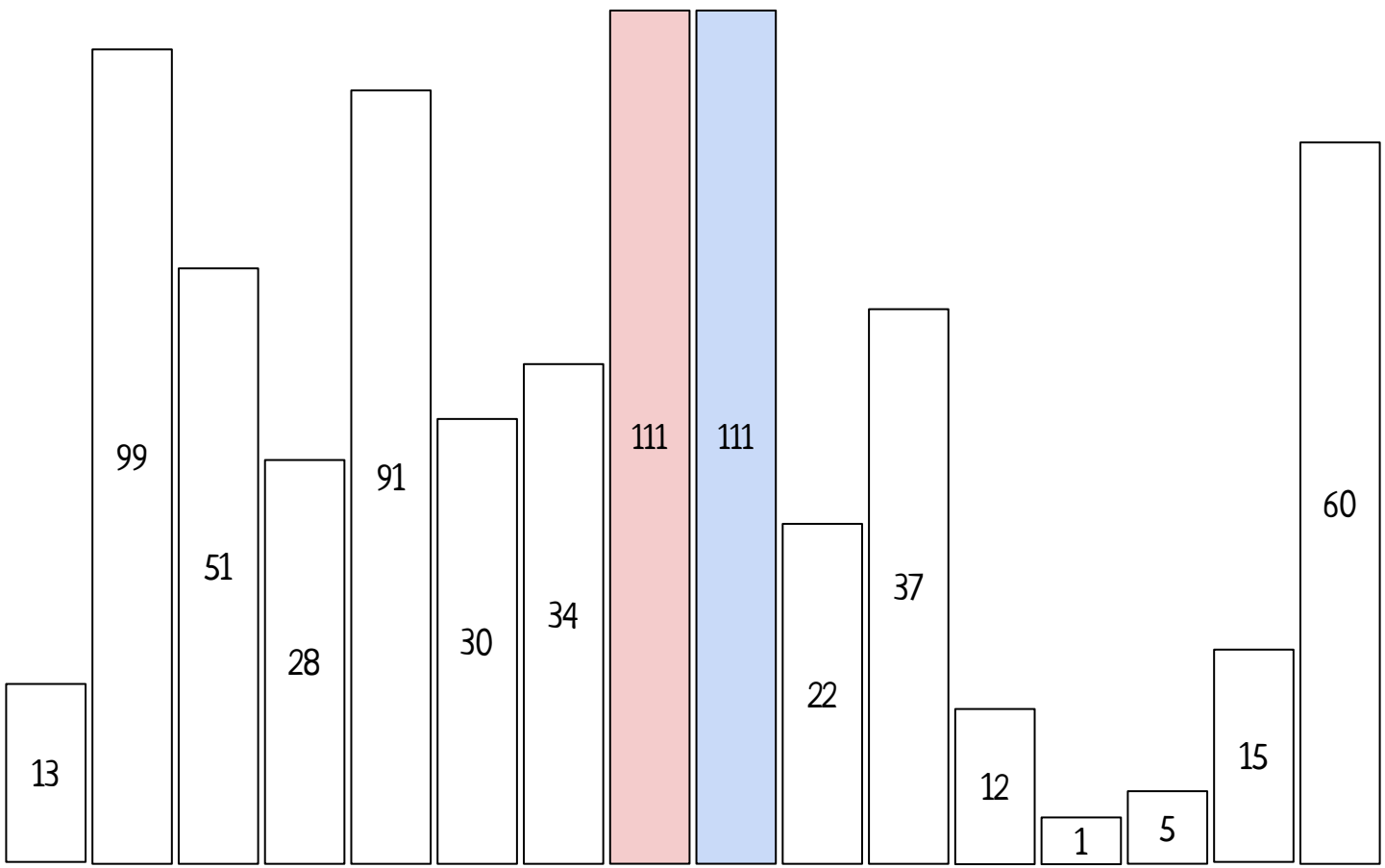
# In-place merge sort

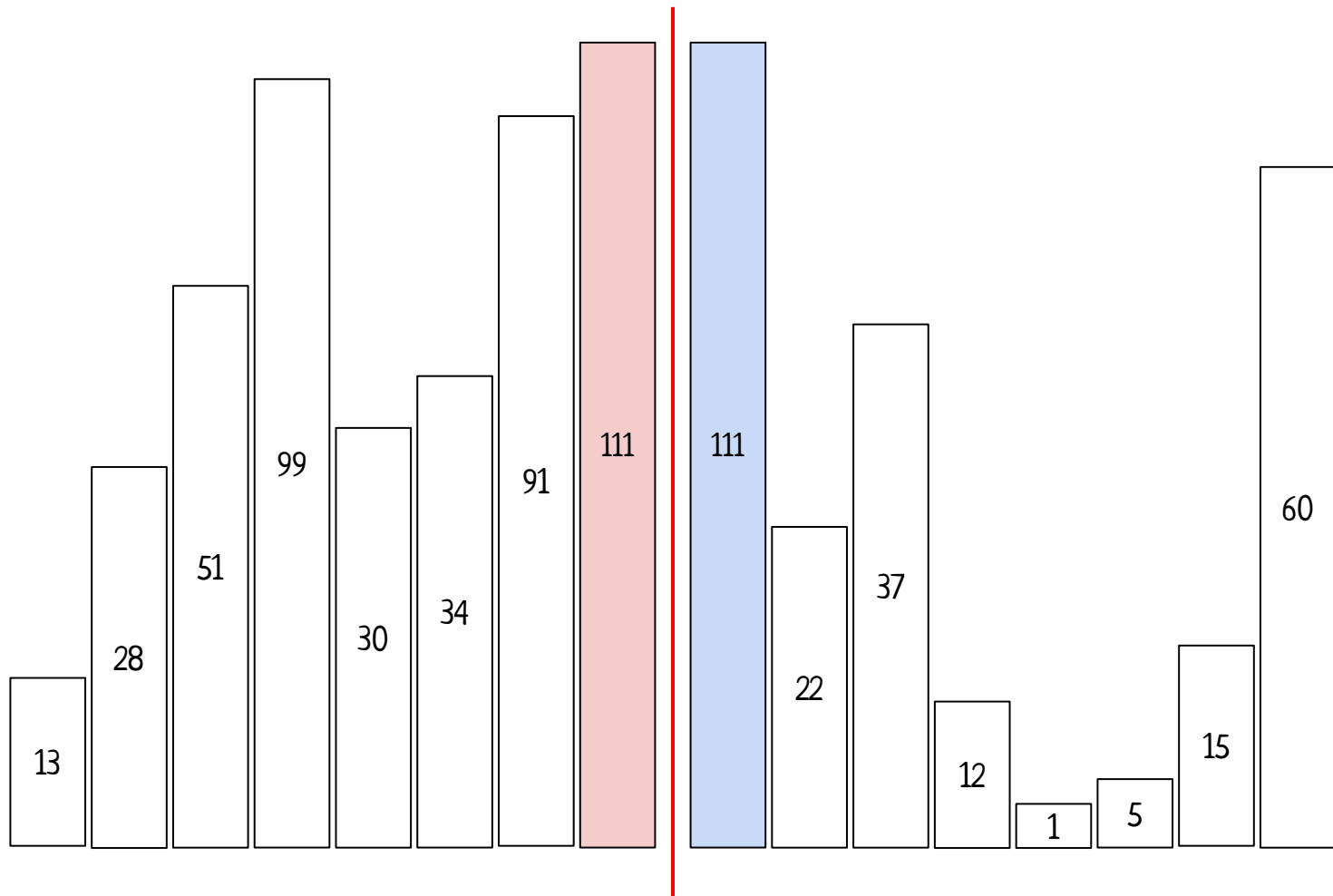
Временная сложность:  $O(N * \log N)$

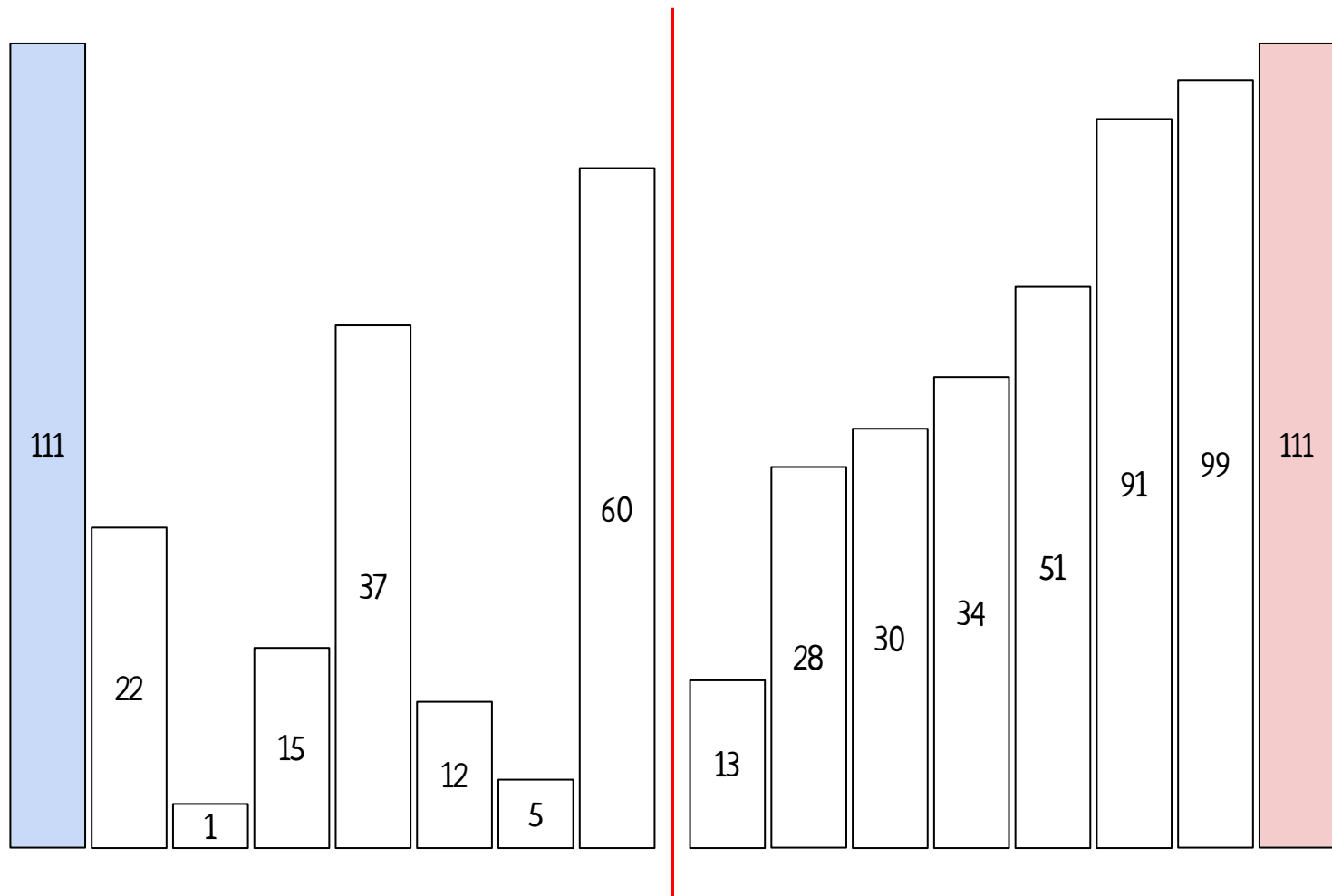
Ёмкостная сложность:  $O(1)$

Стабильность?









# In-place merge sort

Временная сложность:  $O(N * \log N)$

Ёмкостная сложность:  $O(1)$

Стабильность: **нет**

# In-place merge sort

Временная сложность:  $O(N * \log N)$

Ёмкостная сложность:  $O(1)$

Стабильность: **нет**

Paper:

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.8523&rep=rep1&type=pdf>



# In-place merge sort

Временная сложность:  $O(N * \log N)$

Ёмкостная сложность:  $O(1)$

Стабильность: **нет**

Папер (или в вашей папке на гугл диске):

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.8523&rep=rep1&type=pdf>

## Мини-задача #7 (3 балла)

Реализовать сортировку слиянием, используя константное количество дополнительной памяти.

Проверить решение необходимо на литкоде:

<https://leetcode.com/problems/sort-an-array>

Там же сравнить потребление памяти со стандартной реализацией.

**Дополнительно:** реализовать оптимизированную in-place сортировку слиянием из статьи.

# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.



# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.

Общая схема таких алгоритмов:

1. Разделяем большую задачу на подзадачи



# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.

Общая схема таких алгоритмов:

1. Разделяем большую задачу на подзадачи
2. Решить каждую подзадачу **рекурсивно**



# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.

Общая схема таких алгоритмов:

1. Разделяем большую задачу на подзадачи
2. Решить каждую подзадачу **рекурсивно**
3. **Соединить** решения подзадач в решение изначальной задачи



# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.

Общая схема таких алгоритмов:

1. Разделяем большую задачу на подзадачи
2. Решить каждую подзадачу **рекурсивно**
3. **Соединить** решения подзадач в решение изначальной задачи

Еще пример кроме merge sort?



# Divide and Conquer

Сортировка слиянием — классический пример "Divide and Conquer" алгоритма.

Общая схема таких алгоритмов:

1. Разделяем большую задачу на подзадачи
2. Решить каждую подзадачу **рекурсивно**
3. **Соединить** решения подзадач в решение изначальной задачи

Еще пример кроме merge sort? Карацуба! Еще..





# Количество инверсий

Пусть дан массив `array` из натуральных чисел от 0 до  $N-1$ , расположенных в произвольном порядке.

Найти количество **инверсий** в этом массиве.

# Количество инверсий

Пусть дан массив `array` из натуральных чисел от 0 до  $N-1$ , расположенных в произвольном порядке.

Найти количество **инверсий** в этом массиве.

$(i, j)$  - **инверсия**, если  $i < j$ , но  $array[i] > array[j]$

# Количество инверсий

0	2	4	1	3	5
---	---	---	---	---	---

Инверсии?

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии: (1, 3)

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии: (1, 3)  
[2, 1]

значения  
элементов из  
инверсий

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$   
 $[2, 1]$ ,  $[4, 3]$

значения  
элементов из  
инверсий

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

значения  
элементов из  
инверсий

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

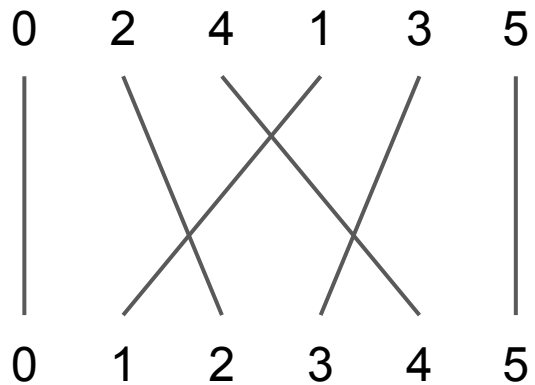


# Количество инверсий

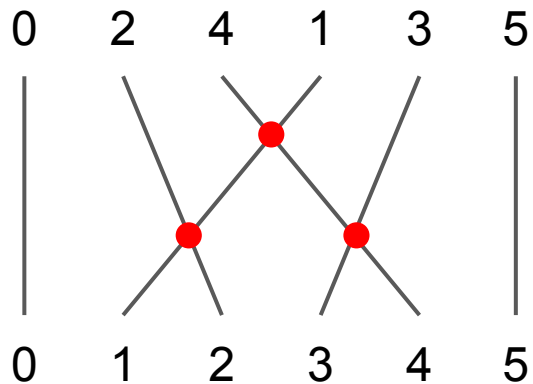
0 2 4 1 3 5

0 1 2 3 4 5

# Количество инверсий



# Количество инверсий



Геометрическая реинтерпретация -  
количество пересечений отрезков,  
выходящих из одних и тех же элементов

## Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Сколько может быть минимум инверсий?

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Сколько может быть минимум инверсий?

0 - в упорядоченном массиве

## Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Сколько может быть максимум инверсий?

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии: (1, 3), (2, 4), (2, 3)  
[2, 1], [4, 3], [4, 1]

Сколько может быть максимум инверсий?

$$\binom{N}{2} = \frac{N*(N-1)}{2}$$

## Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Сколько может быть максимум инверсий?

$$\binom{N}{2} = \frac{N*(N-1)}{2} \quad (\text{т.е. в нашем случае } 15)$$



# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Зачем их вообще считать?

## Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Зачем их вообще считать?

Хорошая метрика, чтобы оценивать  
"близость" рейтингов.

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Как будем их искать?

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

Как будем их искать?

**Brute force**: двойной цикл по массиву,  
ищем для каждого элемента его инверсии

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5



Можем ли  
мы лучше?

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

**Brute force**: двойной цикл по массиву,  
ищем для каждого элемента его инверсии

Сложность:  $O(N^2)$

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5



Можем ли  
мы лучше?

Инверсии: (1, 3), (2, 4), (2, 3)  
[2, 1], [4, 3], [4, 1]

Divide and  
Conquer!

**Brute force**: двойной цикл по массиву,  
ищем для каждого элемента его инверсии

Сложность:  $O(N^2)$

# Количество инверсий

Идея: будем говорить, что инверсия  $(i, j)$ :

1. Левая, когда  $i, j \leq N/2$
2. Правая, когда  $i, j > N/2$
3. ...

# Количество инверсий

Идея: будем говорить, что инверсия  $(i, j)$ :

1. Левая, когда  $i, j \leq N/2$
2. Правая, когда  $i, j > N/2$
3. Разделенная, когда  $i \leq N/2 < j$



# Количество инверсий

**Идея:** будем говорить, что инверсия  $(i, j)$ :

1. Левая, когда  $i, j \leq N/2$
2. Правая, когда  $i, j > N/2$
3. Разделенная, когда  $i \leq N/2 < j$

Левые и правые инверсии можно найти **рекурсивно**, вот и почва для D&C

```
def count_inversions(array: int[]) -> int:
```

```
def count_inversions(array: int[]) -> int:  
    if len(array) == 1:  
        return 0
```

```
def count_inversions(array: int[]) -> int:
    if len(array) == 1:
        return 0

    middle = len(array) / 2
    left    = count_inversions(array[:middle])
    right   = count_inversions(array[middle:])
```

```
def count_inversions(array: int[]) -> int:
    if len(array) == 1:
        return 0

    middle = len(array) / 2
    left    = count_inversions(array[:middle])
    right   = count_inversions(array[middle:])

    split   = count_split_inversions(array)

    return left + right + split
```

```
def count_inversions(array: int[]) -> int:
    if len(array) == 1:
        return 0

    middle = len(array) / 2
    left    = count_inversions(array[:middle])
    right   = count_inversions(array[middle:])

    split   = count_split_inversions(array)

    return left + right + split
```

Есть идеи?

Если бы удалось реализовать `count_split_inversions` за  $O(N)$ , то получили бы суммарную сложность  $O(N \cdot \log N)$

# Количество инверсий

Идея: а что, если нам **буквально**  
переиспользовать сортировку слиянием?

"can I copy your homework?"

"yeah just change it up a bit so it  
doesn't look obvious you copied"

"ok"



```
def count_inversions(array: int[]) -> int:
    if len(array) == 1:
        return 0

    middle = len(array) / 2
    left    = count_inversions(array[:middle])
    right   = count_inversions(array[middle:])

    split   = count_split_inversions(array)

    return left + right + split
```



```
def sort_and_count_inversions(array: int[]) -> int:  
    if len(array) == 1: return 0  
  
    middle = len(array) / 2  
    left    = sort_and_count_inversions(array[:middle])  
    right   = sort_and_count_inversions(array[middle:])
```

```
def sort_and_count_inversions(array: int[]) -> int:
    if len(array) == 1: return 0

    middle = len(array) / 2
    left    = sort_and_count_inversions(array[:middle])
    right   = sort_and_count_inversions(array[middle:])

    split = merge_and_count_split(array[:middle],
                                   array[middle:],
                                   buffer)

    return left + right + split
```

```
def sort_and_count_inversions(array: int[]) -> int:
    if len(array) == 1: return 0

    middle = len(array) / 2
    left    = sort_and_count_inversions(array[:middle])
    right   = sort_and_count_inversions(array[middle:])

    buffer = new int[len(array)]
    split = merge_and_count_split(array[:middle],
                                   array[middle:],
                                   buffer)

    for i in [0; len(array)): array[i] = buffer[i]

    return left + right + split
```

```
def merge(left, right, res: int[]):
    lsize, rsize = len(left), len(right)
    n = len(res)
    assert n == lsize + rsize
    i, j, k = 0, 0, 0
    while k < n and i < lsize and j < rsize:
        if left[i] < right[j]:
            res[k++] = left[i++]
        else:
            res[k++] = right[j++]

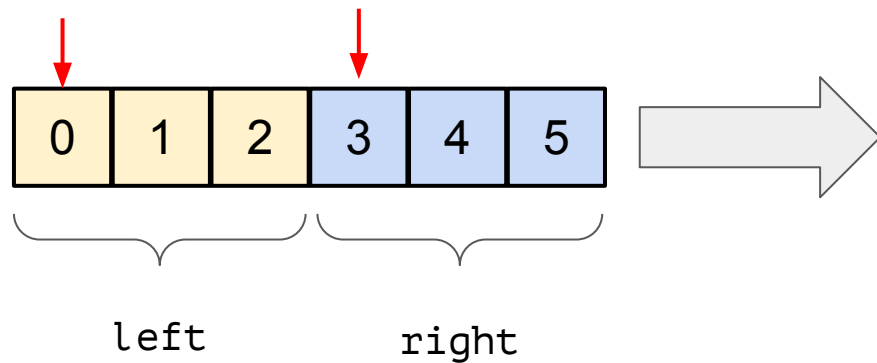
    while i < lsize: res[k++] = left[i++]
    while j < rsize: res[k++] = right[j++]
```

# Количество инверсий

Наблюдение #1: допустим в массиве не было разделенных инверсий. Как тогда будет работать процедура merge?

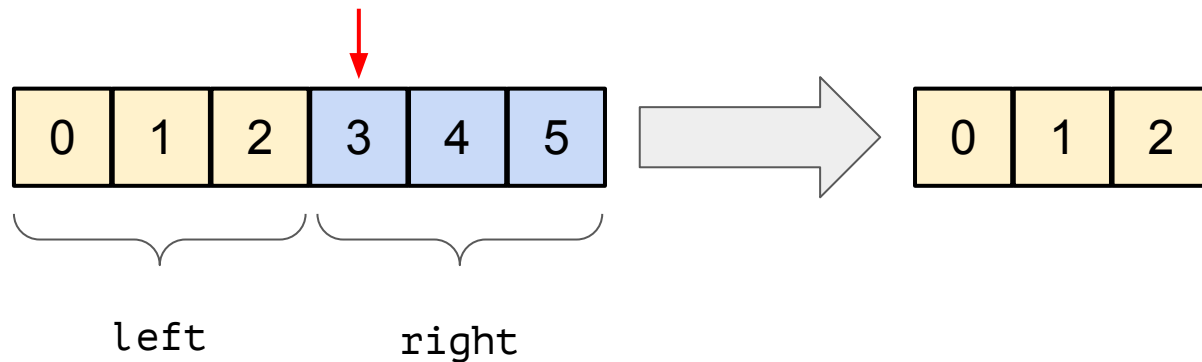
# Количество инверсий

Наблюдение #1: допустим в массиве не было разделенных инверсий. Как тогда будет работать процедура merge?



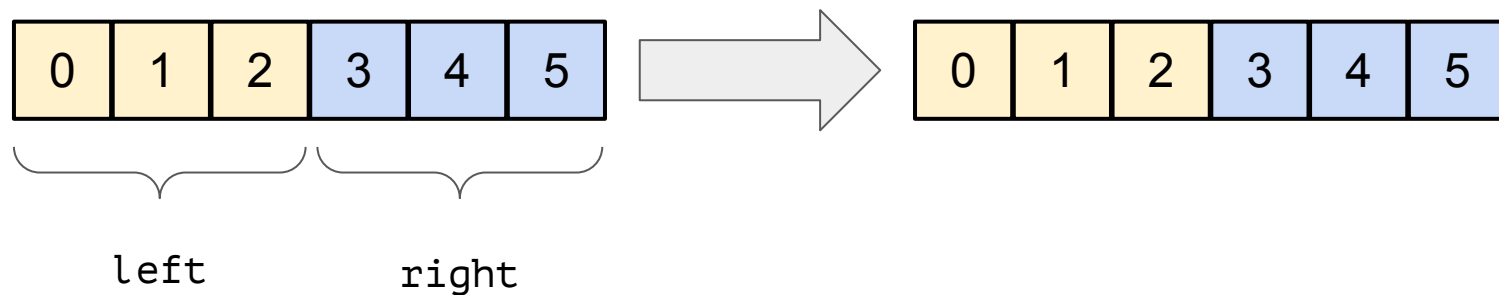
# Количество инверсий

Наблюдение #1: допустим в массиве не было разделенных инверсий. Как тогда будет работать процедура merge?



# Количество инверсий

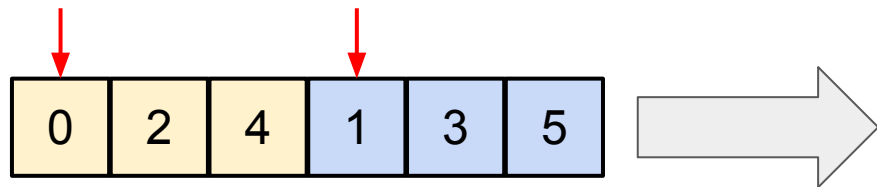
Наблюдение #1: допустим в массиве не было разделенных инверсий. Как тогда будет работать процедура merge?





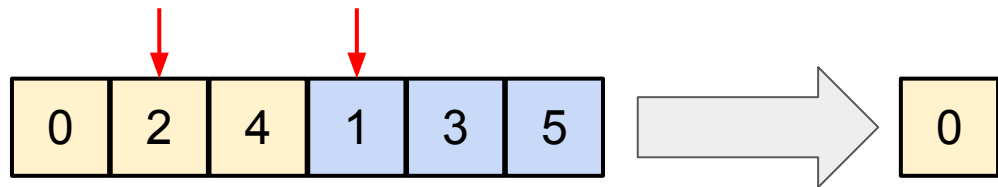
# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



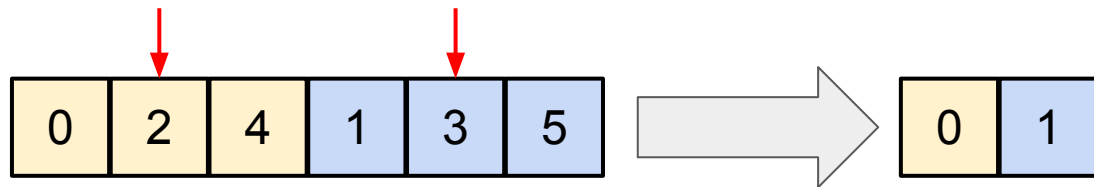
# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



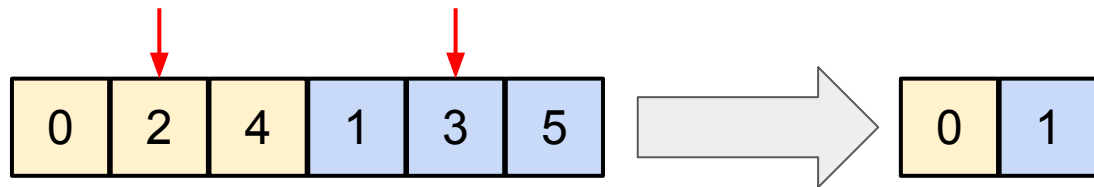
# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



# Количество инверсий

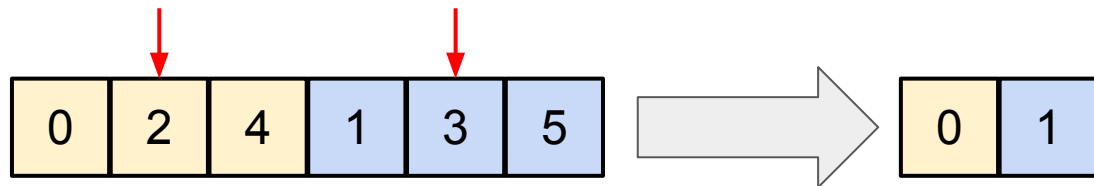
Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



Взяли элемент **справа**, а это же и значит, **слева** были элементы, большие его! Сколько?

# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



Взяли элемент **справа**, а это же и значит, **слева** были элементы, большие его! Сколько?

Столько, сколько слева осталось элементов.

# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

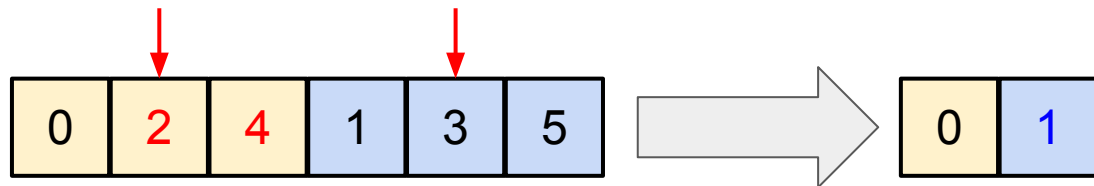
# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[\textcolor{red}{2}, \textcolor{blue}{1}]$ ,  $[4, 3]$ ,  $[\textcolor{red}{4}, \textcolor{blue}{1}]$

# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



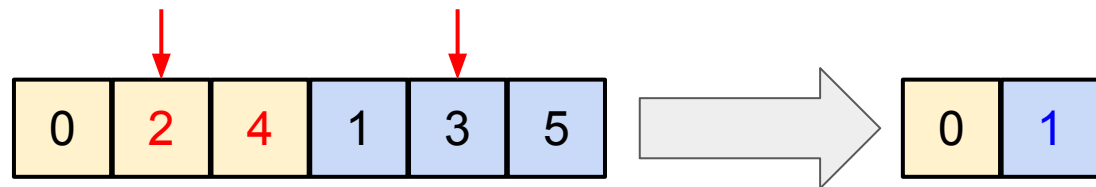
Взяли элемент **справа**, а это же и значит, **слева** были элементы, большие его! Сколько?

Столько, сколько слева осталось элементов. [2, 1]; [4, 1] 168



# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



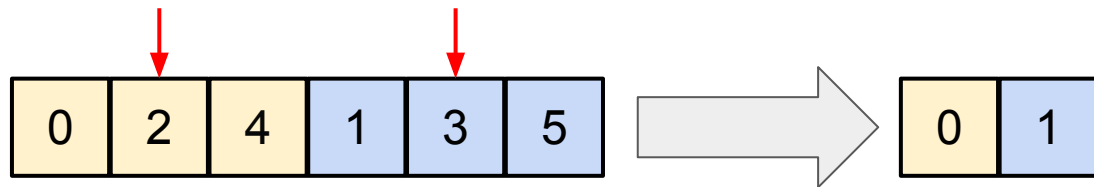
split inversions: 2

Взяли элемент **справа**, а это же и значит, **слева** были элементы, большие его! Сколько?

Столько, сколько слева осталось элементов. [2, 1]; [4, 1] 169

# Количество инверсий

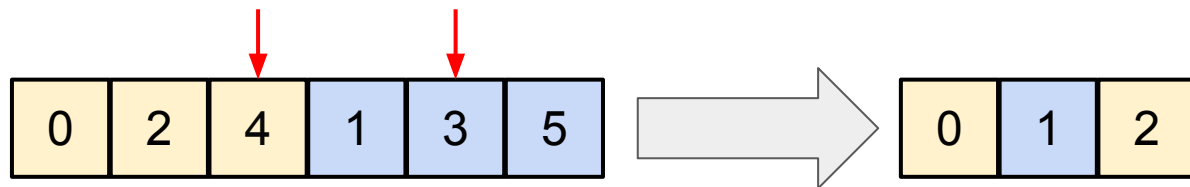
Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



split inversions: 2

# Количество инверсий

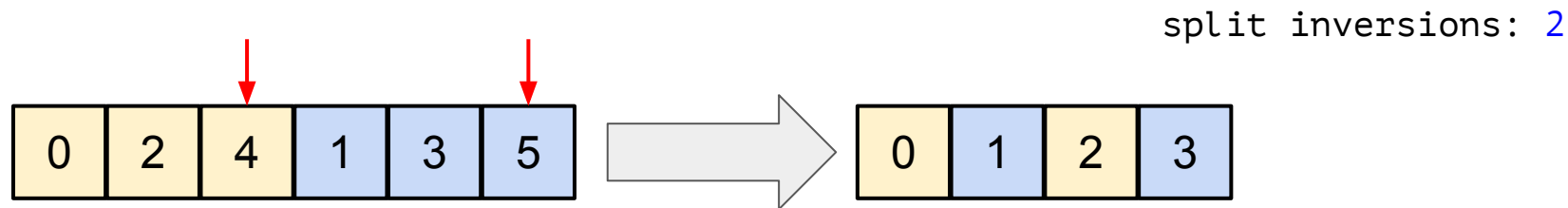
Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



split inversions: 2

# Количество инверсий

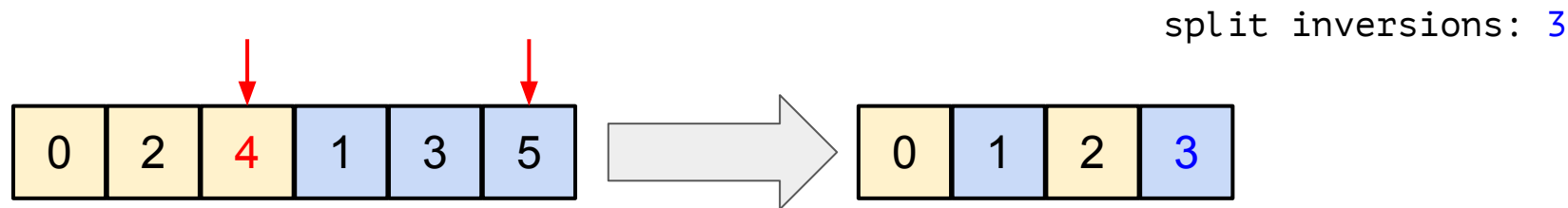
Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



Это опять случилось, тройка была не на своем месте.  
Сколько было ошибок?

# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



Это опять случилось, тройка была не на своем месте.  
Сколько было ошибок? Одна: [4, 3]

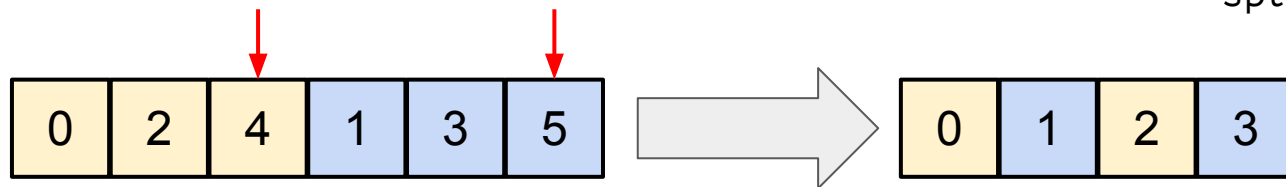
# Количество инверсий

0	2	4	1	3	5
0	1	2	3	4	5

Инверсии:  $(1, 3)$ ,  $(2, 4)$ ,  $(2, 3)$   
 $[2, 1]$ ,  $[4, 3]$ ,  $[4, 1]$

# Количество инверсий

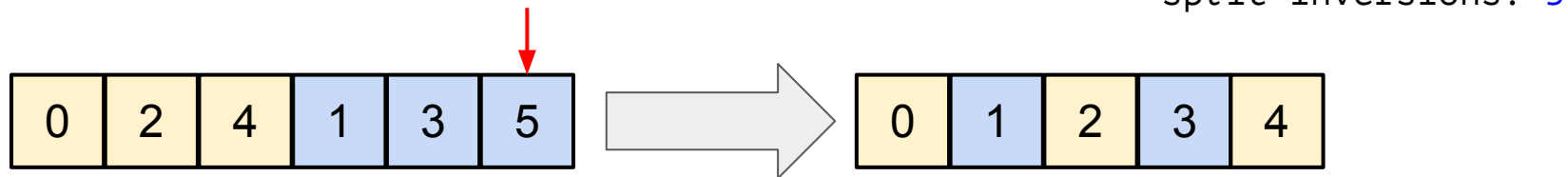
Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть



split inversions: 3

# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть

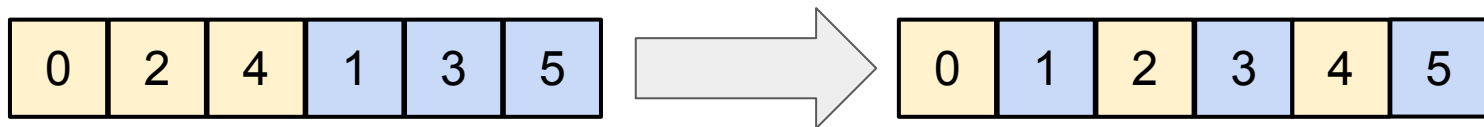




# Количество инверсий

Наблюдение #2: рассмотрим работу merge, когда разделенные инверсии в массиве есть

split inversions: 3



## Количество инверсий

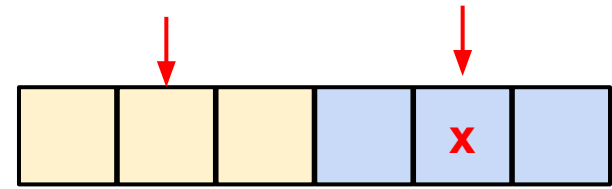
**Утверждение:** пусть на очередном шаге алгоритма слияния мы копируем в результирующий буфер элемент **x** из **правого** отсортированного массива.

## Количество инверсий

**Утверждение:** пусть на очередном шаге алгоритма слияния мы копируем в результирующий буфер элемент **x** из **правого** отсортированного массива.

Тогда этот элемент участвует в стольких разделенных инверсиях, сколько осталось **необработанных** элементов в **левом** массиве.

# Количество инверсий



Действительно: пусть  $y$  - элемент из левого массива (значит его индекс меньше индекса  $x$ ).

## Количество инверсий



**Действительно:** пусть  $y$  - элемент из левого массива (значит его индекс меньше индекса  $x$ ).

Если он был скопирован до  $x$ , то  $y < x$

## Количество инверсий

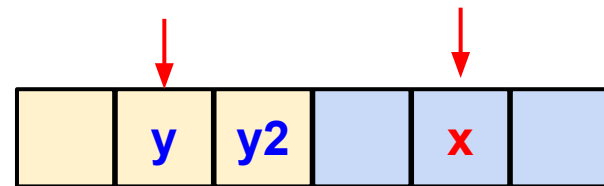


**Действительно:** пусть  $y$  - элемент из левого массива (значит его индекс меньше индекса  $x$ ).

Если он был скопирован до  $x$ , то  $y < x$

Если же он будет скопирован после  $x$ , то  $y > x$ , следовательно это инверсия по определению.

## Количество инверсий



**Действительно:** пусть  $y$  - элемент из левого массива (значит его индекс меньше индекса  $x$ ).

Если он был скопирован до  $x$ , то  $y < x$   
Если же он будет скопирован после  $x$ , то  $y > x$ , следовательно это инверсия по определению. На момент копирования  $x$  все оставшиеся в левом массиве - больше его.  $\square$

```
def sort_and_count_inversions(array: int[]) -> int:
    if len(array) == 1: return 0

    middle = len(array) / 2
    left    = sort_and_count_inversions(array[:middle])
    right   = sort_and_count_inversions(array[middle:])

    buffer = new int[len(array)]
    split  = merge_and_count_split(array[:middle],
                                    array[middle:],
                                    buffer)

    for i in [0; len(array)): array[i] = buffer[i]

    return left + right + split
```



# Количество инверсий

`merge_and_count_split:`

1. Делает обычный merge из сортировки слиянием

# Количество инверсий

`merge_and_count_split:`

1. Делает обычный merge из сортировки слиянием
2. Но каждый раз, когда выбирает элемент из правого массива, обновляет количество `разделенных инверсий`

Сложность?

# Количество инверсий

`merge_and_count_split:`

1. Делает обычный merge из сортировки слиянием
2. Но каждый раз, когда выбирает элемент из правого массива, обновляет количество `разделенных инверсий`

Сложность? Все еще  $O(N)$

```
def count_inversions(array: int[]) -> int:
    if len(array) == 1:
        return 0

    middle = len(array) / 2
    left    = count_inversions(array[:middle])
    right   = count_inversions(array[middle:])

    split   = count_split_inversions(array)

    return left + right + split
```

Если бы удалось реализовать `count_split_inversions` за  $O(N)$ , то получили бы суммарную сложность  $O(N \log N)$

```
def count_inversions(array: int[]) -> int:  
    if len(array) == 1:  
        return 0  
  
    middle = len(array) / 2  
    left    = count_inversions(array[:middle])  
    right   = count_inversions(array[middle:])  
  
    split   = count_split_inversions(array)  
  
    return left + right + split
```

Таким образом, сложность алгоритма -  $O(N * \log N)$



## Мини-задача #8 (1 балл)

Покажите свое ~~кунг-фу~~ умение считать инверсии!

Реализуйте алгоритм, работающий за  $O(N * \log N)$

-----

Попробуйте применить его здесь:

<https://leetcode.com/problems/global-and-local-inversions/>

А потом найдите более простое решение задачи.

# Сортировка слиянием

Еще один взгляд на сложность сортировки слиянием:

# Сортировка слиянием

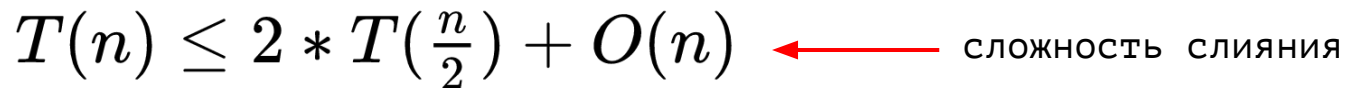
Еще один взгляд на сложность сортировки слиянием:

$$T(n) \leq 2 * T(\frac{n}{2}) + O(n)$$



# Сортировка слиянием

Еще один взгляд на сложность сортировки слиянием:

$$T(n) \leq 2 * T\left(\frac{n}{2}\right) + O(n)$$
A diagram illustrating the recurrence relation for the time complexity of merge sort. The equation is  $T(n) \leq 2 * T\left(\frac{n}{2}\right) + O(n)$ . Three red arrows point to specific parts of the equation: one points from the text 'количество подзадач' to the coefficient '2', another points from 'размер подзадачи' to the term  $T\left(\frac{n}{2}\right)$ , and a third points from 'сложность слияния' to the  $O(n)$  term.

количество  
подзадач

размер  
подзадачи

← сложность слияния

# Divide and Conquer

The **Master** Method:



# Divide and Conquer

## The **Master** Method:

(a.k.a. основная теорема о рекуррентных соотношениях)



# Divide and Conquer

## The **Master** Method:

(а.к.а. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T\left(\frac{n}{b}\right) + O(n^d)$$



# Divide and Conquer

## The **Master** Method:

(а.к.а. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T\left(\frac{n}{b}\right) + O(n^d)$$

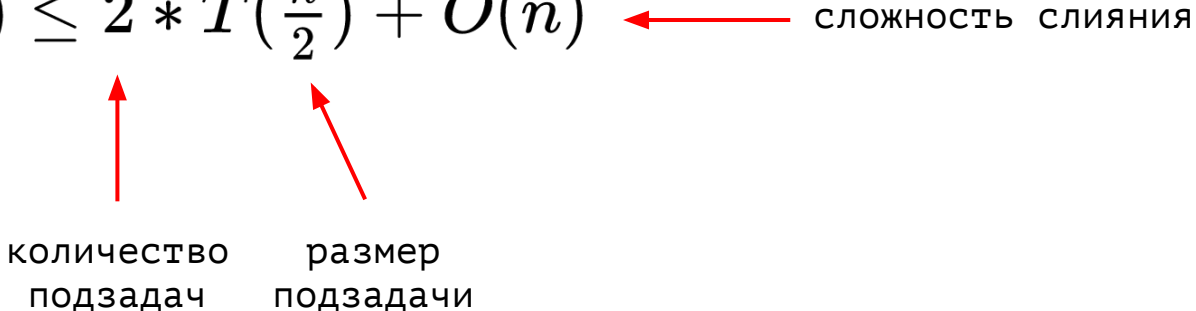
то,

$$T(n) = \begin{cases} O(n^d * \log(n)), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases}$$



# Сортировка слиянием

Еще один взгляд на сложность сортировки слиянием:

$$T(n) \leq 2 * T\left(\frac{n}{2}\right) + O(n)$$


количество  
подзадач

размер  
подзадачи

← сложность слияния

# Divide and Conquer

## The **Master** Method:

(а.к.а. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T\left(\frac{n}{b}\right) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \log(n)), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases}$$



# Divide and Conquer

TO BE CONTINUED...

## The Master Method:

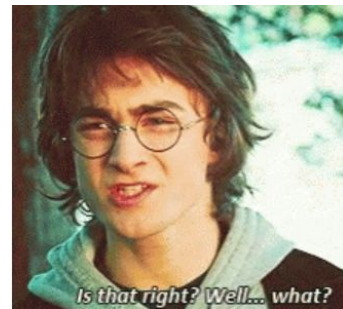
(a.k.a. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T\left(\frac{n}{b}\right) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \log(n)), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases}$$





# Takeaways

- Сортировка слиянием дает качественно новую временную сложность:  $O(N \cdot \log N)$
- За что приходится платить память,  $O(N)$

# Takeaways

- Сортировка слиянием дает качественно новую временную сложность:  $O(N \cdot \log N)$
- За что приходится платить памятью,  $O(N)$
- Есть варианты merge sort и за  $O(1)$  памяти! Но они медленнее и пропадает **стабильность**

# Takeaways

- Сортировка слиянием дает качественно новую временную сложность:  $O(N \log N)$
- За что приходится платить памятью,  $O(N)$
- Есть варианты merge sort и за  $O(1)$  памяти! Но они медленнее и пропадает **стабильность**
- Задача поиска инверсий, как частный случай
- **The Master Method** (wait, what???)

## Мини-задача #7 (3 балла)

Реализовать сортировку слиянием, используя константное количество дополнительной памяти.

Проверить решение необходимо на литкоде:

<https://leetcode.com/problems/sort-an-array>

Там же сравнить потребление памяти со стандартной реализацией.

**Дополнительно:** реализовать оптимизированную in-place сортировку слиянием из статьи.

## Мини-задача #8 (1 балл)

Покажите свое ~~кунг-фу~~ умение считать инверсии!

Реализуйте алгоритм, работающий за  $O(N * \log N)$

-----

Попробуйте применить его здесь:

<https://leetcode.com/problems/global-and-local-inversions/>

А потом найдите более простое решение задачи.