СБЕР

# Переполнение в Collections.rotate

**Сахарин Никита**

Старший разработчик в **SberData**

nikitasa1997@gmail.com

t.me/nikita_sakharin

# Java как «черный ящик»

- Смотрели ли Вы хоть раз реализацию метода **Java SE** или **Spring**?

# Java как «черный ящик»

- Смотрели ли Вы хоть раз реализацию метода **Java SE** или **Spring**?
- Возникала ли у Вам проблема, которая приводила Вас в исходники **Java SE** или **Spring**?

# Java как «черный ящик»

- Смотрели ли Вы хоть раз реализацию метода **Java SE** или **Spring**?
- Возникала ли у Вам проблема, которая приводила Вас в исходники **Java SE** или **Spring**?
- Доводилось ли Вам находить в ней **Bug**-и?

# Java как «черный ящик»

- Смотрели ли Вы хоть раз реализацию метода **Java SE** или **Spring**?
- Возникала ли у Вам проблема, которая приводила Вас в исходники **Java SE** или **Spring**?
- Доводилось ли Вам находить в ней **Bug**-и?
- Получалось ли у Вас довести дело до **merge** в **master**?

# Верните мне мой 20..

- Вы снова студент-первокурсник на CS-факультете одного из ВУЗов

# Верните мне мой 20..

- Вы снова студент-первокурсник на CS-факультете одного из ВУЗов
- За выходные Вам нужно сделать лабораторную по сортировкам и сдать её в понедельник

# Верните мне мой 20..

- Вы снова студент-первокурсник на CS-факультете одного из ВУЗов
- За выходные Вам нужно сделать лабораторную по сортировкам и сдать её в понедельник
- Вам выпал **«Merge Sort»**

# Верните мне мой 20..

- Вы снова студент-первокурсник на CS-факультете одного из ВУЗов
- За выходные Вам нужно сделать лабораторную по сортировкам и сдать её в понедельник
- Вам выпал **«Merge Sort»**
- Вы стремитесь получить рейтинговые балы

# Верните мне мой 20..

- Вы снова студент-первокурсник на CS-факультете одного из ВУЗов
- За выходные Вам нужно сделать лабораторную по сортировкам и сдать её в понедельник
- Вам выпал **«Merge Sort»**
- Вы стремитесь получить рейтинговые балы
- Реализации на **«4»** и **«5»** используют меньше дополнительной памяти

# Что почём?

## На «3»

# Что почём?

## На «3»

- Merge Sort

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

## На «5»

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

## На «5»

- Block Merge Sort

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

## На «5»

- Block Merge Sort
- Память: $O(1)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

## На «5»

- Block Merge Sort
- Память: $O(1)$
- Время: $O(n\,log\,n)$

# Что почём?

## На «3»

- Merge Sort
- Память: $O(n)$
- Время: $O(n\,log\,n)$
- Стабильный

## На «4»

- Rotate Merge Sort
- Память: $O(\log n)$
- Время: $O(n\,log^2\,n)$
- Стабильный

## На «5»

- Block Merge Sort
- Память: $O(1)$
- Время: $O(n\,log\,n)$
- За стабильный **«5+»**


Ah shit, here we go again.

# Merge Sort

- Два подхода:

# Merge Sort

- Два подхода:
  1. Рекурсивный **Top-Down**

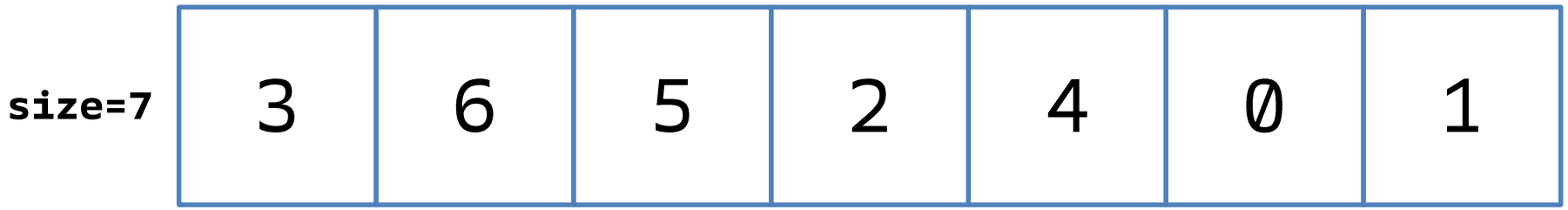
# Merge Sort

- Два подхода:
  1. Рекурсивный **Top-Down**
  2. Итеративный **Bottom-up**

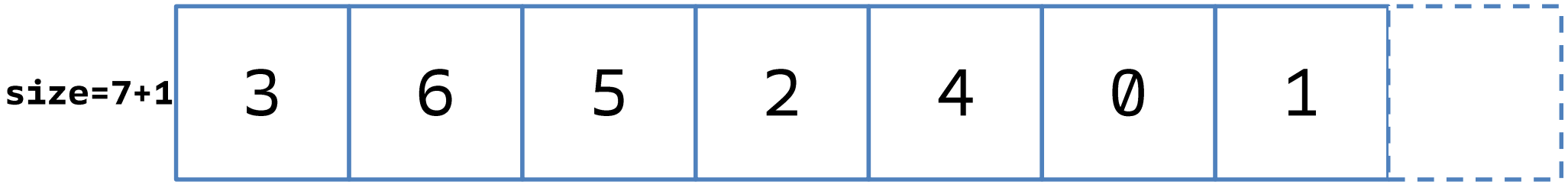
# Merge Sort

- Два подхода:
  1. Рекурсивный **Top-Down**
  2. Итеративный **Bottom-up**
- Будем использовать **Bottom-up**, так как рекурсия - это тоже память

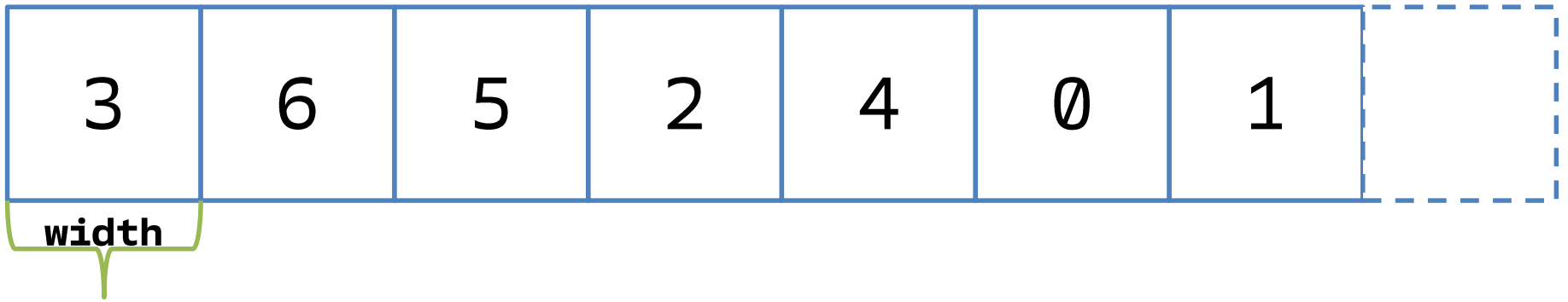
# Merge Sort



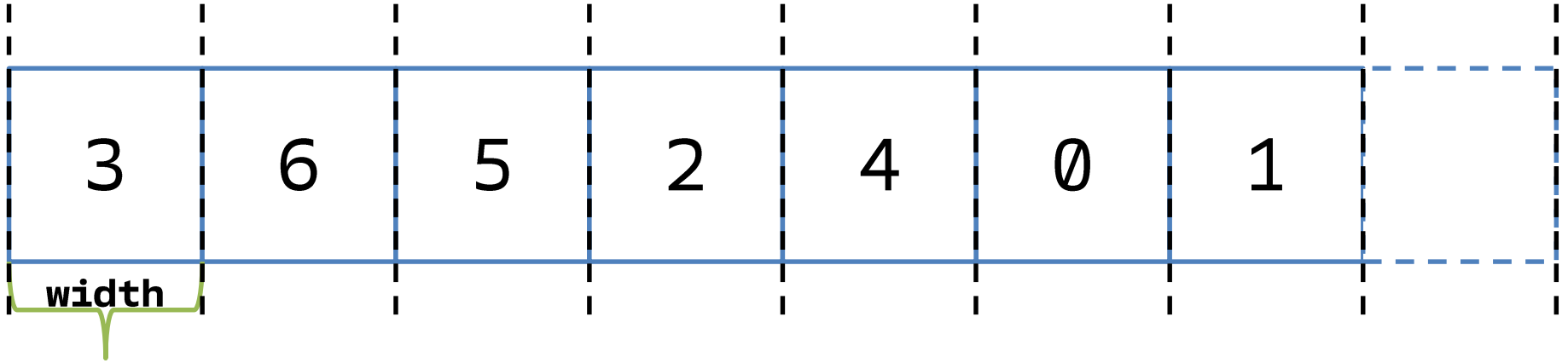
# Merge Sort



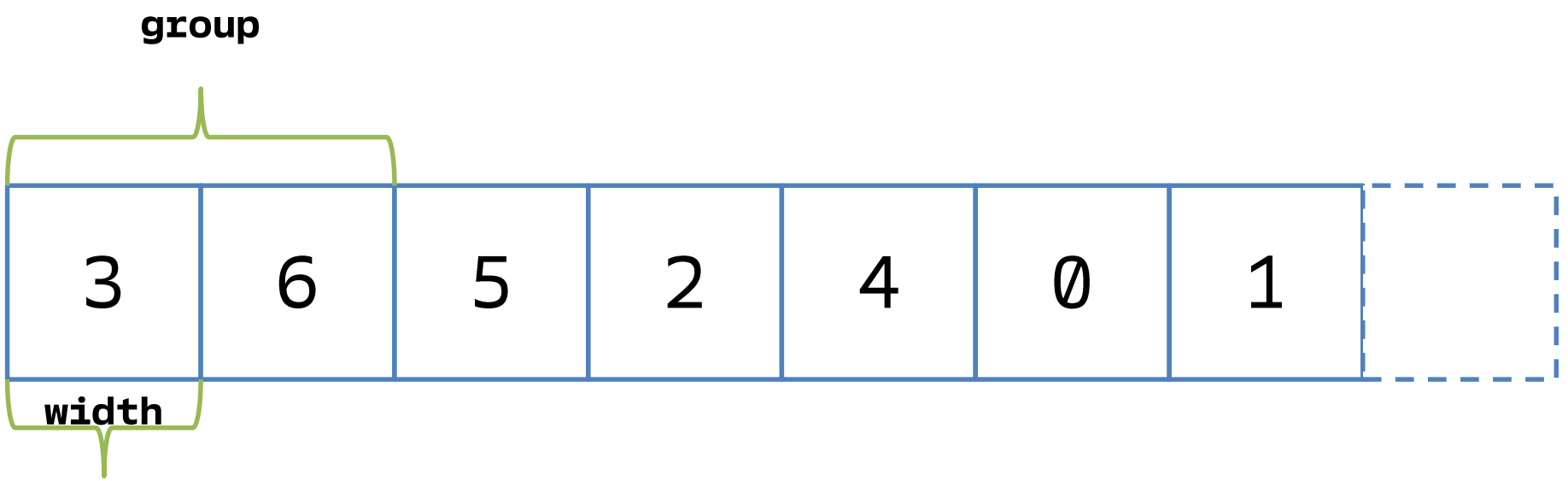
# Merge Sort



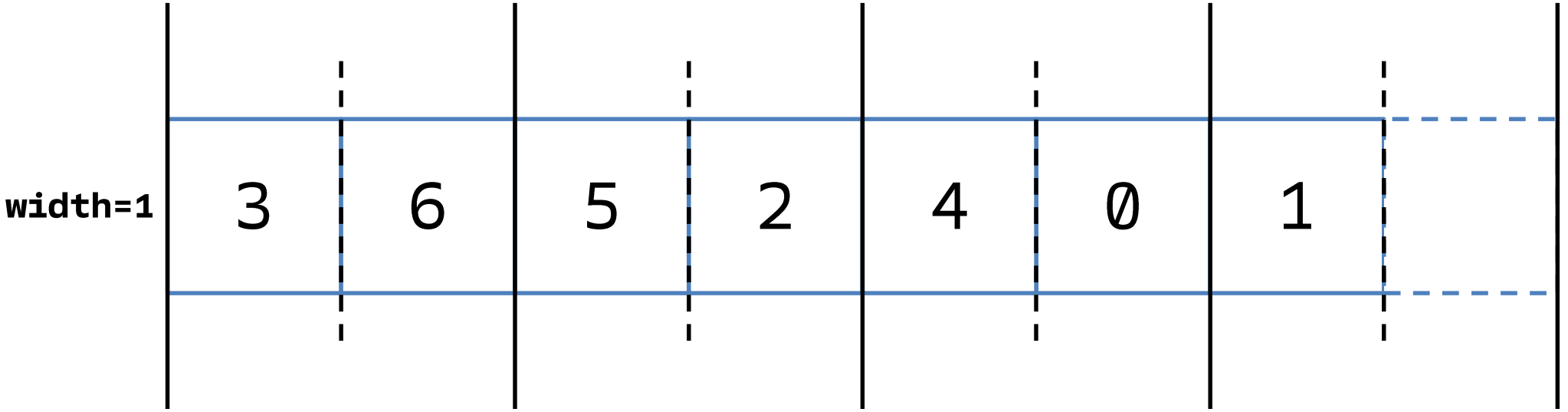
# Merge Sort



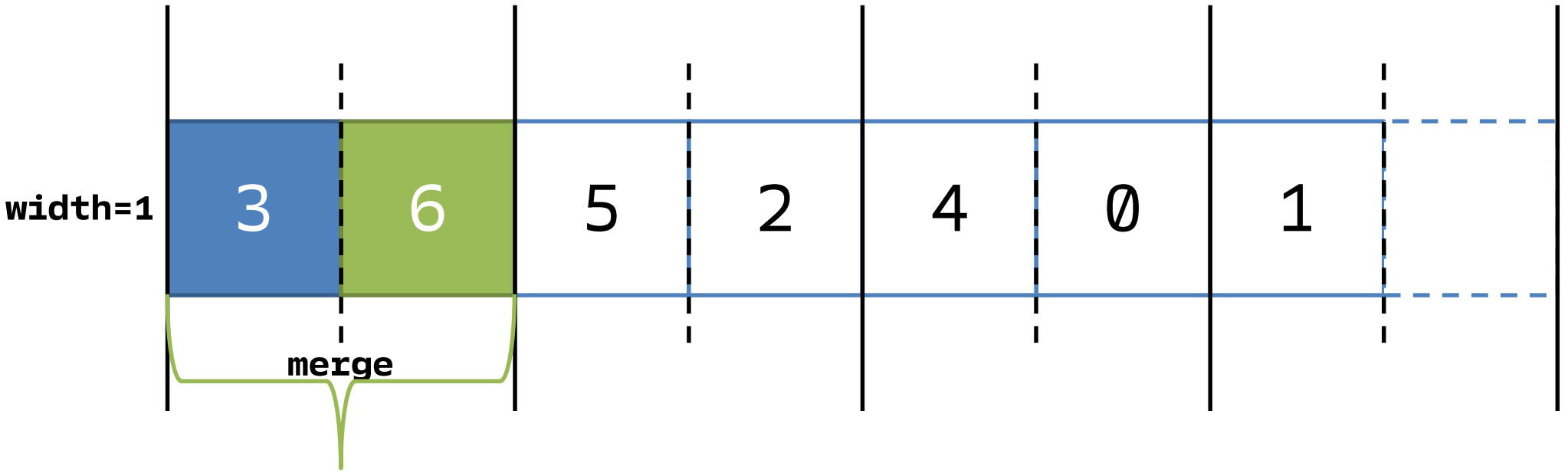
# Merge Sort



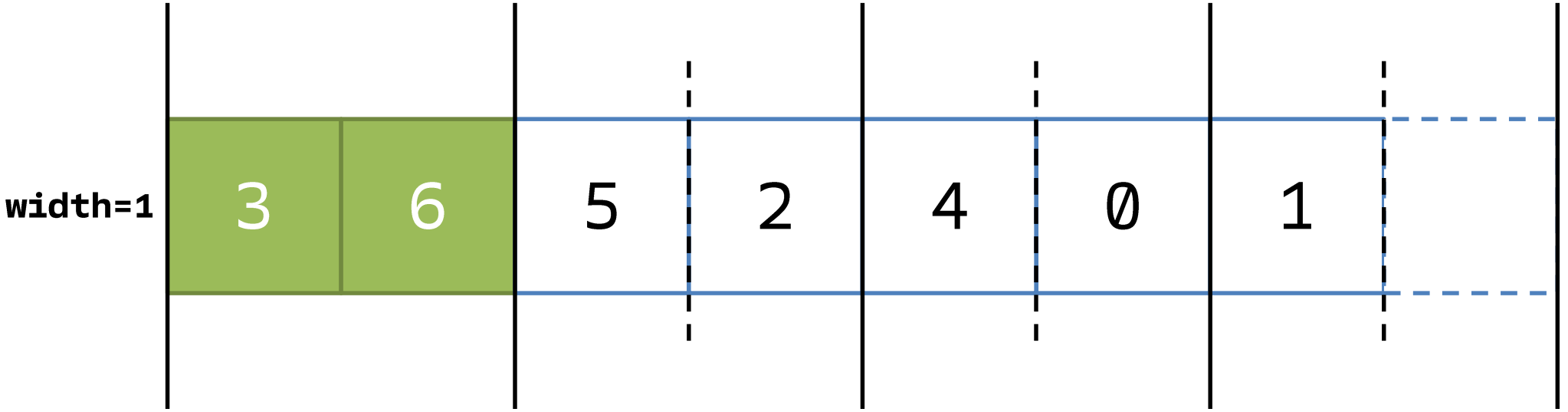
# Merge Sort



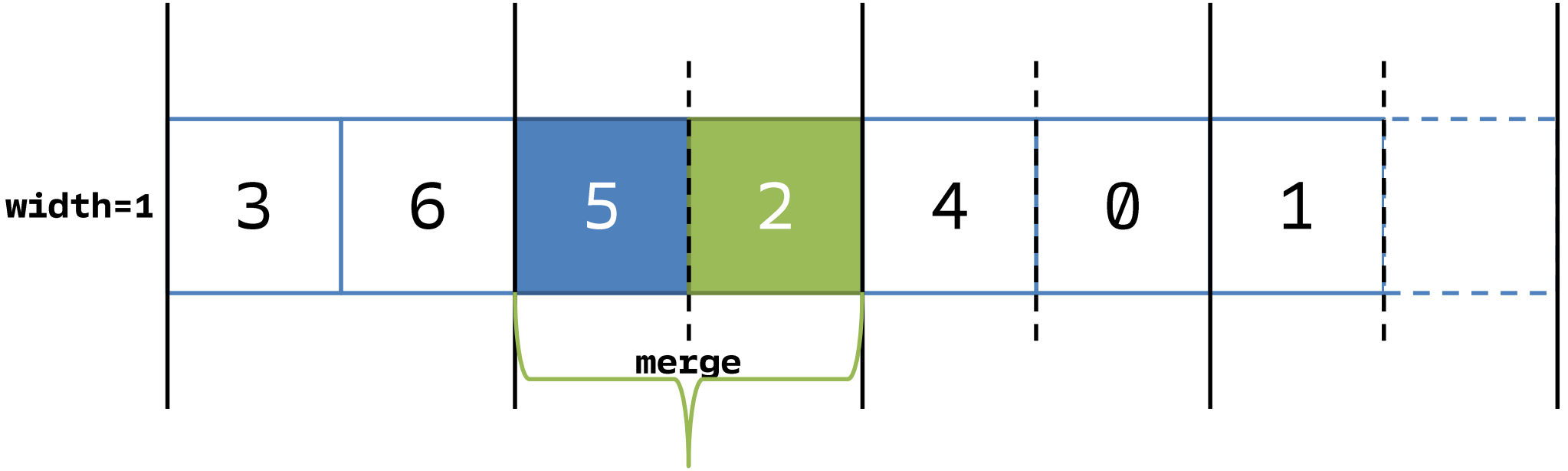
# Merge Sort



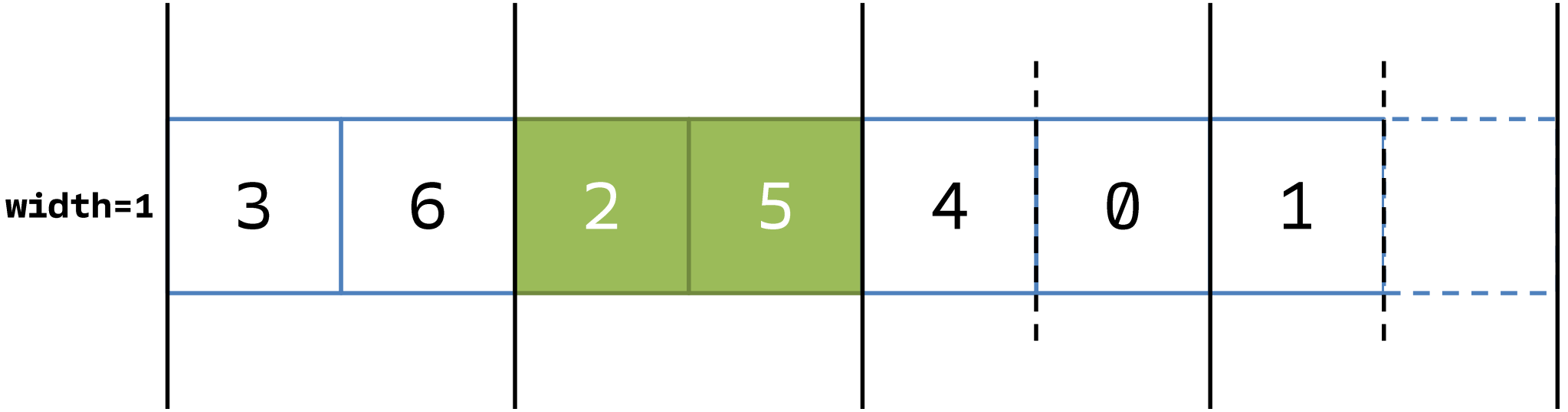
# Merge Sort



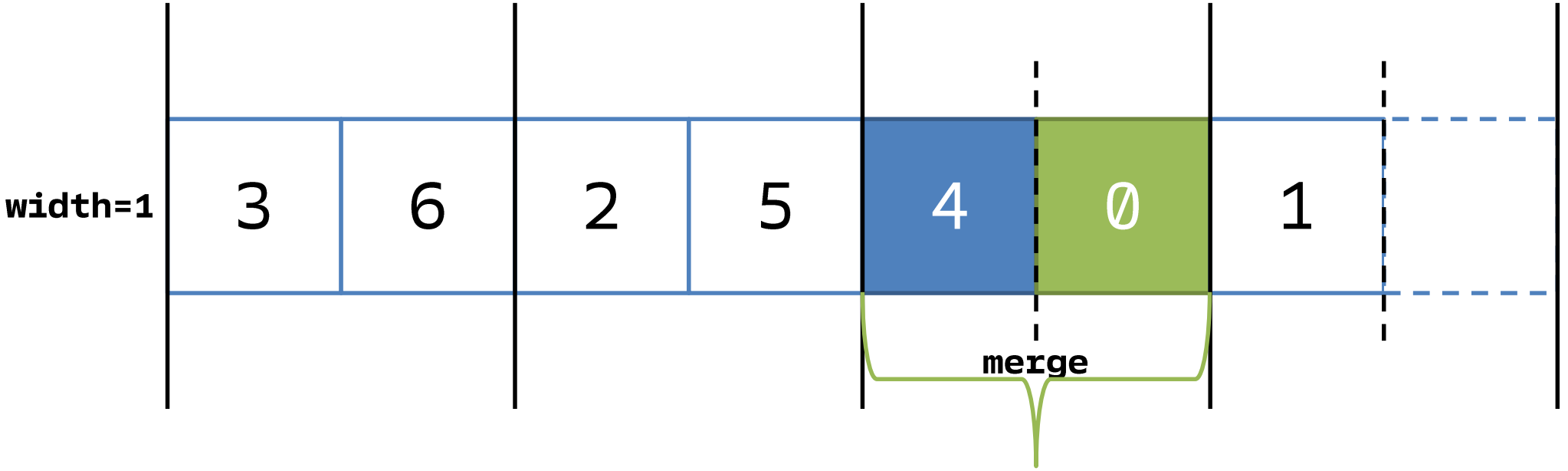
# Merge Sort



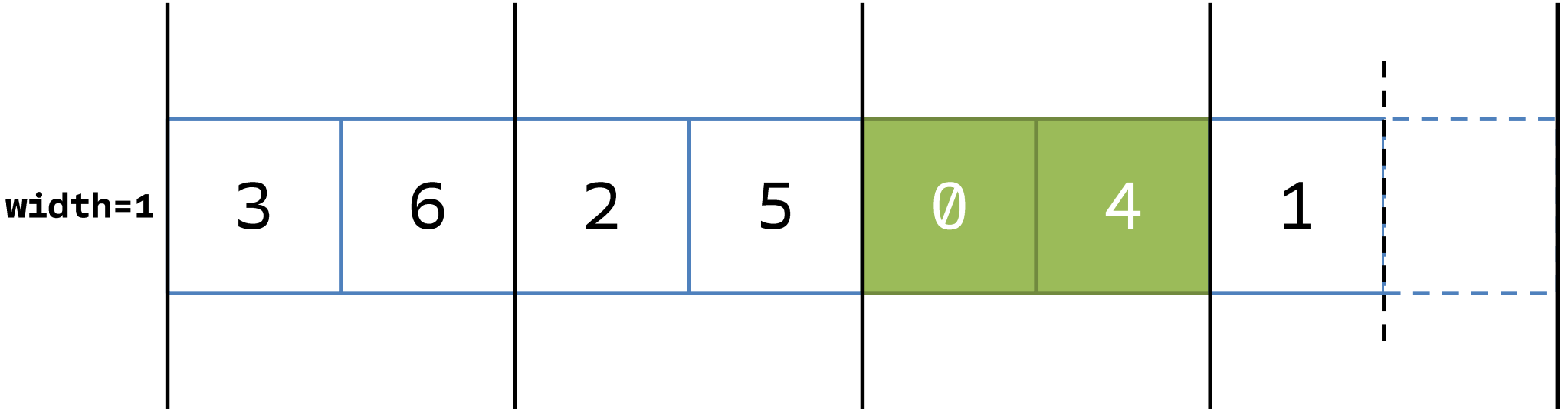
# Merge Sort



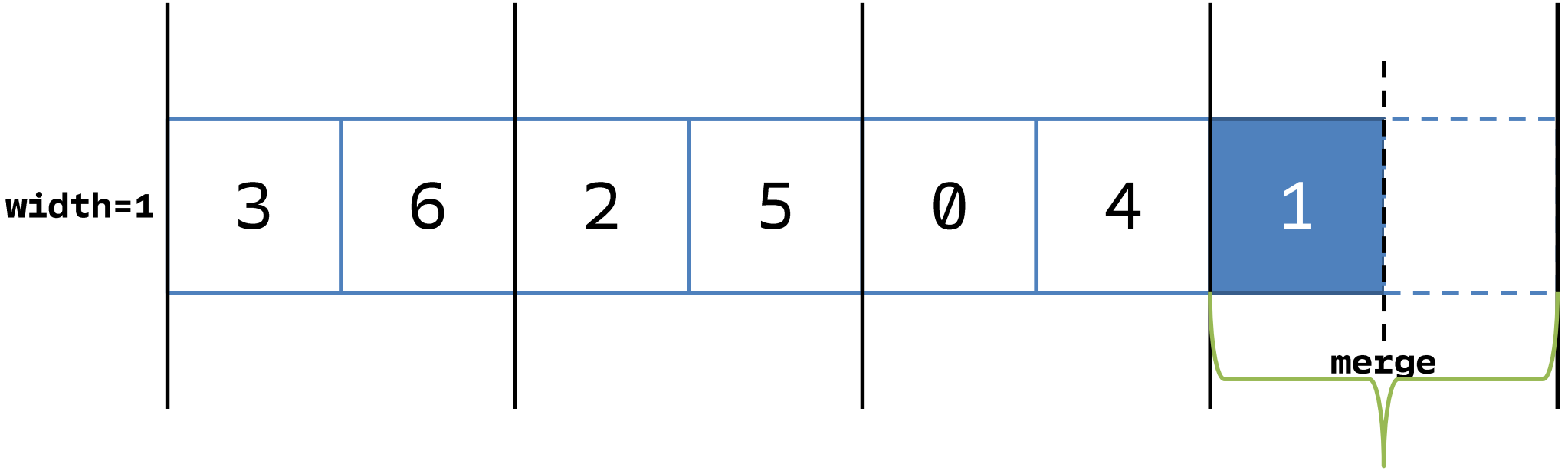
# Merge Sort



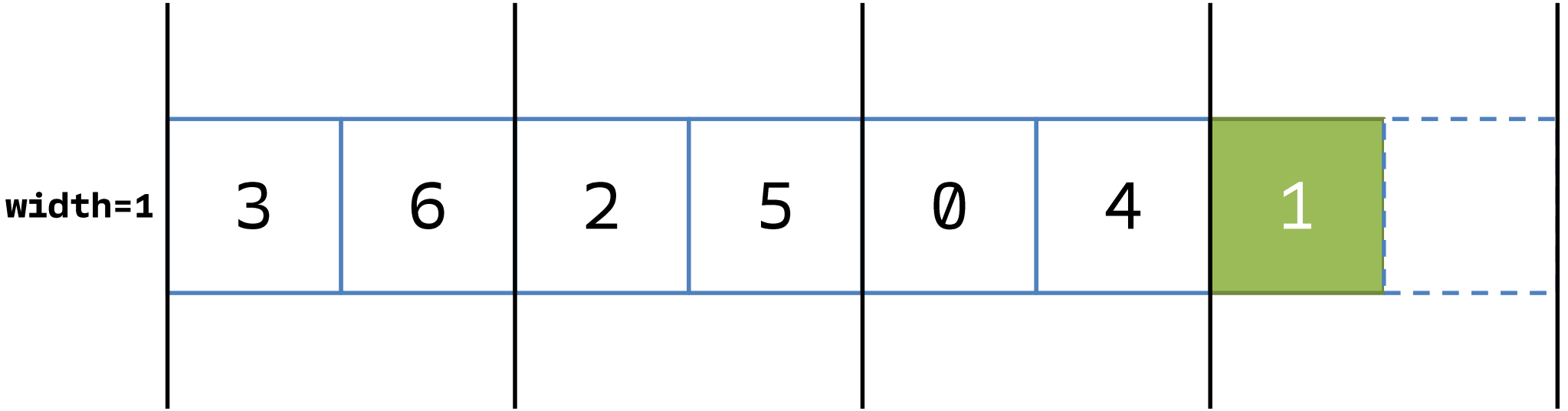
# Merge Sort



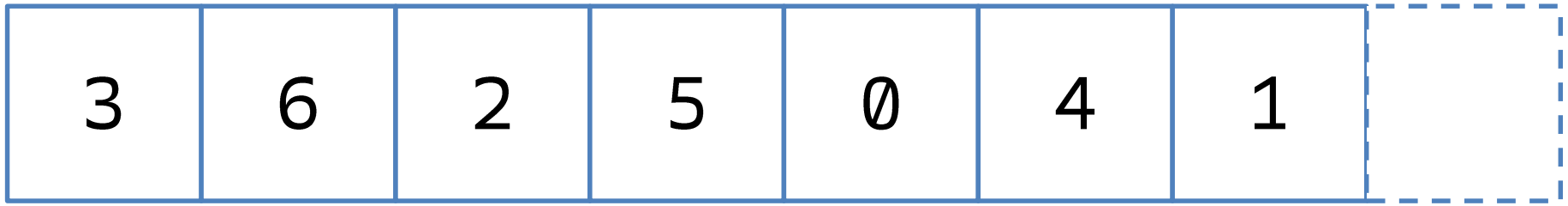
# Merge Sort



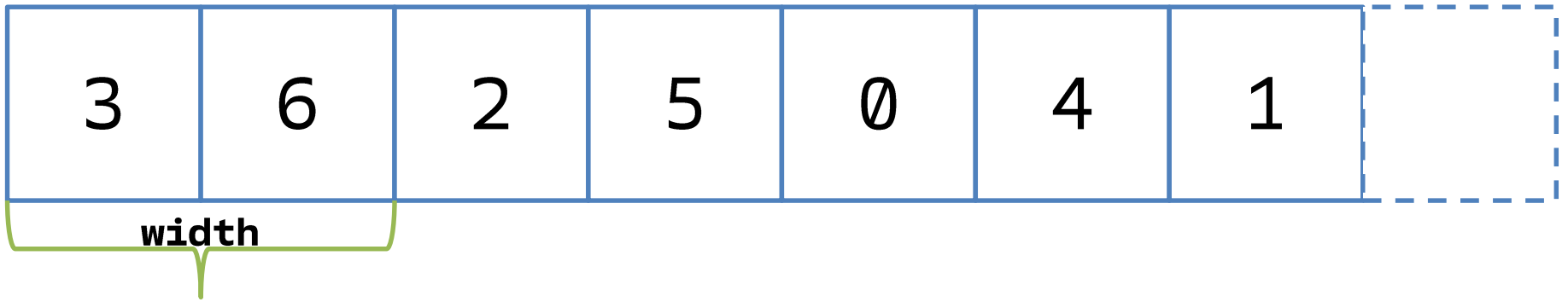
# Merge Sort



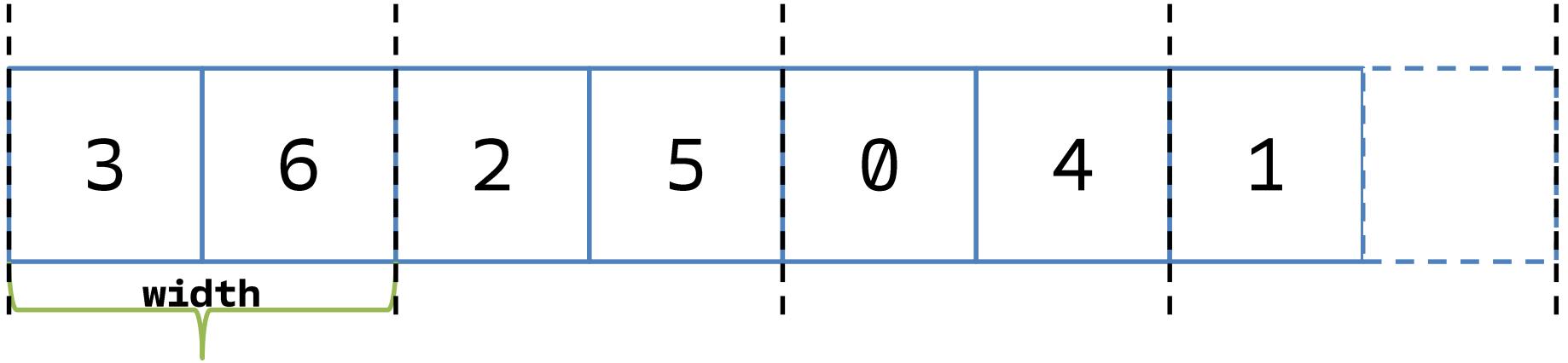
# Merge Sort



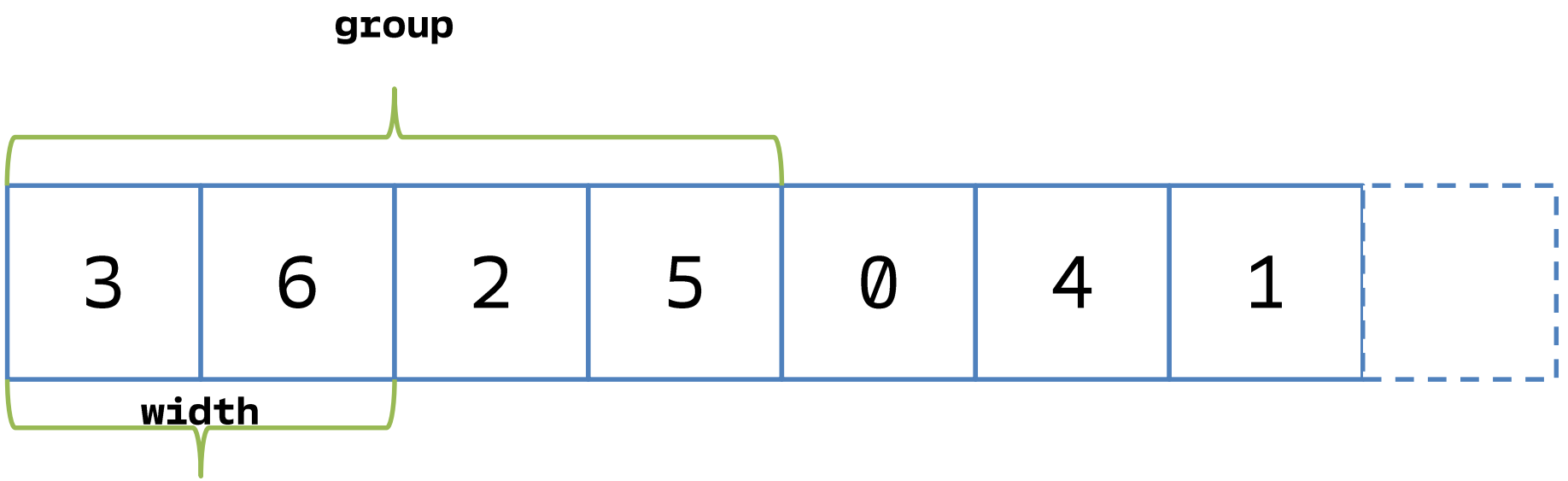
# Merge Sort



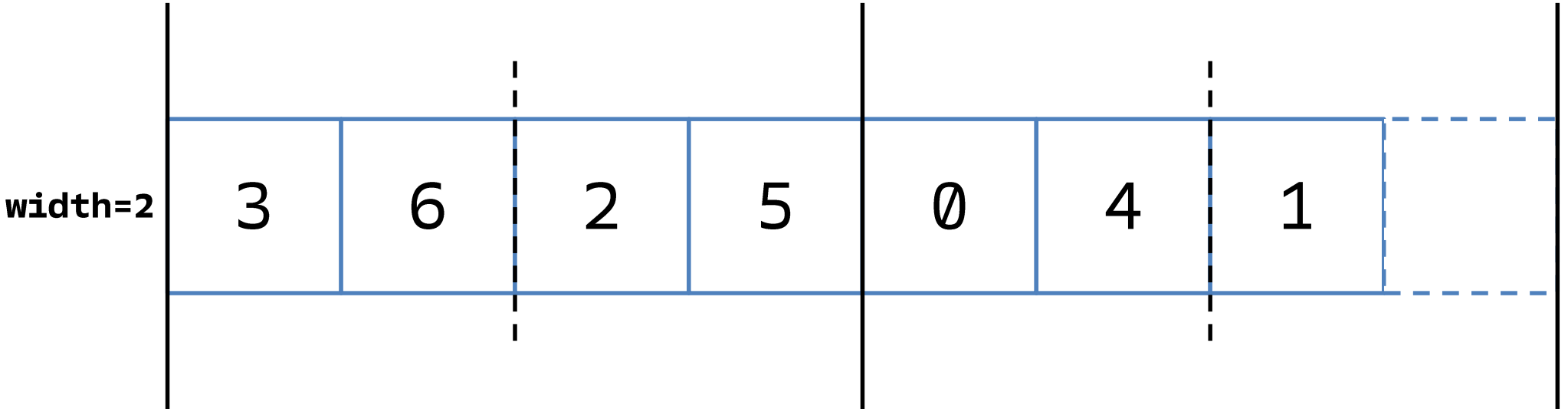
# Merge Sort



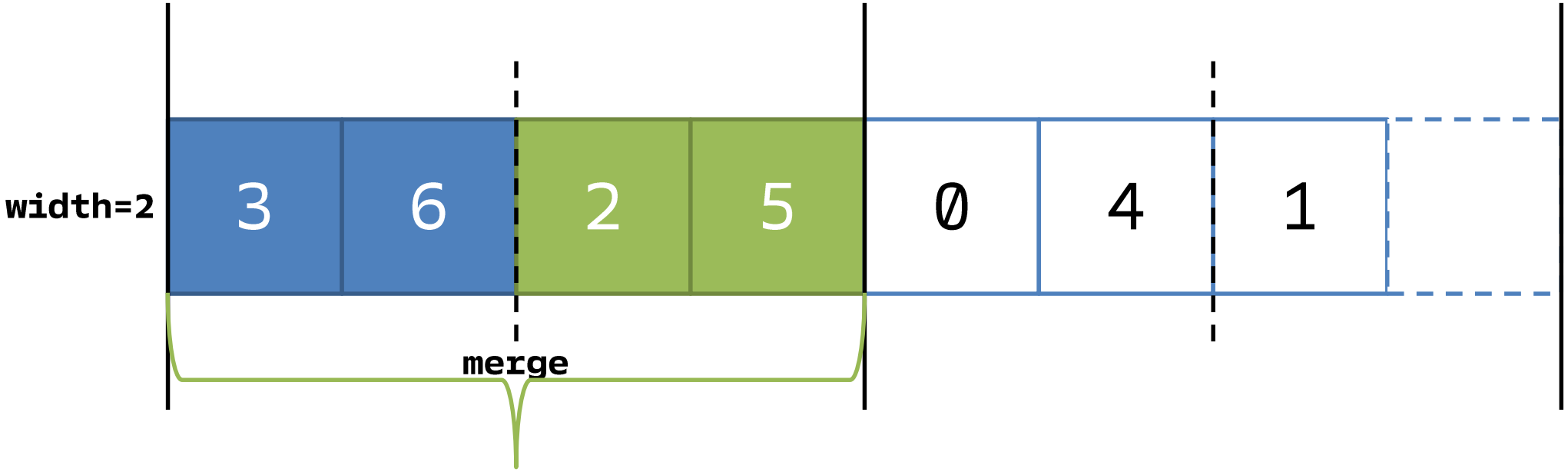
# Merge Sort



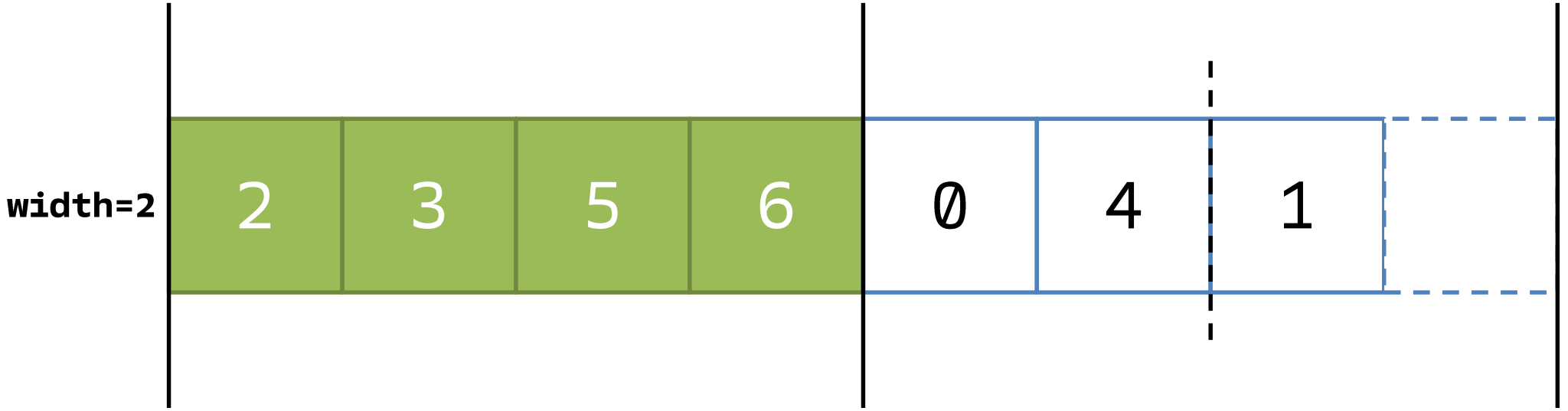
# Merge Sort



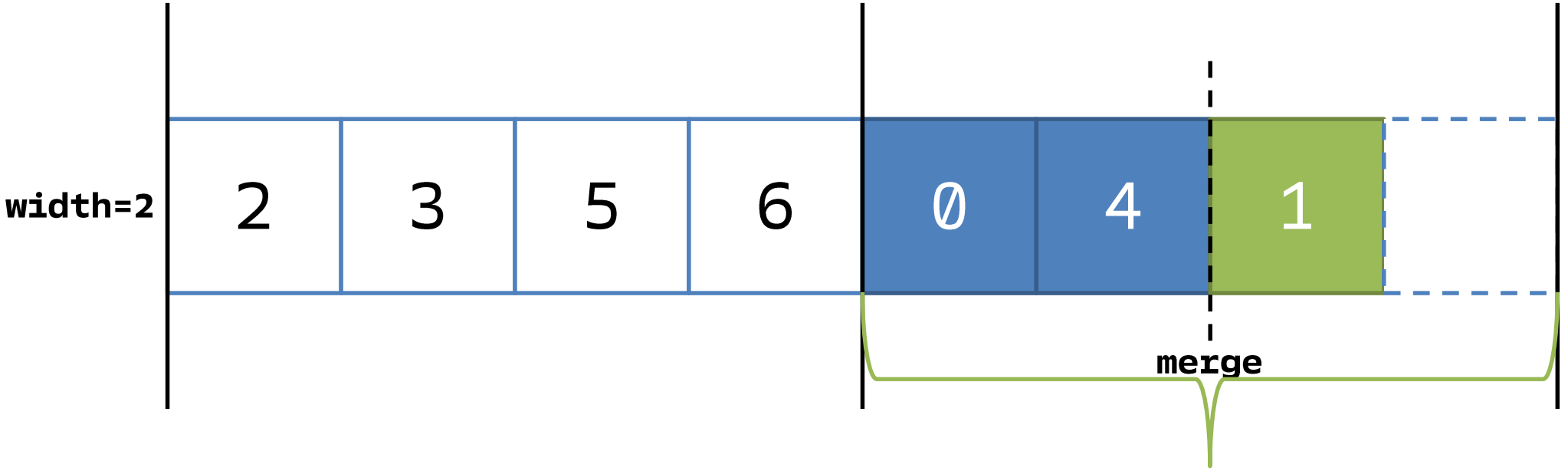
# Merge Sort



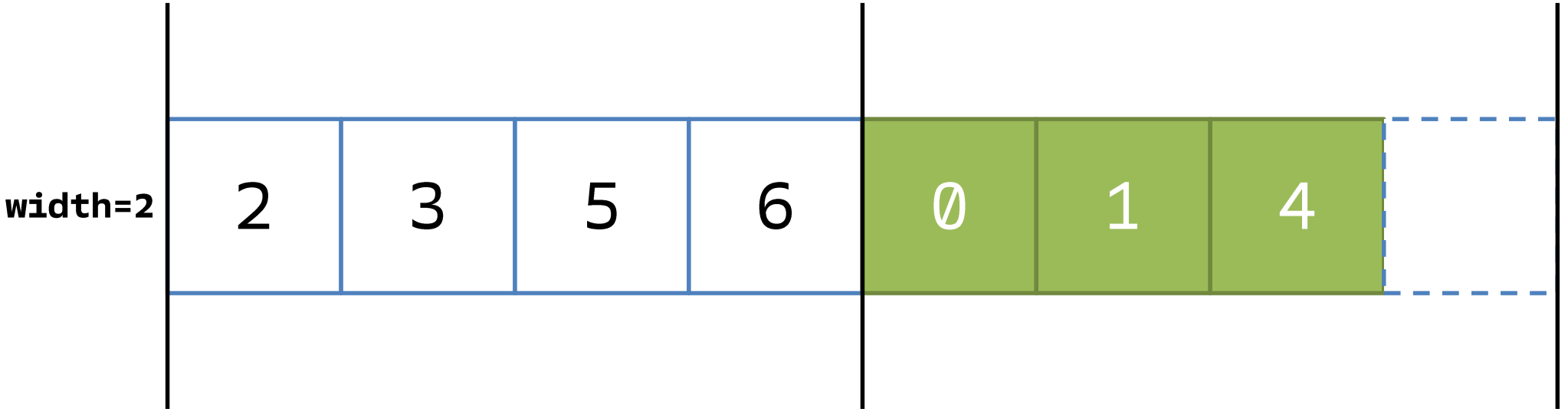
# Merge Sort



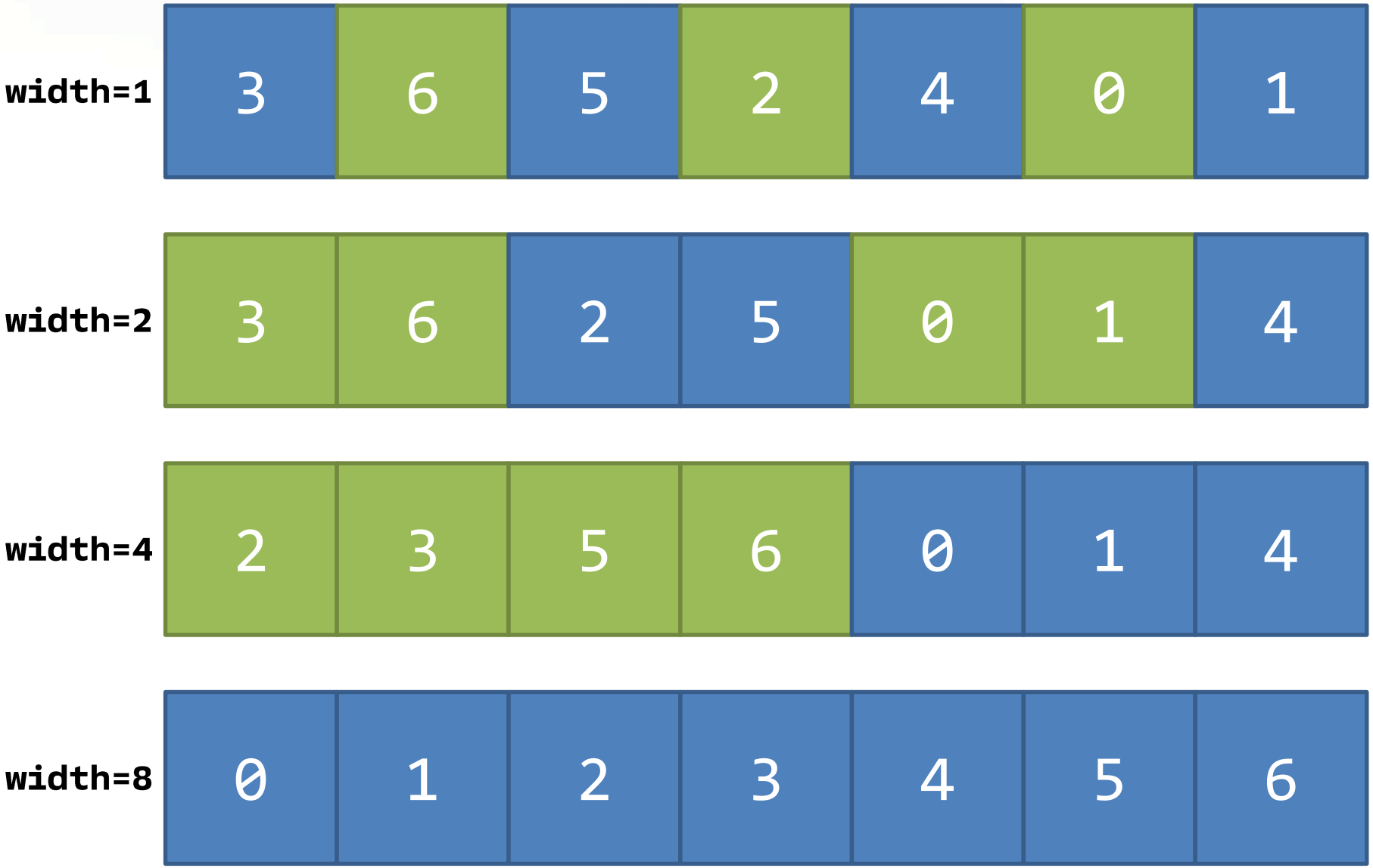
# Merge Sort



# Merge Sort



# Merge Sort


width=1
3 6 5 2 4 0 1
width=2
3 6 2 5 0 1 4
width=4
2 3 5 6 0 1 4
width=8
0 1 2 3 4 5 6

# Merge

- Классический алгоритм **Merge Sort** использует дополнительную память для слияния двух отсортированных частей в один массив

# Merge

- Классический алгоритм **Merge Sort** использует дополнительную память для слияния двух отсортированных частей в один массив
- Вместо этого будем использовать **Rotate Merge**

# Merge

- Классический алгоритм **Merge Sort** использует дополнительную память для слияния двух отсортированных частей в один массив
- Вместо этого будем использовать **Rotate Merge**
- Он рекурсивен и использует $O(\log n)$ памяти, так как глубина рекурсии не более, чем $O(\log n)$

# Merge

- Классический алгоритм **Merge Sort** использует дополнительную память для слияния двух отсортированных частей в один массив
- Вместо этого будем использовать **Rotate Merge**
- Он рекурсивен и использует $O(\log n)$ памяти, так как глубина рекурсии не более, чем $O(\log n)$
- Его время работы $O(n \log n)$, поэтому весь алгоритм **Rotate Merge Sort** будет работать за $O(n \log^2 n)$

# Binary Search

- Бинарный поиск позволяет за $O(log\, n)$ найти ключ в отсортированном массиве

# Binary Search

- Бинарный поиск позволяет за $O(log\, n)$ найти ключ в отсортированном массиве
- Потребуются две разновидности бинарного поиска:

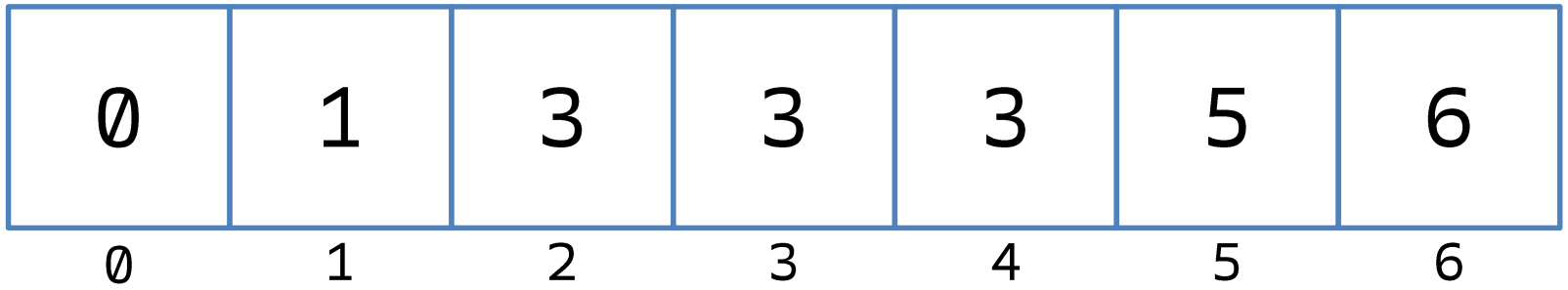
# Binary Search

- Бинарный поиск позволяет за $O(log\, n)$ найти ключ в отсортированном массиве
- Потребуются две разновидности бинарного поиска:
  1. **Lower Bound** находит «первый» элемент не меньше чем ключ, т.е. больше либо равный ключу

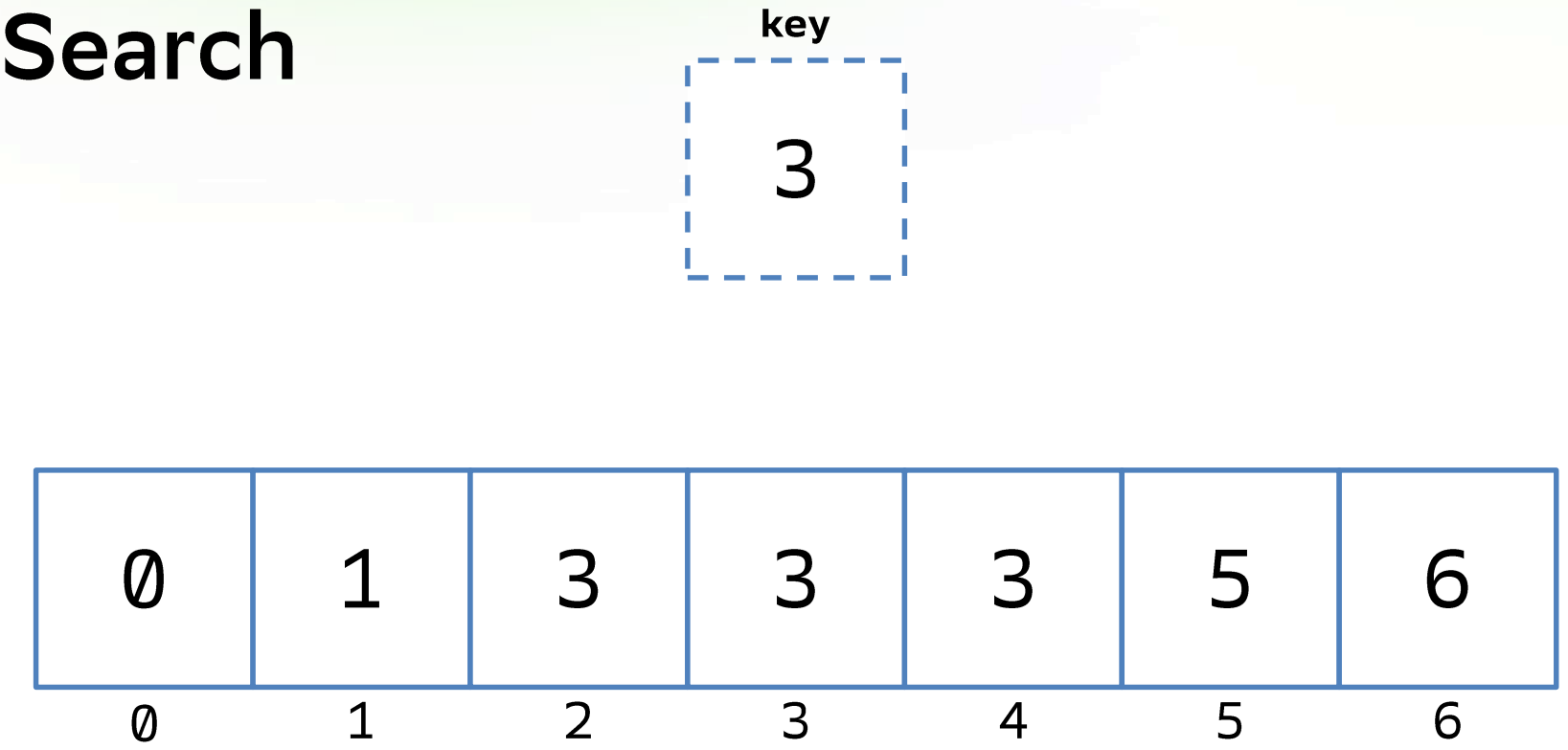
# Binary Search

- Бинарный поиск позволяет за $O(log\, n)$ найти ключ в отсортированном массиве
- Потребуются две разновидности бинарного поиска:
  1. **Lower Bound** находит «первый» элемент не меньше чем ключ, т.е. больше либо равный ключу
  2. **Upper Bound** находит «первый» элемент строго больше ключа

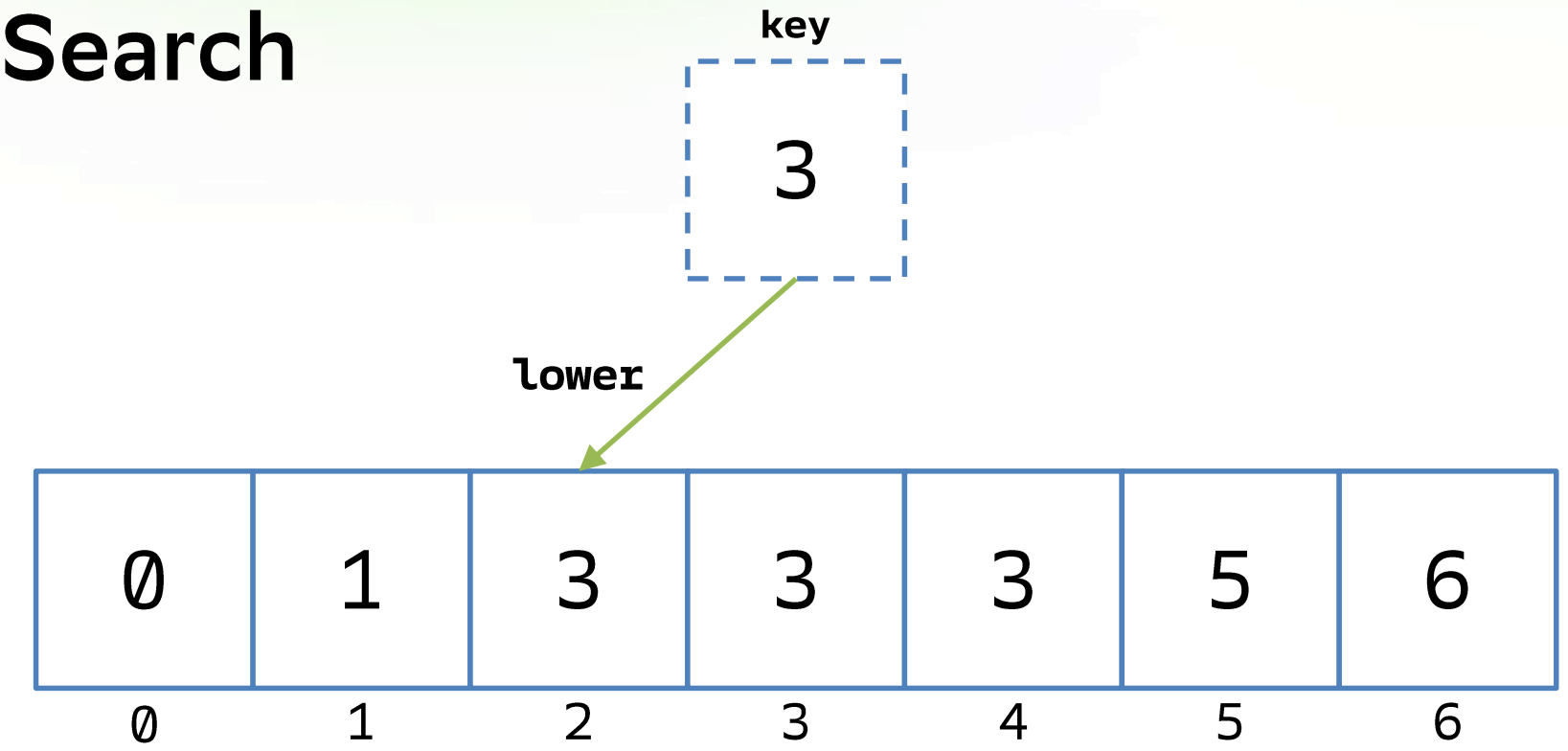
# Binary Search



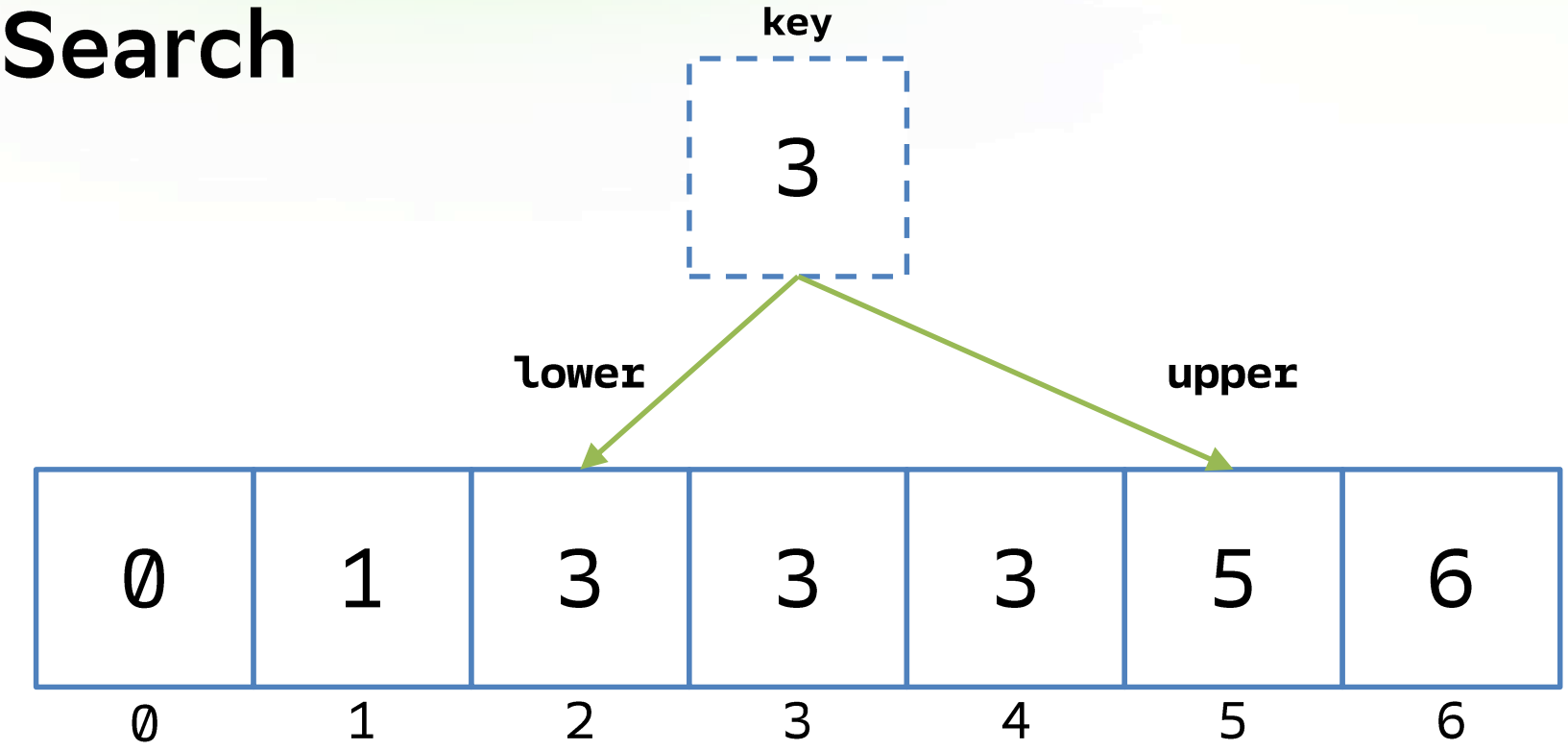
# Binary Search



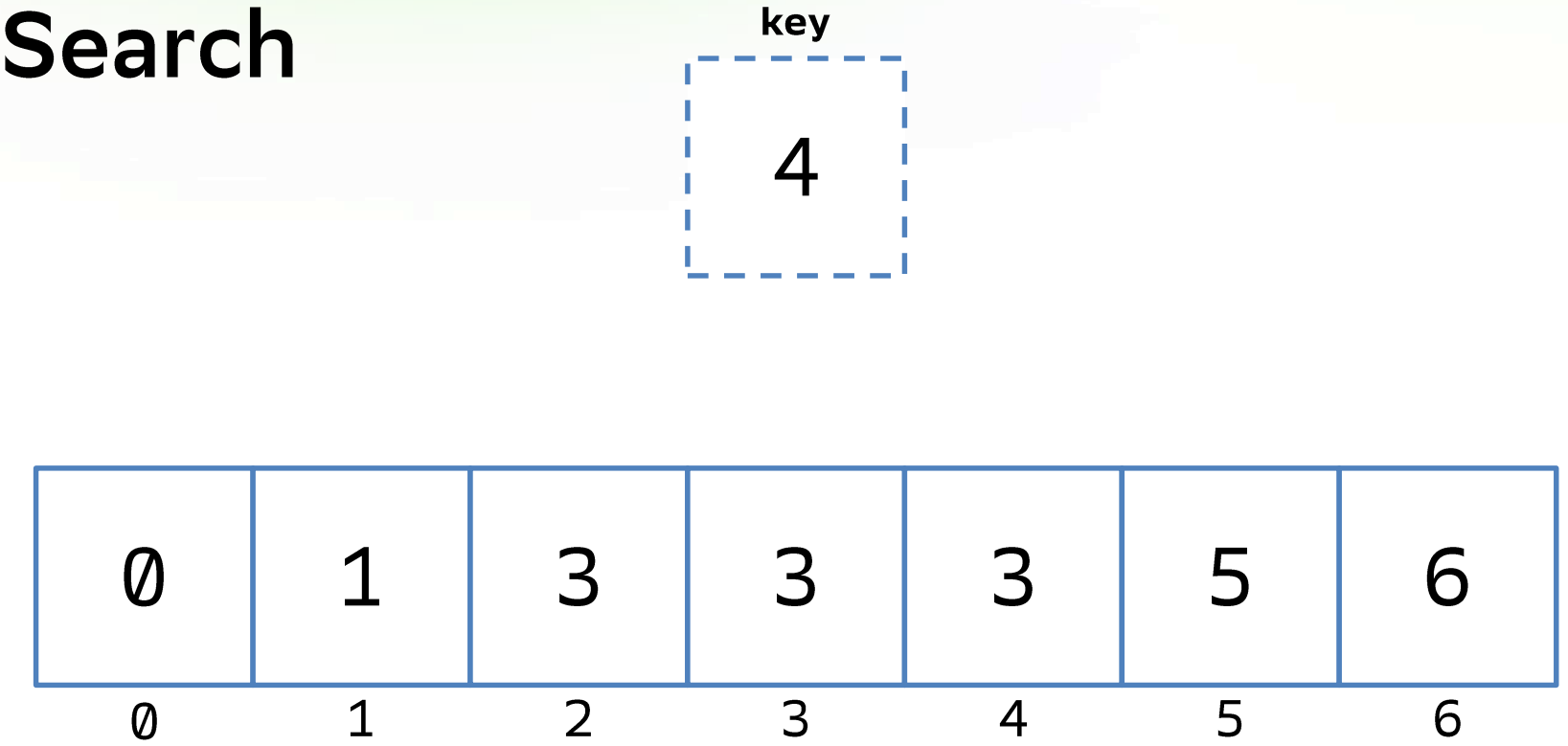
# Binary Search



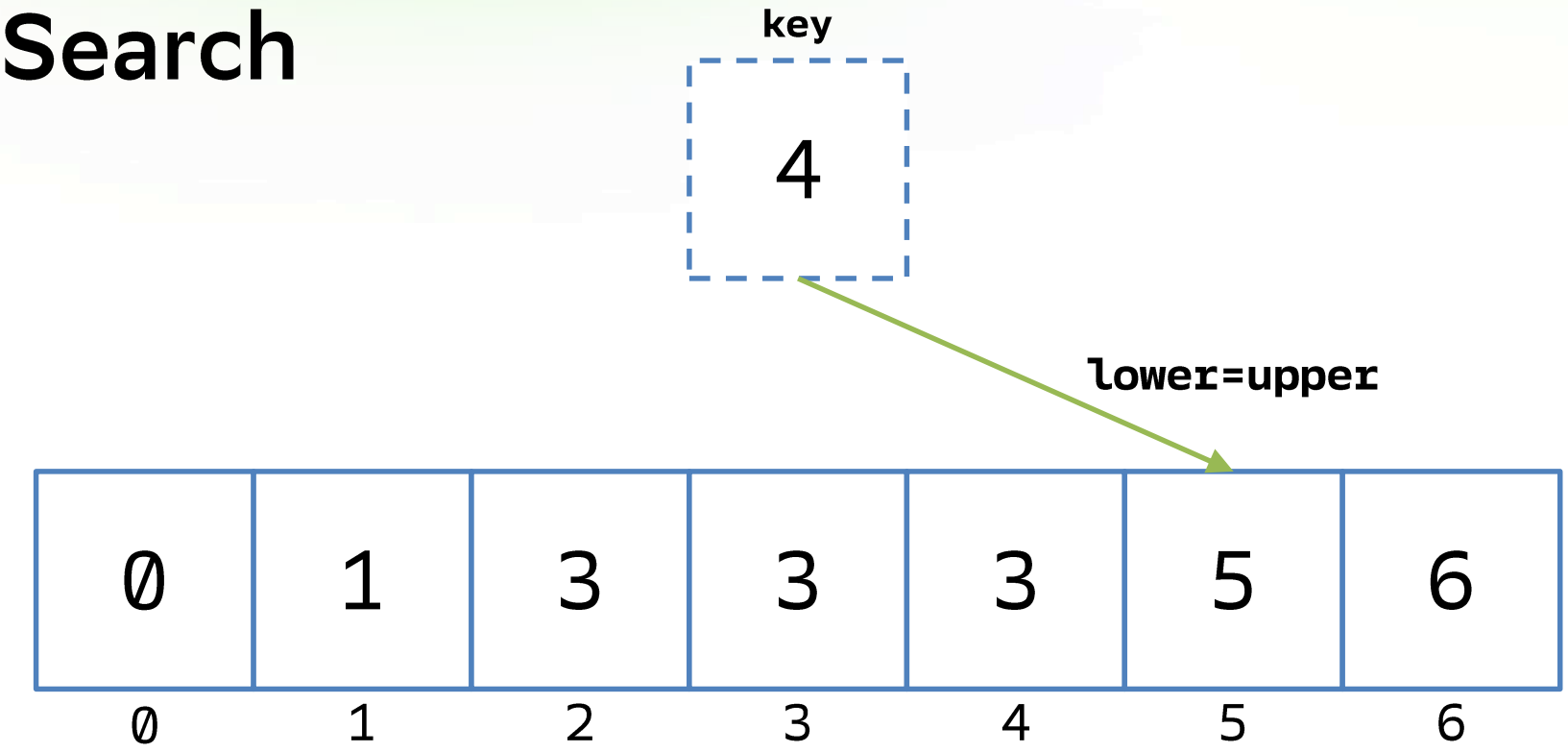
# Binary Search



# Binary Search



# Binary Search



# Rotate Merge

- Если массивы разной длины, то берем из двух тот который длиннее

# Rotate Merge

- Если массивы разной длины, то берем из двух тот который длиннее
- Берем в нём медиану и будем использовать этот элемент как «ключ» в бинарном поиске

# Rotate Merge

- Если массивы разной длины, то берем из двух тот который длиннее
- Берем в нём медиану и будем использовать этот элемент как «ключ» в бинарном поиске
- Ищем «ключ» во втором массиве:

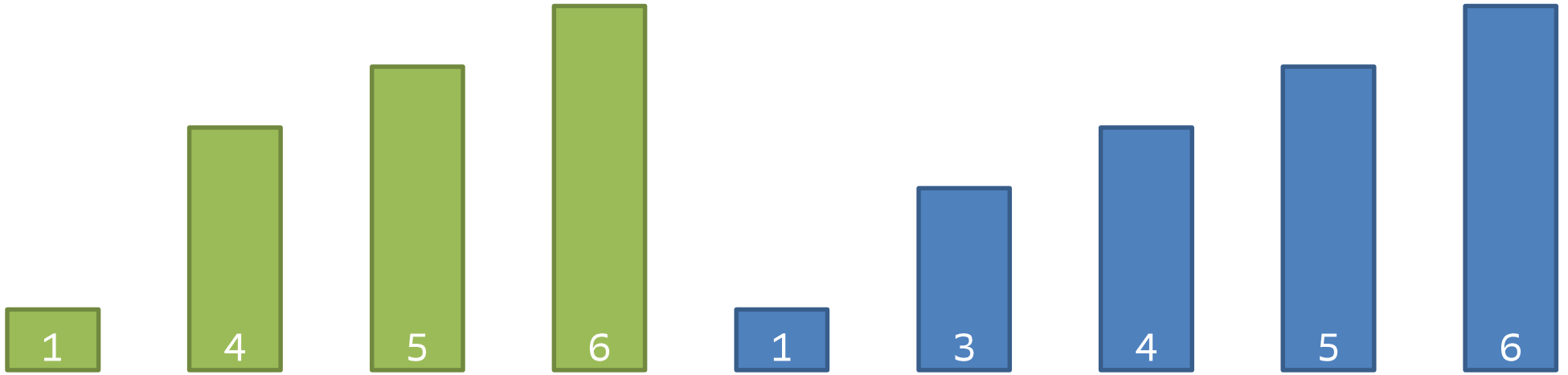
# Rotate Merge

- Если массивы разной длины, то берем из двух тот который длиннее
- Берем в нём медиану и будем использовать этот элемент как «ключ» в бинарном поиске
- Ищем «ключ» во втором массиве:
  1. Если ключ из «левого» массива, то используем **Lower Bound**

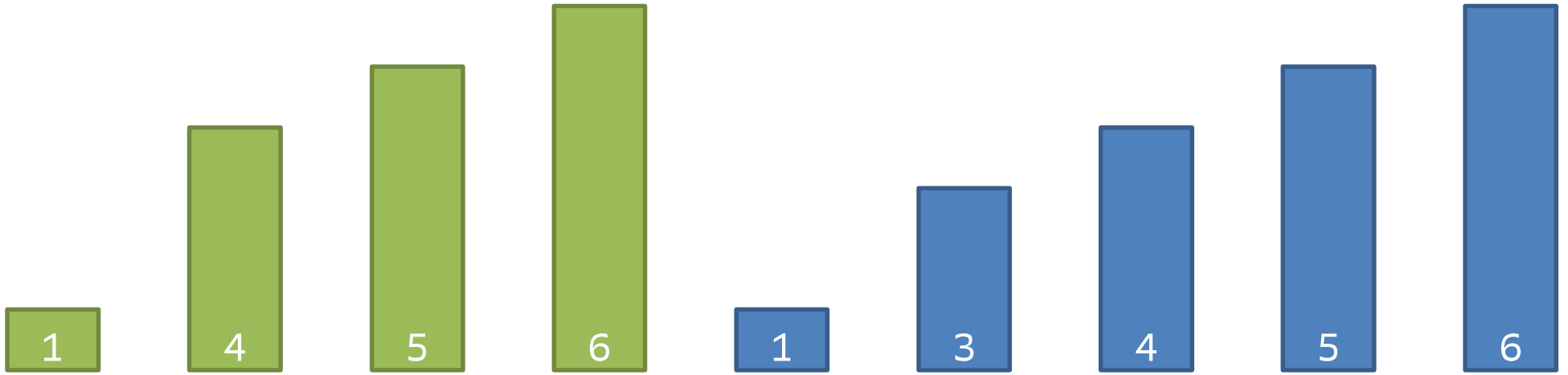
# Rotate Merge

- Если массивы разной длины, то берем из двух тот который длиннее
- Берем в нём медиану и будем использовать этот элемент как «ключ» в бинарном поиске
- Ищем «ключ» во втором массиве:
  1. Если ключ из «левого» массива, то используем **Lower Bound**
  2. Если ключ из «правого» массива, то используем **Upper Bound**

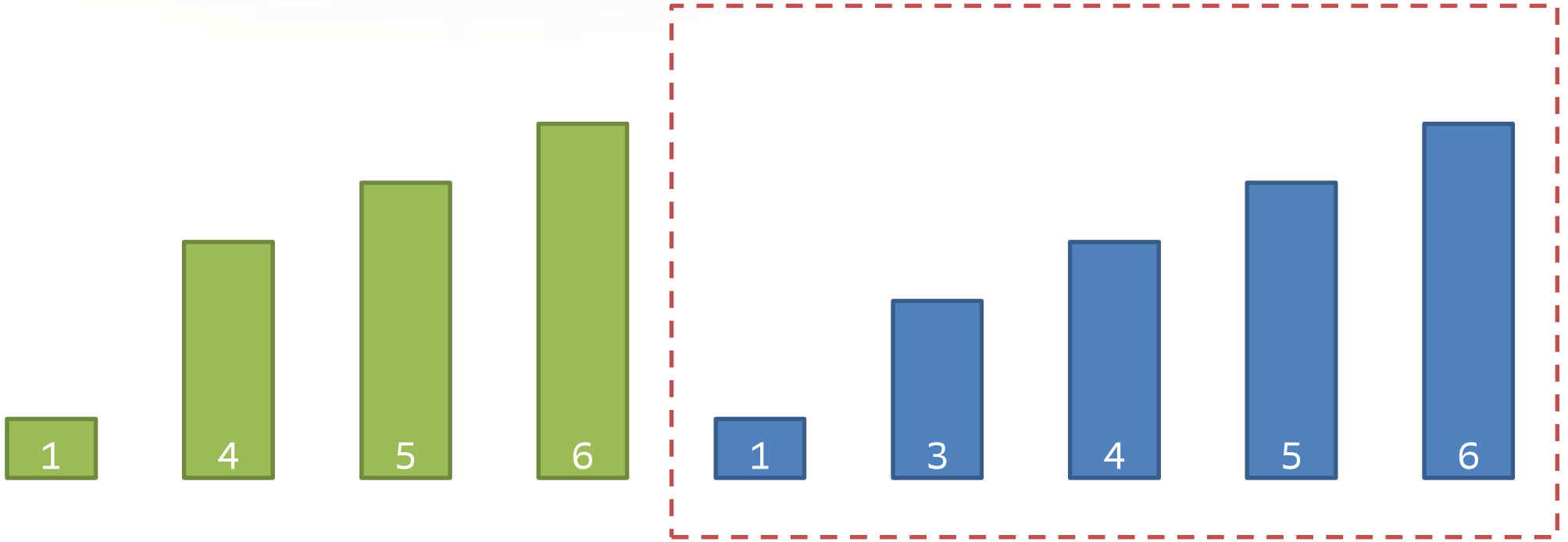
# Rotate Merge



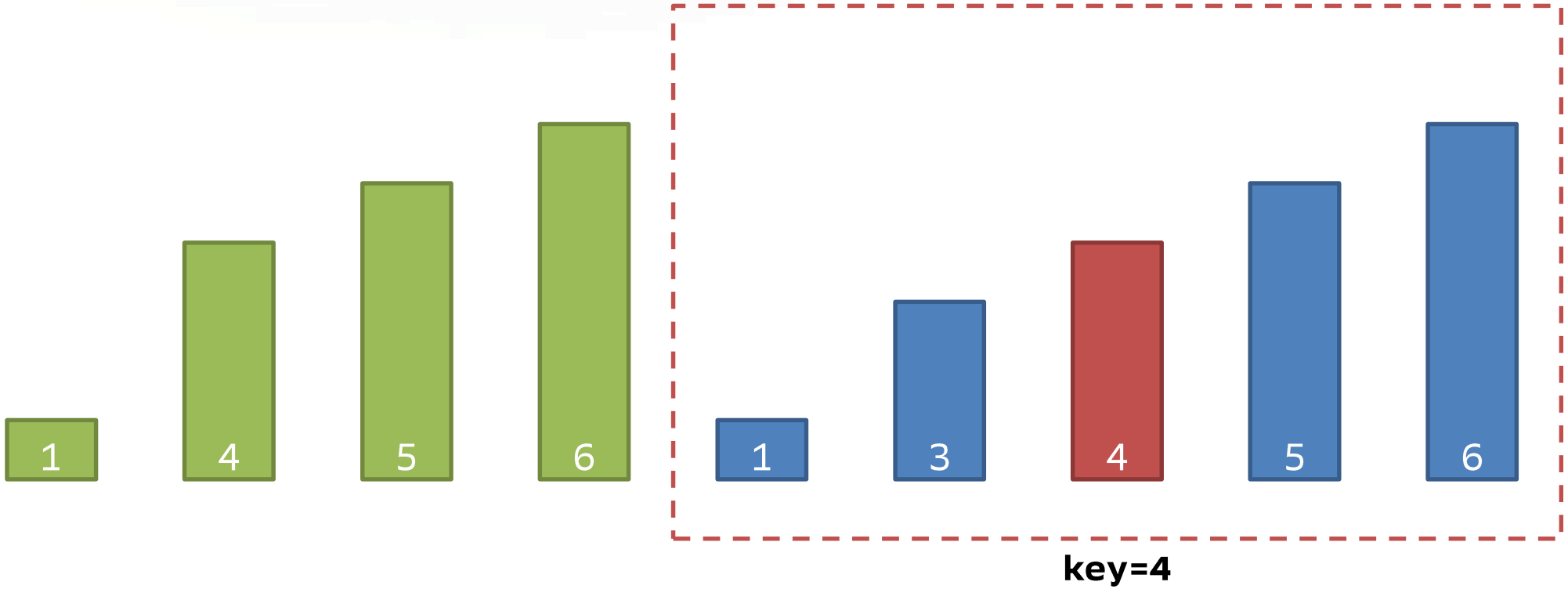
# Rotate Merge



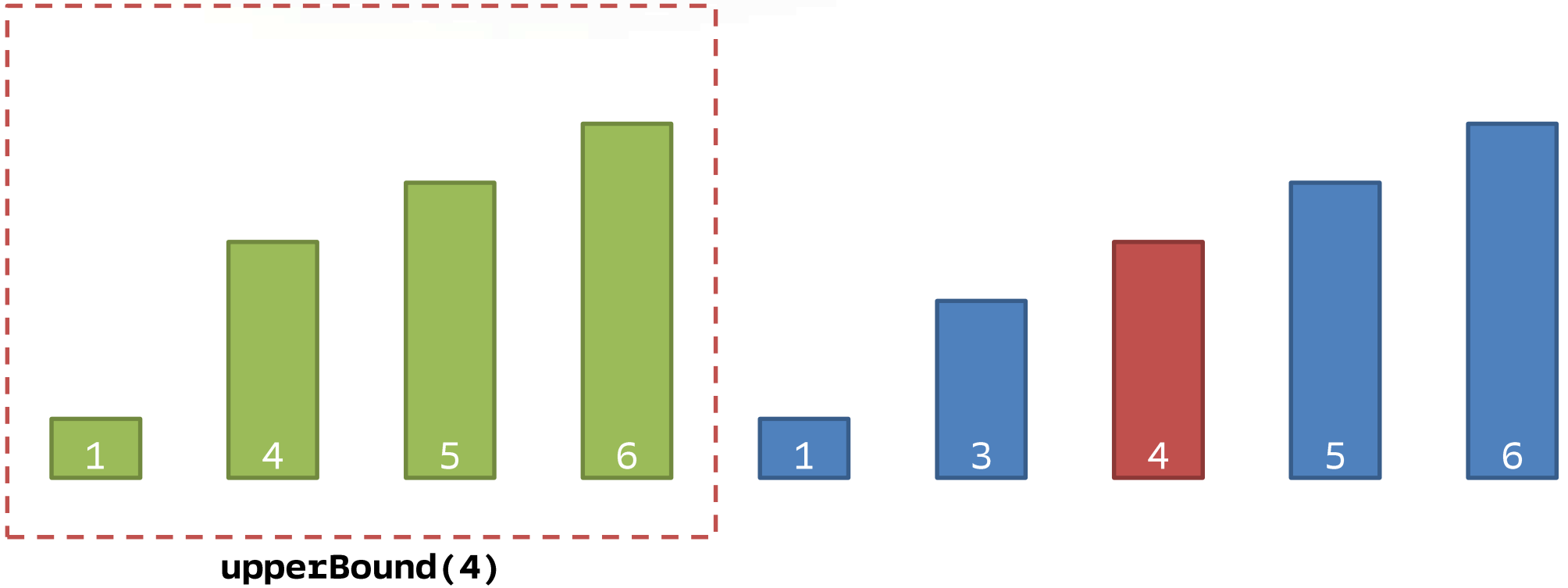
# Rotate Merge



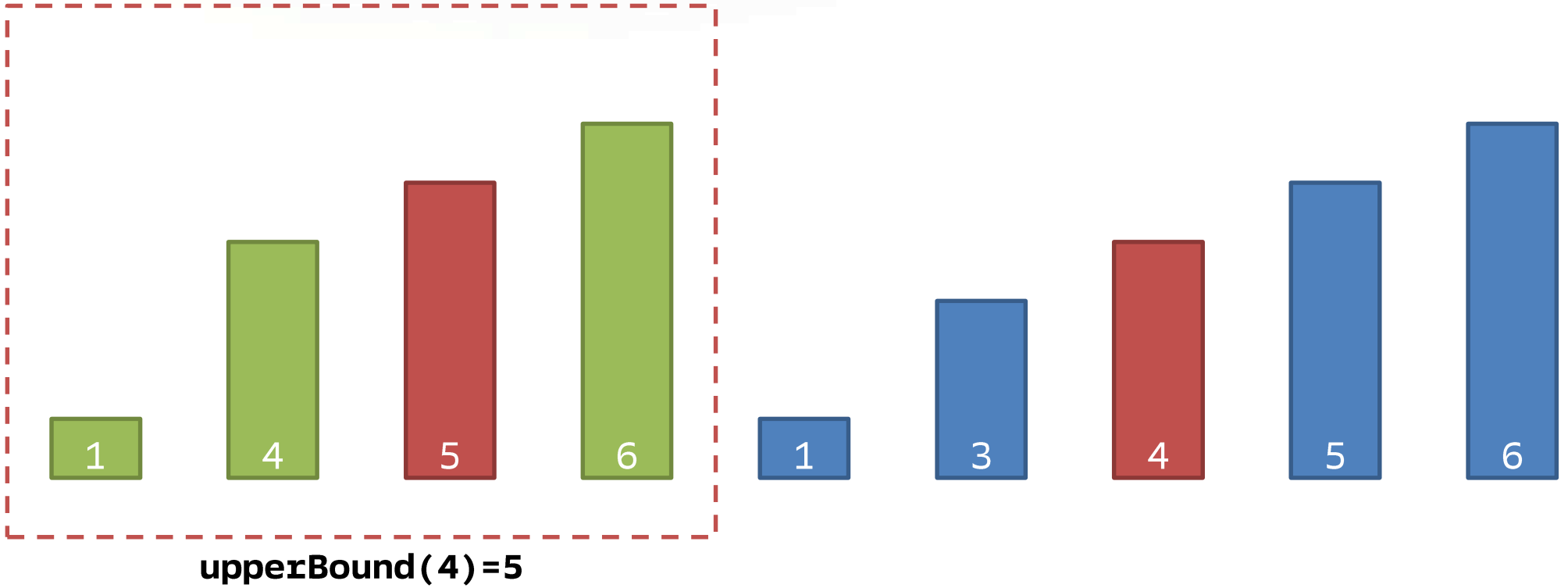
# Rotate Merge



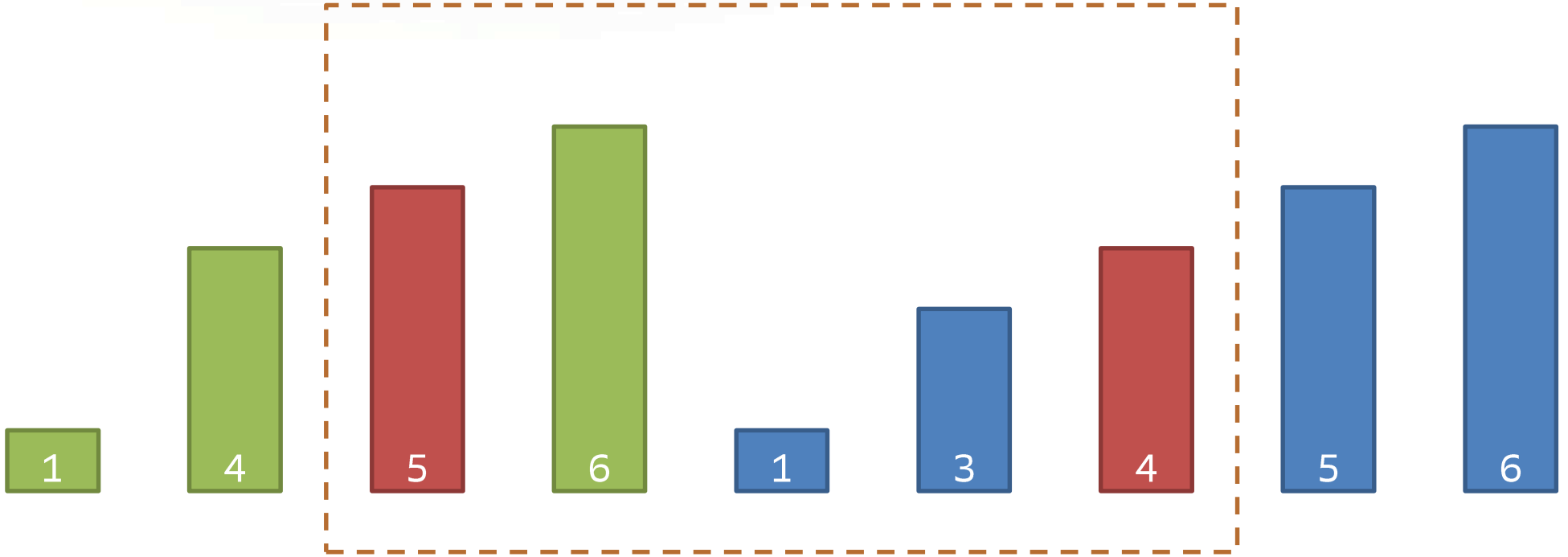
# Rotate Merge



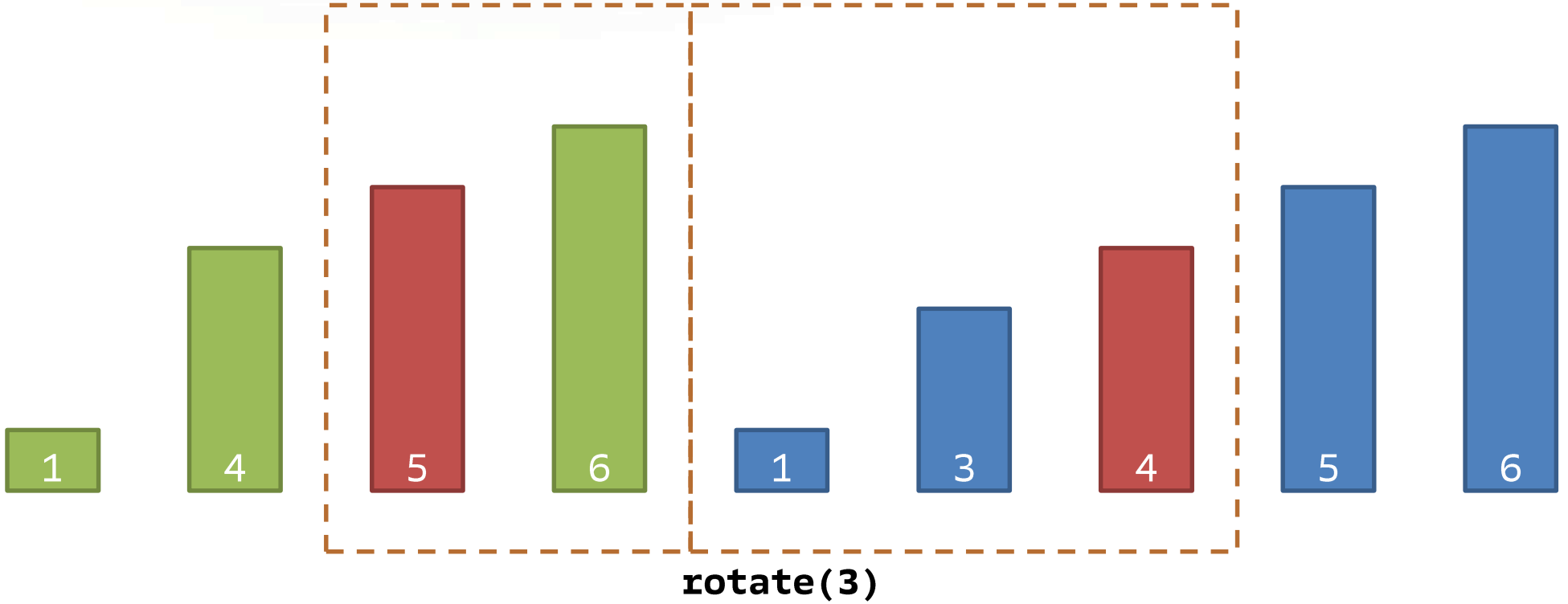
# Rotate Merge



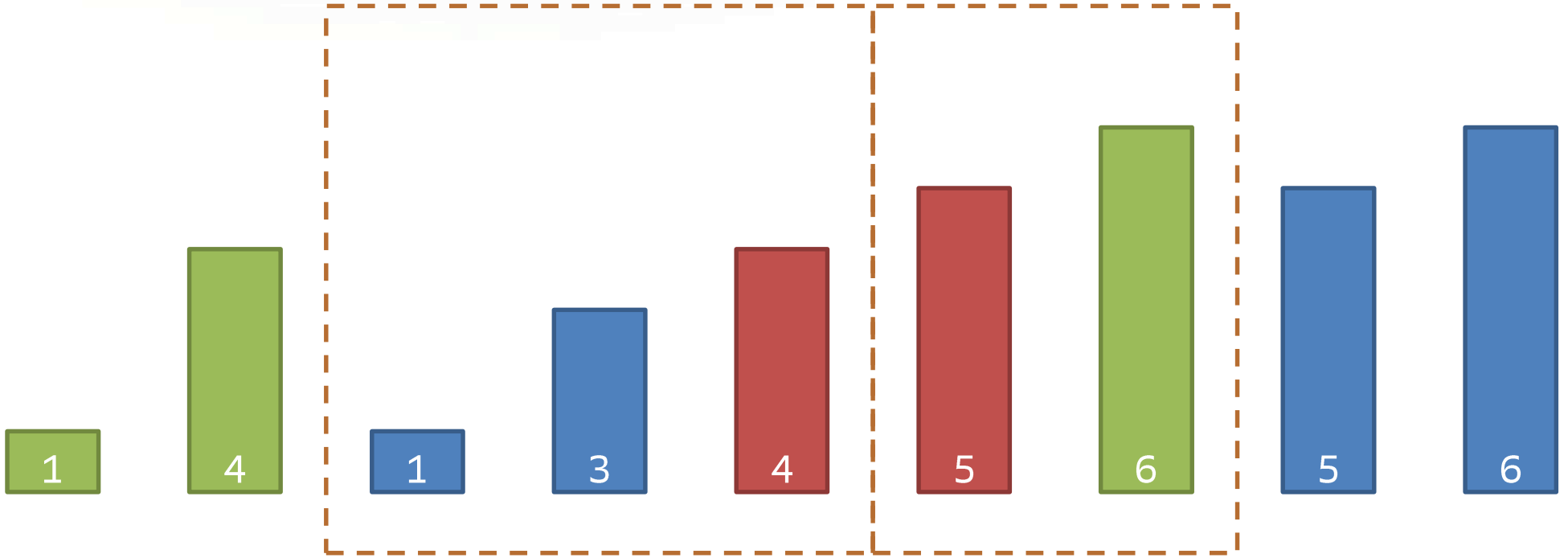
# Rotate Merge



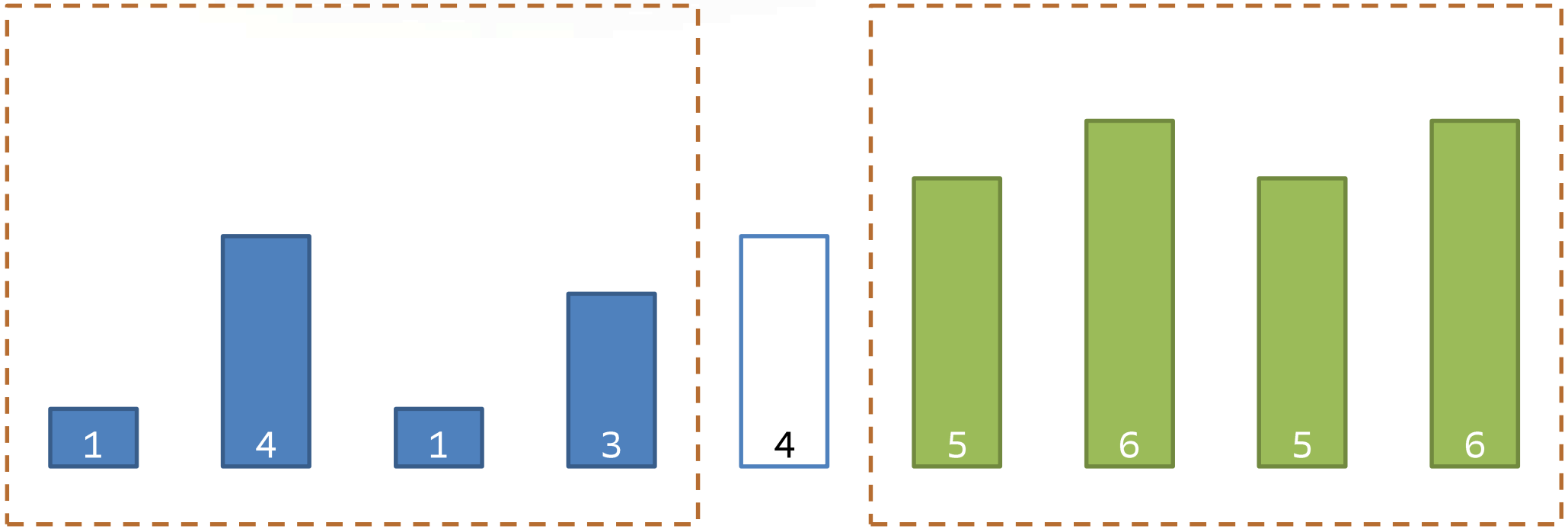
# Rotate Merge



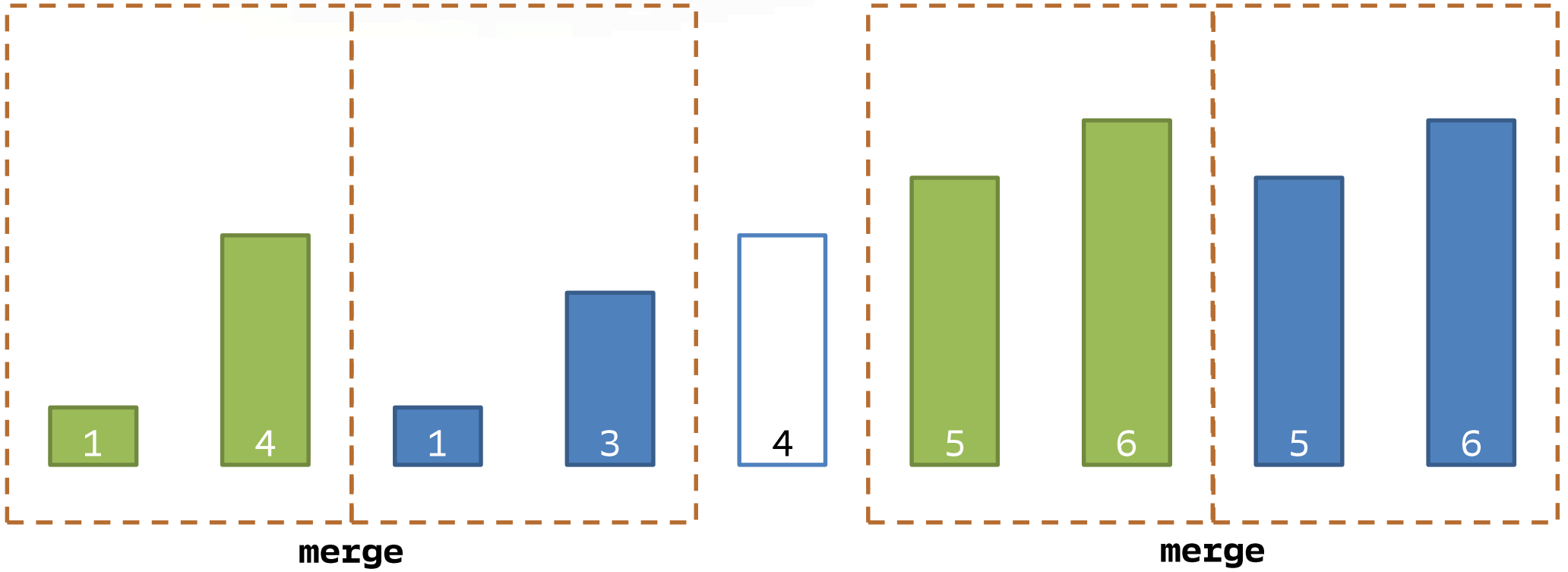
# Rotate Merge



# Rotate Merge



# Rotate Merge



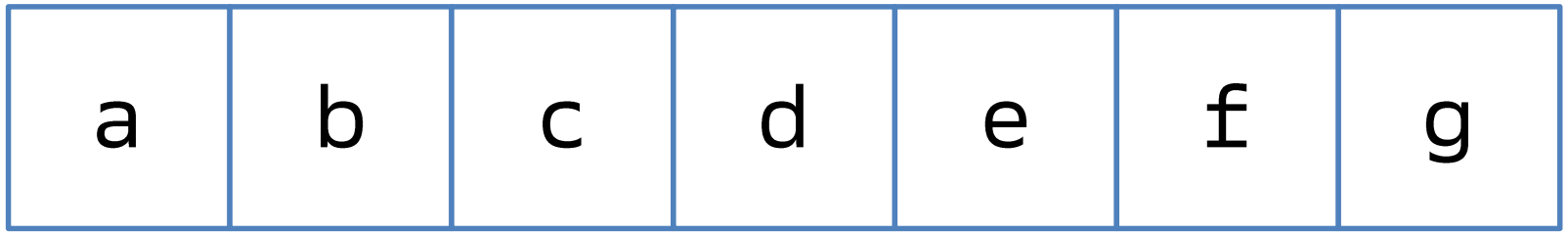
# Rotate

- Реализован в стандартной библиотеке **Java**, **Python**, **C++**

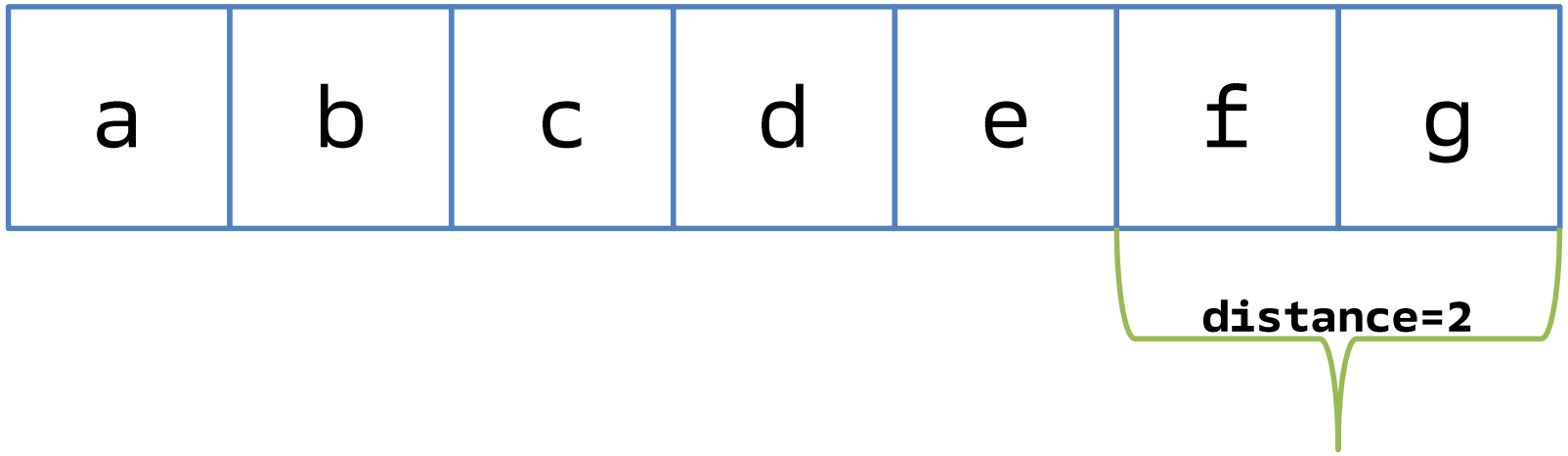
# Rotate

- Реализован в стандартной библиотеке **Java**, **Python**, **C++**
- На **LeetCode** есть задача **«Rotate Array»** и её могу предложить на первый этап собеседования

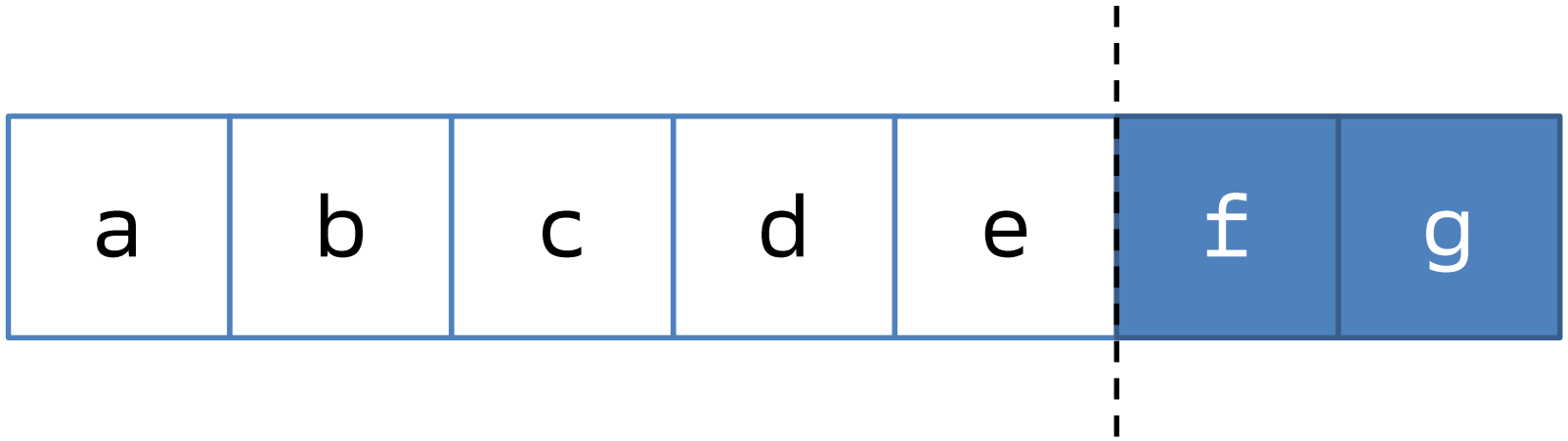
# Rotate



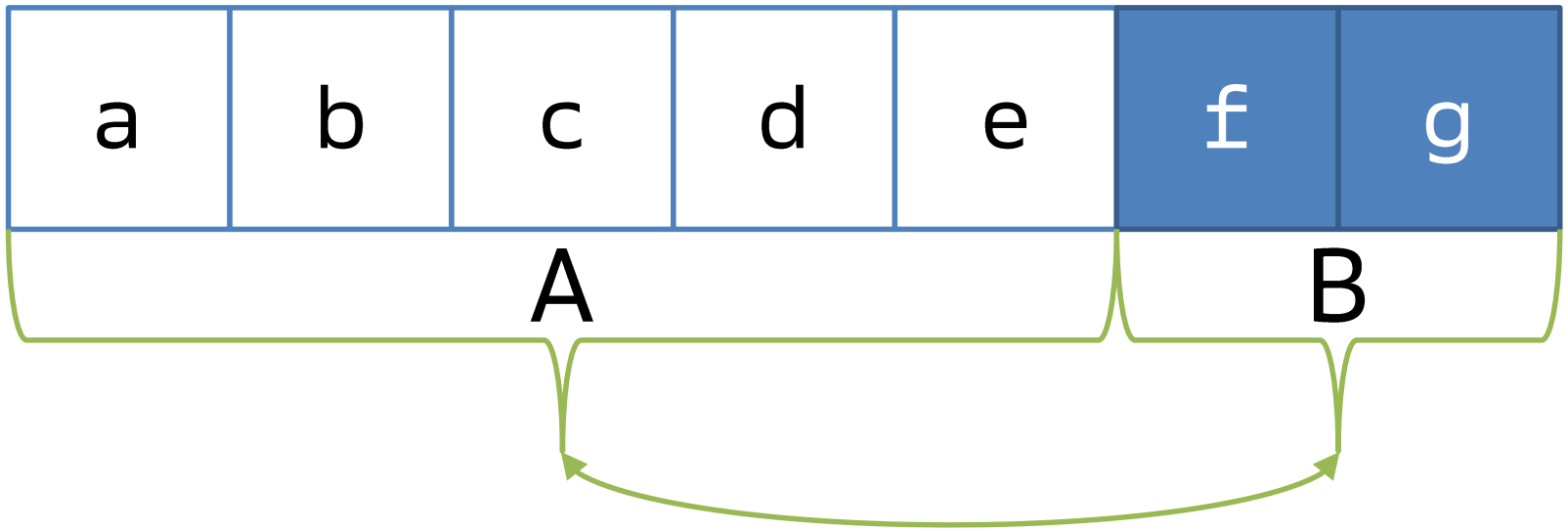
# Rotate



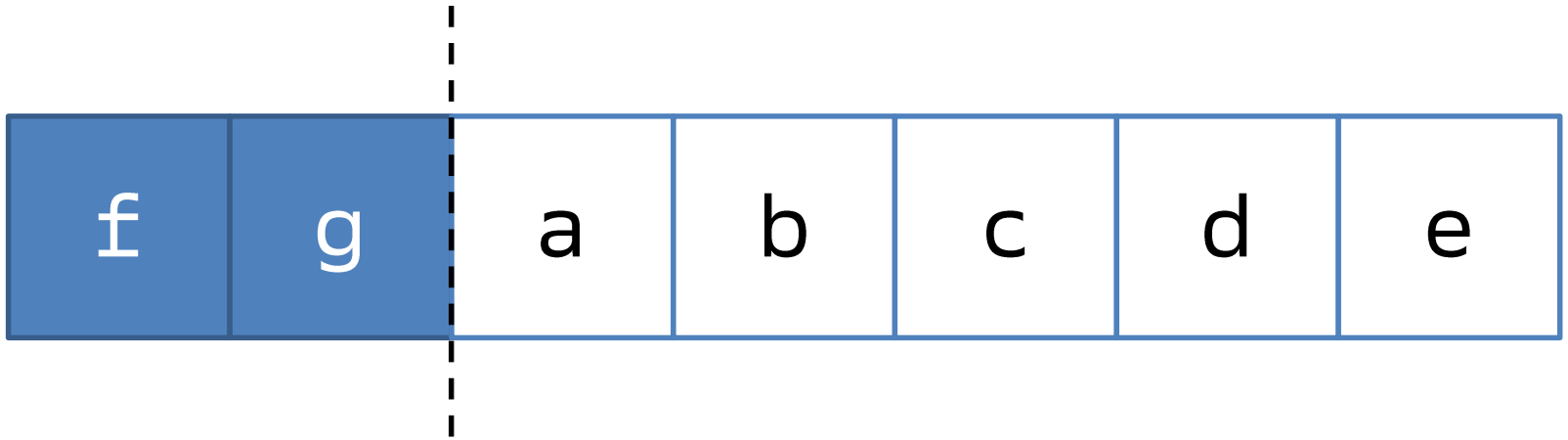
# Rotate



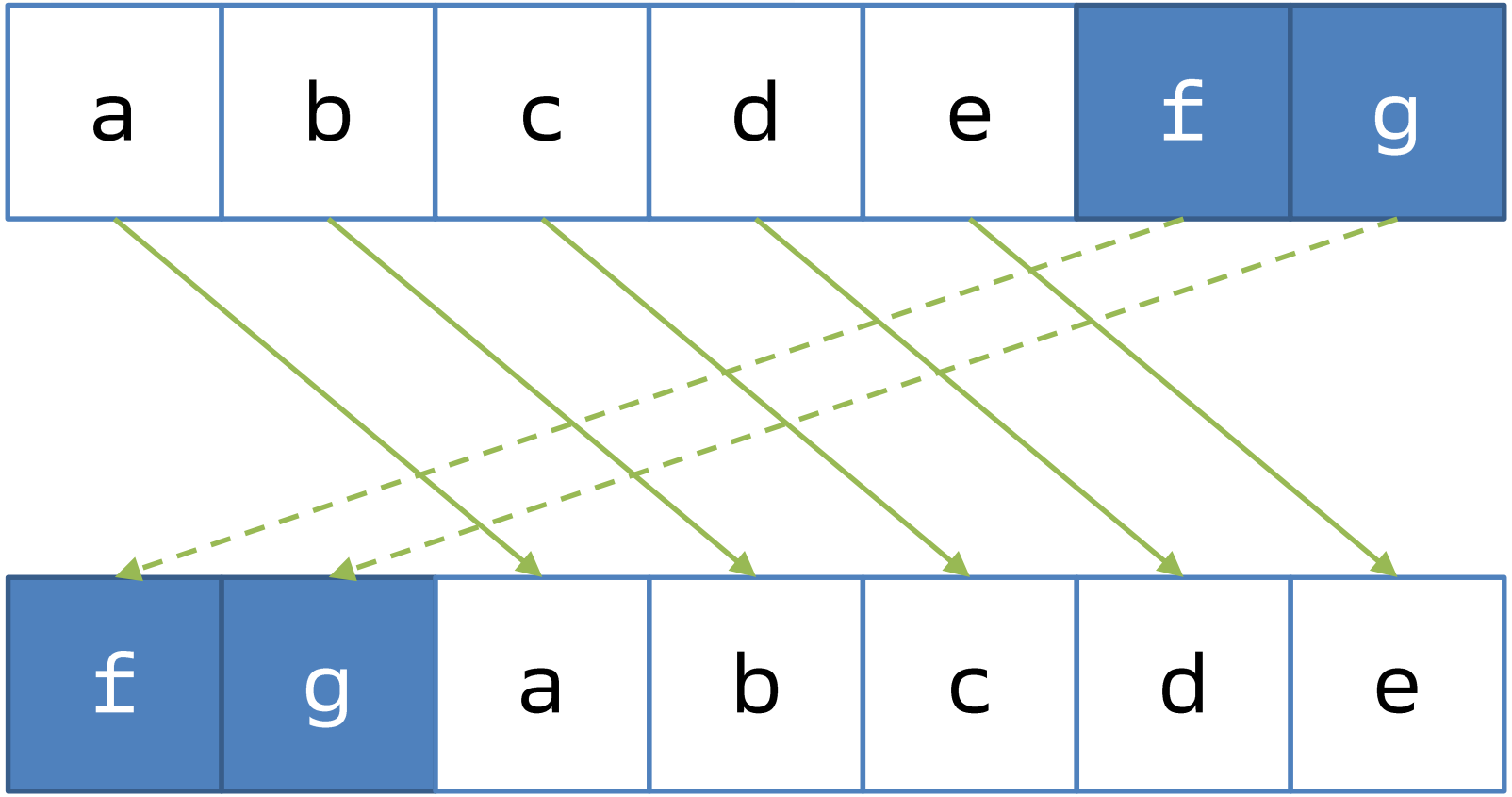
# Rotate



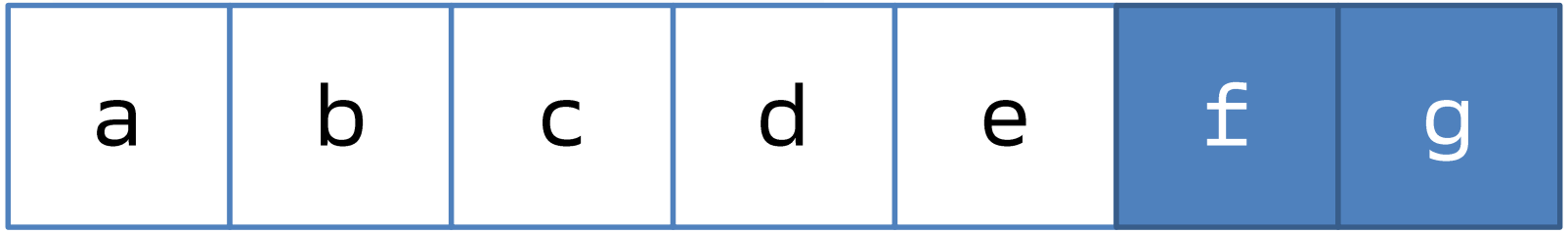
# Rotate

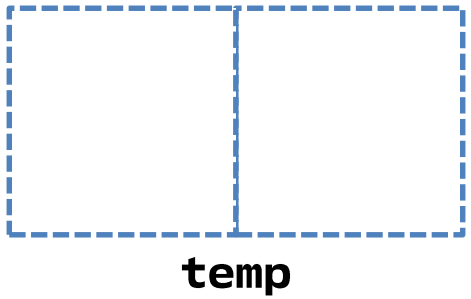


# Rotate

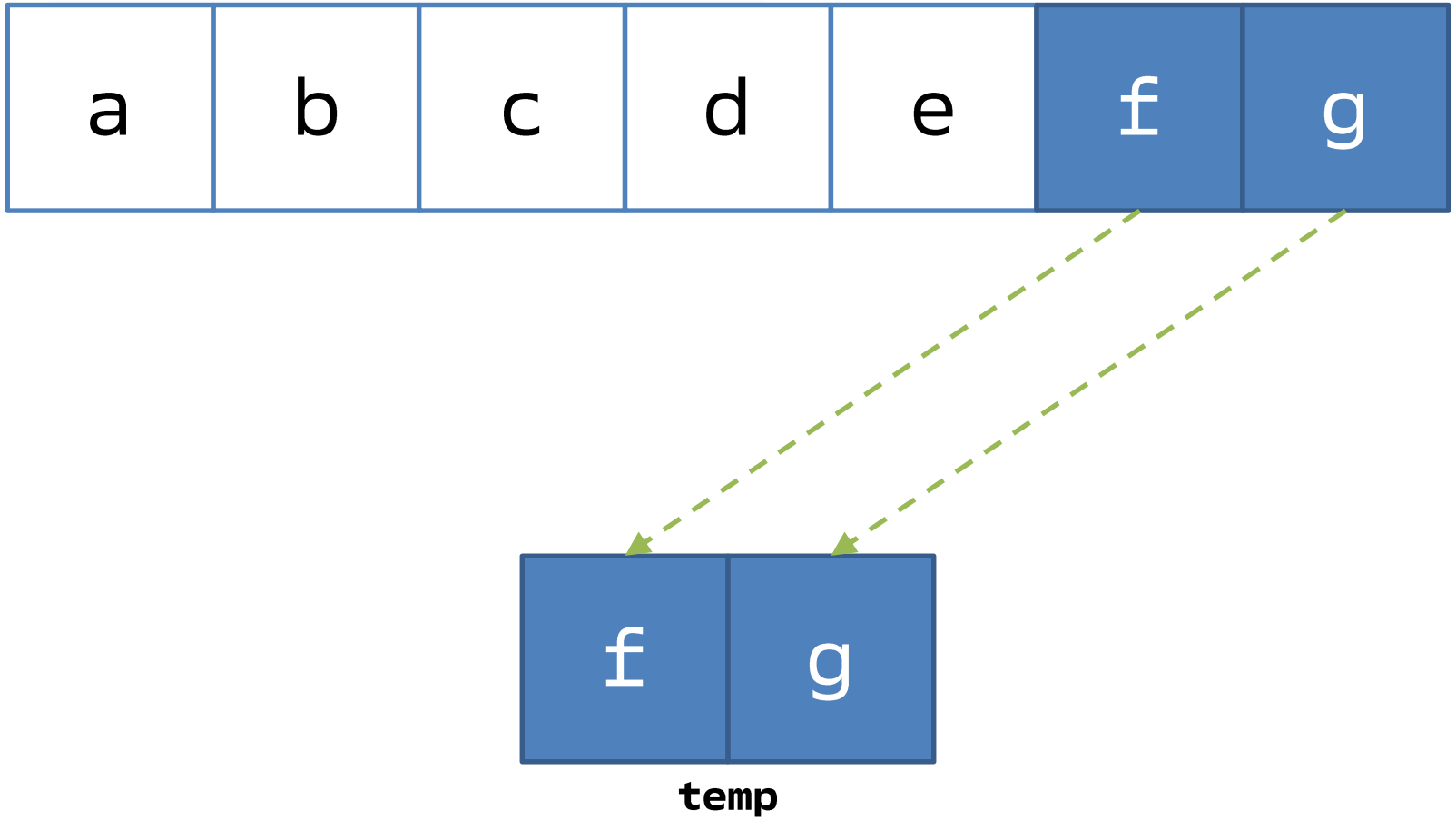


# С дополнительной памятью

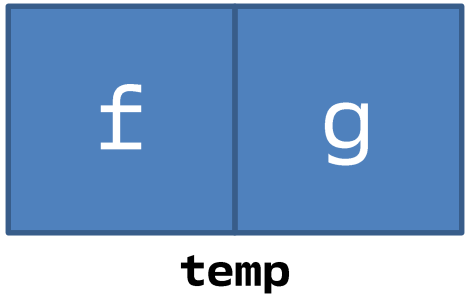


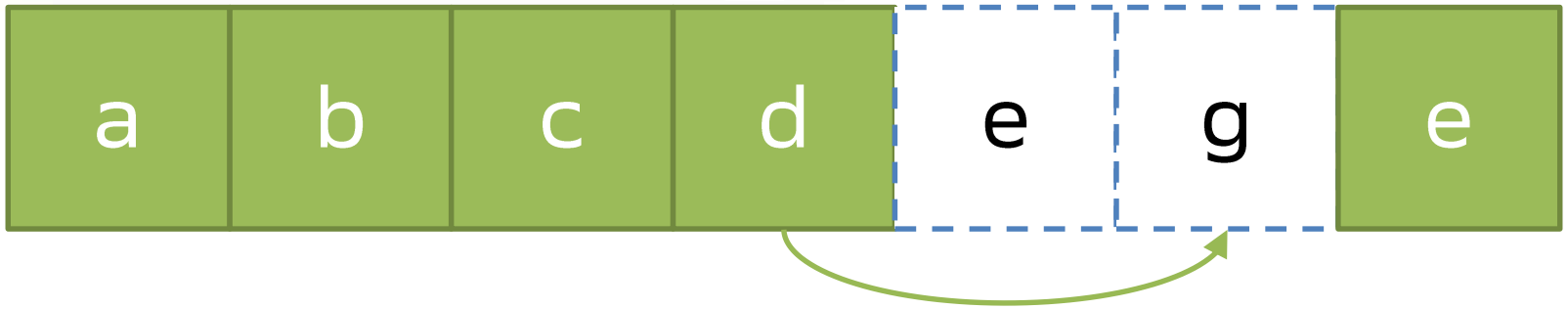
# С дополнительной памятью

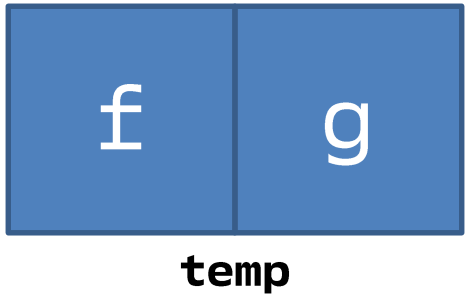


# С дополнительной памятью

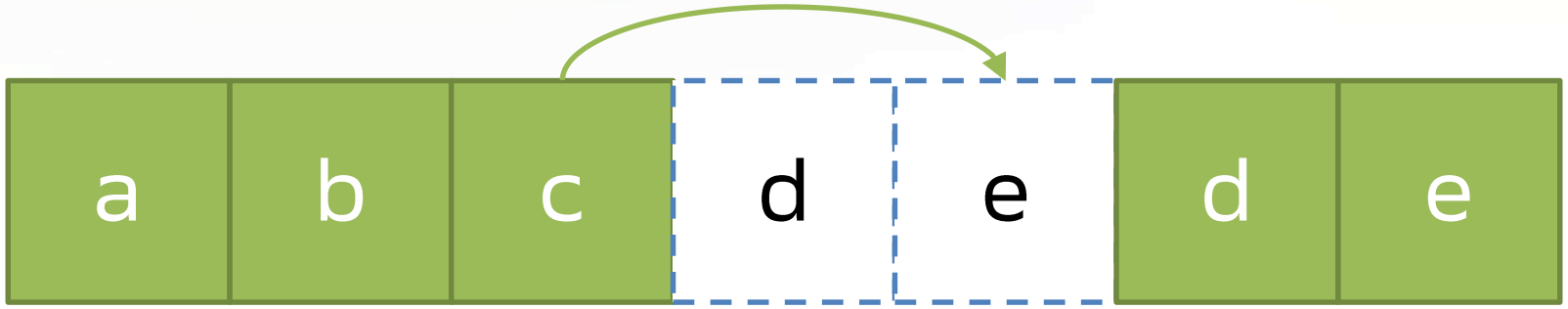



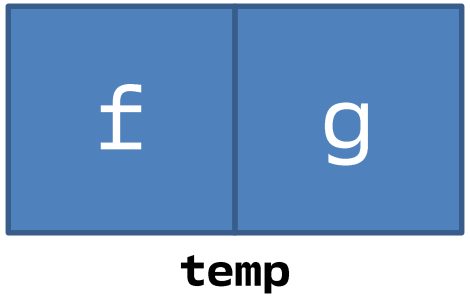
# С дополнительной памятью



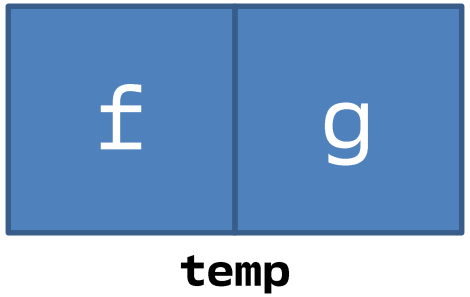



# С дополнительной памятью

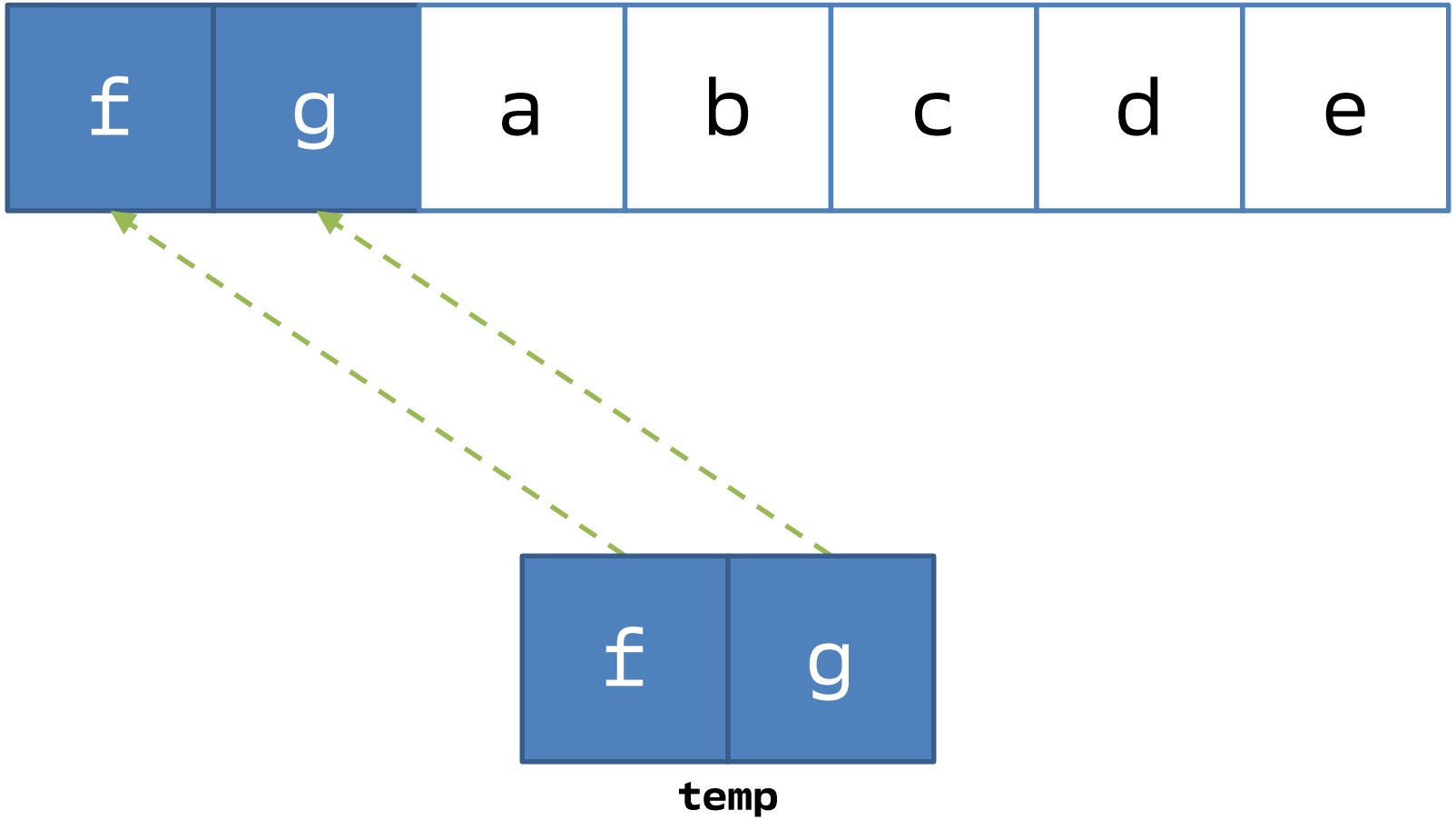


# С дополнительной памятью

# С дополнительной памятью



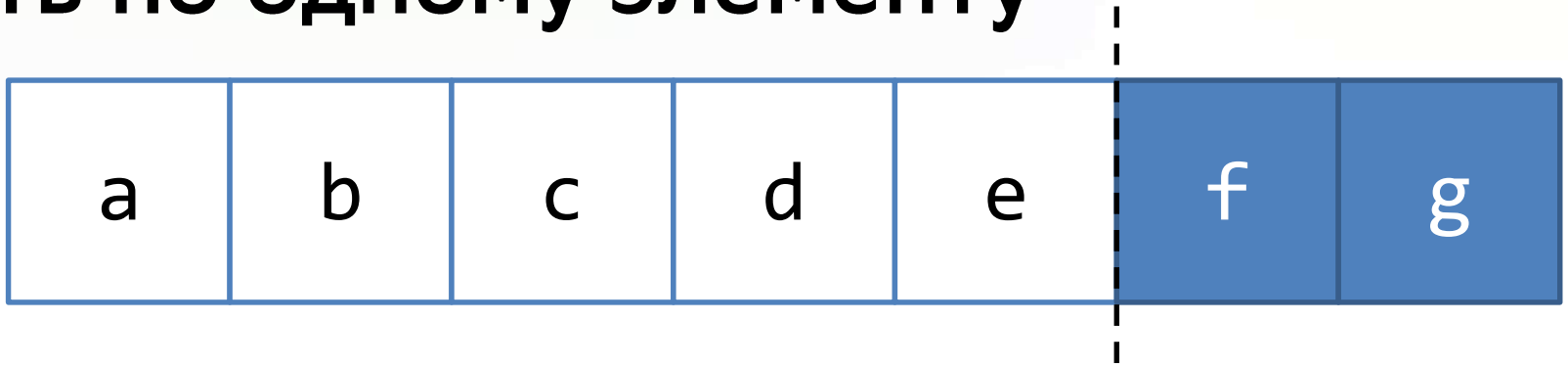
# С дополнительной памятью

- $O(n)$ памяти в худшем случае

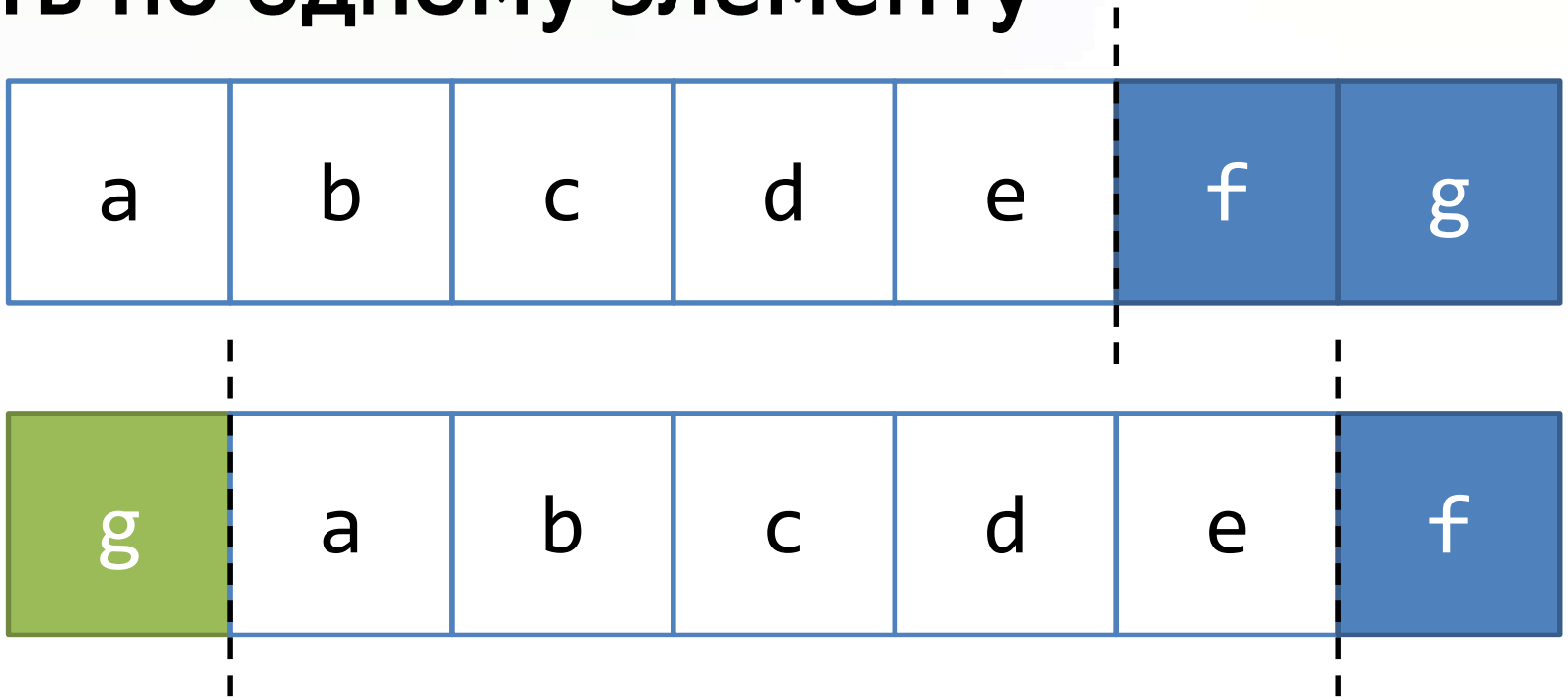
# С дополнительной памятью

- $O(n)$ памяти в худшем случае
- Но можно использовать буфер фиксированной длины (независимой от $n$)

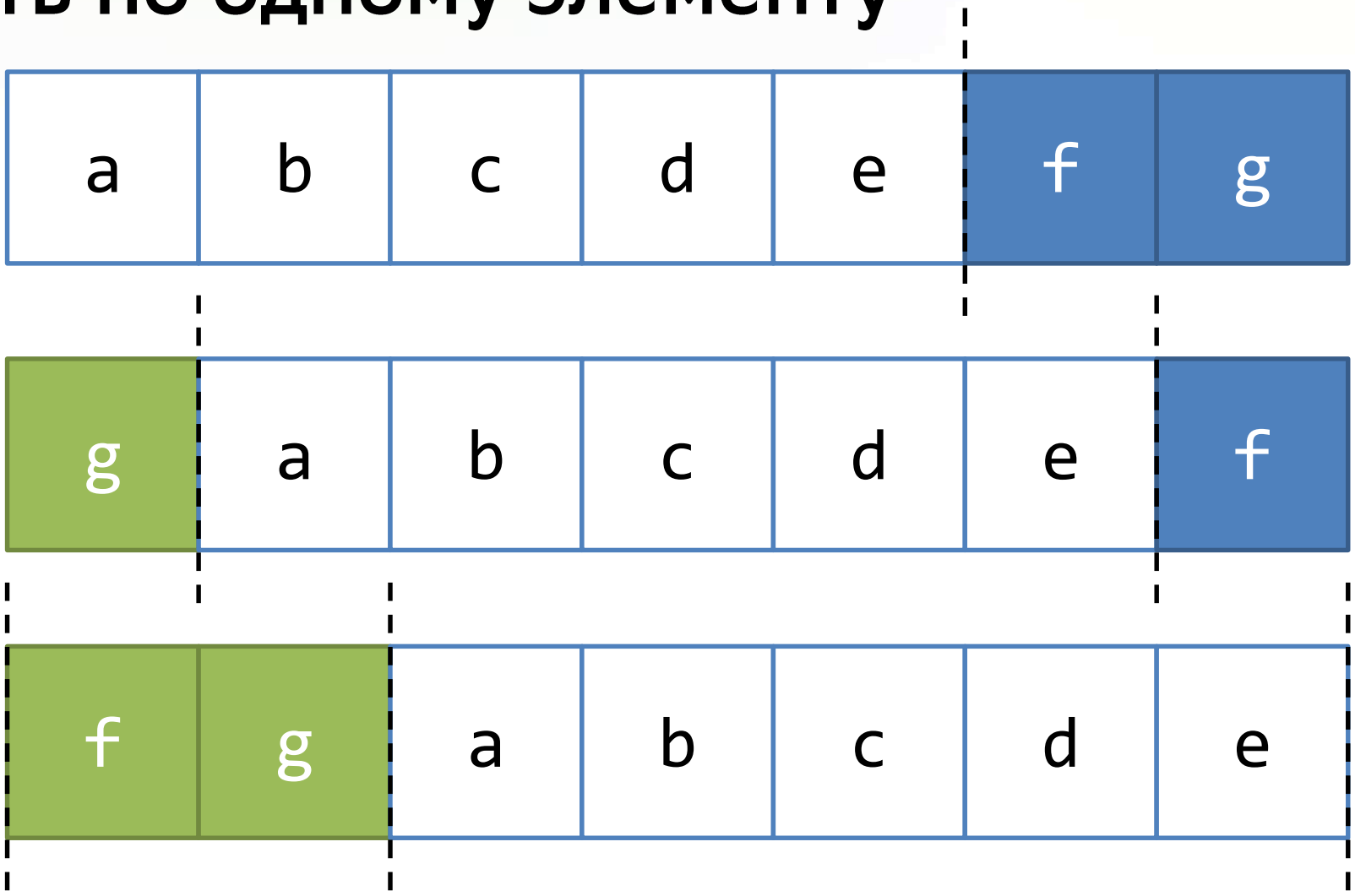
# Сдвигать по одному элементу



# Сдвигать по одному элементу



# Сдвигать по одному элементу



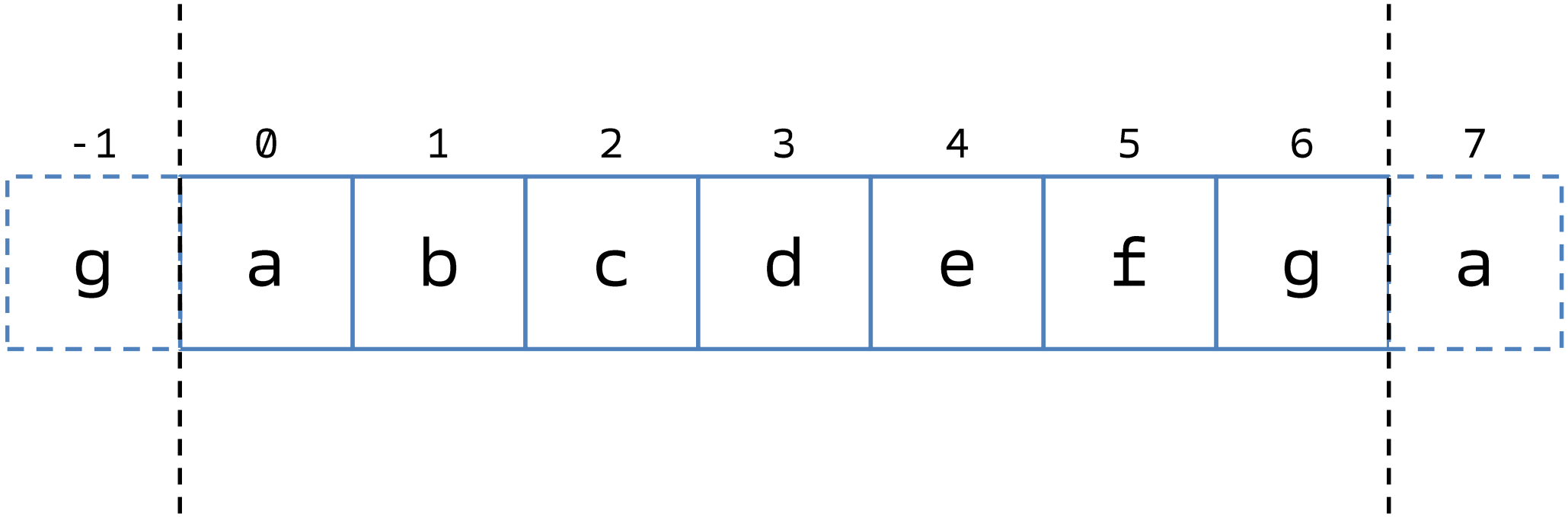
# Сдвигать по одному элементу

- Переставить один элемент с конца в начало занимает $O(n)$ времени в случае `ArrayList`

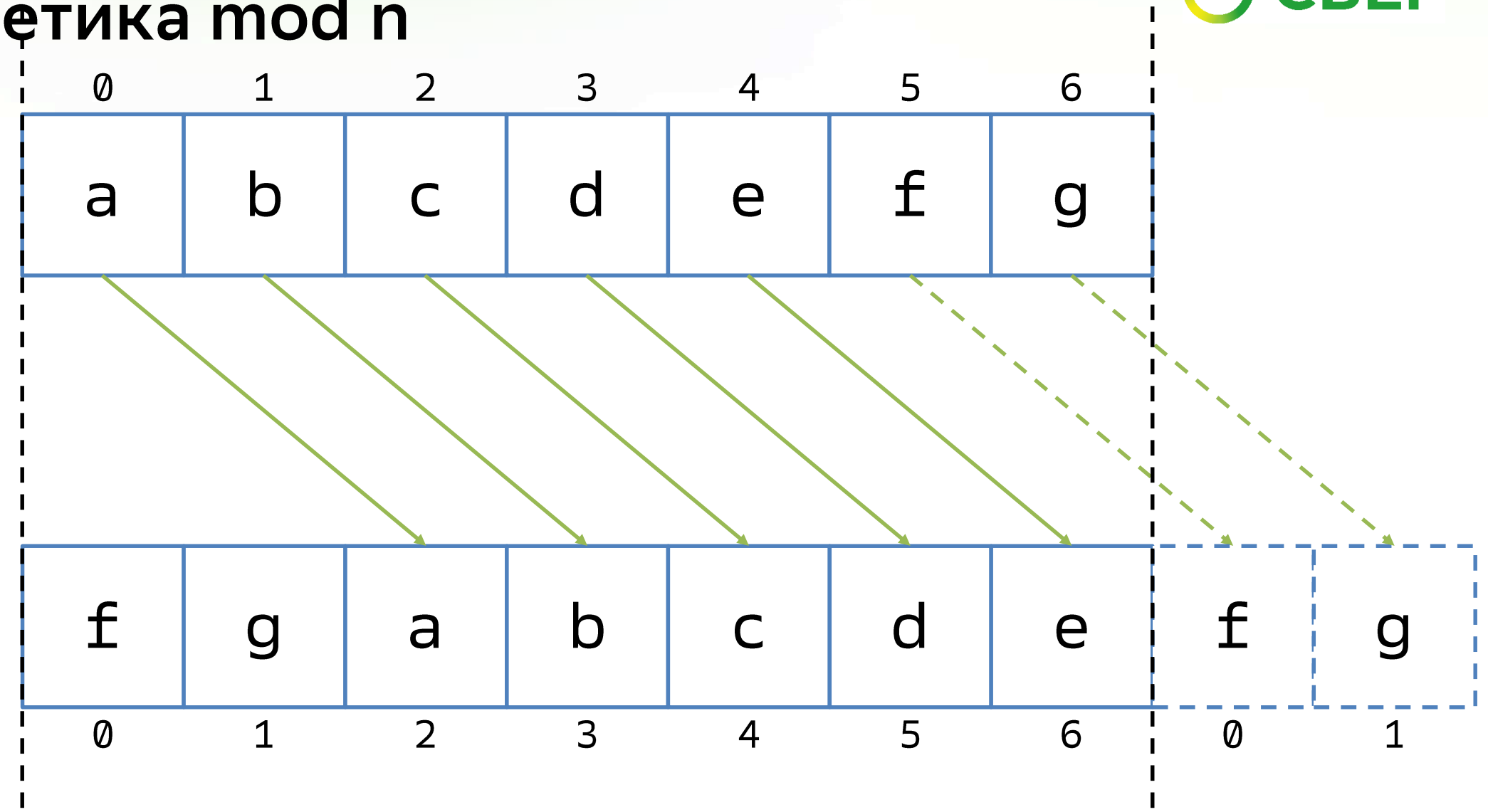
# Сдвигать по одному элементу

- Переставить один элемент с конца в начало занимает $O(n)$ времени в случае `ArrayList`
- В худшем случае сдвиг будет пропорционален $n$, и тогда получаем $O(n^2)$ времени

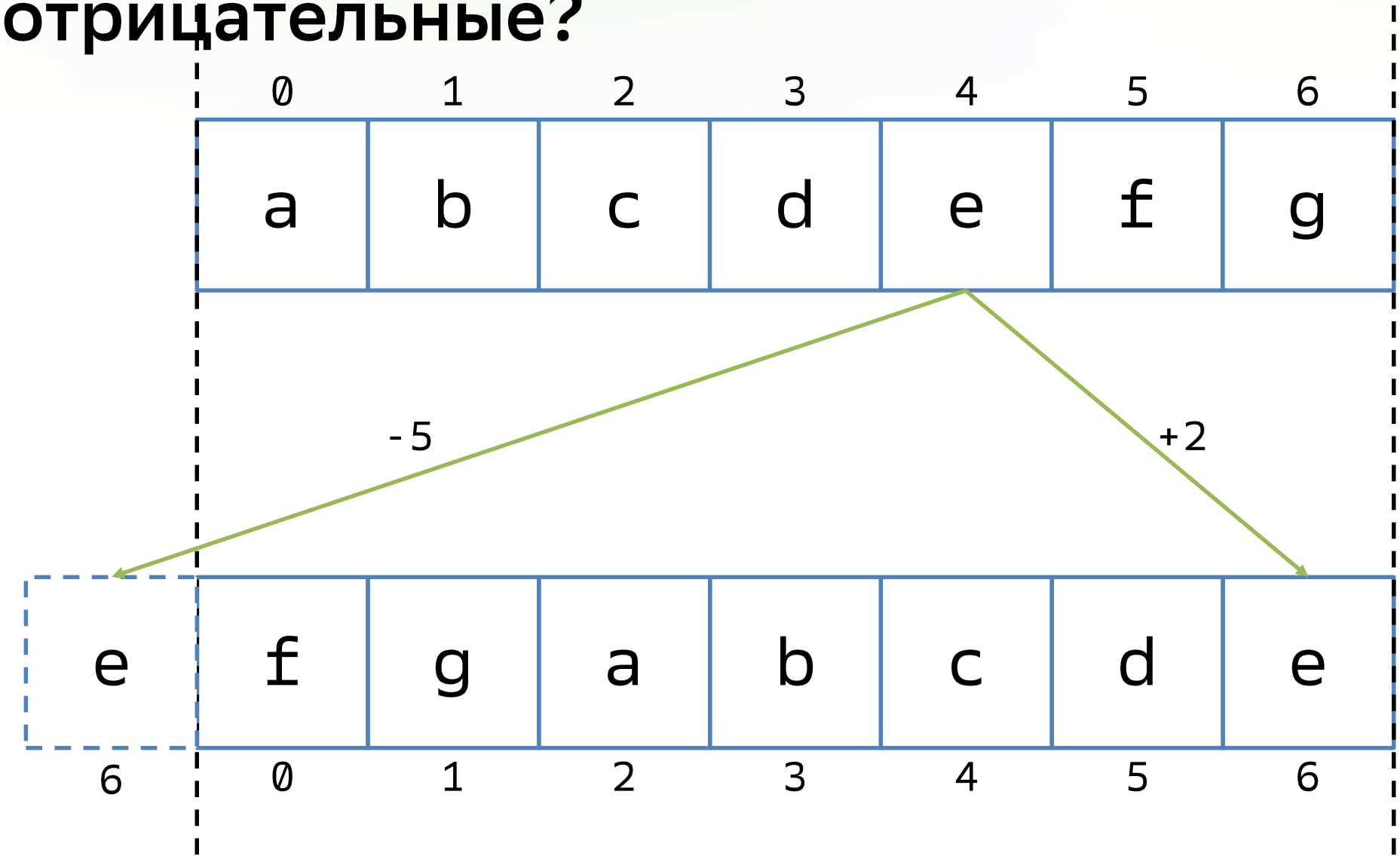
# Circular array



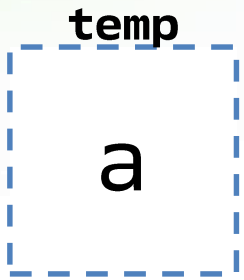
# Арифметика mod n

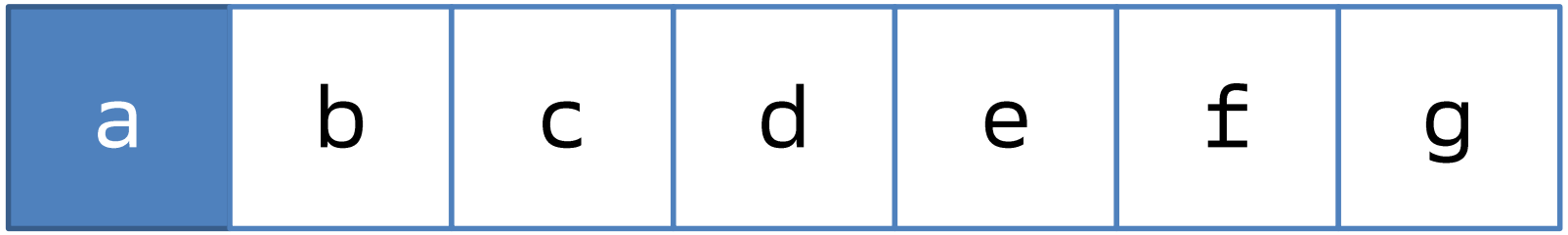


# А отрицательные?

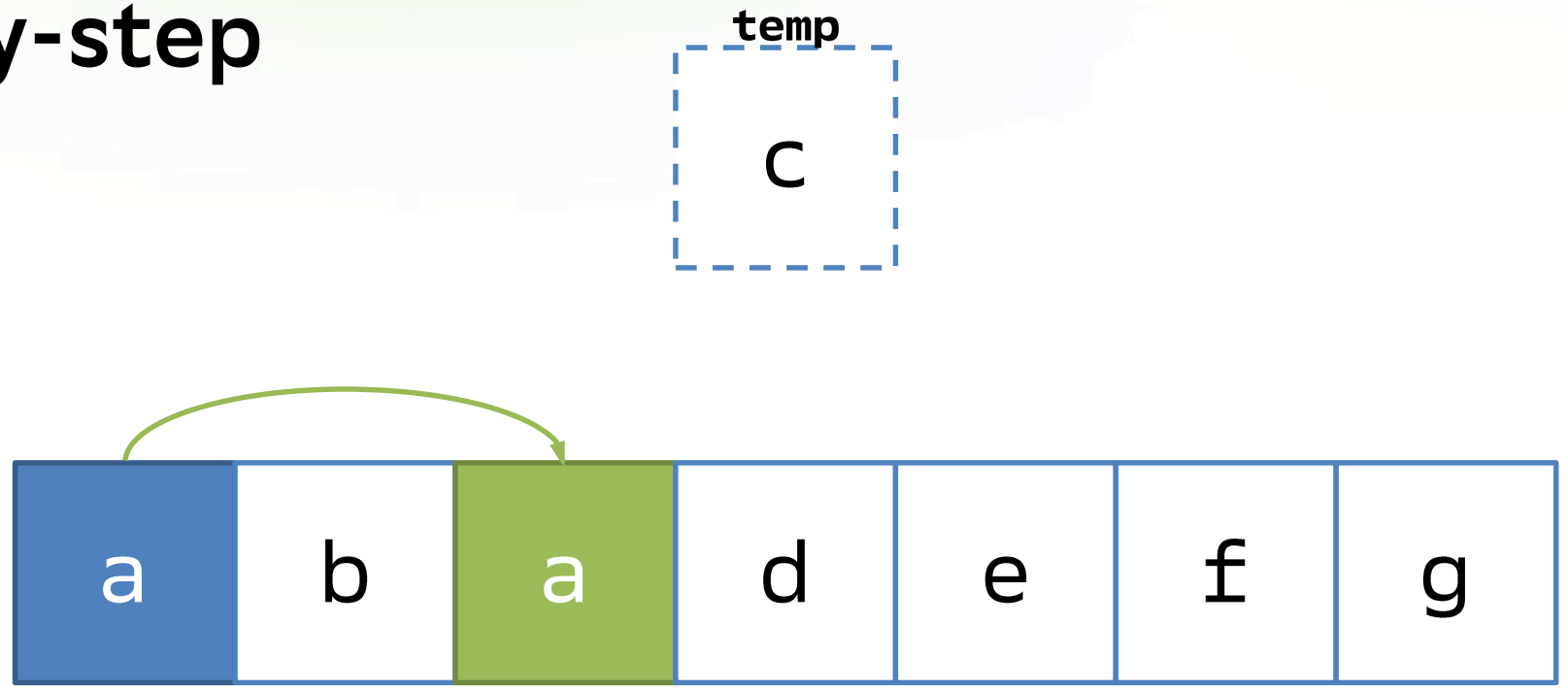


# Step-by-step

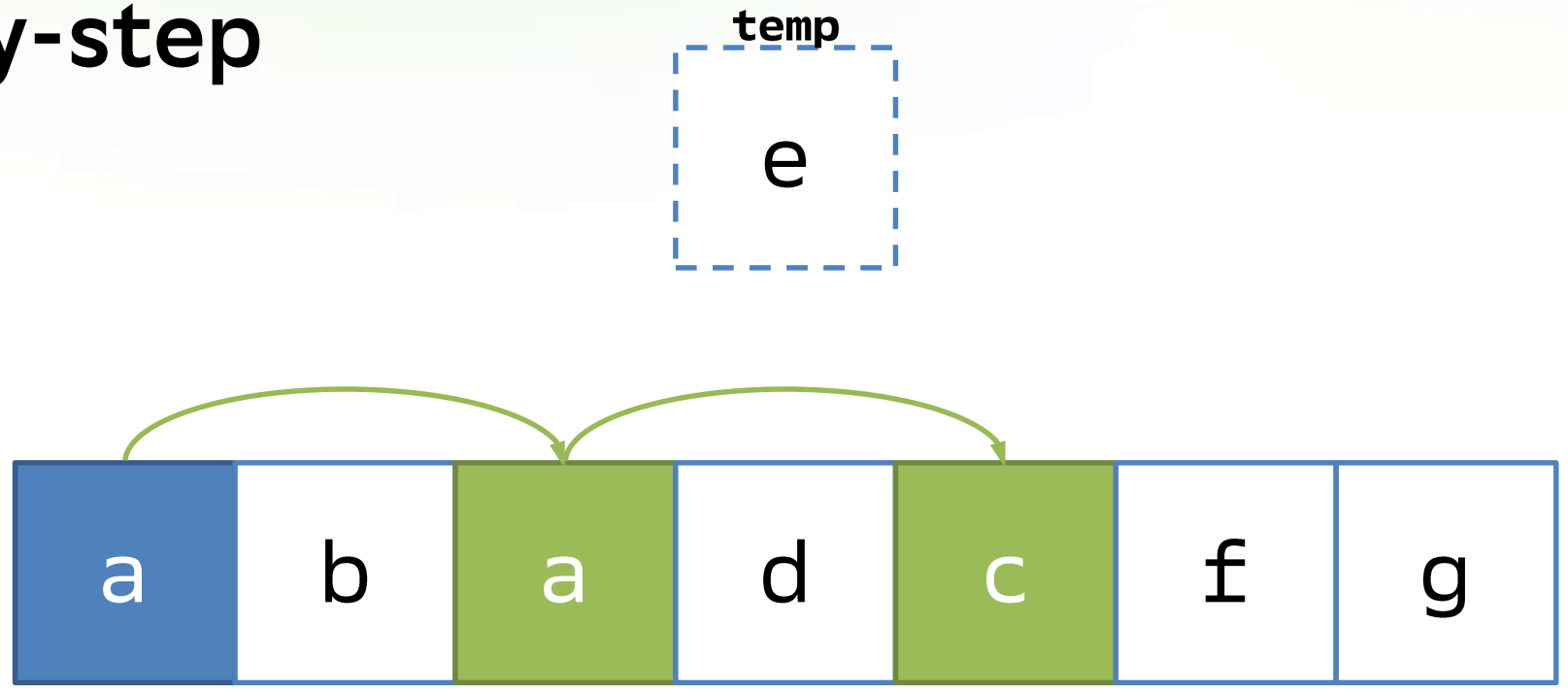


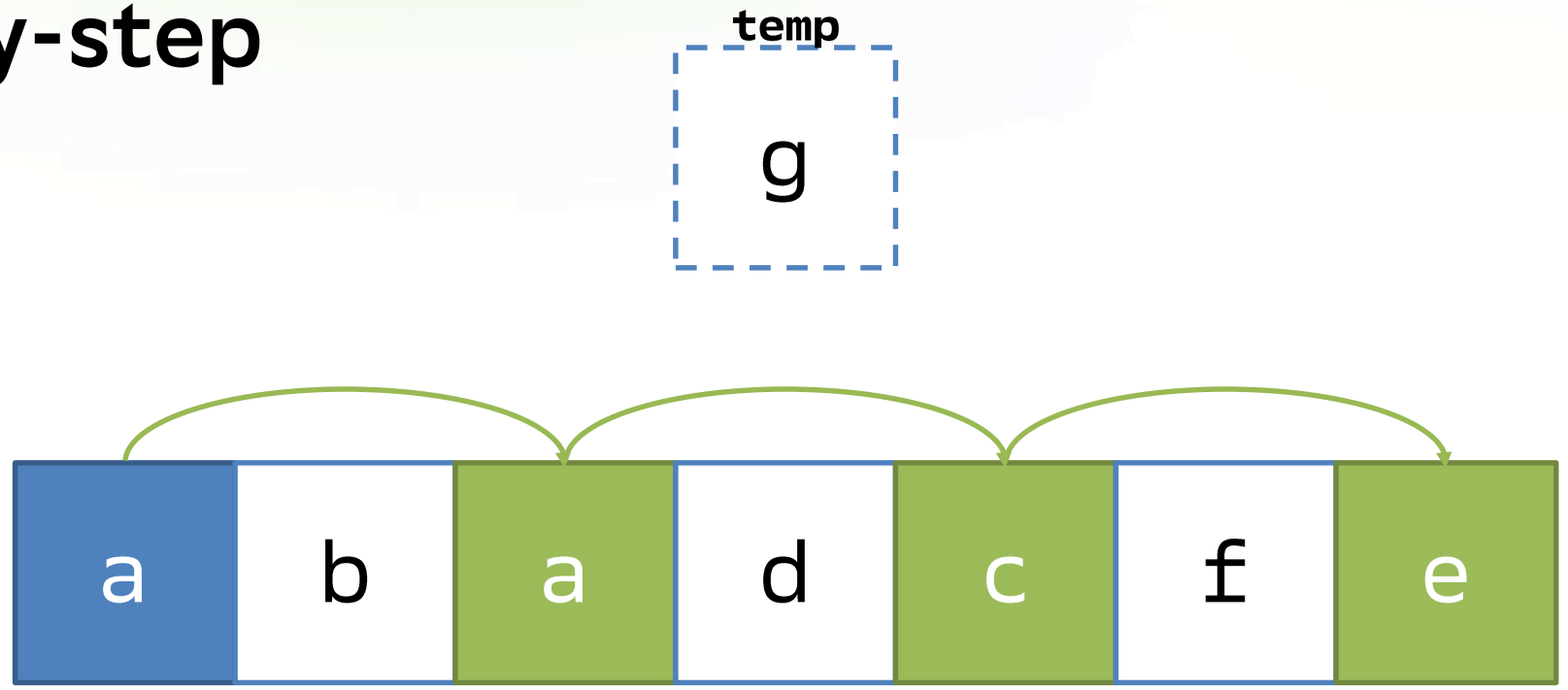
# Step-by-step



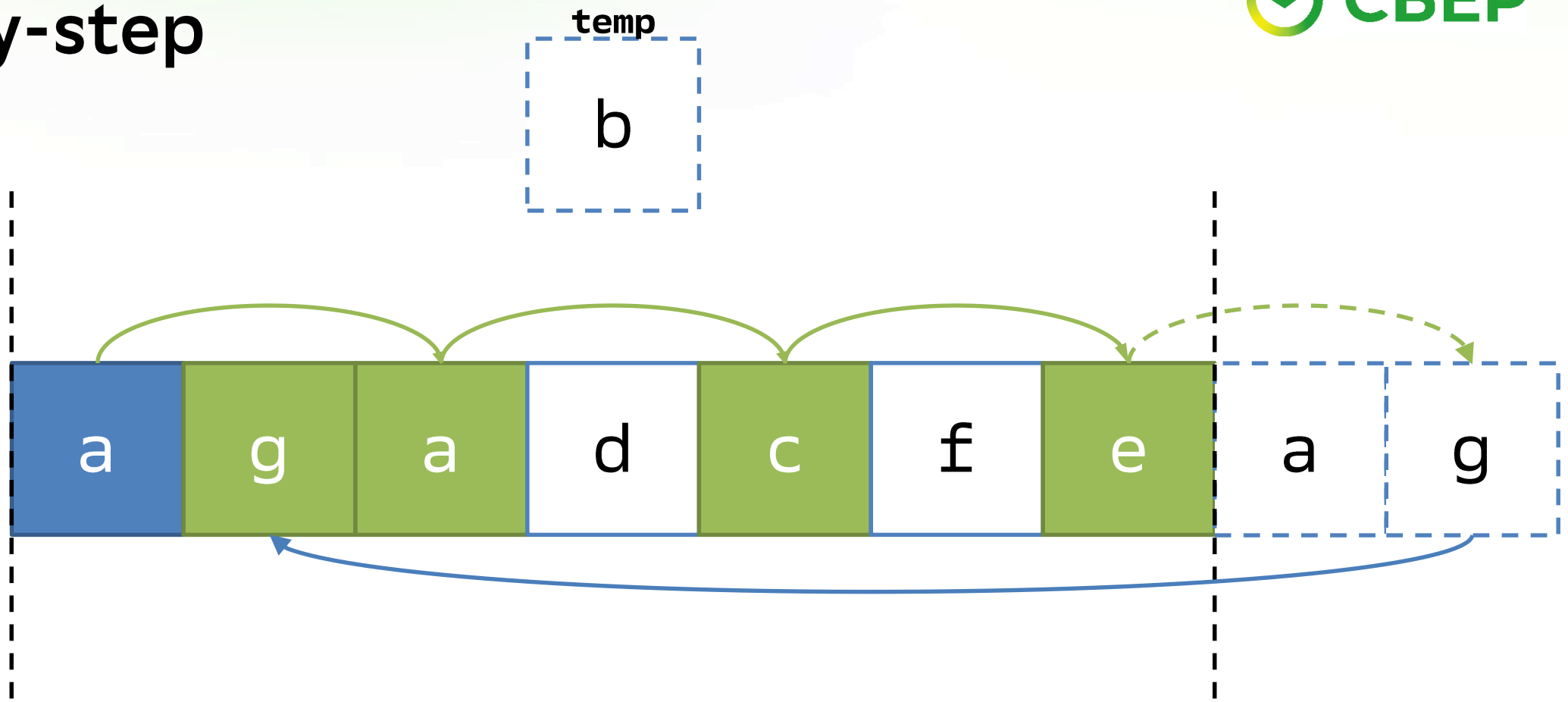
# Step-by-step



# Step-by-step



# Step-by-step



# Circular array

СБЕР

```
public static <T> void rotate(List<T> list, int distance);
```

# Circular array

СБЕР

```java
public static <T> void rotate(List<T> list, int distance) {
    int size = list.size();
    T temp = list.get(0);
    for (int i = 0, nMoved = 0; nMoved < size; ++nMoved) {
        i = (i + distance) % size;
        temp = list.set(i, temp);
    }
}
```

# Circular array

СБЕР

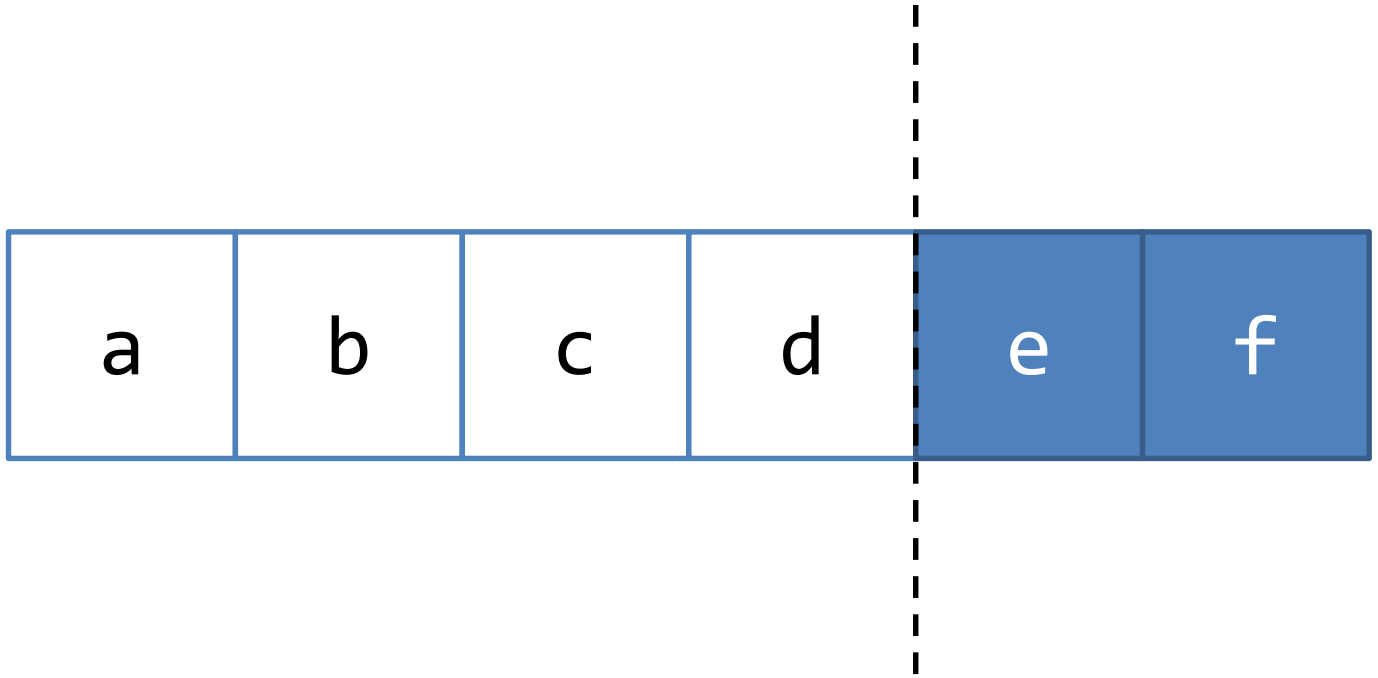
```java
public static <T> void rotate(List<T> list, int distance) {
    int size = list.size();
    if (size == 0)
        return;
    T temp = list.get(0);
    for (int i = 0, nMoved = 0; nMoved < size; ++nMoved) {
        i = (i + distance) % size;
        temp = list.set(i, temp);
    }
}
```

# Пишем за полгода 5 строчек кода

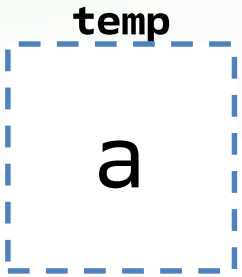
СБЕР

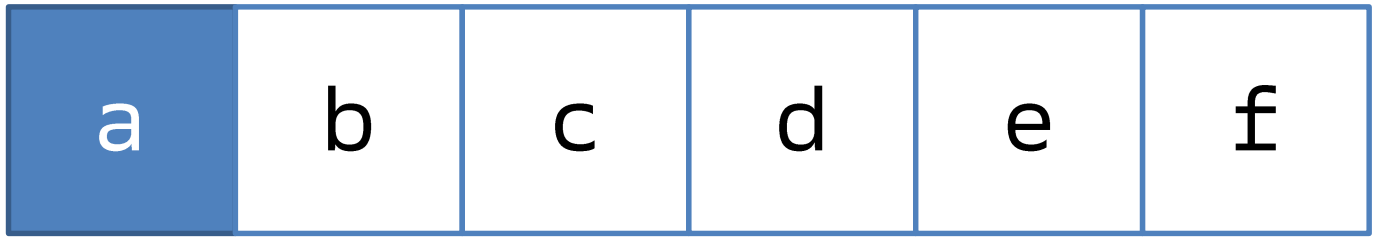
```java
public static <T> void rotate(List<T> list, int distance) {
    int size = list.size();
    if (size == 0)
        return;
    distance = distance % size;
    if (distance < 0)
        distance += size;
    if (distance == 0)
        return;
    T temp = list.get(0);
    for (int i = 0, nMoved = 0; nMoved < size; ++nMoved) {
        i = (i + distance) % size;
        temp = list.set(i, temp);
    }
}
```

# А если так?

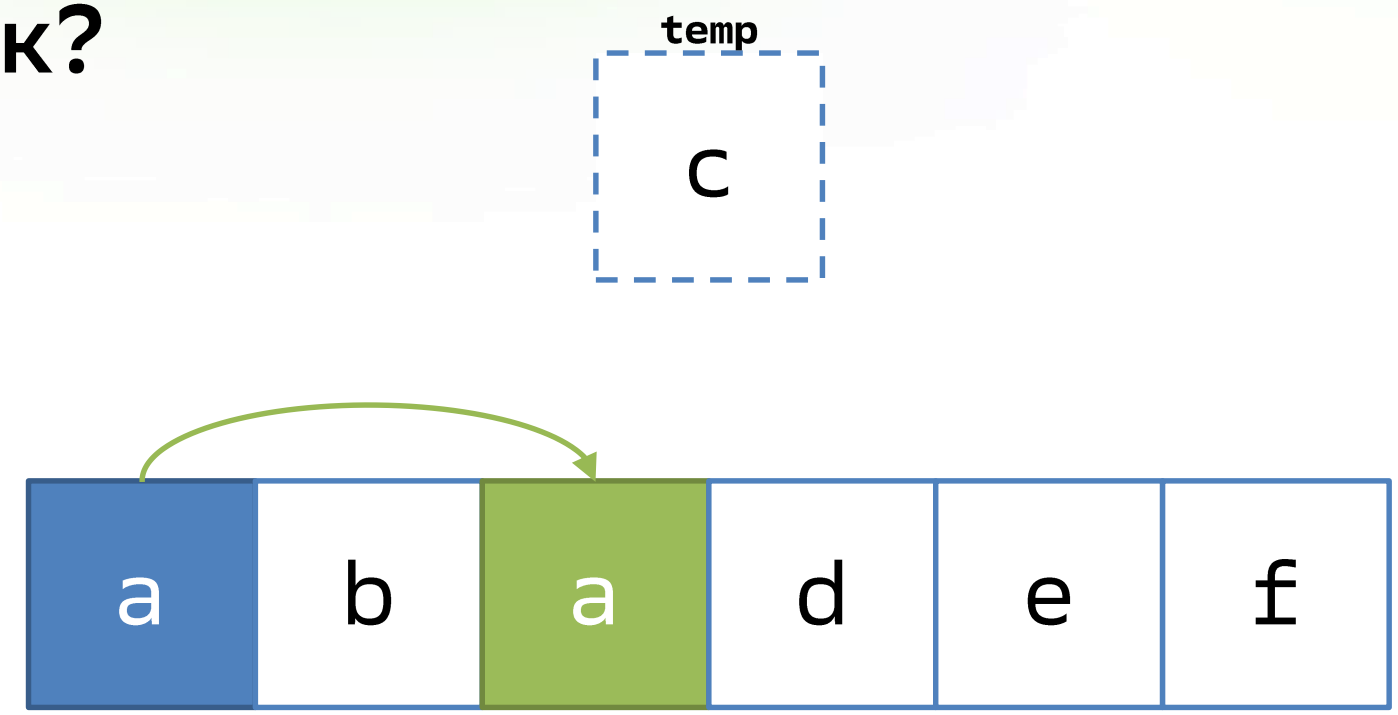


# А если так?

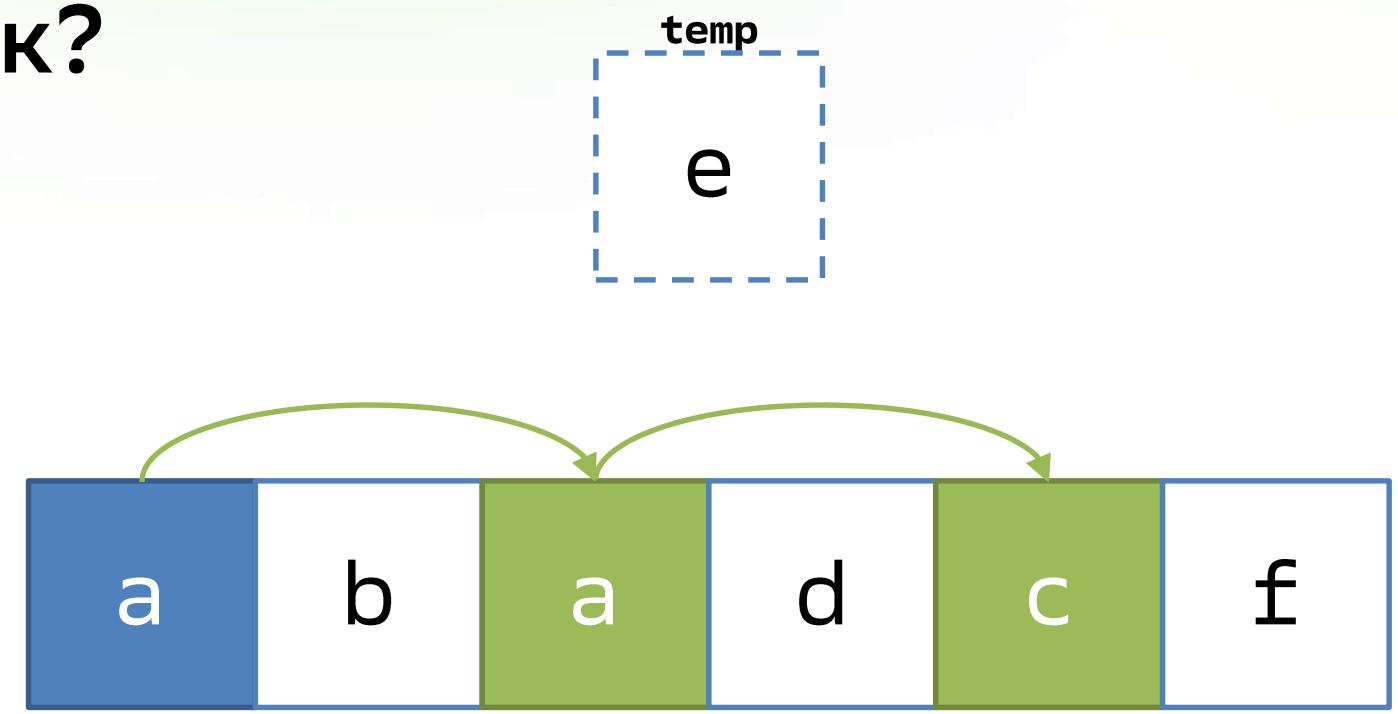


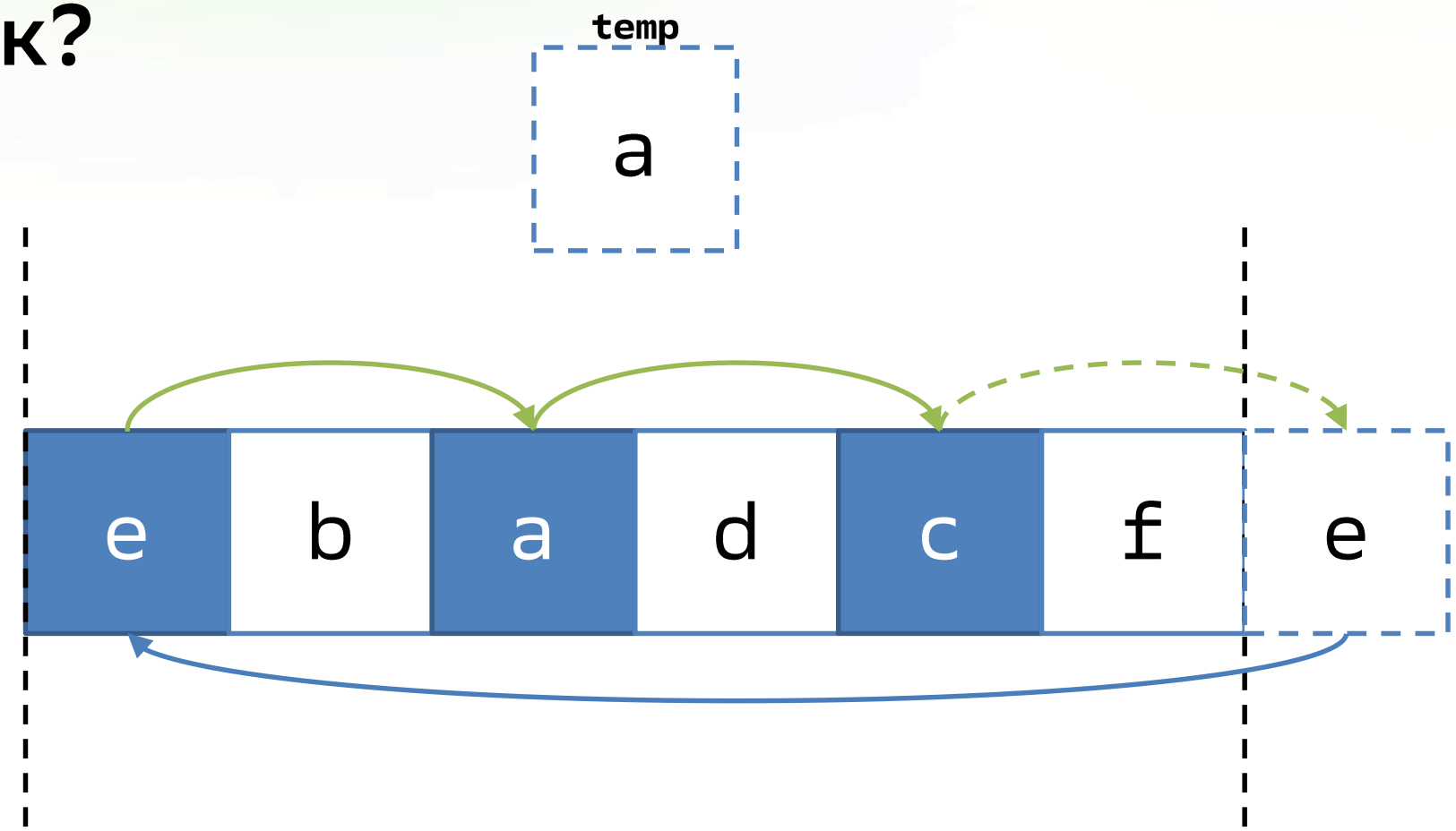
# А если так?



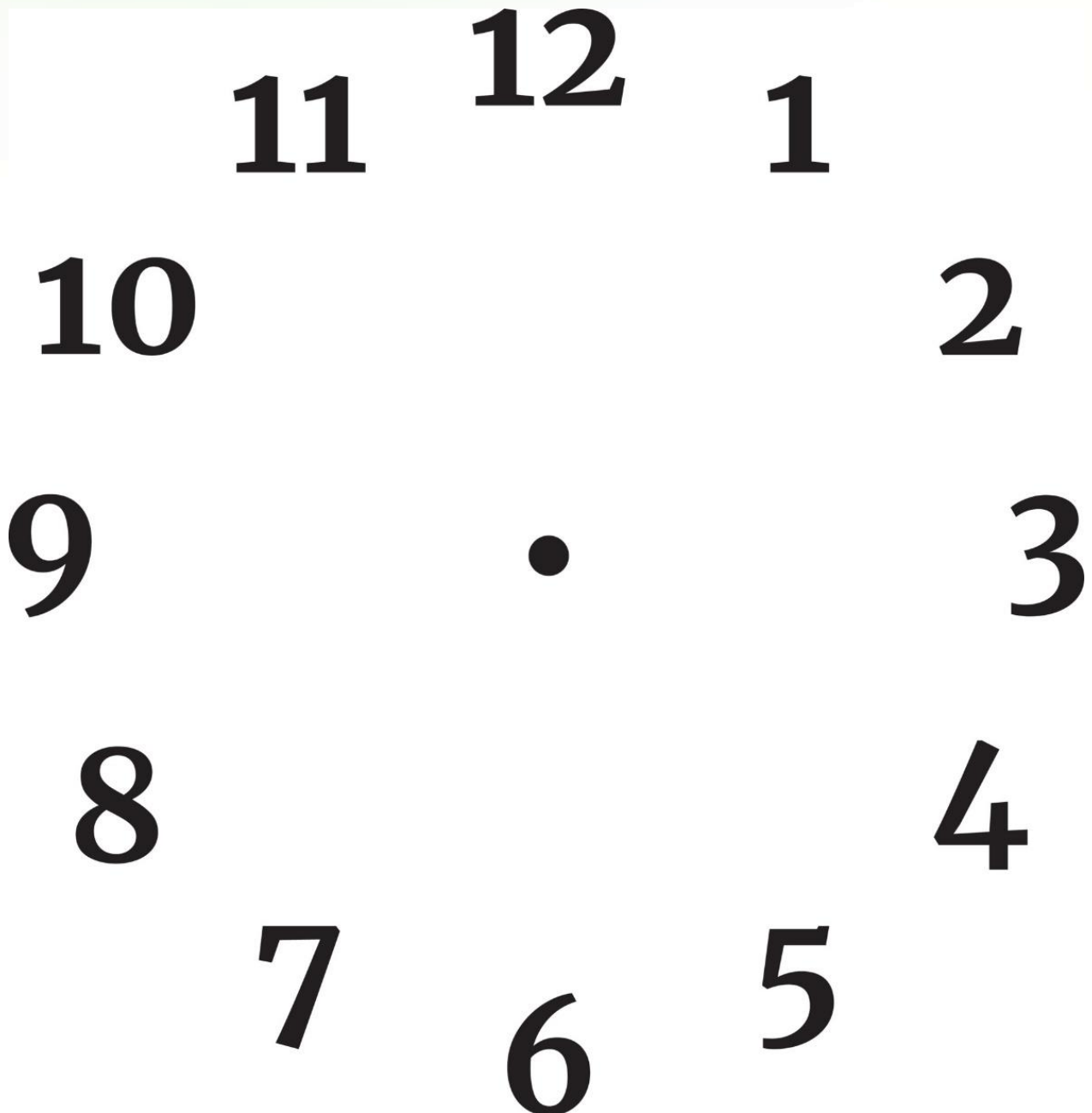
# А если так?



# А если так?



12-3-6-9



# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:
  - `0-9-6-3`

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:
  - `0-9-6-3`
  - `1-10-7-4`

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:
  - `0-9-6-3`
  - `1-10-7-4`
  - `2-11-8-5`

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:
  - `0-9-6-3`
  - `1-10-7-4`
  - `2-11-8-5`
- Циклы не «пересекаются» и их количество равно **НОД**: **`gcd(size, distance)`**

# Не все так «просто»

- Если **`size`** и **`distance`** взаимно просты, то одного **`for`** достаточно
- Но при **`size=12`** и **`distance=9`**:
  - `0-9-6-3`
  - `1-10-7-4`
  - `2-11-8-5`
- Циклы не «пересекаются» и их количество равно **НОД**: **`gcd(size, distance)`**
- Нужен внешний **`for`** по различным «стартам» циклов

# Теперь то всё?

СБЕР

```
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do {
        i = (i + distance) % size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Делить или не делить, вот в чем вопрос

- Старые **ARM** процессоры не имели инструкции целочисленного деления. Она появилась только в **ARMv7**

# Делить или не делить, вот в чем вопрос

- Старые **ARM** процессоры не имели инструкции целочисленного деления. Она появилась только в **ARMv7**
- **iPhone 3G** «не умел» делить целые числа аппаратно

# Делить или не делить, вот в чем вопрос

- Старые **ARM** процессоры не имели инструкции целочисленного деления. Она появилась только в **ARMv7**
- **iPhone 3G** «не умел» делить целые числа аппаратно
- **RISC-V** предусматривает расширение (**M-extension**) содержащее данную инструкцию

# Делить или не делить, вот в чем вопрос

- Старые **ARM** процессоры не имели инструкции целочисленного деления. Она появилась только в **ARMv7**
- **iPhone 3G** «не умел» делить целые числа аппаратно
- **RISC-V** предусматривает расширение (**M-extension**) содержащее данную инструкцию
- Но не все процессоры **RISC-V** данное расширение реализуют

# Делить или не делить, вот в чем вопрос

СБЕР

```
// i = (i + distance) % size;
i += distance;
if (i >= size)
    i -= size;
```

# Преподавание – это своего рода QA

- От студентов требовалось:

# Преподавание – это своего рода QA

- От студентов требовалось:
  - Модульная структура кода

# Преподавание – это своего рода QA

- От студентов требовалось:
  - Модульная структура кода
  - Unit-тесты под каждый метод

# Преподавание – это своего рода QA

- От студентов требовалось:
  - Модульная структура кода
  - Unit-тесты под каждый метод
- У преподавателя есть свой набор Unit-тестов неизвестных студентам заранее

# Преподавание – это своего рода QA

- От студентов требовалось:
  - Модульная структура кода
  - Unit-тесты под каждый метод
- У преподавателя есть свой набор Unit-тестов неизвестных студентам заранее
- Эти тесты содержат много граничных случаев

# Преподавание – это своего рода QA

- От студентов требовалось:
  - Модульная структура кода
  - Unit-тесты под каждый метод
- У преподавателя есть свой набор Unit-тестов неизвестных студентам заранее
- Эти тесты содержат много граничных случаев
- Проверяет Checker

# Checker в осаде

СБЕР

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index -2147483648 out of bounds
for length 1073741825
        at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
        at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
        at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:266)
        at java.base/java.util.Objects.checkIndex(Objects.java:359)
        at java.base/java.util.ArrayList.set(ArrayList.java:441)
        ...
```

# Откуда списывал?

- Реализации написанные студентами разделились на две категории:

# Откуда списывал?

- Реализации написанные студентами разделились на две категории:
  - «Всё очень плохо»

# Откуда списывал?

- Реализации написанные студентами разделились на две категории:
  - «Всё очень плохо»
  - «Только не списывай точь в точь»

Начальная школа: списывать неприемлемо

Старшая школа: списывать неприемлемо

Университет: списывать неприемлемо

Работа:

Чел, я украл твой код

Это не мой код

# А если из Java?

СБЕР

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index -2147483648 out of bounds
for length 1073741825
        at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
        at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
        at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:248)
        at java.base/java.util.Objects.checkIndex(Objects.java:372)
        at java.base/java.util.ArrayList.set(ArrayList.java:473)
        at java.base/java.util.Collections.rotate1(Collections.java:802)
        at java.base/java.util.Collections.rotate(Collections.java:780)
        ...
```

# Идём в OpenJDK

СБЕР

```
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# И придумываем тест

СБЕР

```
int size = (1 << 30) + 1;
List<Object> list = new ArrayList<>(Collections.nCopies(size, null));
rotate(list, size - 1);
```


просчитался,
но...
где?

# Просчитался, но... где?

СБЕР

```
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start); // list.get(0)
    int i = start;
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start; // i = 0
    do {
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 0
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 0
        i += distance; // i = 0 + (1 << 30)
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance; // i = 1 << 30
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size) // (1 << 30) >= (1 << 30) + 1
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size) // false
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp); // list.set(1 << 30, ...)
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved; // nMoved = 1
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start); // (1 << 30) != 0
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start); // true
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = 1 << 30
        i += distance; // i = (1 << 30) + (1 << 30)
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = -2147483648
        i += distance; // i = -2147483648
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = Integer.MIN_VALUE
        i += distance; // i = Integer.MIN_VALUE
        if (i >= size)
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = Integer.MIN_VALUE
        i += distance;
        if (i >= size) // Integer.MIN_VALUE >= (1 << 30) + 1
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = Integer.MIN_VALUE
        i += distance;
        if (i >= size) // false
            i -= size;
        temp = list.set(i, temp);
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = Integer.MIN_VALUE
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp); // list.set(Integer.MIN_VALUE, ...)
        ++nMoved;
    } while (i != start);
}
```

# Просчитался, но... где?

СБЕР

```
// size = (1 << 30) + 1, distance = (1 << 30)
for (int start = 0, nMoved = 0; nMoved < size; ++start) {
    T temp = list.get(start);
    int i = start;
    do { // i = Integer.MIN_VALUE
        i += distance;
        if (i >= size)
            i -= size;
        temp = list.set(i, temp); // ArrayIndexOutOfBoundsException
        ++nMoved;
    } while (i != start);
}
```

# И это действительно Bug

СБЕР

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index -2147483648 out of bounds
for length 1073741825
        at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
        at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
        at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:248)
        at java.base/java.util.Objects.checkIndex(Objects.java:372)
        at java.base/java.util.ArrayList.set(ArrayList.java:473)
        at java.base/java.util.Collections.rotate1(Collections.java:802)
        at java.base/java.util.Collections.rotate(Collections.java:780)
        ...
```

# План работ

Заполняем Compliance

# Заполняем Compliance

7. Please place an "**x**" on one of the applicable statement below. Please do NOT mark both statements:

_____ I am signing on behalf of myself as an individual and no other person or entity, including my employer, has or will have rights with respect my contributions.

_____ I am signing on behalf of my employer or a legal entity and I have the actual authority to contractually bind that entity.

| | |
|---|---|
| Name*: | |
| Company's Name (if applicable): | |
| Title or Role (if applicable): | |
| Mailing Address*: | |
| Telephone, Fax and Email*: | |
| Signature*: | |
| Date*: | |
| Project Name*: | |
| Username (if applicable): | |

* Required field

Oracle Contributor Agreement – version 1.7.1

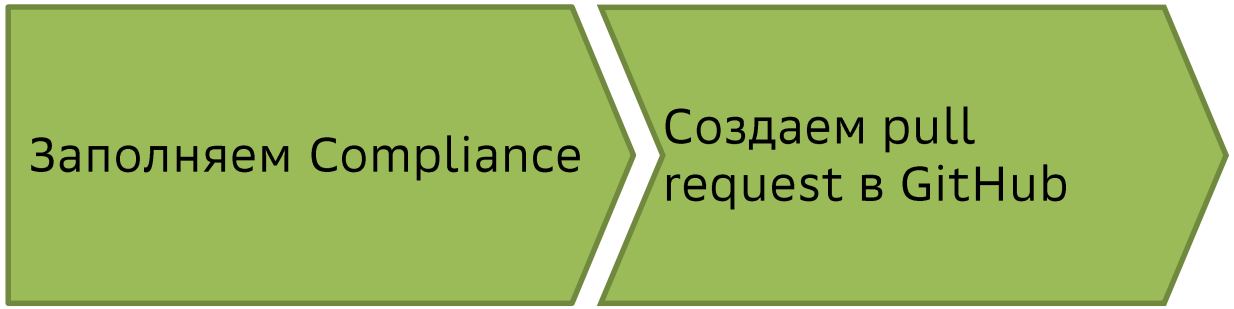
# План работ

Нормативный срок рассмотрения составляет 3 месяца

Заполняем Compliance

# План работ

Нормативный срок
рассмотрения
составляет 3 месяца



# Создаем pull request в GitHub

**nikita-sakharin** commented on Aug 14 • edited by openjdk bot Contributor

`Collections.rotate` method contains a bug. This method throws IndexOutOfBoundsException on arrays larger than $2^{30}$ elements. The way to reproduce:

```
final int size = (1 << 30) + 1;
final List<Byte> list = new ArrayList<>(size);
for (int i = 0; i < size; ++i)
    list.add((byte) 0);
Collections.rotate(list, size - 1);
```

Output:

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index -2147483648 out of bounds for length 1073741825
```

In that case private method `Collections.rotate1` will be called. And the line:

```
i += distance;
```

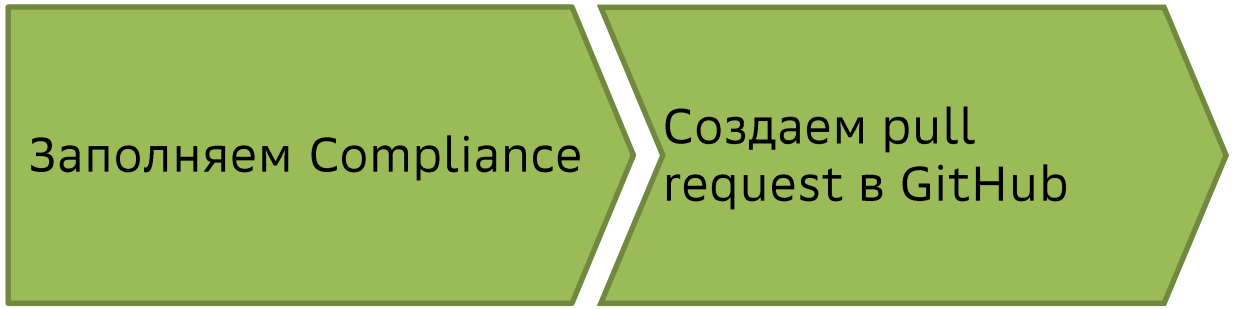
will cause overflow. I fixed this method and wrote a test for it.

I've signed the Oracle Contributor Agreement, but I don't have permission to raise a bug in the JDK Bug System.

Kindly ask you to raise a bug.

# План работ

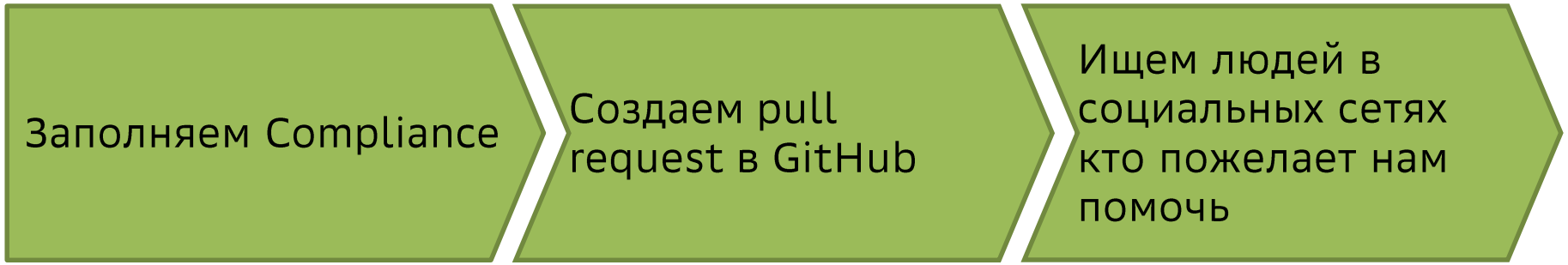
Нормативный срок рассмотрения составляет 3 месяца




Ждем пока на него отреагируют

# План работ

Нормативный срок рассмотрения составляет 3 месяца

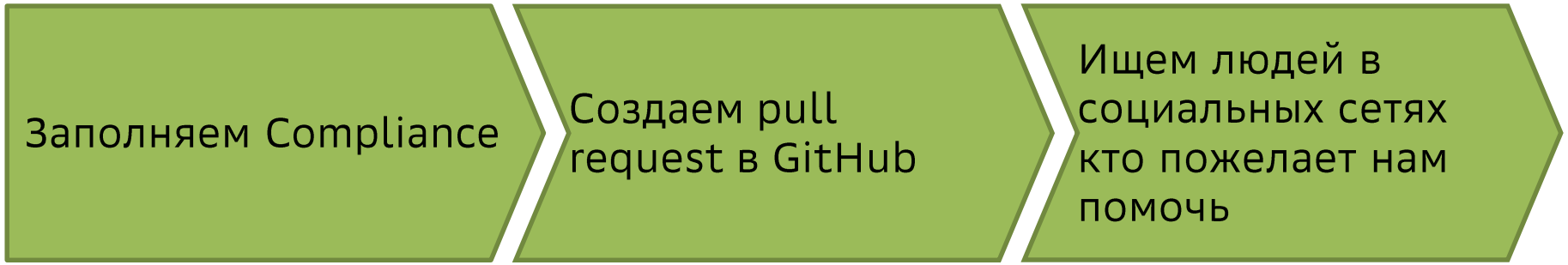



Ждем пока на него отреагируют

# План работ

Нормативный срок рассмотрения составляет 3 месяца

Можно попытаться начать со спикеров **JUG Ru Group**




Ждем пока на него отреагируют

# Алексей Шипилёв

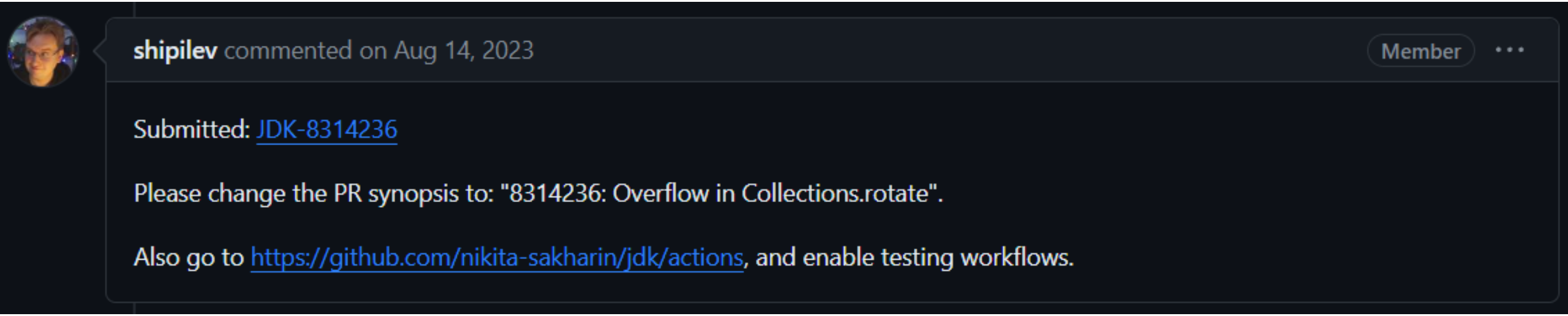
- **Перформанс: Что В Имени Тебе Моём?**
- Прагматика Java Memory Model
- Java Benchmarking: как два таймстампа прочитать!
- Катехизис java.lang.String
- Shenandoah: сборщик мусора, который смог
- Java-объекты наизнанку
- Performance Optimization 101

# Счастливый билет


JDK / JDK-8314236

## Overflow in Collections.rotate

# Процесс пошел



**shipilev** commented on Aug 14, 2023 Member

Submitted: JDK-8314236

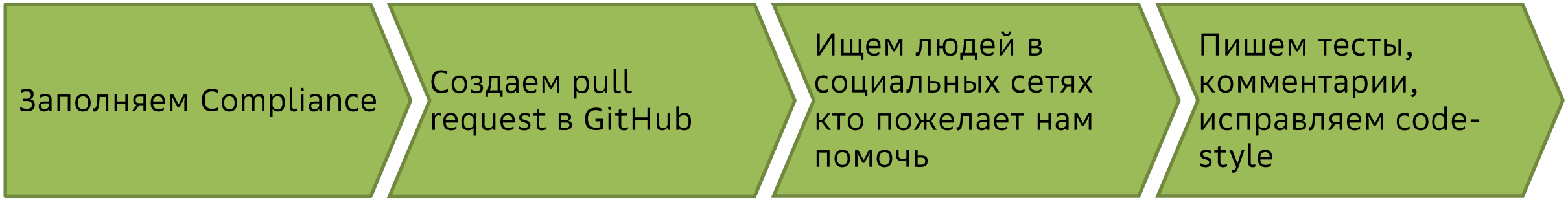
Please change the PR synopsis to: "8314236: Overflow in Collections.rotate".

Also go to https://github.com/nikita-sakharin/jdk/actions, and enable testing workflows.

# План работ

Нормативный срок рассмотрения составляет 3 месяца

Можно попытаться начать со спикеров **JUG Ru Group**




Ждем пока на него отреагируют

# Первая версия тестов

СБЕР

```
public class RotateHuge {
    public static void main(String[] args) {
        int size = (1 << 30) + 1;
        List<Object> list = new ArrayList<>(Collections.nCopies(size, null));
        Collections.rotate(list, distance);
    }
}
```

# Нужно больше ресурсов!

СБЕР

**shipilev** on Aug 15, 2023

Member

Since this test takes >4G of heap to hold the list with compressed oops, and >8G of heap without compressed oops, we need to run it in a separate VM with enough headroom, something like this:

```
* @test
* @bug 8314236
* @summary Overflow in Collections.rotate
* @requires (sun.arch.data.model == "64" & os.maxMemory >= 16g)
* @run main/othervm -Xmx12g RotateHuge
```

# А если чуть-чуть подумать?

СБЕР

**stuart-marks** on Aug 17, 2023

Member

**@nikita-sakharin**

Thanks for finding this bug and offering to fix it! (And **@shipilev** thanks for your assistance on this.)

Putting the test into a separate JVM will work, but I don't think it's necessary to actually allocate the space. The test is only testing the indexes sent to `get` and `set` on the list, and it doesn't actually verify the contents of the list. (Presumably that's done by other tests.) Therefore it should be possible to create a "virtual" list of a given size that checks that the indexes are all in bounds but that doesn't actually store any elements. It should be fairly straightforward to do this by subclassing AbstractList and overriding a few methods.

The advantage of not actually allocating 4G of memory is that it makes it easier to run a bunch of cases that test the boundary conditions. In fact I'd like to see that in the test, as opposed to testing this one case.

👍 1

# Жизнь без Mockito

СБЕР

```
private static final class MockList extends AbstractList<Object>
    implements RandomAccess {
    // ...
}
```

# Жизнь без Mockito

СБЕР

```
private final int size;

public MockList(final int size) {
    if (size < 0)
        throw new IllegalArgumentException("Illegal size: " + size);
    this.size = size;
}
```

# Пишем за полгода 5 строчек кода

СБЕР

```java
@Override
public Object get(int index) {
    Objects.checkIndex(index, size);
    return null;
}

@Override
public Object set(int index, Object element)
{   Objects.checkIndex(index, size);
    return null;
}

@Override
public int size() {
    return size;
}
```

# Наращиваем покрытие

СБЕР

**stuart-marks** commented on Aug 24 Member

@nikita-sakharin

Thanks for the updates. With the "Mock List" implementation we can run the test in-JVM and we can avoid allocating several GB of memory. Great!

The implementation logic in the `rotate1` method looks correct.

Now that an individual test case is much less expensive, it becomes feasible to add multiple test cases. In particular, for this kind of testing of arithmetic errors, I like to test a variety of edge cases. The one test case you have is for size= `(1 << 30) - 1` and distance= `(1 << 30)`. It would be good to have a few other cases where the existing code fails and where the modified code should pass. I was able to come up with a few examples quickly:

```
size                 distance
Integer.MAX_VALUE    2
Integer.MAX_VALUE    Integer.MIN_VALUE
Integer.MAX_VALUE    Integer.MAX_VALUE - 1
```

Please add these cases, and any others that you think might be interesting.

# Неуловимый Джо

СБЕР

```
/*
 * This test covers only index computations.
 * Correctness of elements rotation is not checked.
 */
private static void testRotate(int size, int distance)
{    List<Object> list = new MockList(size);
     Collections.rotate(list, distance);
}
```

# Неуловимый Джо

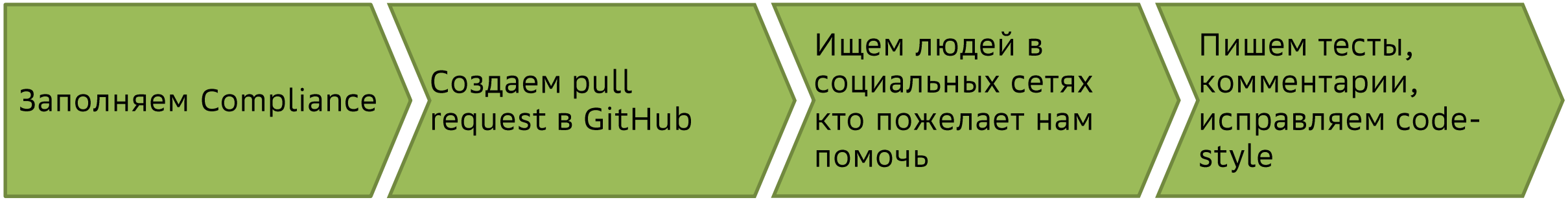
СБЕР

```
public static void main(String[] args) {
    testRotate((1 << 30) + 1, -(1 << 30) - 2);
    testRotate((1 << 30) + 1, 1 << 30);
    testRotate(Integer.MAX_VALUE, Integer.MIN_VALUE);
    testRotate(Integer.MAX_VALUE, Integer.MIN_VALUE + 3);
    testRotate(Integer.MAX_VALUE, 2);
    testRotate(Integer.MAX_VALUE, Integer.MAX_VALUE - 1);
}
```

# План работ

Нормативный срок рассмотрения составляет 3 месяца

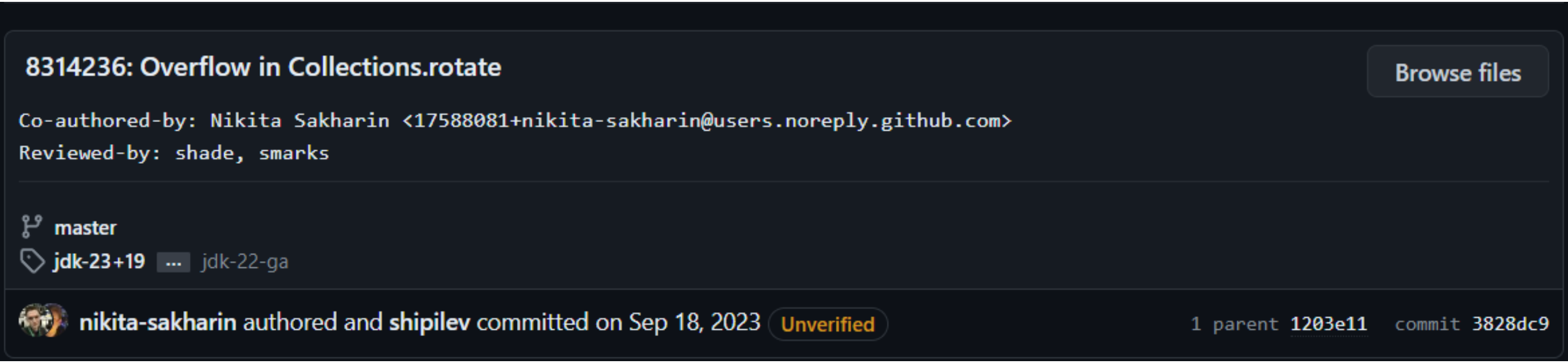
Можно попытаться начать со спикеров **JUG Ru Group**




Ждем пока на него отреагируют

**`/integrate`** для **merge** в **master**

# Достучаться до mater-a



# Пишем за полгода 5 строчек кода

СБЕР

9 src/java.base/share/classes/java/util/Collections.java

```
@@ -816,15 +816,16 @@ private static <T> void rotate1(List<T> list, int distance) {
816 816         if (distance == 0)
817 817             return;
818 818
819     -       for (int cycleStart = 0, nMoved = 0; nMoved != size; cycleStart++) {
    819 +       int bound = size - distance;
    820 +       for (int cycleStart = 0, nMoved = 0; nMoved < size; cycleStart++) {
820 821             T displaced = list.get(cycleStart);
821 822             int i = cycleStart;
822 823             do {
823     -               i += distance;
824     -               if (i >= size)
    824 +               if (i >= bound)
825 825                     i -= size;
    826 +               i += distance;
826 827                 displaced = list.set(i, displaced);
827     -               nMoved ++;
    828 +               nMoved++;
828 829             } while (i != cycleStart);
829 830         }
830 831     }
```

# А сколько же Вам лет?

- Метод **Collections.rotate** был добавлен в **J2SE 1.4**

# А сколько же Вам лет?

- Метод **Collections.rotate** был добавлен в **J2SE 1.4**
- **J2SE 1.4** была выпущена 6-го февраля 2002 года

# А сколько же Вам лет?

- Метод **Collections.rotate** был добавлен в **J2SE 1.4**
- **J2SE 1.4** была выпущена 6-го февраля 2002 года
- Bug был исправлен 18-го сентября 2023 года

# А сколько же Вам лет?

- Метод **Collections.rotate** был добавлен в **J2SE 1.4**
- **J2SE 1.4** была выпущена 6-го февраля 2002 года
- Bug был исправлен 18-го сентября 2023 года
- За **22 года** этот Bug мог бы уже закончить университет и стать Junior-ом

# Резюме

- Написали лабораторную по сортировке **Merge Sort**

# Резюме

- Написали лабораторную по сортировке **Merge Sort**
- Нашли ошибку переполнения в **Collections.rotate**

# Резюме

- Написали лабораторную по сортировке **Merge Sort**
- Нашли ошибку переполнения в **Collections.rotate**
- Отправили исправление в **OpenJDK**

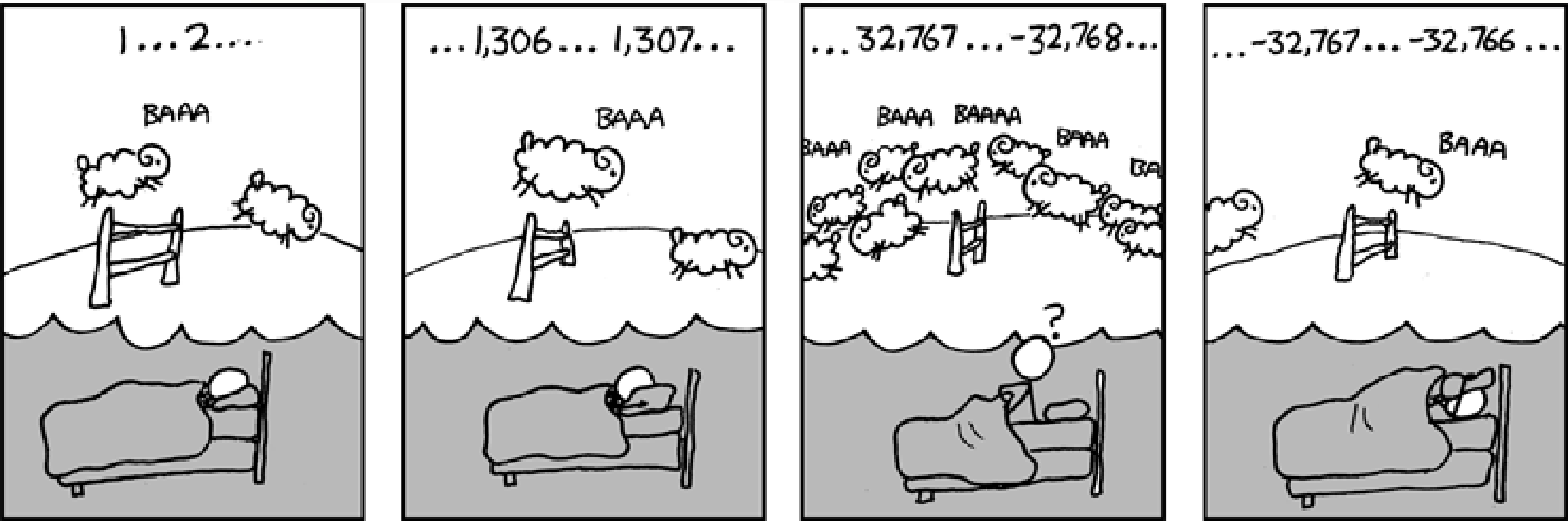
# Резюме

- Написали лабораторную по сортировке **Merge Sort**
- Нашли ошибку переполнения в **Collections.rotate**
- Отправили исправление в **OpenJDK**
- Добились **merge** в **master**

# Резюме

- Написали лабораторную по сортировке **Merge Sort**
- Нашли ошибку переполнения в **Collections.rotate**
- Отправили исправление в **OpenJDK**
- Добились **merge** в **master**
- Бесценный опыт: полгода и 5 строчек кода

# Самая частая ошибка в мире IT



# Эту ошибку в IT совершают все...

- В 2006 году в `Collections` было исправлено переполнение в бинарном поиске `binarySearch`:
  **`mid = (low + high) / 2`**

# Эту ошибку в IT совершают все...

- В 2006 году в `Collections` было исправлено переполнение в бинарном поиске `binarySearch`: **`mid = (low + high) / 2`**
- В 2015 году был обнаружен и исправлен выход за границы внутреннего массива в `Collections.sort`, которая реализуется через **Timsort**

# Выводы

- Стандартные библиотеки языков, компиляторы, ОС и даже «железо» могут содержать дефекты

# Выводы

- Стандартные библиотеки языков, компиляторы, ОС и даже «железо» могут содержать дефекты
- **Java** не исключение – это не Святой Грааль

# Выводы

- Стандартные библиотеки языков, компиляторы, ОС и даже «железо» могут содержать дефекты
- **Java** не исключение – это не Святой Грааль
- Open Source «не кусается», но нужно набраться терпения

# Выводы

- Стандартные библиотеки языков, компиляторы, ОС и даже «железо» могут содержать дефекты
- **Java** не исключение – это не Святой Грааль
- Open Source «не кусается», но нужно набраться терпения
- Обучая других, можно и самому чему-то научиться

# Ссылки

**Данный дефект**:

- https://bugs.openjdk.org/browse/JDK-8314236
- https://github.com/openjdk/jdk/pull/15270
- https://github.com/openjdk/jdk/commit/3828dc913a3ea28d622b69bd07f26949128eb5f7

**Habr**:

- Баг в **binarySearch**: https://habr.com/ru/articles/203398
- Баг в **Timsort**: https://habr.com/ru/articles/251751

# Список литературы


- B-C. Huang, M. A. Langston (1992). Fast Stable Merging and Sorting In Constant Extra Space
- **Antonios Symvonis (1994). Optimal Stable Merging**
- Viliam Geert, Jyrki Katajainenb, Tomi Pasanen (1995). Asymptotically Efficient in-Place Merging
- Katajainen J., Träff J.L. (1997). A meticulous analysis of mergesort programs
- Bing-Chao Huang, Michael A. Langston (1998). Practical In-Place Merging
- Jyrki Katajainen, Tomi A. Pasanen (1999). In-place sorting with fewer moves
- Pok-Son Kim, Arne Kutzner (2006). On Optimal and Efficient in Place Merging

**Сахарин Никита**

Старший разработчик в **SberData**

nikitasa1997@gmail.com

t.me/nikita_sakharin