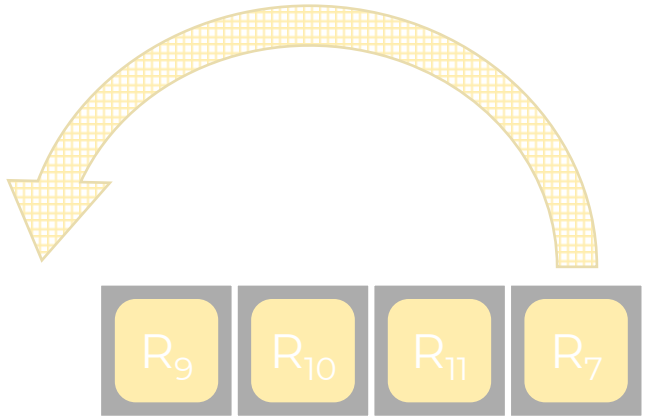
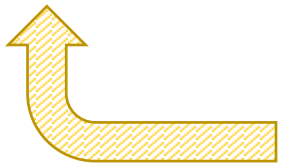


виикоорСт

Пришло время учиться оперировать данными!



Сортировки

Что у нас сегодня?

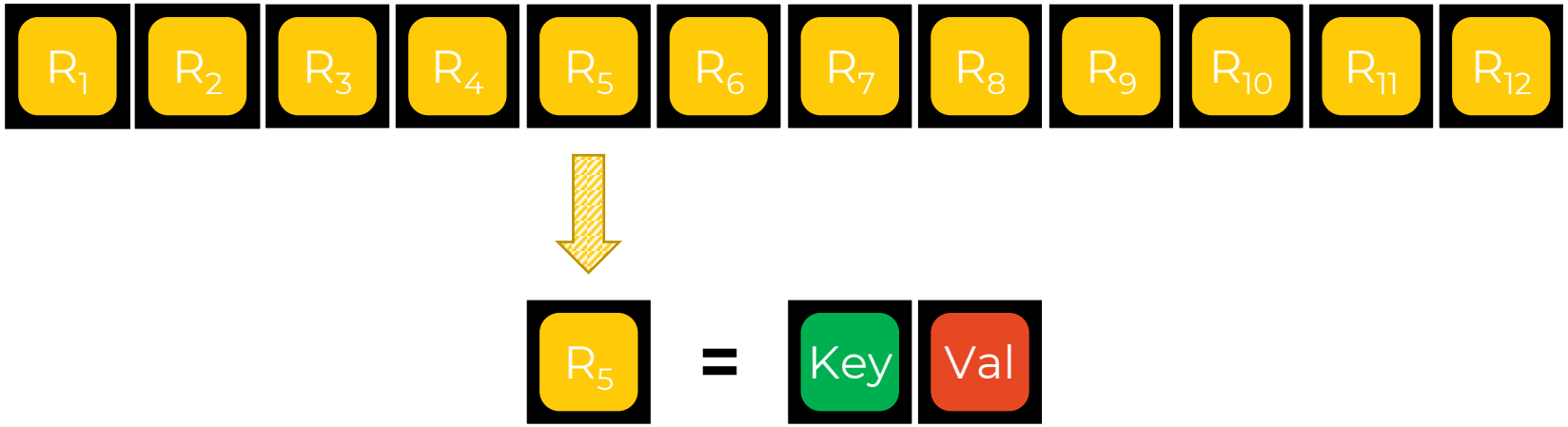
1. Сортированный массив
2. Сортировка слиянием
3. Быстрая сортировка
4. k-я порядковая статистика
5. Асимптотика сортировки
6. Сортировка подсчетом

Что такое сортированный массив?

Это массив, который уже отсортирован!

Что значит «отсортирован»?

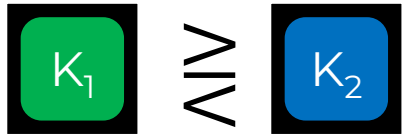
Дан массив элементов:



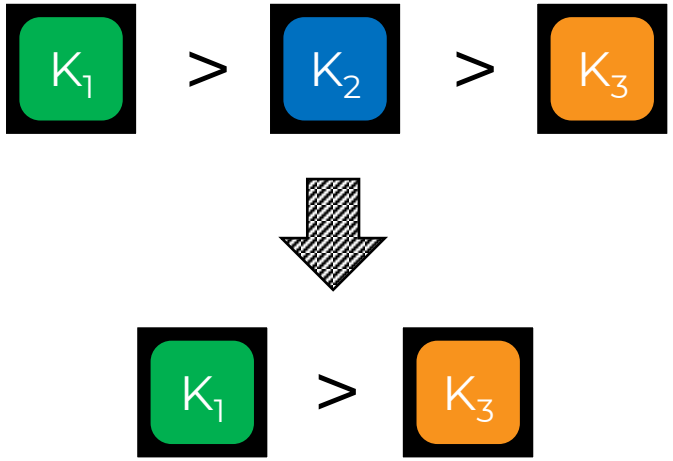
Условия для ключей

Для любых ключей

- Доступно сравнение:

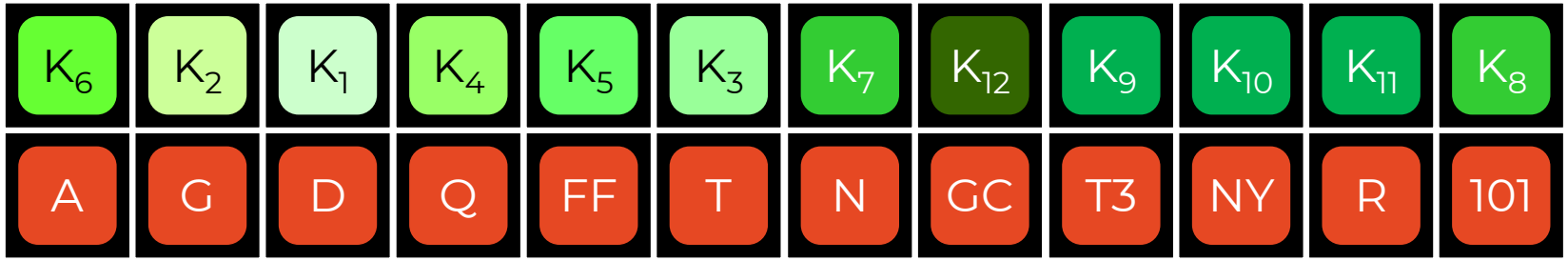


- Сравнение транзитивно:



Что значит «отсортирован»?

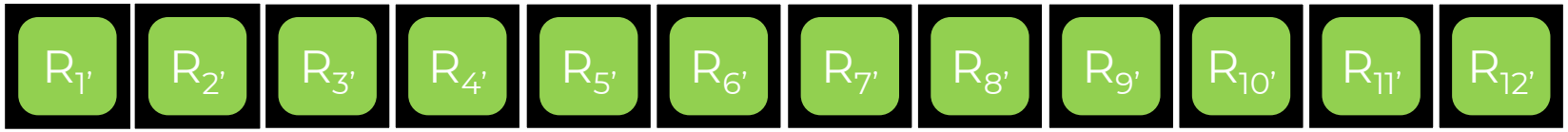
Дан массив элементов:



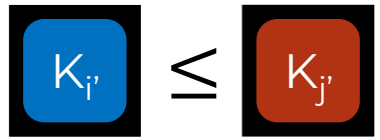
Задача сортировки

Дан массив элементов:

Нужно переставить элементы в новом порядке:



так, чтобы для любых 

, где $i' < j'$: 

Задача сортировки

Таким образом, задача сортировки элементов:

Эквивалентна задаче сортировки их ключей:

Так как в большинстве случаев ключи – **числа** или сводимые к ним сущности, то без ограничения общности можем рассматривать **сортировку чисел**.

Сортированный массив

В общем случае это тот массив, который удовлетворяет задаче сортировки.

На каком из массивов – сортированном или несортированном – проще производить операции?

- Операцию поиска?
- Операцию индексации?
- Операцию вставки/удаления?

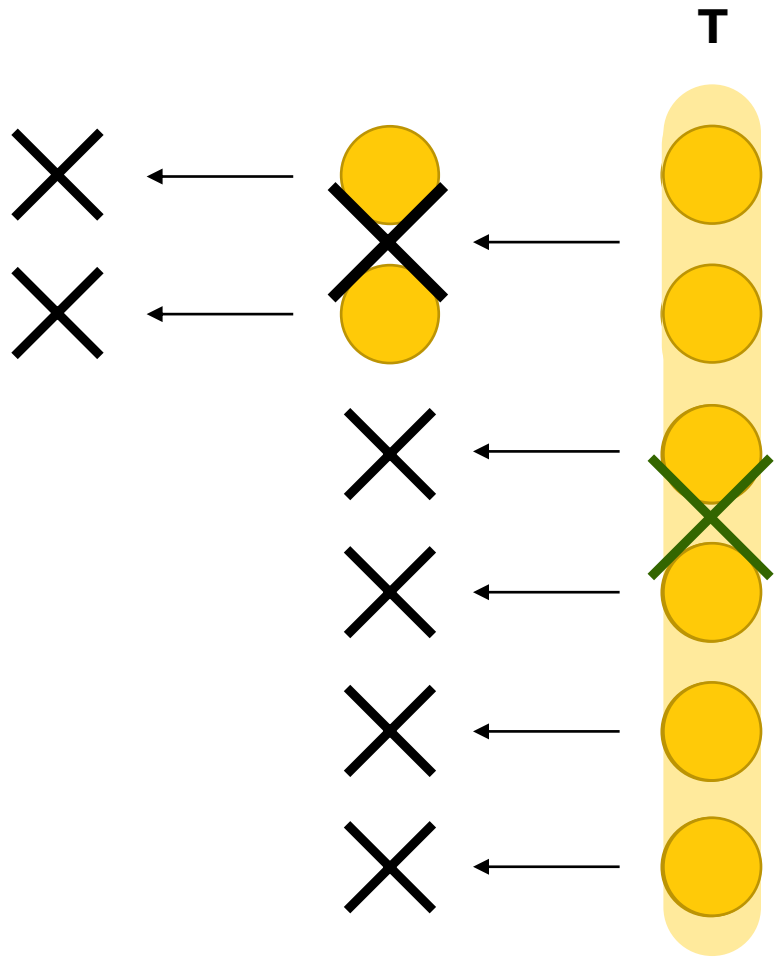
Алгоритмы сортировки

Какие алгоритмы сортировки вы бы могли предложить?

Стратегия «Divide and conquer»

Пусть у нас есть проблема **T**.

- **Divide**: разделим на несколько проблем поменьше.
- **Conquer**: разбиваем до тех пор, пока не получим такие части, для которых есть тривиальное решение.
- **Combine**: объединяя ответы на маленькие задачи, получаем ответы для больших.



Какой алгоритм из уже изученных в курсе использует эту стратегию?

Бинарный поиск

Сортировка слиянием (Merge sort)

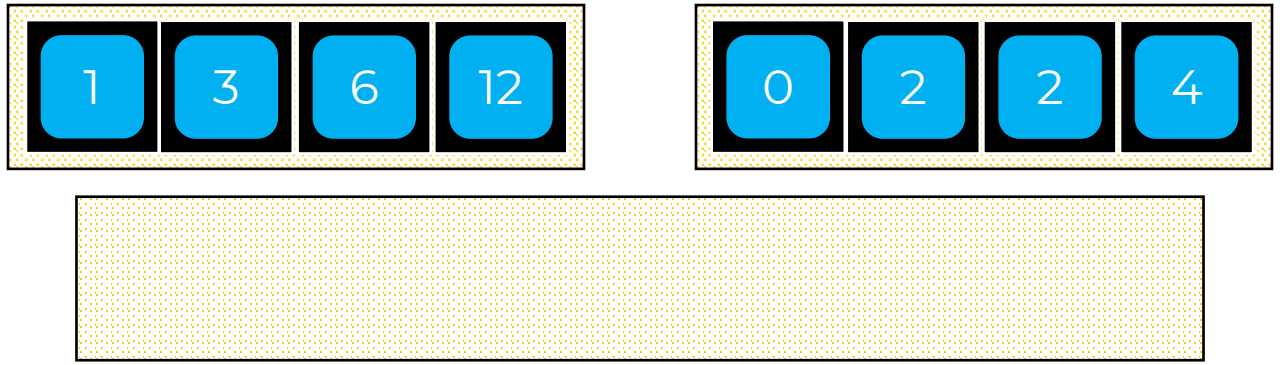
Как сортировать?

- **База**: Массив из одного элемента уже отсортирован;
- **Шаг**: Объединяем два отсортированных подмассива.
- По индукции, этот способ сможет отсортировать массив любой длины.

Как делать шаг?

Процедура Merge

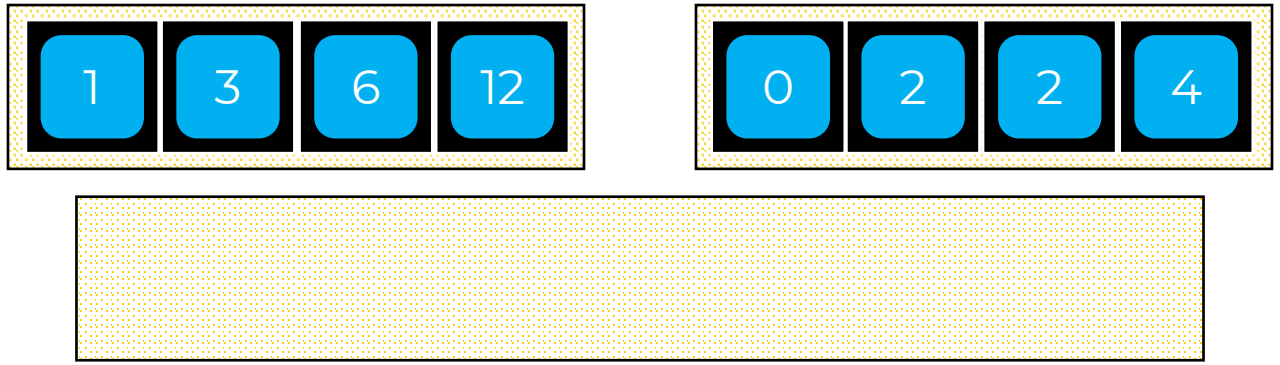
Пусть у нас уже есть два отсортированных подмассива:



Создаем пустой массив, размер которого равен суммарному размеру двух подмассивов.

Процедура Merge

Смотрим на начала подмассивов и копируем наименьший элемент из двух:

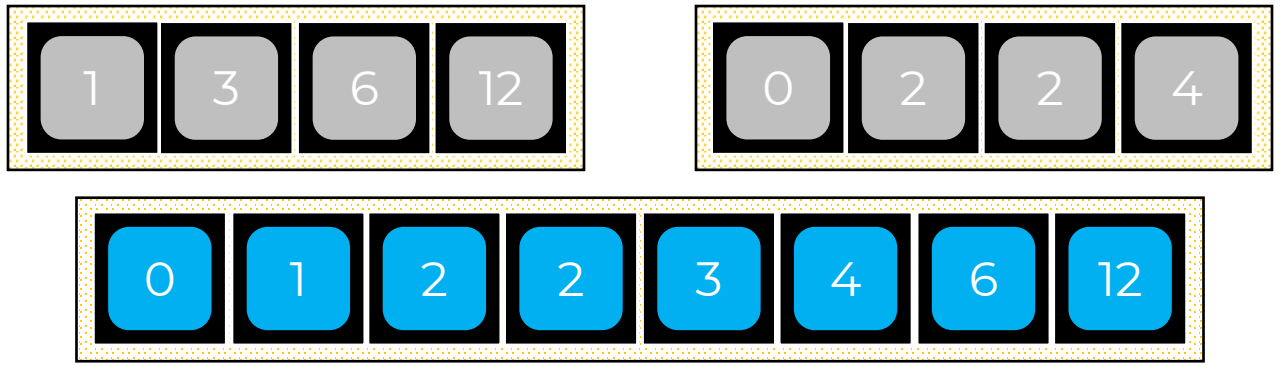


Сдвигаемся на один элемент в массиве, из которого копировали элемент на прошлом шаге.

Когда элементов в одном из массивов не осталось – просто копируем оставшиеся из другого

Процедура Merge

Смотрим на начала подмассивов и копируем наименьший элемент из двух:



Сдвигаемся на один элемент в массиве, из которого копировали элемент на прошлом шаге.

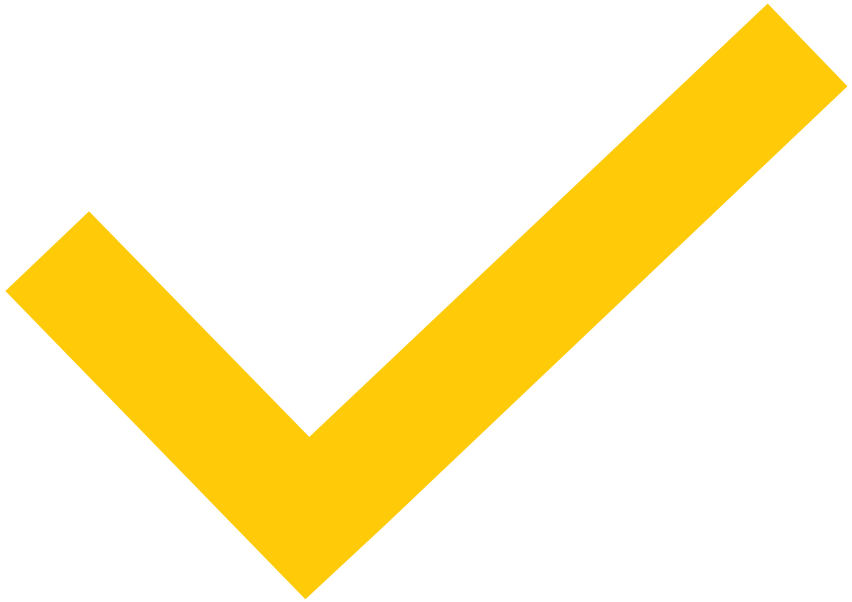
Когда элементов в одном из массивов не осталось – просто копируем оставшиеся из другого

Как произвести сортировку слиянием?

Делим массив до подмассивов длины 1

Подмассивы длины 1 уже отсортированы

Объединяем подмассивы Merge



Пишем псевдокод!

Какова асимптотика **MergeSort**?

В лучшем случае?

O(NlogN)

В среднем случае?

O(NlogN)

В худшем случае?

O(NlogN)

Затраты памяти?

По крайней мере O(N)

Быстрая сортировка

«Быстрая» сортировка

- **Divide**: Массив отсортирован тогда и только тогда, когда для каждого его элемента все элементы слева меньше или равны ему, а справа – больше или равны;
- **Conquer**: Массив из одного элемента уже отсортирован;
- **Combine**: Стараемся избежать явной комбинации для сохранения памяти.

Процедура Partition

Выбираем какой-то элемент массива (например, первый). Назовем его **опорным** (pivot). Идем с **правого** конца, пока не встретим элемент **меньше опорного**.

Когда встретили элемент, который **меньше опорного**, меняем его местами с опорным.

Процедура Partition

Дальше идем с **левого** конца, пока не встретим элемент, который **больше** **опорного**.

Когда встретили элемент, который **больше** **опорного**, меняем его местами с опорным.

Процедура Partition

Снова идем с **правого** конца, пока не встретим элемент, который **меньше** **опорного**.

Когда встретили элемент, который **меньше** **опорного**, меняем его местами с опорным.

Продолжаем операцию, пока не дойдем до **опорного** элемента.

За какое время работает процедура *Partition*?

За O(N)

Combine в QuickSort

- После каждой процедуры Partition опорный элемент занимает свое итоговое место.
- Перемещение элементов происходит «на месте».

Благодаря этим двум факторам мы можем обойтись без явного шага **Combine**.

Пишем **псевдокод!**

Какова асимптотика **QuickSort**?

В лучшем случае?

O(NlogN)

В среднем случае?

O(NlogN)

В худшем случае?

$O(N^2)$

Затраты памяти?

O(logN)

Что лучше:
MergeSort или
QuickSort?

k-я порядковая статистика

k-й порядковой статистикой набора элементов линейно упорядочиваемого множества называется такой его элемент, который является **k**-м элементом набора в порядке сортировки.

TL;DR: Ответ на вопрос: «Какой элемент окажется на **k**-м месте?»

Как искать k-ю порядковую статистику?

Подсказка: вспомнить процедуру *Partition*

Действительно, если опорный элемент оказался на k-й позиции, то k-я порядковая статистика найдена!

Если же нет, рекурсивно найдем ее в одной из частей массива.

Какова асимптотика поиска k-й порядковой статистики?

В лучшем случае?

O(N)

В среднем случае?

O(N)

В худшем случае?

$O(N^2)$

Затраты памяти?

O(1)

Какова оптимальная асимптотика сортировки в общем случае?

Вопрос нетривиальный :^)

Воспользуемся теорией информации, чтобы получить ответ.

Найдем оптимальную асимптотику

Сколько информации дает нам каждое сравнение?

1 бит

Сколько существует вариантов упорядочения массива длины N?

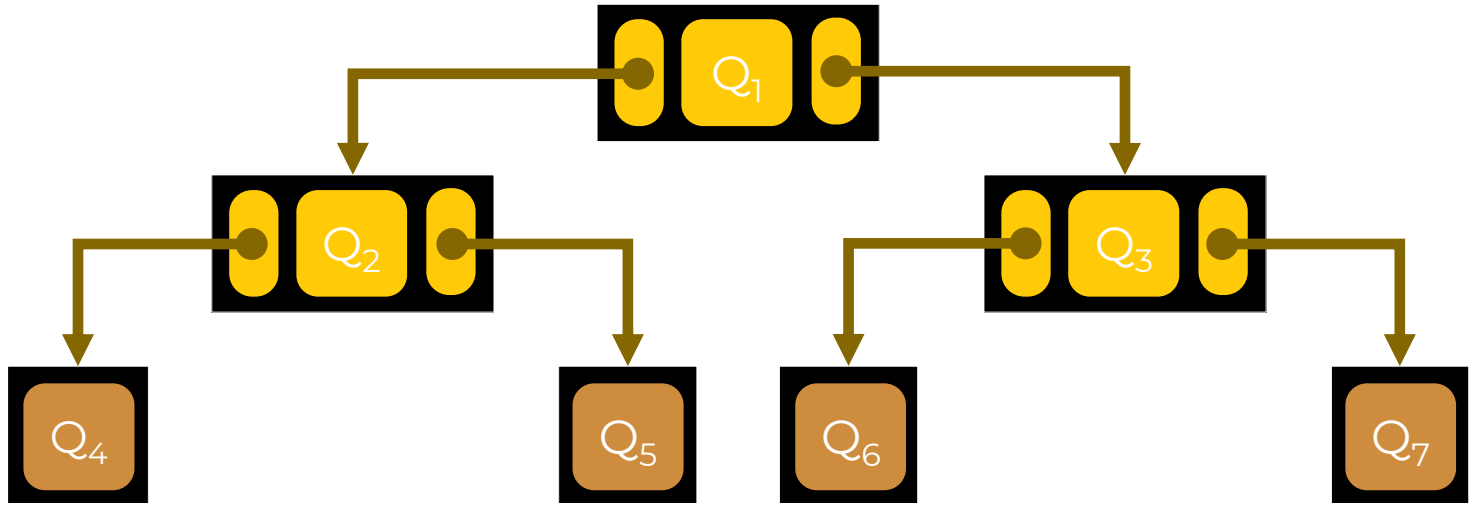
N!

Сколько бит нам необходимо, чтобы однозначно указать на вариант сортировки?

$\log_2 N!$

Или проще:

- Каждое сравнение дает 1 бит информации.
- Строим бинарное дерево с **N!** листьями.
- Его минимальная высота – $\mathbf{\log_2(N!)}$.



Найдем асимптотику количества операций по формуле Стирлинга...

$$\log_2 N! = \log_2 \left(\sqrt{2\pi N} \left(\frac{N}{e} \right)^N \right) =$$

$$= N \log N + O(N) =$$

$$= \Omega(N \log N)$$

Оценка снизу!

Сортировка подсчетом

Дан массив целых числовых элементов, находящихся в определенном диапазоне **[A; B]**:

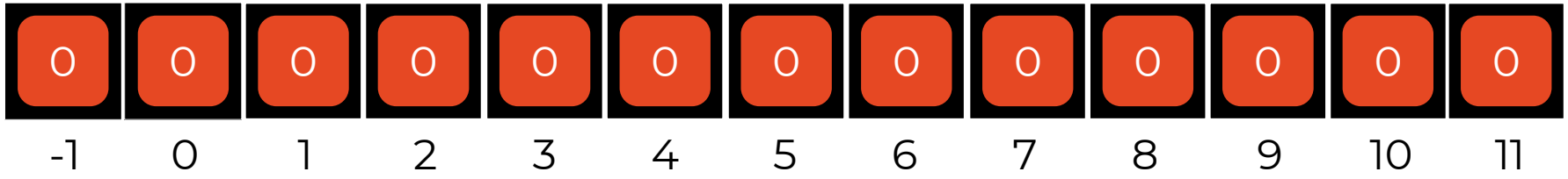
В этом примере мы знаем, что все ключи лежат в **[-1; 11]**

Min и **Max** можно найти за **O(N)**.

Сортировка подсчетом

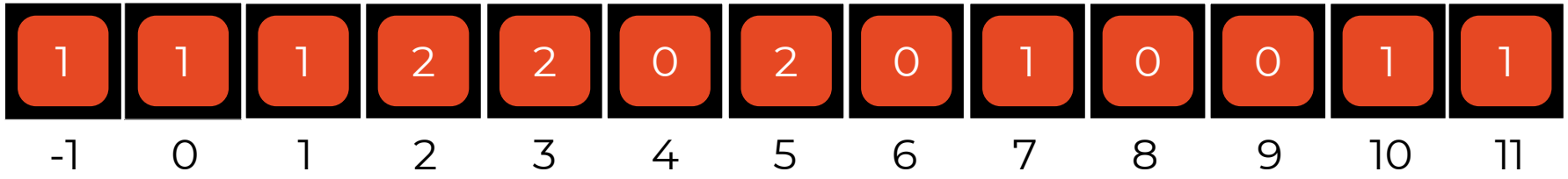
Дан массив целых числовых элементов, находящихся в определенном диапазоне **[A; B]**:

Создадим массив размера **B - A + 1** = 11 - (-1) + 1 = **13** и заполним его нулями



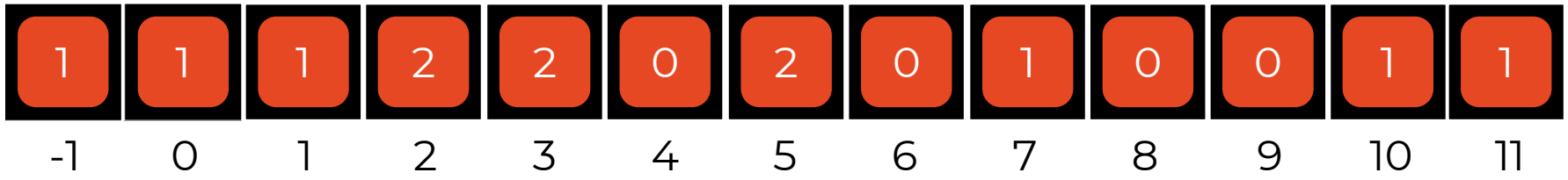
Сортировка подсчетом

Затем проходимся по массиву один раз и для каждого индекса делаем: **count[arr[i] - A] += 1**



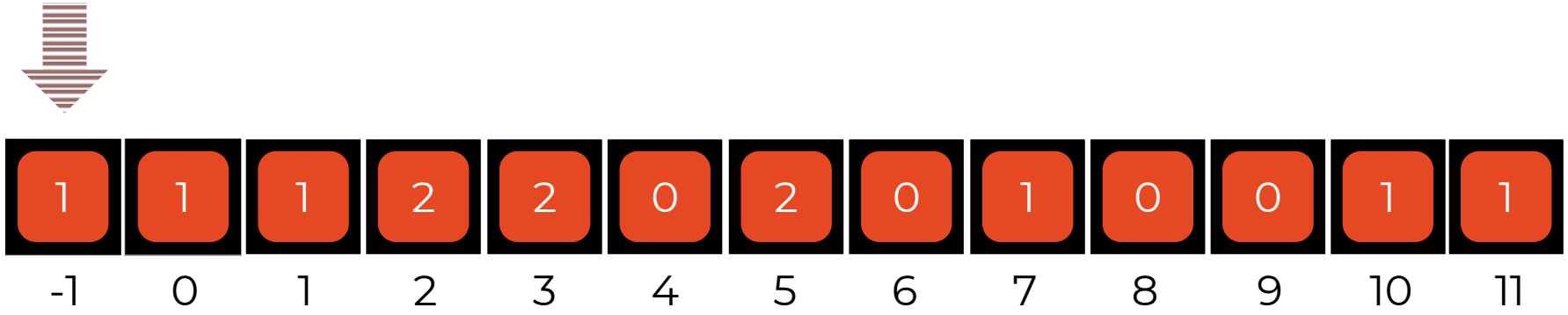
Сортировка подсчетом

Теперь создаем пустой массив размера исходного



Сортировка подсчетом

И заполняем пустой массив по массиву с подсчетами.

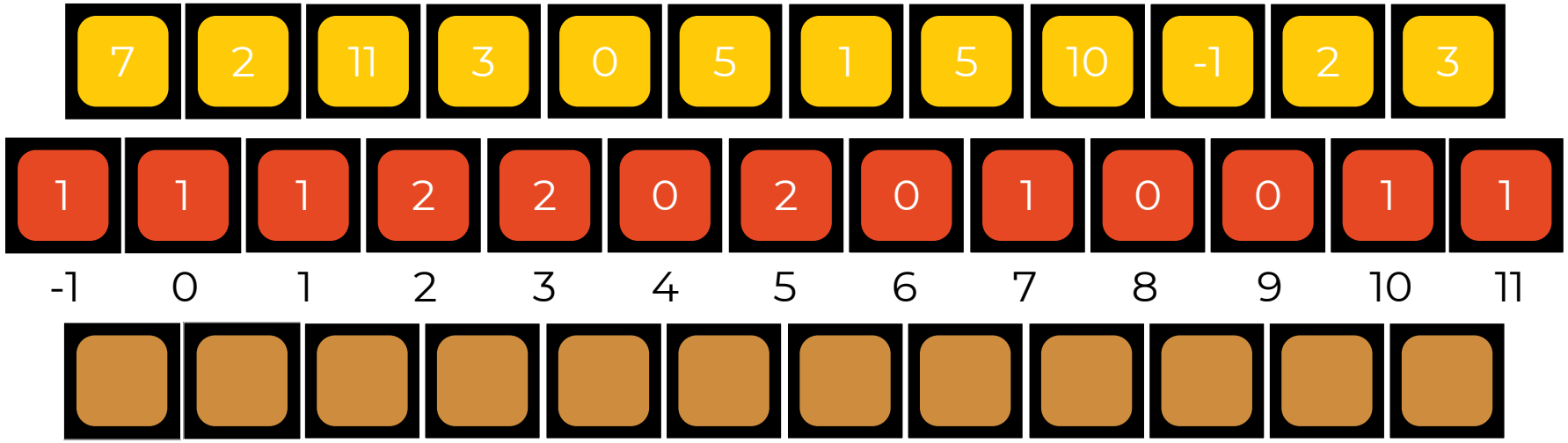


Что не так?

Каждый элемент – это пара:
(Ключ, значение).
Непонятно, что делать со значениями.

Сортировка подсчетом

Придется делать иначе: сделаем так, чтобы массив счетчиков указывал, на какую позицию нам нужно ставить элемент...

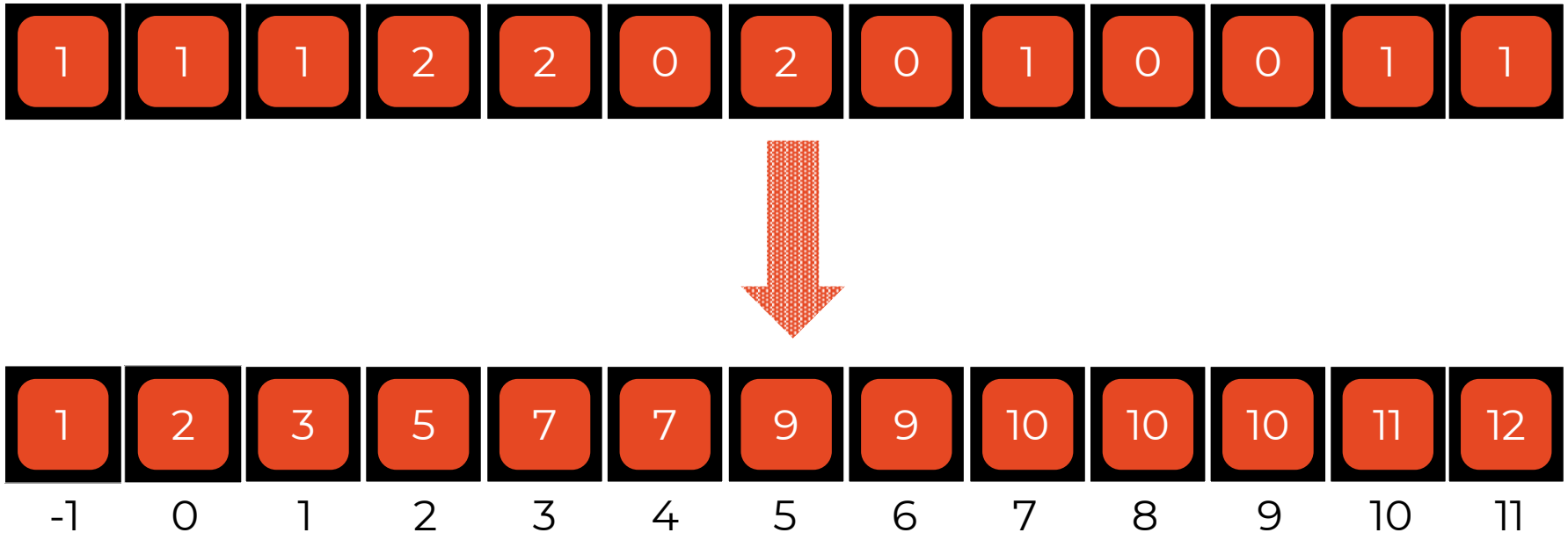


А сам новый массив будем заполнять, идя по старому в обратную сторону.

Сортировка подсчетом

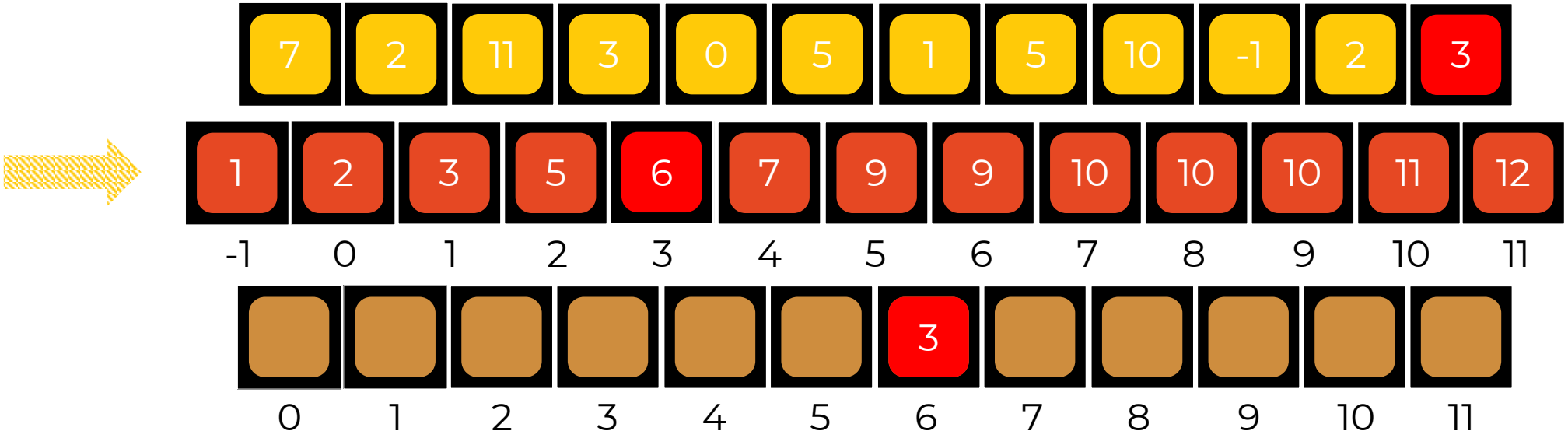
Для этого делаем массив счетчиков кумулятивным:

Проходимся **count[i] += count[i - 1]**.



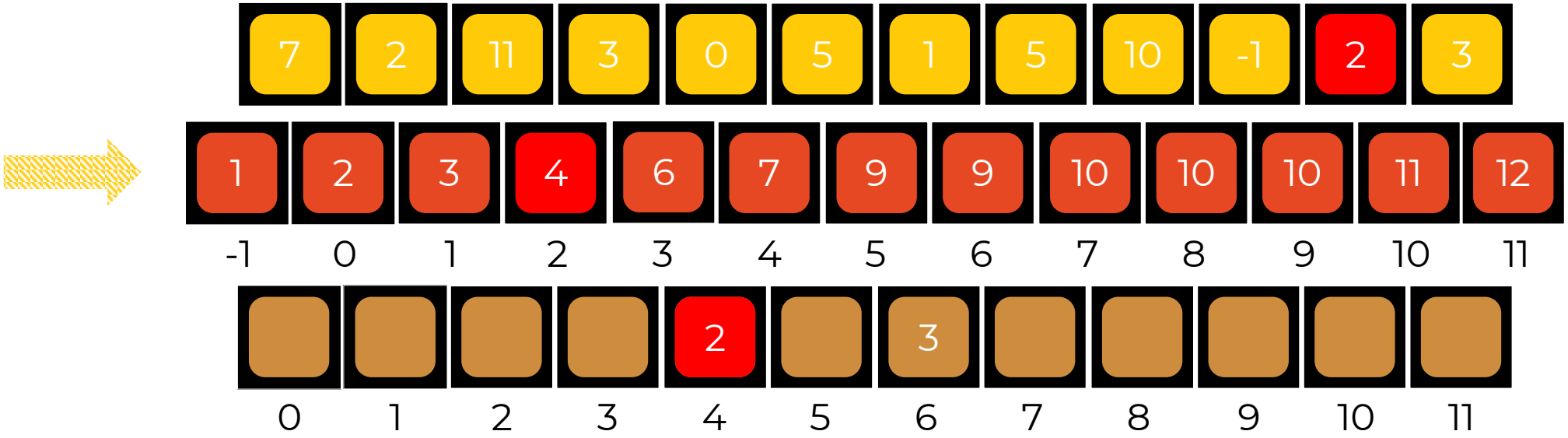
Сортировка подсчетом

Теперь массив счетчиков содержит позицию последнего элемента с таким номером **+ 1**.



Сортировка подсчетом

Теперь массив счетчиков содержит позицию последнего элемента с таким номером **+ 1**.



Какова асимптотика сортировки подсчетом?

В общем случае?

O(N + B - A)

Затраты памяти?

O(N + B - A)

Какова мораль?

Каждая задача может иметь лучшее решение, если у вас есть дополнительная информация!

Спасибо за внимание!