Удивительная история развития сортировки в JDK

Владимир Ярославский

Руководитель направления, Сбербанк

пятница, 13 октября 2023

Сортировка в JDK

`java.util.Arrays.sort()` — какая сортировка будет вызвана?

Сортировка в JDK

для объектов — Timsort

для простых типов данных (int, long, float, ...) — Dual-Pivot Quicksort

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Сортировка в JDK

Зачем нам два алгоритма: Timsort и Dual-Pivot Quicksort

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Сортировка в JDK 6

Методы `java.util.Arrays.sort()`

Объекты — Merge sort

стандартная реализация

Простые типы данных (int, long, float, ...) — классический Quicksort

реализация Jon L. Bentley, M. Douglas McIlroy

Сортировка в JDK 7

Методы `java.util.Arrays.sort()`

Объекты — Timsort

New! Tim Peters

Простые типы данных (int, long, float, ...) — Dual-Pivot Quicksort

New! Vladimir Yaroslavskiy,

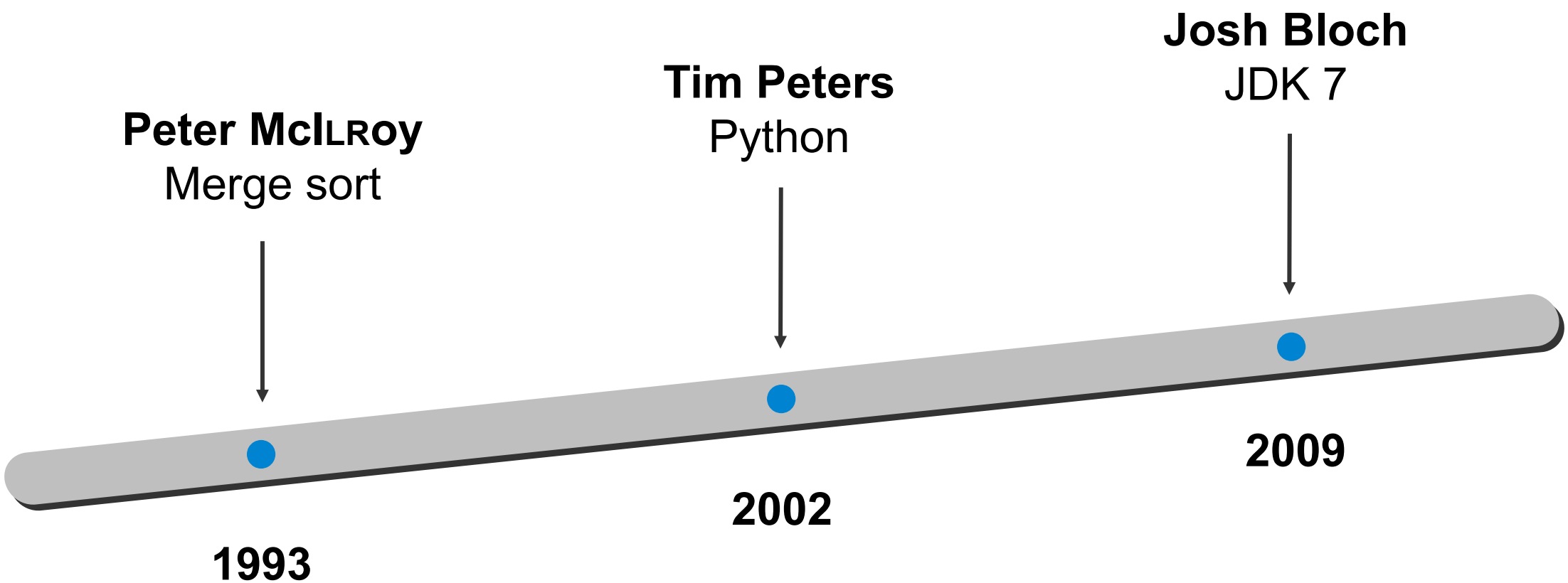
Jon Bentley, Josh Bloch, 2009

Сортировка в JDK

Сортировка Dual-Pivot Quicksort используется более 14 лет не только в JDK, но и в Android

Содержание

История Timsort

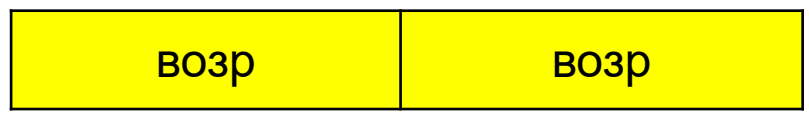


Merge sort (сортировка слиянием)

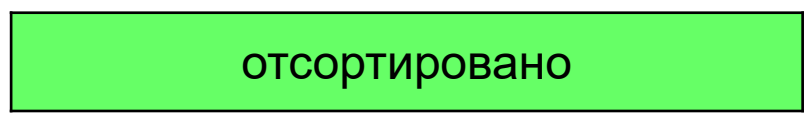
Алгоритм

1. Массив делится на две равные части

2. Каждая часть сортируется рекурсивно



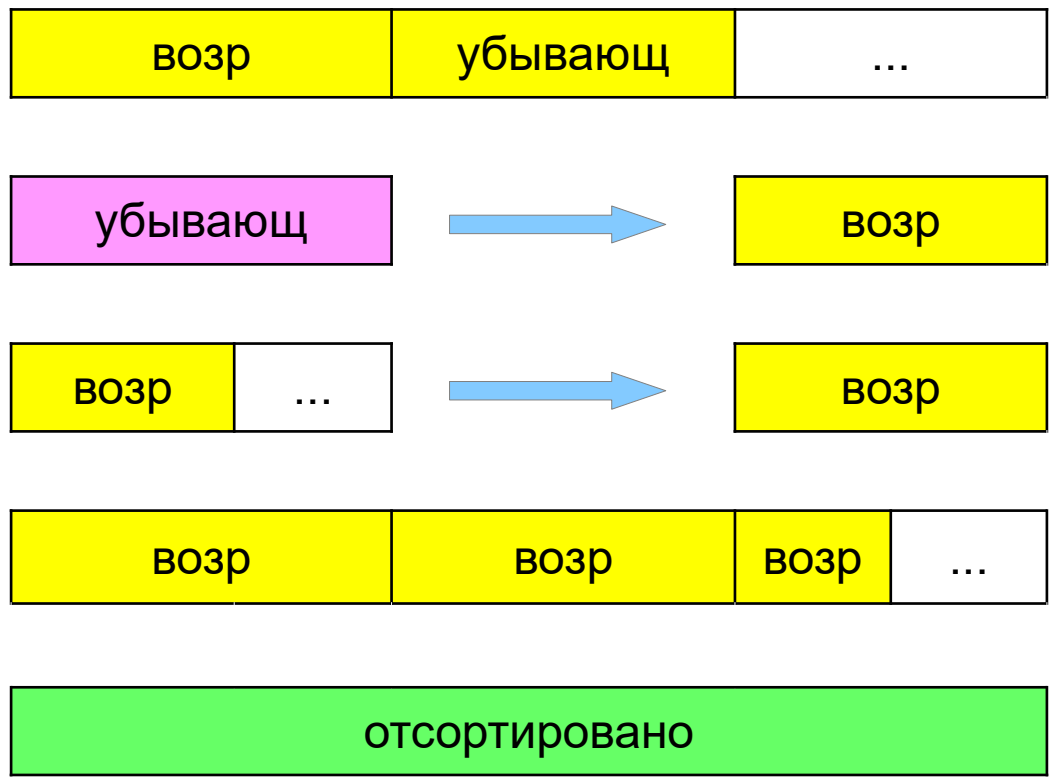
3. Обе части сливаются обратно в массив



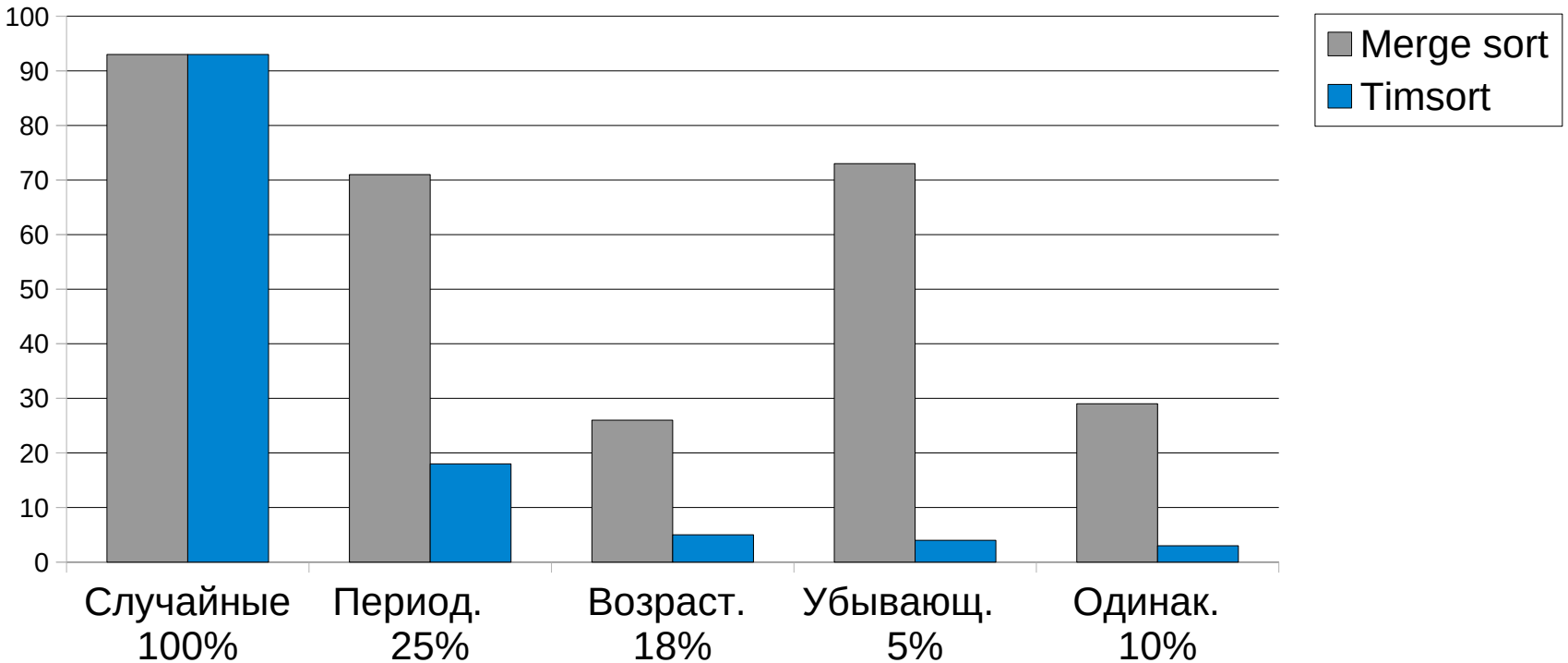
Timsort

Алгоритм

1. Ищутся возр. и убыв. послед-сти
2. Убыв. последовательности переворачиваются
3. Короткие последовательности расширяются до мин. длины
4. Последовательности сливаются обратно в массив



Timsort vs. Merge sort



Сравнение алгоритмов на тестовом наборе Jon'a Bentley из 1 000 000 элементов

Тестовый набор Jon'a Bentley

Типы

random:	a[i] = random(m)
sawtooth:	a[i] = i % m
stagger:	a[i] = (i * m) % n
plateau:	a[i] = min(i, m)
...	...

Модификации

copy(a)
reverse(a, 0, n)
reverse(a, 0, n/2)
reverse(a, n/2, n)
...

Параметры

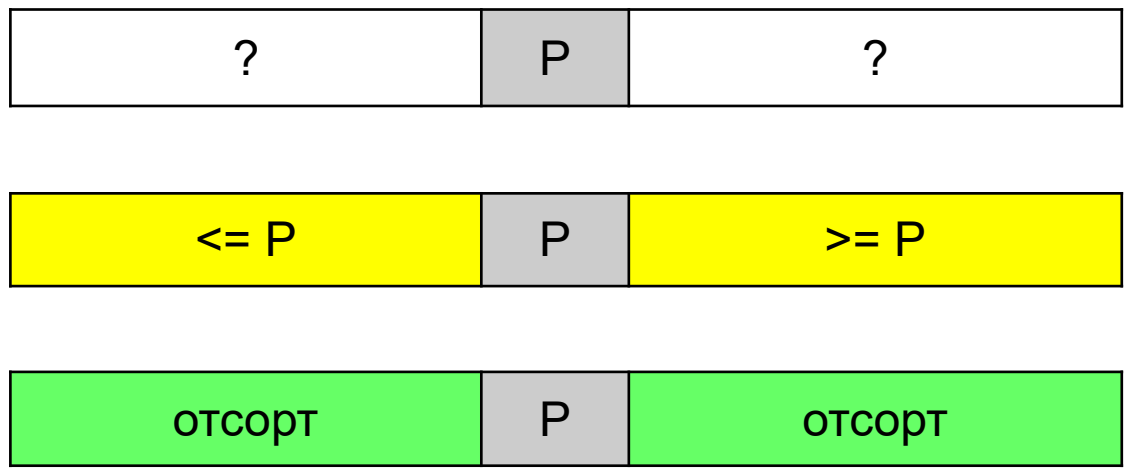
m = 1, 2, 4, 8, ... , 2*n, где n — длина массива
n = 10, 100, 1 000, 10 000, 1 000 000

Содержание

Классический Quicksort

Алгоритм (C. A. R. Hoare, 1959)

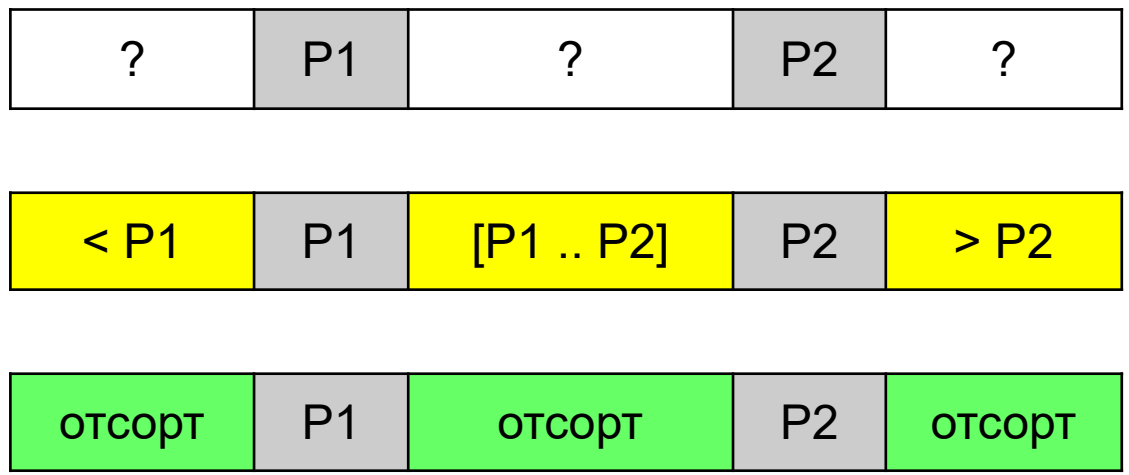
1. Выбирается один опорный элемент, P
2. Элементы перегруппируются
3. Каждая часть сортируется рекурсивно



Один хорошо, а два лучше

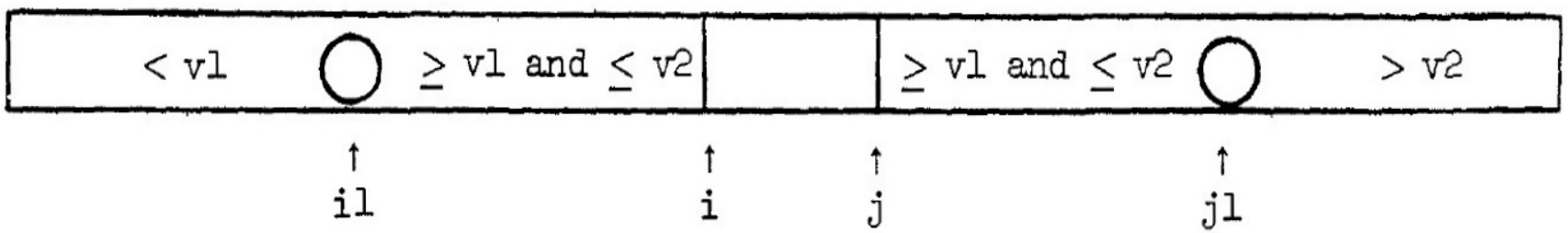
Алгоритм

1. Выбираются два опорных элемента, P1 <= P2
2. Элементы перегруппируются
3. Каждая часть сортируется рекурсивно



Извлечен из пыльного сундука истории

“Two-partion” Quicksort



“This is nearly 2.5 times worse than the exchanges required by our other algorithms. We need go no further with analysis, for we can never hope to recoup this loss.”

Robert Sedgewick, PhD Thesis "Quicksort", 1975
Chapter 5, pp.150 – 159

Как все начиналось

август 2008 — идея: что если взять 2 опорных элемента?

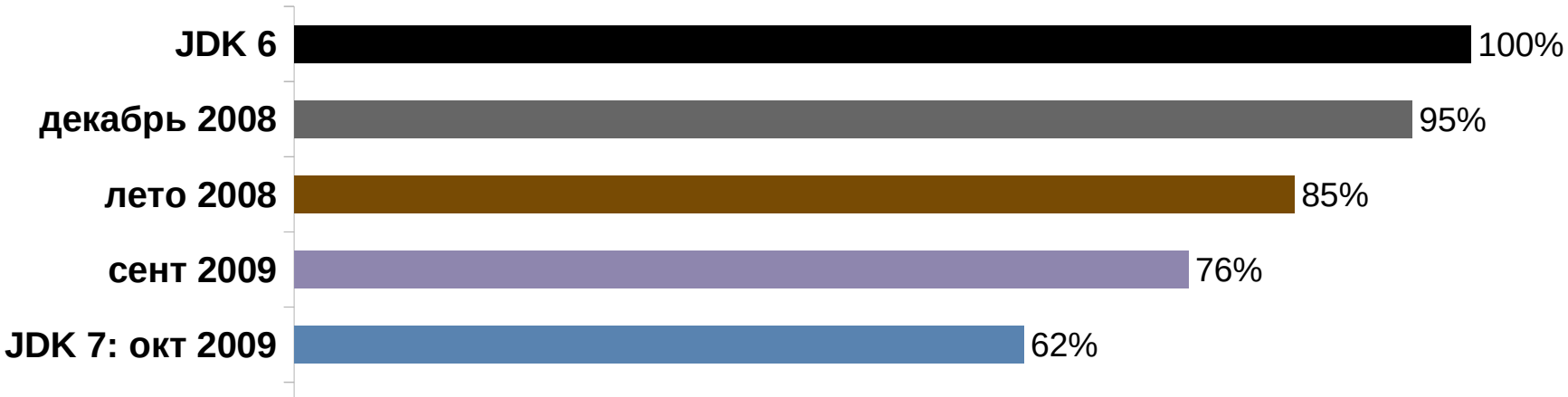
сентябрь 2008 — проверка идеи на списках

декабрь 2008 — черновая версия для массивов

весна 2009 — обсуждение сортировки с Joch Bloch

лето 2009 — улучшение сортировки

Как все начиналось

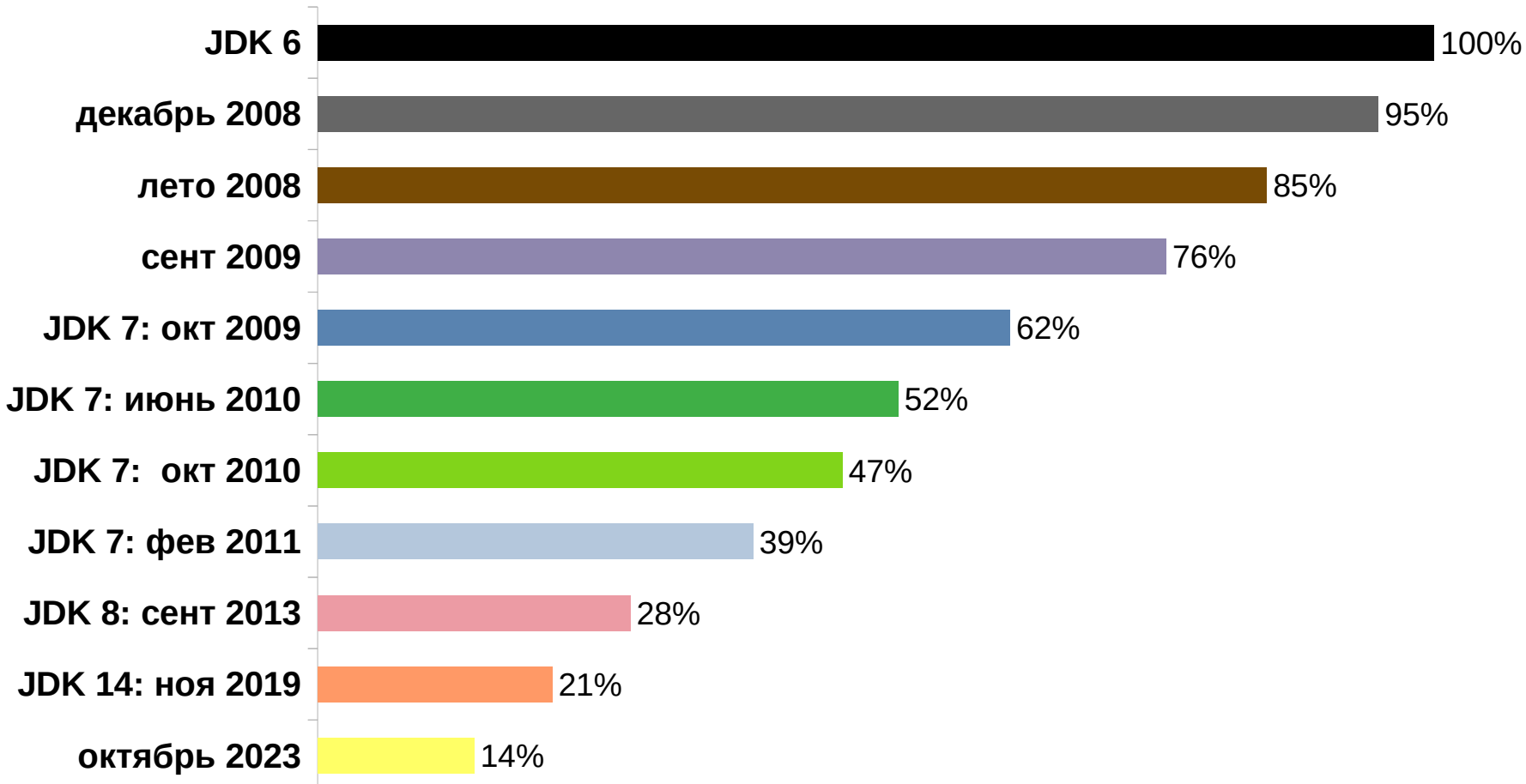


август 2009 — Joch Bloch: невозможно отмахнуться от результатов

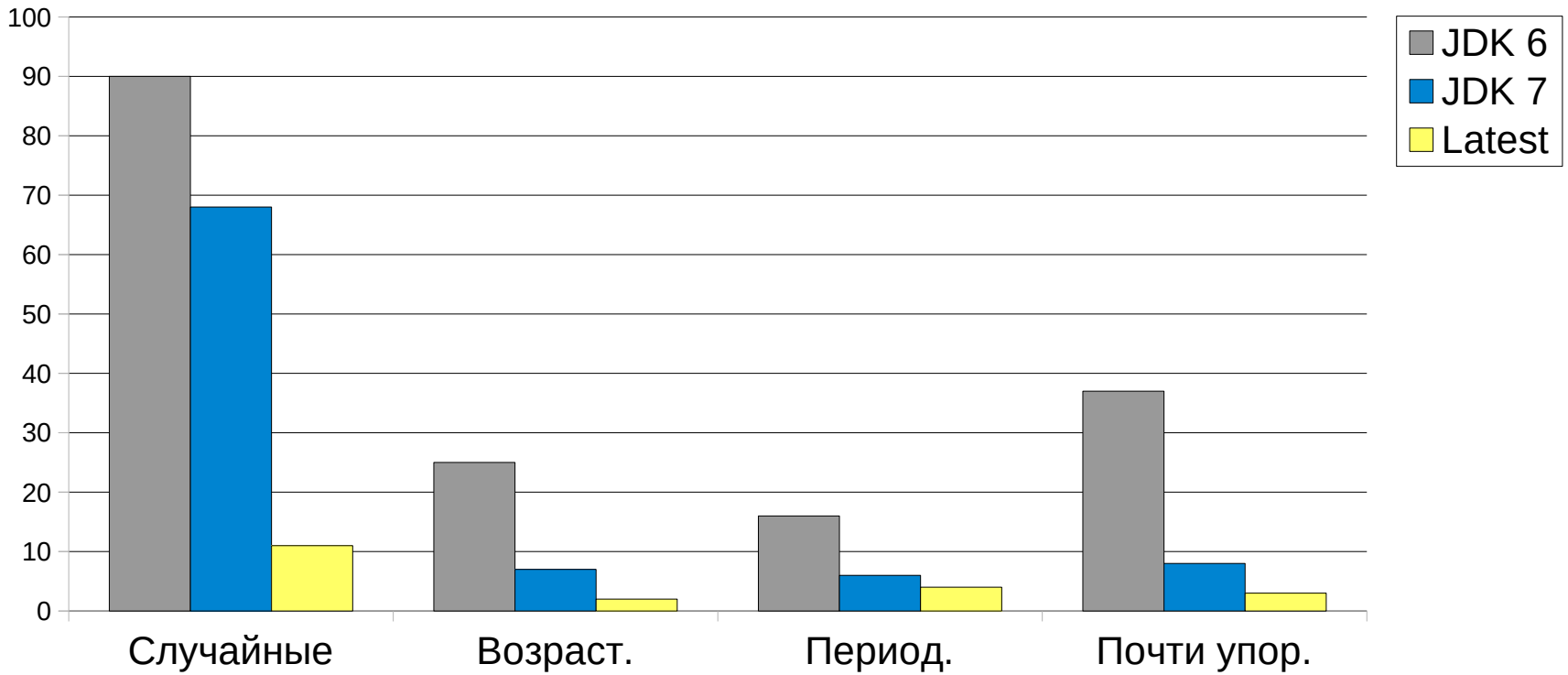
сентябрь 2009 — Jon’a Bentley дал ценные советы по улучшению

октябрь 2009 — Joch Bloch помог интегрировать сортировку в JDK

Развитие Dual-Pivot Quicksort



Dual-Pivot Quicksort vs. Quicksort



Сравнение алгоритмов на тестовом наборе Jon'a Bentley из 1 000 000 элементов

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

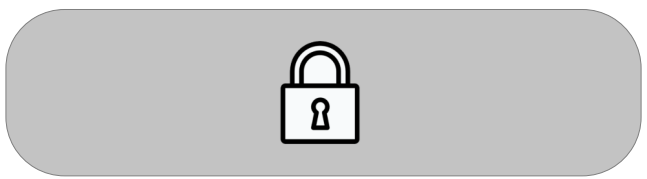
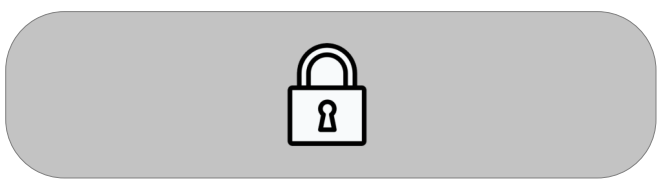
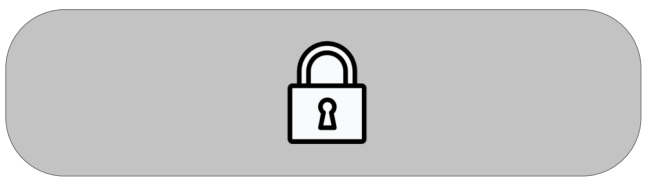
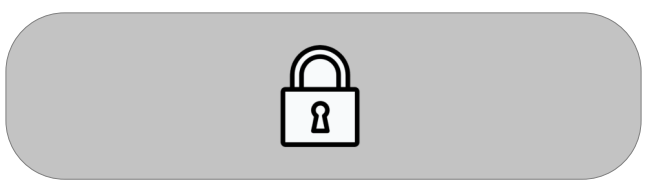
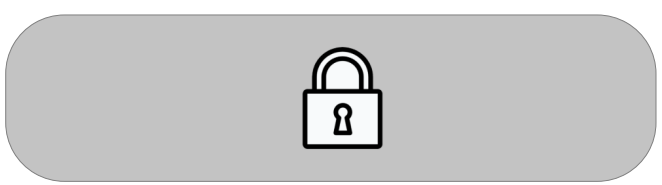
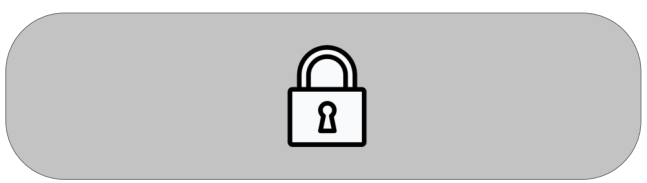
Сортировка вещественных чисел

Параллельная сортировка

Лучшая сортировка

Dual-Pivot Quicksort



Сортировка небольших массивов

Insertion sort (сортировка вставками)

Сортировка небольших массивов

```
for (int j, i = left+1; i <= right; ++i) {
  int ai = a[i];
  for (j = i-1;    j >= left &&    ai < a[j]; --j) {
    a[j+1] = a[j];
  }
  a[j+1] = ai;
}
```

Сортировка небольших массивов

```
for (int j, i = left+1; i <= right; ++i) {
  int ai = a[i];
  for (j = i-1; /* j >= left && */ ai < a[j]; --j) {
    a[j+1] = a[j];
  }
  a[j+1] = ai;
}
```

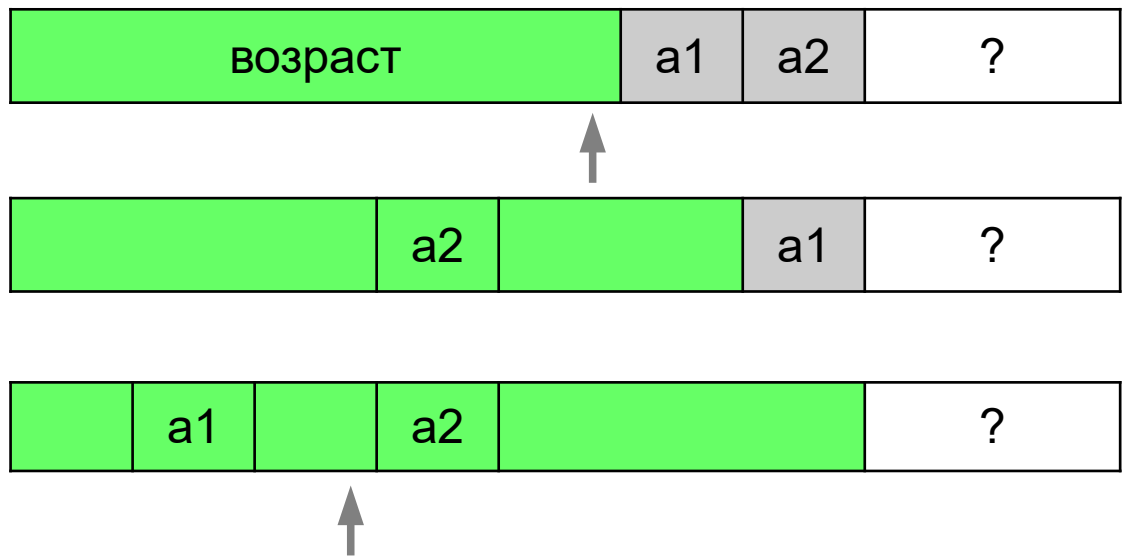
Первый опорный элемент играет роль часового

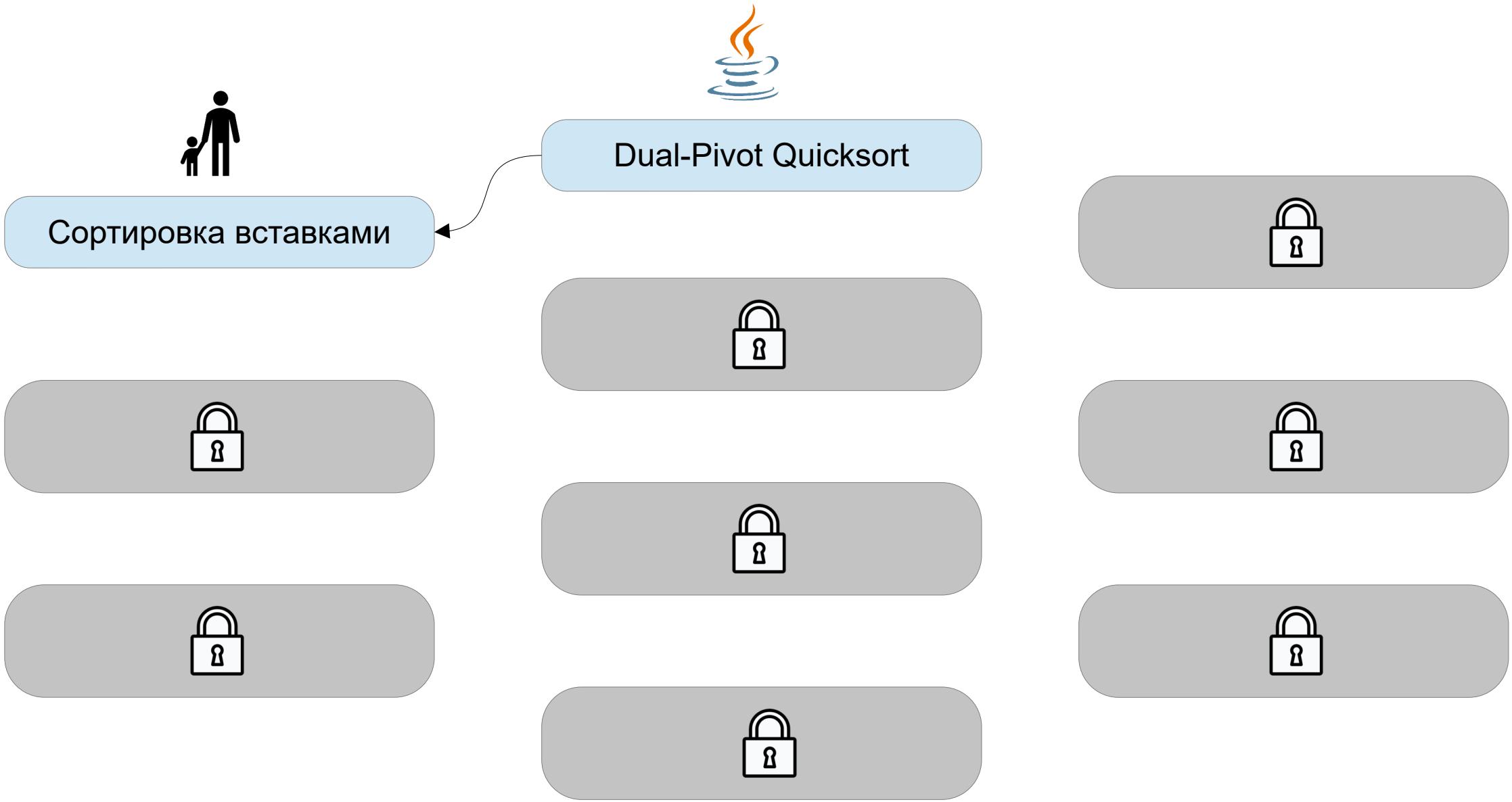
P1 <	P1	P1 <= ai <= P2	P2	> P2

Pair insertion sort (парная сортировка вставками)

Алгоритм

1. Берем два элемента, a2 >= a1
2. Вставляем больший, a2
3. Вставляем элемент a1 перед элементом a2




Dual-Pivot Quicksort
Сортировка вставками

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

Сортировка вещественных чисел

Параллельная сортировка

Лучшая сортировка

Оптимальный Quicksort

Один опорный элемент — хорошо

Два опорных элемента — еще лучше

Три опорных элемента — уже хуже

Оптимальное количество — ??

Оптимальный Quicksort

Один опорный элемент — хорошо

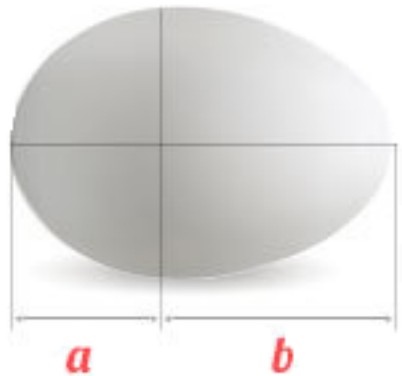
Два опорных элемента — еще лучше

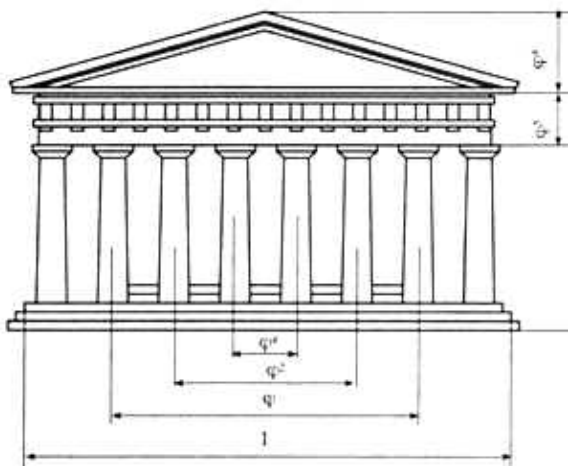
Три опорных элемента — уже хуже

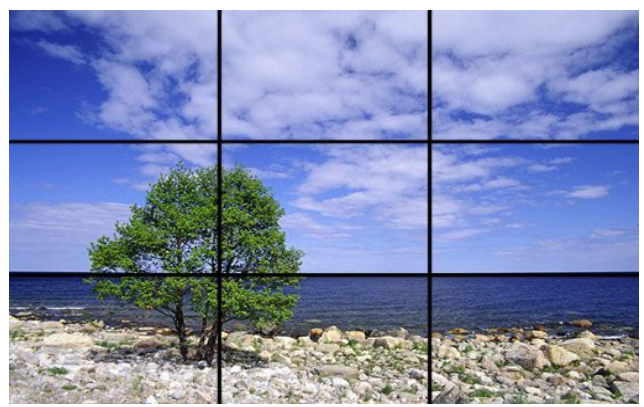
Оптимальное количество — 1.6180339...

«Золотое» сечение

$\varphi = 1.6180339...$





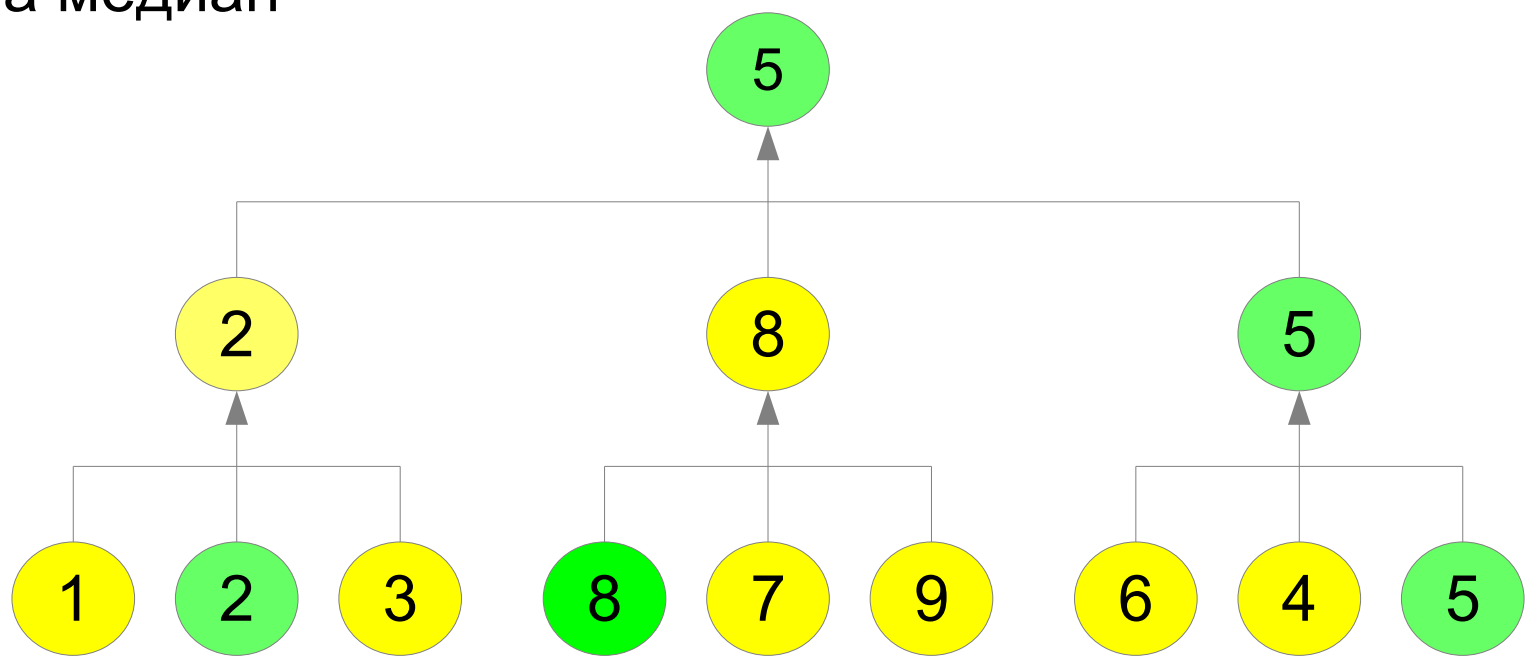


Опорные элементы

Случайные элементы

Средние или крайние

Медиана медиан



Опорные элементы

Сколько отсортированных кандидатов рассматривать?

5

На каких позициях?

——————— **[1]** — [2] — [3] — [4] — **[5]** ———————

+шаг середина -шаг

На каком расстоянии от края?

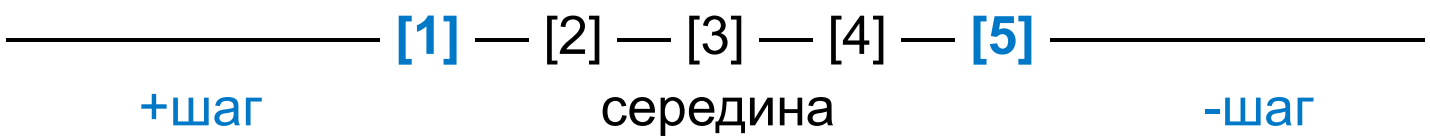
шаг = длина * 0.381966 // делим в "золотом" сечении

Опорные элементы

Сколько отсортированных кандидатов рассматривать?

5

На каких позициях?

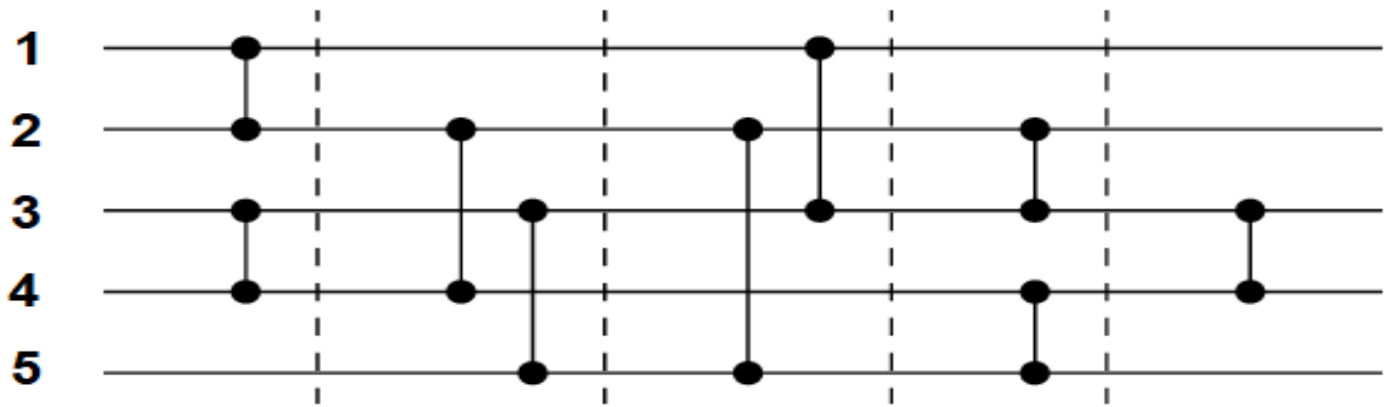


На каком расстоянии от края?

шаг = (длина >> 2) + (длина >> 3) + (длина >> 7);

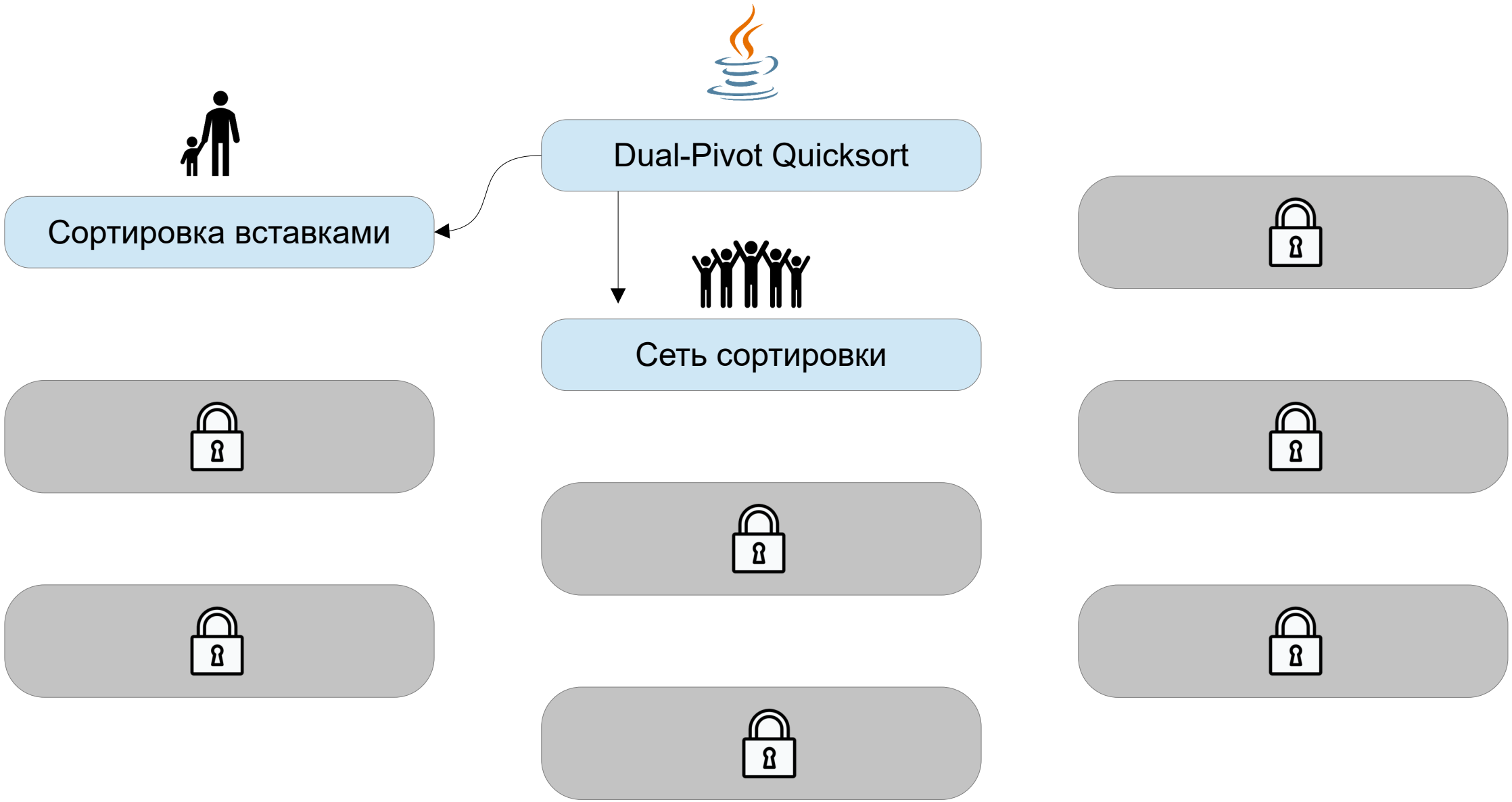
Опорные элементы

Как отсортировать 5 элементов для выбора опорных?



Сеть сортировки (sorting network) — 9 сравнений

(сортировка вставками — 10 сравнений)


Dual-Pivot Quicksort
Сортировка вставками
Сеть сортировки

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

Сортировка вещественных чисел

Параллельная сортировка

Лучшая сортировка

Разбиение массива

```
if (pivot1 != pivot2) {
```

< P1	P1 <= .. <= P2	> P2

```
} else { // Разбиение из задачи о национальном флаге
```

< P	== P (отсортировано)	> P

```
}
```

Разбиение массива

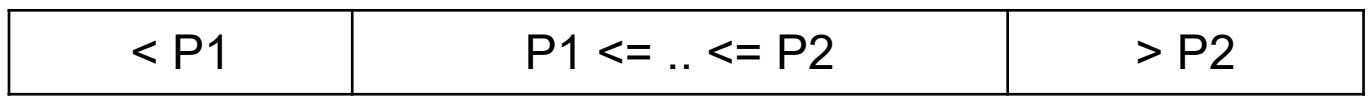
Задача о национальном флаге

```
0 0 0 0 1 1 1 1 ? ? ? ? ? 2 2 2 2
          ^       ^         ^
          |       |         |
          lo      mi        hi
```

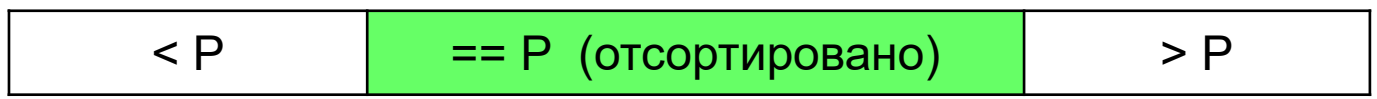
https://leetcode.com/problems/sort-colors

Разбиение массива

```
if (pivot1 != pivot2) {
```




```
} else { // "Dutch National Flag" (DNF)
```

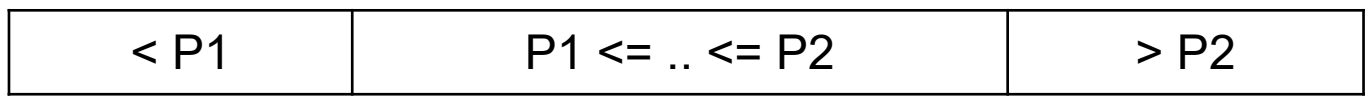



```
}
```

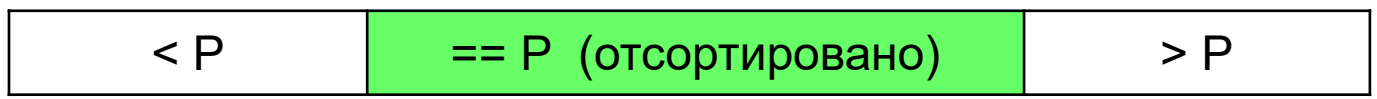
На повторяющихся данных классический Quicksort быстрее двухопорного на 10-25%

Разбиение массива

```
if  (a[e1] != a[e2] != a[e3] != a[e4] != a[e5])  {
```

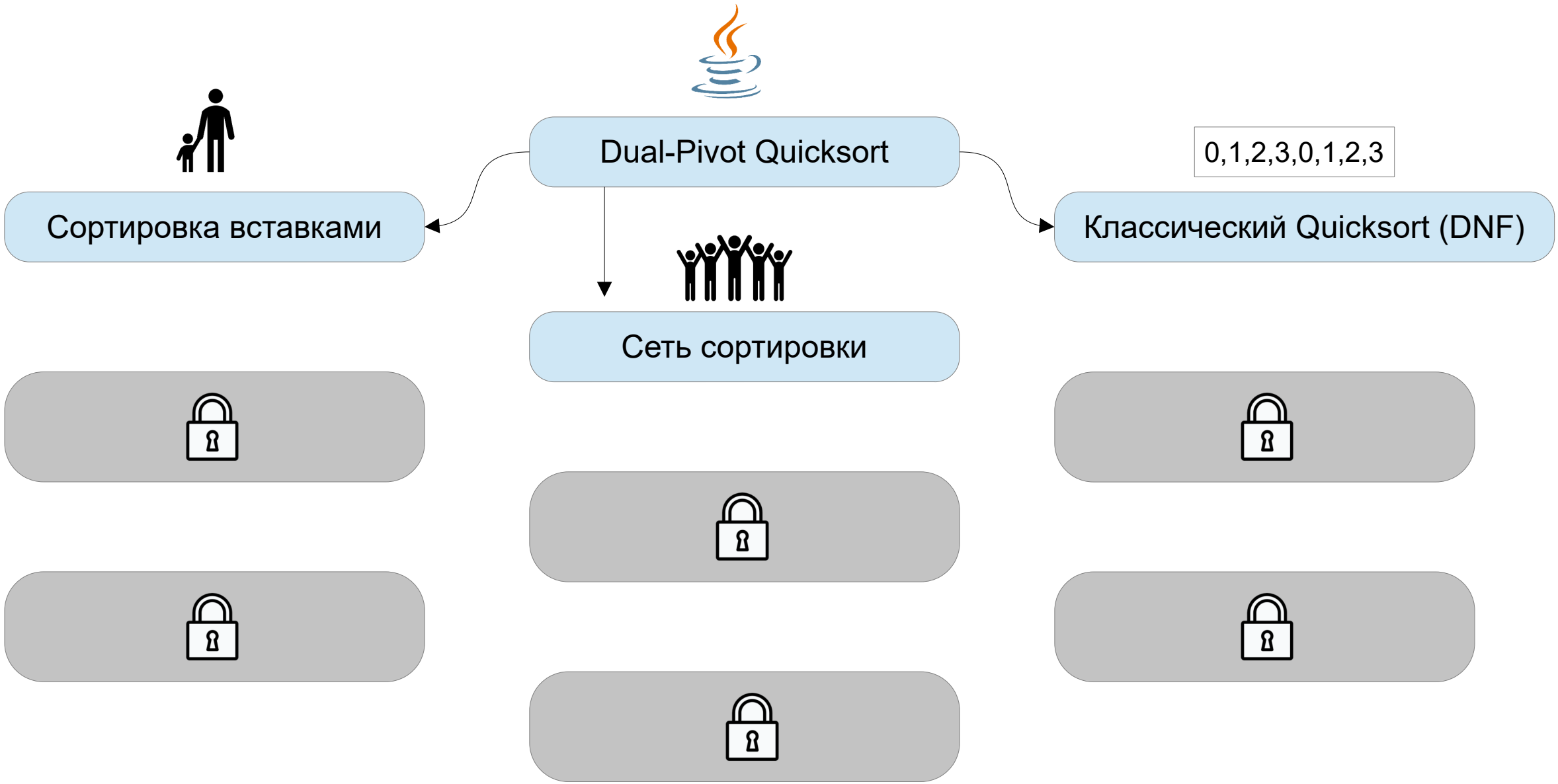



```
}  else  { // "Dutch National Flag" (DNF)
```




```
}
```

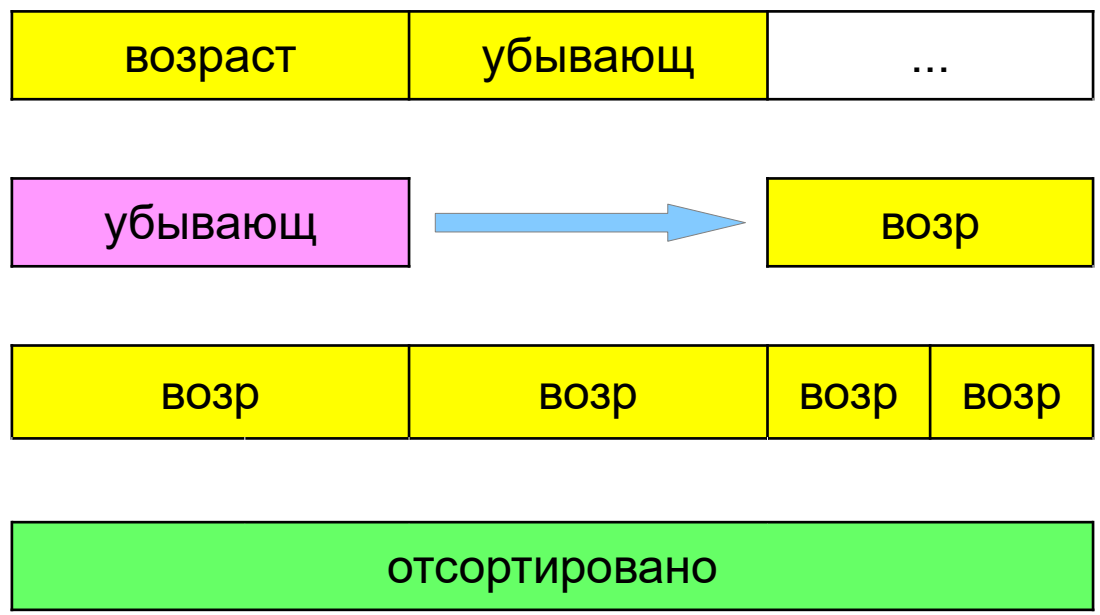
На повторяющихся данных классический Quicksort быстрее двухопорного на 10-25%


Dual-Pivot Quicksort
Сортировка вставками
Сеть сортировки
Классический Quicksort (DNF)
0,1,2,3,0,1,2,3

Merging sort — сортировка [почти] упорядоченных данных

Алгоритм

1. Ищутся возр. и убыв. последовательности
2. Убыв. последовательности переворачиваются
3. Последовательности сливаются обратно в массив



Merging sort — сортировка [почти] упорядоченных данных

Количество последовательностей < MAX_COUNT

Допустимо нестрогое возрастание

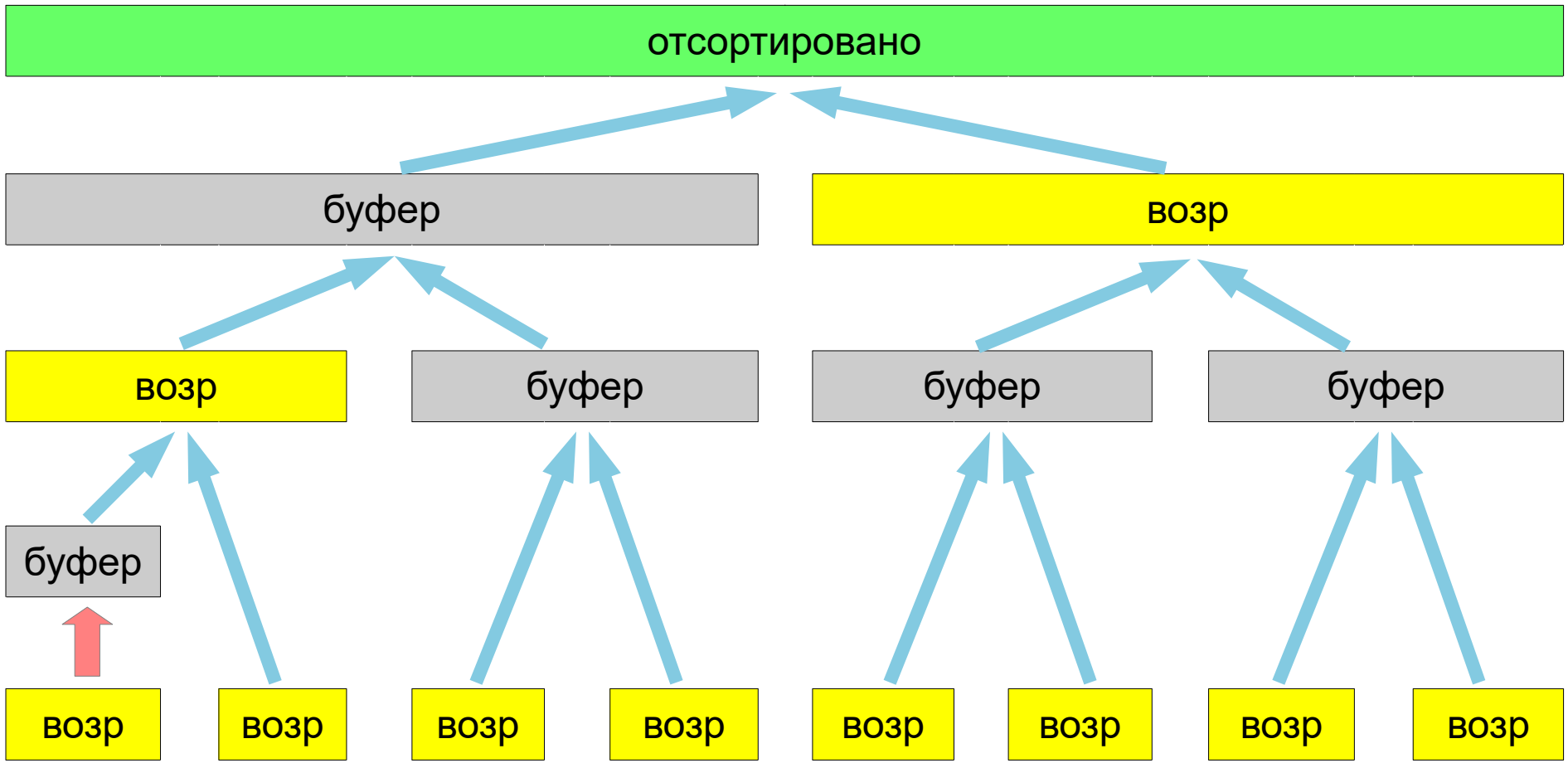
a[k] <= a[k + 1] <= a[k + 2] <= ... <= a[m]

Допустимо нестрогое убывание

a[k] >= a[k + 1] >= a[k + 2] >= … >= a[m]

Короткие последовательности не расширяем

Merging sort — оптимальное слияние в Dual-Pivot Quicksort

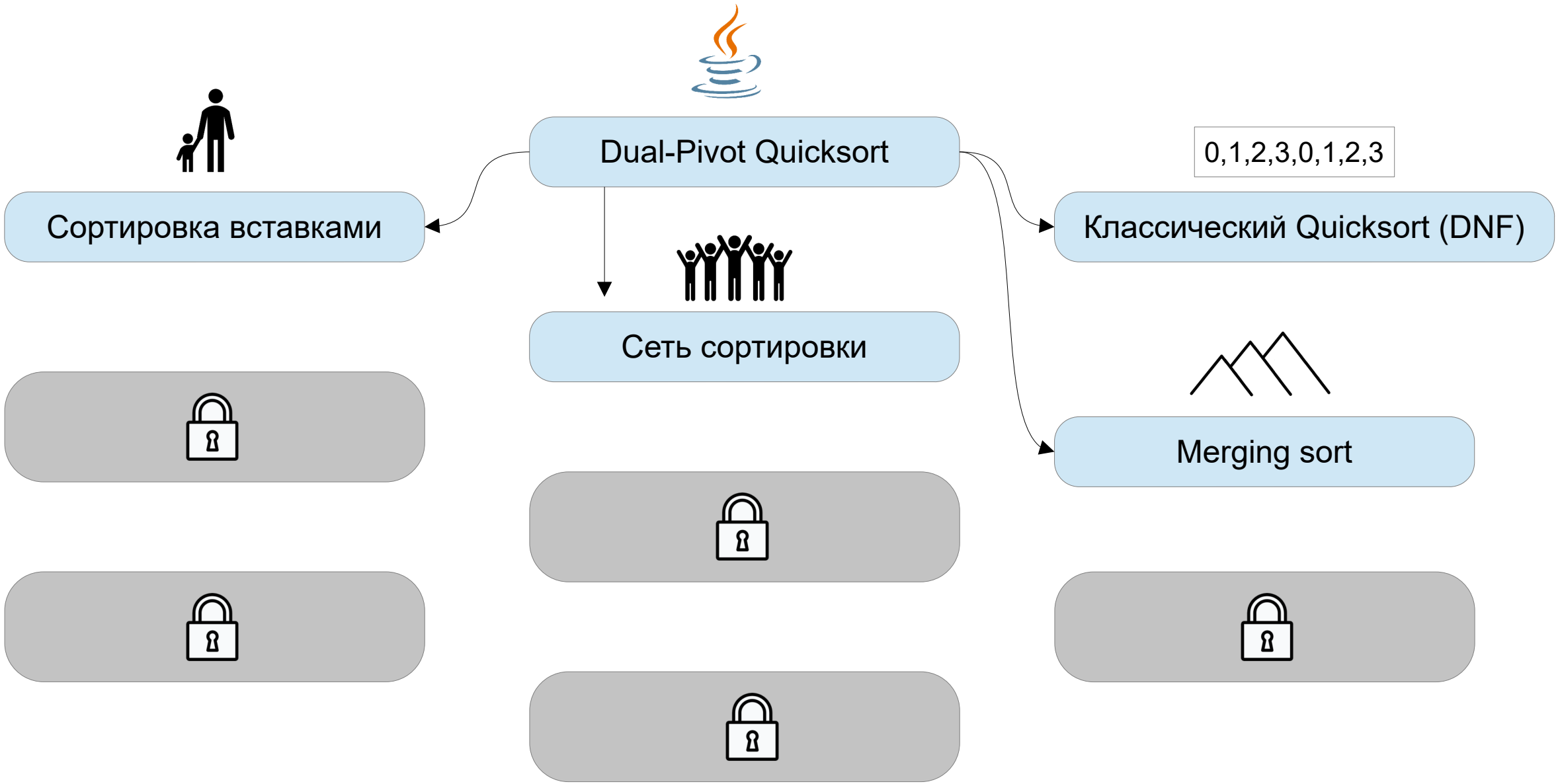


Выделение дополнительной памяти под буфер*

Особенности:

- запрашиваем "грязную" память
- если нет свободной памяти, то не падаем с OutOfMemoryError, а переключаемся только на inline алгоритмы

* in progress


Dual-Pivot Quicksort
Сортировка вставками
Сеть сортировки
0,1,2,3,0,1,2,3
Классический Quicksort (DNF)
Merging sort

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

Сортировка вещественных чисел

Параллельная сортировка

Лучшая сортировка

Сортировка вещественных чисел

A == B => A и B: идентичное представление в памяти?

Да / Нет ?

Сортировка вещественных чисел

A == B => A и B: идентичное представление в памяти?

Да / Нет ?

A != B => A и B: различное представление в памяти?

Да / Нет ?

Сортировка вещественных чисел

A == B => A и B: идентичное представление в памяти?

Да — математика

A != B => A и B: различное представление в памяти?

Да — математика

Сортировка вещественных чисел

A == B => A и B: идентичное представление в памяти?

Нет! — Java

A != B => A и B: различное представление в памяти?

Нет! — Java

Сортировка вещественных чисел

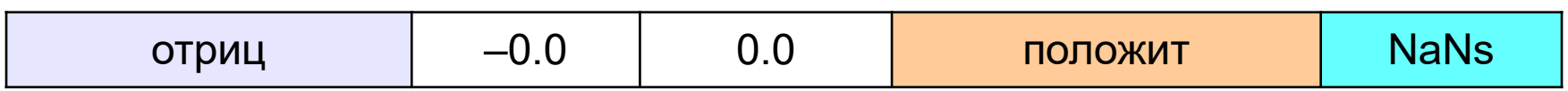
A == B => A и B: идентичное представление в памяти?

Нет! -0.0 и 0.0 различаются

A != B => A и B: различное представление в памяти?

Нет! NaN != NaN (Not-a-Number)

Javadoc: порядок вещественных чисел



Сортировка вещественных чисел

Алгоритм #1

1. Переместить все NaN’ы в конец массива

2. Отсортировать оставшиеся элементы

3. Переместить -0.0 перед 0.0, зная что

```
Float.floatToRawIntBits(0.0) == 0
Float.floatToRawIntBits(-0.0) < 0
```

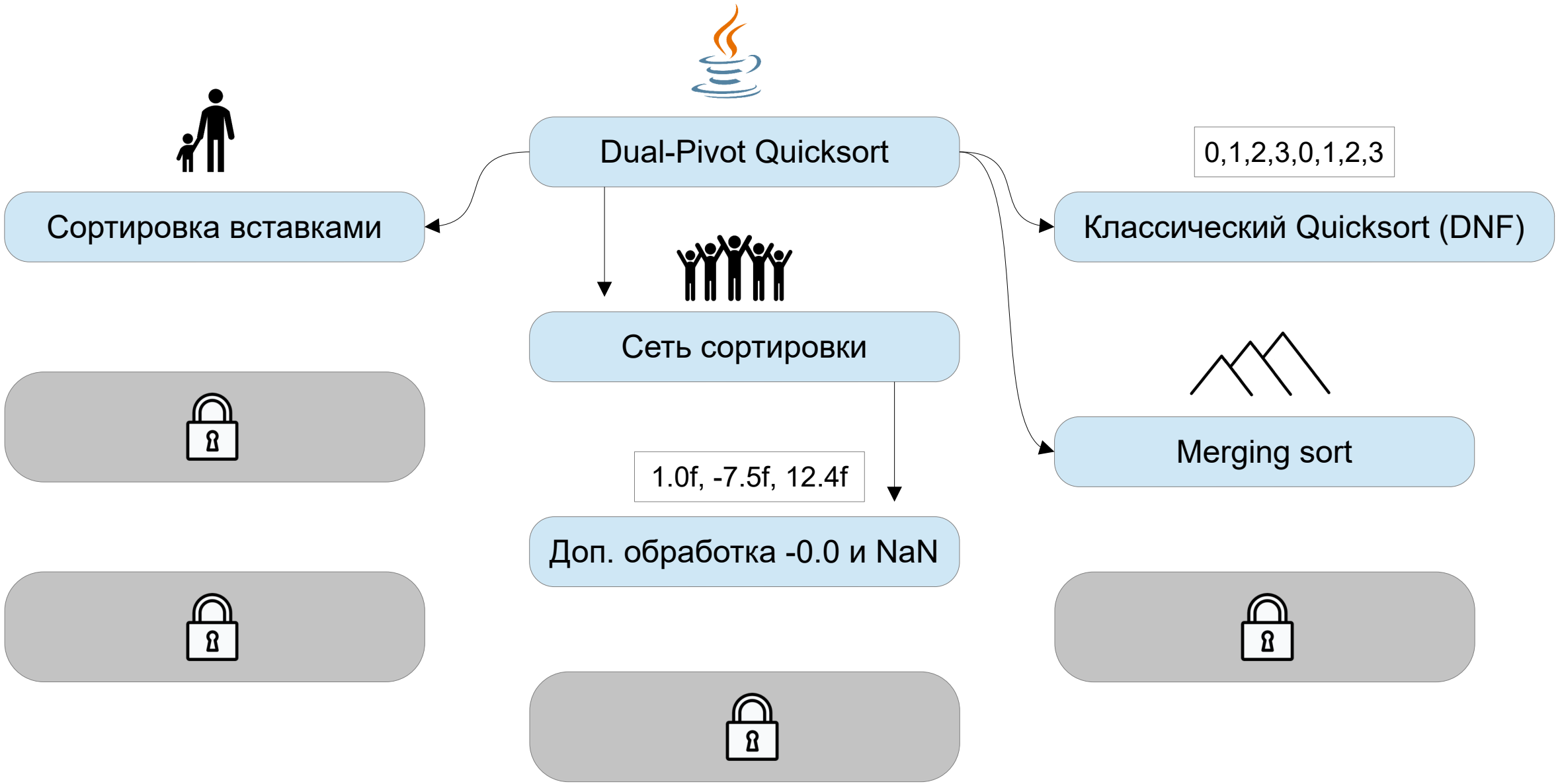
Сортировка вещественных чисел

Алгоритм #2

1. Переместить все NaN’ы в конец массива,
   посчитать количество -0.0 и заменить их на 0.0

2. Отсортировать оставшиеся элементы

3. Заменить нужное количество первых 0.0 на -0.0


Dual-Pivot Quicksort
Сортировка вставками
Сеть сортировки
Классический Quicksort (DNF)
Merging sort
Доп. обработка -0.0 и NaN
0,1,2,3,0,1,2,3
1.0f, -7.5f, 12.4f

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

Сортировка вещественных чисел

Параллельная сортировка

Лучшая сортировка

Параллельная сортировка

Пакет `java.util.concurrent`

Классы

* `RecursiveTask`
* `RecursiveAction`
* `CountedCompleter`
* ...

Методы

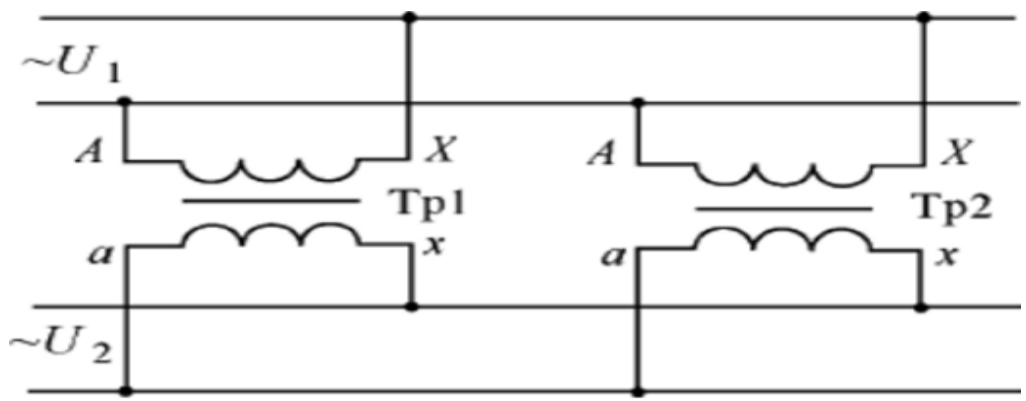
* `compute`() — реализация алгоритма
* `fork`(), `invoke`(), `invokeAll`(), `tryComplete`()

Параллельная сортировка

JDK 8: параллельная сортировка в несколько раз быстрее!

```
Arrays.parallelSort()
```

→ основан на Merge sort

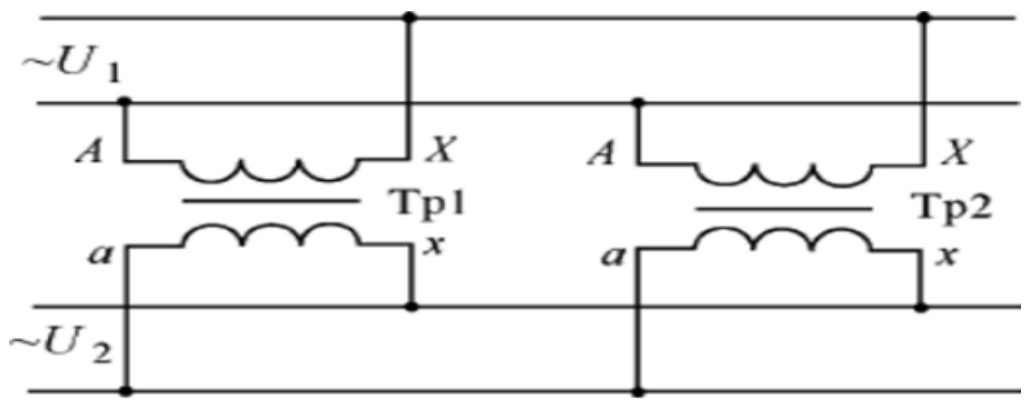


Параллельная сортировка

JDK 8: параллельная сортировка в несколько раз быстрее!

```
Arrays.parallelSort()
```

→ основан на Dual-Pivot Quicksort

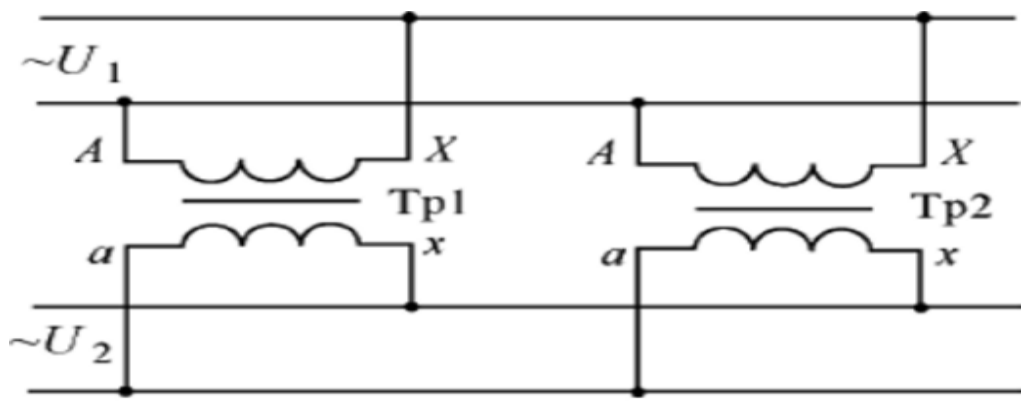


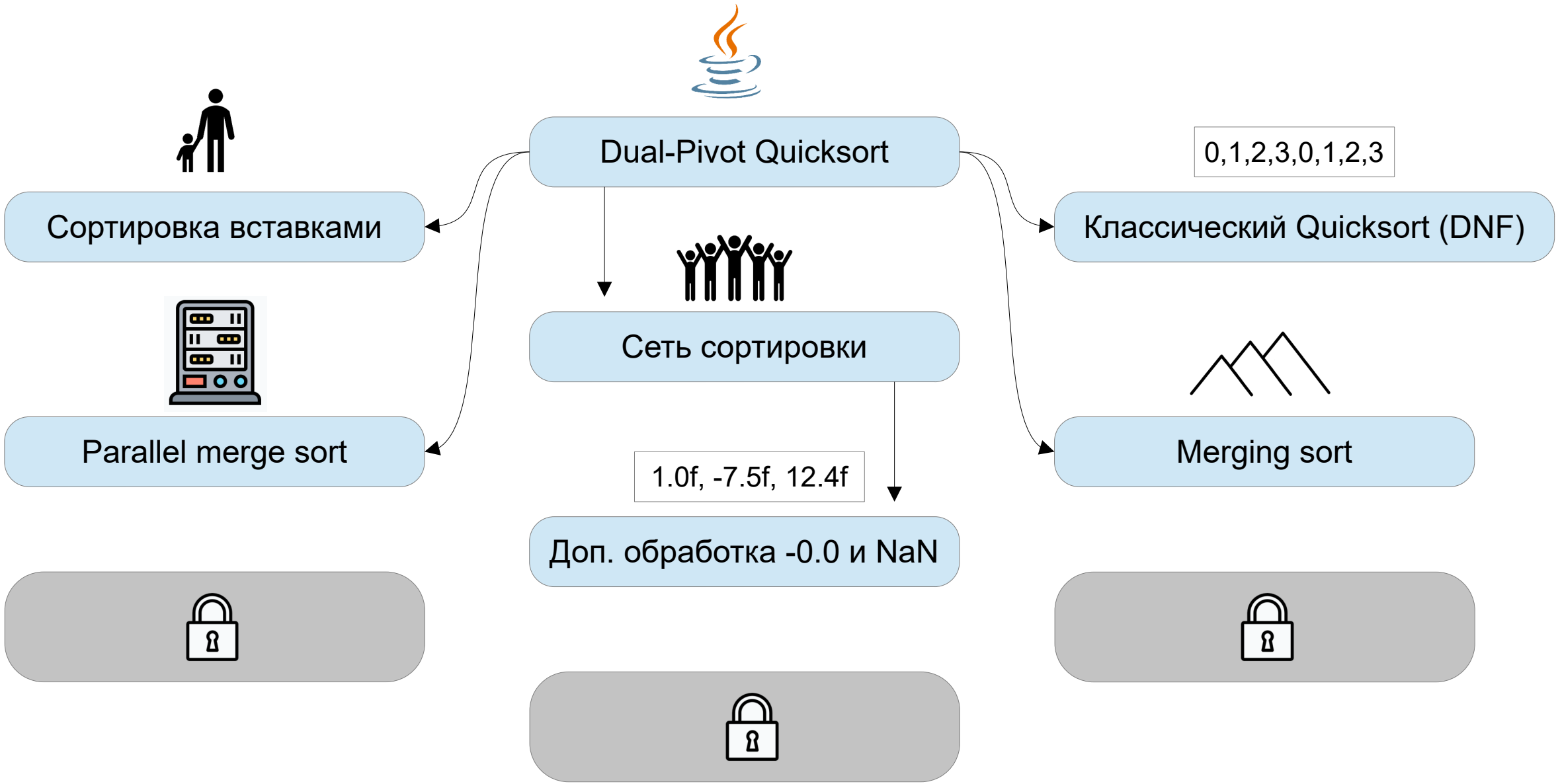
Параллельная сортировка

JDK **14**: параллельная сортировка в несколько раз быстрее!

`Arrays.parallelSort()`

⟶ основан на Merge sort + Dual-Pivot Quicksort




Dual-Pivot Quicksort
Сортировка вставками
Parallel merge sort
Сеть сортировки
Доп. обработка -0.0 и NaN
1.0f, -7.5f, 12.4f
0,1,2,3,0,1,2,3
Классический Quicksort (DNF)
Merging sort

Dual-Pivot Quicksort изнутри

Сортировка небольших массивов

Опорные элементы

Разбиение массива

Сортировка вещественных чисел

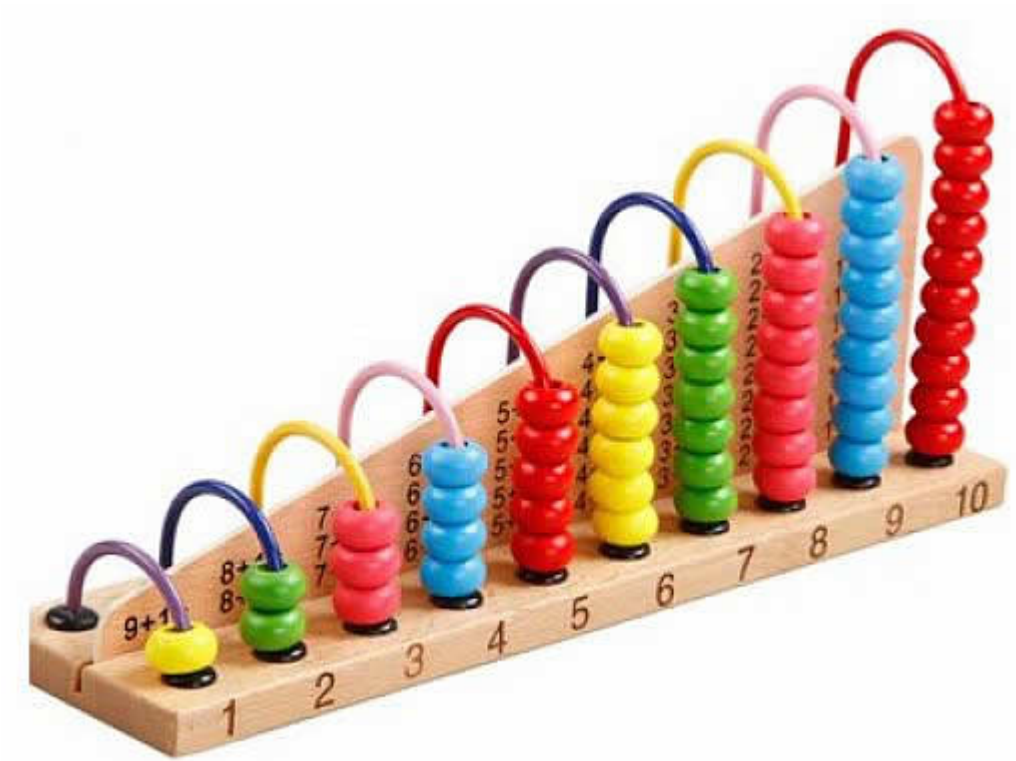
Параллельная сортировка

Лучшая сортировка

Сортировка чисел

Какая сортировка самая лучшая?
Сложность?

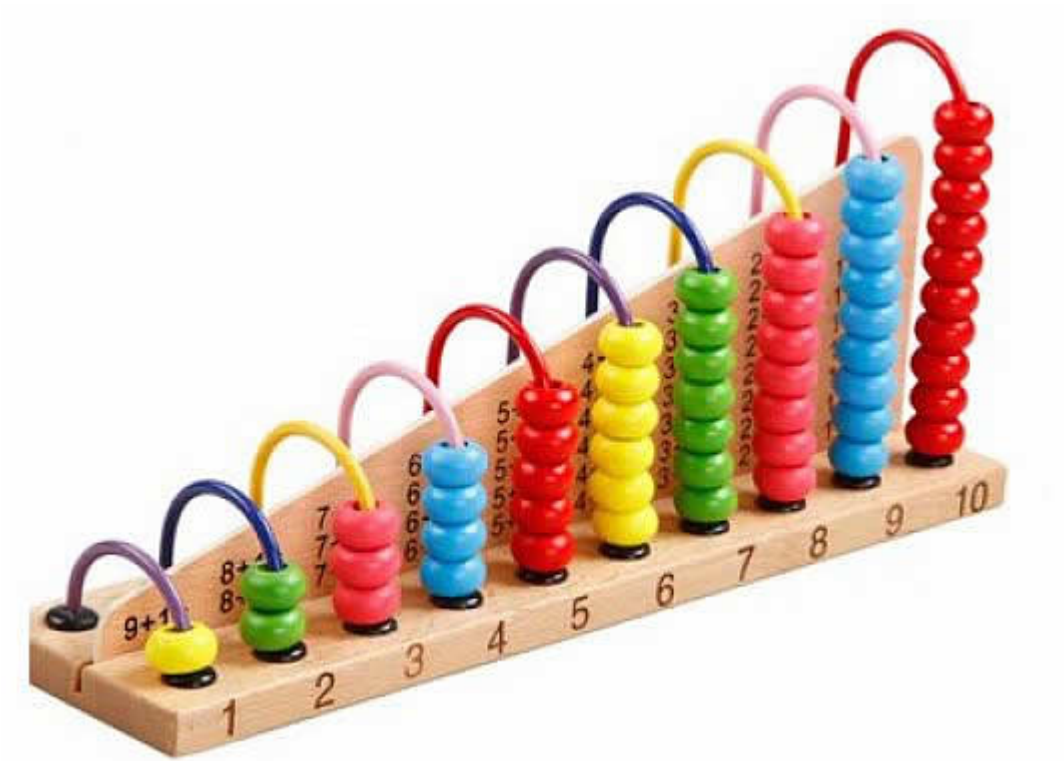
O(n)
O(n^2)
O(n*ln(n))



Сортировка чисел

Какая сортировка самая лучшая? Быстрая сортировка (Quicksort)?
Сложность?

O(n)
O(n^2)
O(n*ln(n))



Counting sort (сортировка подсчетом)

0 1 0 1 0 2 1 0 1 1 0 2 1 1 0 0 1 2 0 0

Counting sort (сортировка подсчетом)

0 1 0 1 0 2 1 0 1 1 0 2 1 1 0 0 1 2 0 0

Количество "0" — 9 шт.
Количество "1" — 8 шт.
Количество "2" — 3 шт.

Counting sort (сортировка подсчетом)

0 1 0 1 0 2 1 0 1 1 0 2 1 1 0 0 1 2 0 0

Количество "0" — 9 шт.
Количество "1" — 8 шт.
Количество "2" — 3 шт.

0 0 0 0 0 0 0 0 0

Counting sort (сортировка подсчетом)

0 1 0 1 0 2 1 0 1 1 0 2 1 1 0 0 1 2 0 0

Количество "0" — 9 шт.
Количество "1" — 8 шт.
Количество "2" — 3 шт.

0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

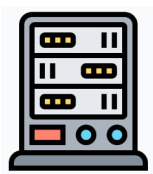
Counting sort (сортировка подсчетом)

0 1 0 1 0 2 1 0 1 1 0 2 1 1 0 0 1 2 0 0

Количество "0" — 9 шт.
Количество "1" — 8 шт.
Количество "2" — 3 шт.

0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2

Сортировка вставками



Parallel merge sort

8 / 16 / 16 bits
byte / char / short

Сортировка подсчетом

Dual-Pivot Quicksort

Сеть сортировки

1.0f, -7.5f, 12.4f

Доп. обработка -0.0 и NaN

0,1,2,3,0,1,2,3

Классический Quicksort (DNF)

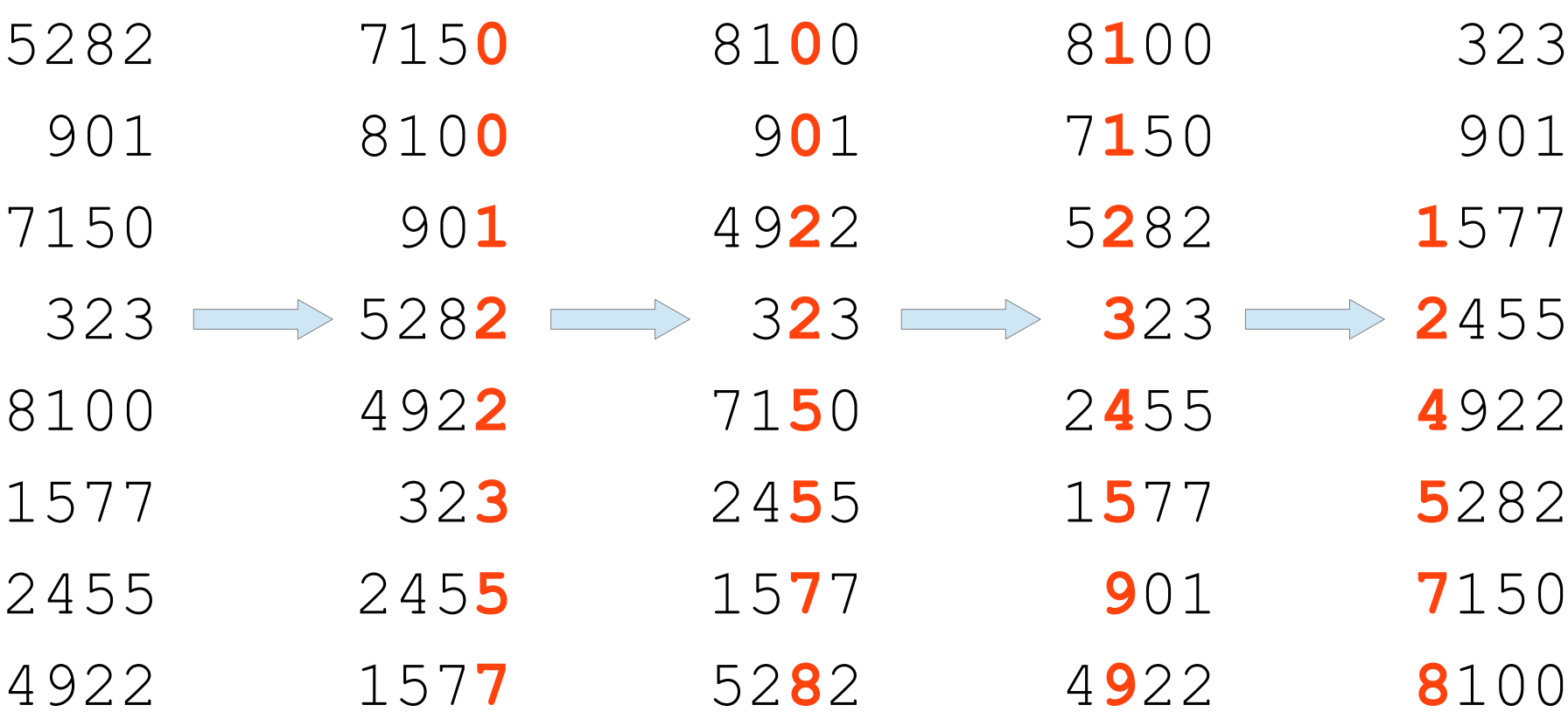
Merging sort

Radix sort (LSD, Least Significant Digit)*

```
5282
 901
7150
 323
8100
1577
2455
4922
```

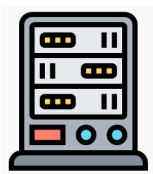
* in progress

Radix sort (поразрядная сортировка)*



* in progress

Сортировка вставками



Parallel merge sort

8 / 16 / 16 bits
byte / char / short

Сортировка подсчетом

Dual-Pivot Quicksort

Сеть сортировки

1.0f, -7.5f, 12.4f

Доп. обработка -0.0 и NaN

0,1,2,3,0,1,2,3

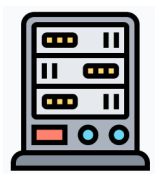
Классический Quicksort (DNF)

Merging sort

693993751058209
628608998628031

Radix sort

Сортировка вставками



Parallel merge sort

8 / 16 / 16 bits
byte / char / short

Сортировка подсчетом

Dual-Pivot Quicksort

Сеть сортировки

1.0f, -7.5f, 12.4f

Доп. обработка -0.0 и NaN

Heap sort

0,1,2,3,0,1,2,3

Классический Quicksort (DNF)

Merging sort

693993751058209
628608998628031

Radix sort

Dual-Pivot Quicksort vs. Timsort (JDK 7)

Timsort — для сортировки объектов

- стабильный
- требует дополнительную память
- O(n*ln(n)) во всех случаях

Dual-Pivot Quicksort — для сортировки простых типов

- сортирует "на месте"
- меняет порядок одинаковых элементов
- *O(n*ln(n))* в наилучшем случае
- *O(n*ln(n))* в среднем случае
- *O(n^2)* в наихудшем случае

Dual-Pivot Quicksort vs. Timsort (JDK **14**)

Timsort — для сортировки объектов

- стабильный
- требует дополнительную память
- O(n*ln(n)) во всех случаях

Dual-Pivot Quicksort — для сортировки простых типов

- сортирует "на месте" (+ использует дополнительную память)
- меняет порядок одинаковых элементов
- *O(n)* в наилучшем случае (с помощью Merging sort)
- *O(n*ln(n))* в среднем случае
- *O(n*ln(n))* в наихудшем случае (с помощью Heap sort)

Dual-Pivot Quicksort vs. Timsort (JDK **14**)

Timsort — для сортировки объектов

- стабильный ⟹ нельзя заменить нестабильным Quicksort'ом

sort

```
public static void sort(Object[] a)
```

Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. All elements in the array must implement the `Comparable` interface. Furthermore, all elements in the array must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw `ClassCastException` for any elements `e1` and `e2` in the array).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

Dual-Pivot Quicksort vs. Timsort

Timsort — для сортировки объектов

- throw IllegalArgumentExcception*("Comparison method violates its contract")*
  поэтому merge sort оставили в качестве запасного варианта

Какая сортировка самая лучшая?

Какой размер массива?

Можем ли использовать дополнительную память?

Требуется ли гарантированная сложность $O(n*ln(n))$?

Нужна ли устойчивость?

Проверяем ли правильность сравнения?

Знаем ли мы природу элементов?

Насколько данные уже упорядочены?

Класс java.util.DualPivotQuicksort

Тип данных	Алгоритм
Небольшие массивы	Комбинированная сортировка вставками
Массивы byte, char, short	Сортировка подсчетом (линейное время)
Повторяющиеся данные	Классический Quicksort (DNF)
Упорядоченные последовательности	Проверка за линейное время
Почти упорядоченные массивы	Merging sort
Вещественные числа	Дополнительная обработка -0.0 и NaN
Опорные элементы	Сеть сортировки (sorting network)
"Killer" массив	Пирамидальная сортировка (Heap sort)
Многопроцессорный сервер	Параллельная сортировка слиянием (merge sort)
Большие массивы случайных чисел	Поразрядная сортировка (Radix sort)* (лин. время)
Остальные массивы	Dual-Pivot Quicksort

* in progress

Содержание

Ошибка в Timsort

В 2015 году система верификации KeY
нашла алгоритмическую ошибку !!!

После "заплатки" не воспроизвести на JVM

Правильное исправление сделали только в 2018

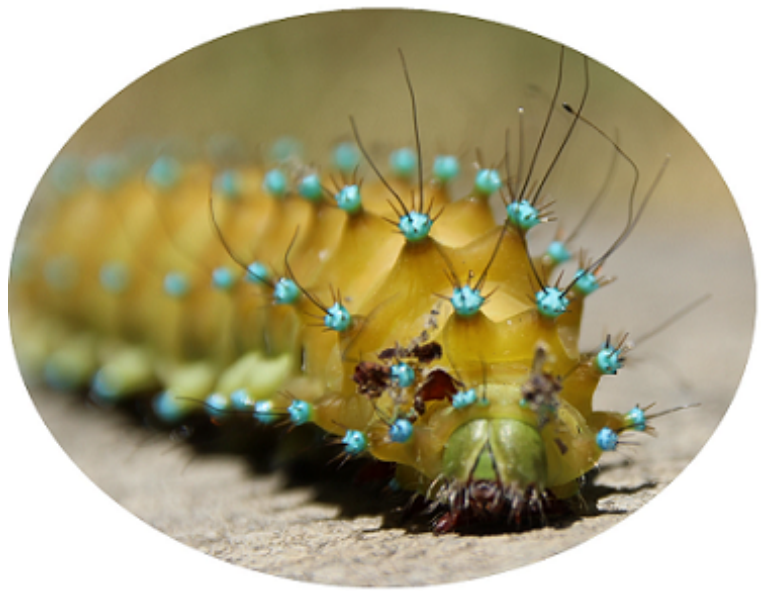
http://habr.com/post/251751

Ошибка в параллельной быстрой сортировке

JDK 8 — JDK 13: параллельная сортировка вещественных чисел `Arrays.parallelSort()` содержала ошибку !!!

Значения -0.0 и NaN сортировались неправильно:

…, 0.0, -0.0, 0.0, ...



Тестирование

Достаточно ли такой проверки a[0] <= a[1] <= … <= a[n – 1] <= a[n] ?

Тестирование

Достаточно ли такой проверки a[0] <= a[1] <= … <= a[n – 1] <= a[n] ?

Напишем такую *"сортировку"*

```
void sort(int[] a) {
    for (int i = 0; i < a.length; ++i) {
        a[i] = 0;
    }
}
```

Тестирование

Достаточно ли такой проверки a[0] <= a[1] <= … <= a[n – 1] <= a[n] ?

Напишем такую *"сортировку"*

```
void sort(int[] a) {
    for (int i = 0; i < a.length; ++i) {
        a[i] = 0;
    }
}
```

Она пройдет наш тест!

Проверка с помощью другой сортировки

1. Подготовить два одинаковых неотсортированных массива

[4, 8, 2, ... , 0]
[4, 8, 2, ... , 0]

2. Отсортировать их разными алгоритмами

[0, 2, 4, ... , 8]
[0, 2, 4, ... , 8]

3. Сравнить результаты

Ok!

Проверка с помощью перемешивания

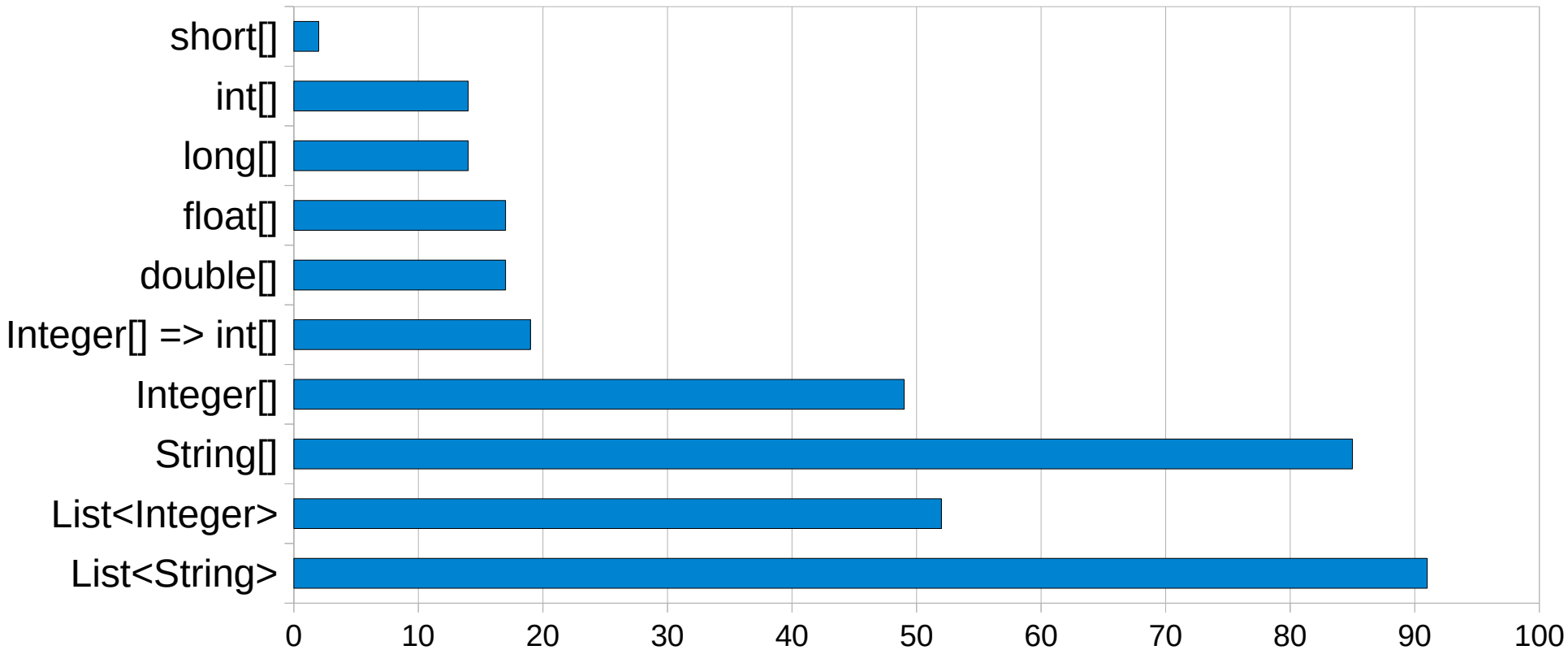
1. Подготовить какой-нибудь упорядоченный массив — [0, 3, 5, 11, … , n]
2. Перемешать элементы — [5, n, 11, 3, … , 0]
3. Отсортировать и сравнить с исходным массивом — Ok!

Проверка с помощью контрольной суммы

1. Подготовить какой-нибудь неотсортир. массив	[3, 2, n, 0, … , 1]
2. Вычислить контр. сумму	a[0] ^ a[1] ^ … ^ a[n - 1]
3. Отсортировать массив	[0, 1, 2, 3, ... , n]
4. Вычислить и сравнить контр. суммы	Ok!

Содержание

Сортировка данных различных типов



Сортировка 60 000 случайных чисел

Статьи про Dual-Pivot Quicksort

Holistic Analysis of Yaroslavskiy's Partitioning Scheme

Markus E. Nebel and Conrado Martinez

Dept. of Computer Science, Univ. Politecnica de Catalunya

Dual-Pivot Quicksort

Vasileios Iliopoulos and David B. Penman

Department of Mathematical Sciences, University of Essex

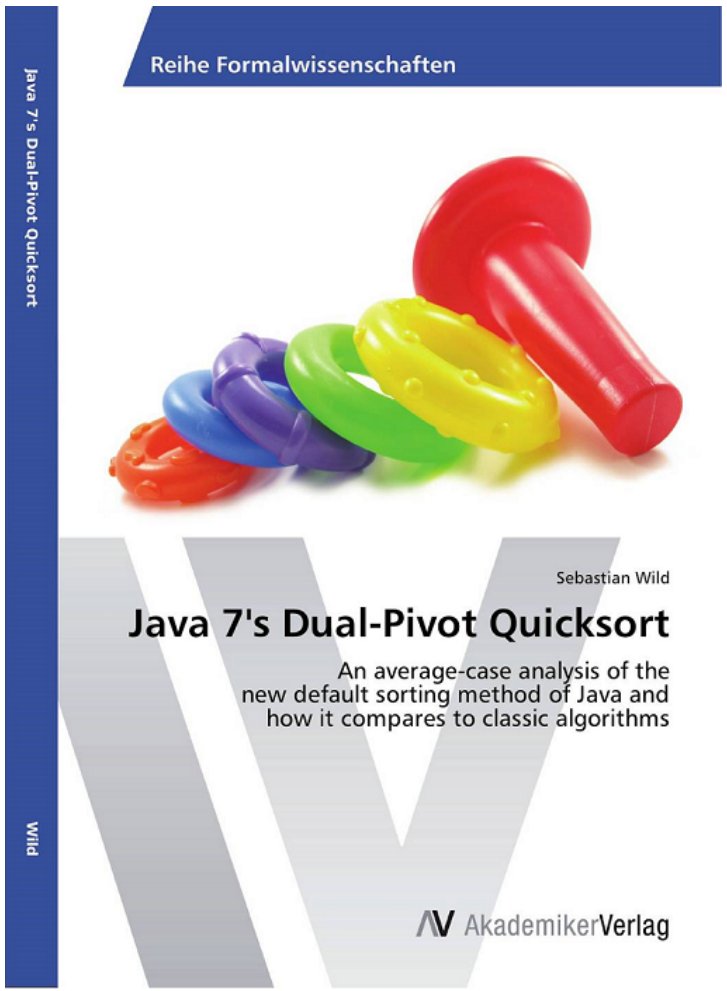
Average Case Analysis of Java 7's Dual Pivot Quicksort

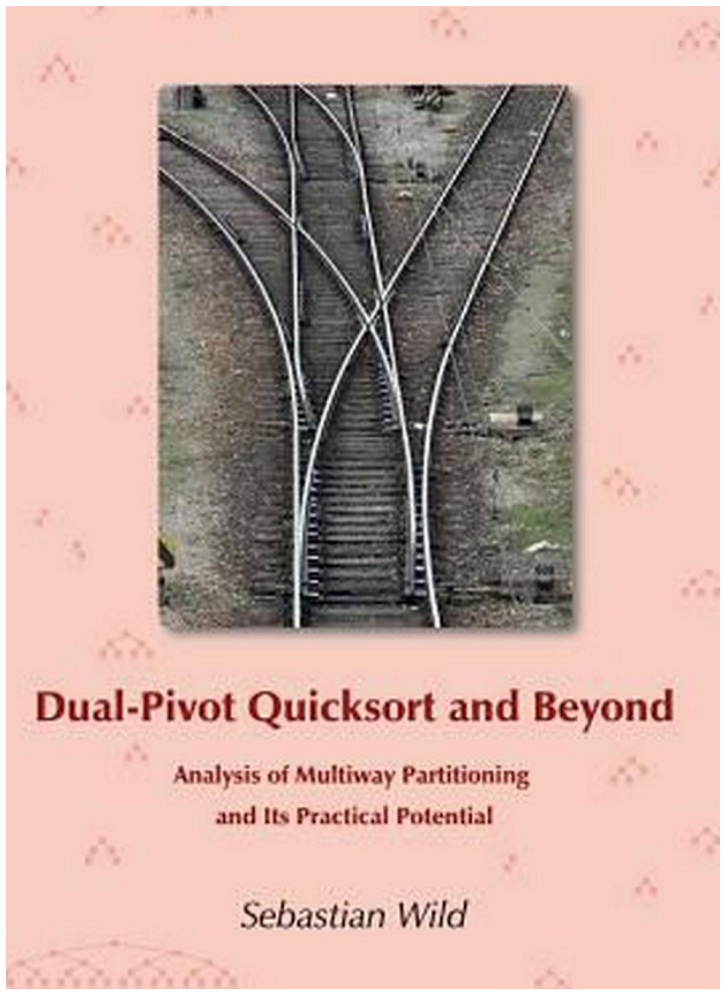
Sebastian Wild and Markus E. Nebel

Computer Science Department, University of Kaiserslautern

. . .

Книги про Dual-Pivot Quicksort





Проект OpenJDK

Pull requests

Open JDK-8266431: Dual-Pivot Quicksort improvements (Radix sort)
https://github.com/openjdk/jdk/pull/13568

Closed JDK-8309130: x86_64 AVX512 intrinsics for Arrays.sort()
https://github.com/openjdk/jdk/pull/14227

Open JDK-8266431: Follow-up to AVX512 intrinsics for Arrays.sort()
https://github.com/openjdk/jdk/pull/16124

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`
* *настоящий* программист: ??

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`
* *настоящий* программист: придумает свою сортировку и
  поедет на конференцию ??

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`
* *настоящий* программист:

дано: a = { 3, n - 1, 2, n, 4, 0, …, 1 }

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`
* *настоящий* программист:

результат: a = { 0, 1, 2, 3, 4, …, n - 1, n }

Математик vs. программист

Задача

Есть *перестановка* чисел от *0* до *n*, нужно отсортировать эти числа

* математик: формулы, теоремы, статьи, конференции
* начинающий программист: напишет сортировку пузырьком
* более опытный (но ленивый) программист: `Arrays.sort(a)`
* старший (и начитанный) программист: `Arrays.parallelSort(a)`
* *настоящий* программист: a[i] = i

a = { 0, 1, 2, 3, 4, …, n - 1, n }

Выводы

Пробуйте, пробуйте и не бойтесь

Сортировка все время улучшалась

История продолжается и сейчас

Интеграция в OpenJDK — это не больно

Java != математика

Смотрите в корень задачи

Полезные ссылки

Группа Core Libraries

https://openjdk.org/groups/core-libs

Dual-Pivot Quicksort

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/DualPivotQuicksort.java или <jdk>/lib/src.zip

Почта

vlv.spb.ru@mail.ru

Какой сегодня день? День программиста!

256-ой день в году — день программиста в России (12 или 13 сентября)

13-ое, пятница — настоящий день программиста (1, 2 или 3 раза в год)

2024: 12 и 13 сентября (пятница) — два праздника подряд

2030: 13 сентября (пятница) — два праздника в один день

Вопросы