

---

АДРИАНОВ Н.М.  
ИВАНОВ А.Б.

# АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

СОРТИРОВКИ

## QUICKSORT: АНАЛИЗ (ВТОРОЙ МЕТОД)

Partitioning:  $n + 1$  сравнение.

Пусть  $C_n$  – мат.ожидание количества сравнений при сортировке массива из  $n$  элементов.

$$C_n = n + 1 + \sum_{k=1}^n \frac{1}{n} (C_{k-1} + C_{n-k}), \quad C_0 = C_1 = 0$$

## QUICKSORT: АНАЛИЗ (ВТОРОЙ МЕТОД)

$$C_n = n + 1 + \sum_{k=1}^n \frac{1}{n} (C_{k-1} + C_{n-k})$$

$$C_n = n + 1 + \frac{2}{n} \sum_{k=1}^n C_{k-1}$$

$$nC_n = n(n + 1) + 2 \sum_{k=1}^n C_{k-1}$$

$$(n - 1)C_{n-1} = (n - 1)n + 2 \sum_{k=1}^{n-1} C_{k-1}$$

$$nC_n - (n - 1)C_{n-1} = 2n + 2C_{n-1}$$

$$nC_n = (n + 1)C_{n-1} + 2n$$

## QUICKSORT: АНАЛИЗ (ВТОРОЙ МЕТОД)

$$nC_n = (n + 1)C_{n-1} + 2n$$

$$\begin{aligned}\frac{C_n}{n+1} &= \frac{C_{n-1}}{n} + \frac{2}{n+1} = \\ &= \frac{C_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \\ &= \frac{C_1}{2} + \frac{2}{3} + \dots + \frac{2}{n} + \frac{2}{n+1}\end{aligned}$$

$$C_n \sim 2(n+1) \sum_{k=1}^{n+1} \frac{1}{k} - C \cdot n \sim 2n \ln n$$

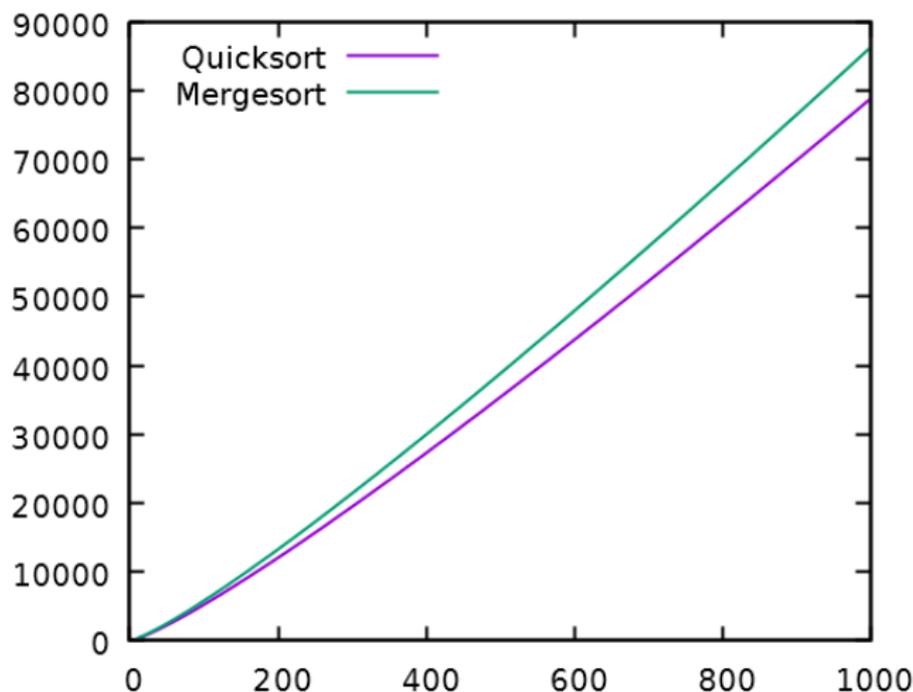
# БЫСТРАЯ СОРТИРОВКА: ЗАЧЕМ?

- Сложность
- Необходимая память
- Устойчивость
  
- $O(n \log n)$
- $\approx \log(n)$
- не является устойчивым

## БЫСТРАЯ СОРТИРОВКА: ЗАЧЕМ?

Сортировка слиянием:  $12.5 x \log(x)$

Быстрая сортировка:  $11.67 x \log(x) - 1.74 x$



# БЫСТРАЯ СОРТИРОВКА С ТРЕХЧАСТНЫМ РАЗБИЕНИЕМ (3-WAY QUICKSORT)

Какое время работы алгоритма на массиве, состоящем из одинаковых элементов?

# БЫСТРАЯ СОРТИРОВКА С ТРЕХЧАСТНЫМ РАЗБИЕНИЕМ (3-WAY QUICKSORT)

Какое время работы алгоритма на массиве, состоящем из одинаковых элементов?

Стандартный quicksort:  $\Theta(n^2)$ .

Решение: 3-way quicksort.

Разбиваем массив на 3 части:

- $a[i] < p$  для  $0 \leq i \leq r$
- $a[i] = p$  для  $r + 1 \leq i \leq s$
- $a[i] > p$  для  $s + 1 \leq i \leq n - 1$

## 3-WAY QUICKSORT



Задача о голландском национальном флаге  
(Edsger Wybe Dijkstra)



Отсортировать массив, состоящий из 0, 1 и 2

## 3-WAY QUICKSORT

Инвариант цикла:

$a[j] < p$   
при  $j \in [low + 1, lt)$

$a[j] = p$   
при  $j \in [lt, i)$

$a[j] > p$   
при  $j \in (gt, high)$

---

```
procedure QuickSort3Way(a[]):  
  // a[] - массив от 0 до n-1  
  перемешать (shuffle) массив a  
  QuickSort3Way(a, 0, n-1)  
end
```

```
procedure QuickSort3Way(a[], low, high):  
  p = a[low]  
  i = low + 1  
  lt = low + 1  
  gt = high  
  while i <= gt:  
    if a[i] < p:  
      Exch(a, lt++, i++)  
    else if a[i] > p:  
      Exch(a, i, gt--)  
    else  
      i++  
  end  
  Exch(a, low, --lt)  
  
  QuickSort3Way(a, low, lt - 1)  
  QuickSort3Way(a, gt + 1, high)  
end
```

---

## QUICKSORT: ПЕРЕПОЛНЕНИЕ СТЕКА

В самом худшем случае будет  $n$  рекурсивных вызовов, что может привести к переполнению стека.

## QUICKSORT: ПЕРЕПОЛНЕНИЕ СТЕКА

В самом худшем случае будет  $n$  рекурсивных вызовов, что может привести к переполнению стека.

Решение: вместо двух рекурсивных вызовов делать только один (для части меньшего размера). Вторую (большую) часть обрабатывать, не используя рекурсивный вызов.

## ПРАКТИЧЕСКИЕ УЛУЧШЕНИЯ

На массивах небольшого размера выгоднее использовать сортировку вставками.

Выбор опорного элемента: медиана из трех (median-of-3).  
Выбираем 3 случайных элемента и в качестве опорного берем второй по порядку (медиану).

Tukey's ninther: выбираем 3 тройки случайных элементов, в каждой тройке находим медиану и в качестве опорного берем второй медиану медиан.

# МЕТОДЫ СОРТИРОВОК: ХАРАКТЕРИСТИКИ

Алгоритм	stable	in-place	Время	Доп. память
Selection	-	+	$N^2$	1
Insertion	+	+	между $N$ и $N^2$	1
Shell	-	+	$N \log N$ ? $N^{6/5}$ ?	1
Quicksort	-	+	$N \log N$	$\log N$
3-way quicksort	-	+	между $N$ и $N \log N$	$\log N$
Mergesort	+	-	$N \log N$	$N$
Timsort	+	-	между $N$ и $N \log N$	между 1 и $N$
Heapsort	-	+	$N \log N$	1
???	+	+	$N \log N$	1

# QUICKSELECT

Дано: массив размера  $n$ ,  $1 \leq k \leq n$

Найти:  $k$ -ю порядковую статистику

## QUICKSELECT

Дано: массив размера  $n$ ,  $1 \leq k \leq n$

Найти:  $k$ -ю порядковую статистику

Пусть  $C(n, k)$  – матожидание количества сравнений

$$C(n, \cdot) \leq n - 1 + \frac{1}{n} \sum_{i=1}^n C(\max\{i - 1, n - i\}, \cdot)$$

$$C(n, \cdot) \leq n - 1 + \frac{2}{n} \sum_{i=1}^{n/2} C(n - i, \cdot)$$

Индукция:  $C(m, \cdot) \leq cm$  для  $m < n \Rightarrow$

$$C(n, \cdot) \leq \left(1 + \frac{3}{4}c\right)n \leq cn$$

# ДЕТЕРМИНИРОВАННЫЙ SELECT<sup>1</sup>

Blum

Floyd: Алгоритм Флойда

Pratt: Алгоритм Кнута — Морриса — Пратта

Rivest: RSA (Rivest, Shamir, Adleman)

Tarjan: Алгоритм Тарьяна

Select(массив размера  $n$ ,  $k$ )

- разбиваем на пятерки ( $m = \lceil n/5 \rceil$ ), сортируем каждую, выбираем медиану в каждой пятерке
- находим медиану  $p$ : Select(массив размера  $m$ ,  $m/2$ )
- выполняем Partition с опорным элементом  $p$
- определяем в какой из частей находится  $k$ -я статистика, вызываем Select рекурсивно

---

<sup>1</sup>Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. (August 1973). "Time bounds for selection". Journal of Computer and System Sciences. 7 (4): 448–461

## ДЕТЕРМИНИРОВАННЫЙ SELECT (МЕДИАНА МЕДИАН)

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

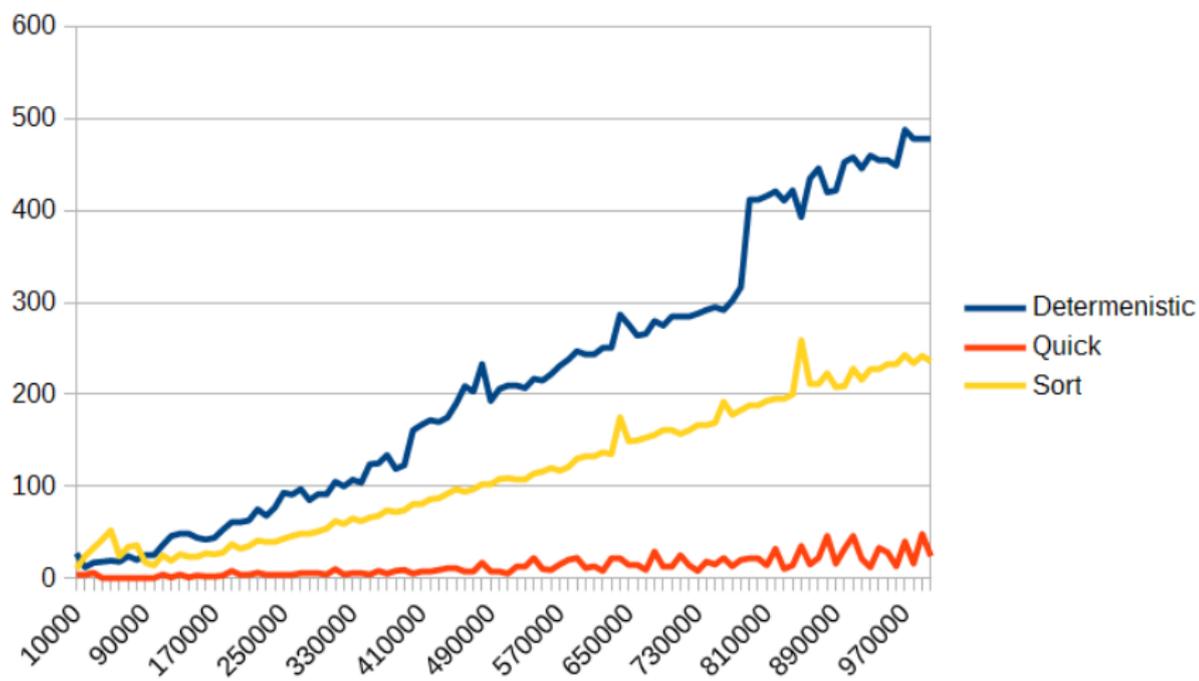
или

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + an$$

угадаем и докажем по индукции:

$$T(n) \leq Cn, \text{ где } C = 10a$$

# СРАВНЕНИЕ РЕАЛИЗАЦИЙ SELECT



# СОРТИРОВКА ПОДСЧЕТОМ

Задача: отсортировать массив из  $N$  чисел от 0 до  $R - 1$ .

Наивная сортировка подсчетом:

Посчитать количество 0, количество 1, количество 2...

Расставить в массиве нужное количество 0, нужное количество 1...

Не годится. На практике надо уметь сортировать объекты по некоторому ключу (например: людей по росту). Уточненная формулировка:

Задача:  $U$  – пространство объектов,  $f : U \rightarrow [0..R - 1]$ .

Отсортировать массив из  $N$  объектов по возрастанию  $f(x)$ .

# СОРТИРОВКА ПОДСЧЕТОМ

Задача:  $U$  – пространство объектов,  $f : U \rightarrow [0..R - 1]$ .

Отсортировать массив из  $N$  объектов по возрастанию  $f(x)$ .

- Посчитаем количество объектов для каждого значения ключа:  $c_i = \{x : f(x) = i\}$ .
- На каких позициях в отсортированном должны стоять элементы с ключом 0? С ключом 1? С ключом  $i$ ?
- Посчитаем нарастающим итогом (running total)...  
 $0, c_0, c_0 + c_1, c_0 + c_1 + c_2, \dots$
- Пройдем по массиву: для каждого элемента знаем позицию, куда его надо поставить.

## СОРТИРОВКА ПОДСЧЕТОМ

Задача:  $U$  – пространство объектов,  $f : U \rightarrow [0..R - 1]$ .

Отсортировать массив из  $N$  объектов по возрастанию  $f(x)$ .

`count[i]` – счетчик объектов/на какой позиции нужно поставить очередной объект. `aux` – вспомогательный массив

---

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

---

## СОРТИРОВКА ПОДСЧЕТОМ

- Сложность  $O(N + R)$
- Дополнительная память  $O(N + R)$
- Устойчивая

Дальше: поразрядные сортировки (radix sorts).

## LSD-СОРТИРОВКА (LEAST SIGNIFICANT DIGIT)

Задача: отсортировать массив из  $N$  строк одинаковой длины.

Применим сортировку подсчетом:  $U =$  строки длины  $W$ ,  
 $f_d(x) = d$ -й символ в строке  $x$ .

Сортируем сначала по последнему символу (используя  $f_W$ ), потом по предпоследнему (используя  $f_{W-1}$ ) и т.д. После сортировки по первому символу массив строк окажется отсортированным в лексикографическом порядке (следствие устойчивости сортировки подсчетом!).

# LSD-СОРТИРОВКА (LEAST SIGNIFICANT DIGIT)

---

```
public static void sort(String[] a, int W)
{
    int R = 256;
    int N = a.length;
    String[] aux = new String[N];

    for (int d = W-1; d >= 0; d--)
    {
        int[] count = new int[R+1];
        for (int i = 0; i < N; i++)
            count[a[i].charAt(d) + 1]++;
        for (int r = 0; r < R; r++)
            count[r+1] += count[r];
        for (int i = 0; i < N; i++)
            aux[count[a[i].charAt(d)]++] = a[i];
        for (int i = 0; i < N; i++)
            a[i] = aux[i];
    }
}
```

---

## MSD-СОРТИРОВКА (MOST SIGNIFICANT DIGIT)

Задача: отсортировать массив из  $N$  строк.

Сортируем сначала по первому символу (используя  $f_1$ ), затем диапазон для каждой буквы на первом месте сортируем рекурсивно по второй букве и т.д.

## MSD-СОПТИРОВКА (MOST SIGNIFICANT DIGIT)

---

```
public static void sort(String[] a)
{
    aux = new String[a.length];
    sort(a, aux, 0, a.length, 0);
}
private static void sort(String[] a, String[] aux, int lo, int hi, int
    d)
{
    if (hi <= lo) return;
    int[] count = new int[R+2];
    for (int i = lo; i <= hi; i++)
        count[charAt(a[i], d) + 2]++;
    for (int r = 0; r < R+1; r++)
        count[r+1] += count[r];
    for (int i = lo; i <= hi; i++)
        aux[count[charAt(a[i], d) + 1]++] = a[i];
    for (int i = lo; i <= hi; i++)
        a[i] = aux[i - lo];
    for (int r = 0; r < R; r++)
        sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);
}
```

---

## LSD - СЛОЖНОСТЬ

LSD применяем к массиву строк одинаковой длины  $W$ :

- сложность  $O((N + R)W)$
- дополнительная память  $O(N + R)$

Если обозначить  $L = NW$  – общее количество символов, то сложность можно записать как  $O(L + RW) = O(L(1 + R/N))$ .

LSD можно применять и к строкам разной длины (нужен дополнительный «символ» для пустого места).

Не очень хорошо ведет себя, если есть (небольшое количество) относительно длинных строк (относительно остальных).

## MSD - СЛОЖНОСТЬ

MSD: можно применять к строкам разной длины  $W_i, i = 1, \dots, N$ .

- В худшем случае сложность  $O(L(1 + R))$ , где  $L = \sum_{i=1}^N W_i$
- дополнительная память  $O(N + R)$

На массиве случайных строк время работы алгоритмы будет даже сублинейным!!

# LSD/MSD = ПОРАЗРЯДНЫЕ СОРТИРОВКИ

Возможно применение для строк в различных алфавитах:

- Десятичные цифры (0123456789)
- Шестнадцатеричные цифры (0123456789ABCDEF)
- ДНК-последовательности (ACGT)
- Английские буквы (a..z,  $R = 26$ )
- ASCII ( $R = 128/256$ )
- Unicode ( $R = 65536$ )?

Вопрос: какой алгоритм выбрать для сортировки миллиона 32-битных целых чисел?

[https://www.youtube.com/watch?v=k4RRi\\_ntQc8](https://www.youtube.com/watch?v=k4RRi_ntQc8)

Barack Obama gets asked a computer science question by Google CEO Eric Schmidt.

## LORE

Вопрос: какой алгоритм выбрать для сортировки миллиона 32-битных целых чисел?

[https://www.youtube.com/watch?v=k4RRi\\_ntQc8](https://www.youtube.com/watch?v=k4RRi_ntQc8)

Barack Obama gets asked a computer science question by Google CEO Eric Schmidt.