

# Сортировка за линейное время

## Дискретный анализ 2017/18

2 сентября 2017 г.

# Литература

- ▶ Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-е издание, М.:Вильямс, 2005, стр. 220-239, глава 8, «Сортировка за линейное время»
- ▶ Седжвик Р. Алгоритмы на C++, М.:Вильямс, 2014, стр. 374-405, глава 10, «Поразрядная сортировка»
- ▶ Кнут Д. Искусство программирования, том 3. Сортировка и поиск, 2-е издание, М.:Вильямс, 2014, стр. 372-378, глава 5.4.7, «Внешняя поразрядная сортировка»

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

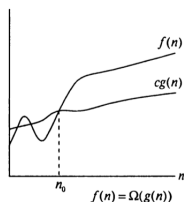
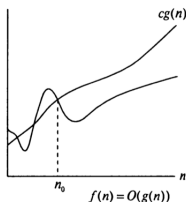
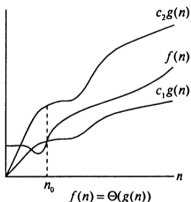
Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Асимптотические обозначения

- ▶  $f(n) = O(g(n))$ , если существуют положительные константы  $c$  и  $n_0$  такие что  $0 \leq f(n) \leq cg(n)$  для всех  $n \geq n_0$  (асимптотическая верхняя граница)
- ▶  $f(n) = \Omega(g(n))$ , если существуют положительные константы  $c$  и  $n_0$ , такие что  $0 \leq cg(n) \leq f(n)$  для всех  $n \geq n_0$  (асимптотическая нижняя граница)
- ▶  $f(n) = \Theta(g(n))$ , если существуют положительные константы  $c_1, c_2$  и  $n_0$  такие что  $0 \leq c_1g(n) \leq f(n) \leq c_2g(n)$  для всех  $n \geq n_0$  (асимптотически точная оценка)



# Линейность

- ▶  $f(n) = O(n)$
- ▶  $n$  — размерность задачи
- ▶ Линейность (функция  $f(n)$ ):
  - ▶ количество элементарных операций;
  - ▶ объём оперативной памяти;
  - ▶ могут оцениваться другие параметры: количество процессорных ядер или серверов.

# Линейность

## ▶ Скользкие места

- ▶ В строгом смысле элементарные операции — выполняемые машиной Тьюринга (или эквивалентной моделью) команды
- ▶ В практическом смысле элементарность операций определяется реальным процессором
- ▶ Не может быть алгоритма, работающего за  $O(n)$  и не укладывающегося в  $O(n)$  по памяти

## ▶ На практике

- ▶ Всегда есть физические ограничения: размер оперативной памяти, жёстких дисков
- ▶ Иерархия оперативной памяти: регистры, кэш-память L1, L2, DRAM, жёсткие диски, распределённые файловые системы
- ▶ Константа  $c$  может быть слишком большой

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка



# Сортировка сравнением

- ▶ Последовательность  $\langle a_1, a_2, \dots, a_n \rangle$
- ▶ При сортировке используются только попарные сравнения:  
 $a_i < a_j$ ,  $a_i \leq a_j$ ,  $a_i = a_j$ ,  $a_i \geq a_j$ ,  $a_i > a_j$
- ▶ Предполагаем, что все элементы различны, тем самым можно считать что используется только  $a_i \leq a_j$

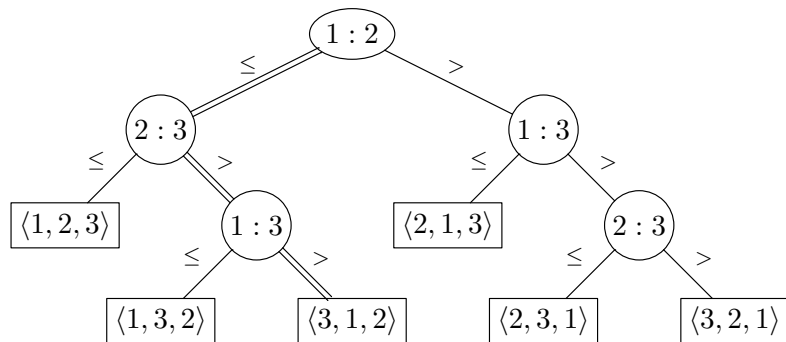
# Модель дерева решений

Дерево решений:

- ▶ полное бинарное дерево, в котором представлены операции сравнения элементов;
- ▶ внутренние узлы помечены меткой  $i : j$ ,  $1 \leq i, j \leq n$ , указывающие на сравнение  $a_i$  и  $a_j$ ;
- ▶ лист помечен перестановкой  $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$ , дающей окончательное упорядочение элементов  $\langle a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)} \rangle$ ;
- ▶ корректность сортировки сравнением: соответствующее дерево решений должно содержать все  $n!$  перестановок исходных  $n$  элементов, к которым можно проложить путь реального выполнения сортировки («достижимые» листья).

## Сортировка вставкой трёх элементов

$$\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$$



# Нижняя оценка для наихудшего случая

## Теорема

*В наихудшем случае в ходе выполнения любого алгоритма сортировки сравнением выполняется  $\Omega(n \log_2 n)$  сравнений.*

## Доказательство.

Для дерева решений сортировки  $n$  элементов и высотой  $h$  с  $l$  достижимыми листьями:

$$n! \leq l \leq 2^h \Rightarrow h \geq \log_2(n!) = \Omega(n \log_2 n)$$

Последнее равенство можно получить из формулы Стирлинга или разложением логарифма произведения. □

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Общая идея

- ▶  $\langle a_1, \dots, a_n \rangle$ ,  $a_i \in \mathbb{N} \cup \{0\}$  и  $a_i \leq k$
- ▶ Если  $k = O(n)$ , то время работы равно  $\Theta(n)$
- ▶ Для каждого  $a_i$  определяется  $c_i = |\{a_k | a_k < a_i\}|$
- ▶  $c_i$  определяет местоположение  $a_i$  в отсортированной последовательности

# Алгоритм

COUNTING-SORT( $A$ )

```
1  for  $i \leftarrow 0$  to  $k$ 
2       $C[i] \leftarrow 0$ 
3  for  $j \leftarrow 1$  to  $length[A]$ 
4       $C[A[j]] \leftarrow C[A[j]] + 1$ 
5  // В  $C[i]$  хранится количество элементов, равных  $i$ 
6  for  $i \leftarrow 1$  to  $k$ 
7       $C[i] \leftarrow C[i] + C[i - 1]$ 
8  // В  $C[i]$  — количество элементов, не превышающих  $i$ 
9  for  $j \leftarrow length[A]$  downto 1
10      $B[C[A[j]]] \leftarrow A[j]$ 
11      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

# Пример

$$A \quad \begin{bmatrix} 2 & 5 & 3 & 0 & 2 & 3 & 0 & 3 \end{bmatrix}, 0 \leq a_i \leq 5$$

$$C \quad \begin{bmatrix} 2 & 0 & 2 & 3 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 2 & 4 & 7 & 7 & 8 \end{bmatrix}$$

*B*

						3	
	0					3	
	0				3	3	
	0		2		3	3	
0	0		2		3	3	
0	0		2	3	3	3	
0	0		2	3	3	3	5
0	0	2	2	3	3	3	5

*C*

2	2	4	6	7	8
1	2	4	6	7	8
1	2	4	5	7	8
1	2	3	5	7	8
0	2	3	4	7	8
0	2	3	3	7	8
0	2	3	3	7	7
0	2	2	3	7	7



# Свойства сортировки подсчётом

- ▶ Не является сортировкой сравнением: ни одна пара элементов не сравнивается друг с другом
- ▶ Линейная (вернее,  $\Theta(k + n)$ , но при  $k = O(n)$  время выполнения  $\Theta(n)$ )
- ▶ Устойчивая (стабильная)
- ▶ Требуется дополнительная память под массивы  $C$  и  $B$  размером  $k$  и  $n$  соответственно

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

**Поразрядная сортировка**

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Общая идея

- ▶  $d$ -значные числа последовательно сортируются по разрядам: от младшего к старшему
- ▶ При использовании устойчивой сортировки — корректное упорядочивание
- ▶ Если внутренняя сортировка линейная, то и поразрядная сортировка тоже линейная

# Пример

329

457

657

839

436

720

355

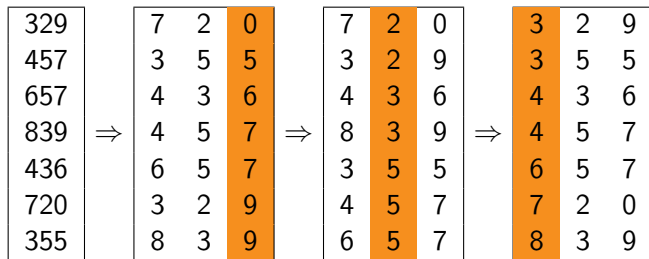
# Пример

329	⇒	7	2	0
457		3	5	5
657		4	3	6
839		4	5	7
436		6	5	7
720		3	2	9
355		8	3	9

# Пример

329	⇒	7	2	0	⇒	7	2	0
457		3	5	5		3	2	9
657		4	3	6		4	3	6
839		4	5	7		8	3	9
436		6	5	7		3	5	5
720		3	2	9		4	5	7
355		8	3	9		6	5	7

# Пример



# Алгоритм

$\text{RADIX-SORT}(A, d)$

1   **for**  $i \leftarrow 1$  **to**  $d$

2       Устойчивая сортировка массива  $A$  по  $i$ -ой цифре

## Теорема

*Для  $n$   $d$ -значных чисел, в которых каждая цифра принимает одно из  $k$  значений, алгоритм RADIX-SORT позволяет выполнить корректную сортировку за время  $\Theta(d(n + k))$ , если внутренняя устойчивая сортировка имеет время работы  $\Theta(n + k)$ .*



# Какие внутренние параметры выбрать?

## Теорема

Для  $n$   $b$ -битовых чисел и натурального числа  $r \leq b$  (цифры из  $r$  битов) алгоритм RADIX-SORT выполнит сортировку за время  $\Theta\left(\frac{b}{r}(n + 2^r)\right)$ .

# Какие внутренние параметры выбрать?

## Теорема

Для  $n$   $b$ -битовых чисел и натурального числа  $r \leq b$  (цифры из  $r$  битов) алгоритм RADIX-SORT выполнит сортировку за время  $\Theta\left(\frac{b}{r}(n + 2^r)\right)$ .

Тем самым:

- ▶ Для  $b < \lfloor \log_2(n) \rfloor$  асимптотически оптимален выбор  $r = b$ .
- ▶ А для  $b \geq \lfloor \log_2(n) \rfloor$ :  $r = \lfloor \log_2(n) \rfloor$

# Свойства поразрядной сортировки

- ▶ Линейная, устойчивая, требуется дополнительная память (из-за сортировки подсчётом)
- ▶ Может понадобиться много проходов
- ▶ Несмотря на асимптотическую линейность, для конкретных значений  $n$  и  $r$  сортировки сравнением могут быть предпочтительнее из-за разных значений постоянных множителей

# Вариации поразрядной сортировки

- ▶ Рассмотренная версия — LSD (Least Significant Digit) — сортирует от младшего разряда к старшему
- ▶ MSD (Most Significant Digit) — идет от старшего разряда к младшему, разбивая исходный список на корзинки по значению разряда
- ▶ Окончание — когда просмотрели все разряды или все корзины имеют размер 1
- ▶ Сложность MSD версии —  $O(kn)$

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Общая идея

- ▶ На вход поступают  $n$  вещественных чисел, порождённых случайным процессом и равномерно распределённых в интервале  $[0, 1)$
- ▶  $[0, 1)$  разбивается на  $n$  одинаковых интервалов (карманов, buckets)
- ▶ Числа помещаются в список, соответствующий каждому карману; так как распределение равномерное, в одном кармане появляется не очень много чисел
- ▶ Все списки сортируются (вставкой)
- ▶ Результат получается объединением (один проход по всем спискам)

# Алгоритм

BUCKET-SORT( $A$ )

- 1  $n \leftarrow \text{length}[A]$
- 2 **for**  $i \leftarrow 1$  **to**  $n$
- 3     Вставить элемент  $A[i]$  в список  $B[\lfloor nA[i] \rfloor]$
- 4 **for**  $i \leftarrow 0$  **to**  $n - 1$
- 5     Сортировка вставкой списка  $B[i]$
- 6 Объединение списков  $B[0], B[1], \dots, B[n - 1]$

# Пример

$A$		$B$	
1	.78	0	$\emptyset$
2	.17	1	$\langle .12, .17 \rangle$
3	.39	2	$\langle .21, .23, .26 \rangle$
4	.26	3	$\langle .39 \rangle$
5	.72	4	$\emptyset$
6	.94	5	$\emptyset$
7	.21	6	$\langle .68 \rangle$
8	.12	7	$\langle .72, .78 \rangle$
9	.23	8	$\emptyset$
10	.68	9	$\langle .94 \rangle$



# Корректность

- ▶  $a_i$  и  $a_j$ ,  $0 \leq i, j \leq n$
- ▶  $a_i \leq a_j \Rightarrow \lfloor n * a_i \rfloor \leq \lfloor n * a_j \rfloor$
- ▶  $a_i$  и  $a_j$  попали в один интервал – по определению
- ▶  $a_i$  и  $a_j$  попали в разные интервалы – сохраняется порядок при слиянии интервалов

# Линейность

$n_i$  — количество элементов в кармане  $B[i]$ . Тогда время работы алгоритма:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \Rightarrow E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

Для всех  $i = 0, 1, \dots, n-1$ :

$$E[n_i^2] = 2 - \frac{1}{n} \Rightarrow T(n) = \Theta(n) + n \cdot O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

## Количество элементов в одном кармане

$$X_{ij} = I\{A[j] \in B[i]\} \Rightarrow n_i = \sum_{j=1}^n X_{ij},$$

$$E[n_i^2] = \sum_{j=1}^n E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{\substack{1 \leq k \leq n \\ k \neq j}} E[X_{ij} X_{ik}],$$

$$E[X_{ij}^2] = 1 \cdot \frac{1}{n} + 0 \cdot \left(1 - \frac{1}{n}\right) = \frac{1}{n},$$

$$E[X_{ij} X_{ik}] = E[X_{ij}] E[X_{ik}] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2},$$

$$E[n_i^2] = n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n}.$$

# Сортировка за линейное время

## Оценки

Асимптотические обозначения

Нижняя оценка сортировок сравнением

## Алгоритмы

Сортировка подсчётом

Поразрядная сортировка

Карманная сортировка

## Внешняя сортировка

Внешняя поразрядная сортировка

# Основные моменты

- ▶ Данные хранятся на внешнем носителе
- ▶ Размера оперативной памяти недостаточно для того, чтобы поместить в нее все данные
- ▶ Необходимо минимизировать количество обращений к носителю
- ▶ Основной метод - сортировка слиянием

## Пример

Файл содержит только целые числа от 0 до 7.

$Pass$	$File_1$	$File_2$	$File_3$	$File_4$
0	$\{0,1,\dots,7\}$			
1			$\{0,2,4,6\}$	$\{1,3,5,7\}$
2	$\{0,4\}\{1,5\}$	$\{2,6\}\{3,7\}$		
3			$\{0\}\{1\}\{2\}\{3\}$	$\{4\}\{5\}\{6\}\{7\}$
3.5			$\{0\}\{1\}\dots\{7\}$	

# Общая идея

- ▶ Первый файл изначально содержит исходную последовательность, второй файл пустой, сортирующий бит – младший
- ▶ Проходим по первому файлу, записываем в третий все элементы, у которых сортирующий бит равен нулю, в четвертый – единице
- ▶ Повторяем алгоритм для второго файла
- ▶ Третий файл становится первым, четвертый – вторым
- ▶ Увеличиваем сортирующий бит на 1, повторяем алгоритм, пока не достигнем старшего бита

# Алгоритм

```
EXT-RADIX-SORT(Files[])
1  bit  $\leftarrow$  0
2  while bit  $\leq$  MAX_BIT
3      while ReadItem(Files[0], item)
4          if GetBit(item, bit) == 0
5              WriteItem(Files[2], item)
6          else
7              WriteItem(Files[3], item)
8      // Аналогично для Files[1]
9      Files[0]  $\leftarrow$  Files[2]
10     Files[1]  $\leftarrow$  Files[3]
11     bit  $\leftarrow$  bit + 1
```



## Еще один пример

Файл содержит целые числа от 0 до 9.

	$File_1$	$File_2$	$File_3$	$File_4$
0	$\{0, 1, \dots, 9\}$			
1		$\{0, 2, 4, 7\}$	$\{1, 5, 6\}$	$\{3, 8, 9\}$
1.4	$\{0\}$		$\{1, 5, 6\}\{2, 7\}$	$\{3, 8, 9\}\{4\}$
1.9	$\{0\}\{1\}\{2\}$	$\{6\}\{7\}$		$\{3, 8, 9\}\{4\}\{5\}$
2.2	$\{0\}\{1\}\{2\}\{3\}$	$\{6\}\{7\}\{8\}$	$\{9\}$	$\{4\}\{5\}$
2.8	$\{0\}\{1\} \dots \{9\}$			

## Еще один пример

Файл содержит целые числа от 0 до 9.

	$File_1$	$File_2$	$File_3$	$File_4$
0	$\{0, 1, \dots, 9\}$			
1		$\{0, 2, 4, 7\}$	$\{1, 5, 6\}$	$\{3, 8, 9\}$
1.4	$\{0\}$		$\{1, 5, 6\}\{2, 7\}$	$\{3, 8, 9\}\{4\}$
1.9	$\{0\}\{1\}\{2\}$	$\{6\}\{7\}$		$\{3, 8, 9\}\{4\}\{5\}$
2.2	$\{0\}\{1\}\{2\}\{3\}$	$\{6\}\{7\}\{8\}$	$\{9\}$	$\{4\}\{5\}$
2.8	$\{0\}\{1\} \dots \{9\}$			

- ▶ Всего 2.8 прохода по содержимому файла
- ▶ Для предыдущего алгоритма было 3.5 прохода
- ▶ При подсчете проходов предполагается, что числа имеют равную частоту