Разработка и анализ алгоритмов

Лекция 4
Сортировка.

Сергей Леонидович Бабичев

Деревья решений

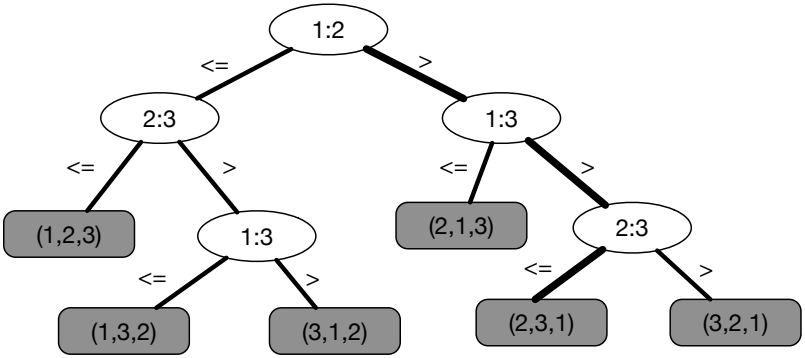
Нижняя оценка сложности алгоритмов сортировки сравнениями

- Рассмотрено несколько сортировок.
- Ни одна не имеет оценки меньше $O(N \log N)$.
- Это — фундаментальное ограничение сортировок сравнениями.

Деревья решений

- Имеются узлы и вершины.
- Каждый узел имеет ровно двух потомков.
- Каждый узел помечен меткой вида $i : j$, где $1 \leqslant i, j \leqslant N$.
- Каждая терминальная вершина содержит окончательное решение задачи в виде одной из перестановок множества $\{1, 2, \ldots, N\}$.
- Выполнение алгоритма — прохождение от корня к вершине.
- Необходимое условие: в терминальных вершинах должны оказаться все возможные перестановки.

Дерево решений



Нижняя граница сложности

- Общее количество терминальных вершин $N!$.
- Узел — нетерминальная вершина.
- Дерево состоит из терминальных вершин и узлов $\rightarrow$ общее количество вершин $V = V_{term} + V_{nonterm} > N!$
- Полное двоичное дерево глубины H состоит из $2^H - 1$ вершин.
- Минимальная глубина дерева $H \geqslant \log_2 V$

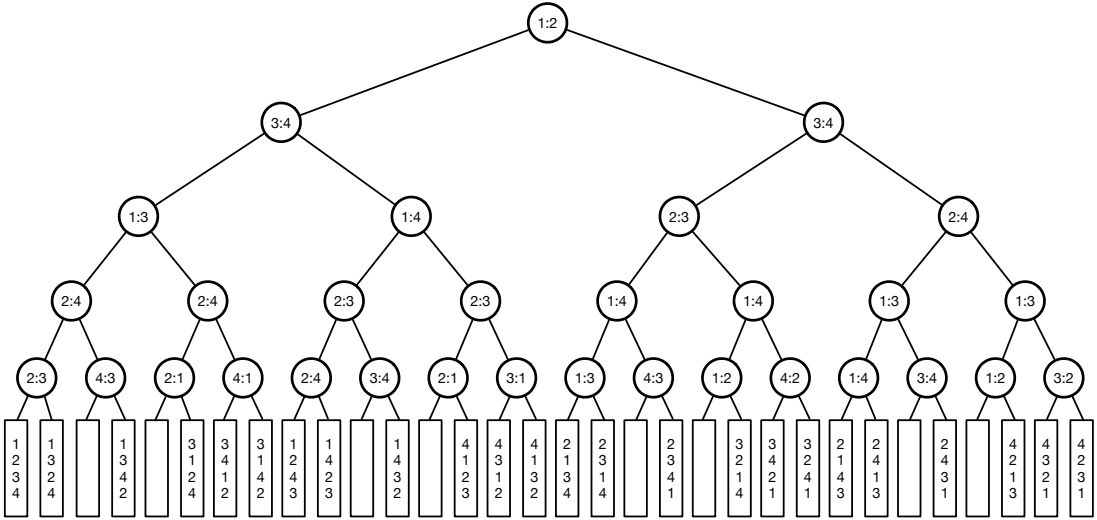
Формула Стирлинга:

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Отсюда $H \sim N \log N$.

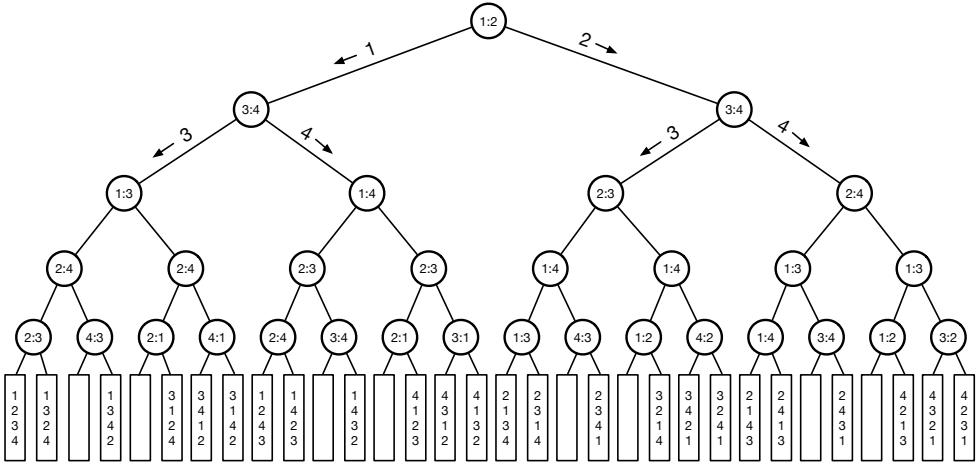
Сортирующие сети.

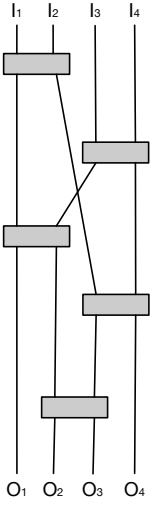
- Оптимальное дерево принятия решений зависит от истории поиска.
- Составим другое дерево, высотой h, в котором участники сравнений могут быть только одним из результатов родительского сравнения.
- Если сравнивались ключи K_i и K_j, то последующие сравнения одинаковые для случая $K_i < K_j$ и для случай $K_i > K_j$ за исключением того, что i и j меняются местами.


1:2
3:4
3:4
1:3
1:4
2:3
2:4
2:4
2:4
2:3
2:3
1:4
1:4
1:3
1:3
2:3
4:3
2:1
4:1
2:4
3:4
2:1
3:1
1:3
4:3
1:2
4:2
1:4
3:4
1:2
3:2
1234
1324
1342
3124
3412
3142
1243
1423
1432
4123
4312
4132
2134
2314
2341
3214
3241
3421
2143
2413
2431
4213
4231
4321

Сортирующие сети

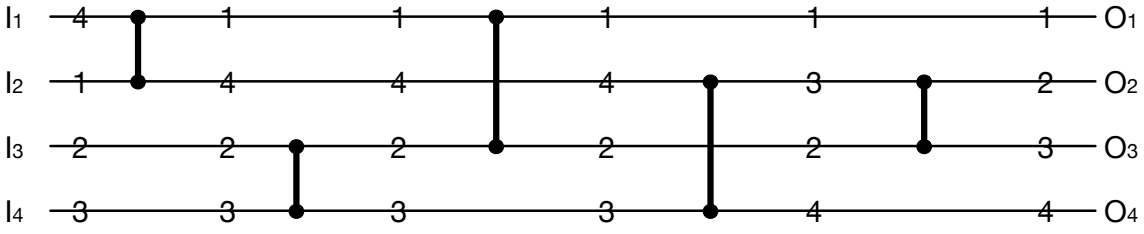
- После сравнения *слева* должен оказаться меньший элемент.
- Устройство сравнения и обмена называется *компаратор* или *компараторный модуль*.
- На вход компаратора подаются две линии с элементами сравнения.
- На выходе компаратора две линии содержат упорядоченные элементы.


1:2
1
2
3:4
3:4
3
4
3
4
1:3
1:4
2:3
2:4
2:4
2:4
2:3
2:3
1:4
1:4
1:3
1:3
2:3
4:3
2:1
4:1
2:4
3:4
2:1
3:1
1:3
4:3
1:2
4:2
1:4
3:4
1:2
3:2
1234
1324
1342
3124
3412
3142
1243
1423
1432
4123
4312
4132
2134
2314
2341
3214
3421
3241
2143
2413
2431
4213
4321
4231

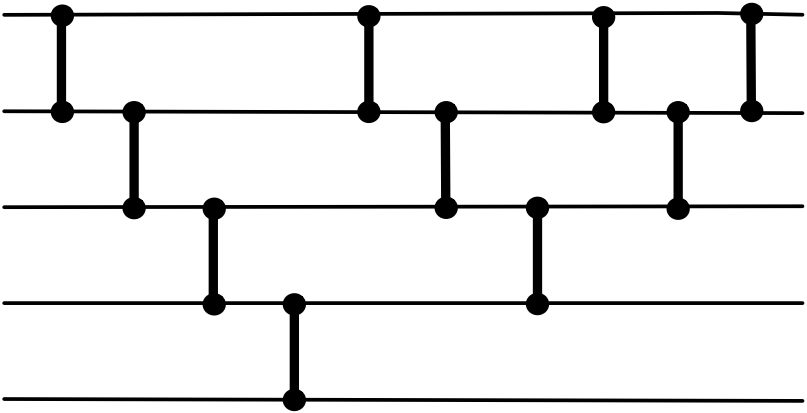

I1
I2
I3
I4
O1
O2
O3
O4

Сортирующие сети

- Удобно рисовать процесс сортировки компараторами горизонтальными линиями, а сами компараторы — вертикальными перемычками.
- Это — сортировка массива $(4, 1, 2, 3)$.

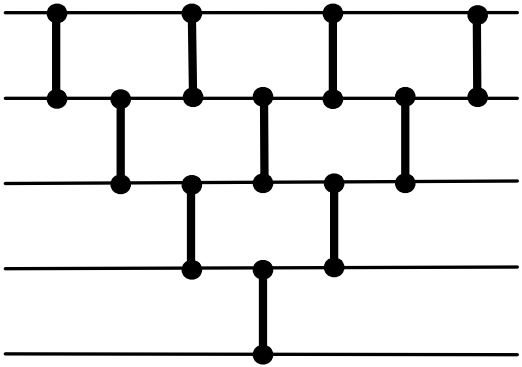


Сортирующие сети: сортировка пузырьком



Сортирующие сети: зачем они?

- В сортирующей сети сортировки пузырьком для 5 элементов требуется 10 операций.
- Пока все операции происходят последовательно.
- Можно переносить перемычки влево, если они не конфликтуют с другими.
- Это означает, что сравнения имеет смысл проводить как можно раньше.
- Сравнения и обмены могут производиться параллельно!



Сортирующие сети: построение

- *Глубина* сортирующей сети — количество шагов сортировки. За один шаг исполняются все независимые сравнения.
- Для существующего последовательного алгоритма построить сортирующую сеть несложно.
- Как доказать, что построенная сортирующая сеть сортирует все $N!$ последовательностей?
- Упростим задачу.

Теорема о сортировке нулей и единиц

Theorem

Если сеть с N входами сортирует в неубывающем порядке все последовательности, состоящие из 0 и 1, то она сортирует и любую произвольную последовательность из N элементов.

Доказательство.

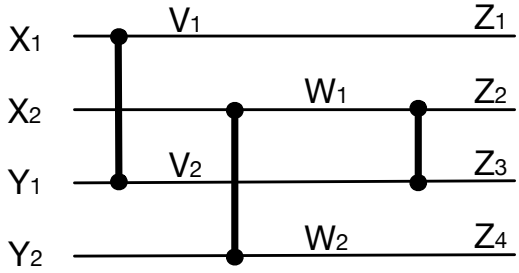
Пусть $f(x)$ — любая монотонная функция, для которой $f(x) \leqslant f(y)$ при $x \leqslant y$. Если данная сеть преобразует $(x_1, x_2, \ldots, x_N)$ в $(y_1, y_2, \ldots, y_N)$, то эта сеть преобразует $(f(x_1), f(x_2), \ldots, f(x_N))$ в $(f(y_1), f(y_2), \ldots, f(y_n))$. Пусть $y_i > y_{i+1}$ для некоторого i. Рассмотрим монотонную функцию f такую, что $f(y) = 0$ для всех $y < y_i$ и $f(y) = 1$ для всех $y \geqslant y_i$. По определению она даёт последовательность 0 и 1, не сортируемую сетью.
Отсюда, если все последовательности 0 и 1 сортируются сетью, то $y_i \leqslant y_{i+1}$ для всех $i \in [1, N)$.

Чётно-нечётная сортировка слиянием (Бэтчера): сеть слияния

- Идея сортировки: для массива с $m + n$ элементами сортировать независимо первые m и последние n элементов, применив затем (m, n) *сеть слияния*.
- Сеть слияния строится по индукции:
  1. База: для $m = 0$ или $n = 0$ сеть пустая. Для $m = n = 1$ нужен один компаратор.
  2. Переход: пусть мы сливаем $(x_1, \dots, x_m)$ и $(y_1, \dots, y_n)$. Рекурсивно сольём последовательности $(x_1, x_3, \dots)$ и $(y_1, y_3, \dots)$ в элементы последовательности $(v_1, v_2, ...)$, а последовательности $(x_2, x_4, \dots)$ и $(y_2, y_4, \dots)$ в элементы последовательности $(w_1, w_2, \dots)$.
  3. Применим компараторы для пар (v_2, w_1), (v_3, w_2) ...
  4. Итоговая последовательность отсортирована.

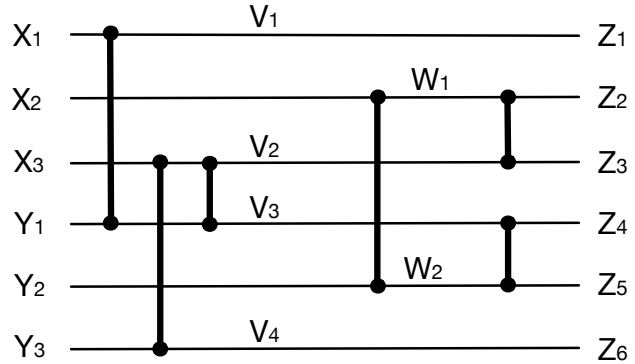
Сортировка Бэтчера: сеть слияния (2+2)

- Нужно слить отсортированные (x_1, x_2) и (y_1, y_2).
- Сливаем (x_1) и (y_1) — база, один компаратор, выход (v_1, v_2)
- Сливаем (x_2) и (y_2) — база, один компаратор, выход (w_1, w_2)
- Сливаем (w_1) и (v_2) — база, один компаратор.



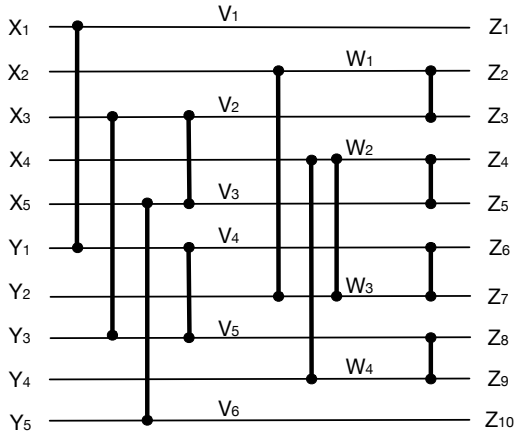
Сортировка Бэтчера: сеть слияния (3+3)

- Нужно слить отсортированные (x_1, x_2, x_3) и (y_1, y_2, y_3).
- Сливаем (x_1) и (y_1) в xа так же (x_2) и (y_2), получая сеть $(2+2)$, которую сливать умеем. — база, один компаратор, выход (v_1, v_2)
- Сливаем (x_2) и (y_2) — база, один компаратор, выход (w_1, w_2)
- Сливаем (w_1) и (v_2) — база, один компаратор.



Сортировка Бэтчера: сеть слияния (5+5)

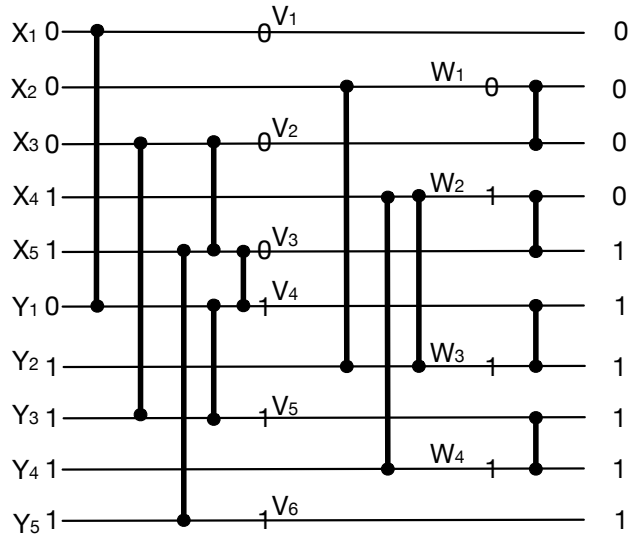
- Нужно слить отсортированные $(x_1, x_2, x_3, x_4, x_5)$ и $(y_1, y_2, y_3, y_4, y_5)$.
- Сливаем (x_1, x_3, x_5) и (y_1, y_3, y_5) — рекурсия, слияние (3+3), получаем v_i.
- Сливаем (x_2, x_4) и (y_2, y_4) — рекурсия, слияние $(2 + 2)$, получаем w_i
- Сливаем (w_i) и (v_2) — база, один компаратор.



Сортировка Бэтчера: корректность слияния *(n+m)*

- Воспользуемся теоремой о сортировке нулей и единиц.
- m-последовательность в слиянии состоит из k нулей и далее $m - k$ единиц.
- n-последовательность состоит из l нулей и $n - l$ единиц.
- Последовательность $(v_1, \ldots, v_m)$ после слияния будет содержать $\lceil k/2 \rceil + \lceil l/2 \rceil$ нулей и далее единицы.
- Последовательность $(w_1, \ldots, w_n)$ после слияния будет содержать $\lfloor k/2 \rfloor + \lfloor l/2 \rfloor$ нулей и далее единицы.
- Выражение
$$d = (\lceil k/2 \rceil + \lceil l/2 \rceil) - (\lfloor k/2 \rfloor + \lfloor l/2 \rfloor)$$
может принимать значения только $0, 1, 2$.
- Если $d = 0$ или $d = 1$, то последовательность $z_1, \ldots,$ уже упорядочена.
- Если $d = 2$, то достаточно одной операции обмена на последнем этапе для упорядочивания.

Сортировка Бэтчера: корректность слияния $(n+m)$: иллюстрация $d = 2$.



Сортировка Бэтчера: алгоритм

- На 1-м шаге параллельно сортируются группы элементов, находящиеся на позициях $(1, 2), (3, 4), \ldots$. Нечётный последний элемент образует отдельную группу.
- На каждом шаге группы укрупняются в два раза и сортируются парами.

Сложность сортировки:
количество компараторов $O(N \log^2 N)$; глубина $O(\log^2(N))$.

Сортировка с использованием свойств элементов

Сортировка подсчётом

- Есть ли алгоритмы сортировки со сложностью меньшей $O(N \log N)$?

  Да, если использовать свойства ключей.
- Пусть множество значений ключей ограничено $D(K) = \{K_{min}, \ldots, K_{max}\}$.
- Тогда при наличии добавочной памяти в $|D(K)|$ ячеек сортировку можно произвести за $O(N)$.

Сортировка подсчётом

- Сортируем массив $S = \{10, 5, 14, 7, 3, 2, 18, 4, 5, 13, 6, 8\}$
- Заранее известно, что значения массива натуральные числа, которые не превосходят 20.
- Заводим массив $F[1..20]$, содержащий вначале нулевые значения.

$$F = \boxed{0}$$

- Проходим по массиву S. $S_1 = 10; F_{10} \leftarrow F_{10} + 1$.

$$F = \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$$

Сортировка подсчётом

- $S_2 = 5; F_5 \leftarrow F_5 + 1.$

$$F = \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$$

- $\ldots$
- $S_{12} = 5; F_8 \leftarrow F_8 + 1.$

$$F = \boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{2}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}$$

- Заключительный вывод по ненулевым элементам F:

$$S = \{2, 3, 4, 5, 5, 6, 7, 8, 10, 13, 14, 18\}$$

Сортировка подсчётом: особенности

- Ключи должны быть перечислимы.
- Пространство значений ключей должно быть ограниченным.
- Требуется дополнительная память $O(|D(K)|)$.
- Сложность $O(N) + O(|D(K)|)$

Поразрядная сортировка

- Усложнение варианта сортировки подсчётом - поразрядная сортировка.
- Разобьём ключ на фрагменты — разряды и представим его как массив фрагментов.
- Все ключи должны иметь одинаковое количество фрагментов.
- Пример: ключ 375 можно разбить на 3 фрагмента $\{3, 7, 5\}$, тогда ключ 5 — тоже на 3 $\{0, 0, 5\}$.

Поразрядная сортировка

Требуется отсортировать массив $S = \{153, 266, 323, 614, 344, 993, 23\}$.

- Будем полагать, что разбиение проведено на 3 фрагмента.
- $\{\{1,5,3\},\{2,6,6\},\{3,2,3\},\{6,1,4\},\{3,4,4\},\{9,9,3\},\{0,2,3\}\}$
- Рассматривая последний фрагмент, как ключ, устойчиво отсортируем фрагменты методом подсчёта.
- $\{\{1,5,3\},\{3,2,3\},\{9,9,3\},\{0,2,3\},\{3,4,4\},\{6,1,4\},\{2,6,6\}\}$
- Теперь устойчиво отсортируем по второму фрагменту.
- $\{\{6,1,4\},\{3,2,3\},\{0,2,3\},\{3,4,4\},\{1,5,3\},\{2,6,6\},\{9,9,3\}\}$
- И, наконец, по первому фрагменту.
- $\{\{0,2,3\},\{1,5,3\},\{2,6,6\},\{3,2,3\},\{3,4,4\},\{6,1,4\},\{9,9,3\}\}$

Поразрядная сортировка: особенности

- Требует ключи, которые можно трактовать как множество перечислимых фрагментов.
- Требует дополнительной памяти $O(|D(K_i)|)$ на сортировку фрагментов.
- Сложность постоянна и равна $O(N \cdot |D(K_i)|)$.

Внешняя сортировка

Сортировка больших данных

- Сортировка больших данных — очень трудоёмкая задача.
- Две основных проблемы:
  1. Для сортируемых данных недостаточно быстрой оперативной памяти.
  2. Время сортировки превосходит приемлемые границы.
- При недостатке оперативной памяти применяют *внешнюю* сортировку.

Сортировка больших данных

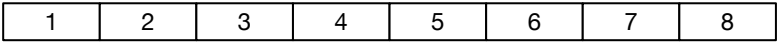
- Обработка всех данных одновременно невозможна.
- Используется абстракция **лента**, основанная на абстракции *последовательность* с методами:
- create
- destroy
- put
- get

Сортировка больших данных

- Применим операцию *слияния*.
- Входными данными *двухпутевого слияния* являются две отсортированные ленты размером N_1 и N_2.
- Выходные данные — другая отсортированная лента.
- Сложность операции слияния — $\Theta(N_1 + N_2)$.
- Чанк (chunk) — фрагмент данных, помещающихся в оперативной памяти.

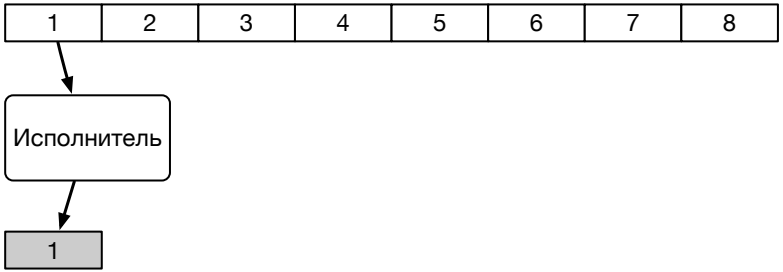
Сортировка слиянием

Пусть исходная лента содержит 8 чанков.



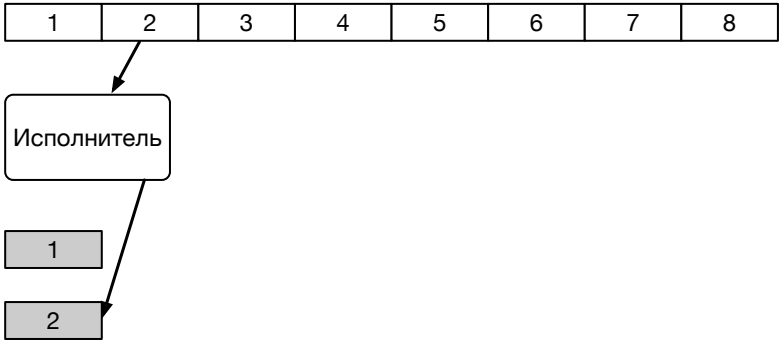
Сортировка слиянием

Первый этап. Считывается первый чанк, сортируется внутренней сортировкой и отправляется на первую временную ленту.



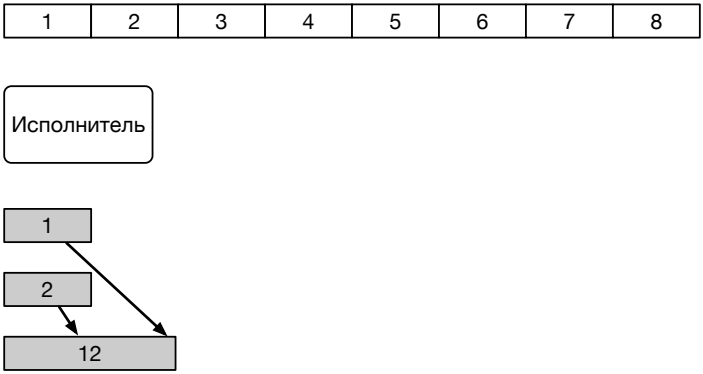
Сортировка слиянием

Второй этап. Второй чанк сортируется и отправляется на вторую временную ленту.



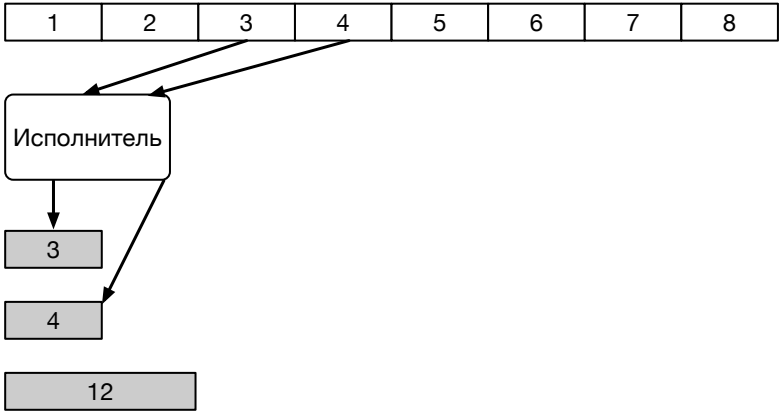
Сортировка слиянием

Третий этап — слияние. Сливаются первая и вторая временные ленты.



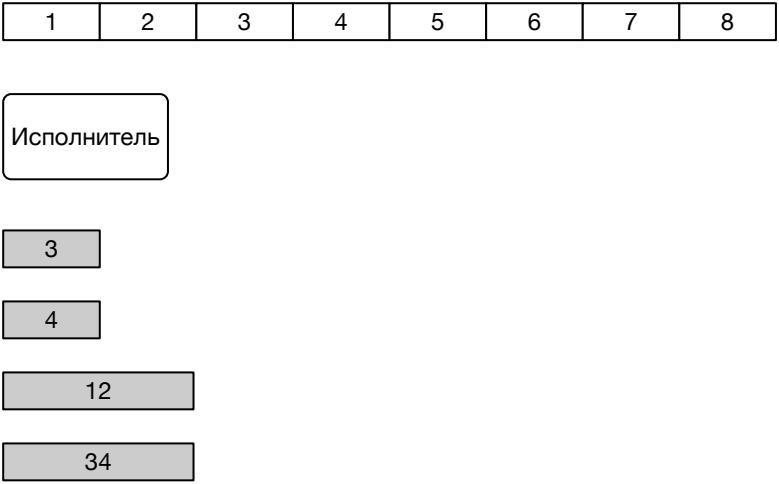
Сортировка слиянием

Аналогично считываются, сортируются и выводятся на временные ленты чанки 3 и 4.



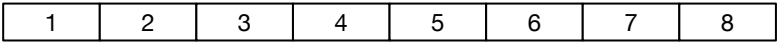
Сортировка слиянием

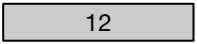
Аналогично временные ленты сливаются в ещё одну, четвёртую. Здесь мы не можем использовать меньшее количество лент.

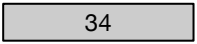


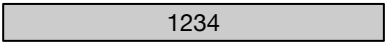
Сортировка слиянием

Сливаем ленты содержащие 12 и 34, получаем ленту 1234.



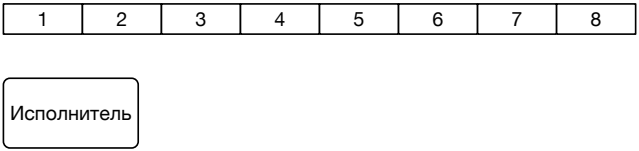






Сортировка слиянием

Для получения ленты 5678 требуется 4 временные ленты. Плюс лента 1234. Итого



— 5 лент.

Оценка сложности внешней сортировки слиянием

- Внутренних сортировок в этом алгоритме — K, количество чанков.
- Сложность внутренней сортировки

$$O\left(\frac{N}{K} \times \log \frac{N}{K}\right) = O(N \log N).$$

- Сложность операции слияния $O(N)$.
- Количество слияний $O(\log K)$.
- Общая сложность $O(N \log N)$.
- Сложность по количеству временных лент $O(\log K)$.

Усовершенствование алгоритма

- Мы заметили, что при операции слияния дополнительной памяти не требуется, достаточно памяти для двух элементов.
- Можно ли произвести внешнюю сортировку без использования большого буфера?
- Да, если отказаться от понятия *чанк* и использовать понятие *серия*.
- *Серия* — неубывающая последовательность на ленте.

Сортировка сериями

Пусть имеется лента

$$\underbrace{14, 4, 2, 7, 5, 9, 6, 11, 3, 1, 8, 10, 12, 13}.$$

Заводим две вспомогательных ленты, в каждую из которых помещаем очередную серию длины 1 из входной ленты.

$$\begin{cases} \underbrace{14}, \underbrace{2}, \underbrace{5}, \underbrace{6}, \underbrace{3}, \underbrace{8}, \underbrace{12} \\ \underbrace{4}, \underbrace{7}, \underbrace{9}, \underbrace{11}, \underbrace{1}, \underbrace{10}, \underbrace{13} \end{cases}.$$

Инвариант: внутри серии длины K все значения упорядочены по неубыванию.

Сортировка сериями

Каждую из серий парами сливаем в исходный файл.

$$\begin{cases} \underbrace{14}, \underbrace{2}, \underbrace{5}, \underbrace{6}, \underbrace{3}, \underbrace{8}, \underbrace{12} \\ \underbrace{4}, \underbrace{7}, \underbrace{9}, \underbrace{11}, \underbrace{1}, \underbrace{10}, \underbrace{13} .. \end{cases}$$

Инвариант операции слияния серий: совокупная последовательность является серией удвоенной длины.

$$\underbrace{4, 14}, \underbrace{2, 7}, \underbrace{5, 9}, \underbrace{6, 11}, \underbrace{1, 3}, \underbrace{8, 10}, \underbrace{12, 13}.$$

Сортировка сериями

Серии длины 2 попеременно помещаем на выходные ленты.

$$\underbrace{4, 14}, \underbrace{2, 7}, \underbrace{5, 9}, \underbrace{6, 11}, \underbrace{1, 3}, \underbrace{8, 10}, \underbrace{12, 13}.$$

$$\begin{cases} \underbrace{4, 14}, \underbrace{5, 9}, \underbrace{1, 3}, \underbrace{12, 13} \\ \underbrace{2, 7}, \underbrace{6, 11}, \underbrace{8, 10}. \end{cases}$$

Сортировка сериями

Снова каждую из серий парами сливаем в исходную ленту.

$$\begin{cases} \underbrace{4, 14}, \underbrace{5, 9}, \underbrace{1, 3}, \underbrace{12, 13} \\ \underbrace{2, 7}, \underbrace{6, 11}, \underbrace{8, 10}. \end{cases}$$

Длина полных серий в выходной ленте не меньше 4. Только последняя серия может иметь меньшую длину.

$$\underbrace{2, 4, 7, 14}, \underbrace{5, 6, 9, 11}, \underbrace{1, 3, 8, 10}, \underbrace{12, 13}.$$

Сортировка сериями

Третий этап: формируем временные ленты сериями по 4.

$$\underbrace{2,4,7,14}, \underbrace{5,6,9,11}, \underbrace{1,3,8,10}, \underbrace{12,13}.$$

$$\begin{cases} \underbrace{2,4,7,14}, & \underbrace{1,3,8,10} \\ \underbrace{5,6,9,11}, & \underbrace{12,13}. \end{cases}$$

Длина полных серий в выходной ленте не меньше 4. Только последняя серия может иметь меньшую длину.

Сортировка сериями

Сливаем серии длины 4.

$$\begin{cases} \underbrace{2, 4, 7, 14}, & \underbrace{1, 3, 8, 10} \\ \underbrace{5, 6, 9, 11}, & \underbrace{12, 13}. \end{cases}$$

Слияние серий длины 4 обеспечивает длину серий 8.

$$\underbrace{2, 4, 5, 6, 7, 9, 11, 14}, \underbrace{1, 3, 8, 10, 12, 13}.$$

Сортировка сериями

Последний этап: разбивка на серии длины 8 с последующим слиянием.

$$\underbrace{2, 4, 5, 6, 7, 9, 11, 14}, \underbrace{1, 3, 8, 10, 12, 13}.$$

Разбивка:

$$\begin{cases} \underbrace{2, 4, 5, 6, 7, 9, 11, 14} \\ \underbrace{1, 3, 8, 10, 12, 13}. \end{cases}$$

Слияние:

$$\underbrace{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}.$$

Сортировка сериями: оценка сложности алгоритма

- За один проход примем операцию разбивки с последующим слиянием.
- На каждом проходе участвуют все элементы лент по два раза.
- Инвариант: длина серии после прохода k равна 2^k.
- Завершение алгоритма: длина серии не меньше N.
- Итого $\log N$ проходов.
- Сложность алгоритма: $O(N \log N)$
- Сложность по памяти: $O(1)$
- Сложность по ресурсам: две временные ленты.

Сортировка сериями: возможные улучшения

- Сортировка сериями: длина серии начинается от 1 и для полной сортировки требуется ровно $\lceil \log_2 N \rceil$ итераций.
- При этом первые итерации производятся с небольшой длиной серии.
- Идея! Сократить число итераций, использовав больше, чем 1 элемент памяти (опять ввести чанки).

Сортировка сериями: улучшенный вариант

- Подбираем такое число k_0, при котором серия длиной 2^{k_0} помещается в доступную память.
- Разбиваем исходную ленту на серии: считывается первый чанк длиной 2^{k_0}, сортируется внутренней сортировкой, пишется на первую ленту. Второй чанк после сортировки пишется на вторую ленту.
- После подготовки возвращаемся к алгоритму сортировки сериями.

Сложность алгоритма — $O(N \log N)$, количество итераций сокращается на k. Пусть $N = 10^8$. Тогда $\log_2 N \approx 27$. Для $k = 20$ в память помещается 2^{20} элементов, что вполне реально. Тогда общее количество итераций составит $27 - 20 + 1 = 8$ вместо 28. Profit!

Сортировка и параллельные вычисления

Сортировка и параллельные вычисления

- Современные компьютеры содержат по несколько исполнителей машинного кода.
- Как использовать несколько исполнителей для сортировки?

Особенности параллельного исполнения

- Каждый из исполнителей может исполнять свой поток инструкций.
- В программном коде это выглядит как одновременное исполнение нескольких функций.
- Все исполнителей могут иметь совместный доступ к общим данным.
- Исполнитель в современной терминологии называется *вычислительный поток*, *thread*.

Проблемы параллельного исполнения

.

- Совместный доступ к общим переменным — и благо и зло одновременно.
- Благо — так как это удобный способ взаимодействия, обмен данными.
- Зло — так как это может привести к конфликтам.

Проблемы параллельного исполнения

.

- Два исполнителя используют совместные переменные:

```
int a = 2, b = 10;

void thread1() {
   a += b; // 1a
   b = 5;  // 1b
}

void thread2() {
   b = 13; // 2b
   a *= b; // 2a
}
```

- Чему равны переменные a и b после окончания обоих исполнителей?

Проблемы параллельного исполнения

- Порядок исполнения недетерминирован и результатов может быть несколько при различных прогонах алгоритма.
- Задача становится комбинаторной.
- Результат зависит от взаимного порядка исполнения.
- Если обозначить за $t(x)$ абсолютное время получения результатов исполнения соответствующих инструкций, то известно лишь, что $t(1a) < t(1b)$ и $t(2b) < t(2a)$.
- Таким образом, возможных путей исполнения несколько:
- $1a \to 1b \to 2a \to 2b$
- $1a \to 2a \to 1b \to 2b$
- $1a \to 2a \to 2b \to 1b$
- $\ldots$

Проблемы параллельного исполнения

- Более того, операции $1a$ и $2a$ атомарны для исполнителя «Язык Си» и не атомарны для исполнителя «современный процессор»!
- Инструкция `a += b` превратится в несколько операций:
  1. Загрузка b в регистр процессора
  2. Загрузка a в регистр процессора
  3. Добавление значения b регистру, содержащему a
  4. Сохранение получившегося значения в a
- На любой из этих операций возможна передача управления другому исполнителю.

Проблемы параллельного исполнения

- Для борьбы с такими проблемами автор алгоритма должен использовать *примитивы синхронизации*
- На исполнение *примитивов синхронизации* требуется значительное время, за которое можно исполнить сотни и тысячи обычных операций.
- Простейший способ избежать проблем — ограничить использование общих данных *критическими секциями*.
- Основные операции лучше всего производить над *локальными* для каждого исполнителя данными и только в отдельные моменты использовать *точки синхронизации* для обмена.

Проблемы параллельного исполнения

- Разработка параллельных версий классических алгоритмов — отдельная, очень сложная задача.
- Алгоритмы, которые наиболее эффективны в варианте для одного исполнителя, чаще всего непригодны для варианта с несколькими исполнителями.

Сортировка и параллельные вычисления

- Параллельная сортировка имеет общие свойства с внешней:
  1. Данные разбиваются на непересекающиеся подмножества.
  2. Каждое подмножество обрабатывается независимо.
  3. После независимой обработки используется слияние.

Сравнение методов сортировки

Algo	Best case	Average case	Worst case	Memory	Stable?
Bubble	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Shell	$O(N^{\frac{7}{6}})$	$O(N^{\frac{7}{6}})$	$O(N^{\frac{4}{3}})$	$O(1)$	Нет
Comb	$O(N \log N)$	$O(\frac{N^2}{2^p})$	$O(N^2)$	$O(1)$	Нет
Insertion	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Selection	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Quick	$O(N \log N)$	$O(N \log N)$	$O(N^2)$	$O(1)$	Нет
Merge	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$	Да
Heap	$O(N)$	$O(N \log N)$	$O(N \log N)$	$O(1)$	Нет
Tim	$O(N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$	Да
Count	$O(N)$	$O(N)$	$O(N)$	$O(N)$	Да
Radix	$O(N)$	$O(N)$	$O(N)$	$O(N)$	Да