ТЕХНОСФЕРА

Сортировка

Лекция 3

План лекции

1. Задача сортировки
2. Сортировки сравнением
3. Нахождение медианы множества
4. Быстрая сортировка
5. Сортировки с использованием свойств элементов
6. Внешняя сортировка
7. Сортировка и параллельные вычисления
8. Сравнительный анализ методов сортировки

Задача сортировки

Задача сортировки

Имеется последовательность из n *ключей*.

$$k_1, k_2, \ldots, k_n$$

Требуется: упорядочить ключи по *не убыванию* или *не возрастанию*.
Это означает: найти перестановку ключей

$$p_1, p_2, \ldots, p_n$$

такую, что

$$k_{p_1} \leqslant k_{p_2} \leqslant \cdots \leqslant k_{p_n}$$

или

$$k_{p_1} \geqslant k_{p_2} \geqslant \cdots \geqslant k_{p_n}$$

Задача сортировки

Элементами сортируемой последовательности могут иметь любые типы данных. Обязательное условие — наличие *ключа*.

Последовательность:
`(Москва, 10000000), (Нью-Йорк, 12000000), (Париж, 9000000), (Токио, 20000000), (Лондон, 10000000), (Дели, 9000000)`

Ключ — число жителей

Устойчивость сортировки

Алгоритм сортировки *устойчивый*, если он сохраняет относительный порядок элементов.

Начальная последовательность:
`(Москва, 10000000), (Нью-Йорк, 12000000), (Париж, 9000000), (Токио, 20000000), (Лондон, 10000000), (Дели, 9000000)`

Устойчивая сортировка:
`(Токио, 20000000), (Нью-Йорк, 12000000), (Москва, 10000000), (Лондон, 10000000), (Париж, 9000000), (Дели, 9000000)`

Неустойчивая сортировка:
`(Токио, 20000000), (Нью-Йорк, 12000000), (Лондон, 10000000), (Москва, 10000000), (Париж, 9000000), (Дели, 9000000)`

Сортировки сравнением.

Сортировка сравнением

Один из видов сортировки: *сортировка сравнением*.

Требования к алгоритму: **для ключей должна существовать операция сравнения**

$$a < b$$

Полагается, что

$$not(a < b) \wedge not(b < a) \rightarrow a = b$$

Это необходимое условие для соблюдения *закона трихотомии*: для любых a, b либо $a < b$, либо $a = b$, либо $a > b$.

Сортировка в языке Си

- Требуется функция сравнения элементов.
- `int cmp(void const *el1, void const *el2);`
- Должна возвращать 0, если элементы равны, что-то отрицательное, если первый меньше и что-то положительное, если первый больше.

```
int cmp_int(const void *el1, const void*el2) {
return *(int *)el1 - *(int *)el2;
}
```

Сортировка в языке Си

```
#include<stdlib.h>

...
int a[100];
...
qsort(a, 100, sizeof a[0], cmp_int);
// Здесь a - отсортирован по возрастанию.
```

Сортировка в языке C++

- Должна быть определена операция сравнения элементов, подчиняющаяся закону трихотомии.

```
struct point {
   double x,y;
};

bool operator<(const point &l, const point &r) {
   if (l.x < r.x) return true;
   if (l.x > r.x) return false;
   return l.y < r.y;
}
```

Сортировка в языке C++

- Тогда сортировка допустима:

```
#include <algorithm>
  ...
  point parr[100];
  ...
  std::vector<point> pvect;

  std::sort(parr,parr+100); // Допустимо для массивов
  std::sort(pvect.begin(), pvect.end()); // И векторов
```

Сортировка в языке C++

- Для шаблонов `pair` и `tuple` компилятор сам генерирует лексикографическую сортировку:

```
#include <algorithm>
#include <stdio.h>
#include <tuple>
using namespace std;

int main() {
  tuple<double,int,double> t[3];
  t[0]={10,10,10};   t[1]={10,5,20};  t[2]={5, 5,10};
  sort(t,t+3);
  for (auto q: t) {
    printf("(%g,%d,%g)\n",
      get<0>(q), get<1>(q), get<2>(q));
  }
}
$ ./sort
(5,5,10)
(10,5,20)
(10,10,10)
$
```

Сортировка в языке C++

- Для нестандартных функций сравнения удобно использовать *замыкания* C++.

```
#include <algorithm>
#include <stdio.h>
using namespace std;

int main() {
  int t[10] = {5,4,9,10,1,3,2,7,6,8};
  sort(t+3,t+7, [](int l, int r) -> bool {return l > r;});
  for (auto q: t) {
    printf("%d ", q);
  }
}
$ ./sort
5 4 9 10 3 2 1 7 6 8
$
```

Понятие *инверсии*

Определение: *Инверсия* — пара ключей с нарушенным порядком следования.

$$\{4, 15, 6, 99, 3, 15, 1, 8\}$$

Имеются следующие инверсии:
`(4,3), (4,1), (15,6), (15,3), (15,1), (15,8), (6,3), (6,1), (99,3), (99,15), (99,1), (99,8), (3,1), (15,1), (15,8)`
Перестановка соседних элементов, расположенных в ненадлежащем порядке, уменьшает инверсию ровно на 1.
Количество инверсий в любом множестве конечно, в отсортированном — равно нулю.
Следовательно, количество обменов для сортировки конечно и не превосходит числа инверсий.

Сортировка пузырьком

Один из простейших в реализации алгоритмов.

Основная идея: до тех пор, пока соседние элементы не в порядке, меняем их местами.

$$\{\underbrace{10, 4}, 14, 25, 77, 2\}$$

$$\{4, \underbrace{10, 14}, 25, 77, 2\}$$

$$\{4, 10, \underbrace{14, 25}, 77, 2\}$$

$$\{4, 10, 14, \underbrace{25, 77}, 2\}$$

$$\{4, 10, 14, 25, \underbrace{77, 2}\}$$

$$\{4, 10, 14, 25, 2, 77\}$$

Сортировка пузырьком: сложность алгоритма

Лучший случай — { 1, 2, 3, 4, 5, 6} – $O(N)$.

Худший случай — { 6, 5, 4, 3, 2, 1} – $O(N^2)$.

Средний случай — $O(N^2)$

Сортировка пузырьком

```
void bubblesort(int *a, int n) {
   bool sorted = false;
   while (!sorted) {
      sorted = true;
      for (int i = 0; i < n-1; i++) {
        if (a[i] > a[i+1]) {
           int tmp = a[i];
           a[i] = a[i+1];
           a[i+1] = tmp;
           sorted = false;
        }
      }
    }
}
```

Сортировка пузырьком: инвариант

Инвариант: после i-го прохода на верных местах находится не менее i элементов «справа»

$$\{\boxed{5, 3}, 15, 7, 6, 2, 11, 13\}$$

$$\{3, \boxed{5, 15}, 7, 6, 2, 11, 13\}$$

$$\{3, 5, \boxed{15, 7}, 6, 2, 11, 13\}$$

$$\{3, 5, 7, \boxed{15, 6}, 2, 11, 13\}$$

$$\{3, 5, 7, 6, \boxed{15, 2}, 11, 13\}$$

$$\{3, 5, 7, 6, 2, \boxed{15, 11}, 13\}$$

$$\{3, 5, 7, 6, 2, 11, \boxed{15, 13}\}$$

$$\{3, 5, 7, 6, 2, 11, 13, \underline{15}\}$$

$$\{3, 5, 6, 2, 7, 11, \underline{13, 15}\}$$

$$\{3, 5, 2, 6, 7, \underline{11, 13, 15}\}$$

$$\{3, 2, 5, 6, \underline{7, 11, 13, 15}\}$$

$$\{2, 3, 5, \underline{6, 7, 11, 13, 15}\}$$

Сортировка пузырьком: особенности

- Крайне проста в реализации и понимании.
- Устойчива.
- Сложность в наилучшем случае $O(N)$.
- Сложность в наихудшем случае $O(N^2)$.
- Сортирует на месте.

Сортировка вставками

- Первый проход — нужно поместить самый лёгкий элемент на первую позицию.
- В i-м проходе ищется, куда поместить очередной a_i внутри левых i элементов.
- Элемент a_i помещается на место, сдвигая вправо остальные внутри области $0 \ldots i$.

Инвариант: после i-го прохода обеспечена упорядоченность левых i элементов.

Сортировка вставками

Инвариант сортировки вставками:

$$\underbrace{a_1, a_2, \ldots, a_{i-1}}_{a_1 \leqslant a_2 \leqslant \cdots \leqslant a_{i-1}}, a_i, \ldots, a_n$$

На шаге i имеется упорядоченный подмассив $a_1, a_2, \ldots, a_{i-1}$ и элемент a_i, который надо вставить в подмассив без потери упорядоченности.

Сортировка вставками

```
void insertion(int *a, int n) {
  for (int i = n-1; i > 0; i--) {
    if (a[i-1] > a[i]) {
      int tmp = a[i-1]; a[i-1] = a[i]; a[i] = tmp;
    }
  }
  for (int i = 2; i < n; i++) {
    int j = i;
    int tmp = a[i];
    while (tmp < a[j-1]) {
       a[j] = a[j-1]; j--;
    }
    a[j] = tmp;
  }
}
```

Сортировка вставками

Определение сложности.

- **Худший случай** — упорядоченный по убыванию массив. Тогда цикл вставки всегда будет доходить до 1-го элемента.
- Для вставки элемента a_i потребуется $i - 1$ итерация.
- Позиции ищутся для $N - 1$ элемента. Общее время

$$T(N) = \sum_{i=2} Nc(i-1) = \frac{cn(n-1)}{2} = O(N^2)$$

- **Лучший случай** — упорядоченный по возрастанию массив. $T(N) = O(N)$

Сортировка вставками: особенности

- Сортировка упорядоченного массива требует $O(N)$.
- Сложность в худшем случае $O(N^2)$.
- Алгоритм устойчив.
- Число дополнительных переменных не зависит от размера (*in-place*)
- Позволяет упорядочивать массив при динамическом добавлении новых элементов — *online*-алгоритм.

Сортировка Шелла

Помним, что один шаг сортировки пузырьком уменьшает инверсию на 1.

$$I(\{8,7,6,5,4,3,2,1\}) = \frac{8\cdot 7}{2} = 28$$

Может быть стоит обменивать элементы с расстоянием $d > 1$? Пусть $d = 4$.

$$I(\{4,7,6,5,8,3,2,1\} = 21$$

За один шаг инверсия уменьшилась на 7.

$$\{8,7,6,5,4,3,2,1\}$$
$$\{4,7,6,5,8,3,2,1\}$$
$$\{4,3,6,5,8,7,2,1\}$$
$$\{4,3,2,5,8,7,6,1\}$$
$$\{4,3,2,1,8,7,6,5\}$$

Сортировка Шелла

Второй проход: $d = 2$

$$\{4, 3, 2, 1, 8, 7, 6, 5\}$$

$$\{2, 3, 4, 1, 8, 7, 6, 5\}$$

$$\{2, 1, 4, 3, 8, 7, 6, 5\}$$

$$\{2, 1, 4, 3, 6, 7, 8, 5\}$$

$$\{2, 1, 4, 3, 6, 5, 8, 7\}$$

Сортировка Шелла

Третий проход: $d = 1$

$$\{2, 1, 4, 3, 6, 5, 8, 7\}$$

$$\{1, 2, 4, 3, 6, 5, 8, 7\}$$

$$\{1, 2, 3, 4, 6, 5, 8, 7\}$$

$$\{1, 2, 3, 4, 5, 6, 8, 7\}$$

$$\{1, 2, 3, 4, 6, 5, 7, 8\}$$

Voilá!

Сортировка Шелла (вариант Седжвика)

```
void shellsort(int  *a, int n) {
   int h;
   for (h = 1; h <= n / 9; h = 3*h + 1)
      ;
   for ( ; h > 0; h /= 3) {
      for (int i = h; i < n; i++) {
        int j = i;
        int tmp = a[i];
        while (j >= h && tmp < a[j-h]) {
          a[j] = a[j-h];
          j -= h;
        }
        a[j] = tmp;
      }
   }
}
```

Сортировка Шелла

Для массива размером $N = 100$ последовательность
$d = \{1, 4, 13, 40\}_{inv}$

Сложность зависит от последовательности d.

Для оригинальной последовательности худшая $= O(N^2)$

Для $d = \{1, 4, 13, \dots\}$ худшая $= O(N^{\frac{3}{2}})$

Для $d = \{1, 8, 23, 77, \dots, 4^{i+1} + 3 \cdot 2^i + 1, \ \dots\}$ худшая $= O(N^{\frac{4}{3}})$

Сортировка Шелла

Особенности:

- Сортировка упорядоченного массива требует $O(N)$.
- Алгоритм неустойчив.
- Число дополнительных переменных не зависит от размера (*in-place*).
- Конкурент популярным алгоритмам при не очень больших N.
- Низкий коэффициент амортизации.

Сортировка выбором

- Шаги нумеруем с нуля.
- В i−м шаге рассматривается область от i до $n-1$
- В области находится минимальный элемент на позиции j
- Элементы a_i и a_j меняются местами.

Инвариант: после i итераций упорядочены первые i элементов.

Сортировка выбором

$$\{5, 3, 15, 7, 6, \boxed{2}, 11, 13\}$$

$$\{\underline{2}, \boxed{3}, 15, 7, 6, 5, 11, 13\}$$

$$\{\underline{2, 3}, 15, 7, 6, \boxed{5}, 11, 13\}$$

$$\{\underline{2, 3, 5}, 7, \boxed{6}, 15, 11, 13\}$$

$$\{\underline{2, 3, 5, 6}, \boxed{7}, 15, 11, 13\}$$

$$\{\underline{2, 3, 5, 6, 7}, 15, \boxed{11}, 13\}$$

$$\{\underline{2, 3, 5, 6, 7, 11}, 15, \boxed{13}\}$$

$$\{\underline{2, 3, 5, 6, 7, 11, 13}, \boxed{15}\}$$

$$\{\underline{2, 3, 5, 6, 7, 11, 13, 15}\}$$

Сортировка выбором: особенности

- Во всех случаях сложность $O(N^2)$!
- Алгоритм неустойчив.
- Число дополнительных переменных не зависит от размера (*in-place*)
- Количество операций обмена $O(N)$ — может пригодиться для сортировок массивом с большими элементами.

Нахождение медианы

Нахождение медианы

Определение. $k-$й порядковой статистикой массива называется $k-$й по величине элемент массива.

- максимальный(минимальный) элемент массива — 1-я(N-я) порядковая статистика
- медиана — «средний» по величине элемент. Примерно половина элементов не больше, примерно половина не меньше. Это — не среднее значение!
  $\text{Median}(\{1, 1, 1, 1, 1, 10\}) = 1.0$
  $\text{Average}(\{1, 1, 1, 1, 1, 10\}) = 2.5$

Нахождение k−й порядковой статистики

Легко ли найти i−ю порядковую статистику?

- `i=1` Нахождение максимума — очевидно, что сложность $O(N)$.
- `i=2` Нахождение второго по величине элемента. Простой способ: хранить значения двух элементов, максимального и второго по величине. **Два** сравнения на каждой итерации.
- `i=3` Нахождение третьего по величине элемента. Простой способ: хранить значения трёх элементов, максимального, второго по величине, третьего по величине. **Три** сравнения на каждой итерации.
- `i=k` Требуется ли использовать $O(k)$ памяти и тратить $O(k)$ операций на одну итерацию?

Нахождение $k-$й порядковой статистики

Алгоритм нахождения $k-$й порядковой статистики методом *разделяй и властвуй*:

1. Выбираем случайным образом элемент v массива S
2. Разобьём массив на три подмассива S_l, элементы которого меньше, чем v; S_v, элементы которого равны v и S_r, элементы которого больше, чем v.
3. $$selection(S,k) = \begin{cases} selection(S_l, k), & \text{если } k \leqslant |S_l| \\ v, & \text{если } |S_l| < k \leqslant |S_l| + |S_v| \\ selection(S_r, k - |S_l| - |S_v|), & \text{если } k > |S_l| + |S_v| \end{cases}$$

Пример: Нахождение $k-$статистики.

Массив $S = \{10, 6, 14, 7, 3, 2, 18, 4, 5, 13, 6, 8\}$
Надо найти $k = 6$ статистику.
Первый проход: выбран произвольный элемент 8.

$$S_l = \{6, 7, 3, 2, 4, 5, 6\} \quad |S_l| = 7$$
$$S_v = \{8\} \quad |S_v| = 1$$
$$S_r = \{10, 14, 18, 13\} \quad |S_r| = 4$$

$k < |S_l| \rightarrow$ первый случай.

Пример: Нахождение $k-$статистики.

Второй проход: $S = \{6, 7, 3, 2, 4, 5, 6\}$

Выбран произвольный элемент 5.

$$\begin{aligned} S_l &= \{3, 2, 4\} & |S_l| = 3 \\ S_v &= \{5\} & |S_v| = 1 \\ S_r &= \{6, 7, 6\} & |S_r| = 3 \end{aligned}$$

$k > |S_l| + |S_v| \rightarrow$ третий случай.

Пример: Нахождение $k-$статистики.

Третий проход: $S = \{7, 6\}$

Выбран произвольный элемент 6.

$$\begin{aligned} S_l &= \{\} & |S_l| &= 0 \\ S_v &= \{6\} & |S_v| &= 1 \\ S_r &= \{7\} & |S_r| &= 1 \end{aligned}$$

$|S_l| < k \leqslant |S_l| + |S_v| \to$ второй случай.
Ответ: 6

Нахождение $k-$статистики. Сложность

Алгоритм типа *разделяй и властвуй* $\rightarrow$ работает основная теорема о рекурсии.

Идеальный случай — уменьшение в 2 раза.

Тогда

$$T(N) = T\left(\frac{N}{2}\right) + O(N)$$

- Количество подзадач $a = 1$
- Размер подзадачи $b = 2$
- Коэффициент $d = 1$.

$\log_b a = \log_2 1 = 0 < 1 \rightarrow T(N) = O(N)$.

Нахождение $k-$статистики. Сложность

Худший случай.
При выборе элемента, при котором каждый раз $|S_l| = 0$ или $|S_r| = 0$

$$T(N) = N + (N-1) + \cdots = \Theta(N^2)$$

Вероятность такого события $p = \prod\limits_{i=N,2} \frac{2}{i}$.
Пусть *хороший элемент* — такой, что его порядковый номер L в отсортированном массиве

$$\frac{1}{4}|S| \leqslant L \leqslant \frac{3}{4}|S|$$

Вероятность p элемента оказаться *хорошим* $p = \frac{1}{2}$.
Математическое ожидание количества испытаний для выпадения *хорошего* элемента $E = 2$.
Следовательно

$$T(N) \leqslant T\left(\frac{3}{4}N\right) + O(N) \to O(N)$$

Сортировка слиянием

Сортировка слиянием

- Алгоритмы нахождения $k-$статистики и быстрой сортировки основаны на *операции выборки* — нахождения $k-$го минимального элемента в массиве.
- *Слияние* — противоположная операция. Это — объединение отсортированных массивов.
- *Двухпутевое* слияние — объединение *двух* отсортированных массивов.
- *Декомпозиция* — разделение массива на подмассивы.

Сортировка слиянием

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

Декомпозиция 1: Массивы $S_l = \{10, 5, 14, 7, 3, 2\}$ и $S_r = \{18, 4, 5, 13, 6, 8\}$
(Рекурсивно) Сортировка $S_l : S_l = \{2, 3, 5, 7, 10, 14\}$
(Рекурсивно) Сортировка $S_r : S_r = \{4, 5, 6, 8, 13, 18\}$
Слияние 1: $S = \{2, 3, 4, 5, 6, 7, 8, 10, 13, 14, 18\}$

Сортировка слиянием

Псевдокод для алгоритма.

```
void mergeSort(int a[], int low, int high) {
   if (high - low < THRESHOLD) {
      plainSort(a, low, high);
   } else {
      int mid = (low + high) / 2;
      mergeSort(a, low, mid);
      mergeSort(a, mid+1, high);
      merge(a, low, mid ,high);
  }
}
```

Сложность сортировки слиянием

Принцип *разделяй и властвуй* — работает основная теорема о рекурсии.

$$T(N) = T\left(\left\lceil \frac{N}{2} \right\rceil\right) + T\left(\left\lfloor \frac{N}{2} \right\rfloor\right) + \Theta(N)$$

- Количество подзадач $a = 2$
- Размер подзадачи $b = 2$
- Коэффициент $d = 1$.
  $\log_b a = \log_2 2 = 1 \to T(N) = O(N \log N)$.

Сортировка слиянием: особенности

- Обычно требует добавочно $O(N)$ памяти.
- Сложность не зависит от входа и равна всегда $O(N \log N)$.
- Прекрасно подходит для *внешней* сортировки.

Быстрая сортировка

Быстрая сортировка

Алгоритм почти повторяет алгоритм поиска медианы.

1. Из элементов **выбирается *ведущий*** (*pivot*). Чем он ближе, к медиане, тем лучше!
2. **Массив разбивается на два**. Левая часть — элементы не больше ведущего, правая часть — элементы, не меньше ведущего.
3. **Рекурсивно** повторяются шаги 1 и 2 для обоих частей.

Левая и правая части остаются внутри массива!

Быстрая сортировка

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

- Разделение 1. Пусть ведущий элемент $= 8$.

$$S_1 l = \{5, 7, 3, 2, 4, 5, 6, 8\}, S_1 r = \{10, 14, 18, 13\}$$

$$S_1 = \{\underbrace{5, 7, 3, 2, 4, 5, 6, 8}, \underbrace{10, 14, 18, 13}\}$$

- Рекурсивное разделение 2. Пусть ведущий элемент $= 5$ $S_1 = \{5, 7, 3, 2, 4, 5, 6, 8\}$

$$S_2 l = \{5, 3, 2, 4, 5\}, S_2 r = \{7, 6, 8\}$$

$$S_2 = \{\underbrace{5, 3, 2, 4, 5}, \underbrace{7, 6, 8}\}$$

Быстрая сортировка

Рассуждения заставляют вспомнить поиск k−й статистики и сортировку слиянием.
Лучший случай: выбирается медианный элемент.

$$T(N) = T\left(\left\lceil \frac{N}{2} \right\rceil\right) + T\left(\left\lfloor \frac{N}{2} \right\rfloor\right) + \Theta(N)$$

- Количество подзадач $a = 2$
- Размер подзадачи $b = 2$
- Коэффициент $d = 1$.
  $\log_b a = \log_2 2 = 1 \to T(N) = O(N \log N)$.

Быстрая сортировка

- Худший случай: ведущим выбирается минимальный или максимальный элемент.
- Вероятность такого события при условии случайного выбора равна

$$p = \frac{2}{N} \cdot \frac{2}{N-1} \cdot \dots \cdot \frac{2}{3} = \frac{2^{N-1}}{N!}$$

- При $N = 10$ $p = 1.4 \times 10^{-4}$, при $N = 20$ $p = 1.1 \times 10^{-13}$
- Один из способов: ведущий элемент есть медиана из трёх случайных чисел.

Быстрая сортировка: особенности

- Может проводиться на месте.
- Сложность в наихудшем случае $O(N^2)$, но с крайне малой вероятностью.
- Сложность в среднем $O(N \log N)$
- В прямолинейной реализации использует до $O(N)$ стека.

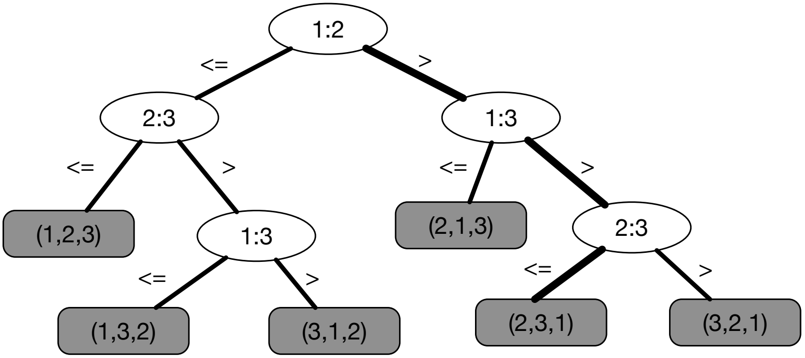
Нижняя оценка сложности алгоритмов сортировки сравнениями

- Рассмотрено много 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 $\to$ 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 $H \sim N \log N$.

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.

$F =$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

$F =$

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	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 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**, имеющая следующие методы:
- create/destroy
- open/close/rewind
- getdata/putdata

Сортировка больших данных

- Одним из хороших способов отсортировать внешние данные является использование операции *слияние*.
- Входными данными *двухпутевого слияния* являются две отсортированные ленты.
- Выходные данные — другая отсортированная лента.
- Чанк (chunk) — фрагмент данных, помещающихся в оперативной памяти.

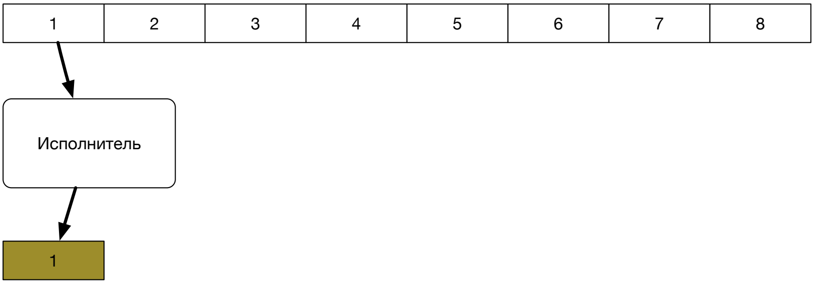
Сортировка слиянием

Пусть исходная лента содержит 8 чанков.

1	2	3	4	5	6	7	8

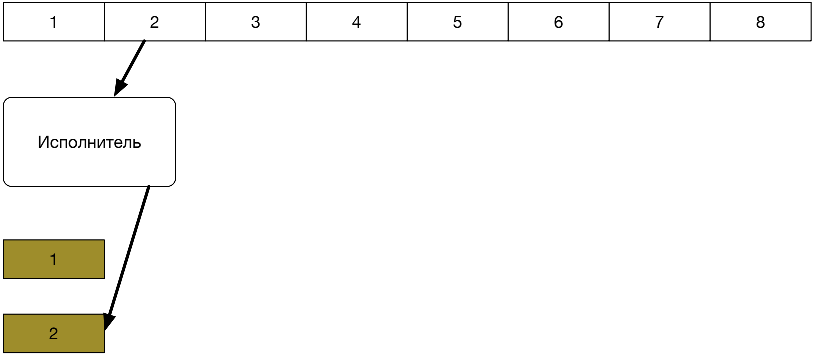
Сортировка слиянием

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



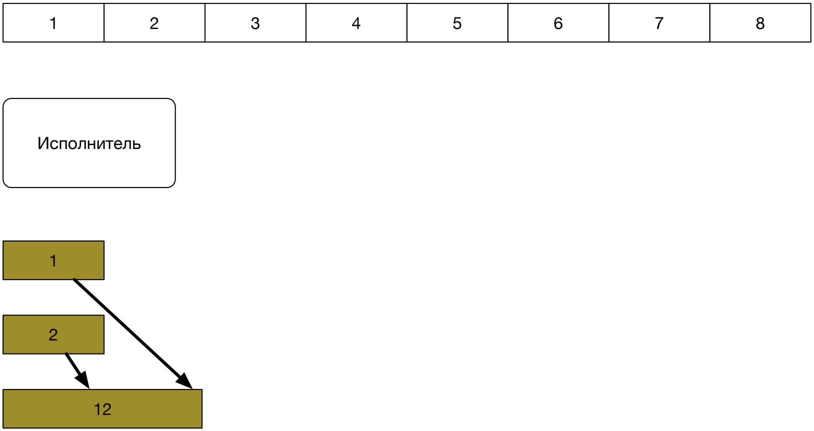
Сортировка слиянием

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



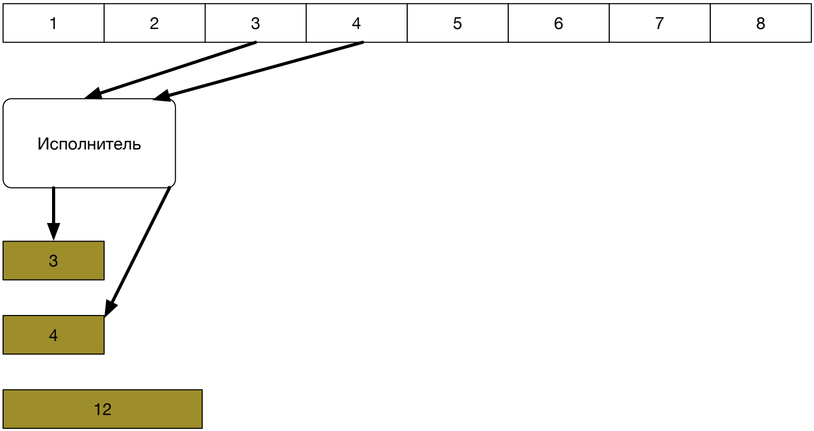
Сортировка слиянием

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



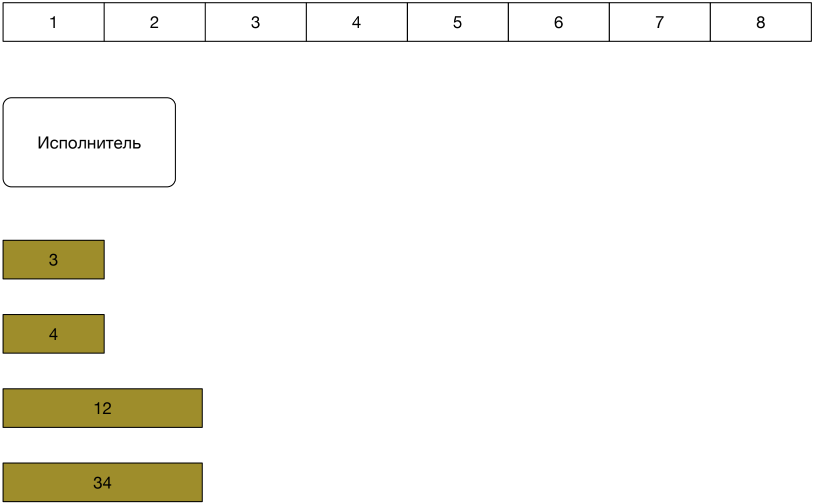
Сортировка слиянием

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



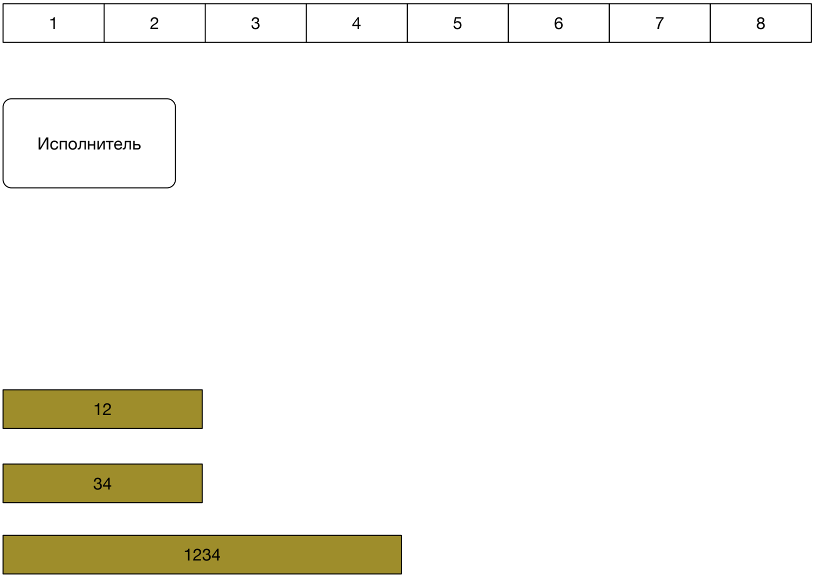
Сортировка слиянием

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



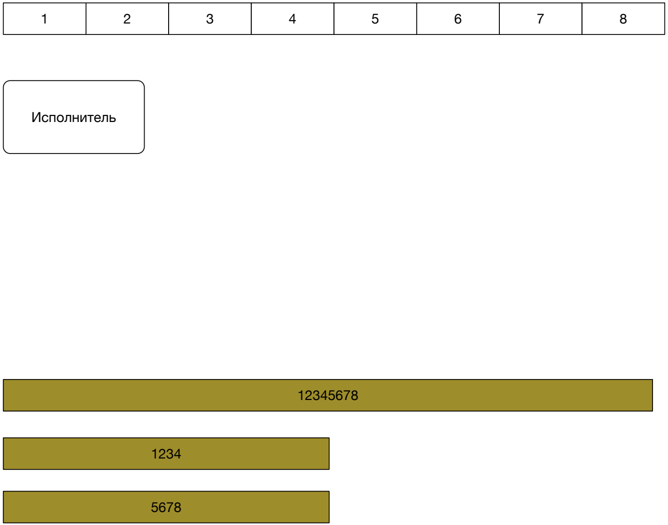
Сортировка слиянием

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

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

- 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.

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

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 (использовав первую идею ленточной).

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, а их может быть несколько.
- Если обозначить за $t(x)$ абсолютное время окончания 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.

Сравнительный анализ методов сортировки

Алгоритм	Лучший случай	Средний случай	Худший случай	Память	Устойчива?
Пузырьком	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Шелла	$O(N^{\frac{7}{6}})$	$O(N^{\frac{7}{6}})$	$O(N^{\frac{4}{3}})$	$O(1)$	Нет
Вставками	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Выбором	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да
Быстрая	$O(N \log N)$	$O(N \log N)$	$O(N^2)$	$O(1)$	Да/Нет
Слиянием	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$	Да/Нет
Heap	$O(N)$	$O(N \log N)$	$O(N \log N)$	$O(1)$	Нет
Подсчётом	$O(N)$	$O(N)$	$O(N)$	$O(N)$	Да
Поразрядная	$O(N)$	$O(N)$	$O(N)$	$O(N)$	Да/Нет

Спасибо за внимание.

Следующая лекция —
поиск