

# Алгоритмы и структуры данных

## Лекция 4

### Сортировка

Сергей Леонидович Бабичев

## План лекции

- 1 Задача сортировки
- 2 Сортировки сравнением
- 3 Нахождение медианы множества
- 4 Быстрая сортировка
- 5 Сортировки с использованием свойств элементов
- 6 Сравнительный анализ методов сортировки

# Задача сортировки

# Задача сортировки

Имеется последовательность из  $n$  ключей.

$$k_1, k_2, \dots, k_n$$

Требуется: упорядочить ключи по *не убыванию* или *не возрастанию*.

Это означает: найти перестановку ключей

$$p_1, p_2, \dots, p_n$$

такую, что

$$k_{p_1} \leq k_{p_2} \leq \dots \leq k_{p_n}$$

или

$$k_{p_1} \geq k_{p_2} \geq \dots \geq k_{p_n}$$

# 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

$$\overline{(a < b)} \wedge \overline{(b < a)} \rightarrow a = b$$

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 *(const int *)el1 - *(const int *)el2;  
}
```

# Сортировка в языке Си

```
#include <stdlib.h>

...
int a[100];
...
qsort(a, 100, sizeof a[0], cmp_int);
// Here a is sorted in ascending order
```

# 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); // Allowed for plain arrays
std::sort(pvect.begin(), pvect.end()); // And for vectors
```

## 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* (*closures*) C++. Другое название — *лямбда-функции*.

```
#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 количество инверсий ровно на 1.
- Количество инверсий в любом множестве конечно, в отсортированном — равно нулю.
- 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.

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

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

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

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

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

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



# 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]) {
                std::swap(a[i],a[i+1]); // C++!
                sorted = false;
            }
        }
        n--;
    }
}
```

## Сортировка пузырьком: инвариант

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

{5, 3, 15, 7, 6, 2, 11, 13}

{3, 5, 15, 7, 6, 2, 11, 13}

{3, 5, 15, 7, 6, 2, 11, 13}

{3, 5, 7, 15, 6, 2, 11, 13}

{3, 5, 7, 6, 15, 2, 11, 13}

{3, 5, 7, 6, 2, 15, 11, 13}

{3, 5, 7, 6, 2, 11, 15, 13}

{3, 5, 7, 6, 2, 11, 13, 15}

{3, 5, 6, 2, 7, 11, 13, 15}

{3, 5, 2, 6, 7, 11, 13, 15}

{3, 2, 5, 6, 7, 11, 13, 15}

{2, 3, 5, 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  $0 \dots i$ .

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

# Сортировка вставками

Инвариант сортировки вставками:

$$\underbrace{a_1, a_2, \dots, a_{i-1}}_{a_1 \leq a_2 \leq \dots \leq a_{i-1}}, a_i, \dots, a_n$$

На шаге  $i$  имеется упорядоченный подмассив  $a_1, a_2, \dots, a_{i-1}$  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]) {
            std::swap(a[i-1]; a[i]);
        }
    }
    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

$$T(N) = \sum_{i=2}^N c(i - 1) = \frac{cN(N - 1)}{2} = O(N^2).$$

- **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.

## Сортировки Шелла и comb

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.

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

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

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

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

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

# Сортировки Шелла и comb

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}

# Сортировки Шелла и comb

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, 5, 6, 7, 8}

# Сортировка comb

```
void combsort(int *a, int n) {  
    double s = n - 1;  
    while (s >= 1) {  
        int d = s;  
        for (int i = d; i < n; i++) {  
            if (a[i-d] > a[i]) {  
                std::swap(a[i-d], a[i]);  
            }  
        }  
        s /= 1.24733;  
    }  
}
```

# 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 =  $O(N^{\frac{4}{3}})$

# Сортировки Шелла и comb

Особенности:

- Сортировка упорядоченного массива требует  $O(N)$ .
- Алгоритм неустойчив.
- Число дополнительных переменных не зависит от размера (*in-place*).
- Конкуренты 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, **2**, 11, 13}

{ 2, **3**, 15, 7, 6, 5, 11, 13}

{ 2, 3, 15, 7, 6, **5**, 11, 13}

{ 2, 3, 5, 7, **6**, 15, 11, 13}

{ 2, 3, 5, 6, **7**, 15, 11, 13}

{ 2, 3, 5, 6, 7, 15, **11**, 13}

{ 2, 3, 5, 6, 7, 11, 15, **13**}

{ 2, 3, 5, 6, 7, 11, 13, **15**}

{ 2, 3, 5, 6, 7, 11, 13, 15}

# Сортировка выбором: особенности

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

# Сортировка слиянием

# Сортировка слиянием

- *Слияние* — объединение отсортированных массивов.
- Сложность операции слияния —  $\Theta(N_1 + N_2)$ .
- *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  $d = 1$ .  
 $\log_b a = \log_2 2 = 1 \rightarrow T(N) = \Theta(N \log N)$ .



# Сортировка слиянием: особенности

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

# 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.  $O(2N)$

$i=3$  Нахождение третьего по величине элемента. Простой способ: хранить значения трёх элементов, максимального, второго по величине, третьего по величине. **Три** сравнения на каждой итерации.  $O(3N)$

$i=k$  Требуется ли использовать  $O(k)$  памяти и тратить  $O(kN)$  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

$$\textit{selection}(S, k) = \begin{cases} \textit{selection}(S_l, k), & \text{если } k \leq |S_l| \\ v, & \text{если } |S_l| < k \leq |S_l| + |S_v| \\ \textit{selection}(S_r, k - |S_l| - |S_v|), & \text{если } k > |S_l| + |S_v| \end{cases}$$

## 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.

$$S_l = \{3, 2, 4\} \quad |S_l| = 3$$

$$S_v = \{5\} \quad |S_v| = 1$$

$$S_r = \{6, 7, 6\} \quad |S_r| = 3$$

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

## Пример: Нахождение $k$ -статистики.

Третий проход:  $S = \{6, 7, 6\}$

Выбран произвольный элемент 6.

$$\begin{aligned}S_l &= \{\} & |S_l| &= 0 \\S_v &= \{6, 6\} & |S_v| &= 2 \\S_r &= \{7\} & |S_r| &= 1\end{aligned}$$

$|S_l| < k \leq |S_l| + |S_v| \rightarrow$  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 каждый раз  $|S_l| = 0$  или  $|S_r| = 0$

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

Вероятность такого события  $p = \prod_{i=N,2} \frac{2}{i}$ .

Пусть *хороший элемент* — такой, что его порядковый номер  $L$  в отсортированном массиве

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

Вероятность  $p$  элемента оказаться *хорошим*  $p = \frac{1}{2}$ .

Математическое ожидание количества испытаний для выпадения *хорошего* элемента  $E = 2$ .

Следовательно

$$T(N) \leq T\left(\frac{3}{4}N\right) + O(N) \rightarrow O(N)$$

# 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 обеих 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.

$$S_{1l} = \{5, 7, 3, 2, 4, 5, 6, 8\}, S_{1r} = \{10, 14, 18, 13\}$$

$$S_1 = \underbrace{\{5, 7, 3, 2, 4, 5, 6, 8\}}_{\text{left part}}, \underbrace{\{10, 14, 18, 13\}}_{\text{right part}}$$

- Рекурсивное разделение 2. Пусть ведущий элемент = 5

$$S_1 = \{5, 7, 3, 2, 4, 5, 6, 8\}$$

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

$$S_2 = \underbrace{\{5, 3, 2, 4, 5\}}_{\text{left part}}, \underbrace{\{7, 6, 8\}}_{\text{right part}}$$

# 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  $d = 1$ .  
 $\log_b a = \log_2 2 = 1 \rightarrow T(N) = O(N \log N)$ .

# Быстрая сортировка

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

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

- 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 элементов массива.

# 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.



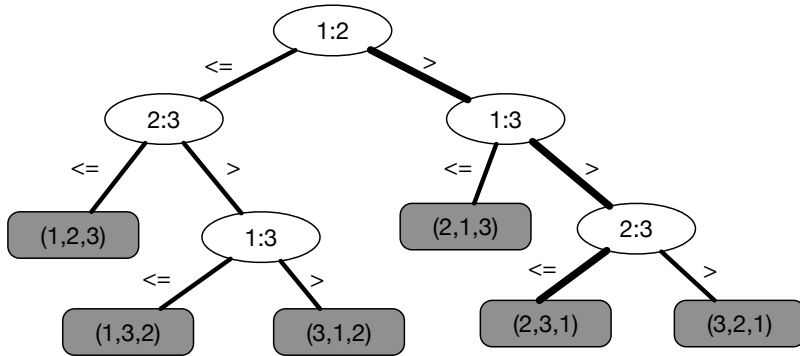
# Нижняя оценка сложности алгоритмов сортировки сравнениями

- Рассмотрено 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  $1 \leq i, j \leq N$ .
- Каждая терминальная вершина содержит окончательное решение задачи в виде одной из перестановок множества  $\{1, 2, \dots, N\}$ .
- 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  $H \geq \log_2 V$

Формула Стирлинга:

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

Отсюда  $H \sim N \log N$ .

Спасибо за внимание.

Следующая лекция —  
сортировка и поиск.