

ДВОИЧНЫЙ ПОИСК В УПОРЯДОЧЕННОМ МАССИВЕ



Пусть дан массив $A = (a_1, a_2, \dots, a_n)$,
упорядоченный по возрастанию, т.е. $a_1 \leq a_2 \leq \dots \leq a_n$.

Найти: 1) Элемент с ключом X ;
2) Все элементы с ключом X .

Если массив не упорядочен, то единственный способ поиска – перебор, трудоемкость $O(n)$.



www.holt-doll.ru

В случае быстрого двоичного поиска трудоемкость $O(\log_2 n)$.



Идея двоичного поиска: Возьмем **средний элемент упорядоченного массива** и сравним с ключом поиска «X». Возможны варианты:

- 1) $a_m = X$ элемент найден
- 2) $a_m < X$ продолжаем поиск в **правой** половине массива
- 3) $a_m > X$ продолжаем поиск в **левой** половине массива

Каким образом?

Алгоритм на псевдокоде (первая версия)

Обозначим

L, R – правая и левая границы рабочей части массива,

Найден – логическая переменная, в которой будем отмечать факт успешного завершения поиска.

L: = 1, R: = n, Найден: = нет

DO (L ≤ R)

m: = $\lfloor (L+R)/2 \rfloor$

IF (a_m = X) Найден: = да OD FI

IF (a_m < X) L: = m+1

ELSE R: = m-1

FI

OD

1 2 3 4 5 6 7 8 9 10 11 12
а б б б в г д е ж з и к X=б

L=1, R=12

1 2 3 4 5
а б б б в

L=1, R=5

Недостатки первой версии алгоритма:

- 1) на каждой итерации цикла **два сравнения**,
- 2) находит **первый попавшийся** элемент из нескольких с заданным ключом.

Рассмотрим **вторую версию** алгоритма,
в которой
уменьшим количество сравнений
путем **исключения из алгоритма**
проверки на равенство.

Алгоритм на псевдокоде (вторая версия)

L: = 1, R: = n

DO (L < R)

m: = $\lfloor (L+R)/2 \rfloor$

IF ($a_m < X$) L: = m+1

ELSE R: = m

FI

OD

IF ($a_R = X$) Найден: = да

ELSE Найден: = нет

FI

1 2 3 4 5 6 7 8 9 10 11 12
а б б б в_г д е ж з и к
X=б

L=1, R=12

1 2 3 4 5 6
а б б б в г

L=1, R=6

1 2 3
а б б

L=1, R=3

1 2
а б

L=1, R=2

2
б

L=2, R=2

Преимущества второй версии алгоритма:

1) на каждой итерации цикла **одно сравнение**,

Трудоёмкость двоичного поиска

Сначала определим

максимальное количество итераций (k).

Рассмотрим **худший случай**, когда

1) часть массива a_L, \dots, a_R содержит **нечетное** количество элементов

2) в начале каждой итерации

слева элементов на один больше

3) на каждом шаге выбирается **левая часть** массива.

Трудоёмкость двоичного поиска

Номер итерации

Наибольшее количество
элементов

0

n - нечетное

1

$$\frac{n}{2} + \frac{1}{2} = \frac{n+1}{2}$$

2

$$\frac{n+1}{4} + \frac{1}{2} = \frac{n+3}{4}$$

3

$$\frac{n+3}{8} + \frac{1}{2} = \frac{n+7}{8}$$

...

...

$k-1$

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

$$\frac{n+2^{k-1}}{2^{k-1}} > 2$$

$$\frac{n}{2^{k-1}} + 1 > 2$$

$$2^{k-1} < n$$

$$k-1 < \log_2 n$$

$$k < \log_2 n + 1$$

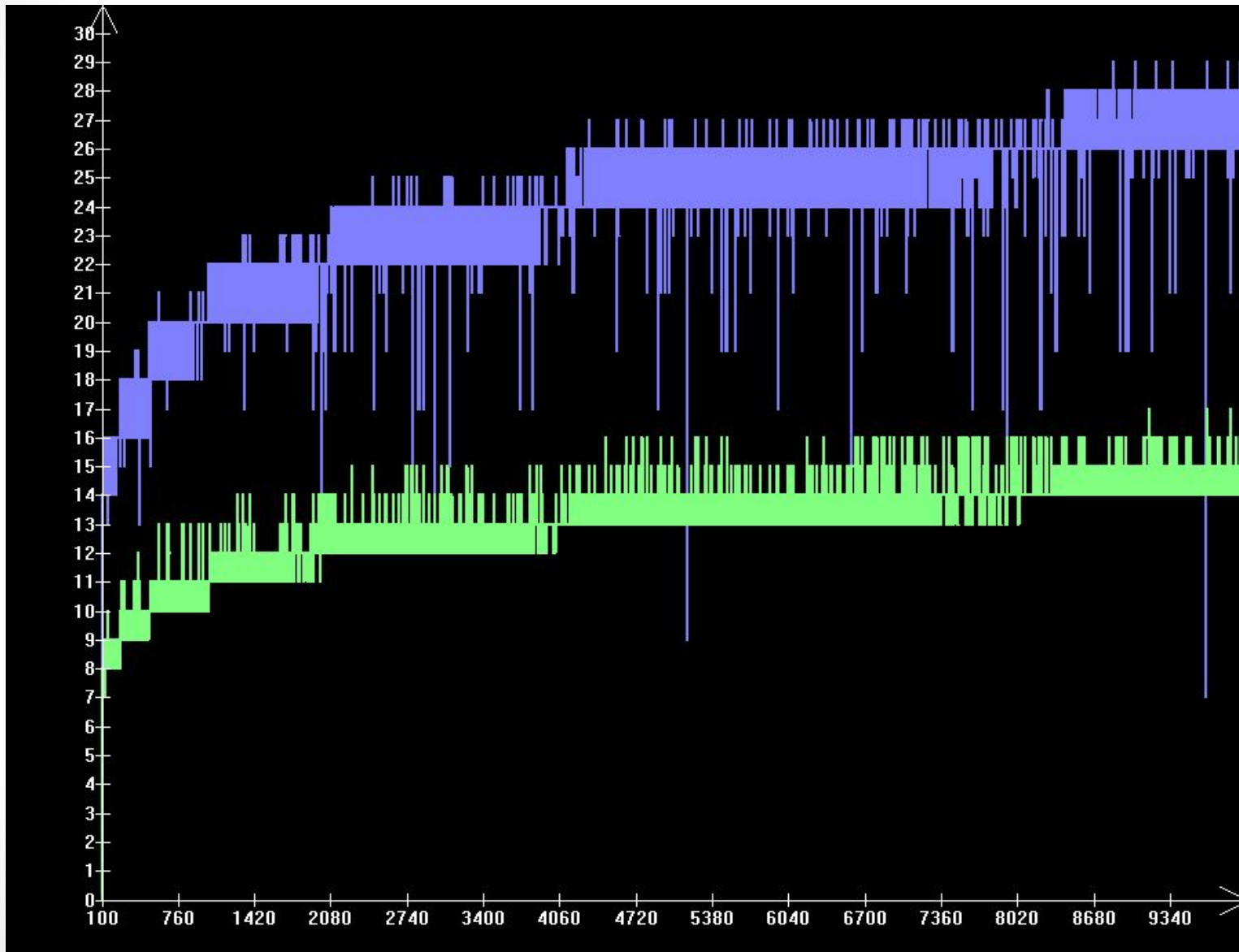
$k \leq \lceil \log_2 n \rceil$ - количество итераций

$$C \leq \lceil \log_2 n \rceil + 1$$

Трудоёмкость двоичного поиска:

$$T = O(\log_2 n)$$

Графики трудоемкости двоичного поиска



Сортировка данных со сложной структурой

Дан массив абонентов

A:	<table border="1"><tr><td>Иванов</td></tr><tr><td>223322</td></tr></table>	Иванов	223322	<table border="1"><tr><td>Петров</td></tr><tr><td>345767</td></tr></table>	Петров	345767	<table border="1"><tr><td>Абрамов</td></tr><tr><td>667891</td></tr></table>	Абрамов	667891
Иванов									
223322									
Петров									
345767									
Абрамов									
667891									

```
Struct abonent { char name[10];  
                long phone;  
                } A[n];
```

Чтобы отсортировать такой массив структур, нужно **определить отношение порядка** его элементов ($>$ $<$ $=$).

Сортировка данных со сложной структурой

Пример. Struct **abonent** { char name[10];
long phone;
} A[n];

Попытка сортировки:

```
DO ( i = 1, 2, ..., n-1 )
    DO ( j = n, n-1, ..., i+1 )
        IF ( Aj < Aj-1 ) Aj ↔ Aj-1 FI
    OD
OD
```

Эта запись **не будет верной**, т.к. компилятор не знает как сравнивать **элементы типа структура**, т.к. они **не являются встроенными элементами языка**.

Чтобы реализовать **операцию сравнения для структур**, необходимо вспомнить, что любая **операция отношения** есть **булева функция двух аргументов**.

$$X < Y$$

$$\text{меньше}(X, Y) = \begin{cases} \text{истина, } X < Y \\ \text{ложь, } X \geq Y \end{cases}$$

Логическая функция Less (меньше)

может выглядеть следующим образом:

```
Int less ( struct abonent X, struct abonent Y)
```

```
{ ... ? ... }
```

Логическая функция **Less** (меньше)

При сортировке по **имени** абонента:

```
int less ( struct abonent X, struct abonent Y)
{ if ( X.name<Y.name) return 1;
  else return 0;
}
```

При сортировке по **номеру телефона** абонента:

```
int less ( struct abonent X, struct abonent Y)
{ if ( X.phone<Y.phone) return 1;
  else return 0;
}
```


Наполовину пуст? Наполовину полон?



Программист считает, что
стакан в два раза больше, чем
нужно

При сортировке **по сложному ключу** так же легко определить функцию `less`.

Для сортировки по фамилии абонента и (дополнительно) по номеру телефона:

```
int less ( struct abonent X, struct abonent Y)
{ if ( X.name < Y.name) return 1;
  else if ( X.name > Y.name) return 0;
    else if ( X.phone < Y.phone) return 1;
      else return 0;
}
```

*Тогда в алгоритмах сортировок вместо оператора сравнения используем вызов функции **less**.*

Например, в пузырьковой сортировке:

```
DO ( i = 1, 2, ..., n-1)
```

```
    DO ( j = n, n-1, ..., i+1)
```

```
        IF ( less ( Aj, Aj-1 ) ) Aj ↔ Aj-1 FI
```

```
    OD
```

```
OD
```

Вывод:

Если структура сортируемых данных
не соответствует



простым (встроенным) типам языка, то

**операции отношения необходимо
переопределить**

с помощью соответствующих **булевых функций**.

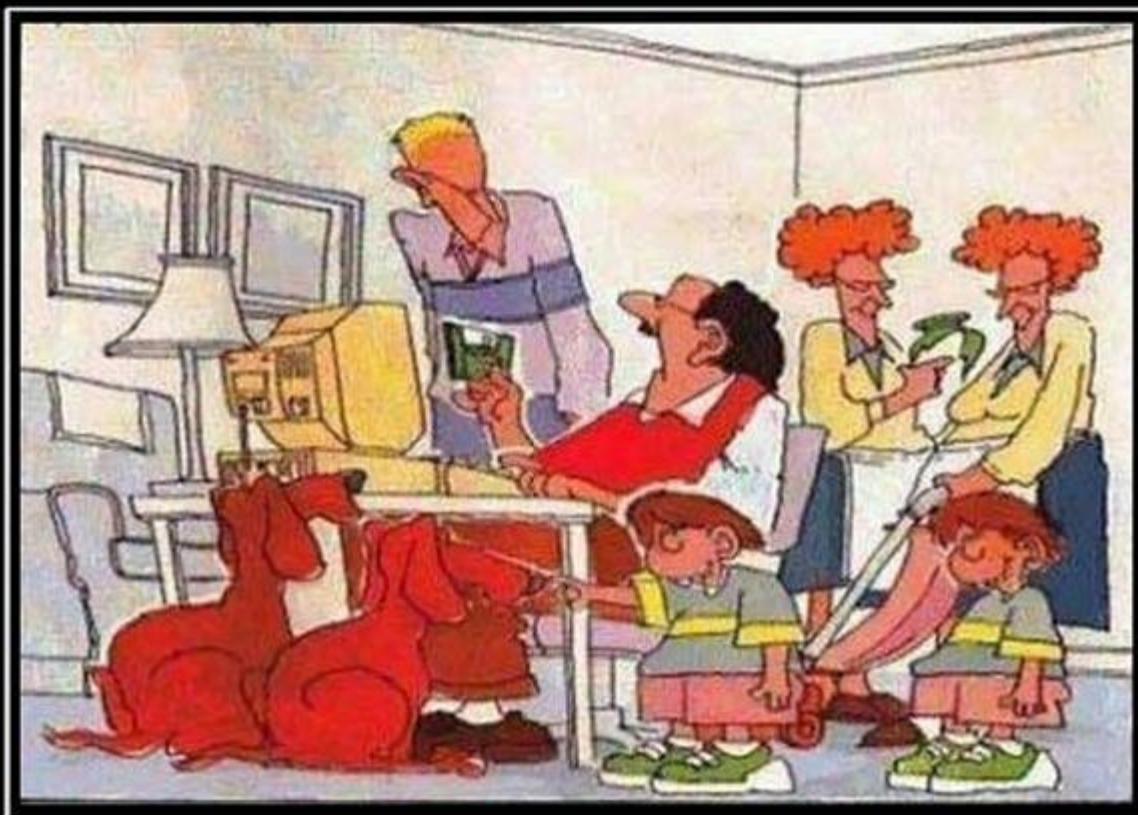
**Аналогичное переопределение операций
сравнений требуется и для организации
поиска!**

Преимущества:

- 1) **Операции отношения** могут быть определены *различными способами* в зависимости от **ключа сортировки** и **условия упорядочения** данных.
- 2) **Изменение направления упорядочения** массива легко достигается с помощью **изменения знака** в операции отношения на **противоположный**.

**Операции пересылки не требуют
переопределения,**

т.к. выполняются путем **побайтового**



Настоящий программист всегда
делает резервные копии

Сортировка по множеству ключей

Пусть рассмотренный **телефонный справочник** хранится в виде **базы данных** в памяти компьютера и мы хотим использовать его для **эффективного решения двух задач**:

1) Быстро искать запись **по заданной фамилии** (справочник должен быть отсортирован по фамилиям абонентов);

2) Быстро искать запись **по заданному номеру телефона** (справочник должен быть отсортирован по номерам телефонов абонентов);

Для одновременного решения этих задач рассмотрим прием, называемый **индексацией**.

Индексация данных

Рассмотрим суть индексации на массиве целых чисел:

	1	2	3	4	5	6	7	8	- физические номера
A:	5	7	3	4	2	6	1	8	
B:	7	5	3	4	1	6	2	8	
	1	2	3	4	5	6	7	8	- логические номера

Массив **B** - **индексный массив** (индекс) массива **A**.

$A [B[i]]$ – обращение к элементу массива **A**
через индекс **B**.

C: **8 2 6 1 4 3 5 7** - номера элементов
массива **A** по убыванию

Массив **C** – **индексный массив** (индекс) массива **A**.

$A [C[i]]$ – обращение к элементу массива **A**
через индекс **C**.

Чтобы упорядочить массив A (по возрастанию),
мы построили индексный массив B ,
в него записали номера элементов массива
 A (по возрастанию элементов) и
**обращаемся к элементам массива A
через индекс B .**

При доступе к массиву A через индекс
мы работаем с ним **как с упорядоченным**
(например, можем производить **быстрый
двоичный поиск**), в то время как
сами **элементы физически не
переставляются**

Пример. Вывод элементов массива (по возрастанию):

```
DO ( i = 1, ..., n)
    ВЫВОД ( A [ B[i] ] )
OD
```

Пример. Двоичный поиск (вторая версия):

```
L: = 1, R: = n
DO ( L < R )
    m: =  $\lfloor (L+R)/2 \rfloor$ 
    IF (A[B[m]] < X) L: = m+1
        ELSE R: = m
    FI
OD
IF (A[B[R]] = X) Найден: = да
ELSE Найден: = нет
FI
```



Построение индексного массива

Построение индексного массива выполняется на базе любого **алгоритма сортировки**.

* Вначале в массив В записываются **физические номера элементов массива А**.

* Затем производится **любая сортировка** *при условии*, что:

1) В **операциях сравнения** элементы массива А индексируются через В;

2) **Перестановки** делаются **только в массиве В**;

Построение индексного массива

Алгоритм на псевдокоде

(на примере пузырьковой сортировки)

$B := (1, 2, \dots, n)$

DO ($i = 1, 2, \dots, n-1$)

DO ($j = n, n-1, \dots, i+1$)

IF ($a[b_j] < a[b_{j-1}]$) $b_j \leftrightarrow b_{j-1}$ FI

OD

OD

Преимущества индексации

1) Появляется возможность построения **нескольких различных индексов**, которые можно использовать по мере необходимости.

2) **Исключается копирование**

информации о базисах данных.

необходимость

индексации данных.



Преимущества индексации

Определение. **Фильтрация** – использование при работе только тех элементов, которые отвечают некоторым условиям.

Совокупность условий называется **фильтром**.

Пример. Из массива A выбираем только четные элементы по возрастанию.

	1	2	3	4	5	6	7	8
A:	5	7	3	4	2	6	1	8

D: 5 4 6 8 - только четные, по возрастанию