<https://prog-cpp.ru/category/sort-posts/>

<https://top-technologies.ru/ru/article/view?id=24620>

**Поиск** — обработка некоторого множества данных с целью выявления подмножества данных, соответствующего критериям поиска.

**Критерий поиска** или **ключ** – это та часть элемента данных, которая идентифицирует его и используется для поиска. Остальная часть несет в себе содержательную информацию, которая извлекается и используется из найденного элемента данных.

Все алгоритмы поиска делятся на

* поиск в неупорядоченном множестве данных;
* поиск в упорядоченном множестве данных.

**Задача поиска***:* найти один или несколько элементов во множестве, причем искомые элементы должны обладать определенным свойством – *абсолютным* (поиск по конкретному значению) или *относительным* (например, минимальный элемент в множестве чисел).

Таким образом, в задаче поиска имеются следующие шаги:

1) вычисление свойства элемента; часто это - просто получение «значения» элемента, ключа элемента и т. д.;

2) сравнение свойства элемента с эталонным свойством (для абсолютных свойств) или сравнение свойств двух элементов (для относительных свойств);

3) перебор элементов множества, т. е. прохождение по элементам множества.

**Линейный (последовательный) поиск**

<http://ermak.cs.nstu.ru/cprog/html/046.htm>

Если данные не упорядочены, то найти что-либо нас интересующее можно только путем последовательного перебора всех элементов.

Поиск значения путем последовательного перебора всех элементов называется **линейным поиском.**

Его трудоемкость в среднем **Tср(N)=N/2 =>****O(N).**По статистике надо просмотреть половину последовательности (по «закону подлости» - всю).

**Бинарный поиск**

<https://prog-cpp.ru/search-binary/>

Бинарный поиск производится в упорядоченном массиве.  
  
При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива.  
  
Алгоритм может быть определен в рекурсивной и нерекурсивной формах.  
  
Бинарный поиск также называют поиском методом ***деления отрезка пополам*** или ***дихотомии***.  
  
Количество шагов поиска определится как

**log2n↑**,

где **n**-количество элементов,  
**↑** — округление в большую сторону до ближайшего целого числа.

Алгортим следующий:

1. Поиск на каждом шаге цикла середины отрезка по формуле

**mid = (left + right)/2**,

где left – индекс левой границы массива

right – индекс правой границы массива

mid – индекс середины границы массива, округленный либо в большую сторону, либо в меньшую

1. Если искомый элемент равен элементу с индексом **mid**, поиск завершается.
2. В случае если искомый элемент меньше элемента с индексом **mid**: **right** = **mid** -1, иначе **left** = **mid** + 1.

*Условие выхода* из цикла: left > right.

**Распределяющий поиск**

<http://intuit.valrkl.ru/course-1215/index.html#ID.12.section.3>

Предположим, что значения ключей представляют собой различные небольшие числа. В этом случае простейший алгоритм поиска основывается на хранении элементов в массиве, индексированном значениями.

В данной реализации предполагается, что значения ключей - положительные целые числа, меньшие сигнального значения M - используются в качестве индексов массива. Конструктор Item создает элементы со значениями ключей, равными сигнальному значению, чтобы конструктор ST мог найти в пустом элементе значение M. Основные затраты этого метода - объем памяти, необходимый при большом размере сигнального значения, и время, необходимое конструктору ST, когда значение N мало по сравнению с M.

template <class Item, class Key>

class ST

{ private:

Item nullItem, \*st;

int M;

public:

ST(int maxN)

{ M = nullItem.key(); st = new Item[M]; }

int count()

{ int N = 0;

for (int i = 0; i < M; i++)

if (!st[i].null()) N+ + ;

return N;

}

void insert(Item x)

{ st[x.key()] = x; }

Item search(Key v)

{ return st[v]; }

void remove(Item x)

{ st[x.key()] = nullItem; }

Item select(int k)

{ for (int i = 0; i < M; i++)

if (!st[i].null())

if (k- == 0) return st[i];

return nullItem;

}

void show(ostream& os)

{ for (int i = 0; i < M; i++)

if (!st[i].null()) st[i].show(os); }

};