Разработка на c++

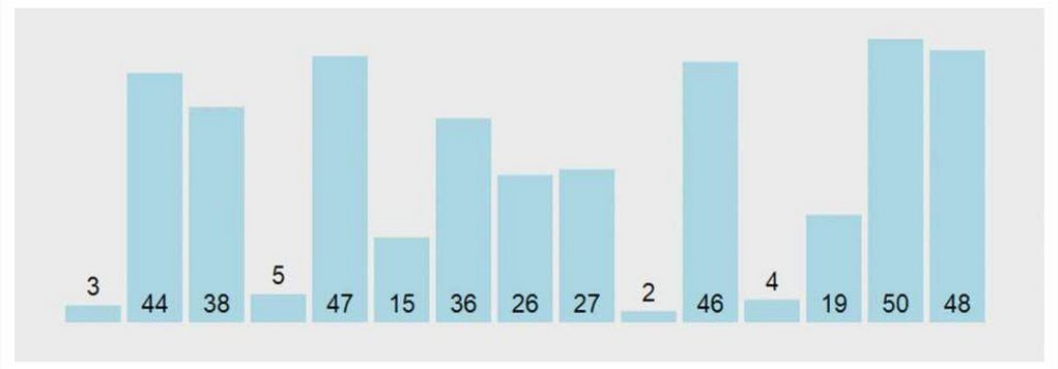
Алгоритмы

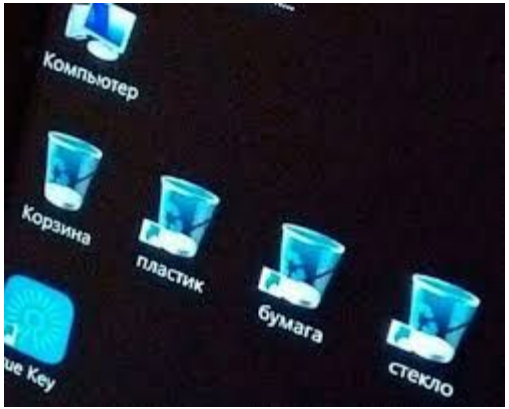
План

1. Что такое сортировка и зачем она нужна?
2. Какие есть популярные алгоритмы сортировки?
3. Что такое компаратор?
4. Перегрузка операторов сравнения и метод sort
5. Метод swap
6. Что такое поиск и зачем он нужен?
7. Популярные алгоритмы поиска
8. Алгоритмическая сложность

Что такое сортировка и зачем она нужна?

Это **упорядочивание данных** по некоторым признакам. В случае, когда элемент в массиве имеет несколько полей, поле, служащее критерием порядка, называется **ключом сортировки**.



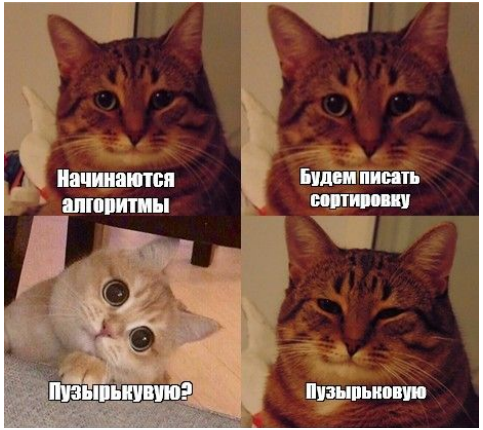


Какие есть популярные алгоритмы сортировки?

- Пузырьковая сортировка (Bubble sort)
- Сортировка выбором (Selection sort)
- Сортировка вставками (Insertion sort)
- Быстрая сортировка (Quick sort)
- Сортировка слиянием (Merge sort)
- Сортировка Шелла (Shell sort)
- Сортировка кучей(пирамидальная) (Heap sort)

Пузырьковая сортировка

С помощью сравнений i и i+1 элемента, на каждом шаге сдвигаем максимальный элемент в конец. Когда дошли до конца начинаем все с начала



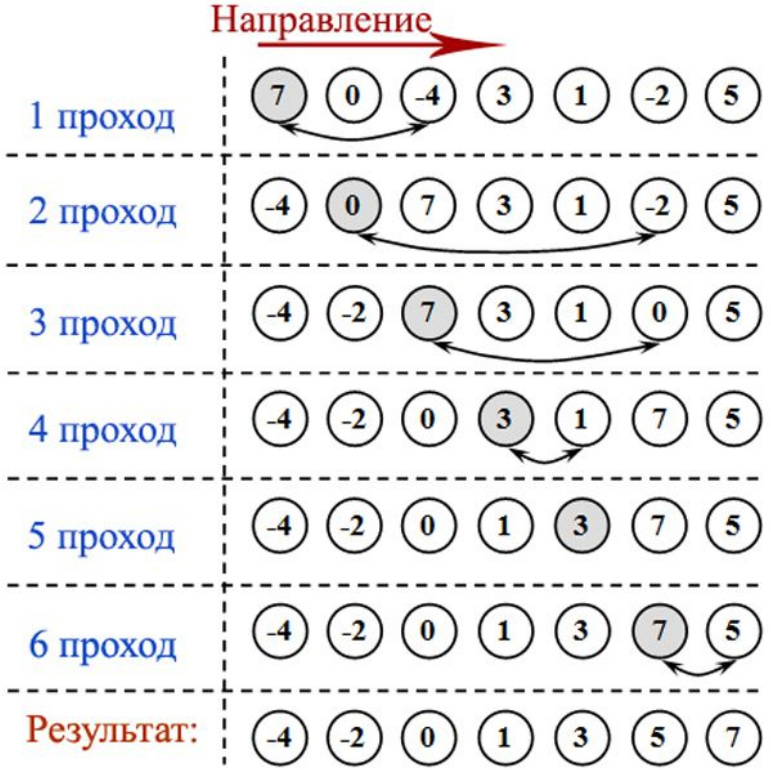
Сортировка выбором

Проходимся по всему массиву и запоминаем меньший элемент, если встретили еще меньше, то запоминаем его.

Когда дошли до конца массива меняем первый элемент с тем который нашли.

Начинаем заново со второго элемента

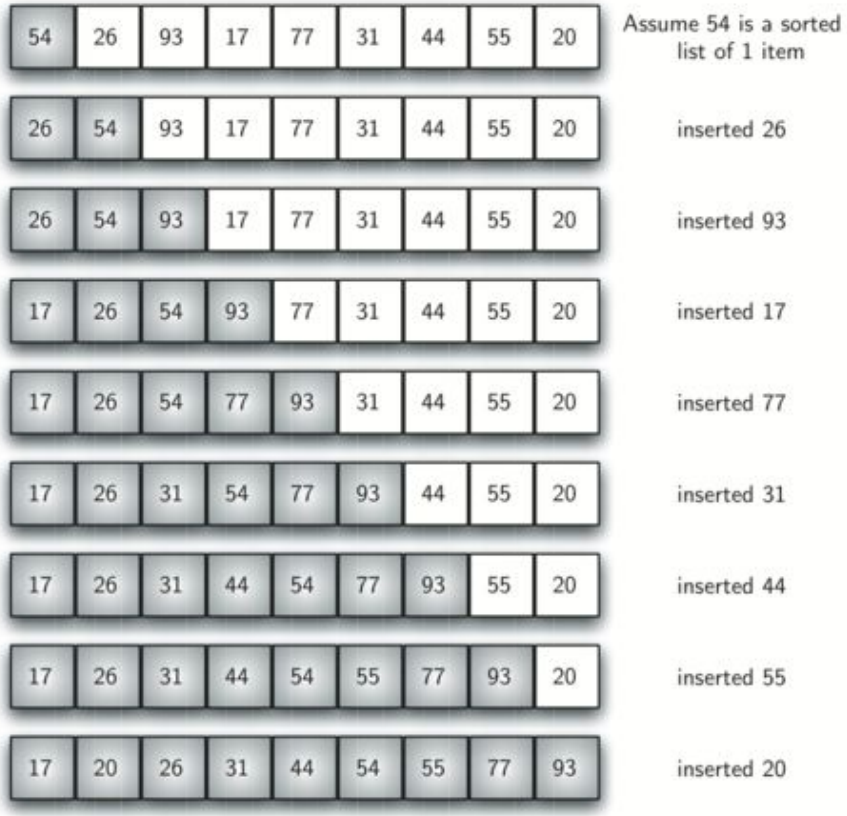
и так далее…



Сортировка вставками

Сортируем сначала первый элемент из массива. Потом первый и второй, потом первый, второй и третий…

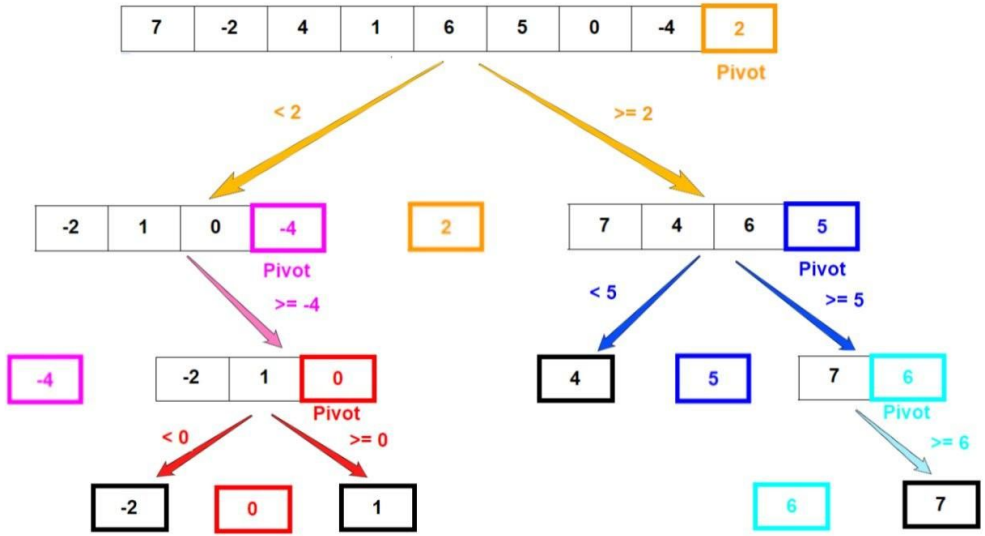
Каждый новый элемент находит свое место в уже отсортированной области массива



Быстрая сортировка

Берем опорный элемент массива и перебрасываем относительно него вправо большие элементы, влево меньшие. Образуются две области.

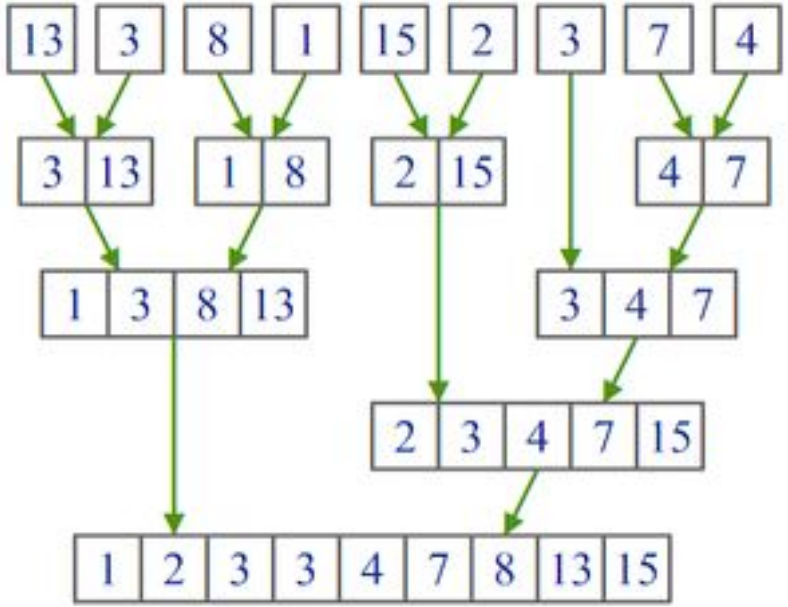
Пока не останется один элемент в области:
Повторяем то же самое в правой и левой области.



Сортировка слиянием

Разбиваем исходный массив на маленькие массивы, в которых сортируем элементы.
Делаем слияние маленьких массивов в средние. Сортируем.
Делаем слияние в большой массив. Сортируем.

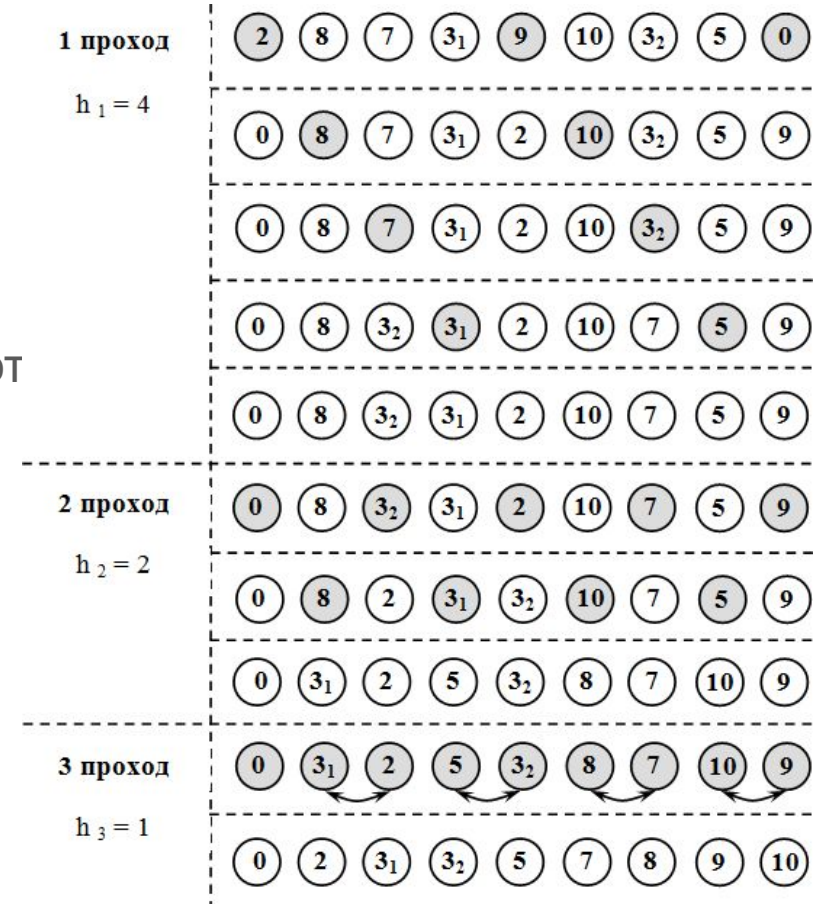
Количество таких шагов зависит от размера, но логика остается та же



Сортировка Шелла

Является усовершенствованным вариантом сортировки вставками.

Сортировка происходит в зависимости от шага (b на рисунке). Элементы в таком случае занимают более выгодные позиции для… обычной **сортировки вставками** в самом конце.



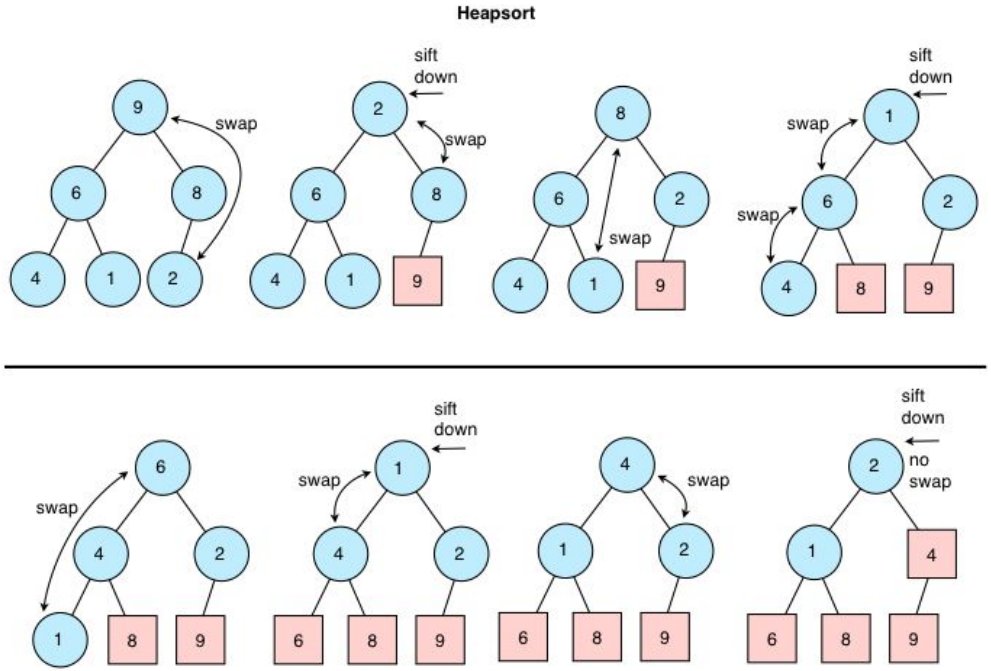
Сортировка кучей

Из исходных данных формируется бинарное дерево.

На данном этапе самый большой элемент хранится в корне кучи. Заменяем его на последний элемент кучи и уменьшаем ее размер на 1.

Преобразуем полученное дерево в бинарное дерево с новым корнем

Повторяем 2 и 3 абзац пока размер не станет равен одному



Как образуется сложность

Сложность O(1):
//действие не требует прохождения по элементам контейнера, а значит не зависит от его размера

Сложность O(logn): (не путать со сложностью O(nlogn))
for(int i=0; i<n; i++) { //на каждом шаге n уменьшается в два раза (n/=2)}

$$\log_a b = x \Leftrightarrow a^x = b$$

$\log_2 1024$ = 10 (столько действий будет при количестве элементов 1024)

Сложность O(n):
for(int i=0; i<n; i++) { //действие }

1024 = 1024 (столько действий будет при количестве элементов 1024)

Сложность O(n^2):
```
for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++){ //действие }
}
```

1024^2 = 1048576 (столько действий будет при количестве элементов 1024)

Алгоритмическая сложность популярных сортировок

Для алгоритмов сортировок сложность O(n) является лучшей, так как при любой сортировке нужно хотя бы один раз пройтись по всем элементам в массиве (n - количество элементов)

Алгоритм сортировки	Худший случай	Средний случай	Лучший случай	Объем памяти
Сортировка пузырьком	n^2	n^2	n	1
Сортировка выбором	n^2	n^2	n^2	1
Сортировка вставками	n^2	n^2	n	1
Быстрая сортировка	n^2	nlog n	nlog n	nlog n
Сортировка слиянием	nlog n	nlog n	nlog n	n
Сортировка Шелла	nlog^2n	nlog^2n	n	1
Сортировка кучей	nlog n	nlog n	nlog n	1

Алгоритмическая сложность (приблизительное время)

размер / сложность	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00005 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 минут	5,2 минут	13 минут
2^n	0,0001 сек.	1 сек.	17,9 минут	12,7 дней	35,7 веков	366 веков
3^n	0,059 сек.	58 минут	6,5 лет	3855 веков	$2x10^8$ веков	$1,3x10^{13}$ веков

*когда случайно обновил страницу на госуслугах

Что такое компаратор?

Это функция, которая как бы учит сортировать функцию sort именно так как задумывает программист

```
bool comp (<первый>, <второй>) {
    return <первый> <условный оператор> <второй>;
}

int main(){
    //…
    sort (vec.begin(), vec.end(), comp);
    //…
}
```

Функция sort

Это функция, которая может сортировать (на алгоритме быстрой сортировки) указанный контейнер или обычный массив. По умолчанию она сортирует по неубыванию, но это можно изменить путем применения компаратора.

```
#include <algorithm>
sort (<начало>, <конец>, <компаратор>);
```

```
vector <int> vec = {1, 9, 4, 0, 5, 4};

//от начала до конца по неубыванию (0 1 4 4 5 9)
sort (vec.begin(), vec.end());

//от середины до конца с помощью компаратора (например 1 9 4 5 4 0)
sort(vec.begin() + 3, vec.end(), comp);
```

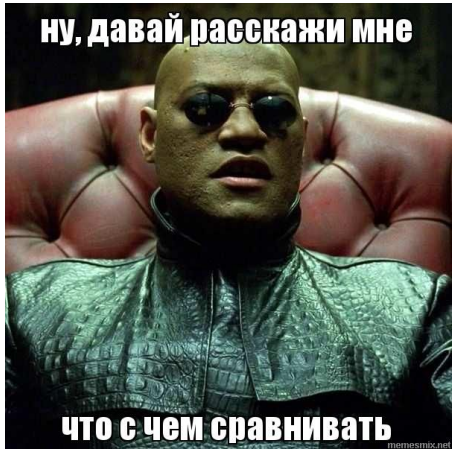
Перегрузка операторов сравнения

```
class Animal{
public:
        int age;

        //перегружаем оператор <
        bool operator<(const Animal& second) {
                return this->age < second.age;
        }
};
```

```
vector<Animal> vec;

//сортировка по оператору <
sort(vec.begin(), vec.end());
```



Метод swap

```
#include <iostream>
using namespace std;

int main()
{
	int a = 4;
	int b = 2;
	swap(a, b);
	cout << "a = " << a << endl; // a = 2
	cout << "b = " << b << endl; // b = 4
}
```

Что такое поиск и зачем он нужен?

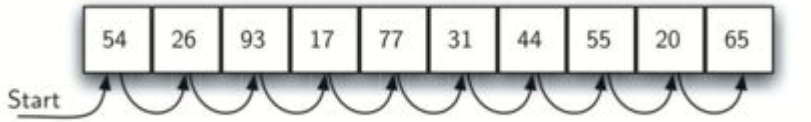
Найти что-то в массиве — довольно распространенная задача. Это может быть поиск целого объекта по его признаку. Например, когда нам нужно найти объект банковской карточки по id.

Или это может быть проверка на вхождение. К примеру, мы можем узнать, может ли игрок участвовать в игровом событии или у него недостаточный “уровень”.

Для оптимизации поиска данные должны храниться упорядочено, иначе придется обходить все элементы в поисках нужного.

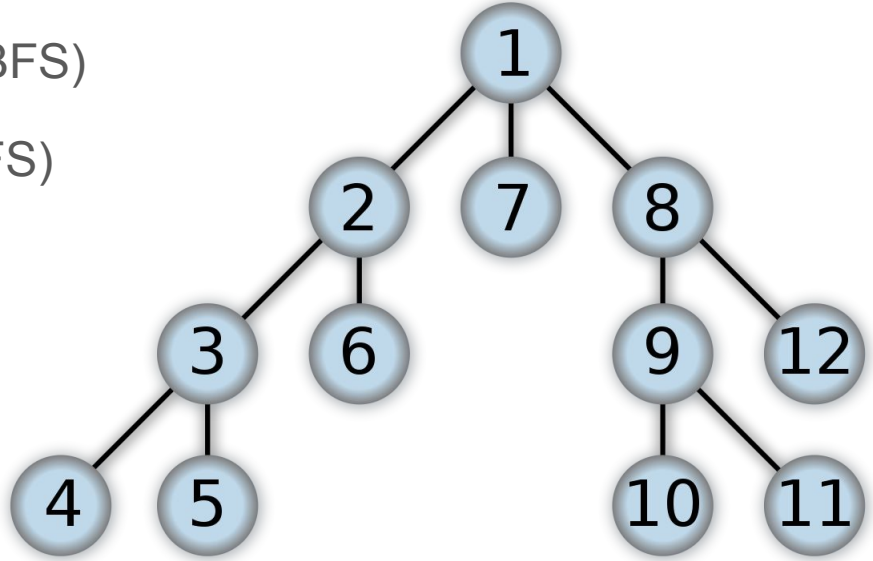
Популярные алгоритмы поиска

Линейный поиск (Linear Search)



Двоичный поиск (Binary Search)

Поиск в ширину (Breadth-First Search, BFS)



Поиск в глубину (Depth-First Search, DFS)