# Программирование

## 5 | Сложность

15 марта 2021

# Сложность

- Какой из двух алгоритмов лучше?
- Как долго выполняется алгоритм?
- Как много памяти он потребляет?

# Сложность

- Какой из двух алгоритмов лучше?
- Как долго выполняется алгоритм?
- Как много памяти он потребляет?
- Что важнее?

# Поиск элемента в массиве

Для данного значения `x` и массива `arr` найти такой индекс `i`, что `arr[i] == x`.

```
int find(int arr[], int size, int x) {
    ...
}
```

# Поиск элемента в массиве

Для данного значения `x` и массива `arr` найти такой индекс `i`, что `arr[i] == x`.

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

# Поиск элемента в массиве

А если массив упорядочен?

# Поиск элемента в массиве

А если массив упорядочен?

```
int arr[] = {4, 8, 15, 16, 23, 42, 108};

int find(int arr[], int size, int x) {
    assert(isSorted(arr, size));
    ...
}
```

# Поиск элемента в массиве

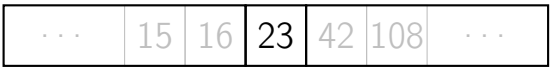
А если массив упорядочен?

1. Если `size == 1`

# Поиск элемента в массиве

А если массив упорядочен?

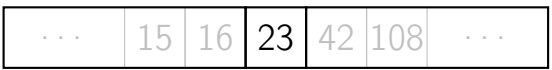
1. Если `size == 1`
2. Если `size == N > 1`



# Поиск элемента в массиве

А если массив упорядочен?

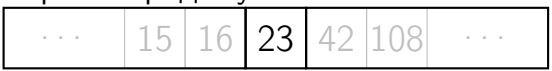
1. Если `size == 1`
2. Если `size == N > 1`
   Положим, что задача решена для `size < N`



# Поиск элемента в массиве

А если массив упорядочен?

1. Если `size == 1`
2. Если `size == N > 1`
   Положим, что задача решена для `size < N`
   Выберем середину массива



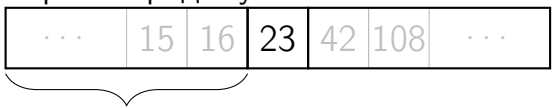
# Поиск элемента в массиве

А если массив упорядочен?

1. Если `size == 1`
2. Если `size == N > 1`

   Положим, что задача решена для `size < N`

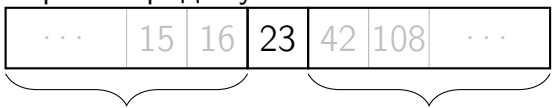
   Выберем середину массива



# Поиск элемента в массиве

А если массив упорядочен?

1. Если `size == 1`
2. Если `size == N > 1`
   Положим, что задача решена для `size < N`
   Выберем середину массива



# Поиск элемента в массиве

Для данного значения `x` и отсортированного массива `arr` найти такой индекс `i`, что `arr[i] == x`.

```
int find_binary(int arr[], int l, int r, int x) {
    if (l > r) { return -1; }
    int pivot = (l + r) / 2;
    if (arr[pivot] == x) { return pivot; }
    if (arr[pivot] > x) {
        return find_binary(arr, l, pivot - 1, x);
    }
    return find_binary(arr, pivot + 1, r, x);
}
```

# Сравнение алгоритмов

- Какой алгоритм лучше?

# Сравнение алгоритмов

- Какой алгоритм лучше?
- Какой быстрее работает на отсортированном массиве?

# Сравнение алгоритмов

- Какой алгоритм лучше?
- Какой быстрее работает на отсортированном массиве?
- Как много операций приходится выполнять?

# Сравнение алгоритмов

- Какой алгоритм лучше?
- Какой быстрее работает на отсортированном массиве?
- Как много операций приходится выполнять?
  - Зависит от размера входных данных (массива)

# Сравнение алгоритмов

- Какой алгоритм лучше?
- Какой быстрее работает на отсортированном массиве?
- Как много операций приходится выполнять?
  - Зависит от размера входных данных (массива)
  - Зависит от характера входных данных

# Сравнение алгоритмов

## Как много операций приходится выполнять алгоритму?

- Функция размера входных данных
- Функция распределения входных данных

# Сравнение алгоритмов

## Как много операций приходится выполнять алгоритму?

- Функция размера входных данных
- Функция распределения входных данных
  - В лучше случае

# Сравнение алгоритмов

## Как много операций приходится выполнять алгоритму?

- Функция размера входных данных
- Функция распределения входных данных
  - В лучше случае
  - В худшем случае

# Сравнение алгоритмов

## Как много операций приходится выполнять алгоритму?

- Функция размера входных данных
- Функция распределения входных данных
  - В лучше случае
  - В худшем случае
  - В наиболее вероятном (среднем) случае

# Сравнение алгоритмов

## Что аткое операция?

- Арифметика

# Сравнение алгоритмов

## Что аткое операция?

- Арифметика
- Сравнения

# Сравнение алгоритмов

## Что аткое операция?

- Арифметика
- Сравнения
- Чтение и запись памяти

# Сравнение алгоритмов

## Что аткое операция?

- Арифметика
- Сравнения
- Чтение и запись памяти
- ...

# Поиск перебором

- Арифметика
- Сравнения
- Чтение и запись памяти

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

# Поиск перебором

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- 4 операции на итерацию

# Поиск перебором

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- 4 операции на итерацию
- От 1 до size итераций

# Поиск перебором

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- 4 операции на итерацию
- От 1 до size итераций

- 4 * size операций в худшем случае

# Поиск перебором

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- 4 операции на итерацию
- От 1 до size итераций

- 4 * size операций в худшем случае
- Ещё инициализация i и return

# Поиск перебором

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- 4 операции на итерацию
- От 1 до size итераций
- (4 * size) + 2 операций в худшем случае

# Бинарный поиск

```
int find_binary(int arr[], int l, int r, int x) {
    if (l > r) { return -1; }
    int pivot = (l + r) / 2;
    if (arr[pivot] == x) { return pivot; }
    if (arr[pivot] > x) {
        return find_binary(arr, l, pivot - 1, x);
    }
    return find_binary(arr, pivot + 1, r, x);
}
```

# Бинарный поиск

```
int find_binary(int arr[], int l, int r, int x) {
    if (l > r) { return -1; }
    int pivot = (l + r) / 2;
    if (arr[pivot] == x) { return pivot; }
    if (arr[pivot] > x) {
        return find_binary(arr, l, pivot - 1, x);
    }
    return find_binary(arr, pivot + 1, r, x);
}
```

- Не менее 12 операции в каждом вызове!

# Бинарный поиск

```
int find_binary(int arr[], int l, int r, int x) {
    if (l > r) { return -1; }
    int pivot = (l + r) / 2;
    if (arr[pivot] == x) { return pivot; }
    if (arr[pivot] > x) {
        return find_binary(arr, l, pivot - 1, x);
    }
    return find_binary(arr, pivot + 1, r, x);
}
```

- Не менее 12 операции в каждом вызове!
- `[log_2(size)] + 1` вызовов

# Бинарный поиск

```
int find_binary(int arr[], int l, int r, int x) {
    if (l > r) { return -1; }
    int pivot = (l + r) / 2;
    if (arr[pivot] == x) { return pivot; }
    if (arr[pivot] > x) {
        return find_binary(arr, l, pivot - 1, x);
    }
    return find_binary(arr, pivot + 1, r, x);
}
```

- Не менее 12 операции в каждом вызове!
- `[log_2(size)] + 1` вызовов
- `([log_2(size)] + 1) + 1` операций в худшем случае

# Сравнение сложности

- `(4 * size) + 2`
- `([log_2(size)] + 1) + 1`

- $O$

# Сравнение сложности

- `(4 * size) + 2`
- `([log_2(size)] + 1) + 1`
- Как сравнить?
- $O$

# TODO

- Подробнее про классы функций
- Средний случай
- Ёмкостная сложность

# Q & A