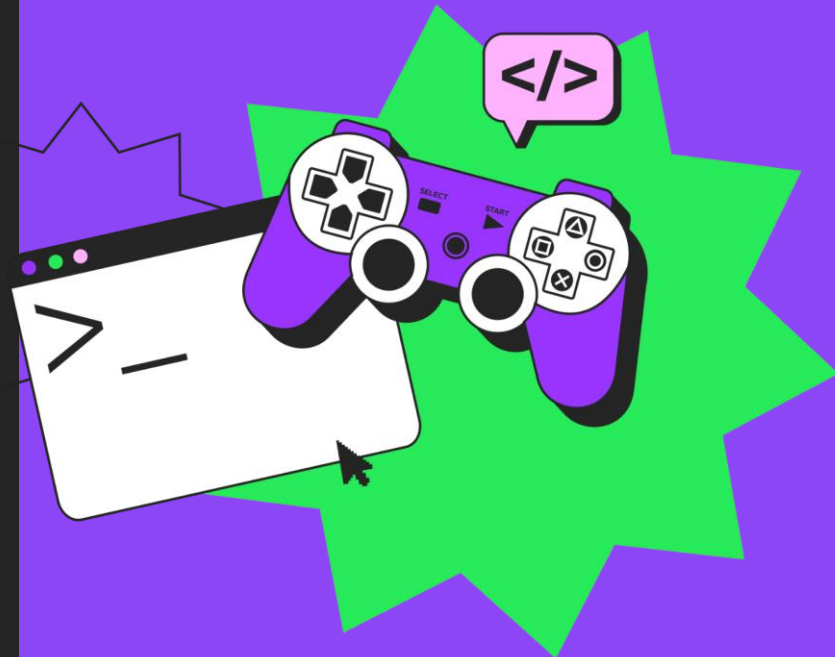


Алгоритмы. Оценка сложности алгоритмов

Урок 1 Алгоритмы и структуры данных





Алексей Плеханов

Ведущий разработчик компании IBS

Специализируюсь на разработке финтех продуктов, работаю на проектах крупнейшего банка РФ

- ✧ Разрабатываю системы для корпоративного инвестиционного бизнеса
- ✧ Оптимизирую логику и ускоряю работу сервисов
- ✧ Выступаю в роли наставника для молодых специалистов
- ✧ Отвечаю за выбор технических решений

План курса





Что будет на уроке сегодня

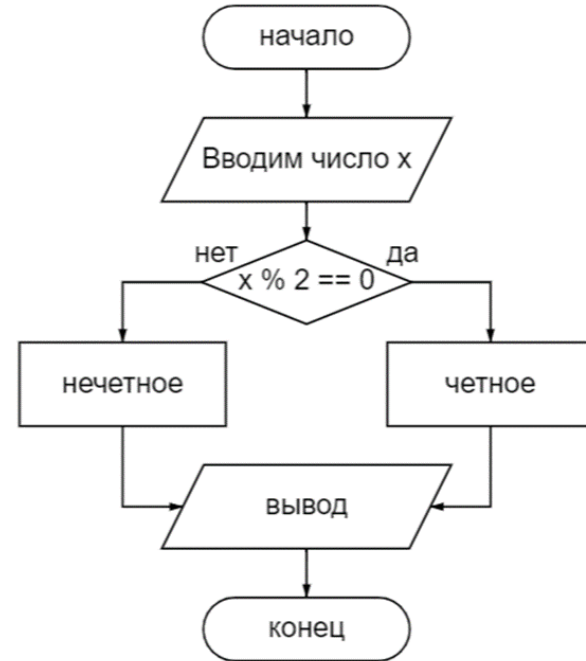
- ✧ Что такое “сложность алгоритма”
- ✧ Алгоритмы какой сложности встречаются чаще всего
- ✧ Примеры кода с различной сложностью
- ✧ Какие показатели сложности существуют
- ✧ Вычисление сложности для комплексных алгоритмов

Что же такое алгоритм?

Алгоритм — это точно определённая инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи

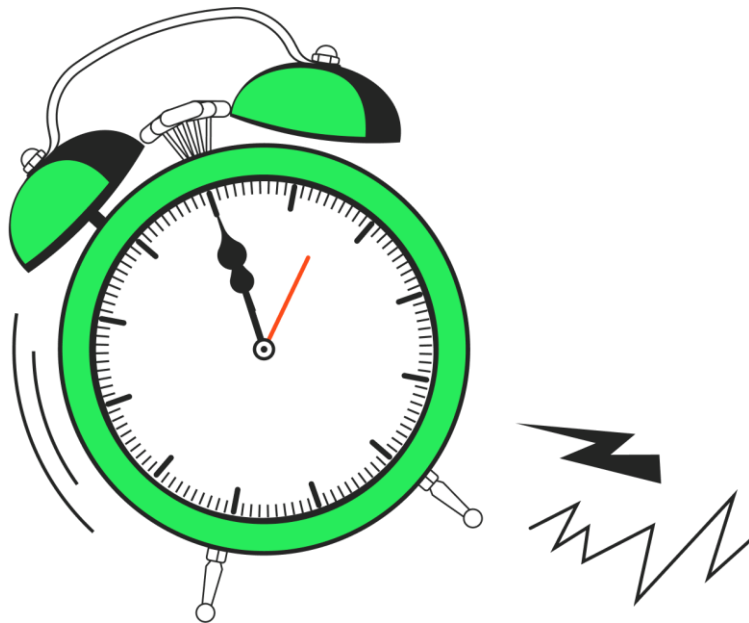
Часто алгоритм можно представить в виде блок-схемы.

Например - алгоритм вычисления, является ли введенное число четным



Критерии сложности алгоритма

- Скорость работы
- Объем потребляемой памяти (Оперативной и/или постоянной)



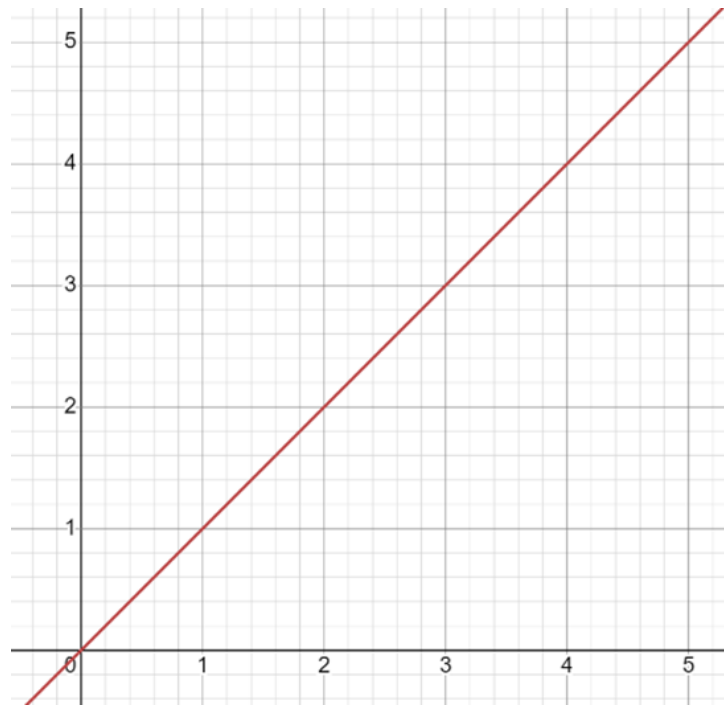
Алгоритм поиска допустимых делителей

```
1 public static List<Integer> availableDivider(int number) {  
2     List<Integer> result = new ArrayList<>();  
3     for (int i = 1; i ≤ number; i++) {  
4         if (number % i == 0) {  
5             result.add(i);  
6         }  
7     }  
8     return result;  
9 }
```



График линейной зависимости

Такая зависимость характеризуется симметричным ростом количества шагов относительно увеличения объема входных данных.



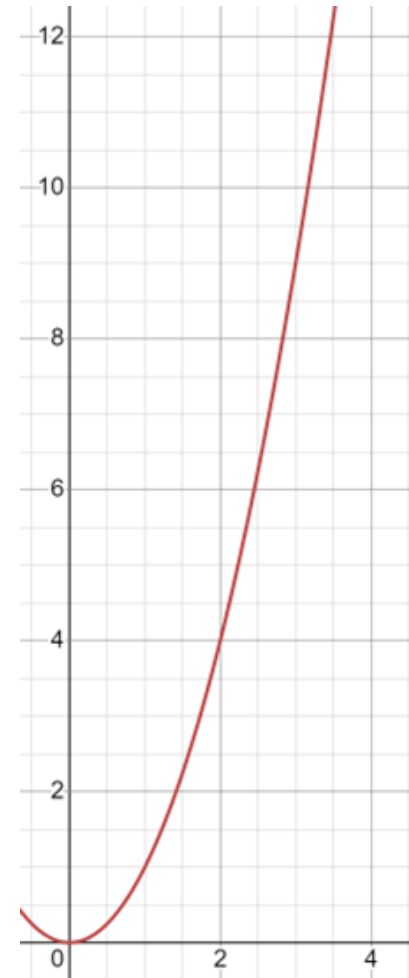
Алгоритм поиска простых чисел

```
1 public static List<Integer> findSimpleNumbers(int max) {  
2     List<Integer> result = new ArrayList<>();  
3     for (int i = 1; i ≤ max; i++) {  
4         boolean simple = true;  
5         for (int j = 2; j < i; j++) {  
6             if (i % j == 0) {  
7                 simple = false;  
8             }  
9         }  
10        if (simple) {  
11            result.add(i);  
12        }  
13    }  
14    return result;  
15 }
```



График квадратичной зависимости

Такая зависимость характеризуется резким ростом сложности относительно роста размера входных данных



Как описывается сложность алгоритма

Для описания сложности существует общепринятая нотация - $O(f(n))$, где n - размер входных данных.

Например, алгоритм перебора массива циклом `for` имеет сложность $O(n)$. С ростом n на x , количество шагов алгоритма тоже вырастает на x

А использование вложенного цикла `for` уже будет иметь сложность $O(n^2)$, например, при $n = 3$ цикл сделает 9 итераций, а при $n = 4$ уже 16 и т.д.

```
1 for (int i = 0; i < n; i++) {  
2     //do something  
3 }
```

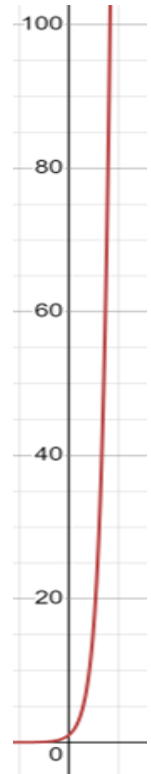
```
1 for (int i = 0; i < n; i++) {  
2     for (int j = 0; j < n; j++) {  
3         //do something  
4     }  
5 }
```



График экспоненциальной зависимости

Яркий пример - задача поиска шанса выпадения определенной суммы на игральный костях.

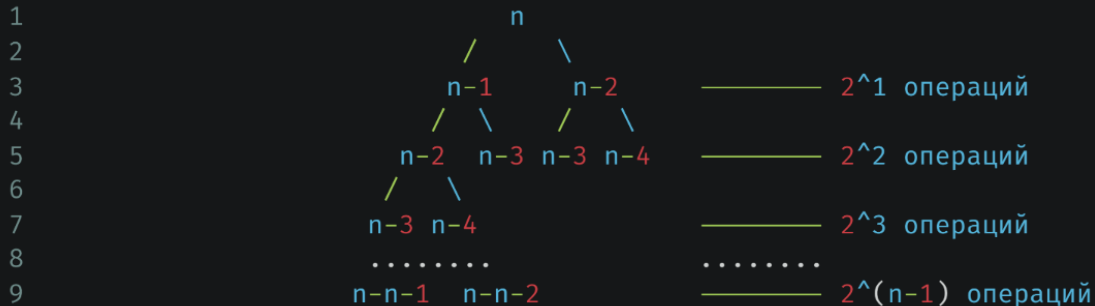
```
1 //Для трех шестигранных кубиков
2 public static double findSum(int sum) {
3     int count = 0;
4     int successResult = 0;
5     for (int i = 1; i ≤ 6; i++) {
6         for (int j = 1; j ≤ 6; j++) {
7             for (int k = 1; k ≤ 6; k++) {
8                 if (i + j + k = sum) {
9                     successResult++;
10                }
11            }
12        }
13    }
14    return ((double) successResult) / ((double) count);
15 }
16 }
```



Функция вычисления чисел Фибоначчи

Последовательность чисел Фибоначчи начинается с чисел 0 и 1, а все последующие элементы вычисляется путем сложения двух предыдущих. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 и т.д.

```
1 public static int fib(int position) {  
2     if (position == 1 || position == 2) {  
3         return 1;  
4     }  
5     return fib(position - 1) + fib(position - 2);  
6 }
```



Правила объединения сложности

Вызов нескольких методов на каждом шаге: **$O(2n) == O(n)$**

Обход половины размерности массива:

$O(n/2) == O(n)$

Цифровые множители сокращаются.

Вызов нескольких методов вне цикла:

$O(2+n) == O(n)$. Цифровые слагаемые сокращаются

```
1 for (int i = 0; i < n; i++) {  
2     //method1()  
3     //method2()  
4 }
```

```
1 for (int i = 0; i < n / 2; i++) {  
2     //method()  
3 }
```

```
1 for (int i = 0; i < n; i++) {  
2     //method()  
3 }  
4 //method1()  
5 //method2()
```



Правила объединения сложности

method1() - имеет сложность $O(n^3)$

method2() - имеет сложность $O(n^2)$

Если внутри method1() будет вызываться method2(),
то их сложности перемножаются:

$$O(n^3) * O(n^2) == O(n^5)$$

Если методы будут вызываться последовательно, то
их сложности складываются, т.е. берется
максимальная из них:

$$O(n^3) + O(n^2) == O(n^3)$$

```
1 void method1() {
2     for (int i = 0; i < n; i++){
3         for (int j = 0; j < n; j++){
4             for (int k = 0; k < n; k++){
5                 //do something
6             }
7         }
8     }
9 }
10
11 void method2() {
12     for (int i = 0; i < n; i++){
13         for (int j = 0; j < n; j++){
14             //do something
15         }
16     }
17 }
```





Какая бывает сложность алгоритмов

- **$O(1)$** - константная. Не зависит от объема данных. Например - поиск по хэш-таблице
- **$O(\log n)$** - логарифмическая. Увеличение размера почти не сказывается на количестве итераций. Например - бинарный поиск, поиск по сбалансированному дереву
- **$O(n)$** - линейная. Увеличение сложности эквивалентно увеличению размера. Например - поиск по неотсортированному массиву
- **$O(n * \log n)$** - увеличение размера заметно сказывается на сложности. Например - быстрая сортировка
- **$O(n^2)$** - квадратичная. Увеличение размера очень сильно сказывается на сложности. Например - пузырьковая сортировка
- **$O(2^n)$** - экспоненциальная. С увеличением размера на 1, сложность возрастает вдвое

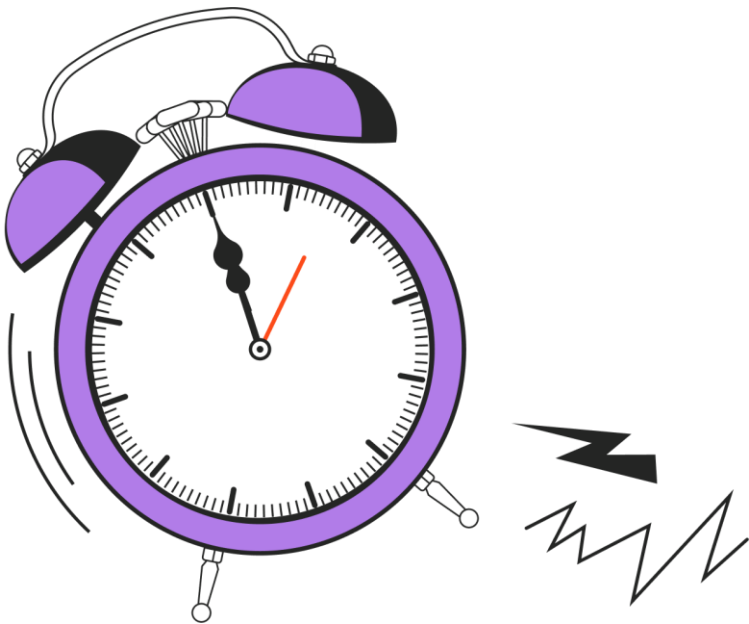
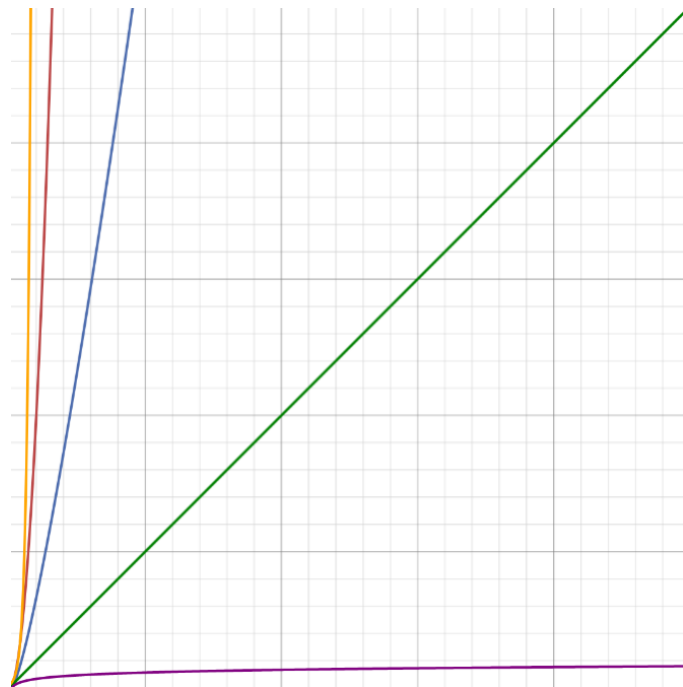


График сравнения роста сложности алгоритма

- Желтое - $O(2^n)$
- Красное - $O(n^2)$
- Синее - $O(n * \log n)$
- Зеленое - $O(n)$
- Фиолетовое - $O(\log n)$







Время расчета алгоритмов разной сложности


	N=10	N=20	N=30	N=40	N=50
N^2	0,0001 с	0,0004 с	0,0009 с	0,0016 с	0,0025 с
N^3	0,001 с	0,008 с	0,027 с	0,064 с	0,125 с
2^N	0,001 с	1,05 с	17,9 мин	12,7 дней	35,7 лет
3^N	0,05 с	58,1 мин	6,5 лет	$3,8 \cdot 10^5$ лет	$2,27 \cdot 10^{10}$ лет





Итоги урока

-  Познакомились с понятием “алгоритм” и его представлением в виде блок-схемы
-  Узнали, что такое сложность алгоритма и почему важно ее оценивать
-  Научились вычислять сложность для произвольных функций
-  Узнали особенность сложения сложности в композитных алгоритмах

Спасибо 
за внимание

