

# Алгоритмы и анализ сложности

Отчет

Свистунова М.П., 1ИВТ

# Разделы:

- [Исследование алгоритмов сортировки](#)
- [Рекурсия](#)
- [Алгоритм половинного деления](#)
- [Типизация](#)
- [Автоматное программирование](#)

# Исследование алгоритмов сортировки

## Цели:

- Сравнить алгоритмы сортировки массива вставками и пузырьком
- Рассмотреть наихудшие случаи (отсортированность в обратном порядке) и определить зависимость количества  $f$  совершаемых перестановок от числа  $N$  элементов в массиве. Вывести общую формулу  $f(N)$

# № 1

**Bubble sort (сортировка пузырьком).** Если первый элемент массива больше второго, то элементы меняются местами. Алгоритм выполняется до тех пор, пока все элементы массива не будут отсортированы.

**Insertion sort (сортировка вставками).** В алгоритме при каждой итерации выбирается элемент и происходит сравнение с элементами отсортированной части массива. При нахождении правильного места для данного элемента он переставляется на нужную позицию. Так происходит до тех пор, пока алгоритм не пройдет по всему массиву.

## № 2

При отсортированности в обратном порядке выявляется следующая зависимость количества  $f$  совершаемых перестановок от числа  $N$  элементов в массиве:

$$\text{при } N = 4, f(N) = 6$$

$$\text{при } N = 5, f(N) = 10$$

$$\text{при } N = 6, f(N) = 15$$

$$\text{при } N = 7, f(N) = 21$$

Общая формула:  $f(N) = (N-1)*N/2$

# Выводы

- Проведено сравнение алгоритмов сортировки массива вставками и пузырьком
- Рассмотрены наихудшие случаи (отсортированность в обратном порядке) и определена зависимость количества  $f$  совершаемых перестановок от числа  $N$  элементов в массиве. Выведена общая формула  $f(N)$

# Рекурсия



## Цели:

- рассмотреть рекурсивную реализацию возведения в степень
- найти степень с основанием 5, которая вызовет переполнение стека
- вычислить количество вызовов при  $\text{pwr}(3,2)$

# № 1

Задание: по аналогии напишите рекурсивное определение `pwr(a,b)` - возведения в степень, опираясь на **`mpy`**.

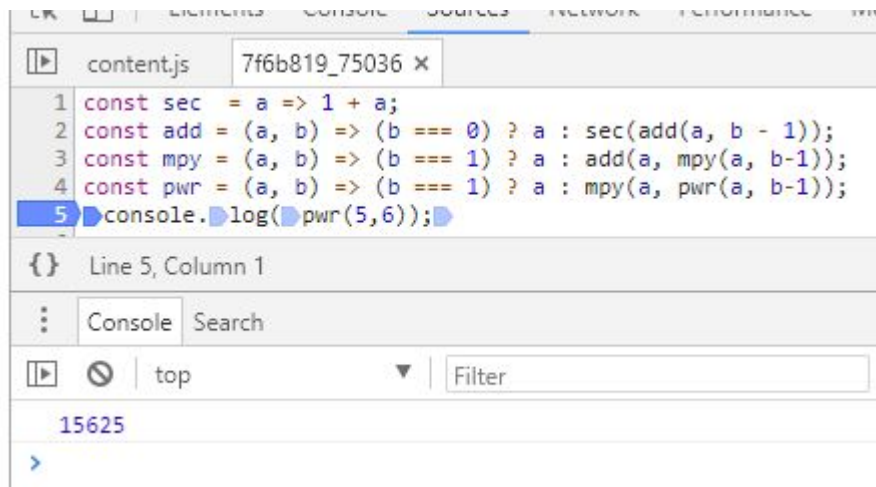
Решение:

```
const sec = a => 1 + a;  
const add = (a, b) => (b === 0) ? a : sec(add(a, b - 1));  
const mpy = (a, b) => (b === 1) ? a : add(a, mpy(a, b-1));  
const pwr = (a, b) => (b === 1) ? a : mpy(a, pwr(a, b-1));
```

## № 2

Задание: какая степень с основанием 5 вызовет переполнение стека?

Решение: переполнение стека при  $5^7$



The screenshot shows a web browser's developer console with the 'Sources' tab open. A file named 'content.js' is loaded at address '7f6b819\_75036'. The code defines a recursive function 'sec' and two helper functions 'add' and 'mpy'. The 'mpy' function calls 'pwr' with arguments (a, b-1), and 'pwr' calls 'mpy' with arguments (a, b-1). The 'pwr' function is called with '5, 6' in line 5. The console output shows the result '15625'.

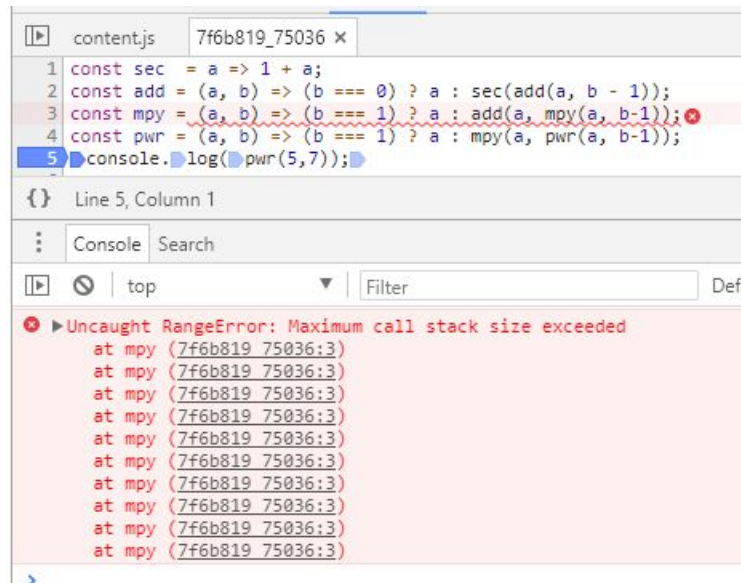
```
1 const sec = a => 1 + a;  
2 const add = (a, b) => (b === 0) ? a : sec(add(a, b - 1));  
3 const mpy = (a, b) => (b === 1) ? a : add(a, mpy(a, b-1));  
4 const pwr = (a, b) => (b === 1) ? a : mpy(a, pwr(a, b-1));  
5 console.log(pwr(5,6));
```

Line 5, Column 1

Console Search

top Filter

15625



The screenshot shows a web browser's developer console with the 'Sources' tab open. A file named 'content.js' is loaded at address '7f6b819\_75036'. The code is the same as in the previous screenshot, but the 'pwr' function is called with '5, 7' in line 5. The console output shows an 'Uncaught RangeError: Maximum call stack size exceeded' error, indicating a stack overflow.

```
1 const sec = a => 1 + a;  
2 const add = (a, b) => (b === 0) ? a : sec(add(a, b - 1));  
3 const mpy = (a, b) => (b === 1) ? a : add(a, mpy(a, b-1));  
4 const pwr = (a, b) => (b === 1) ? a : mpy(a, pwr(a, b-1));  
5 console.log(pwr(5,7));
```

Line 5, Column 1

Console Search

top Filter Def

Uncaught RangeError: Maximum call stack size exceeded

at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)  
at mpy (7f6b819\_75036:3)

# № 3

Задание: сколько вызовов sec, add, mpy и pwr будет при **pwr(3,2)**?

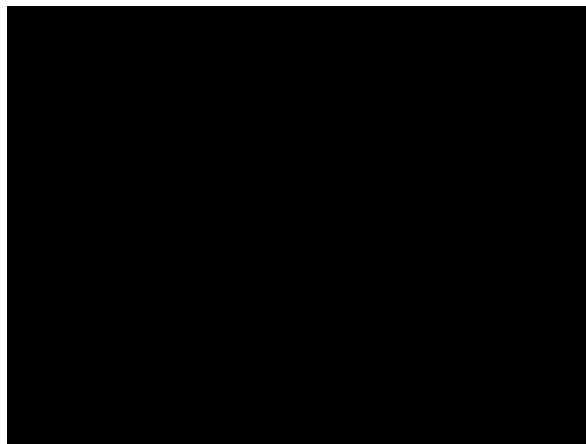
Решение:

sec - 12 вызовов

add - 11 вызовов

mpy - 3 вызова

pwr - 2 вызова



pwr( 3 )# 1	
pwr( 3 )# 2	
mpy( 3 )# 1	add( 3 )# 7
mpy( 3 )# 2	add( 3 )# 8
mpy( 3 )# 3	add( 3 )# 9
add( 3 )# 1	add( 3 )# 10
add( 3 )# 2	add( 3 )# 11
add( 3 )# 3	sec( 3 )# 4
add( 3 )# 4	sec( 4 )# 5
sec( 3 )# 1	sec( 5 )# 6
sec( 4 )# 2	sec( 6 )# 7
sec( 5 )# 3	sec( 7 )# 8
add( 3 )# 5	sec( 8 )# 9
add( 3 )# 6	
add( 3 )# 7	

# Выводы

- рассмотрена рекурсивная реализация возведения в степень
- найдена степень с основанием 5, которая вызывает переполнение стека
- вычислено количество вызовов при  $\text{pwr}(3,2)$

# Алгоритм половинного деления

## Цели:

- Узнать, за какое количество шагов будет достигнута точность до десятитысячных при выполнении алгоритма половинного деления для вычисления корня уравнения

# Решение

```
solve=(p,q)=>{  
  while (Math.abs(q-p)>0.0001) {  
    if (f(p)!=0) {  
      if (f(q)!=0) {  
        M=(p+q)/2;  
        if (f(p)*f(M)<0) {  
          q=M;  
          D++;  
        } else {  
          p=M;  
          D++;  
        }  
      }  
    }  
  }  
  return D;  
}
```

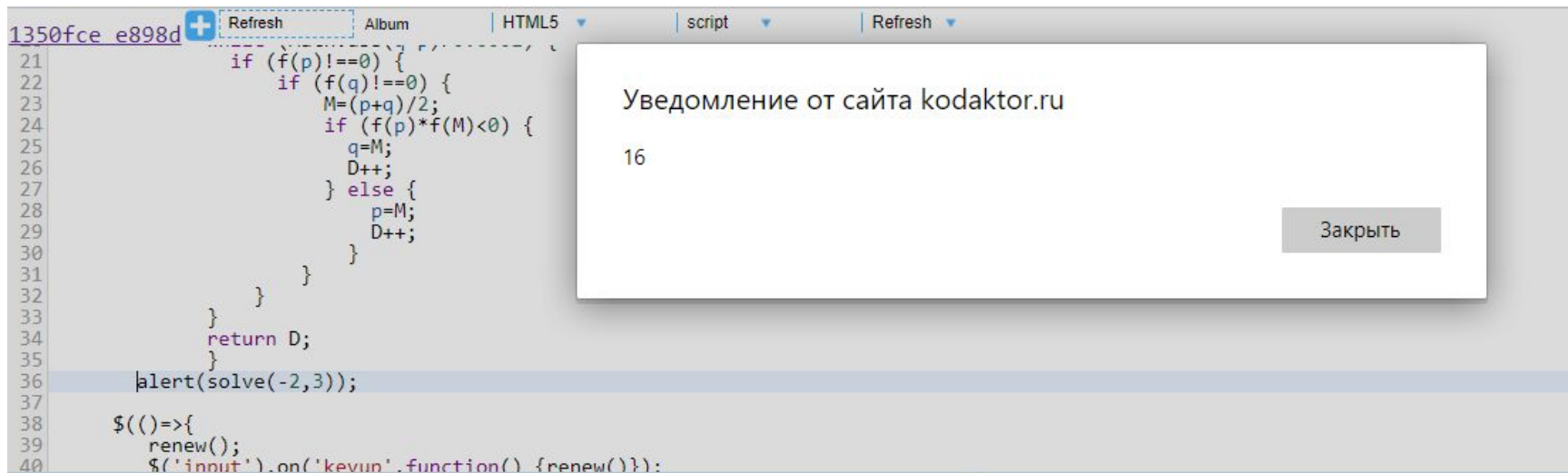
Ссылка на решение:

[https://kodaktor.ru/1350fce\\_defed](https://kodaktor.ru/1350fce_defed)



# Результаты

D - количество шагов, за которое достигнута необходимая точность



The screenshot shows a web browser window with a JavaScript code editor. The code is a bisection method for finding roots of a function. An alert dialog is displayed over the code, showing the number 16, which represents the number of steps (D) required to reach the necessary accuracy. The dialog has a 'Закрыть' (Close) button.

```
1350fce e898d + Refresh Album HTML5 script Refresh
21   if (f(p) !== 0) {
22     if (f(q) !== 0) {
23       M = (p+q)/2;
24       if (f(p)*f(M) < 0) {
25         q = M;
26         D++;
27       } else {
28         p = M;
29         D++;
30       }
31     }
32   }
33 }
34 return D;
35 }
36 alert(solve(-2,3));
37
38 $({})=>{
39   renew();
40   $('input').on('keyup', function() {renew()});
```

# Выводы

- Было найдено количество шагов, за которое можно достигнуть точности до десятитысячных при выполнении алгоритма половинного деления для вычисления корня уравнения

# Типизация

## Цели:

- представить числа -312.3125 и 3.1 в формате Double
- найти в справочных материалах по современному JavaScript, как можно получить это представление и увидеть конкретные байты прямо в программе (без попыток записи в файл и т.п.)
- создать борд с результатами перевода

№ 1 Число -312.3125

$$312.3125_{10} = 100111000.00101_2 * 2^0 = 1.0011100000101_2 * 2^8$$

$8 + 1023 = 1031_{10} = 10000000111_2$  – порядок

**1** – знак

001110000010100000000000000000000000000000000000000 - мантисса

Итого:

[illegible]

Разбиение на байты:

11000000 01110011 10000010 10000000 00000000 00000000 00000000 00000000

Запись в обратном порядке:

00000000 00000000 00000000 00000000 10000000 10000010 01110011 11000000

## № 1 Число 3.1

$$3.1_{10} = 11.00(0011)_2 * 2^0 = 1.10(0011)_2 * 2^1$$

$$1 + 1023 = 1024_{10} = 100000000000_2 - \text{порядок}$$

0 – знак

1000110011001100110011001100110011001100110011001100

Итого:

0 100000000000 100011001100110011001100110011001100110011001100

Разбиение на байты:

01000000 00001000 11001100 11001100 11001100 11001100 11001100 11001100

Запись в обратном порядке:

11001100 11001100 11001100 11001100 11001100 11001100 00001000 01000000

## № 2

Увидеть конкретные байты в представлении числа в формате Double можно с помощью типизированного представления массива.

Число -312.3125 [https://kodaktor.ru/js01a\\_50bc0](https://kodaktor.ru/js01a_50bc0)

Число 3.1 [https://kodaktor.ru/js01a\\_804fc](https://kodaktor.ru/js01a_804fc)

```
0
0
0
0
0
85
73
c0
```

-312.3125

```
cd
cc
cc
cc
cc
cc
8
40
```

3.1

# Выводы

- представлены числа -312.3125 и 3.1 в формате Double
- найдено в справочных материалах по современному JavaScript, как можно получить это представление и увидеть конкретные байты прямо в программе (без попыток записи в файл и т.п.)
- создан борд с результатами перевода

[Вернуться на слайд Разделы](#)



# Автоматное программирование

# Цели:

- написать программу, которая распознает слова shot, shoot, shooot, ...
- написать программу, которая распознает сбалансированные скобочные последовательности из 3-х видов скобок типа `([(){}])`

# № 1 shot, shoot, shoooot, ...

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    system("chcp 1251 > nul");
    char str[] = "shoooooooooot";
    int i = strlen(str), k, b=0;
    if ((str[0] == 's') && (str[1] == 'h') && (str[2] == 'o') && (str[i-1] == 't') && (b == 0)) {
        for (k = 3; k<=i-2; k++){
            if (str[k] != 'o'){
                b = 1;
            }
        }
    }
    if ((k == i - 1) && (b==0)){
        printf("Верно! Слово: %s\n", str);
    } else {
        printf("Не верно! Слово: %s\n", str);
    }
    return 0;
}
```

Ссылка на Google Документ: [решение на C](#)

Решение на kodaktor.ru:

[https://kodaktor.ru/js01a\\_a6ca3](https://kodaktor.ru/js01a_a6ca3)

## № 2 Скобочная последовательность

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    system("chcp 1251 > nul");
    char str[] = "{}{}";
    char mas[strlen(str)];
    int i = 0, k = i, b=0;
    for (i; i<=strlen(str)-1; i++){
        if (b==0) {
            switch(str[i]){
                case '(': mas[k] = str[i]; k++; break;
                case '[': mas[k] = str[i]; k++; break;
                case '{': mas[k] = str[i]; k++; break;
                default: {
                    switch(str[i]){
                        case ')': if (mas[k-1] == '(') { mas[k-1] = 0; k--;} else {b = 1;}; break;
                        case ']': if (mas[k-1] == '[') { mas[k-1] = 0; k--;} else {b = 1;}; break;
                        case '}': if (mas[k-1] == '{') { mas[k-1] = 0; k--;} else {b = 1;}; break; }}}}
        if (b==0){ printf("Верная последовательность\n");}
        else { printf("Не верная последовательность\n");}
    }
    return 0;}
```

Ссылка на Google Документ: [решение на C](#)

Исправленный вариант на kodaktor.ru:  
[https://kodaktor.ru/js01a\\_bea01](https://kodaktor.ru/js01a_bea01)

# Выводы

- написана программа, которая распознает слова shot, shoot, shooot, ...
- написана программа, которая распознает сбалансированные скобочные последовательности из 3-х видов скобок типа `([(){}])`