

Алгоритмы обработки данных на JavaScript

Фундаментальные алгоритмы программирования

Строгий режим

Включается директивой 'use strict'

Она должна быть расположена в **начале** файла

Цель директивы "use strict" — указать, что код должен выполняться в, так называемом, "строгом режиме"

- Нельзя использовать переменные без декларирования
- Одинаковые имена параметров запрещены
- Нельзя удалять переменную

Линейный поиск

- Заключается в итерации по массиву до тех пор, пока искомый элемент не будет найден



```
1 function linearSearch(array, element) {  
2     for (let i = 0; i < array.length; i++) {  
3         if (array[i] === element) {  
4             return i;  
5         }  
6     }  
7  
8     return -1;  
9 }
```

Поиск максимума/минимума

1. Объявляем переменную, которая будет хранить текущий максимум/минимум. Изначально она равна элементу с индексом 0.
2. Обходим массив с $i=1$ до $length-1$ и сравниваем каждый элемент массива с текущим минимум/максимум.
3. Если элемент массива больше (меньше) текущего минимума (максимума), то обновляем минимум (максимум).
4. После окончания цикла максимум либо минимум хранится в переменной текущего максимума (минимума)

Максимум/минимум

```
function max(array) {  
    if (array.length === 0) {  
        return undefined;  
    }  
    let maxElement = array[0];  
  
    for (let i = 1; i < array.length; i++) {  
        if (maxElement < array[i]) {  
            maxElement = array[i];  
        }  
    }  
  
    return maxElement;  
}
```

```
function min(array) {  
    if (array.length === 0) {  
        return undefined;  
    }  
    let minElement = array[0];  
  
    for (let i = 1; i < array.length; i++) {  
        if (minElement > array[i]) {  
            minElement = array[i];  
        }  
    }  
  
    return minElement;  
}
```

Сортировка

Сортировка - упорядочение списка произвольных элементов по возрастанию

Арифметический порядок - встречается при сортировке массива чисел.

Лексикографический порядок - за основу берётся порядок символов в таблице Unicode

При упорядочении списков нужно уметь **сравнивать** пары элементов, то есть устанавливать, меньше ли первый элемент второго, больше, или элементы равны

Метод sort

Метод `sort(comparatorFn)` на месте сортирует элементы массива и возвращает отсортированный массив

Если функция `comparatorFn` предоставлена, то элементы массива сортируются в соответствии с её возвращаемым значением

Если функция `comparatorFn` не предоставлена, то элементы массива сортируются путём преобразования в строки и сравнения строк в порядке следования кодов Unicode

Если функция `comparatorFn` сравнивает два элемента `a`, `b`, то

```
function compare(a, b) {  
    if (a меньше b по некоторому критерию  
    сортировки) {  
        return -1;  
    }  
    if (a больше b по некоторому критерию  
    сортировки) {  
        return 1;  
    }  
    // a должно быть равным b  
    return 0;  
}
```

Метод sort




```
1  [32, 2, 1, 5, 6, 7].sort((a, b) => a - b); // [1, 2, 5, 6, 7, 32]
2  [32, 2, 1, 5, 6, 7].sort(); // [1, 2, 32, 5, 6, 7]
```


Поиск в отсортированном массиве

1. Определяем значение элемента в середине массива. Полученное значение сравниваем с искомым элементом
2. Если искомый элемент меньше среднего, то продолжаем поиск в первой половине, иначе - во второй
3. Если интервал для поиска пустой, то элемент не найден. Иначе переходим к пункту один

Поиск в отсортированном массиве



```
1 function binarySearch(array, element) {
2     let min = 0;
3     let max = array.length - 1;
4     let guess;
5
6     while (min <= max) {
7         guess = Math.floor((max + min) / 2);
8         if (array[guess] === element) {
9             return guess;
10        }
11        else if (array[guess] < element) {
12            min = guess + 1;
13        }
14        else {
15            max = guess - 1;
16        }
17    }
18 }
19
20 return -1;
21 }
```