education

Сложность алгоритмов

Алгоритмы и структуры данных
Илья Почуев

Что будет на занятии

- Как оценивать сложность алгоритмов
- Зачем нужен анализ сложности
- Виды сложности алгоритмов
- Большое O (Big O): определение, основная идея, примеры использования
- Сложность алгоритма на примере вычисления чисел Фибоначчи
- Подходы к анализу алгоритмов

education

Сложность алгоритмов: понятие, виды, особенности

Сложность алгоритмов: понятие и виды

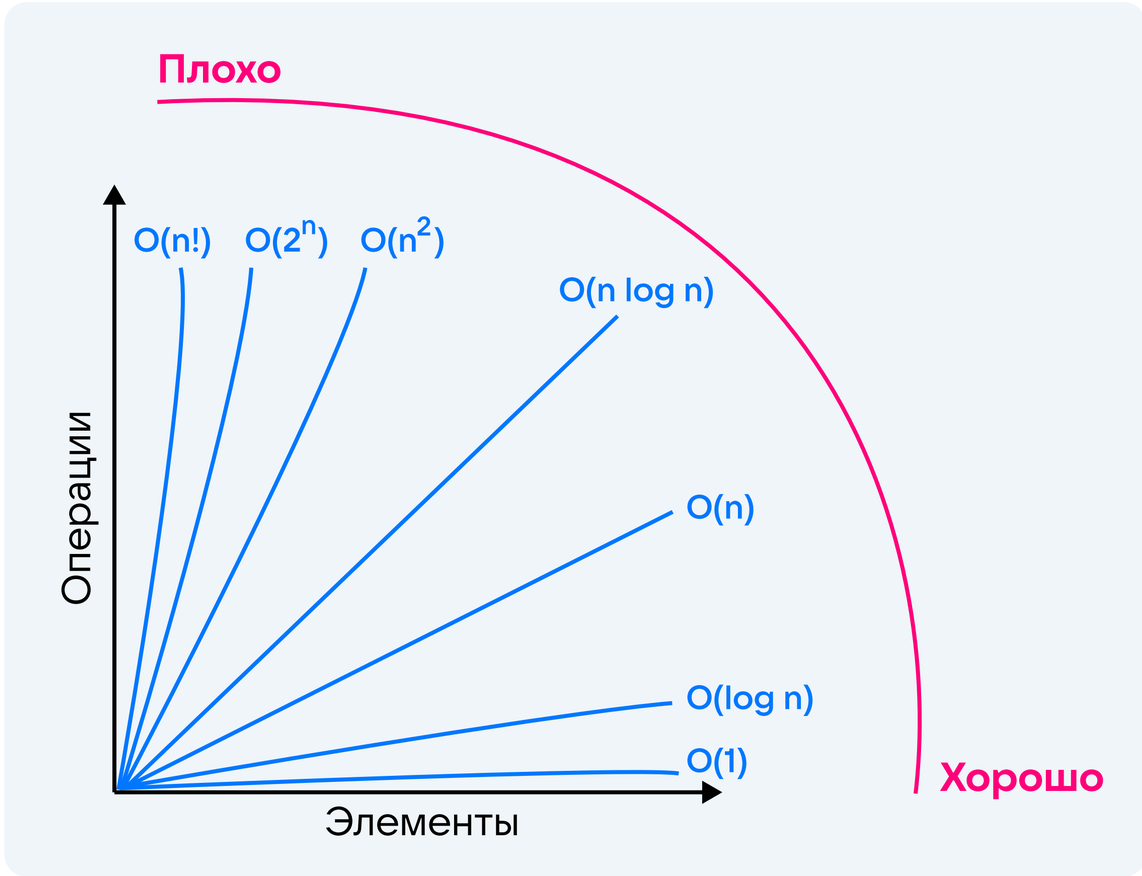
«О» большое — связь между количеством входных данных и временем или памятью, что понадобятся для выполнения алгоритма.

Виды сложностей:

- **временная сложность (time complexity) —** количество времени, которое нужно для выполнения алгоритма
- **пространственная сложность (space complexity) —** количество памяти, которое нужно для выполнения алгоритма

Сложность алгоритмов: классы и особенности

- Основные классы сложности — от O(1) до O(N!)
- Константы не влияют на сложность: O(2*N)~O(20*N)
- Учитываются все входные параметры
- Округление всегда до наихудшей сложности



education

Оценка сложности алгоритмов на примерах

O(n) — линейная сложность

```
function foo(n) {
        result = 0
        for (i = 0; i <= n; i++){
            result += i
        }
        return result
}
```

O(n) — линейная сложность

```
function foo(n) {
  result = 0
  for (i = 0; i <= n; i++) {
      result += i
  }
  for (i = 0; i <= n; i++) {
      result += i
  }
  for (i = 0; i <= n; i++) {
      result += i
  }
  return result
}
```

O(N + N + N) ~ O(3N) ~ O(N)

O(1) — константная сложность

```
function foo(n) {
    return 100*n + n2
}
```

O(n^2) — квадратичная сложность

```
function foo(n) {
        result = 0
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= n; j++) {
                result += i*j
            }
        }
        return result
}
```

Какая сложность?

```
function foo(n) {
    result = 0
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= n; j++) {
            result += i*j
        }
    }
    for (i = 0; i <= n; i++) {
        result += i
    }

    return result
}
```

```
function foo(n, m) {
    result = 0
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            result += i*j
        }
    }
    return result
}
```

O(n^2)

```
function foo(n) {
    result = 0
    for (i = 0; i <= n; i++) {
    for (j = 0; j <= n; j++) {
            result += i*j
        }
    }
    for (i = 0; i <= n; i++) {
        result += i
    }
                                O(n^2)
    return result
}
```

O(n*m)

```
function foo(n, m) {
    result = 0
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            result += i*j
        }
    }
    return result
}
```

education

Экспоненциальная сложность O(2^N) на примере вычисления чисел Фибоначчи

Числа Фибоначчи: определение

Числа Фибоначчи — последовательность чисел, где каждое следующее число — сумма двух предыдущих

Числа Фибоначчи: особенности

- Первые несколько чисел Фибоначчи — 0, 1, 1, 2, 3, 5, 8
- Для 0 и 1 определяется как (0) = 0, (1) = 1 соответственно
- Для n ≥ 2 определяется как F(n) = F(n − 1)+ +F(n − 2)

Вычисление числа Фибоначчи

Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Вычисление числа Фибоначчи

Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Если n больше 1, функция вызывает саму себя дважды: один раз с аргументом **n – 1** и один раз с аргументом **n – 2**

Вычисление числа Фибоначчи

Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Если n больше 1, функция вызывает саму себя дважды: один раз с аргументом **n – 1** и один раз с аргументом **n – 2**

fib(4) вызывает fib(3) и fib(2)

Вычисление числа Фибоначчи

Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Если n больше 1, функция вызывает саму себя дважды: один раз с аргументом **n – 1** и один раз с аргументом **n – 2**

fib(4) вызывает fib(3) и fib(2)

fib(3) вызывает fib(2) и fib(1)

Вычисление числа Фибоначчи

Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Если n больше 1, функция вызывает саму себя дважды: один раз с аргументом **n – 1** и один раз с аргументом **n – 2**

fib(4) вызывает fib(3) и fib(2)

fib(3) вызывает fib(2) и fib(1)

fib(2) вызывает fib(1) и fib(0)

Вычисление числа Фибоначчи

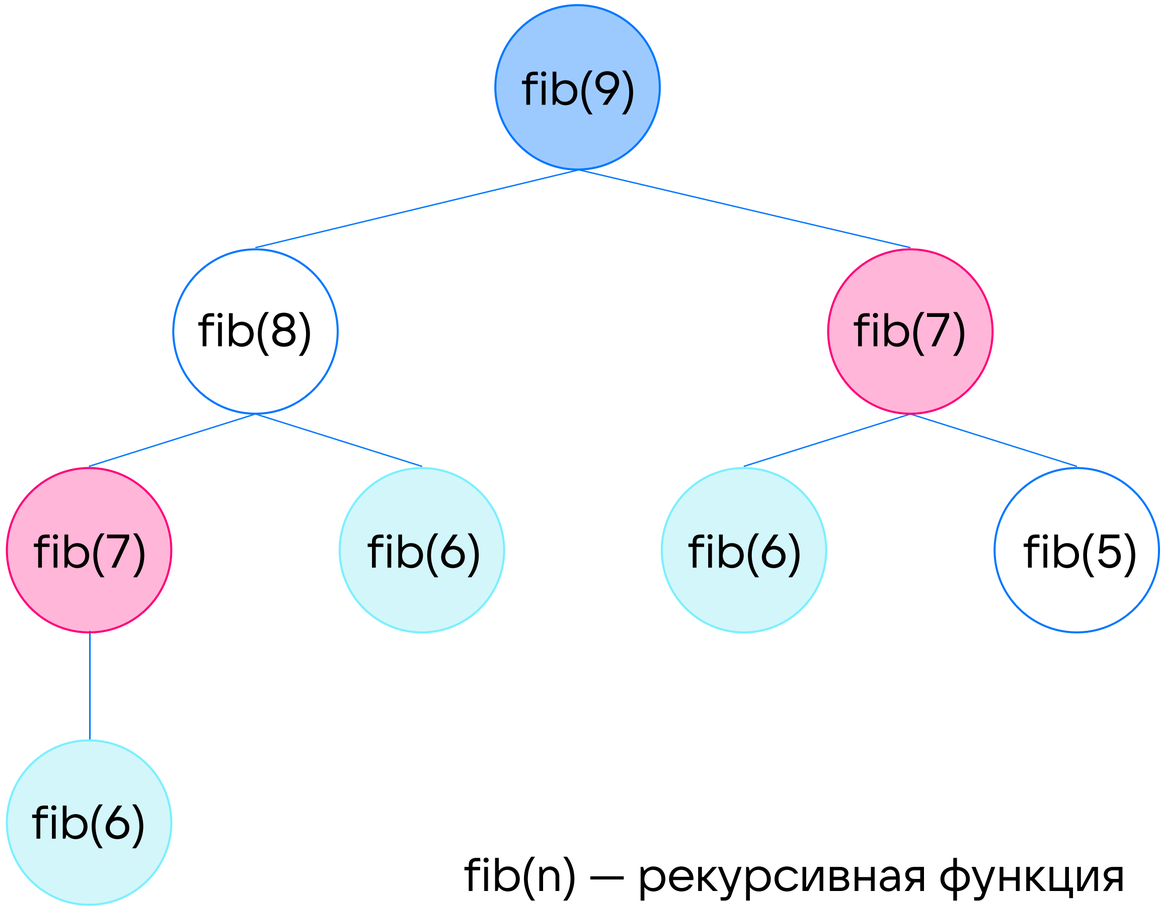
Рекурсивный подход

```
function fib(n) {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

Сложность — экспоненциальная,(2^n). Это неэффективный алгоритм для больших значений n

Недостатки рекурсивного подхода

- Часть значений нужно высчитывать по многу раз
- Каждый новый вызов fib() ничего не знает о предыдущих вызовах
- На каждом вызове все значения вычисляются заново
- Большие затраты на вычисления



fib(n) — рекурсивная функция

Вычисление числа Фибоначчи

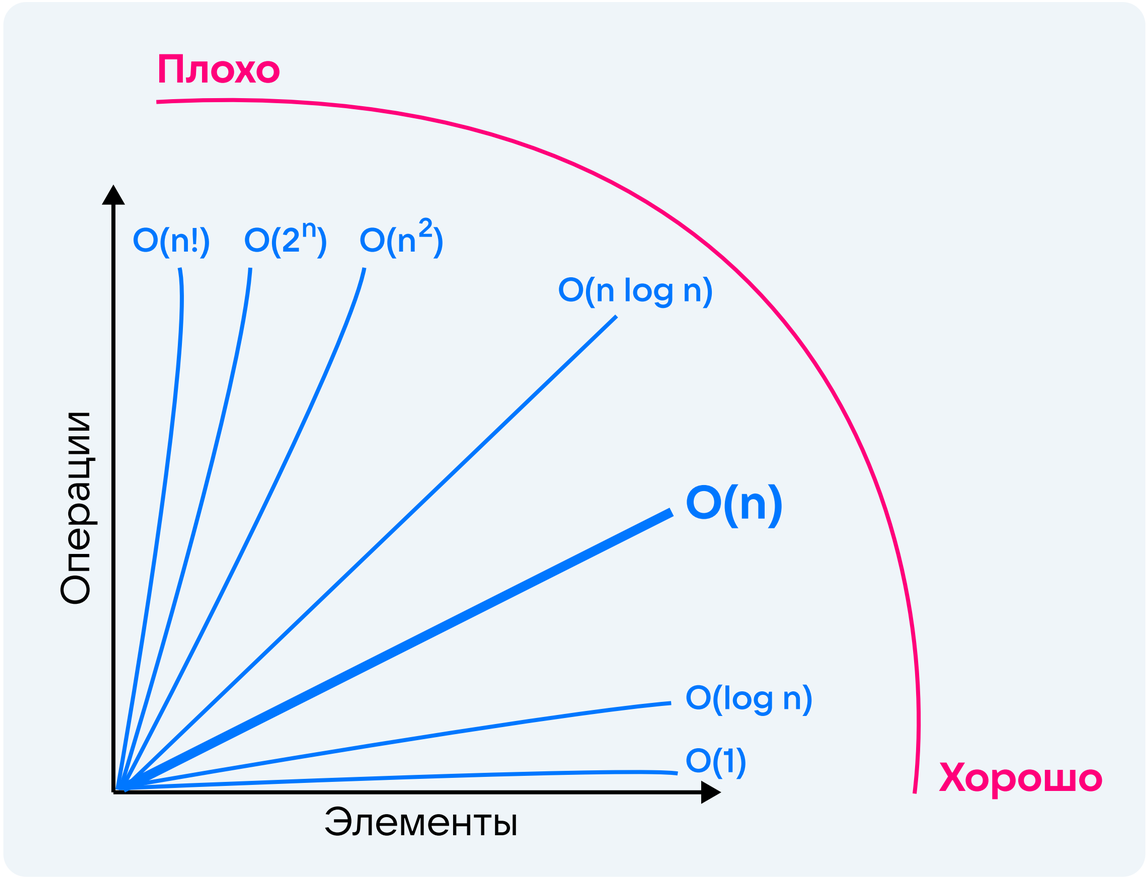
Итерационный подход

```
function fib(n) {
    if (n <= 1) {
        return n;
    }

    f1 = 0;
    f2 = 1;

    for (i = 2; i <= n; i++) {
        temp = f1 + f2;
        f1 = f2;
        f2 = temp;
    }

    return f2;
}
```

Вычисление числа Фибоначчи

Итерационный подход

```
function fib(n) {
    if (n <= 1) {
        return n;
    }

    f1 = 0;
    f2 = 1;

    for (i = 2; i <= n; i++) {
        temp = f1 + f2;
        f1 = f2;
        f2 = temp;
    }

    return f2;
}
```



education

Квадратичная сложность на примере сортировки пузырьком

Квадратичная сложность

```
function bubbleSort(arr) {
 n = len(arr);
 for (i = 0; i < n; i++) {
   for (j = 0; j < n - i - 1; j++) {
     if (arr[j] > arr[j + 1]) {
       // Поменять элементы местами
       tmp = arr[j];
       arr[j] = arr[j + 1];
       arr[j + 1] = tmp;
     }
   }
 }
 return arr;
}

// Чтобы поменять элементы местами, можно использовать
// swap(arr[j], arr[j + 1])
// Эквивалентный вариант:
// arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

Квадратичная сложность

```
function bubbleSort(arr) {
 n = len(arr);
 for (i = 0; i < n; i++) {
   for (j = 0; j < n - i - 1; j++) {
     if (arr[j] > arr[j + 1]) {
       // Поменять элементы местами
       tmp = arr[j];
       arr[j] = arr[j + 1];
       arr[j + 1] = tmp;
     }
   }
 }
 return arr;
}

// Чтобы поменять элементы местами, можно использовать
// swap(arr[j], arr[j + 1])
// Эквивалентный вариант:
// arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

Если на вход придёт отсортированный массив, сложность всё равно останется квадратичной

Квадратичная сложность

```
function bubbleSort(arr) {
 sorted = false;
 while !sorted {
  sorted = true;
  for (i = 0; i < len(arr) - 1; i++) {
   if (arr[i] > arr[i + 1]) {
    // Поменять элементы местами
    temp = arr[i];
    arr[i] = arr[i + 1];
    arr[i + 1] = temp;
    sorted = false;
   }
  }
 }
 return arr;
}
```

Квадратичная сложность

```
function bubbleSort(arr) {
 sorted = false;
 while !sorted {
  sorted = true;
  for (i = 0; i < len(arr) - 1; i++) {
   if (arr[i] > arr[i + 1]) {
    // Поменять элементы местами
    temp = arr[i];
    arr[i] = arr[i + 1];
    arr[i + 1] = temp;
    sorted = false;
   }
  }
 }
 return arr;
}
```

Если на вход придёт отсортированный массив, сложность уже будет O(n)

education

Оценка сложности на примере игры «Угадай число»

Правила игры «Угадай число»

Загадка числа

Один игрок загадывает число в определённом диапазоне (например, от 1 до 100)

Угадывание числа

Другой игрок пытается угадать это число, называя свои варианты

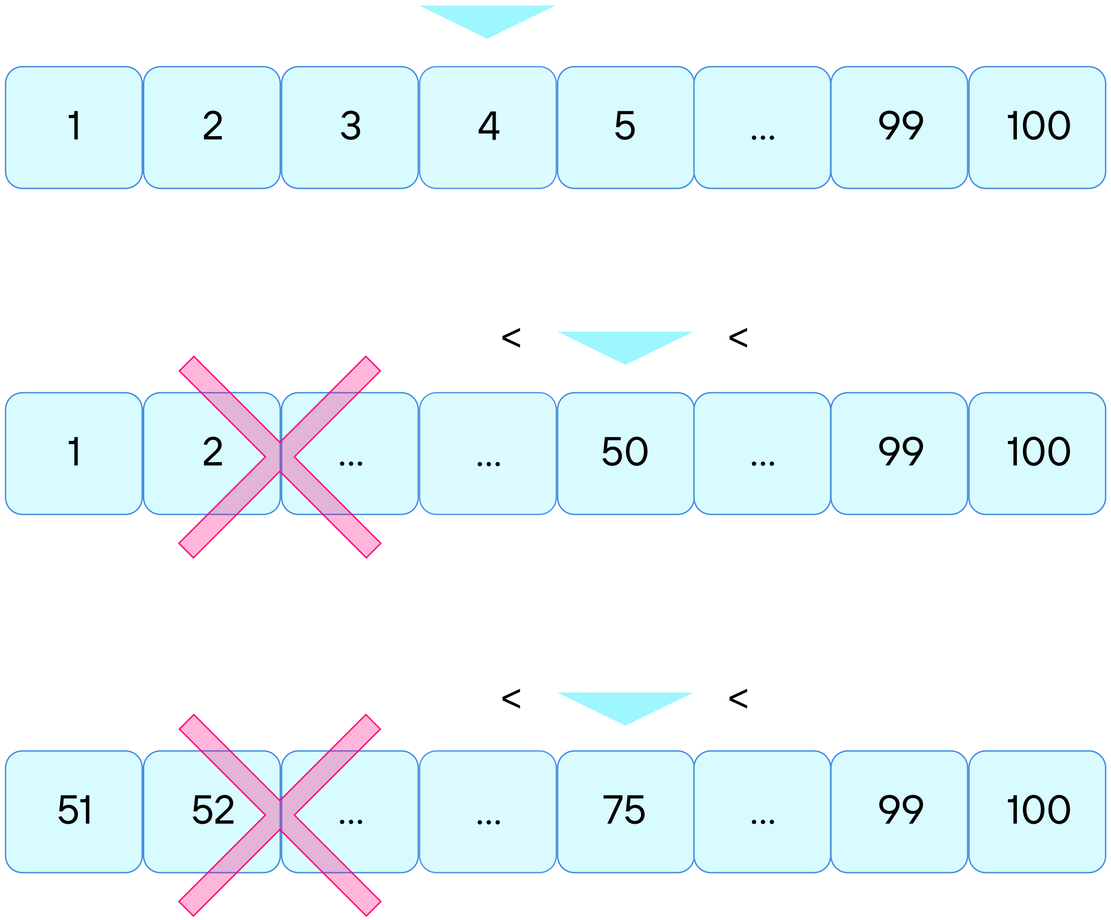
Подсказки

После каждой попытки загадавший число говорит, больше или меньше загаданного числа была попытка

Победа

Игра продолжается до тех пор, пока угадывающий не назовёт правильное число

Игра «Угадай число»

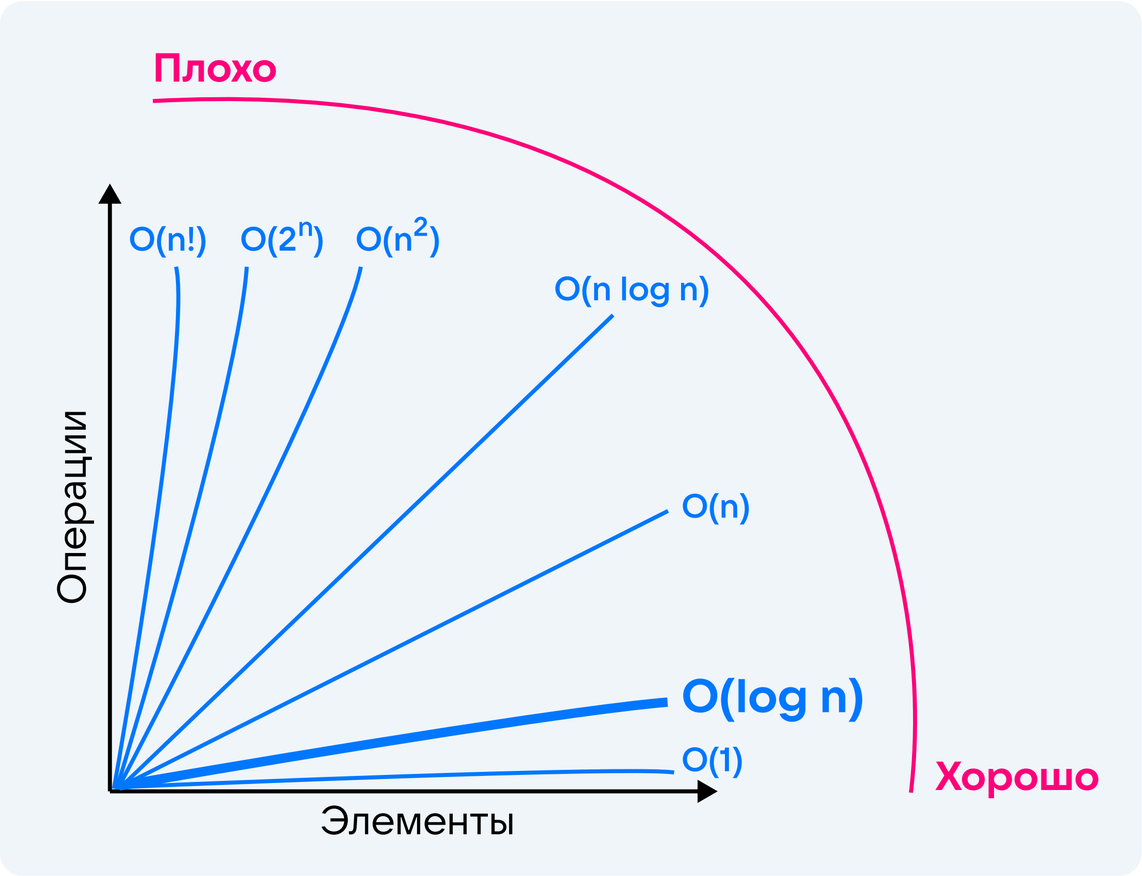


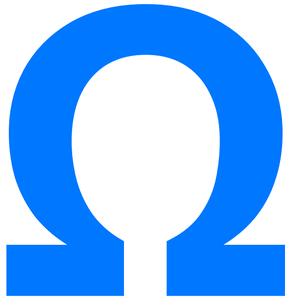
```
function binarySearch(data, needle, left, right) {
 if (left >= right) {
   if (data[left] == needle) {
     return left;
   }
   return -1; // Не найдено
 }

 mid = left + (right - left) / 2;

 if (data[mid] > needle) {
   return binarySearch(data, needle, left, mid - 1);
 } else if (data[mid] < needle) {
   return binarySearch(data, needle, mid + 1, right);
 }

 return mid;
}
```

Перед каждым рекурсивным вызовом отрезок делится пополам

education

Сложность алгоритма





(Big Omega)

```
function foo(n, target) {
    for (i = 0; i <= n; i++) {
        if (i == target) {
            return true
        }
    }
    return false
}
```

Θ (Big Theta)

- Точная оценку временной сложности алгоритма, включающаю как верхнюю, так и нижнюю границу
- Полный анализ сложности: худший, средний и лучший случаи

```
function foo(n, target) {
    for (i = 0; i <= n; i++) {
        if (i == target) {
            return true
        }
    }
    return false
}
```

```
const arr = [36, 18, 9, 11, 8]
for (let i = 0; i < arr.length - 1; i++) {
 setTimeout(
   () => {console.log(arr[i])},
   arr[i]
 )
}

// Функция-обработчик:
// () => {console.log(arr[i])} —
// анонимная функция, которая выводит
// в консоль текущий элемент массива arr[i].
// Задержка: arr[i] — время задержки
// в миллисекундах,
// зависящее от значения текущего элемента
// массива.
```

```
const arr = [36, 18, 9, 11, 8]
for (let i = 0; i < arr.length - 1; i++) {
 setTimeout(
   () => {console.log(arr[i])},
   arr[i]
 )
}

// Функция-обработчик:
// () => {console.log(arr[i])} —
// анонимная функция, которая выводит
// в консоль текущий элемент массива arr[i].
// Задержка: arr[i] — время задержки
// в миллисекундах,
// зависящее от значения текущего элемента
// массива.
```

Каждый элемент массива будет выведен в консоль с задержкой, равной значению этого элемента в миллисекундах


Спустя 15 лет
я нашёл его...
Вот он!!! Идеальный
алгоритм сортировки
const arr = [36, 18, 9, 11, 8]
for (let i = 0;i < arr.length - 1;i++) {
setTimeout(
() => {console.log(arr[i])},
arr[i]
)
}
Да чтоб
тебя

Итоги

- Разобрали понятие “О большое”
- На примере разных алгоритмов посмотрели, как можно дать оценку сложности
- На примере чисел Фибоначчи увидели, как два алгоритма, которые делают одно и то же имеют существенную разницу в оценке сложности
- Рассмотрели сценарии, когда при одной и той же итоговой сложности один алгоритм может быть предпочтительней другого

education

Спасибо
за внимание