PHASE 1

WEEK 1

DAY 3

ELBRUS/BOOTCAMP

План

1. Отладка (debugging)
2. Рекурсия (recursion)
3. Алгоритмы (algorithms)

Debugging

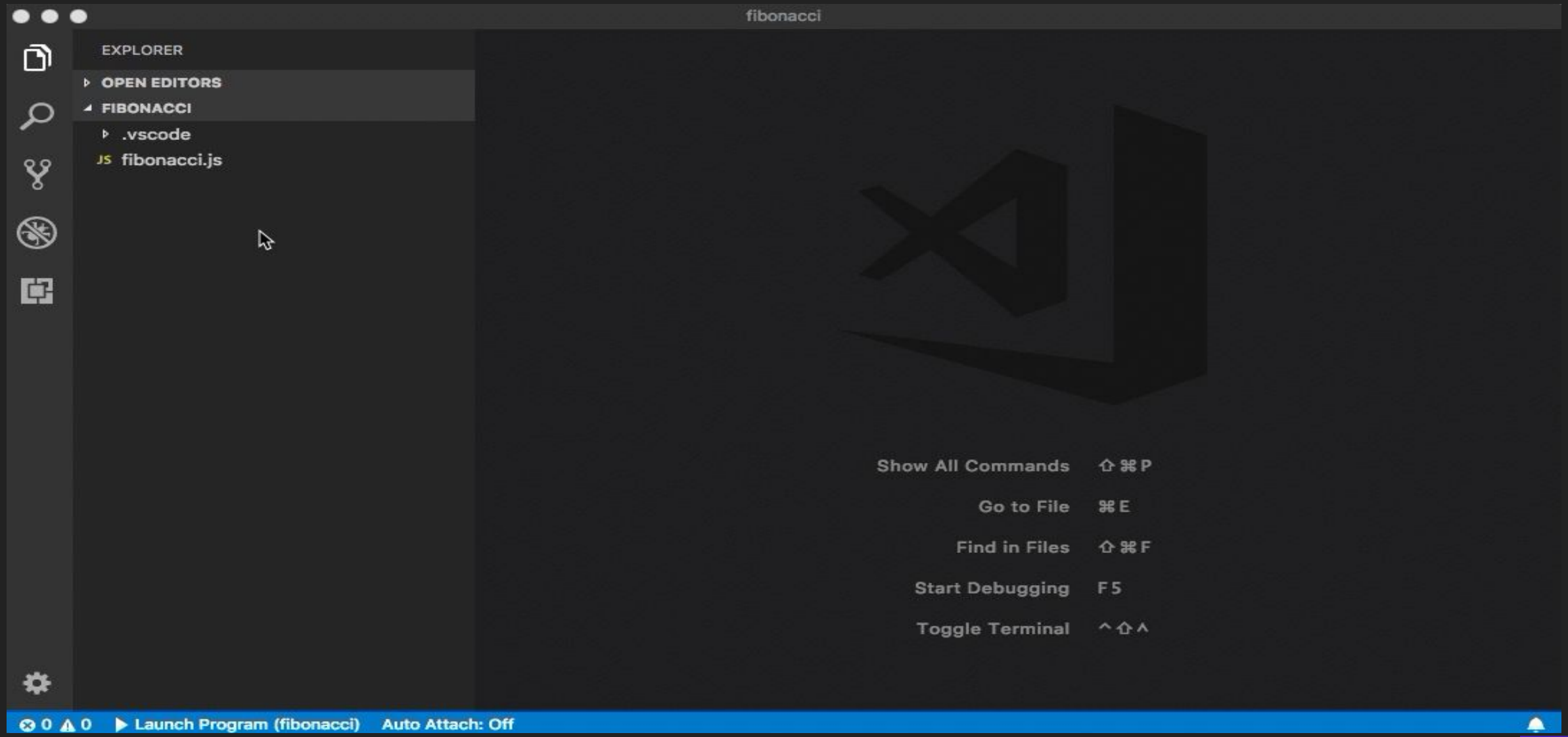
Debugging / отладка

Отладка – это процесс поиска и исправления ошибок в скрипте. Все современные браузеры и большинство других сред разработки поддерживают инструменты для отладки – специальный графический интерфейс, который сильно упрощает отладку. Он также позволяет по шагам отследить, что именно происходит в нашем коде.

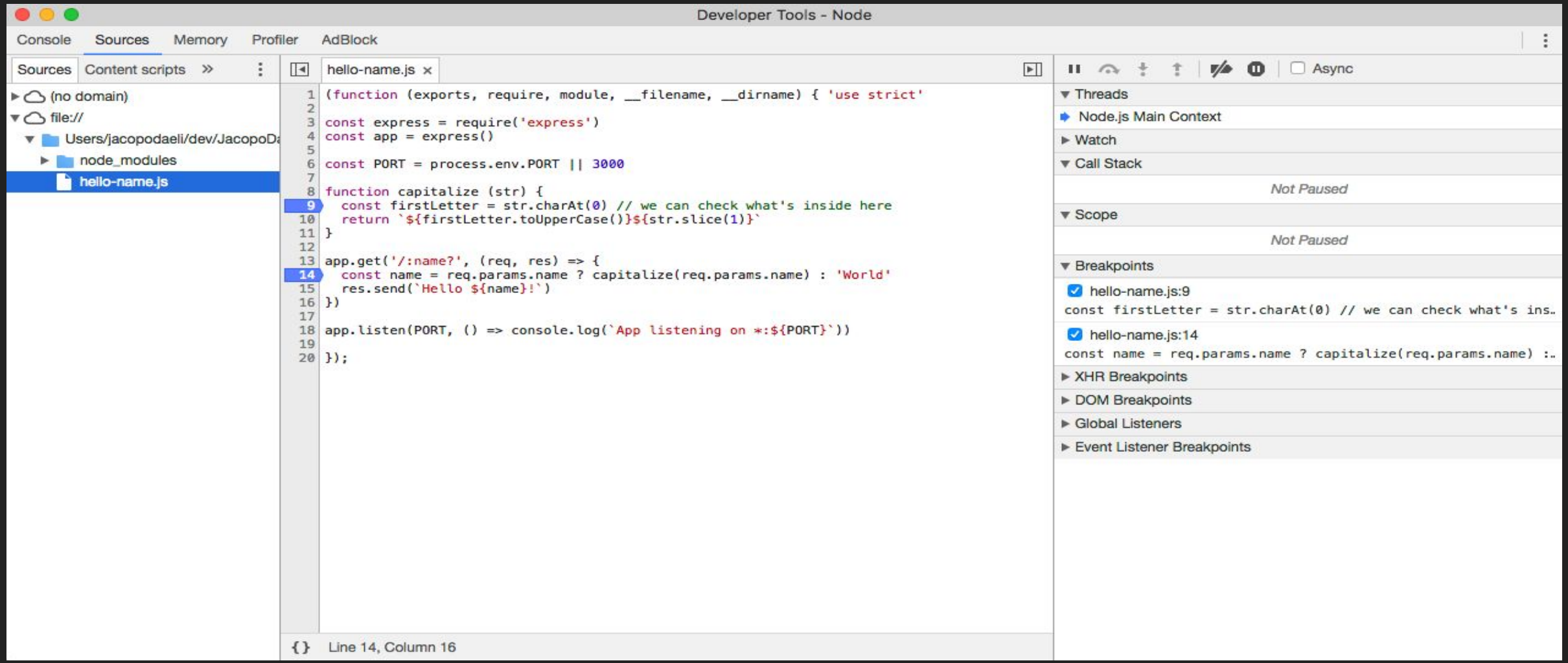
Debugging / отладка

```
// Отладка кода - пошаговая проверка для выявления ошибок

let one = 10
let two = 5

const array = [1, 2, 3, 4, 5]

for (let index = 0; index < array.length; index++) {
  one += array[index] // отследить через отладку изменения в цикле
}

one = one - two // узнать через отладку
```

Debugger VSCode



Google Chrome для отладки Node.js



Recursion

Recursion / рекурсия

Рекурсия — вызов функции из неё самой.

Также рекурсию можно объяснить, как один из способов построения абстрактной логики / мышления для выполнения задачи.

Мудрость из Интернета гласит:

«Для того чтобы понять рекурсию, надо сначала понять рекурсию»

Recursion / рекурсия, применение

Рекурсия применима, когда задача:

- делится на несколько простых действий
- эти действия нужно повторить неизвестное заранее количество раз

Два основных условия:

- базовый случай
- шаг (рекурсивный случай)

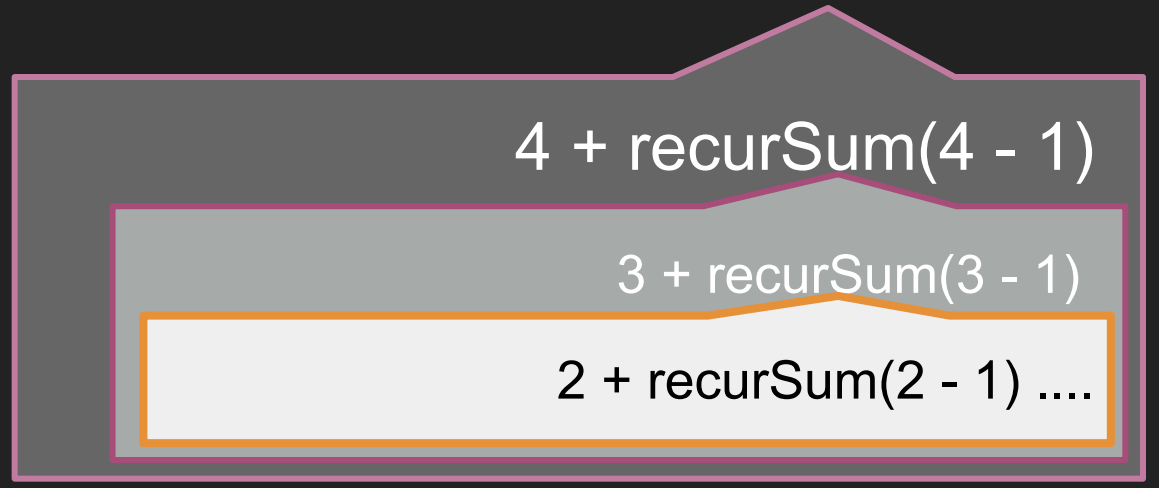
Рекурсия без базового случая

```
let i = 0

function infinityRecursion() {
  console.log(i += 1)
  infinityRecursion()
}

infinityRecursion() // RangeError: Maximum call stack size exceeded
```

Рекурсия с базовым случаем

```
function recurSum(n) {
 // базовый случай
 if (n === 1) {
   return n;
 }

 // рекурсивный случай
 return n + recurSum(n - 1);
}

recurSum(3); // 1 + 2 + 3
```

```
recurSum(5) = 5 + recurSum(5 - 1)
```



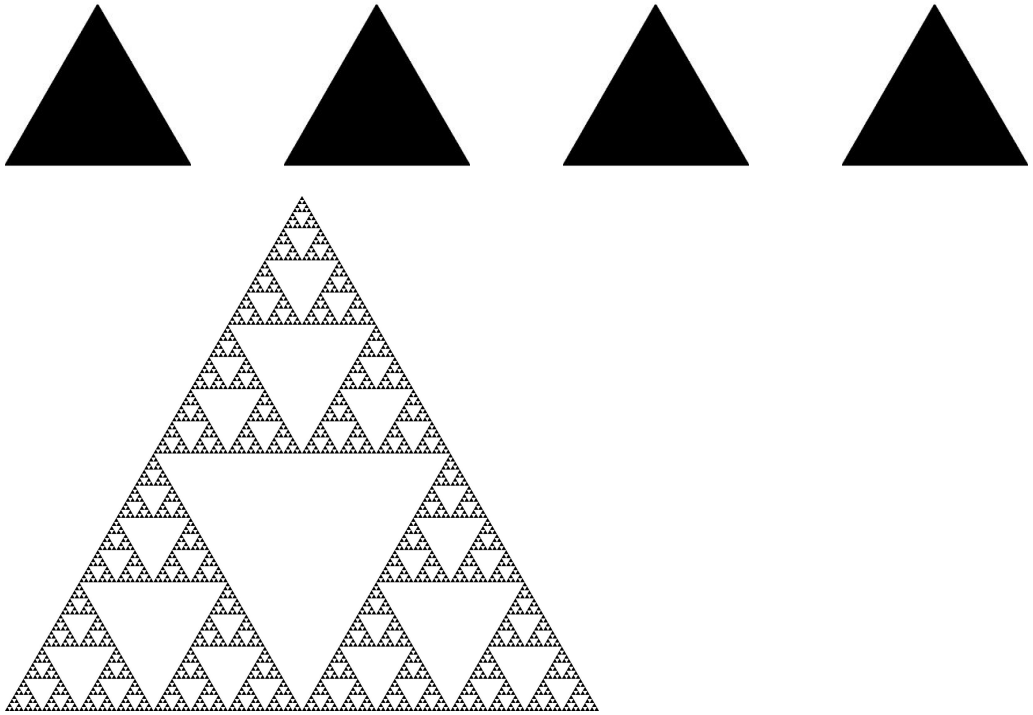
Рекурсивный счётчик обратного отсчёта

```
function countdown(num) {
  console.log(`Осталось: ${num} секунд`);

  if (num <= 0) {
    return console.log(num); // базовый случай
  } else {
    countdown(num - 1); // рекурсивный случай
  }
}

countdown(15)
```

Итерация

- Повторение, но не вызов самого себя
- Например, цикл for

Итерация vs. рекурсия



Algorithms

Алгоритм — это...

«Конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи.»

(определение из Википедии)

Поисковые алгоритмы

Линейный поиск

Алгоритм по очереди сравнивает элементы заданного массива с искомым элементом.

Если искомый элемент найден, алгоритм прекращает работу — поиск прошёл успешно.

Если весь массив проверен, но требуемый элемент не найден, это считается неудачным поиском.

Бинарный поиск

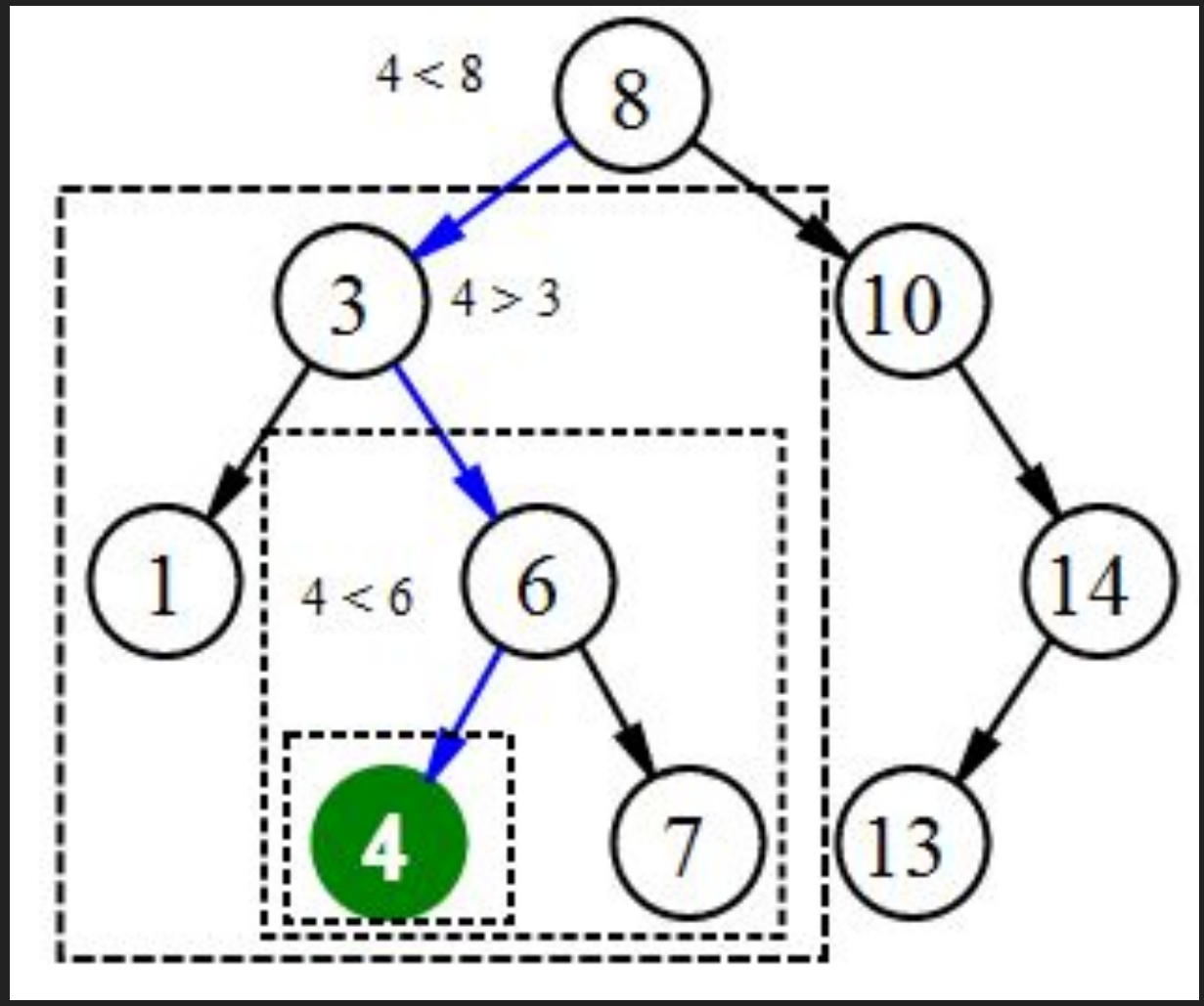
Выполняется по **отсортированному массиву**.

Алгоритм выбирает центральный элемент массива и сравнивает его с искомым. Если они равны, алгоритм возвращает это значение и прекращает работу.

Если центральное значение меньше искомого, можно игнорировать правую половину массива и повторить поиск.

Если оно больше искомого, можно игнорировать левую половину массива и повторить алгоритм.

Бинарный поиск в картинке



Алгоритмы сортировки

Алгоритм: сортировка пузырьком

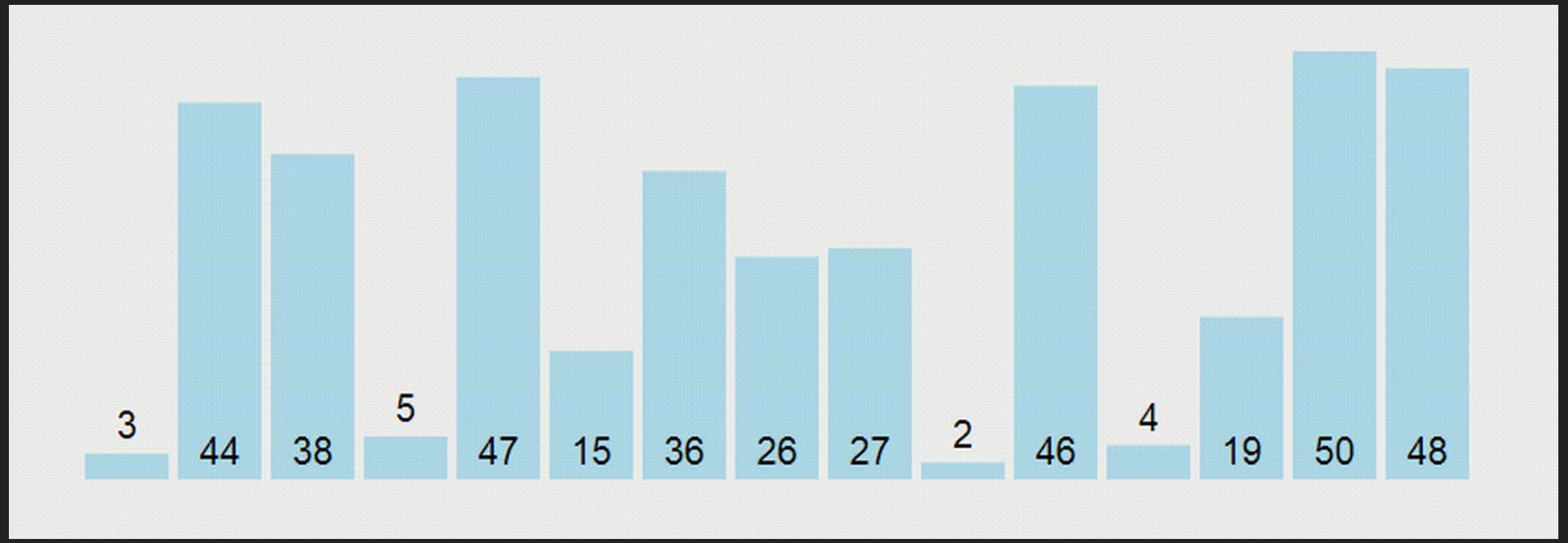
Алгоритм состоит из повторяющихся проходов по сортируемому массиву.

За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

Сортировка пузырьком



Визуализация в виде танцев: https://www.youtube.com/watch?v=lyZQPjUT5B4

Сортировка пузырьком


```
const arr = [5, 2, 1, 3, 9, 0, 4, 6, 8, 7];

for (let i = 0; i < arr.length; i += 1) {
  for (let j = 0; j < arr.length - i; j += 1) {
    if (arr[j] > arr[j + 1]) {
      const temp = arr[j];
      arr[j] = arr[j + 1];
      arr[j + 1] = temp;
    }
  }
}
```

Алгоритм: быстрая сортировка / quicksort

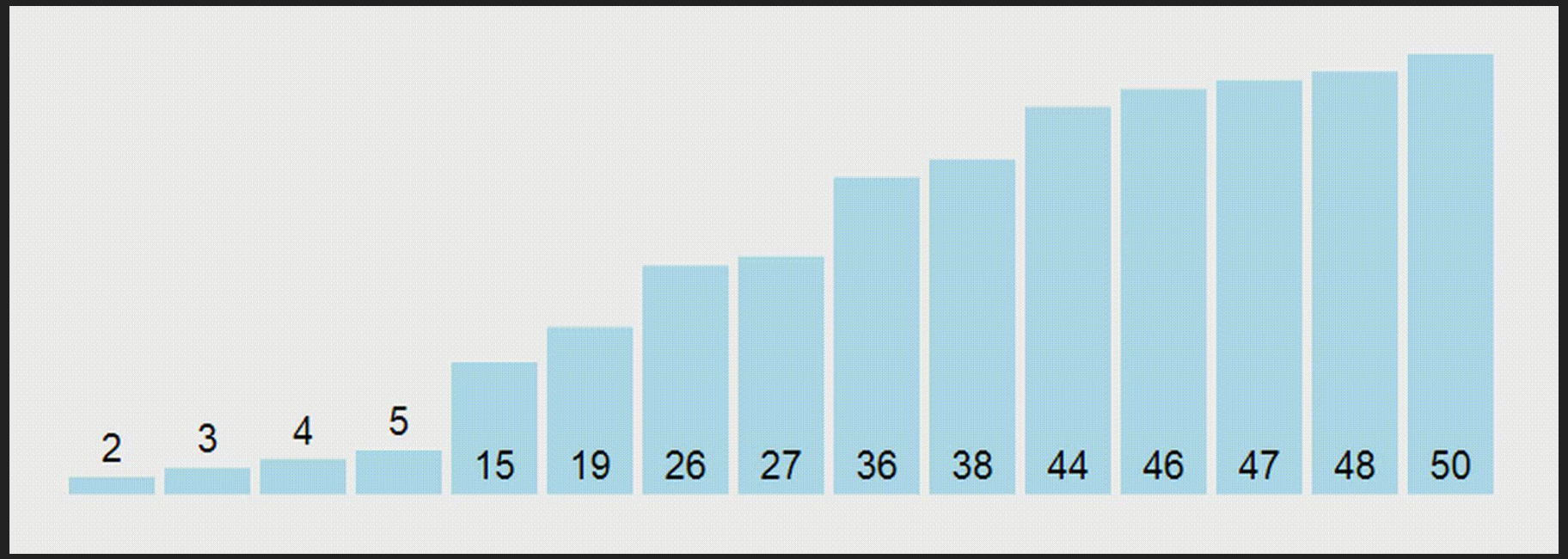
В начале выбирается “опорный” элемент массива. Это может быть любое число, но от выбора этого элемента сильно зависит эффективность алгоритма.

Если нам известна медиана, то лучше выбирать элемент, который как можно ближе к медиане. В нашей реализации алгоритма, мы будем брать самый левый элемент, который в результате займет свое место.

Элементы в массиве делятся на две части: слева те кто меньше опорного элемента, справа те кто больше. Таким образом опорный элемент занимает свое место и больше никуда не двигается.

Для левого и правого массива действия повторяются рекурсивно.

Быстрая сортировка / quicksort



Визуализация в виде танцев: https://www.youtube.com/watch?v=ywWBy6J5gz8

Быстрая сортировка / quicksort: слайд 1

```
const arr = [15, 4, 10, 100, 2, 34, 6, 8];

function quickSort(items, left, right) {
 let index;
 if (items.length > 1) {
   index = partition(items, left, right);
   if (left < index - 1) {
     quickSort(items, left, index - 1);
   }
   if (index < right) {
     quickSort(items, index, right);
   }
 }
 return items;
}

quickSort(arr, 0, arr.length - 1);
```

Быстрая сортировка / quicksort: слайд 2

```
function partition(items, left, right) {
  const pivot = items[Math.floor((right + left) / 2)];
  let i = left;
  let j = right;

  while (i <= j) {
    while (items[i] < pivot) i += 1
    while (items[j] > pivot) j -= 1
    if (i <= j) {
      const temp = items[i];
      items[i] = items[j];
      items[j] = temp;
      i += 1;
      j -= 1;
    }
  }
  return i;
}
```