ОГЛАВЛЕНИЕ

Разбор домашнего задания

Level 1

Дано натуральное число N. Выведите слово **YES**, если число **N** является точной степенью двойки, или слово **NO** в противном случае.
Операцией возведения в степень пользоваться нельзя!

Ввод	Вывод
8	YES
3	NO

Дано натуральное число **N**. Вычислите сумму его цифр.
При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

Ввод	Вывод
179	17

Level 2

Напишите рекурсивный метод, который выводит на экран числа Фибоначчи до **N**-ого элемента.

Напишите рекурсивный метод, который проверяет, является ли строка палиндромом.

Решение

Level 1: Задача с степенью двойки

Задача с степенью двойки на **java**	Задача с степенью двойки на **java script**
``java import java.util.Scanner; public class Main { public static void main(String[] args) { Scanner input = new Scanner(System.in); int n = input.nextInt(); if (isPowerOfTwo(n)) { System.out.println("YES"); } else { System.out.println("NO"); } } public static boolean isPowerOfTwo(int n) { if (n == 1) { return true; } else if (n % 2 == 0 && n != 0) { return isPowerOfTwo(n / 2); } else { return false; } } } ```	```javascript function isPowerOfTwo(n) { if (n === 1) { return true; } else if (n % 2 === 0 && n !== 0) { return isPowerOfTwo(n / 2); } else { return false; } } const n = prompt("Введите число: "); if (isPowerOfTwo(n)) { console.log("YES"); } else { console.log("NO"); } ```
Код на Java, который проверяет, является ли число N точной степенью двойки, с использованием рекурсии. Здесь функция isPowerOfTwo вызывает саму себя рекурсивно до тех пор, пока число не станет равным 1 или не окажется нечетным. Если число равно 1, то оно является точной степенью двойки, и функция возвращает true. В противном случае, если число нечётное или равно 0, то оно не является точной степенью двойки, и функция возвращает false.	Код на JavaScript, который проверяет, является ли число N точной степенью двойки, с использованием рекурсии. Здесь функция isPowerOfTwo вызывает саму себя рекурсивно до тех пор, пока число не станет равным 1 или не окажется нечетным. Если число равно 1, то оно является точной степенью двойки, и функция возвращает true. В противном случае, если число нечётное или равно 0, то оно не является точной степенью двойки, и функция возвращает false.

Level 1: Задача с суммой цифр в натуральном числе

Задача с суммой цифр в натуральном числе на **java**	Задача с суммой цифр в натуральном числе на **java script**
``java public static int sumDigits(int n) { if (n < 10) { // базовый случай: если число однозначное return n; } else { // рекурсивный случай: вычисляем сумму цифр числа, деля его на 10 return n % 10 + sumDigits(n / 10); } } int n = 123456; int sum = sumDigits(n); System.out.println(sum); // выводит 21 ```	```javascript function sumDigits(n) { if (n < 10) { // базовый случай: если число однозначное return n; } else { // рекурсивный случай: вычисляем сумму цифр числа, деля его на 10 return n % 10 + sumDigits(Math.floor(n / 10)); } } const n = 123456; const sum = sumDigits(n); console.log(sum); // выводит 21 ```
Кода на Java, который рекурсивно вычисляет сумму цифр натурального числа без использования строк, списков, массивов и циклов. В этом коде мы используем рекурсию для вычисления суммы цифр числа n. Если число n однозначное (меньше 10), мы просто возвращаем его. В противном случае мы берем остаток от деления n на 10 (это дает нам последнюю цифру числа) и добавляем его к сумме цифр оставшейся части числа (путем рекурсивного вызова sumDigits(n / 10)).	Кода на JavaScript, который рекурсивно вычисляет сумму цифр натурального числа без использования строк, списков, массивов и циклов. В этом коде мы используем рекурсию для вычисления суммы цифр числа n. Если число n однозначное (меньше 10), мы просто возвращаем его. В противном случае мы берем остаток от деления n на 10 (это дает нам последнюю цифру числа) и добавляем его к сумме цифр оставшейся части числа (путем рекурсивного вызова sumDigits(n / 10)).

Level 2: Задача про числа фибоначчи

Задача про числа фибоначчи на **java**	Задача про числа фибоначчи на **java script**
``java public static void main() { int n = 10; printFibonacci(n); // Выводит: 0 1 1 2 3 5 8 13 21 34 } public static void printFibonacci(int n) { if (n < 1) { // базовый случай: если N меньше 1, ничего не выводим return; } else if (n == 1) { // базовый случай: если N равно 1, выводим первое число Фибоначчи System.out.print("0 "); } else if (n == 2) { // базовый случай: если N равно 2, выводим первые два числа Фибоначчи System.out.print("0 1 "); } else { // рекурсивный случай: выводим следующее число Фибоначчи и рекурсивно вызываем этот же метод для следующего числа printFibonacci(n - 1); // рекурсивный вызов для предыдущего числа int fib = fibonacci(n - 1); // вычисляем текущее число Фибоначчи System.out.print(fib + " "); } } public static int fibonacci(int n) { if (n < 2) { // базовый случай: первые два числа Фибоначчи - 0 и 1 return n; } else { // рекурсивный случай: вычисляем число Фибоначчи, складывая два предыдущих числа return fibonacci(n - 1) + fibonacci(n - 2); } } ```	```javascript function printFibonacci(n) { if (n < 1) { return; // базовый случай: если N меньше 1, ничего не делаем } else if (n === 1) { console.log('0 '); // базовый случай: если N равно 1, выводим первое число Фибоначчи } else if (n === 2) { console.log('0 1 '); // базовый случай: если N равно 2, выводим первые два числа Фибоначчи } else { printFibonacci(n - 1); // рекурсивный вызов для предыдущего числа let fib = fibonacci(n - 1); // вычисляем текущее число Фибоначчи console.log(fib + ' '); } } function fibonacci(n) { if (n < 2) { return n; // базовый случай: первые два числа Фибоначчи - 0 и 1 } else { return fibonacci(n - 1) + fibonacci(n - 2); // рекурсивный случай: вычисляем число Фибоначчи, складывая два предыдущих числа } } let n = 10; console.log(`Fibonacci sequence up to ${n}:`); printFibonacci(n); // Выводит: 0 1 1 2 3 5 8 13 21 34 ```
Код на Java, который рекурсивно выводит на экран числа Фибоначчи до N-го элемента. В этом коде мы используем рекурсию для вывода на экран чисел Фибоначчи до N-го элемента. Если n меньше 1, мы ничего не делаем. Если n равно 1, мы выводим первое число Фибоначчи (0). Если n равно 2, мы выводим первые два числа Фибоначчи (0 и 1). В противном случае мы сначала рекурсивно вызываем этот же метод для предыдущего числа (N-1), а затем вычисляем и выводим текущее число Фибоначчи. Для вычисления чисел Фибоначчи также можно использовать рекурсивный метод fibonacci(), который вычисляет число Фибоначчи для заданного индекса.	Рекурсивная функция на JavaScript, которая выводит на экран числа Фибоначчи до N-го элемента. Здесь мы определили две функции: printFibonacci и fibonacci. Функция printFibonacci выводит на экран числа Фибоначчи до N-го элемента, используя рекурсию. Если n меньше 1, мы ничего не делаем. Если n равно 1, мы выводим первое число Фибоначчи (0). Если n равно 2, мы выводим первые два числа Фибоначчи (0 и 1). В противном случае мы сначала рекурсивно вызываем эту же функцию для предыдущего числа (N-1), а затем вычисляем и выводим текущее число Фибоначчи. Функция fibonacci вычисляет число Фибоначчи для заданного индекса, также используя рекурсию.

Level 2: Задача про строку палиндромом.

Задача про строку палиндромом на **java**	Задача про строку палиндромом на **java script**
``java public static void main(String[] args) { String str = "racecar"; if (isPalindrome(str)) { System.out.println(str + " is a palindrome"); } else { System.out.println(str + " is not a palindrome"); } // Выводит: "racecar is a palindrome" } public static boolean isPalindrome(String str) { // базовый случай: если строка пустая или состоит из одного символа, она является палиндромом if (str.length() <= 1) { return true; } // рекурсивный случай: если первый и последний символы совпадают, проверяем внутреннюю подстроку if (str.charAt(0) == str.charAt(str.length() - 1)) { return isPalindrome(str.substring(1, str.length() - 1)); } // иначе строка не является палиндромом return false; } ```	```javascript function isPalindrome(str) { // базовый случай: если строка пустая или состоит из одного символа, она является палиндромом if (str.length <= 1) { return true; } // рекурсивный случай: если первый и последний символы совпадают, проверяем внутреннюю подстроку if (str[0] === str[str.length - 1]) { return isPalindrome(str.slice(1, -1)); } // иначе строка не является палиндромом return false; } let str = "racecar"; if (isPalindrome(str)) { console.log(str + " is a palindrome"); } else { console.log(str + " is not a palindrome"); } // Выводит: "racecar is a palindrome" ```
Рекурсивная функция на Java, которая проверяет, является ли строка палиндромом. Здесь мы определили статическую функцию isPalindrome, которая принимает строку str в качестве аргумента и возвращает true, если строка является палиндромом, и false в противном случае. В первом условии мы проверяем базовый случай: если строка состоит из одного символа или меньше, она является палиндромом. Во втором условии мы проверяем, что первый и последний символы строки совпадают. Если это так, мы вызываем эту же функцию для внутренней подстроки (без первого и последнего символа) и возвращаем результат. В противном случае строка не является палиндромом, и мы возвращаем false.	Рекурсивная функция на JavaScript, которая проверяет, является ли строка палиндромом. Здесь мы определили статическую функцию isPalindrome, которая принимает строку str в качестве аргумента и возвращает true, если строка является палиндромом, и false в противном случае. В первом условии мы проверяем базовый случай: если строка состоит из одного символа или меньше, она является палиндромом. Во втором условии мы проверяем, что первый и последний символы строки совпадают. Если это так, мы вызываем эту же функцию для внутренней подстроки (без первого и последнего символа) и возвращаем результат. В противном случае строка не является палиндромом, и мы возвращаем false.

Техника Разделяй и властвуй

Техника **"Разделяй и властвуй"** (англ. "divide and conquer") - это метод решения задач, который заключается в разбиении сложной задачи на более простые, которые решаются независимо друг от друга, а затем объединении их решений в конечный результат.

Этот метод применяется в различных областях, включая математику, программирование, инженерию и управление проектами. В программировании техника "Разделяй и властвуй" применяется для решения сложных задач, которые могут быть разделены на несколько меньших подзадач.

Процесс решения задачи

Процесс решения задачи с помощью техники "Разделяй и властвуй" обычно включает три шага:

1. Divide Разделение задачи на несколько подзадач: сложная задача разбивается на несколько более простых задач, которые могут быть решены независимо друг от друга.
2. Conquer рекурсивно вызываем подзадачи до тех пор, пока они не будут решены. Решение каждой подзадачи: каждая подзадача решается независимо, используя наиболее подходящий метод.
3. Combine Объединение решений подзадач: результаты решения каждой подзадачи объединяются в конечный результат, который является решением исходной задачи.

Например, если вам нужно отсортировать большой **массив** чисел, вы можете разделить массив на несколько более мелких массивов и отсортировать их независимо. Затем вы можете объединить отсортированные подмассивы в отсортированный полный массив.

Таким образом, техника "Разделяй и властвуй" является мощным инструментом для решения сложных задач, которые могут быть разделены на более простые подзадачи.

Пример из жизни

 Алгоритм разделяй и властвуй может применяться в быту для решения различных задач. Одним из примеров может быть приготовление еды. Рассмотрим, например, задачу приготовления пиццы.

Сначала вы можете разделить задачу на несколько подзадач, например:

1. Приготовление теста для пиццы
2. Приготовление соуса для пиццы
3. Нарезка и приготовление ингредиентов для начинки пиццы
4. Сборка пиццы

Затем вы можете решать каждую из этих подзадач по отдельности, используя подход разделяй и властвуй. Например, для приготовления теста для пиццы вы можете разделить процесс на следующие шаги:

1. Подготовка ингредиентов: мука, дрожжи, сахар, соль, масло, вода
2. Смешивание муки, дрожжей, сахара и соли
3. Добавление масла и воды и замешивание теста
4. Оставление теста на определенное время для подъема
5. Раскатывание теста в нужную форму

Вы можете решать каждый из этих шагов по отдельности, разделяя задачу на более мелкие подзадачи и решая их по отдельности. Затем вы можете объединить результаты каждого шага и получить готовое тесто для пиццы.

Алгоритмы «разделяй и властвуй»

- **Quick Sort** алгоритм сортировки. Алгоритм выбирает опорный элемент и переупорядочивает элементы массива таким образом, чтобы все элементы, меньшие, чем выбранный опорный элемент, переместились в левую часть опорного элемента, а все большие элементы перемещались в правую сторону.

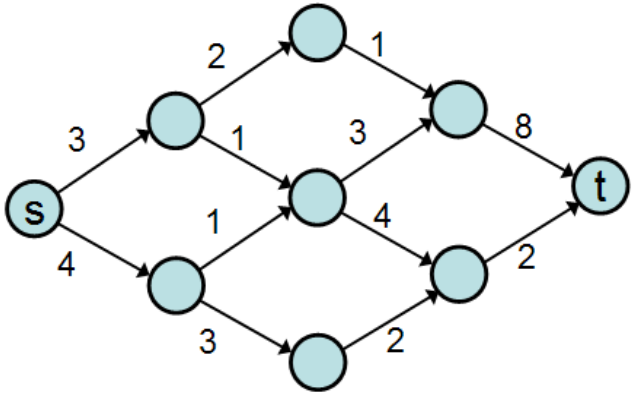
- **Merge Sort** также является алгоритмом сортировки. Алгоритм делит массив на две половины, рекурсивно сортирует их и, наконец, объединяет две отсортированные половины.

- **Closest Pair of Points** Задача состоит в том, чтобы найти ближайшую пару точек в наборе точек на плоскости xy. Задача может быть решена за время O(n^2) путем вычисления расстояний каждой пары точек и сравнения расстояний для поиска минимума. Алгоритм «разделяй и властвуй» решает проблему за время O(N log N).

- **Strassen's Algorithm** эффективный алгоритм умножения двух матриц. Простой метод умножения двух матриц требует 3 вложенных цикла и составляет O (n ^ 3). Алгоритм Штрассена умножает две матрицы за время O(n^2,8974).

Преимущества и недостатки

Преимущества алгоритма «разделяй и властвуй»	**Недостатки алгоритма «разделяй и властвуй»**
• Сложная проблема решается легко. • Делит задачу на подзадачи, поэтому ее можно решать параллельно, обеспечивая многопроцессорность. • Эффективно использует кэш-память, не занимая много места • Снижает временную сложность задачи	• Включает в решение рекурсию, которая иногда медленная • Эффективность зависит от реализации логики • Это может привести к сбою системы, если в рекурсии есть ошибки



Примеры

Get max and min element in array

Найти максимальный элемент в заданном массиве.

Ввод: `{40, 250, 80, 88, 240, 12, 148}`

Вывод:

Минимальное число в данном массиве: **12**

Максимальное число в данном массиве: **250**

Реализуем на уроке схему

Java script

```
/** Практика: наименьшее и наибольшее число */
function mainInit() {
    let arr = [70, 250, 50, 300, 1];
    console.log(getMaxElement(arr, 0));
    console.log(getMinElement(arr, 0));
}
function getMaxElement(arr, index) {
    let max;
    let length = arr.length;
    if (length > index) {
        max = getMaxElement(arr, index + 1);
        if (arr[index] > max)
```

Java

```
public static void main(String[] args) {
    int arr[] = {70, 250, 50, 300, 1};
    System.out.println(getMaxElement(arr, 0));
    System.out.println(getMinElement(arr, 0));
}
static int getMaxElement(int arr[], int index)
{
    int max;
    int length = arr.length;
    if (length > index)
    {
        max = getMaxElement(arr, index + 1);
        if (arr[index] > max)
            return arr[index];
        else
            return max;
    } else {
        return arr[index - 1];
    }
}
```

```
            return arr[index];
        else
            return max;
    } else {
        return arr[index - 1];
    }
}

function getMinElement(arr, index) {
    let min;
    let length = arr.length;
    if (length > index) {
        min = getMinElement(arr, index + 1);
        if (arr[index] < min)
            return arr[index];
        else
            return min;
    } else {
        return arr[index - 1];
    }
}
mainInit();
```

```
static int getMinElement(int arr[], int index)
{
    int min;
    int length = arr.length;
    if (length > index)
    {
        min = getMinElement(arr, index + 1);
        if (arr[index] < min)
            return arr[index];
        else
            return min;
    } else {
        return arr[index - 1];
    }
}
```

Quick Sort

Алгоритм быстрой сортировки – это один из самых быстрых существующих алгоритмов сортировки, который является примером стратегии “разделяй и властвуй”. Большинство готовых библиотек и методов по сортировке используют quick sort алгоритм как основу.

Кстати говоря, алгоритм быстрой сортировки как и алгоритм в худшем случае работает за время O(n^2)

что довольно медленно.

Но не торопитесь делать поспешные выводы. В среднем алгоритм быстрой сортировки выполняется за время O(n logn); причем время сортировки очень зависит от опорного элемента, о котором Вы узнаете далее.

Алгоритм быстрой сортировки – **это рекурсивный алгоритм.** Как уже было сказано выше quicksort использует стратегию “разделяй и властвуй”.

Ее суть заключается в том, что задача **разбивается на подзадачи** до тех пор, пока не будет элементарной единицы. Так и с алгоритмом: массив делится на несколько массивов, каждый из который сортируется по отдельности и потом объединяется в один массив.

Пошаговое описание работы алгоритма быстрой сортировки

1. Выбрать опорный (pivot) элемент из массива. Опорный элемент станет средним элементом.
2. Разделить массив на два подмассива: элементы, меньше опорного и элементы, больше опорного.
3. Рекурсивно применить сортировку к двум подмассивам.

В результате таких действий рекурсии, программа дойдет до того, что массивы будут делиться пока не будет один элемент, который потом и отсортируется.

Звучит немного запутанно и странно. Для наглядности посмотрите анимацию и картинки ниже.

Для просмотра

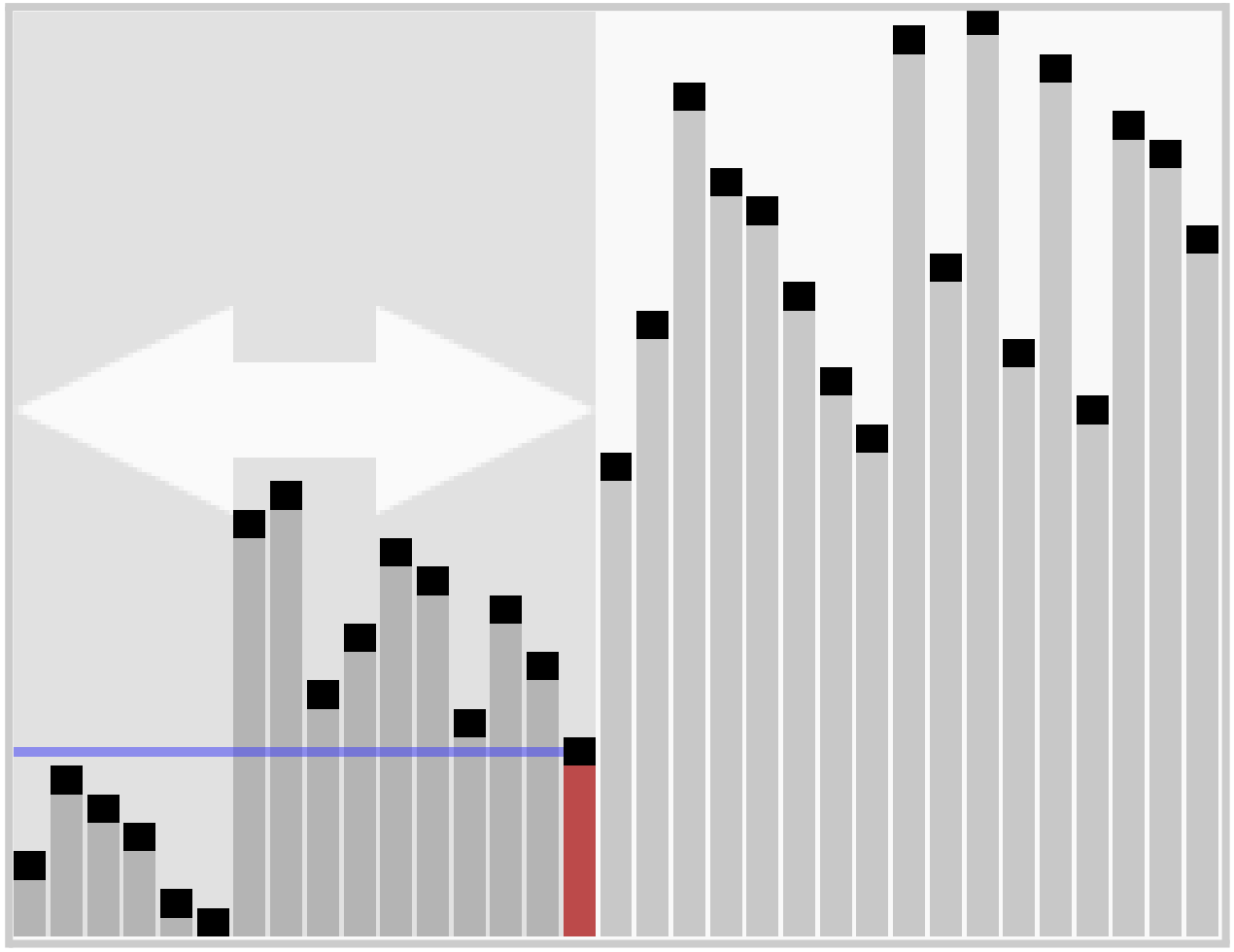
ССЫЛКА

ССЫЛКА

Для чтения

ССЫЛКА

ССЫЛКА

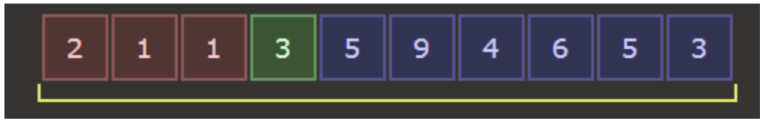


Анализ алгоритма быстрой сортировки

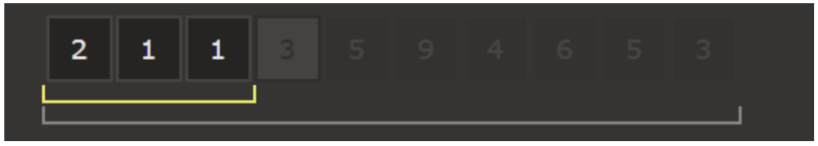
Подробный анализ алгоритма быстрой сортировки

Исходный массив

Выбор опорного элемента. В данном случае мы взяли первый элемент 3



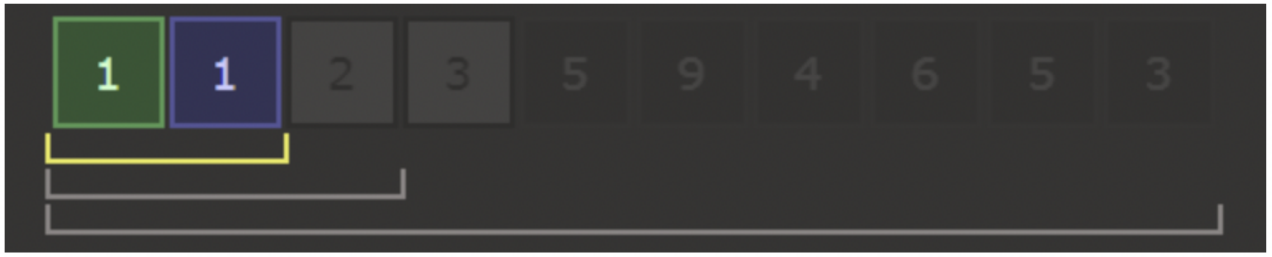
Разбиваем массив на подмассивы: те, что больше 3 и те, что меньше 3



Проделаем то же самое с левый подмассивом

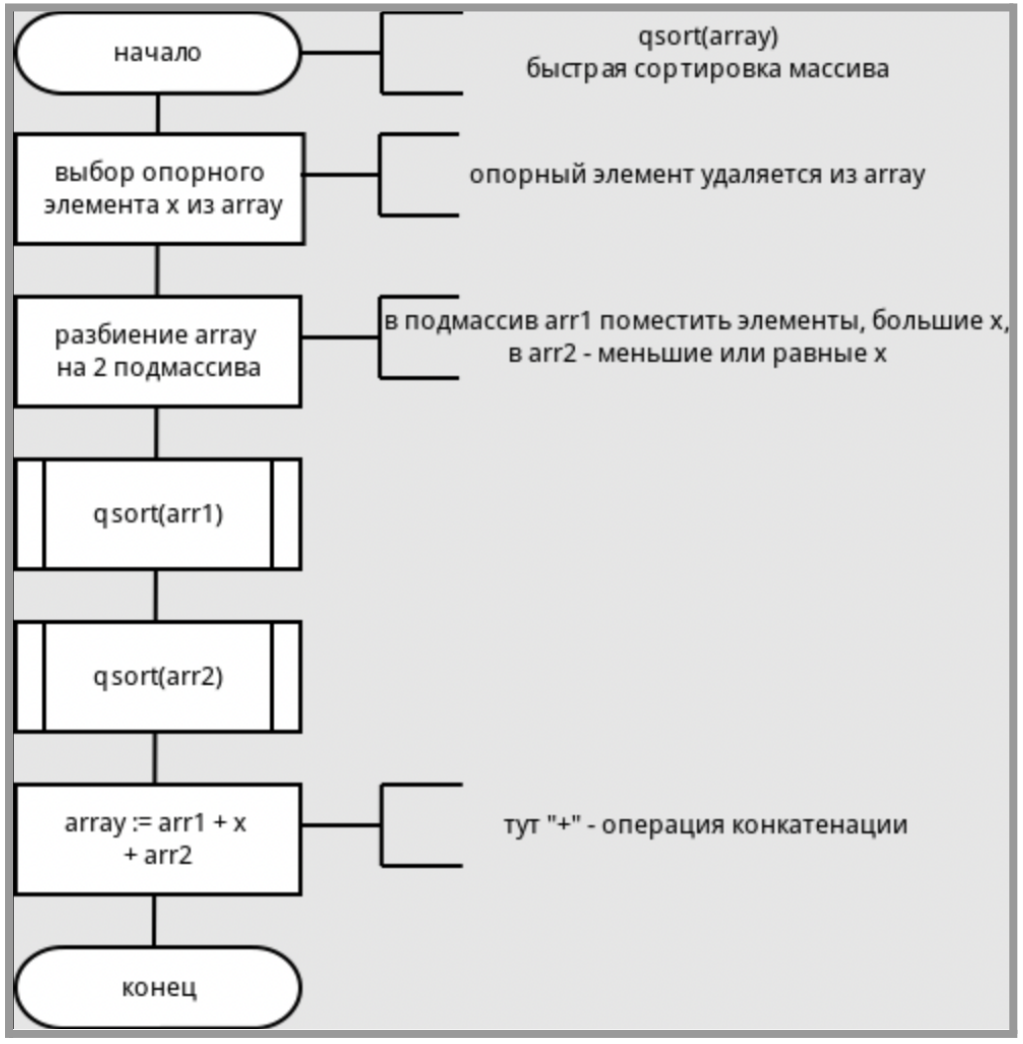
Выбор опорного элемента

Разбиение на подмассивы



Выбор опорного и разбиение на подмассивы. Не забываем, что мы проделываем эти шаги пока не будет один элемент в подмассиве.

Блок схема



Пример интерактив

https://app.diagrams.net/#G1mTv3S08oXRrrCwNqqvlOnWGG_iWJzlIj

Код

Java script

```
function quickSort(array, start, end) {
if (start < end) {
let indexPivot = partition(array, start, end);

quickSort(array, start, indexPivot - 1); // меньшие
значения слева
quickSort(array, indexPivot + 1, end); // большие значения
справа
}
}

function partition(array, start, end) {
let pivot = array[end]; // выбор опорного элемента
let indexPivot = start; // Индекс меньшего элемента и
указание правильной позиции

for (let i = start; i < end; i++) {
if (array[i] <= pivot) {
swap(array, i, indexPivot);
indexPivot++;
}
}
swap(array, end, indexPivot); // перемещение опорного
элемента на правильную позицию

return indexPivot;
}

function swap(array, first, second) {
let temp = array[first];
array[first] = array[second];
array[second] = temp;
}

let testArray = [39, 22, 2, 55, 6, 20];
```

Java

```
public class QuickSort {

   public static void main(String[] args) {
       int[] testArray = {39, 22, 2, 55, 6, 20};
       quickSort(testArray, 0, testArray.length - 1);
       for (int num : testArray) {
           System.out.print(num + " ");
       }
   }

   // Метод для вызова быстрой сортировки
   public static void quickSort(int[] array, int start, int
end) {
       if (start < end) {
           int indexPivot = partition(array, start, end);

           // Рекурсивный вызов для левой части массива
(меньшие значения)
           quickSort(array, start, indexPivot - 1);
           // Рекурсивный вызов для правой части массива
(большие значения)
           quickSort(array, indexPivot + 1, end);
       }
   }

   // Метод для разделения массива на части и выбора
опорного элемента (pivot)
   public static int partition(int[] array, int start, int
end) {
       int pivot = array[end]; // Выбор опорного элемента
       int indexPivot = start; // Индекс меньшего элемента,
указывает на правильную позицию

       for (int i = start; i < end; i++) {
           if (array[i] <= pivot) {
```

```
quickSort(testArray, 0, testArray.length - 1);
console.log(testArray);
```

```
                swap(array, i, indexPivot);
                indexPivot++;
            }
        }
        swap(array, end, indexPivot); // Перемещение
опорного элемента на правильную позицию

        return indexPivot;
    }

    // Метод для обмена элементов в массиве
    public static void swap(int[] array, int first, int
second) {
        int temp = array[first];
        array[first] = array[second];
        array[second] = temp;
    }
}
```

Пример бинарного поиска

Binary Search

Дан отсортированный массив arr[] из n элементов.
Напишите функцию для поиска заданного элемента x в arr[] и возврата индекса x в массиве.

Примеры:

Ввод: arr[] = {11, 22, 44, 50, 60, 86, 114, 140, 145, 190}, x = 114
Вывод: 6
Объяснение: Элемент x присутствует в индексе 6.

Ввод: arr[] = {1, 24, 30, 46, 60, 100, 120, 133, 270}, x = 114
Вывод: -1
Объяснение: Элемент x отсутствует в arr[].

Реализуем на уроке схему

Java script

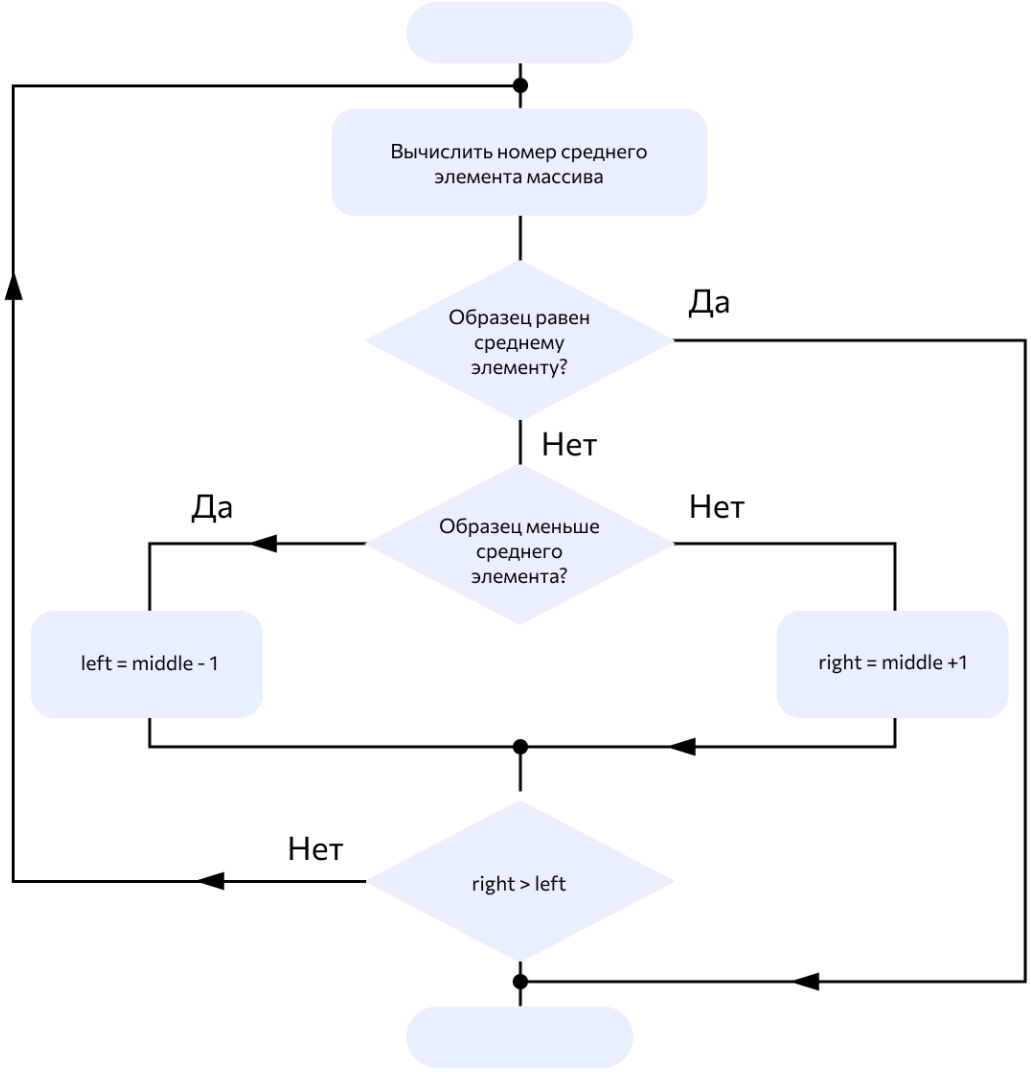
```javascript
function binarySearch(arr, x) {
 let left = 0;
 let right = arr.length - 1;

 while (left <= right) {
   const mid = Math.floor((left + right) / 2);

   if (arr[mid] === x) {
```

Java

```java
public class BinarySearch {
   // Функция для поиска элемента x в массиве arr[] и возврата его индекса
   // Возвращает -1, если элемент отсутствует в массиве
   public static int binarySearch(int[] arr, int x) {
       int left = 0, right = arr.length - 1;
       while (left <= right) {
           int mid = left + (right - left) / 2;

           // Проверяем, находится ли элемент в середине массива
           if (arr[mid] == x)
               return mid;
```

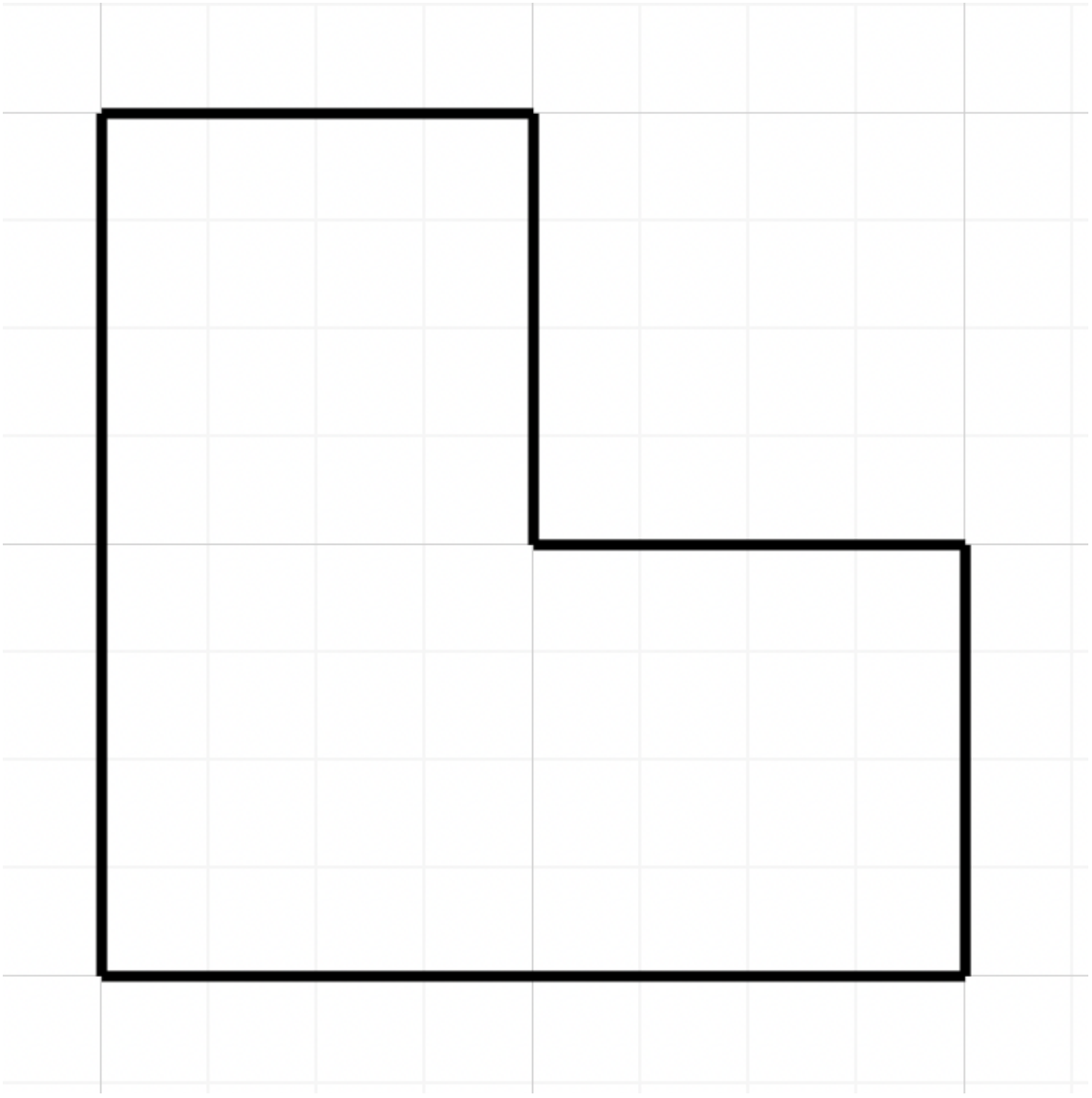
```
      return mid;
    } else if (arr[mid] < x) {
      left = mid + 1;
    } else {
      right = mid - 1;
    }
  }
  return -1;
}

// Тестовые примеры
const arr1 = [11, 22, 44, 50, 60, 86, 114, 140, 145,
190];
const x1 = 114;
console.log(`Индекс элемента ${x1} в массиве arr1:
${binarySearch(arr1, x1)}`);

const arr2 = [1, 24, 30, 46, 60, 100, 120, 133, 270];
const x2 = 114;
console.log(`Индекс элемента ${x2} в массиве arr2:
${binarySearch(arr2, x2)}`);
```

```
            // Если x больше, чем элемент в середине,
игнорируем левую половину
            if (arr[mid] < x)
                left = mid + 1;

                // Если x меньше, чем элемент в середине,
игнорируем правую половину
            else
                right = mid - 1;
        }

        // Если элемент не найден, возвращаем -1
        return -1;
    }


}
// Тестовые примеры
public static void main(String[] args) {
    int[] arr1 = {11, 22, 44, 50, 60, 86, 114, 140, 145,
190};
    int x1 = 114;
    System.out.println("Индекс элемента " + x1 + " в массиве
arr1: " + binarySearch(arr1, x1));

    int[] arr2 = {1, 24, 30, 46, 60, 100, 120, 133, 270};
    int x2 = 114;
    System.out.println("Индекс элемента " + x2 + " в массиве
arr2: " + binarySearch(arr2, x2));
}
```



Поиск алгоритма решения

#1	#2
Задача: У вас есть **50 пар** носков разных цветов. Как бы вы быстро найдете пару носков одного цвета, не заглядывая в каждую пару?	Задача: Задача с белыми и черными шляпами.



Домашнее задание

1. Очистить долги с предыдущих домашних заданий

2. Повторить пройденное

3. Составить блок схему следующей задачи:

Имея два отсортированных массива размера m и n соответственно, вам нужно найти элемент, который будет находиться на k-й позиции в конечном отсортированном массиве.

```
Массив 1 - 100 112 256 349 770
Массив 2 - 72 86 113 119 265 445 892
к = 7
Вывод : 256
```

4. Решите задачу: Расставьте в алфавитном порядке буквы. Разрешается использование техники Разделяй и властвуй. Полученные данные напечатайте.

👉👉👉 `На вход строка: "poiuytrewqlkjhgfdsamnbvcxz"`

На выходе должно быть: `ABCDEFGHIJKLMNOPQRSTUVWXYZ` (с большой буквы)