

ОГЛАВЛЕНИЕ

Разбор домашнего задания	2
Сортировка слиянием (Merge sort)	6
Сам алгоритм	7
Схема алгоритма	8
Для чтения	9
Пример	10
Пример кода	11
Разница между быстрой сортировкой и сортировкой слиянием	14
Блок схема	15
Вопросы на собеседовании	16
Поиск алгоритма решения	17
Домашнее задание	18

Разбор домашнего задания

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)

Решение задачи с алфавитом



JAVA

```
public static void main(String[] args) {
    String str = "poiuytrewqlkjhgfdsamnbvcxz";
    String sortedStr = sortAlphabet(str);
    System.out.println(sortedStr);
}

public static String sortAlphabet(String str)
{
    if (str.length() <= 1) {
        return str.toUpperCase();
    } else {
        String leftStr = str.substring(0,
str.length() / 2);
        String rightStr =
str.substring(str.length() / 2);
        String sortedLeftStr =
sortAlphabet(leftStr);
        String sortedRightStr =
sortAlphabet(rightStr);
        return merge(sortedLeftStr,
sortedRightStr);
    }
}

public static String merge(String leftStr,
String rightStr) {
    StringBuilder mergedStr = new
StringBuilder();
    int i = 0;
```

JAVA SCRIPT

```
function sortAlphabet(str) {
    if (str.length <= 1) {
        return str.toUpperCase();
    } else {
        const leftStr = str.slice(0,
Math.floor(str.length / 2));
        const rightStr =
str.slice(Math.floor(str.length / 2));
        const sortedLeftStr = sortAlphabet(leftStr);
        const sortedRightStr = sortAlphabet(rightStr);
        return merge(sortedLeftStr, sortedRightStr);
    }
}

function merge(leftStr, rightStr) {
    let mergedStr = "";
    let i = 0;
    let j = 0;
    while (i < leftStr.length && j < rightStr.length) {
```

```
int j = 0;
while (i < leftStr.length() && j <
rightStr.length()) {
    if (leftStr.charAt(i) <
rightStr.charAt(j)) {

mergedStr.append(leftStr.charAt(i));
        i++;
    } else {

mergedStr.append(rightStr.charAt(j));
        j++;
    }
}
while (i < leftStr.length()) {
    mergedStr.append(leftStr.charAt(i));
    i++;
}
while (j < rightStr.length()) {
    mergedStr.append(rightStr.charAt(j));
    j++;
}
return mergedStr.toString().toUpperCase();
}
```

Объяснение для java:

```
if (leftStr.charCodeAt(i) <
rightStr.charCodeAt(j)) {
    mergedStr += leftStr.charAt(i);
    i++;
} else {
    mergedStr += rightStr.charAt(j);
    j++;
}
}
while (i < leftStr.length) {
    mergedStr += leftStr.charAt(i);
    i++;
}
while (j < rightStr.length) {
    mergedStr += rightStr.charAt(j);
    j++;
}
return mergedStr.toUpperCase();
}
```

```
const str = "poiuytrewqlkjhgfdsamnbvcxz";
const sortedStr = sortAlphabet(str);
console.log(sortedStr);
```

Объяснение для java script:

1. В методе `main` мы создаем объект класса `AlphabetSorter`, чтобы вызвать его метод `sortAlphabet` и передать ему строку, которую нужно отсортировать в алфавитном порядке.
2. В методе `sortAlphabet` мы используем метод "Разделяй и властвуй". Если длина строки меньше или равна 1, то мы просто возвращаем ее в верхнем регистре. В противном случае мы делим строку на две части и рекурсивно вызываем `sortAlphabet` для каждой части. Затем мы объединяем отсортированные части с помощью метода `merge`.
3. В методе `merge` мы сравниваем буквы в левой и правой частях строки. Если левая буква меньше, то мы добавляем ее в новую строку и переходим к следующей букве в левой части. Если правая буква меньше, то мы добавляем ее в новую строку и переходим к следующей букве в правой части. Мы продолжаем делать это до тех пор, пока не достигнем конца хотя бы одной из частей. Затем мы добавляем оставшиеся буквы из левой и правой частей в новую строку и возвращаем эту строку в верхнем регистре.

Результатом работы программы будет строка
"ABCDEFGHIJKLMNOPQRSTUVWXYZ".

1. Мы определяем функцию `sortAlphabet`, которая принимает строку `str`. Если длина строки меньше или равна 1, то мы возвращаем ее в верхнем регистре. В противном случае мы делим строку на две части и рекурсивно вызываем `sortAlphabet` для каждой части. Затем мы объединяем отсортированные части с помощью функции `merge`.
2. Мы определяем функцию `merge`, которая принимает две строки `leftStr` и `rightStr`. Мы сравниваем буквы в левой и правой частях строки. Если левая буква меньше, то мы добавляем ее в новую строку и переходим к следующей букве в левой части. Если правая буква меньше, то мы добавляем ее в новую строку и переходим к следующей букве в правой части. Мы продолжаем делать это до тех пор, пока не достигнем конца хотя бы одной из частей. Затем мы добавляем оставшиеся буквы из левой и правой частей в новую строку и возвращаем эту строку в верхнем регистре.
3. Мы создаем переменную `str`, которая содержит строку, которую нужно отсортировать в алфавитном порядке, и вызываем функцию `sortAlphabet` с этой строкой. Результат сохраняем в переменной `sortedStr`. Затем мы выводим результат с помощью функции `console.log`.

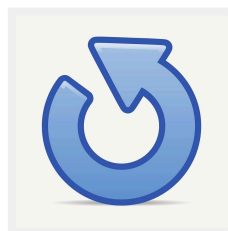
Результатом работы программы будет строка
"ABCDEFGHIJKLMNOPQRSTUVWXYZ".

Сортировка слиянием (Merge sort)

Подобно быстрой сортировке (Quick sort), сортировка слиянием представляет собой алгоритм «разделяй и властвуй».

В парадигме «разделяй и властвуй» проблема разбивается на более **мелкие задачи**, каждая из которых сохраняет все свойства более крупной проблемы, кроме своего размера.

Алгоритм делит входной массив на **две половины**, вызывает себя рекурсивно для этих двух половин, а затем объединяет две отсортированные половины.



1. Сортировка	2. Слияние
Разделяем массив, пока не достигнем length = 1	Слияние отсортированных элементов.

Сам алгоритм



Временная сложность алгоритма сортировки слиянием одинакова в лучшем, среднем и худшем случаях и равна **$O(n \cdot \log(n))$**

Шаг 1 : Разделите несортированный список на n подсписков; каждый подсписок имеет один элемент (который сортируется по одному элементу)

Шаг 2 : Объедините два списка одновременно. При объединении сравните элементы для двух подсписков и подготовьте новый отсортированный список.

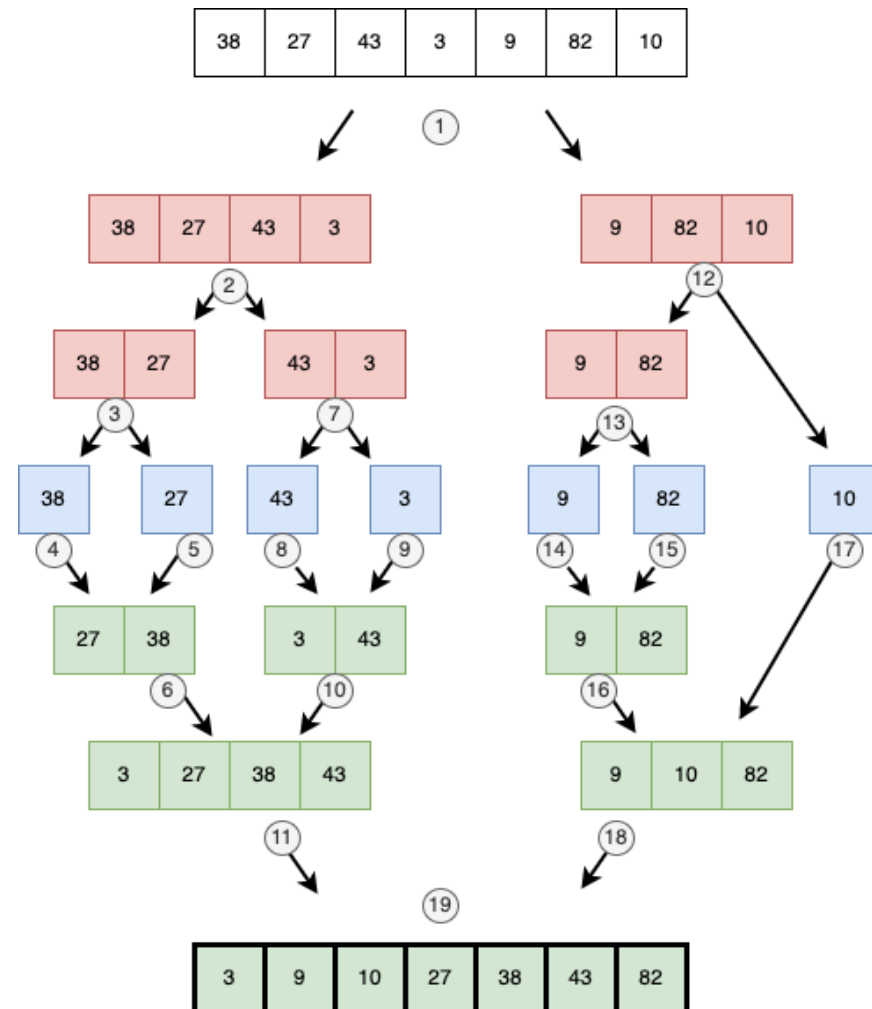
Шаг 3 : Выполните шаг 2 для всех пар подсписков.

Шаг 4 : Повторяйте шаги со 2 по 3 для нового списка подсписков, пока у нас не будет только один список.

Схема алгоритма

- #1** Делим массив по середине
- #2** Если делится еще делим по середине
- #3** Если делится еще делим по середине
- #4 #5** Берем цифру 38 и 27 сравниваем и сортируем
- #6** Складываем отсортированное (merge)
- #7** Если делится еще делим по середине
- #8 #9** Берем цифру 43 и 3 сравниваем и сортируем
- #10** Складываем отсортированное левой части (merge)
- #11** Складываем отсортированное в результат (merge)
- #12** Если делится еще делим по середине
- #13** Если делится еще делим по середине
- #14 #15** Берем цифру 9 и 82 сравниваем и сортируем
- #16** Складываем отсортированное правой (merge)
- #17** Складываем отсортированное правой (merge)
- #18** Складываем отсортированное в результат (merge)
- #19** Складываем отсортированное в результат (merge)

[Diagrams.net source link](https://www.diagrams.net)



Для чтения



https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC

Для джавистов:

<https://javarush.com/groups/posts/2202-sortirovka-slijaniem-merge-sort>

Создатель алгоритма: **Джон фон Нейман** [WIKI](#)

(придумал алгоритм в 1945 году на 45-ом году жизни)

👁 Для просмотра: <https://www.youtube.com/watch?v=k2AqGongii0>

https://www.youtube.com/watch?v=dENca26N6V4&list=PLOmdoKois7_FK-ySGwHBkltzB11snW7KQ&index=3



Summary

Конечная цель алгоритма сортировки Merge sort - разделить массив данных на две равные части, отсортировать каждую из них отдельно, а затем объединить их в отсортированный массив.

Пример

ШАГ 1: Сначала проверьте, меньше ли левый индекс массива, чем правый индекс, если да, то вычислить его среднюю точку.

ШАГ 2: Теперь, как мы уже знаем, сортировка слиянием сначала итеративно делит весь массив на равные половины, если не достигнуты атомарные значения.

ШАГ 3: Здесь мы видим, что массив из 7 элементов разбит на два массива размером 4 и 3 соответственно.

ШАГ 4: Теперь снова найдите, что левый индекс меньше правого индекса для обоих массивов, если найдено да, затем снова вычислите средние точки для обоих массивов.

ШАГ 5: Теперь разделите эти два массива еще на две половины, пока не будут достигнуты атомарные единицы массива и дальнейшее деление станет невозможным.

ШАГ 6: После разделения массива на наименьшие единицы снова начните объединять элементы на основе сравнения размеров элементов.

Пример кода

JAVA	JAVA SCRIPT
<pre>public static void main(String[] args) { /* mergeSort(arr) - function that sorts arr[leftIndex..rightIndex] using merge() merge(arr, leftArr, rightArr) - Merges two sub arrays of arr[]. */ int[] arr = {15, 21, 13, 5, 10, 7}; System.out.println("Given Array"); System.out.println(Arrays.toString(arr)); mergeSort(arr); System.out.println("Sorted array"); System.out.println(Arrays.toString(arr)); } private static void mergeSort(int[] arr) { int lengthArr = arr.length; // длина массива if (lengthArr == 1) { // условие выхода return; } int mid = lengthArr / 2; int[] leftArr = new int[mid]; // левый подмассив* int[] rightArr = new int[lengthArr - mid]; // правый подмассив* // копируем элементы из массива в подмассивы for (int i = 0; i < mid; ++i) { leftArr[i] = arr[i]; } for (int j = 0; j < lengthArr - mid; ++j) {</pre>	<pre>function mergeSort(arr) { if (arr.length === 1) { return arr; } let mid = Math.floor(arr.length / 2); let leftArr = arr.slice(0, mid); let rightArr = arr.slice(mid); return merge(mergeSort(leftArr), mergeSort(rightArr)); } function merge(leftArr, rightArr) { let result = []; let leftIndex = 0; let rightIndex = 0; while (leftIndex < leftArr.length && rightIndex < rightArr.length) { if (leftArr[leftIndex] < rightArr[rightIndex]) { result.push(leftArr[leftIndex]); leftIndex++; } else { result.push(rightArr[rightIndex]); rightIndex++; } } return result.concat(leftArr.slice(leftIndex)).concat(rightArr. slice(rightIndex)); } console.log(mergeSort([15, 21, 13, 5, 10, 7]));</pre>

```

        rightArr[j] = arr[j + mid];
    }

    mergeSort(leftArr);
    mergeSort(rightArr);

    // когда достигли базового условия, начинаем слияние
    merge(arr, leftArr, rightArr);
}

private static void merge(int[] arr, int[] leftArr, int[]
rightArr) {
    int leftArrLength = leftArr.length;
    int rightArrLength = rightArr.length;

    // контролируем индексы подмассивов
    int leftIndex = 0;
    int rightIndex = 0;

    // контролируем индекс в основном массиве
    int arrIndex = 0;

    while (leftIndex < leftArrLength && rightIndex <
rightArrLength) {
        if (leftArr[leftIndex] < rightArr[rightIndex]) { //
сравниваем элемент в левом и правом подмассиве
            arr[arrIndex] = leftArr[leftIndex]; // если
условие верно, то элемент в левом меньше, сохраняем его в
0 ячейку основного массива
            leftIndex++;
            arrIndex++;
        } else {
            arr[arrIndex] = rightArr[rightIndex];
            rightIndex++;
            arrIndex++;
        }
    }

    // копируем элементы если остались из левого

```

Псевдокод

```

mergeSort(array[], leftIndex, rightIndex)
если leftIndex < rightIndex
1. Находим среднюю точку, чтобы разделить массив на две
половины:
    middle = (leftIndex + rightIndex) / 2
2. Вызов mergeSort для первой половины:
    Вызов mergeSort(array, leftIndex, middle)
3. Вызов mergeSort для второй половины:
    Вызов mergeSort(array, middle + 1, rightIndex)
4. Объедините две половины, отсортированные на шаге 2 и
3:
    Вызов merge(array, leftIndex, middle, rightIndex)

// Объединим два отсортированных массива
merge(array[], leftIndex, middle, rightIndex)
left_i = 1, right_i = middle
while(left_i < leftIndex AND right_i < rightIndex)
if(element at left_i > element at right_i)
    left_i ++
else if(element at left_i < element at right_i)
    left_i ++ and right_i ++

```

```
while (leftIndex < leftArrLength) {
    arr[arrIndex] = leftArr[leftIndex];
    leftIndex++;
    arrIndex++;
}

// копируем элементы из правого
while (rightIndex < rightArrLength) {
    arr[arrIndex] = rightArr[rightIndex];
    rightIndex++;
    arrIndex++;
}
}
```

Таким образом, мы получили отсортированный массив данных.

Общее количество шагов, необходимых для выполнения сортировки Merge sort, равно

$O(n \cdot \log n)$, где n - это количество элементов в исходном массиве.



Это время работы является довольно эффективным для больших массивов данных.

Проходим по стопам работы алгоритма

Диаграмма блок схемы: <https://app.diagrams.net/#G1EEvZmqK0ZEg3jWTAfvdSa1ij50ho6Qj5>

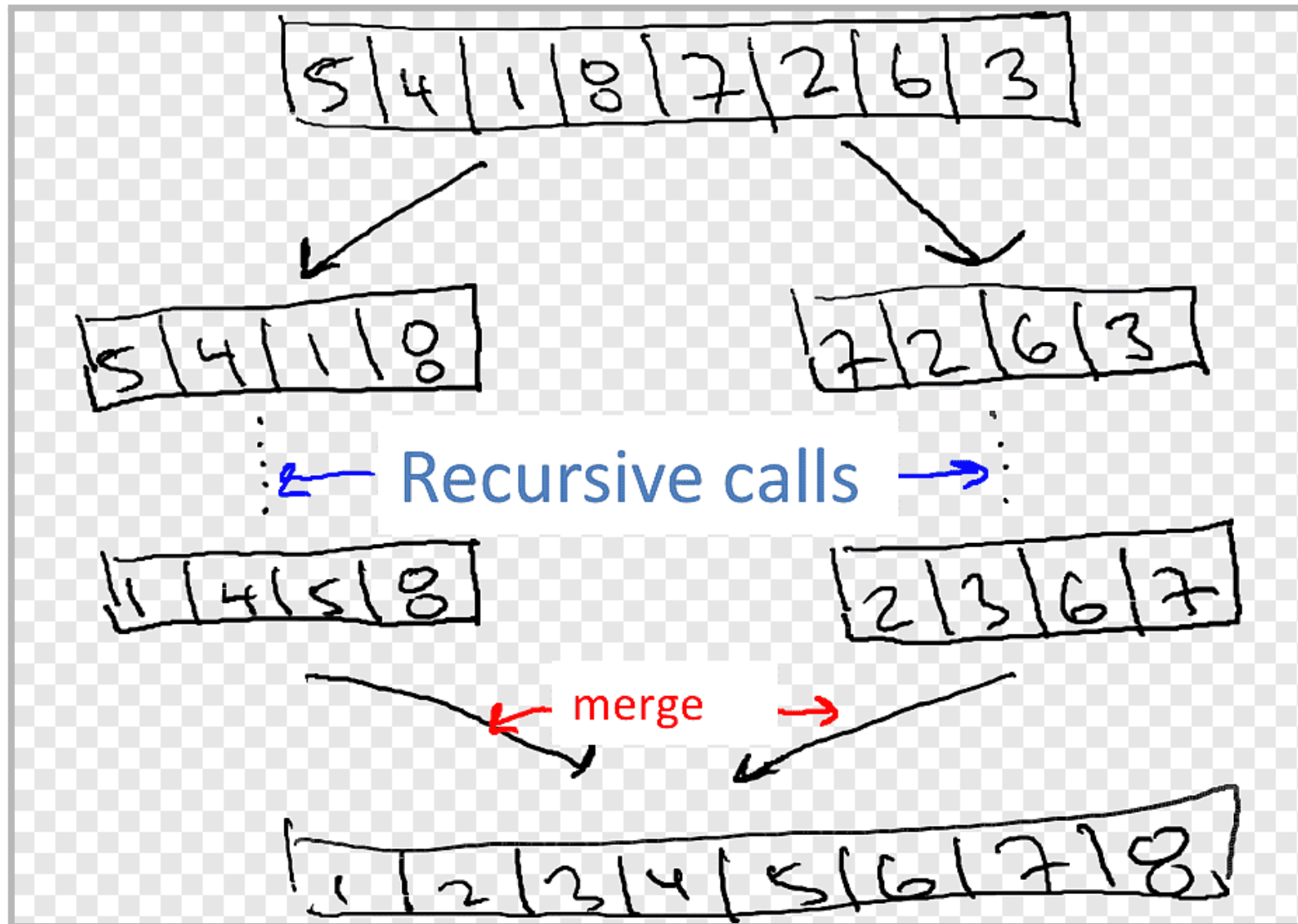
Диаграмма для пошагового разбора: https://drive.google.com/file/d/1kg49lleMSQ25l_-nJ0glAYPoPKOAO4ZS/view?usp=sharing

Код `{/}`


Разница между быстрой сортировкой и сортировкой слиянием


Основа для сравнения	Быстрая сортировка	Сортировка слиянием
Разбиение элементов в массиве	Разделение списка элементов не обязательно делится на половину.	Массив всегда делится на половину ($n / 2$).
Хорошо работает на	Меньший массив	Работает нормально для любого типа массива.
Скорость	Быстрее, чем другие алгоритмы сортировки для небольшого набора данных.	Постоянная скорость во всех типах наборов данных.
Требуется дополнительное место для хранения	Меньше	Больше
КПД	Неэффективно для больших массивов.	Более эффективен.

Блок схема





Вопросы на собеседовании

 Как работает алгоритм Merge Sort?

 Какова временная сложность алгоритма Merge Sort?

 Какие преимущества и недостатки имеет Merge Sort по сравнению с другими алгоритмами сортировки?

 Как происходит слияние двух отсортированных массивов в алгоритме Merge Sort?

 Можно ли использовать алгоритм Merge Sort для сортировки не целых чисел? *

Поиск алгоритма решения

Задача #1

У вас есть **12 монет** среди которых **1** монета фальшивая, весом отличающаяся от других монет.

Вы должны определить фальшивую монету, используя только двухчашечные весы.

Какое количество минимальных взвешиваний потребуется?

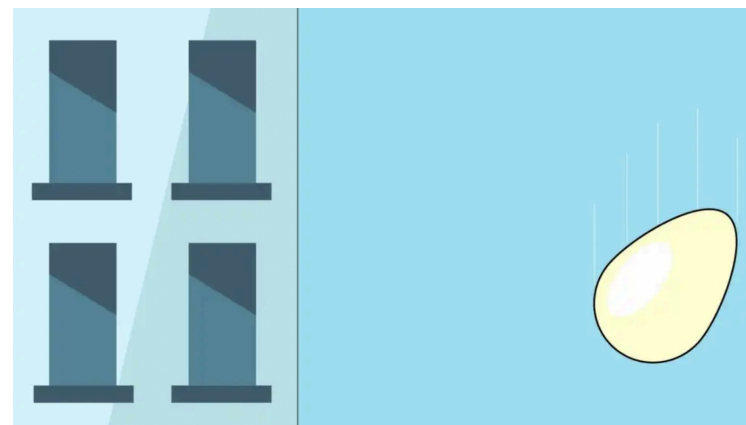
Как мы можем определить фальшивую монету и узнать, тяжелее она или легче остальных?

Задача #2

Вы должны доставить несколько яиц из точки **A** в точку **B**, расположенных на расстоянии **100** этажей.

У вас есть лестница, но у яиц есть свойство разбиваться, если они падают с определенной высоты и выше.

Какое минимальное количество яиц вам понадобится, чтобы гарантировать доставку яиц без их разбивания, и как вы будете использовать лестницу, чтобы определить эту высоту?



Домашнее задание

Level 0

Нарисуйте блок схему для задач по поиску алгоритма решения.

Level 1

У вас есть список из **n** элементов, которые представляют собой оценки студентов по математике. Вам нужно отсортировать этот список в порядке убывания оценок с использованием алгоритма сортировки Merge sort. Для решения этой задачи напишите функцию, которая принимает на вход список оценок и возвращает новый список, отсортированный в порядке убывания.

```
{3, 8, 1, 9, 4, 2, 7, 6, 5};
```

Level 2

Дан массив объектов типа Person, который содержит поля name (тип String) и age (тип int). Необходимо отсортировать этот массив по возрасту в порядке убывания, используя алгоритм Merge sort.

JAVA	JAVA SCRIPT
<pre>Person[] people = { new Person("Alice", 25), new Person("Bob", 20), new Person("Charlie", 30), new Person("David", 35), new Person("Eve", 28) };</pre>	<pre>let people = [{ name: "Alice", age: 25 }, { name: "Bob", age: 20 }, { name: "Charlie", age: 30 }, { name: "David", age: 35 }, { name: "Eve", age: 28 }];</pre>