Merge — сортировка слиянием в Java
Подразумевает принцип "разделяй и властвуй". В чем идея и ее смысл?

1. *Сортировка.*

Разделяем массив на части, пока он не будет равен 1 элементу. Каждый 1 элемент является отсортированным.

2. *Слияние.*

Слияние отсортированных элементов.

По принципу двух колод карт. Кладем 2 колоды карт на стол вверх значениями, и карту которая из них младше, кладем в третью результирующую стопку карт. В конечном итоге, если карты в какой-то колоде закончились, перекладываем их по очереди в результирующую. Получится слитый из двух отсортированных массивов, один, новый, отсортированный массив.
Затрачиваемое время O(n log2 n).

Приведем пример к нашей сортировке. Есть массив {6, 1, 3, 5, 2, 4, 7, 8}, если его длина больше 1, то мы делим его на 2 части и получаем левую часть {6, 1, 3, 5} и правую часть {2, 4, 7, 8}. Продолжаем действие деления на 2 части, пока его длинна будет больше 1. В итоге получим кучу массивов длинной в 1 элемент, а именно: {6} {1} {3} {5} {2} {4} {7} {8}.

Реализация на Java примерно следующая:

```
public int [] sortArray(int[] arrayA){ // сортировка Массива который передается в функцию
    // проверяем не нулевой ли он?
    if (arrayA == null) {
       return null;
    }
    // проверяем не 1 ли элемент в массиве?
    if (arrayA.length < 2) {
       return arrayA; // возврат в рекурсию в строки ниже см комменты.
    }
    // копируем левую часть от начала до середины
    int [] arrayB = new int[arrayA.length / 2];
    System.arraycopy(arrayA, 0, arrayB, 0, arrayA.length / 2);

    // копируем правую часть от середины до конца массива, вычитаем из длины первую часть
    int [] arrayC = new int[arrayA.length - arrayA.length / 2];
    System.arraycopy(arrayA, arrayA.length / 2, arrayC, 0, arrayA.length - arrayA.length / 2);

    // рекурсией закидываем поделенные обе части обратно в наш метод, он будет крутится до тех пор,
    // пока не дойдет до 1 элемента в массиве, после чего вернется в строку и будет искать второй такой же,
```

```
        // точнее правую часть от него и опять вернет его назад
        arrayB = sortArray(arrayB); // левая часть возврат из рекурсии строкой return
arrayA;
        arrayC = sortArray(arrayC); // правая часть возврат из рекурсии строкой return
arrayA;

        // далее опять рекурсия возврата слияния двух отсортированных массивов
        return mergeArray(arrayB, arrayC);
    }
```

Далее нужно слить эти массивы в 1. Как это делается? Чтобы не проходить многократно по каждому массиву, введем индексы позиции для каждого массива. После чего один раз пройдем по циклу, равному длине суммы этих двух массивов. Берется первый массив и второй массив, и берется первый элемент, сравнивается больше *элемент номер 1 в первом массиве* и *элемент номер 1 во втором массиве*? Меньший из них кладется в результирующий массив. Тут важно, если мы взяли элемент из первого массива, то при переходе цикла, он должен ссылаться на 2 элемент первого массива и на 1 элемент второго массива. Для этого нужно увеличить индекс второго массива на +1 и при проверке вычитать его из номера цикла, аналогично и для первого массива. Например есть 2 массива: {1}{4}{8} и {3}{6}{7} И есть цикл:

```
for (int i = 0; i < arrayA.length + arrayB.length; i++) {
            if (arrayA[i] < arrayB[i]) {
            arrayC[i] = arrayA[i];
            } else {
            arrayC[i] = arrayB[i];
            }
}
```

При первом проходе цикла получится что arrayC[1] = {1}: мы взяли этот элемент из первого массива. То при проходе по второму циклу, мы уже должны сравнивать элемент {4} и {3}, но чтобы это сделать нам нужны учитывать индексы позиций и смещение обоих массивов, для этого вводим их.

```
int positionA = 0, positionB = 0;
for (int i = 0; i < arrayA.length + arrayB.length; i++) {
            if (arrayA[i - positionA] < arrayB[i - positionB]) {
            arrayC[i] = arrayA[i - positionA];
            positionB++;
            } else {
            arrayC[i] = arrayB[i - positionB];
            positionA++;
            }
}
```

Но это еще не все, нужно учесть, что какой-то массив может закончиться раньше. Например есть 3 массива: {1}{3}{5} и {6}{7}{9} Первый массив закончится еще до того, как подойдет второй, для этого нужно ввести проверку и, в приципе, готова функция слияния.

```
public int [] mergeArray(int [] arrayA, int [] arrayB) {

int [] arrayC = int[arrayA.length + arrayB.length];
int positionA = 0, positionB = 0;

for (int i = 0; i < arrayC.length; i++) {
        if (positionA == arrayA.length){
        arrayC[i] = arrayB[i - positionB];
        positionB++;
        } else if (positionB == arrayB.length) {
        arrayC[i] = arrayA[i - positionA];
        positionA++;
        } else if (arrayA[i - positionA] < arrayB[i - positionB]) {
        arrayC[i] = arrayA[i - positionA];
        positionB++;
        } else {
        arrayC[i] = arrayB[i - positionB];
        positionA++;
        }
}
return arrayC;
```

Самое тяжелое в этой сортировке — это принцип перехода рекурсии. Т.е. мы закидываем в рекурсию левую часть до тех пор, пока она делится на 2, а потом раскручиваем ее обратно, на словах это очень сложно и запутанно, а когда пытаешься представить, если еще и не понятно, то совсем лес. Берем массив: {2}{1}{4}{3}. Первая рекурсия сортировки поделит его на 2 части и запустит функцию еще раз с элементами *2-1*, потом еще раз с элементом *2* и *1*, вернет их по очереди, таким образом в функцию слияния попадут первыми они, а выйдут уже *1-2*, потом рекурсия вернется обратно и закинет в слияние уже *4-3*, потом *4* и *3*, после чего слияние вернет *3-4*, а уже потом рекурсия раскрутится еще раз обратно и в слияние попадет уже *1-2* и *3-4*, а вернется отсортированный массив *1-2-3-4*. Ну вот в целом и все, сортировка состоит из двух функций.

```
sortArray(array);                   // кладем массив который нужно отсортировать
mergeArray(arrayA, arrayB);         // кладем 2 массива которые нужно слить в один
```

```
public static void main(String[] args) {
      Merge testMerge = new Merge();
      int [] result = testMerge.sortArray(new int[]{2,3,1,4});

      for (int i = 0; i < result.length ; i++) {
         System.out.print(result[i] + " ");
      }
   }
```