

Сортировки

- ♦ поразрядная сортировка (radix sort)
- ♦ пузырьковая сортировка (bubble sort)
- ♦ сортировка кучей (heap sort)
- ♦ сортировка слиянием (merge sort)

Поразрядная сортировка

♦ Другие названия: цифровая сортировка, radix sort

- ✓ чисел
- ✓ строк
- ✓ дат
- ✓ структур с несколькими полями



♦ Класс: сортировки без сравнения, distribution sort

♦ Основная идея: последовательное (многократное) раскидывание сортируемых элементов по кучкам (buckets) на основании равенства определенного поля этих элементов и последующее объединение кучек в определенном порядке.

После того, как эта процедура будет повторена последовательно для всех полей (напр., разрядов числа), получившаяся кучка будет содержать отсортированную последовательность элементов.

03	-	января	-	2014
14	-	января	-	2013
03	-	апреля	-	2013

порядок действий:

1. отсортировать по годам
2. отсортировать по месяцам
3. отсортировать по числам

Поразрядная сортировка

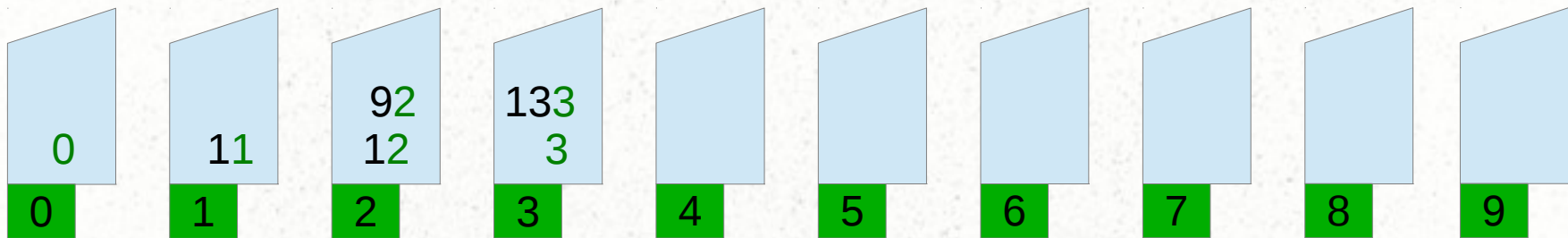
- ♦ Сортировка массива чисел: [12, 11, 0, 92, 3, 133]

Самое длинное число 133 имеет 3 разряда

Значит описанную процедуру необходимо повторить 3 раза

- ♦ Шаг 1а: разложить числа по корзинам по младшему разряду (по единицам)

[12, 11, 0, 92, 3, 133]



- ♦ Шаг 1б: обойти корзины и собрать из них числа, сохраняя порядок чисел в корзине (в каком порядке числа попали в корзину, в таком они и достаются)

[0, 11, 12, 92, 3, 133]

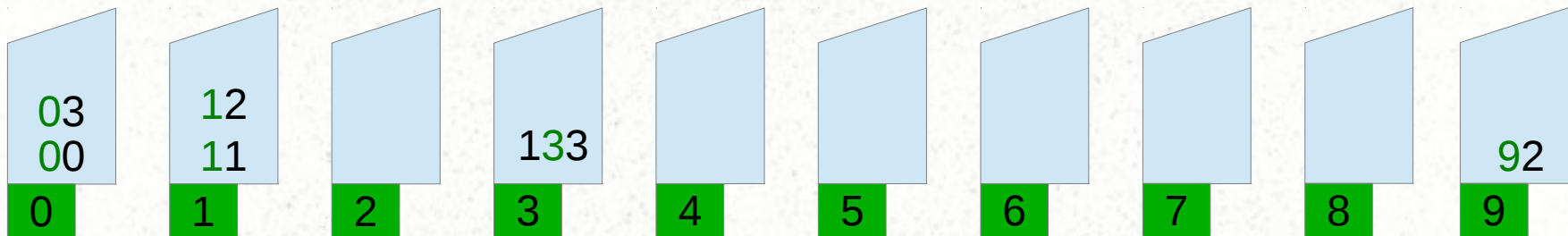
Поразрядная сортировка

- Массив после Шага 1:

[0, 11, 12, 92, 3, 133]

- Шаг 2а: разложить числа по корзинам по второму разряду (по десяткам)

[00, 11, 12, 92, 03, 133]



- Шаг 2б: обойти корзины и собрать из них числа, сохраняя порядок чисел в корзине (в каком порядке числа попали в корзину, в таком они и достаются)

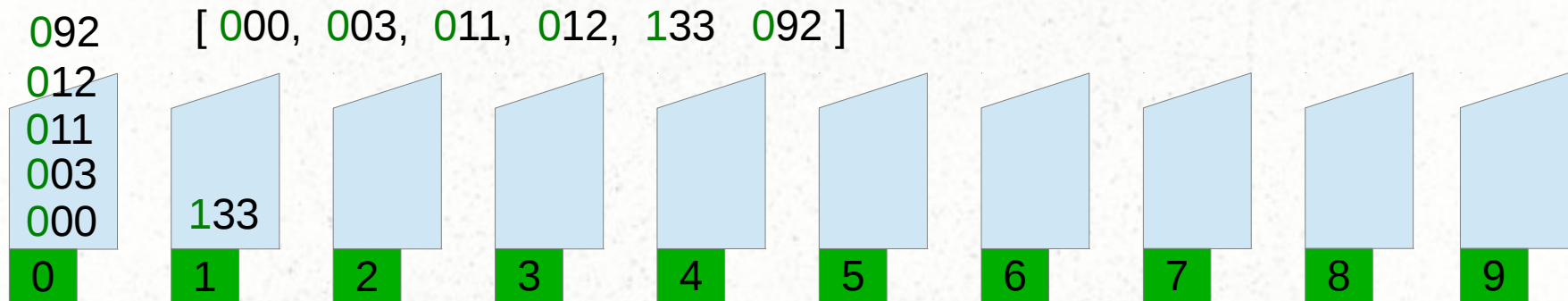
[00, 03, 11, 12, 133, 92]

Поразрядная сортировка

- Массив после Шага 2:

[0, 3, 11, 12, 133, 92]

- Шаг 3а: разложить числа по корзинам по третьему разряду (по сотням)



- Шаг 3б: обойти корзины и собрать из них числа, сохраняя порядок чисел в корзине (в каком порядке числа попали в корзину, в таком они и достаются)

[000, 003, 011, 012, 092, 133]

- Готово! Отсортированный массив:

[0, 3, 11, 12, 92, 133]

- Вопрос: какие корзины были бы заполнены на 4-ой итерации?

Все попало бы в корзину #0

Структура данных

- ♦ Как представить одну корзину?
- ♦ Как представить все корзины?

♦ Корзина (bucket)

#4	092
#3	012
#2	011
#1	003
#0	000

- ♦ должна уметь хранить множество чисел
- ♦ должна уметь сохранять порядок добавления

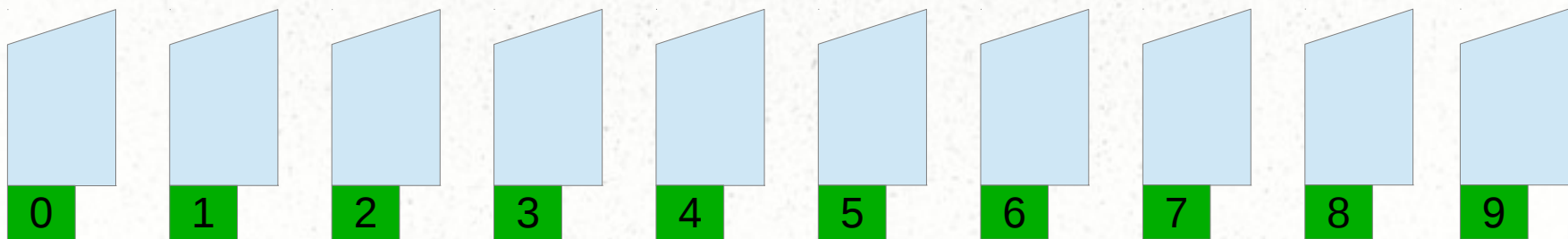


Для этих целей подходит массив!



♦ Все корзины:

- ♦ отдельных корзин много
- ♦ порядок корзин имеет значение



♦ Вся система корзин: массив массивов

bucket 0 bucket 1 bucket 9 6

buckets = [[n1, n2], [n3], ... , [nX]]

Получение значения разряда

- ♦ Задача: как получить значение заданного разряда заданного числа?

12345 #=>5
 #=>4
 #=>3

- ♦ Способ 1: преобразовать в строку и получить нужную позицию из строки

```
num=12345  
num.to_s[4,1].to_i #=> 5  
num.to_s[3,1].to_i #=> 4  
num.to_s[2,1].to_i #=> 3
```

Получение значения разряда

- ◆ Способ 2: при помощи арифметических операций

num=12345

- ◆ Чтобы получить значение в разряде единиц (в позиции 1 с конца) = 5

12345 : 10 = 1234.5

обычное, не целочисленное, деление

12345 % 10 = 5

остаток от деления

- ◆ Чтобы получить значение в разряде десятков (в позиции 2 с конца) = 4

12345 : 100 = 123.45

обычное, не целочисленное, деление

12345 % 100 = 45

остаток от деления

45 / 10 = 4

целочисленное деление

- ◆ Чтобы получить значение в разряде сотен (в позиции 3 с конца) = 3

12345 : 1000 = 12.345

обычное, не целочисленное, деление

12345 % 1000 = 345

остаток от деления

345 / 100 = 3

целочисленное деление

Получение значения разряда

- Можно ли сформулировать закономерность?

$n=12345$	$p=1$	$12345 \% 10 / 1 = 5$	$12345 \% 10^1 / 10^0 = 5$
$n=12345$	$p=2$	$12345 \% 100 / 10 = 4$	$12345 \% 10^2 / 10^1 = 4$
$n=12345$	$p=3$	$12345 \% 1000 / 100 = 3$	$12345 \% 10^3 / 10^2 = 3$
$n=12345$	$p=4$	$12345 \% 10000 / 1000 = 2$	$12345 \% 10^4 / 10^3 = 2$

- Можно ли вывести одну формулу? Можно ли выразить изменяемые части в арифметических выражениях через переменную p и через 10 (основание исчисления в десятиричной системе)?

$$n \% 10^{**} p / 10^{**} (p-1)$$

**

возведение
в степень

приоритет операции возведения в степень ** выше, чем у операций % и /, выполнение происходит в таком порядке (сначала вычисляются скобки):

$$n \% (10^{**} p) / (10^{**} (p-1))$$

Задания (наконец!)

- ♦ Разработайте метод, возвращающий значение заданного разряда в заданном числе. Добавьте тесткейсы (см. примеры в начальном скрипте).

`digit_at_pos(12345, 1) #=> 5`

`digit_at_pos(12345, 2) #=> 4`

`digit_at_pos(12345, 3) #=> 3`

Начальный скрипт: `lesson.20/radix_sort/radix_sort_stub`

- ♦ Разработайте метод `radix_sort`, реализующий поразрядную сортировку массива чисел произвольной разрядности (длины). Добавьте тесткейсы.

продолжайте скрипт: `lesson.20/radix_sort/radix_sort_stub`

ответ: `lesson.20/radix_sort/radix_sort`

Задание radix_sort_for_strings - 1

- Разработайте скрипт radix_sort_for_strings сортирующий строки в словарном порядке по алгоритму поразрядной сортировки.

Начальный скрипт: ваш собственный из lesson.20/radix_sort/radix_sort_stub
но используйте тесткейсы из скрипта
lesson.21/radix_sort/radix_sort_for_strings_stub

- На шаге 1 произведите рефакторинг скрипта и добейтесь, чтобы он работал для следующего массива (тесткейс #1):

```
strings = ["ramp", "lamp", "camp"]
```

Проблемы?

```
buckets = []  
...  
(buckets [char] ||= []) << str
```

String, а надо Integer

Решение:

```
buckets = {}
```

– НЕТ

хэш не гарантирует порядок среди ключей, тем более, алфавитный

- преобразовывать символы с числа так, чтобы:

String#ord

number('a') = number('a')

number('a') < number('b')

см. подробнее в basic_ruby

Задание radix_sort_for_strings - 2

- На шаге 2 убедитесь, что сортировка работает для следующего массива (тесткейс #2):

["apples", "apple pie", "ramp", "lamp", "camping", "apple"]

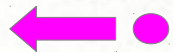
отсортированный вид:

["apple", "apple pie", "apples", "camping", "lamp", "ramp"]

Внимание: "apple pie" < "apples" так как пробел сортируется **перед** буквенными символами

- Какой *порядок обхода* разрядов и *выравнивание* д.б. при сортировке строк?

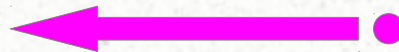
[12, 11, 0, 92, 3, 133]



0 0 0
0 0 3
0 1 1
0 1 2
0 9 2
1 3 3

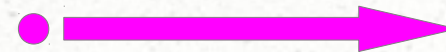
обход: справа налево
выравнивание: по правому краю

[apple apple pie apples camping lamp ramp]



" " " " a p p l e
a p p l e p i e
" " " a p p l e s
" " c a m p i n g
" " " " " l a m p
" " " " " r a m p

2 обхода x 2 выравнивания = 4 комбинации



a p p l e " " " "
a p p l e p i e
a p p l e s " " "
c a m p i n g " "
l a m p " " " " "
r a m p " " " " "

обход: слева направо
выравнивание: по левому краю

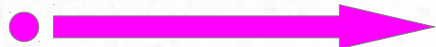
Задание radix_sort_for_strings - 3

- 2 порядка обхода x 2 выравнивания = 4 комбинации = 4 варианта реализации метода char_at_pos. И никаких других изменений!

(считая позиции от 1)

- обход слева направо, выравнивание по левому краю

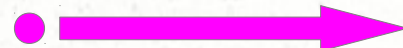
```
char_at_pos('apple', 1)    #=> a
char_at_pos('camping', 1)  #=> c
char_at_pos('apple', 6)    #=> "
```



```
a p p l e " " " "
a p p l e   p i e
a p p l e s " " "
c a m p i n g " "
l a m p " " " " "
r a m p " " " " "
```

- обход слева направо, выравнивание по правому краю

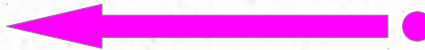
```
char_at_pos('apple', 1)    #=> "
char_at_pos('apple pie', 1) #=> a
char_at_pos('apple', 6)    #=> p
```



```
" " " " a p p l e
a p p l e p i e
" " " a p p l e s
" " c a m p i n g
" " " " " l a m p
" " " " " r a m p
```

- обход справа налево, выравнивание по левому краю

```
char_at_pos('apple', 1)    #=> "
char_at_pos('apple pie', 1) #=> e
char_at_pos('apple', 6)    #=> l
```



```
a p p l e " " " "
a p p l e   p i e
a p p l e s " " "
c a m p i n g " "
l a m p " " " " "
r a m p " " " " "
```

- обход справа налево, выравнивание по правому краю

см. реализацию digit_at_pos в radix_sort

Доп. задания

- ♦ Изучите реализацию в скрипте `lesson.21/radix_sort/radix_sort_for_strings`
см. `radix_sort_for_strings_clean` – то же самое, но все лишнее удалено
- ♦ Изучите еще одну реализацию `radix_sort_for_strings_2`, в которой используется метод `Integer#downto`, чтобы сделать обход справа налево.

Также изменена реализация метода `char_at_pos`

- ♦ Исследуйте, какой будет результат сортировки, если в сортируемом наборе есть слова в прописном, так и в строчном регистре.
 - Согласно какой локали были отсортированы символы?

Как в локали C, потому что	'A'.ord	==> 65
	'a'.ord	==> 97

Сортировка кучей

Hello

Сортировка слиянием

- Реализует стратегию «разделяй и властвуй».
 - Использует рекурсию
- Шаг 1. Разделить массив на минимальные части.
- Шаг 2. Сливать соседние фрагменты, выстраивая элементы в отсортированном порядке.

- танцевальная демонстрация

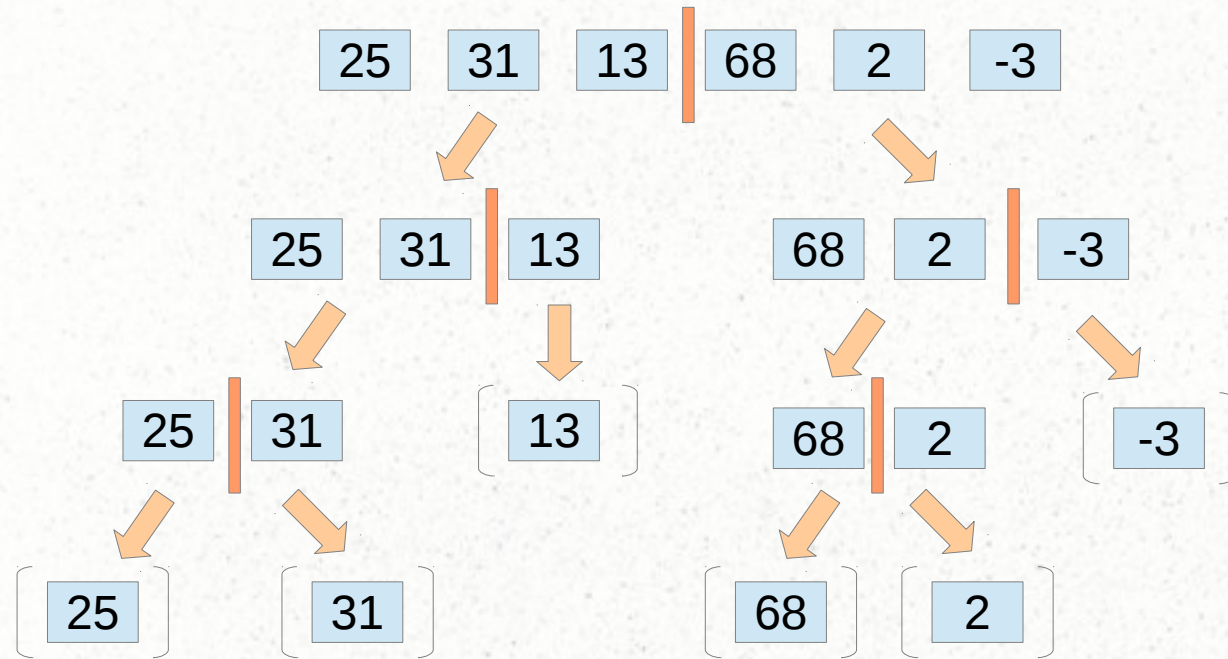
https://www.youtube.com/watch?v=XaqR3G_NVoo

- объяснение

<https://www.youtube.com/watch?v=TzeBrDU-JaY>

Сортировка слиянием. Шаг 1

- Шаг 1. Разделить массив на **минимальные** части.
- реализуется рекурсивно

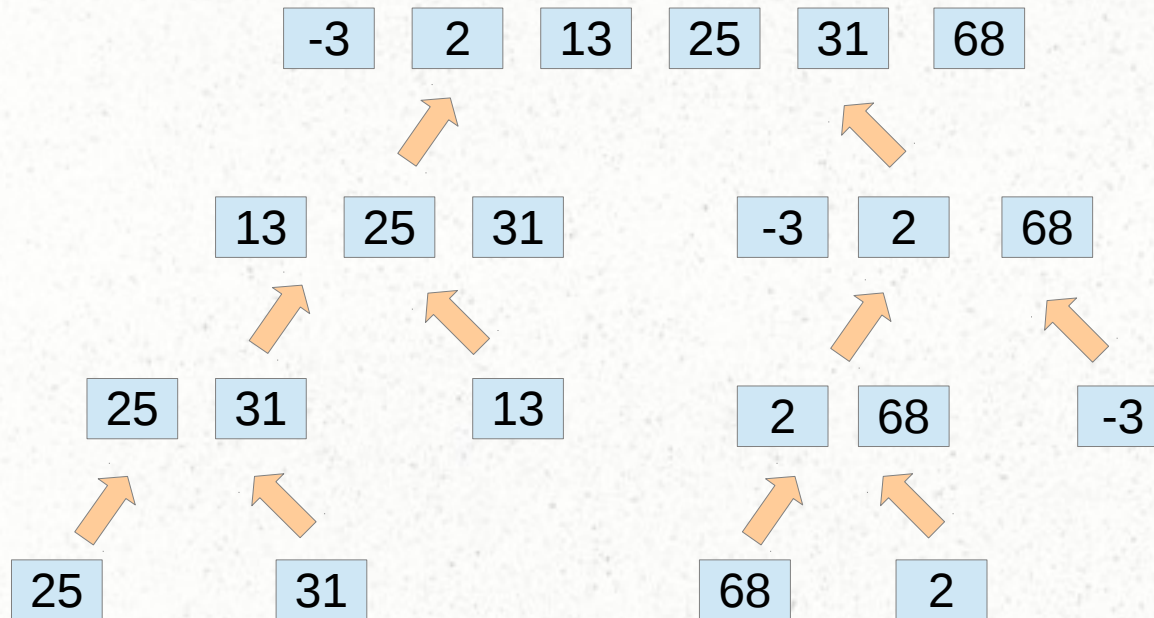


*примерно
по середине*

- Base case — пустой массив или массив с единственным элементом
 - такой массив является отсортированным

Сортировка слиянием. Шаг 2

- Шаг 2. Объединить фрагменты, выстраивая элементы в отсортированном порядке.
 - делается на возврате из рекурсии



- Процедура слияния двух (already sorted!) массивов порождает новый массив.
 - ♦ реализация на-месте отсутствует
 - ♦ реализация на-месте имитируется последующим замещением элементов исходного (под)массива элементами из отсортированного вспомогательного массива

Сортировка слиянием. Слияние

■ Алгоритм (реализуется при помощи цикла):

- в начале: «текущие» значения это первые элементы каждого массива
- сравнить текущие значения из обоих массивов, выбрать меньшее и положить его во временный массив для отсортированных значений
- в массиве, давшем меньшее значение, текущим становится следующее по порядку значение.
- когда один из массивов закончился, из другого массива перенести оставшиеся элементы в массив для отсортированных значений

array LEFTS	array RIGHTS	min value	sorted (tmp) array					
13 25 31	-3 2 68	-3	-3					
13 25 31	2 68	2	-3	2				
13 25 31	68	13	-3	2	13			
25 31	68	25	-3	2	13	25		
31	68	31	-3	2	13	25	31	

- перенести оставшиеся элементы из RIGHTS

			68	-3	2	13	25	31	68
--	--	--	----	----	---	----	----	----	----

- для сортировки на месте, скопировать отсортированный массив поверх двух фрагментов исходного массива

-3	2	13	25	31	68	-3	2	13	25	31	68
----	---	----	----	----	----	----	---	----	----	----	----

Задания

- Задание: реализуйте процедуру слияния двух отсортированных массивов в новый массив

начальный скрипт: `lesson.36/merge_sort_ivan/merge_stub.rb`
ответ: `lesson.36/merge_sort_ivan/merge.rb`

или:

начальный скрипт: `lesson.36/merge_sort/merging_stub`
ответ: `lesson.36/merge_sort/merging`

`Array#[idx]`

`Array#concat` и `Array#+`

`Array#shift`

цикл `while`

- Задание: реализуйте сортировку слиянием в новый массив

начальный скрипт: `lesson.36/merge_sort_ivan/mergesort_stub.rb`
ответ: `lesson.36/merge_sort/mergesort.rb`

`Array#[s,l]`

Задания

- Задание «побыть скриптом»: глядя на реализацию метода mergesort в скрипте merge_sort.rb, продемонстрируйте на карточках, как происходит сортировка. Если испытываете затруднения, изучите выход:

```
lesson.36/merge_sort_ivan/merge_sort_verbose.rb  
lesson.36/merge_sort_ivan/merge_sort_verbose.out
```

- Задание: добавить в ваш собственный скрипт возможность выводить подобную информацию.

```
@debug = true  
  
def verboser msg, level=0  
  if @debug  
    puts "  " * level + msg  
  end  
end
```

ответ: lesson.36/merge_sort_ivan/merge_sort_verbose.rb

Задания

- Задание: подсчитайте, сколько раз вызывается метод mergesort и исследуйте, как меняется количество вызовов метода mergesort, если делить массив не на равные части, а, например, 2:1. Что меняется?

> mid = array.size*2 / 3

начальный скрипт: lesson.36/merge_sort_ivam/merge_sort_verbose.rb

ответ: lesson.36/merge_sort_ivam/merge_sort_verbose_with_counts.rb

Сортировка слиянием. На месте

- Сортировка на месте.
 - вместо того, чтобы выделять два новых массива из одного, подмассивы эмулируются посредством указания индексов начала и конца фрагмента

```
a = [0,10,20, 30,40,50,60]
```

```
merge_sort(a, 0, 2)  
merge_sort(a, 3, 6)
```

- в методе merge используется временный массив, в который складываются элементы в отсортированном порядке
- потом данные из временного массива копируются поверх элементов исходного массива (в диапазоне между индексами начала и конца релевантных фрагментов)
- Задание: реализуйте сортировку слиянием на месте

начальный скрипт: `lesson.36/merge_sort/merge_sort_stub`

ответ: `lesson.36/merge_sort/merge_sort`