education

Массивы. Часть 2

Алгоритмы и структуры данных

Илья Почуев

Что будет на занятии

- Саморасширяющийся массив. Принцип работы
- Реализация саморасширяющегося массива в коде
- Амортизированная сложность
- Почему надо избегать лишних аллокаций памяти
- Задача: слияние двух отсортированных массивов

Область применения саморасширяющегося массива

- Заранее размер массива неизвестен
- Есть необходимость получения элементов по индексу *

* По-прежнему хотим O(1) при выборке

education

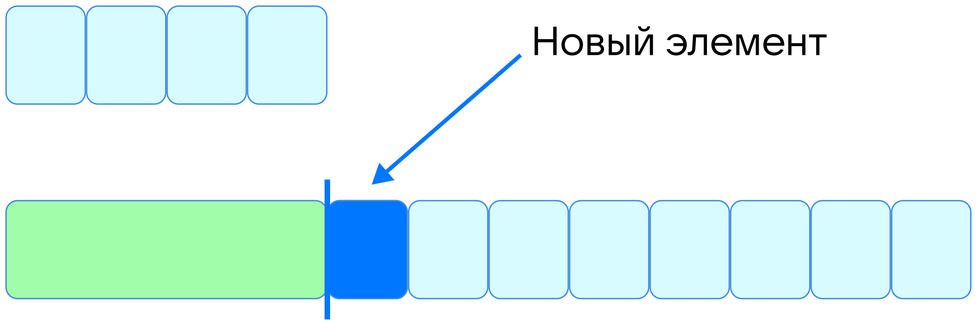
Алгоритм работы самораширяющегося массива

Как должна работать такая структура данных?

- У массива есть два основных параметра: size и capacity
- Если size становится равным capacity, то в такой массив невозможно добавить элемент
- Если необходимо продолжить вставку, то необходим алгоритм увеличения capacity (реаллокация памяти)
- Необходимо заново найти свободное непрерывное пространство в памяти, чтобы перенести уже существующие элементы последовательно и было место под новый элемент

Алгоритм саморасширяющегося массива

- Скопировать все значения из старого массива в новый
- Вставить новый элемент
- Удалить старый массив, чтобы избежать утечки по памяти



education

Алгоритм увеличения capacity

Алгоритм увеличения capacity

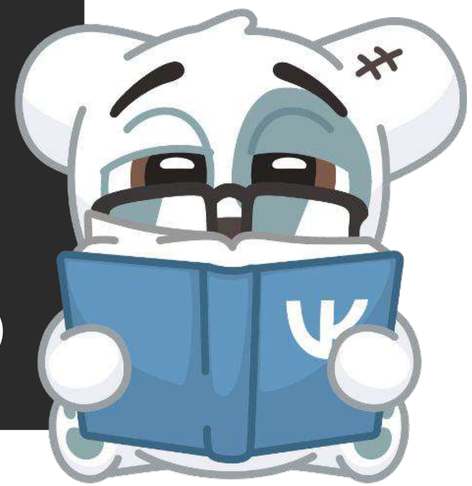
Сделаем capacity равным единице при инициализации

```
arr = DynamicArray()
for i in range(0, 31):
    arr.append(i)

print (arr.count)
```

```
import ctypes


class DynamicArray (object):
    def __init__(self):
        self.count = 1
        self.size = 0
        self.capacity = 1
        self.array = self.make_array (self.capacity)
```



Алгоритм увеличения capacity

Сделаем capacity равным единице при инициализации

Попробуем понять, как лучше увеличивать ёмкость массива

```
arr = DynamicArray()
for i in range(0, 31):
    arr.append(i)

print (arr.count)
```

```
import ctypes

class DynamicArray (object):
    def __init__(self):
        self.count = 1
        self.size = 0
        self.capacity = 1
        self.array = self.make_array (self.capacity)

    def append (self, element):
        if self.size == self.capacity:
            self.resize()
            print (f"slow: {self.capacity}")
        else:
            print (f"fast")

        self.array [self.size] = element
        self.size += 1
```

Алгоритм увеличения capacity

Сделаем capacity равным единице при инициализации

Попробуем понять, как лучше увеличивать ёмкость массива

Попробуем увеличивать на 5 всякий раз, когда пытаемся вставить элемент в заполненный массив

```
arr = DynamicArray()
for i in range(0, 31):
    arr.append(i)

print (arr.count)
```

```
import ctypes


class DynamicArray (object):
    def __init__(self):
        self.count = 1
        self.size = 0
        self.capacity = 1
        self.array = self.make_array(self.capacity)

    def append(self, element):
        if self.size == self.capacity:
            self.resize()
            print(f"slow: {self.capacity}")
        else:
            print(f"fast")

        self.array[self.size] = element
        self.size += 1

    def resize(self):
        new_cap = self.capacity + 5
        new_array = self.make_array(new_cap)

        for i in range(self.size):
            self.count += 1
            new_array[i] = self.array[i]

        self.array = new_array
        self.capacity = new_cap
```

Результат

```
arr = DynamicArray()
for i in range(0, 31):
    arr.append(i)

print (arr.count)
```

```
fast
slow: 6
fast
fast
fast
fast
slow: 11
fast
fast
fast
fast
slow: 16
fast
fast
fast
fast
slow: 21
fast
fast
fast
fast
slow: 26
fast
fast
fast
fast
slow: 31
fast
fast
fast
fast
```

```
import ctypes


class DynamicArray (object):
    def __init__(self):
        self.count = 1
        self.size = 0
        self.capacity = 1
        self.array = self.make_array(self.capacity)

    def append(self, element):
        if self.size == self.capacity:
            self.resize()
            print(f"slow: {self.capacity}")
        else:
            print(f"fast")

        self.array[self.size] = element
        self.size += 1

    def resize(self):
        new_cap = self.capacity + 5
        new_array = self.make_array(new_cap)

        for i in range(self.size):
            self.count += 1
            new_array[i] = self.array[i]

        self.array = new_array
        self.capacity = new_cap
```

education

Амортизационная сложность

Ситуации вставки в конец массива

Следует различать две ситуации вставки в конец массива:

- Когда не надо увеличивать capacity – сложность O(1)
- Когда нужно увеличивать capacity: копируем все значения в новый массив, а значит, сложность стремится к O(n)

Амортизационная сложность

Как правильно подсчитать сложность, если O(n) будет далеко не всегда?

- Проведём множество операций вставки
- Подсчитаем общее количество элементарных действий и время на их выполнение
- Разделим общую сложность на количество операций

Такая усреднённая сложность называется амортизационной сложностью, а анализ называется амортизационным

- При заполнении массива 10 000 элементов метод resize будет вызываться 2 000 раз
- При увеличении объёма на 10 элементов — 1000 реаллокаций и т. д.
- Вывод: чем больше элементов в массиве, тем больше будет реаллокаций, а амортизационная сложность будет стремиться к O(n)

```
def resize(self):
    new_cap = self.capacity + 5
    new_array = self.make_array(new_cap)

    for i in range(self.size):
        self.count += 1  # 9997001
        new_array[i] = self.array[i]

    self.array = new_array
    self.capacity = new_cap
```

- Увеличиваем объем в константное значение. Например, в 2 раза
- Количество вызовов функции resize будет всего 15

```
def resize(self):
    new_cap = self.capacity * 2
    new_array = self.make_array(new_cap)

    for i in range(self.size):
        self.count += 1  # 16384
        new_array[i] = self.array[i]

    self.array = new_array
    self.capacity = new_cap
```

- Количество простых операций при увеличении capacity на 5 — 9997001
- Количество простых операций при увеличении capacity в 2 раза — 16384

```
def resize(self):
    new_cap = self.capacity + 5
    new_array = self.make_array(new_cap)

    for i in range(self.size):
        self.count += 1  # 9997001
        new_array[i] = self.array[i]

    self.array = new_array
    self.capacity = new_cap
```

```
def resize(self):
    new_cap = self.capacity * 2
    new_array = self.make_array(new_cap)

    for i in range(self.size):
        self.count += 1  # 16384
        new_array[i] = self.array[i]

    self.array = new_array
    self.capacity = new_cap
```

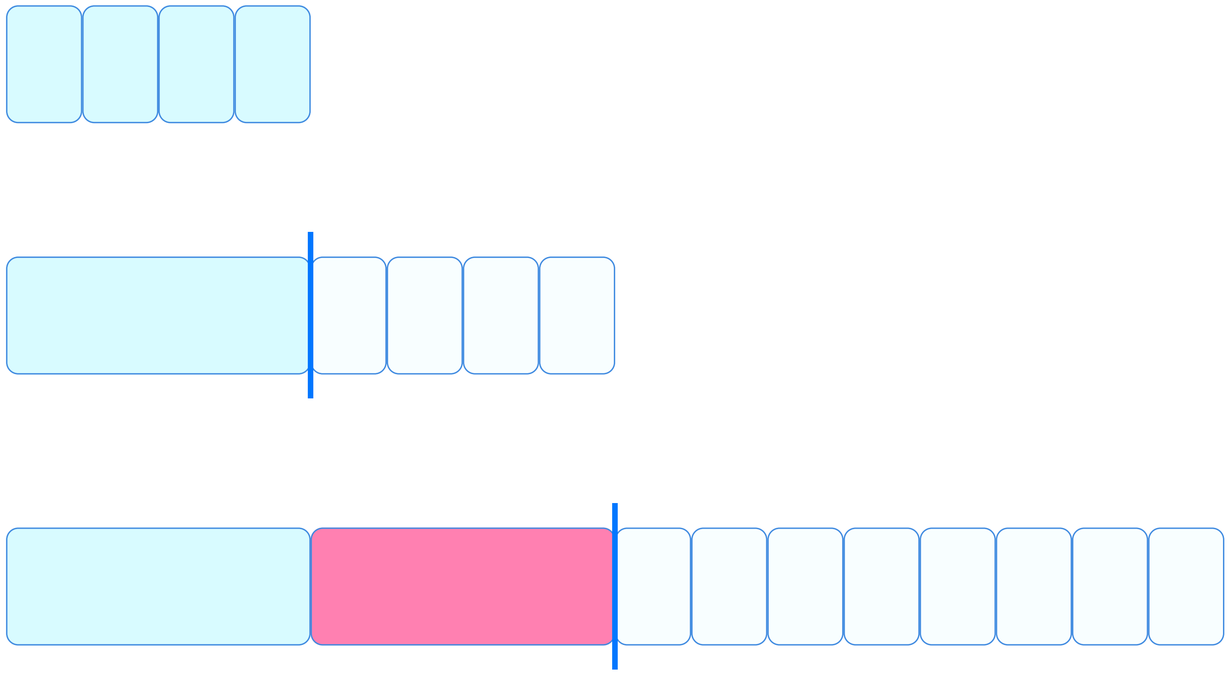
Резюме

- До вызова метода resize, когда у нас есть место в массиве, мы произвели n вставок, каждая из которых занимала O(1)
- При реаллокации мы произвели вставку, которая заняла O(n)
- В итоге у нас есть n + 1 операций, которые у нас заняли 2n (одно n до вызова resize, другое n непосредственно в момент вызова resize)
- Всё это можно представить в виде 2n/(n + 1), что в нотации O большое эквивалентно O(1)

Почему надо избегать аллокаций

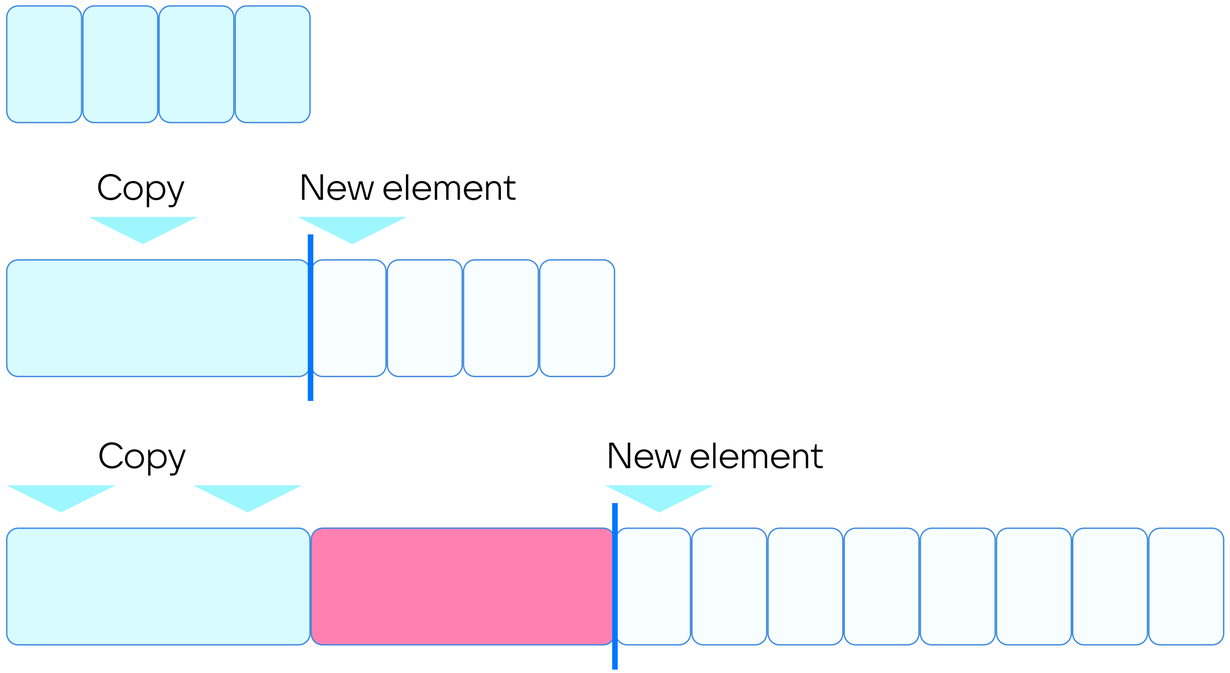
“Right” branch of “if”	1–2			0.3 ns
L1 read	3–4			0.5 ns
L2 read		10–12		5–7 ns
“Wrong” branch of “if”		10–20		5–10 ns
L3 read		30–70		> 20 ns
Allocation + deallocation pair (small objects)			200–500	> 500 ns

Резюме



Аллокация памяти происходит каждый раз, когда мы пытаемся добавить элементы в массив, в котором size == capacity

Резюме



Помимо аллокации памяти, необходимо скопировать все элементы из предыдущего массива: вставка в таком случае O(n)

- Необязательно знать конечный размер массива
- При превышении capacity создаётся новый массив с capacity*2 (в какое-то константное число раз)

education

Удаление элемента из самораширяющегося массива

Обратная ситуация

- При увеличении ёмкости в два раза capacity увеличивается тоже в 2 раза
- Возможно, что при удалении надо следовать той же стратегии и уменьшать capacity в два раза, когда массив освободился наполовину?
- Если после уменьшения capacity в два раза сразу последует вставка, то придётся вновь аллоцировать в два раза больше памяти. Если эти операции будут повторяться, то мы рискуем получить в алгоритме сложность O(n)

Обратная ситуация

Решение

Уменьшают в два раз объём, когда справедливо равенство:

$$size = capacity/4$$

То есть когда реальное количество элементов в массиве в 4 раза меньше, чем его ёмкость, тогда уменьшают capacity в 2 раза.

education

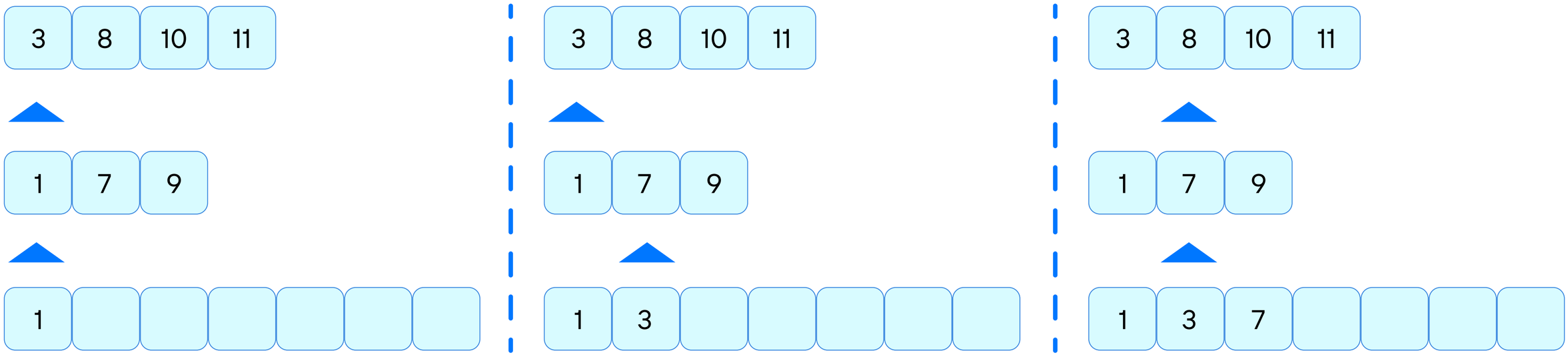
Задача с использованием массива

Слияние двух отсортированных массивов

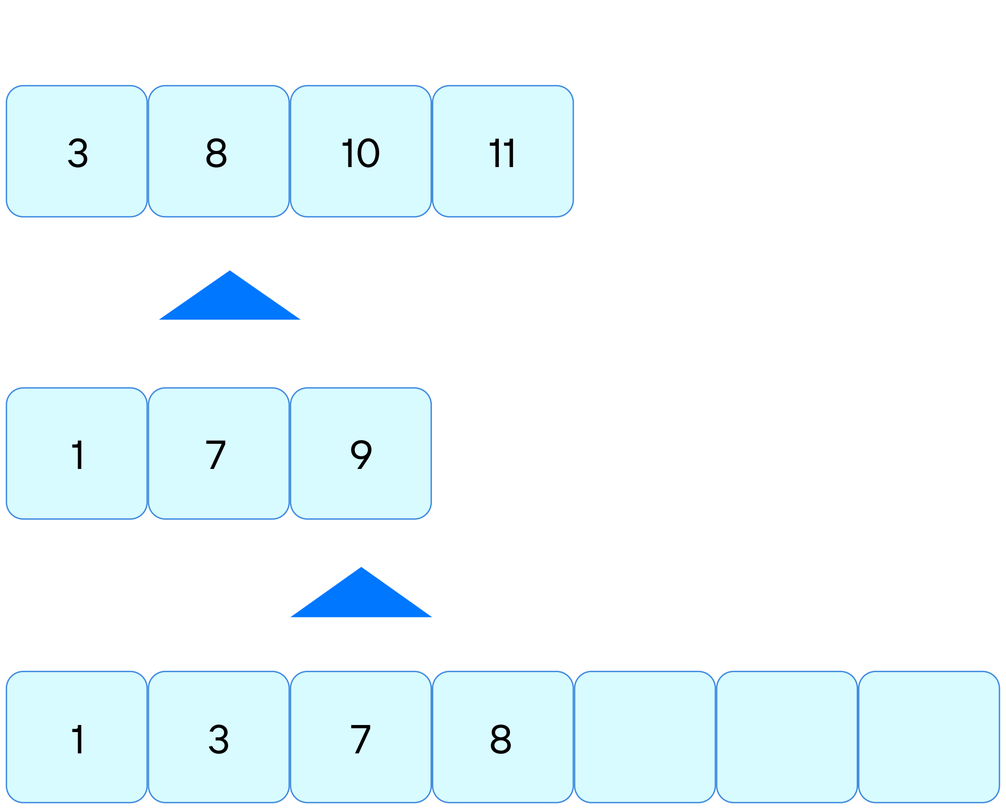
Дано: два отсортированных по возрастанию массива.

Необходимо написать функцию, которая объединит эти два массива в третий. Получившийся в итоге массив тоже должен быть отсортирован по возрастанию.

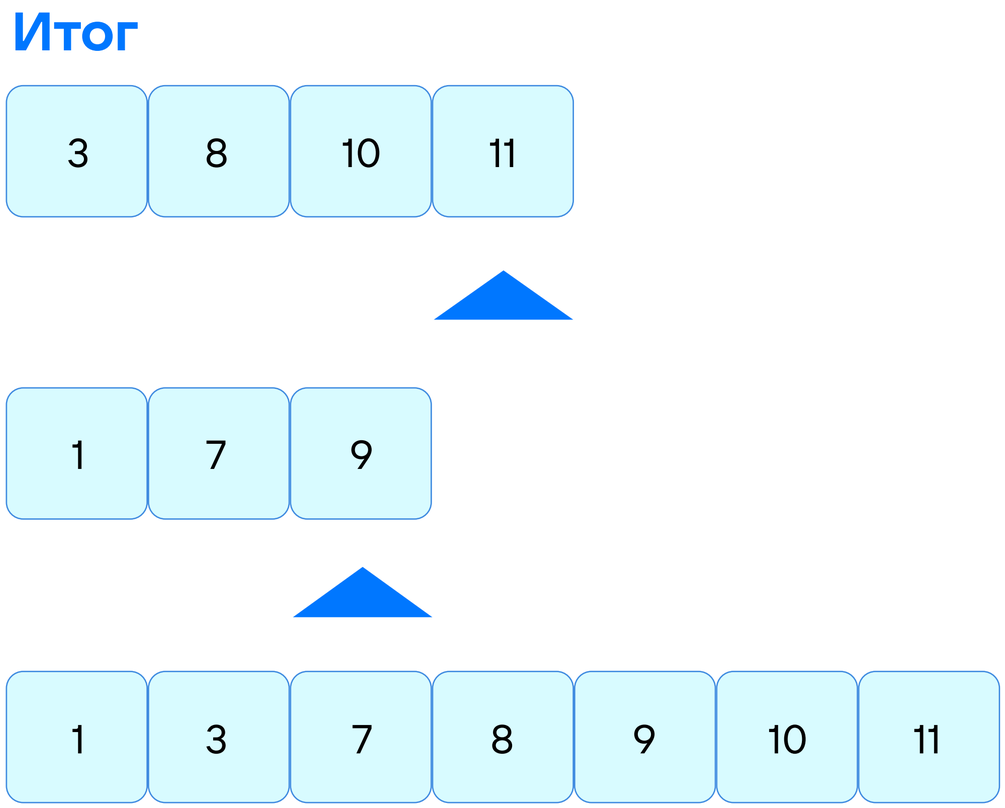
Слияние двух отсортированных массивов



Слияние двух отсортированных массивов



Итог



Слияние двух отсортированных массивов

```
function merge_sorted_arrays(arr1, arr2) {
  merged_array = []
  i = 0; j = 0

  while i < len(arr1) and j < len(arr2) {

  }
  return merged_array
}
```

Создаём две переменные — i и j. Цикл продолжается до тех пор, пока не дошли до конца одного из массивов.

Слияние двух отсортированных массивов

Сравниваем значения под индексами i и j в соответствующих массивах.

В зависимости от результата сравнения, в итоговый массив записываем элемент из первого или второго массива и инкрементируем соответствующую переменную.

```
function merge_sorted_arrays(arr1, arr2) {
  merged_array = []
  i = 0; j = 0

  while i < len(arr1) and j < len(arr2) {
    if arr1[i] < arr2[j] {
      merged_array.append(arr1[i])
      i++
    } else {
      merged_array.append(arr2[j])
      j++
    }
  }
  return merged_array
}
```

Слияние двух отсортированных массивов

Если в каком-то из массивов остались элементы, добавляем их в конец результирующего массива

```
function merge_sorted_arrays(arr1, arr2) {
  merged_array = []
  i = 0; j = 0

  while i < len(arr1) and j < len(arr2) {
    if arr1[i] < arr2[j] {
      merged_array.append(arr1[i])
      i++
    } else {
      merged_array.append(arr2[j])
      j++
    }
  }

  merged_array.append(arr1[i:])
  merged_array.append(arr2[j:])

  return merged_array
}
```

Резюме

- При попытке добавить элемент в заполненный массив его объём увеличивается в 2 раза
- При удалении элементов объём массива сокращается в два раза, когда количество элементов меньше объёма в 4 раза
- Амортизационная сложность добавления в конец осуществляется по-прежнему O(1)

education

Спасибо
за внимание!