Лекция по АиСД

Разбор задачи о минимуме в очереди. Бинарный поиск.

Минимум в очереди

Реализовать очередь с операциями
1. Push (добавляем в конец)
2. Pop (удаляем из начала)
3. GetMin (узнать минимум среди всех элементов в очереди)

Минимум в очереди

Тривиальное реешение
1. Заведем deque
2. Push – deque.push()
2. Pop – deque.popleft()
3. GetMin – Бежим по всему deque, честно ищем минимум

Сложность GetMin – O(n), хотим быстрее (O(1))

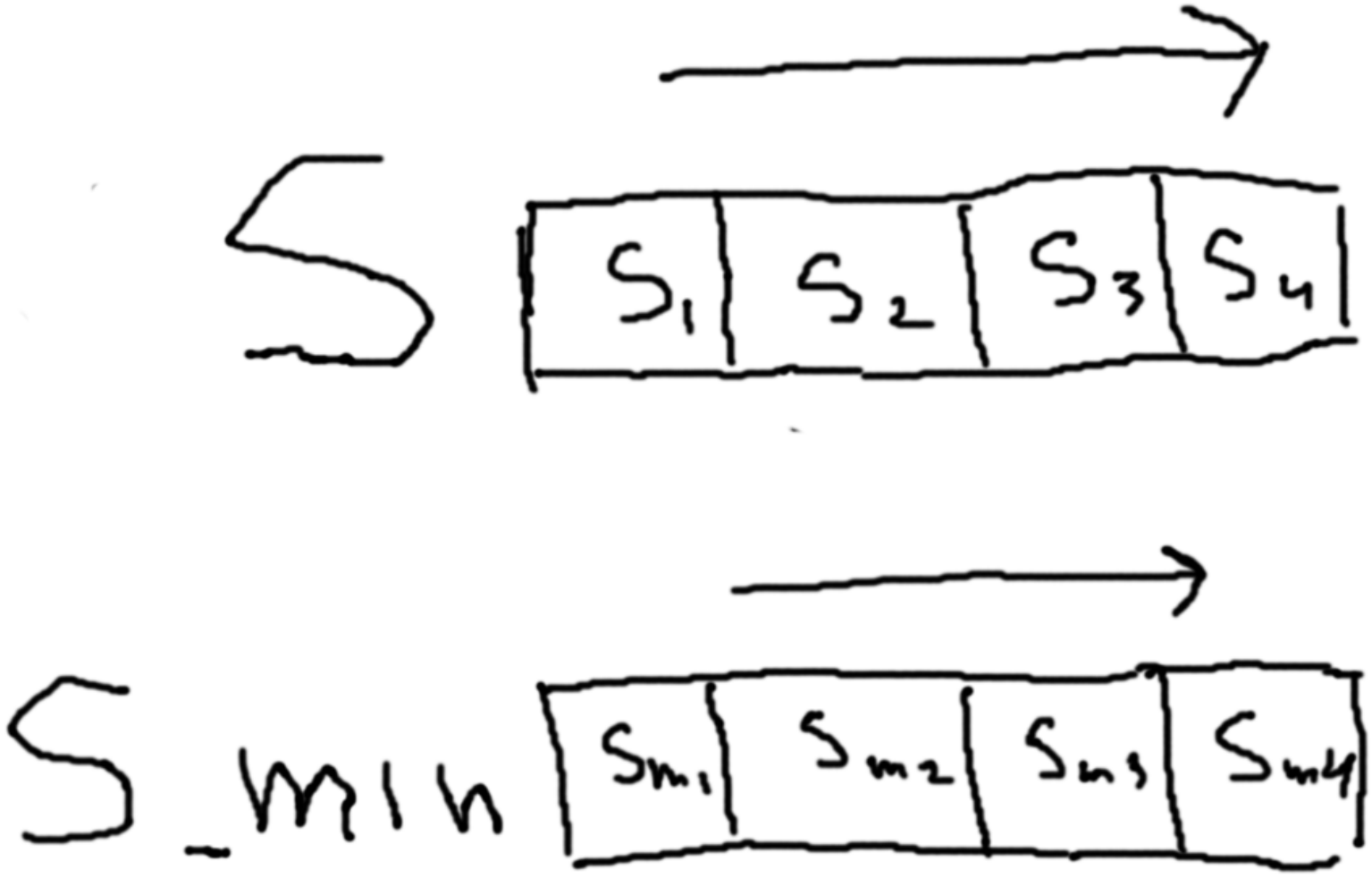
Подзадача: Минимум в стеке

Реализовать стек с операциями
1. Push (добавляем в конец)
2. Pop (удаляем из концв)
3. GetMin (узнать минимум среди всех элементов в стеке)

Все операции хотим за O(1)

Минимум в стеке

Заведем вспомогательный стек для минимумов

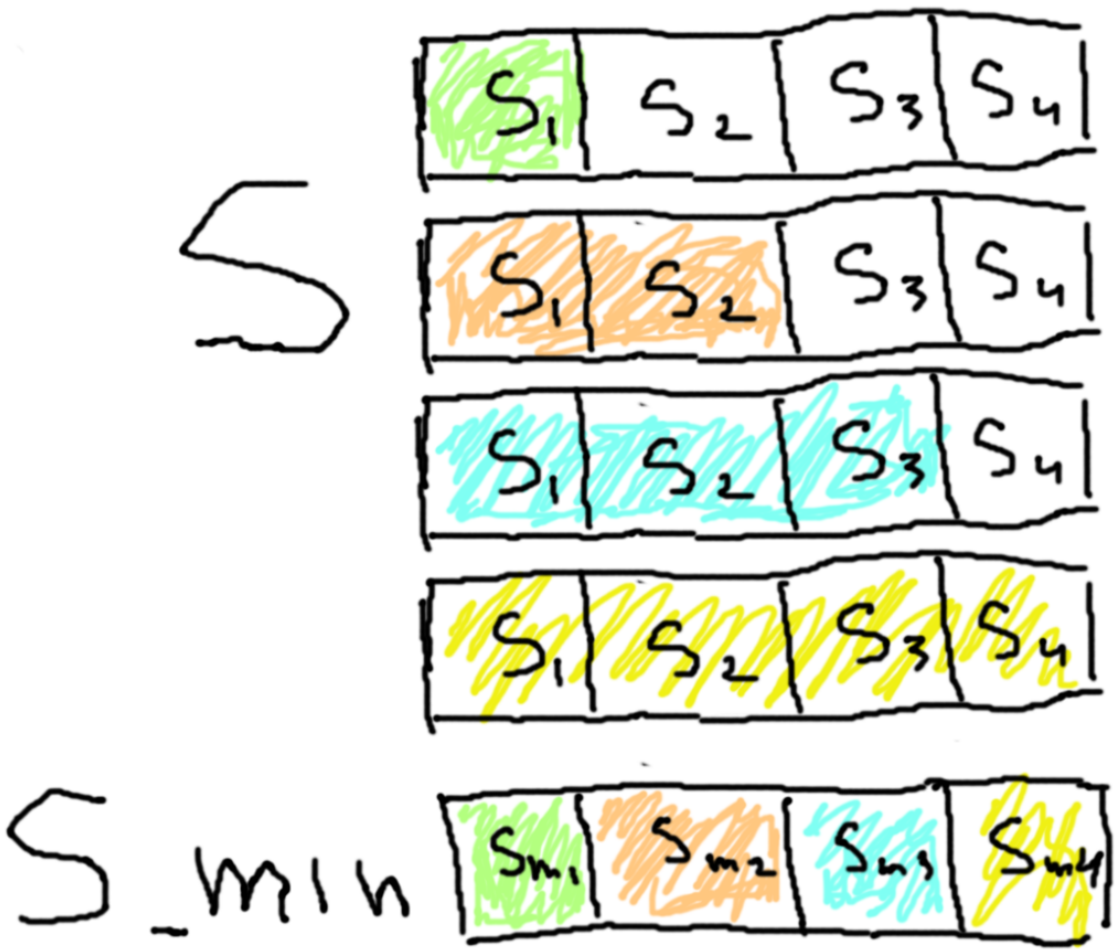


$S_{min} = Min(S_0, S_1, ..., S_i)$

Минимум в стеке

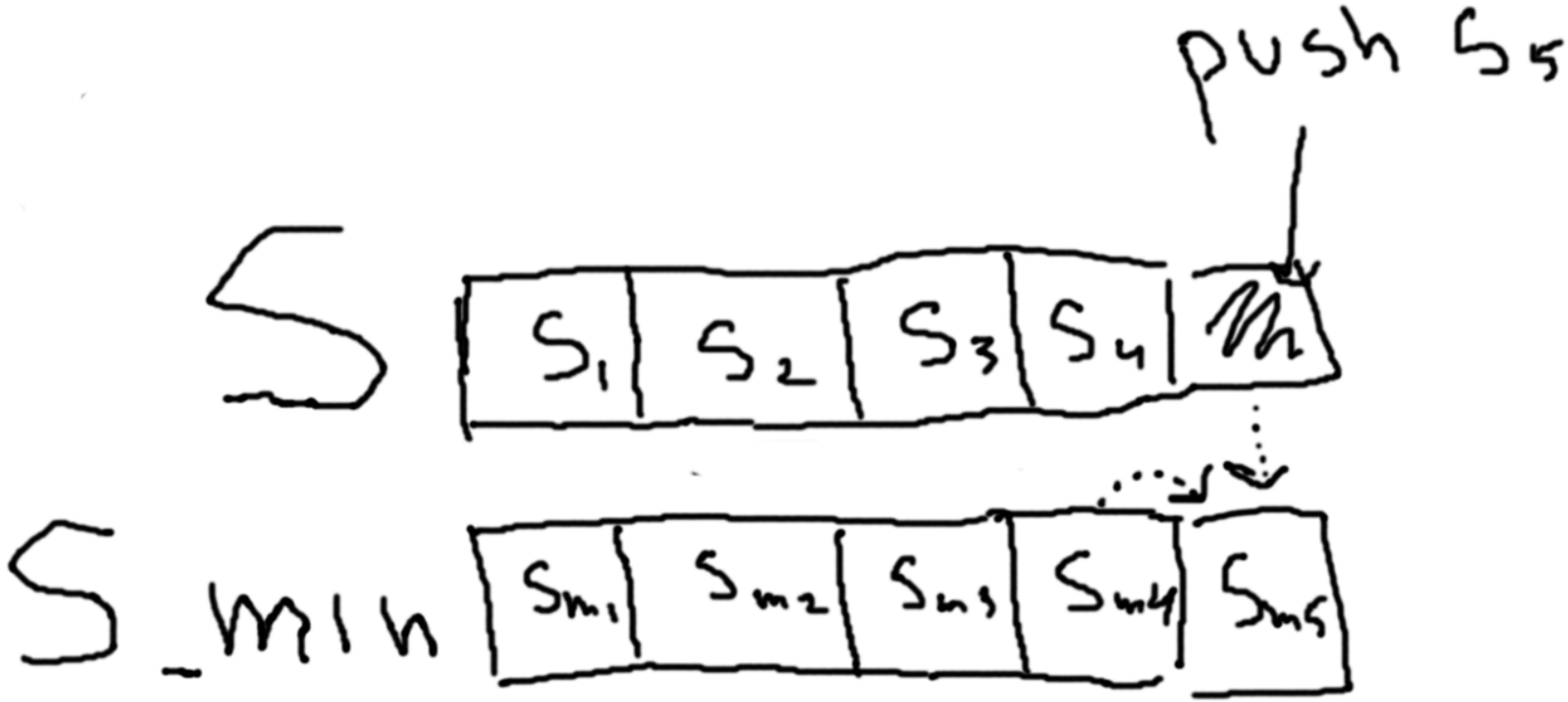
$S_{min} = Min(S_0, S_1, ..., S_i)$

Минимум в стеке



Заметим, что S_{minn} – минимум для всего стека S
Тогда ответ на запрос GetMin – вершина стека S_min

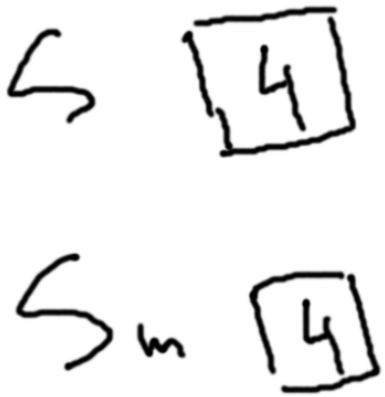
Как поддерживать стек минимумов?



$S_{min5} = Min(S_{min4}, S_5)$ – минимум для всего стека S

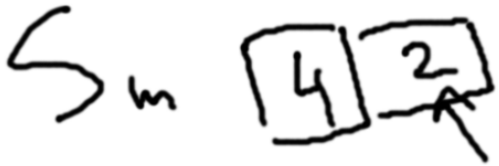
pop – просто удаляем элементы из обоих стеков

Пример



push 4
push 2
push 3
pop
pop

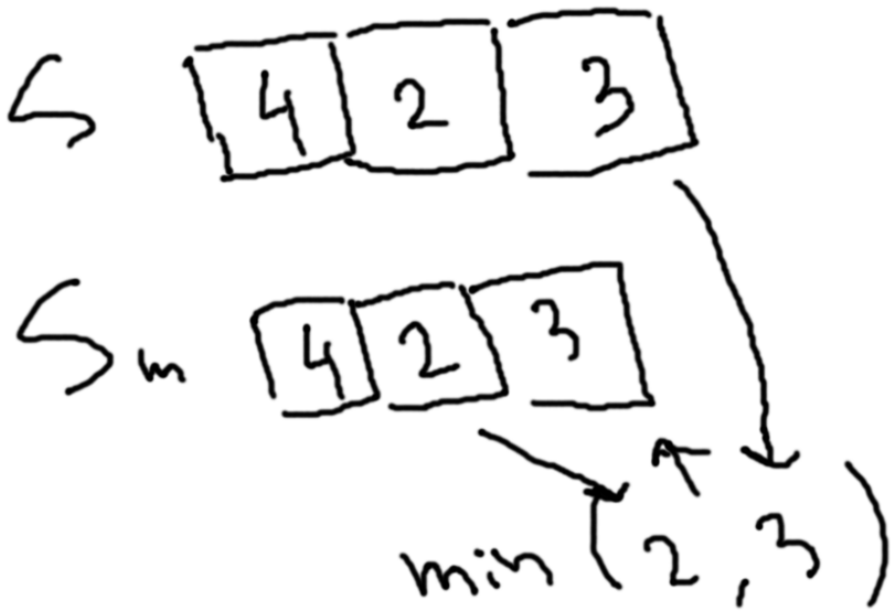
Пример



min (4,2)

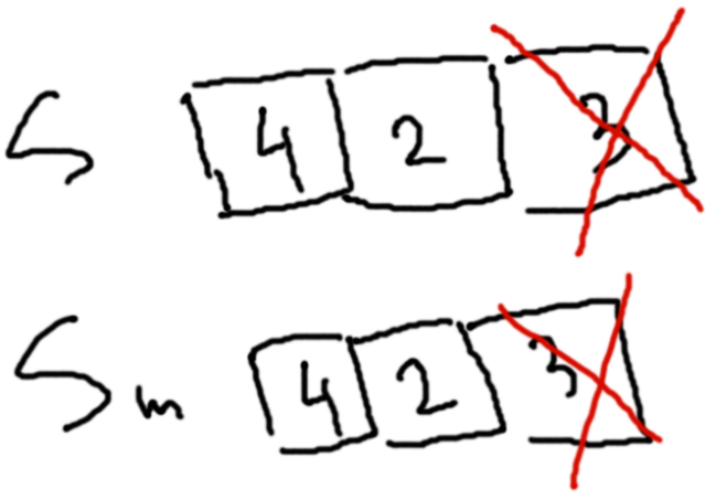
push 4
push 2
push 3
pop
pop

Пример



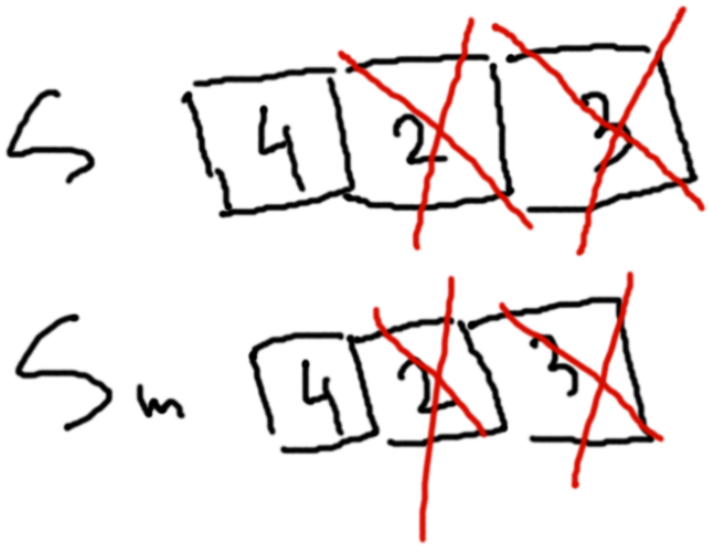
push 4
push 2
push 3
pop
pop

Пример



push 4
push 2
push 3
pop
pop

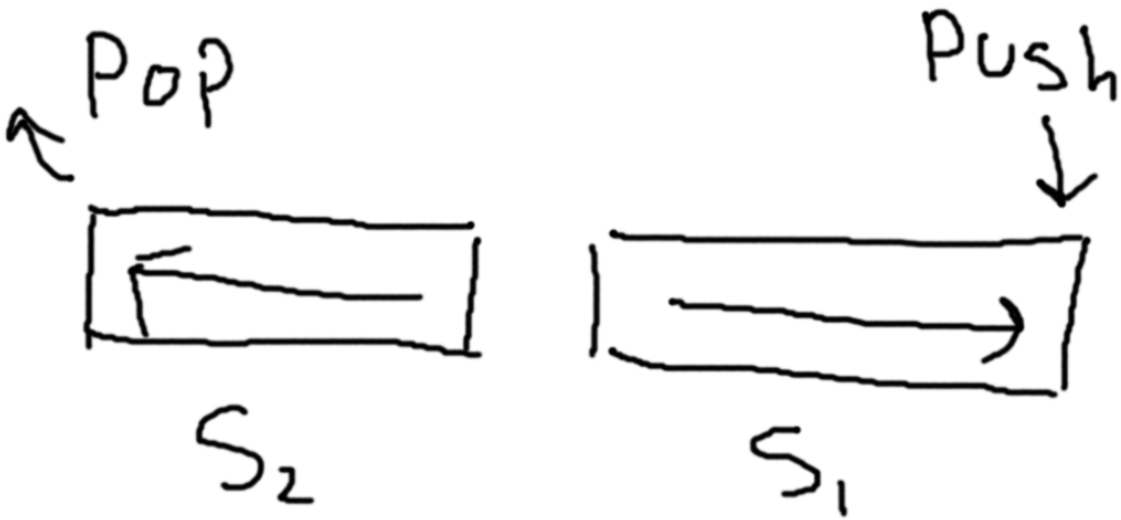
Пример



push 4
push 2
push 3
pop
pop

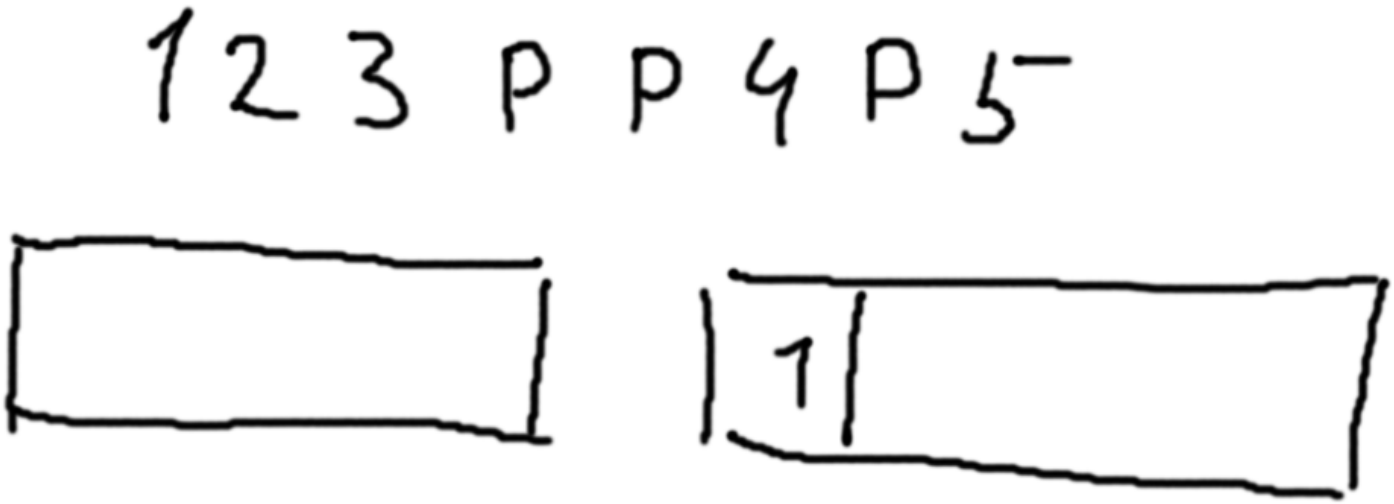
Очередь из 2х стеков

Заведем 2 стека – $S1$ и $S2$

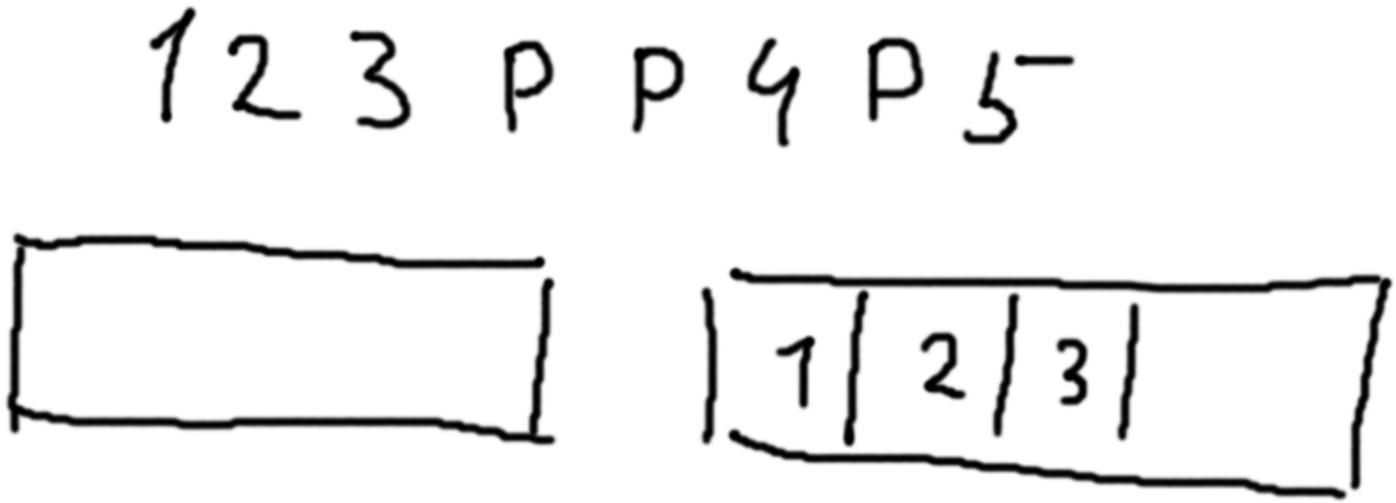


push будем делать в $S1$
pop будем делать из $S2$
Если $S2$ пуст, переложим в него все элементы из $S1$

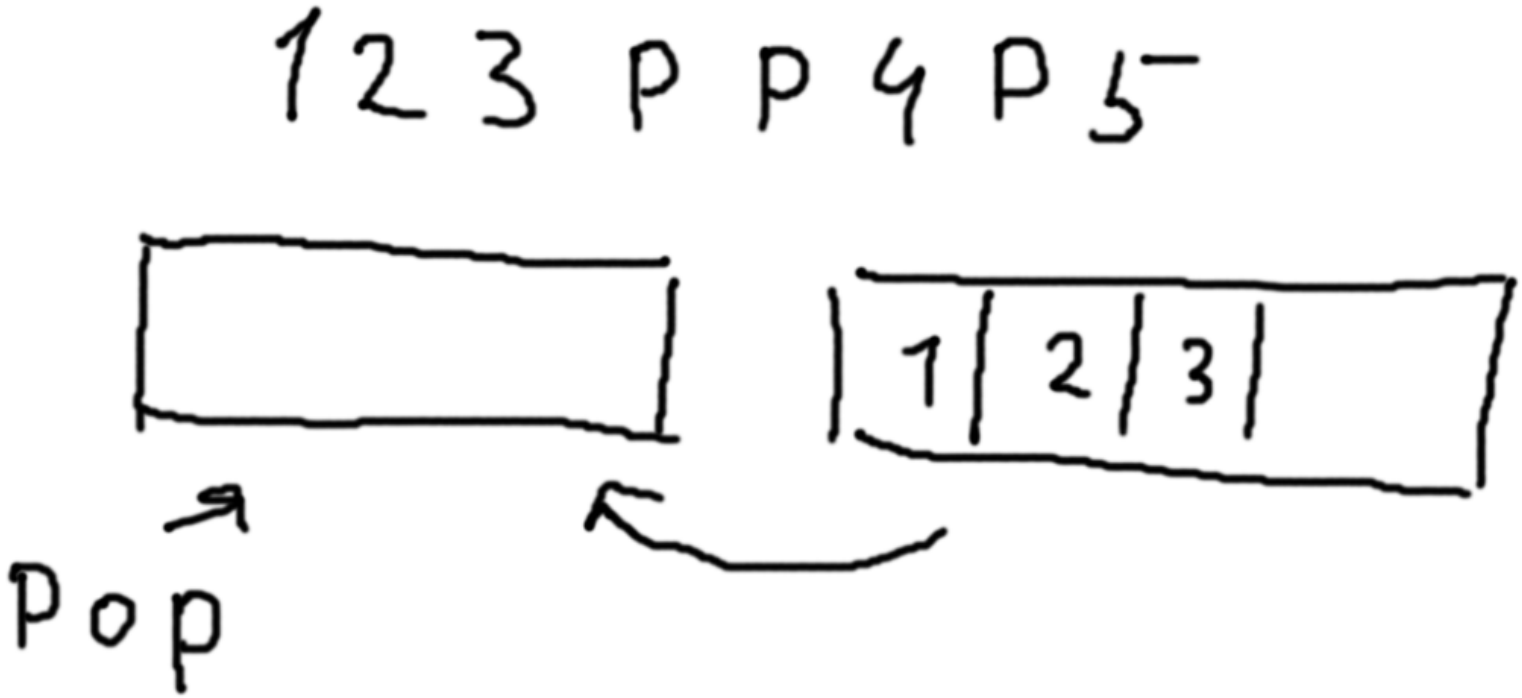
Пример



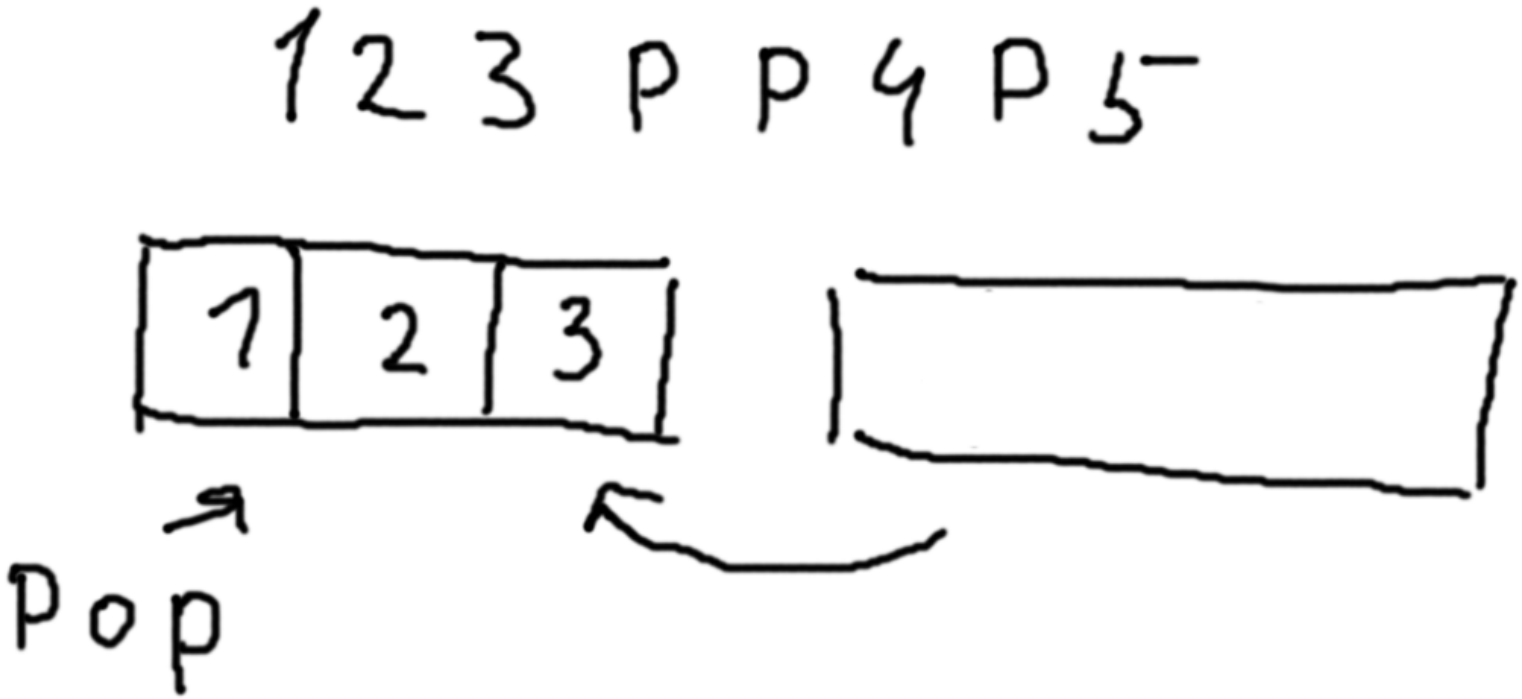
Пример



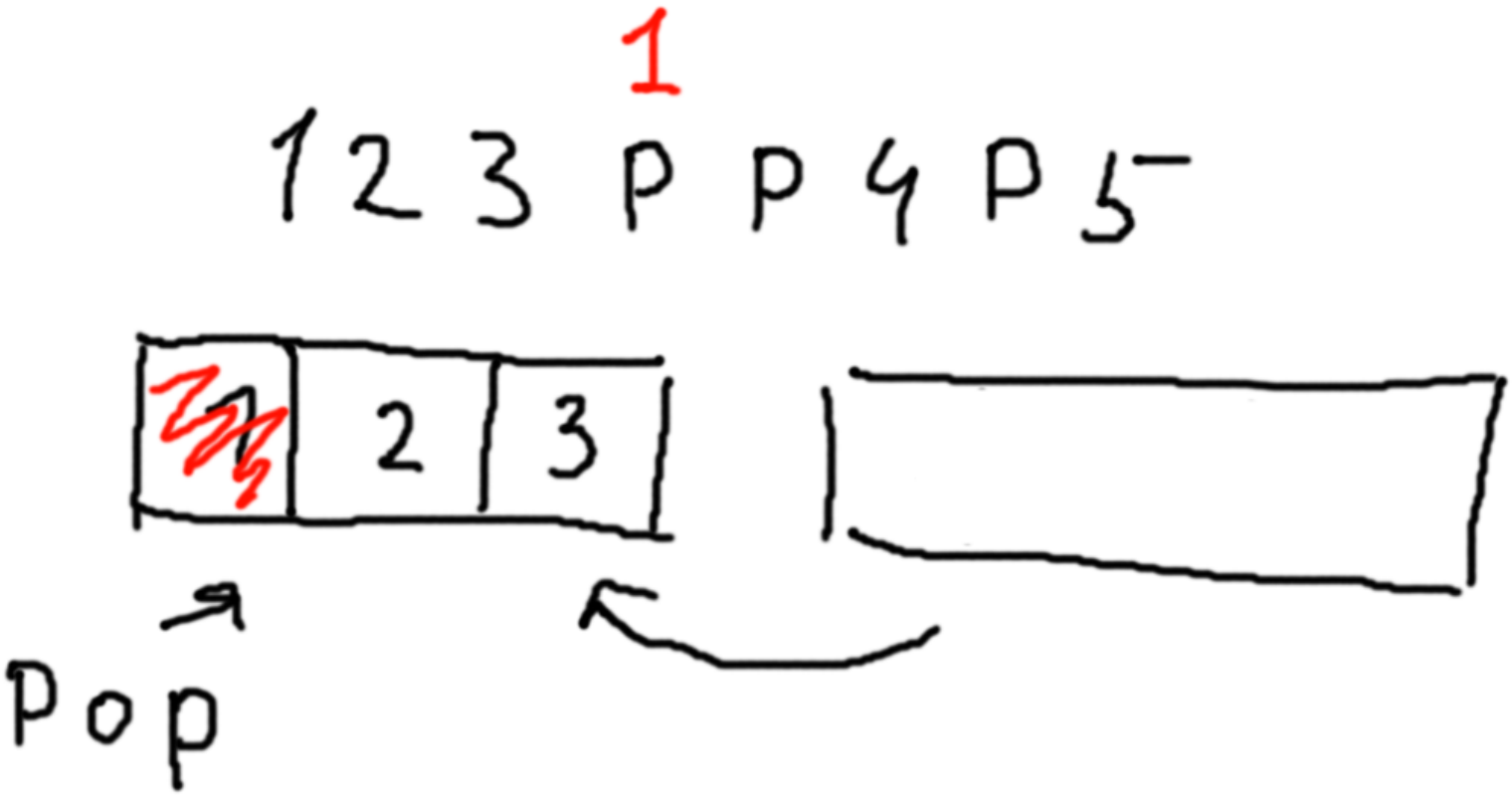
Пример



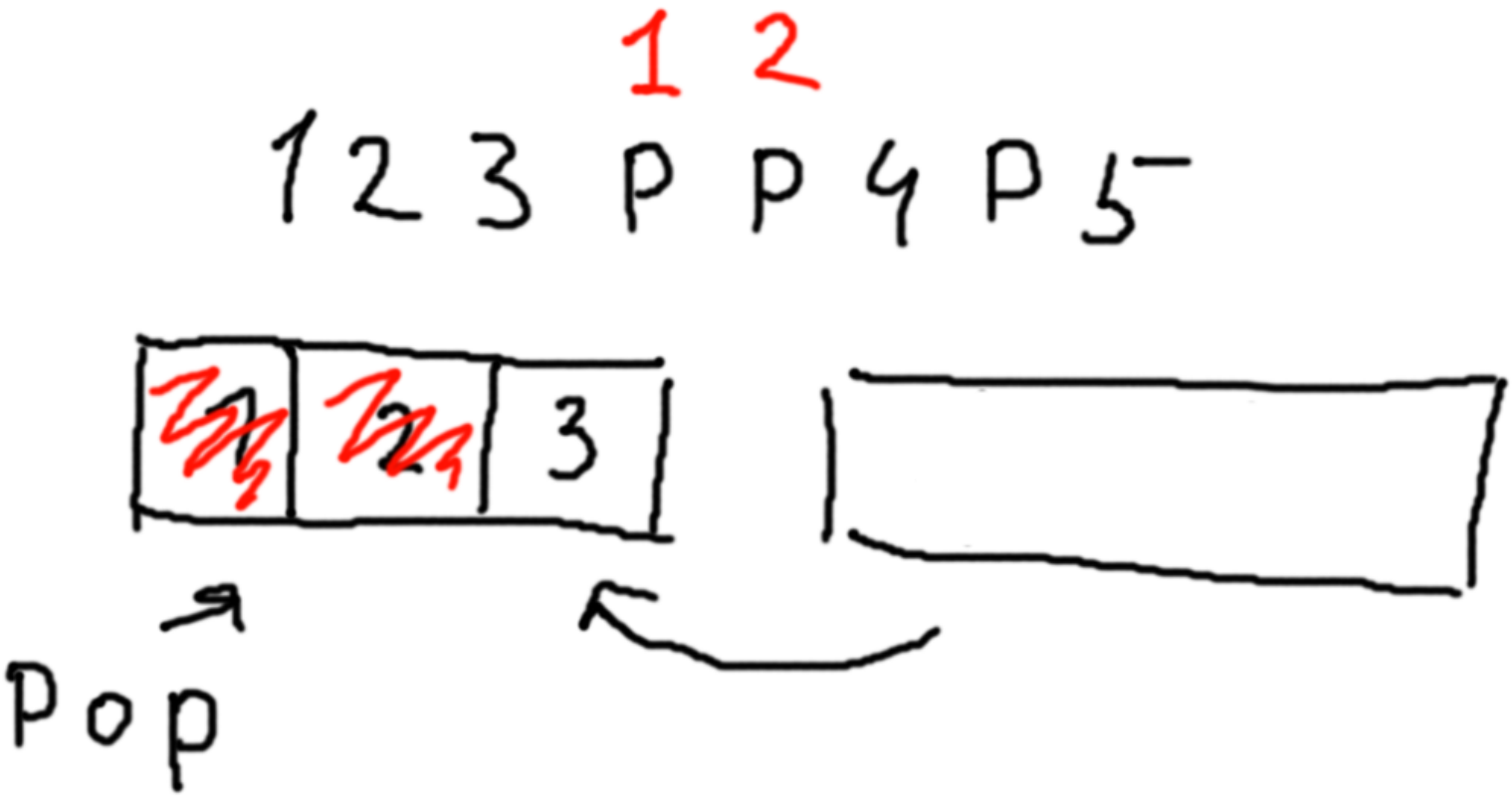
Пример



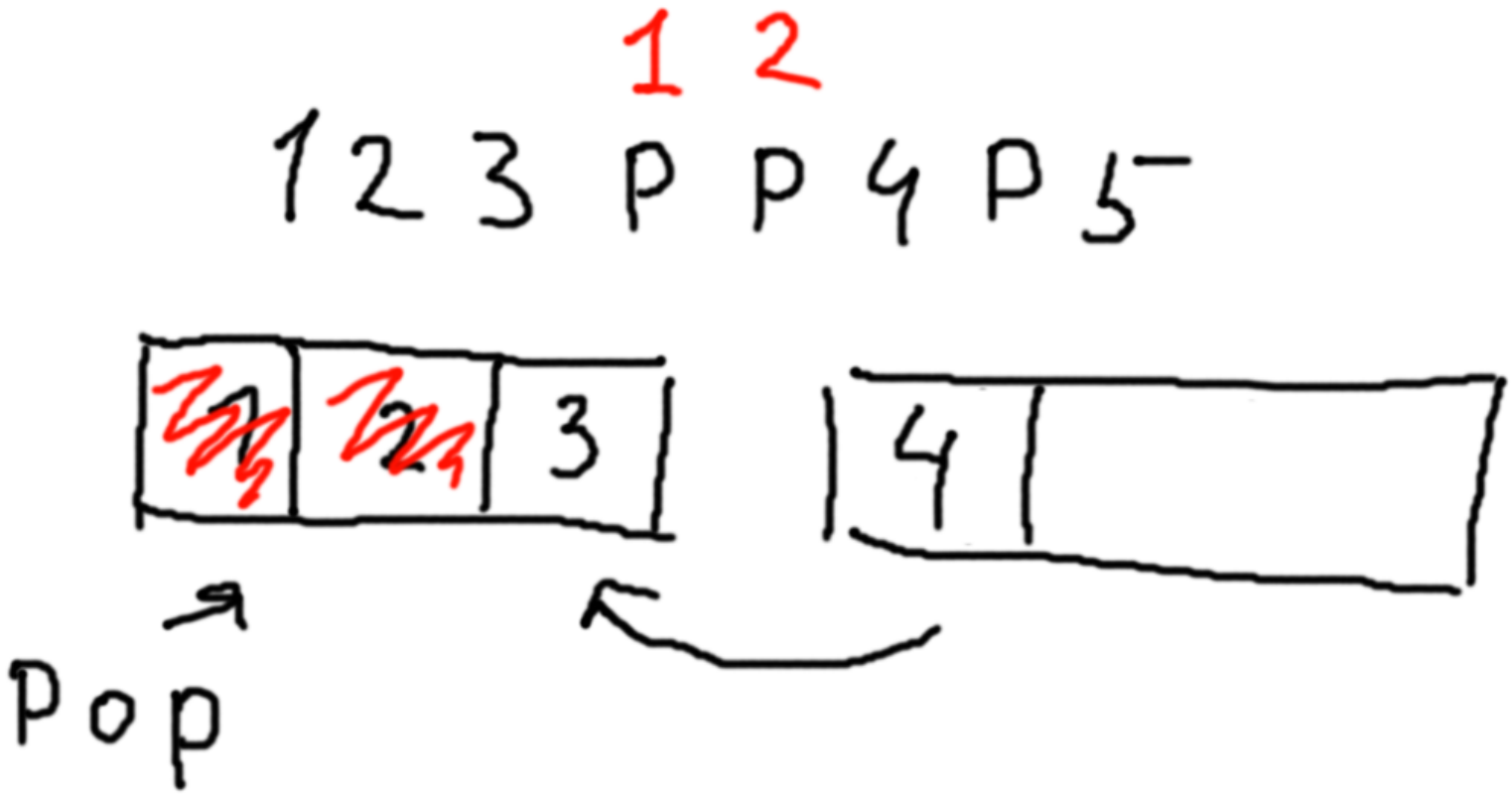
Пример



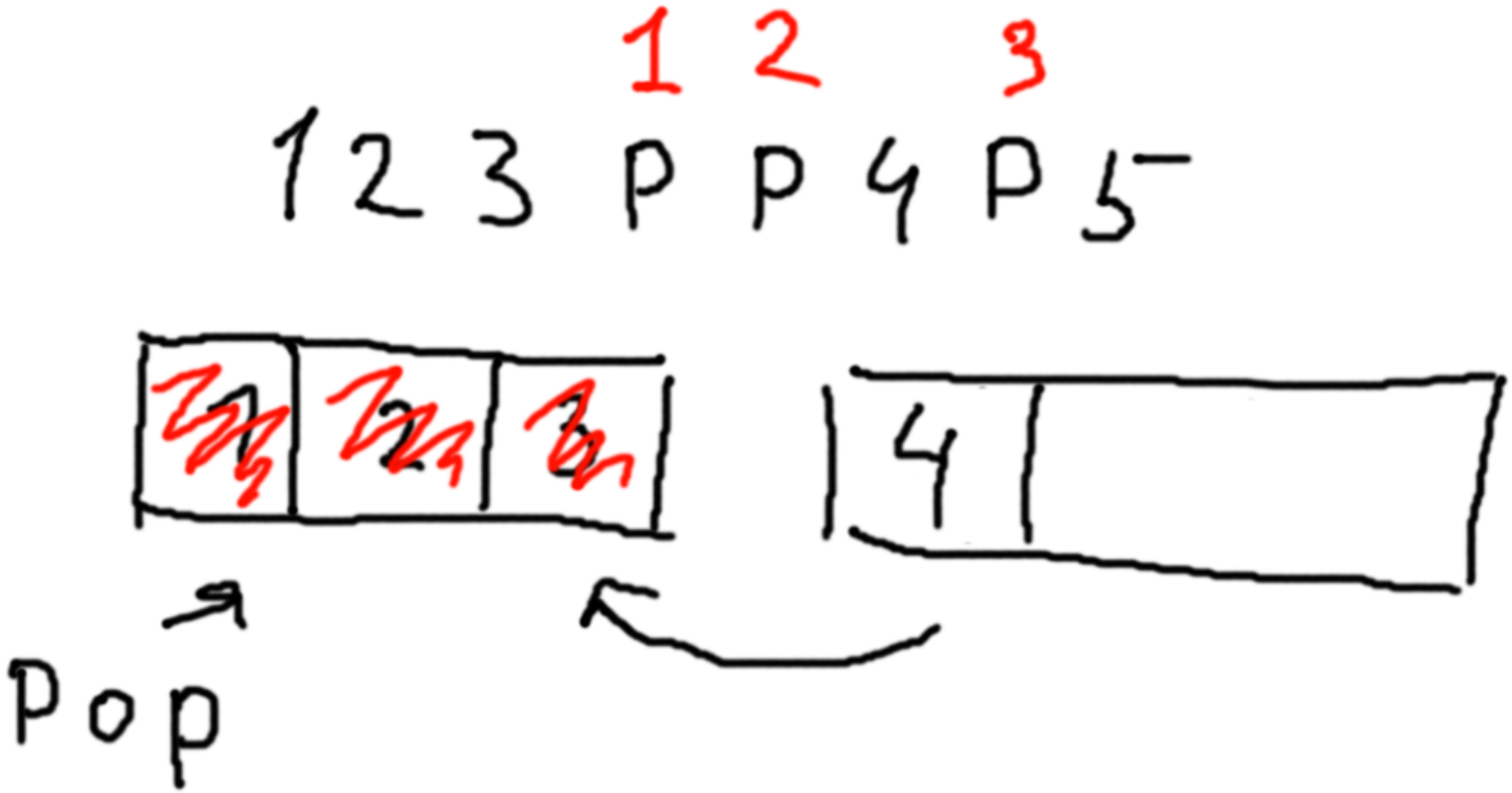
Пример



Пример



Пример



Очередь из 2х стеков

Можно заметить, что такая конструкция сохраняет порядок очереди.
Учетная стоимость push и pop при этом равна O(1)

Очередь из 2х стеков: Оценка времени работы

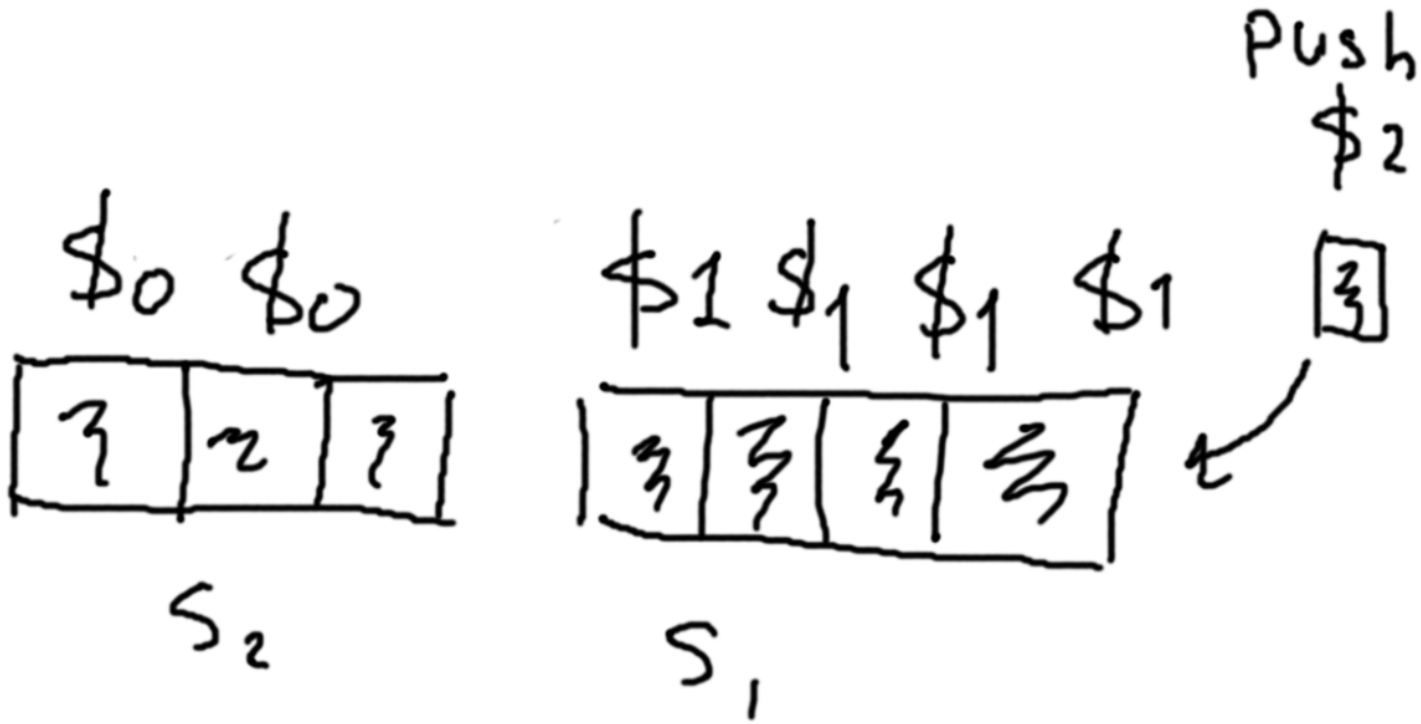
Воспользуемся банковским методом

Будем платить \$2 за push, \$1 тратить сразу, \$1 сохраним для перекладывания элементов (тогда над каждым элементом в $S1$ будет \$1)

Если нам нужно переложить элементы из $S1$ в $S2$, возьмем деньги, которые лежат над элементами в $S1$

Можно заметить, что каждый элемент мы переложим только 1 раз

Очередь из 2х стеков: Оценка времени работы



Минимум в очереди

Для реализации очереди, поддерживающей минимум, сделаем очередь из 2x стеков, поддерживающих минимум ($S1$ и $S2$)

Теперь GetMin в очереди – $Min(GetMin(S1), GetMin(S2))$

Бинарный поиск

Пусть у нас есть отсортированный массив $a1, a2, ..., a_n$, $a_i < a_{i+1}$.

Мы хотим найти в нем число x

Тривиальное решение

Пишем цикл

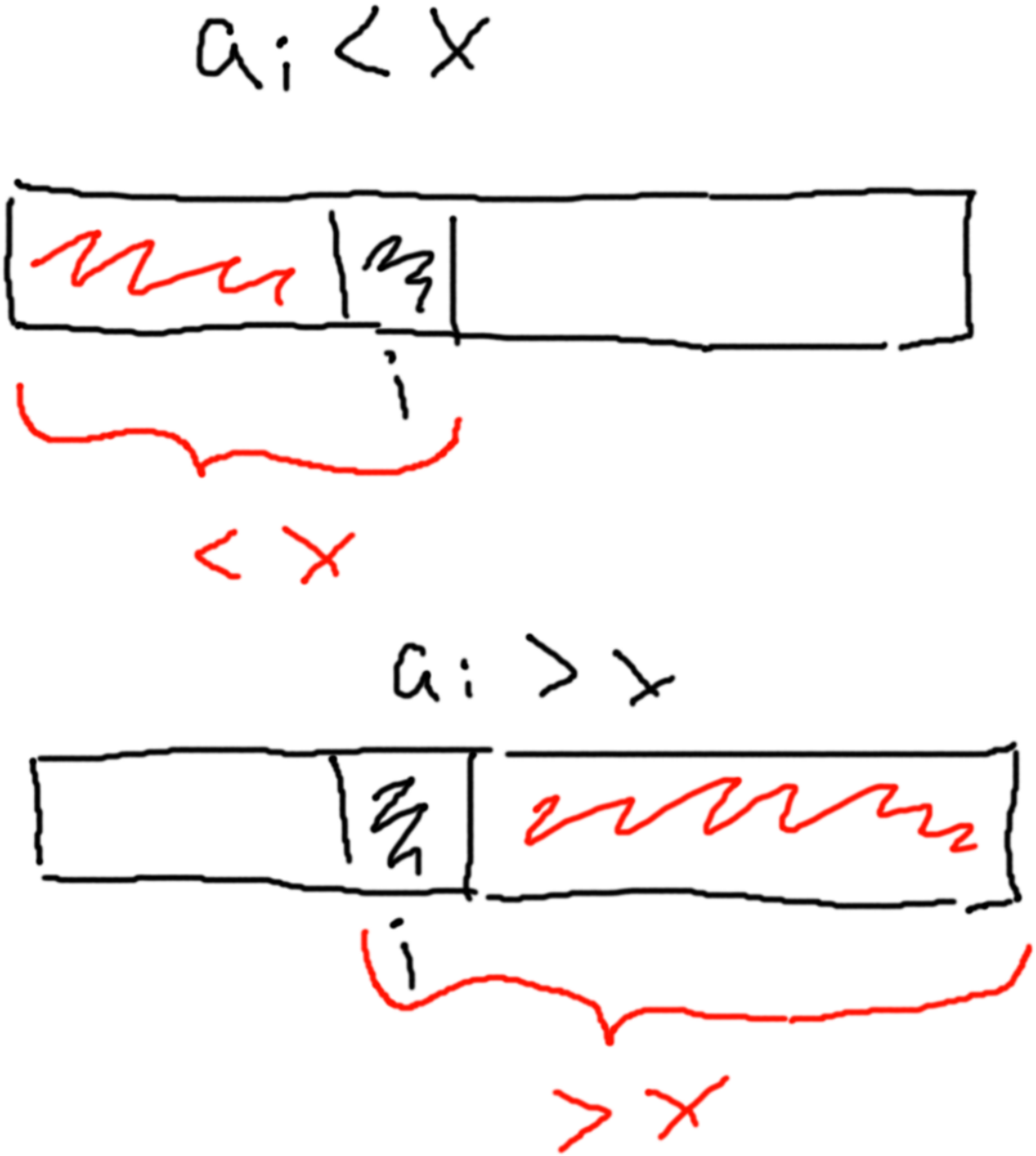
```
for elem in a:
    if elem == x:
        return True
return False
```

Сложность – O(n), хотим быстрее

Бинарный поиск

Рассмотрим некоторый элемент массива a_i и сравним его с x
Вспомним, что массив отсортирован
1. Если $a_i < x$, то и все $a_j < x, j < i$ (Левее i нашего элемента точно нет)
2. Если $a_i > x$, то и все $a_j > x, j > i$ (Правее i нашего элемента точно нет)
3. Если $a_i = x$, то мы нашли наш элемент

Бинарный поиск



Бинарный поиск

Пусть l, r – границы отрезка, в котором мы сейчас ищем элемент

Изначально $l = 0$, $r = \text{len(a)}$, то есть ищем во всем массиве

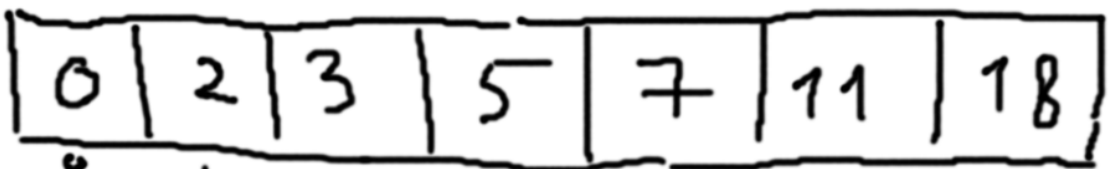
Рассмотрим середину текущего отрезка $a[m], m = (l + r)//2$ 1.
Если $a_m < x$, то продолжим поиск только в правой половине массива (l = m)
2. Если $a_m > x$, то продолжим поиск только в левой половине массива (r = m)
3. Если $a_m = x$, то мы нашли наш элемент

Остановимся, когда l + 1 = r (когда мы сошлись к отрезку из одного элемента)

Сложность – O(log len(a))

Пример

$X = 7$



$l = 0$

$2^2 = 7$

Пример

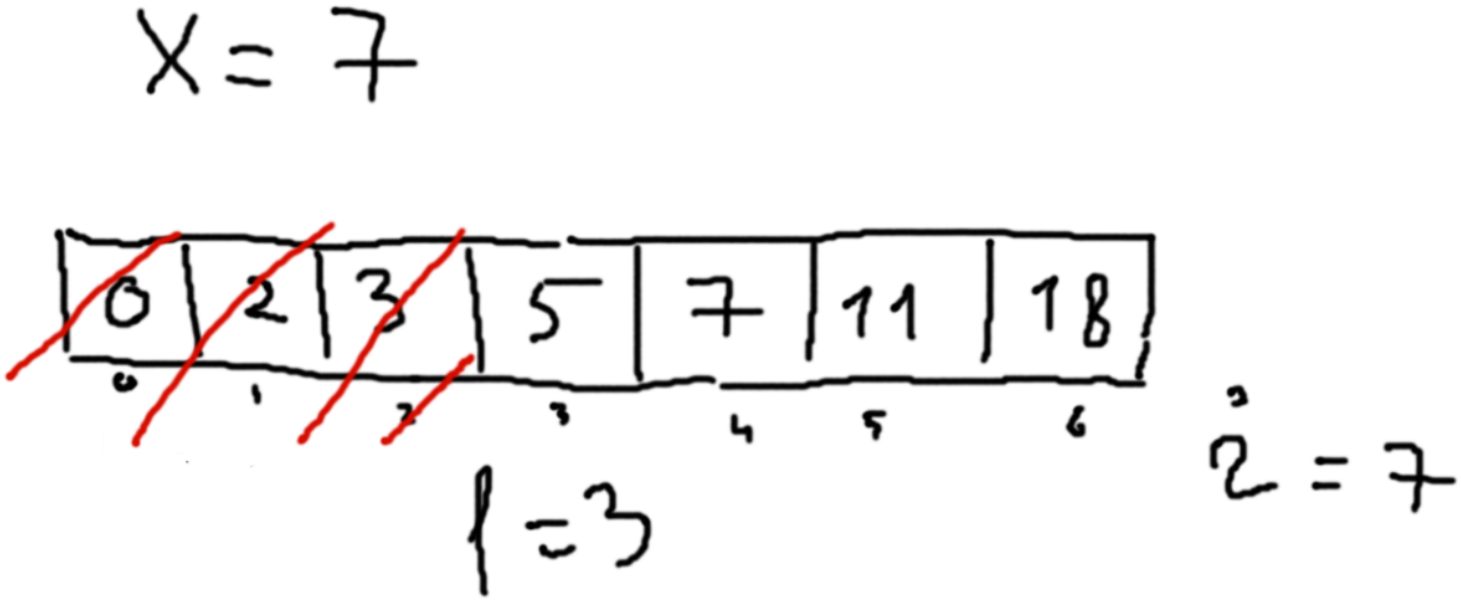
$X = 7$

$$m = \frac{0 + 7}{2} = 3$$

$a[m] < X \Rightarrow l = m$

$5 \quad 7$

Пример



Пример

$X = 7$

$l = 3$

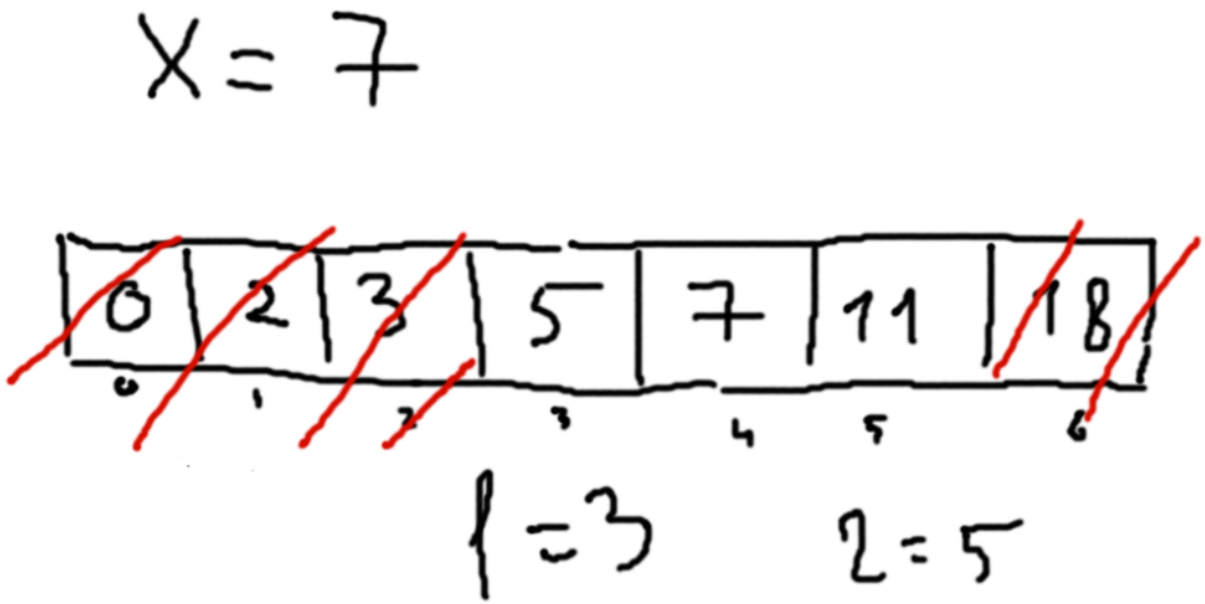
$r = 7$

$m = \frac{3+7}{2} = 5$

$a[m] > x \Rightarrow r = m$

11 7

Пример



Пример

$X = 7$

$l = 3 \quad r = 5$

$$m = \frac{l + r}{2} = \frac{3 + 5}{2} = 4$$

$a[m] = X!$

Бинарный поиск: код

```
def binary_search(a, k):
    l = 0
    r = len(a)

    while (r - l > 1):
        m = l + (r - l) // 2
        if a[m] > k:
            r = m
        else:
            l = m
    return a[l] == k
```