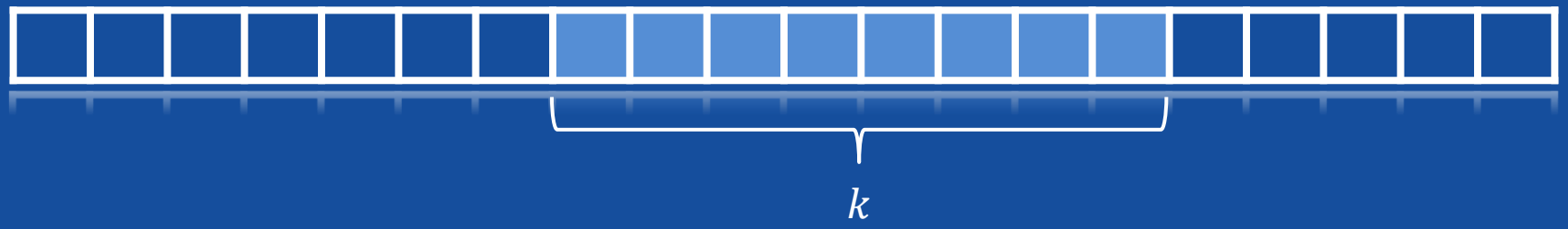
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Структуры данных для выполнения интервальных запросов



С. А. Соболь — магистр математики и информационных технологий
Е.П. Соболевская — *доцент кафедры ДМА ФПМИ БГУ*

Минск, 2025

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Постановка задачи

Пусть в памяти хранится последовательность из n элементов.

Нужно уметь выполнять какую-либо операцию над k **последовательными элементами сразу** (например, суммировать, находить минимум/максимум).

Такие запросы называют ***интервальными***, потому что они затрагивают целый интервал значений.

Терминология

- Путаница
  - Промежуток
  - Интервал
  - Отрезок (сегмент)
- Будем использовать в основном целочисленные полуинтервалы (левый конец включён, правый не включён):

$$[l, r) = \{l, l + 1, l + 2, \dots, r - 1\}$$

- В языках программирования часто правый конец диапазона не включается (C++, Python)

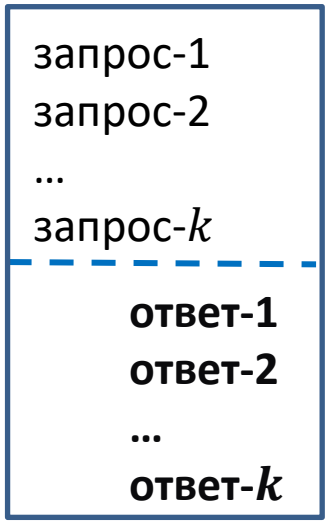
Классификация задач

Если элементы последовательности не изменяются (не предусмотрено выполнение операции модификации элемента), то в названии задачи фигурирует слово **статическая** (англ. **static**), иначе– **динамическая** (англ. **dynamic**).

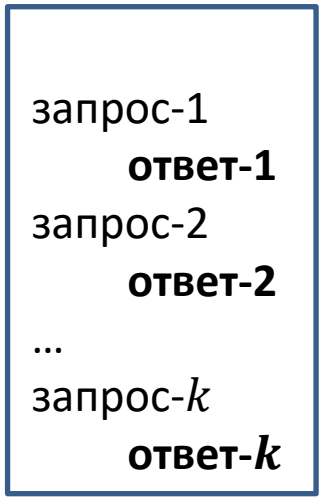
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Классификация задач

Если сначала все интервальные запросы поступили (после чего они все могли быть проанализированы), а только потом формируются ответы на них, то говорят про **офлайн** (англ.**off line)** версию задачи.



Если же поступает запрос, сразу даётся на него ответ и только после ответа на предыдущий запрос идёт следующий запрос, то говорят про **онлайн** (англ. **online)** версию задачи.



Мы будем в нашей лекции рассматривать online версию задачи.

Постановка задачи

Выполнить один любой интервальный запрос можно, конечно, «в лоб» циклом по k элементам интервала — это $\Omega(k)$.

Можно ли это сделать быстрее?

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Постановка задачи

Покажем (на примере задач о сумме и минимуме на интервале), что с помощью специальных структур:

- **корневая эвристика;**
- **дерево отрезков;**
- **разрежённая таблица;**

выполнить интервальный запрос можно быстрее.

> При этом сначала над данными выполняют **препроцессинг (предподсчёт)** – предварительная подготовка, которая в последующем позволит эффективно обрабатывать запросы. Очевидно, что время предподсчёта должно быть не больше, чем суммарное время ответа на все запросы.

Задача RSQ

RSQ — Range Sum Query (запрос суммы на отрезке)

- Задана последовательность из n чисел:

$$A = [a_0, a_1, a_2, \dots, a_{n-1}]$$

- Поступают запросы двух типов:
  - ✓запрос модификации $\mathbf{Add}(\boldsymbol{i}, \boldsymbol{x})$ — прибавить к i-му элементу число x;
  - ✓запрос суммы $\mathbf{FindSum}(\boldsymbol{l}, \boldsymbol{r})$ — вычислить сумму на $[l, r)$

$$S = \sum_{k=l}^{r-1} a_k$$

Задача RSQ. Обычный массив

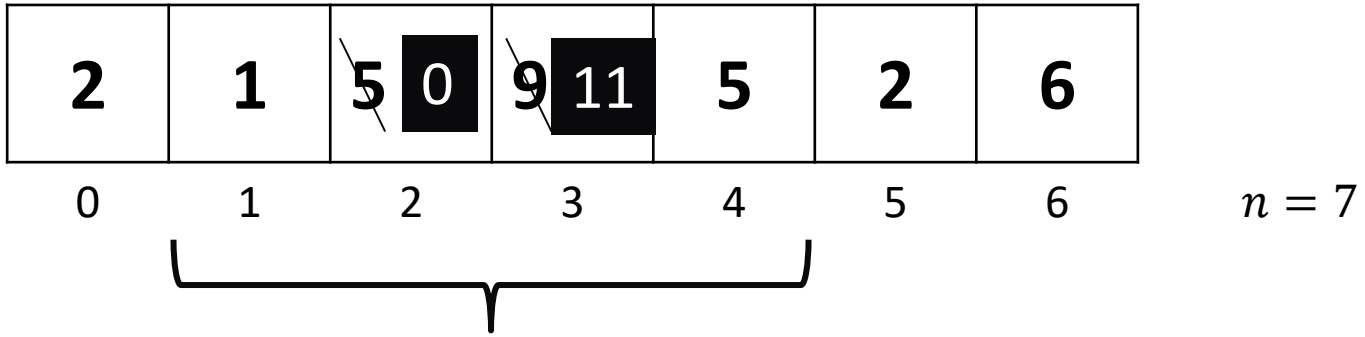
Используем **обычный массив**, в котором размещены элементы:

$\mathrm{Add}(3, 2)$

$\mathrm{Add}(2, -5)$

$\mathrm{FindSum}(1, 5)$

$1 + 0 + 11 + 5 = 17$



Время работы:

модификация	$\Theta(1)$
сумма	$\Theta(n)$

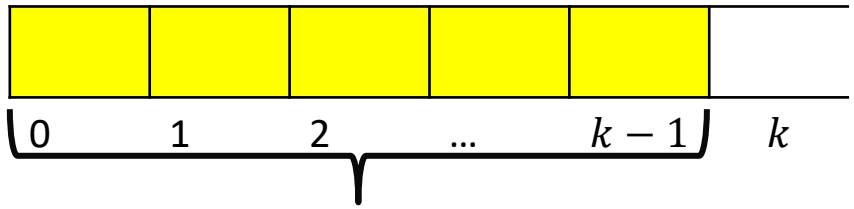
хотелось бы быстрее

Задача RSQ. Префиксные суммы

Выполним предподсчёт

Введём понятие частичной суммы, или суммы на префиксе:

$$s_k = \sum_{i=0}^{k-1} a_i = a_0 + a_1 + \cdots + a_{k-1}$$

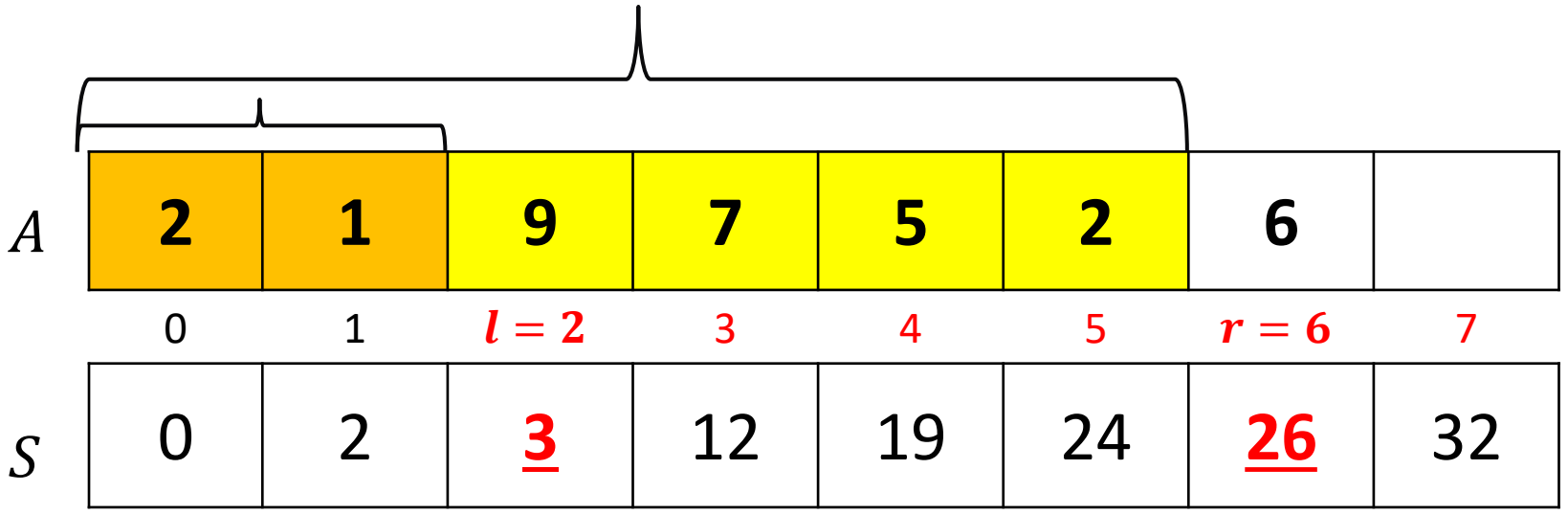


По исходной последовательности A **массив S строится за $O(n)$**, используя следующее рекуррентное соотношение:

$$s_0 := 0,$$
$$s_i := s_{i-1} + a_{i-1}, \qquad i = 1, \ldots, n$$

Задача RSQ. Префиксные суммы

Сумма на $[l, r)$ равна $s_r - s_l$

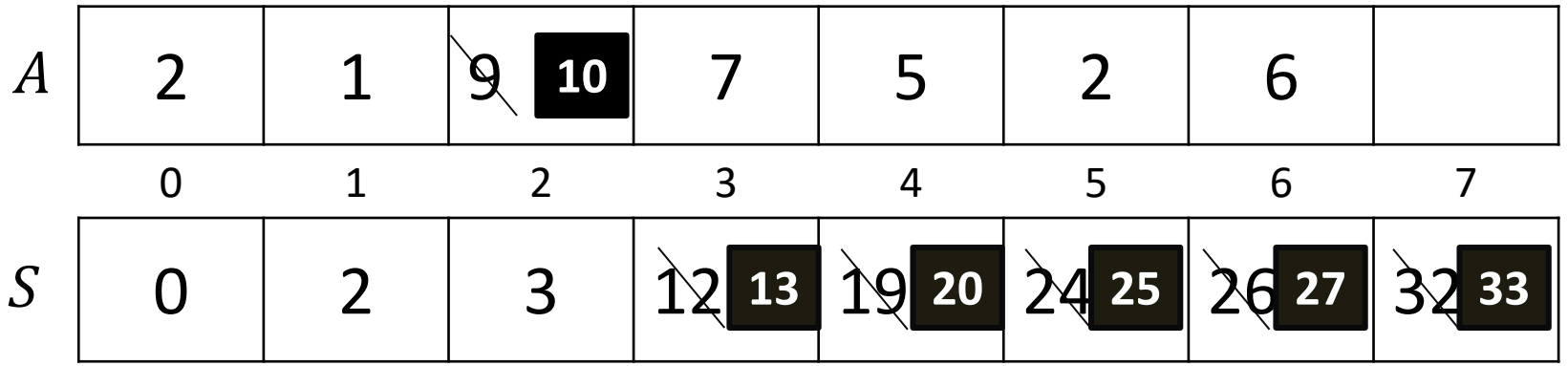


$\text{FindSum}(2, 6) = s_6 - s_2$
$= 26 - 3 = 23$

Время на запрос суммы: $O(1)$

Задача RSQ. Префиксные суммы

Add(i, x)



$Add(2, 1)$

Время на запрос модификации: $\Theta(n)$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Задача RSQ. Префиксные суммы

A	**2**	**1**	**9**	**7**	**5**	**2**	**6**	
	0	1	2	3	4	5	6	7
S	**0**	**2**	**3**	**12**	**19**	**24**	**26**	**32**

Время на предподсчёт	$\Theta(n)$
Время на запрос модификации	$\Theta(n)$
Время на запрос суммы	$\Theta(1)$
Объем дополнительной памяти	$\Theta(n)$

можно $O(1)$, если использовать память, выделенную под массив A

Оценки

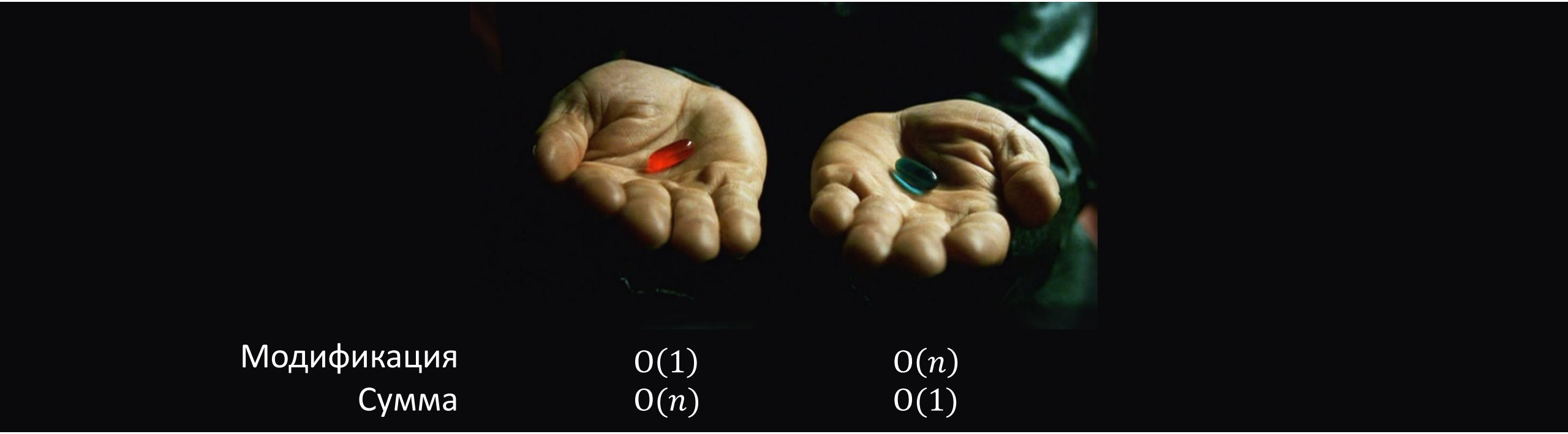
	Обычный массив	Префиксные суммы
Время на предподсчёт		$\Theta(n)$
Время на запрос модификации	$\Theta(1)$	$\Theta(n)$
Время на запрос суммы	$\Theta(n)$	$\Theta(1)$
Объем дополнительной памяти		$\Theta(n)$

$O(1)$, если хранить при предподсчёте массив префиксных сумм на месте исходного массива

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Промежуточный вариант

- Одна операция быстрая, вторая медленная
- Нужно компромиссное решение

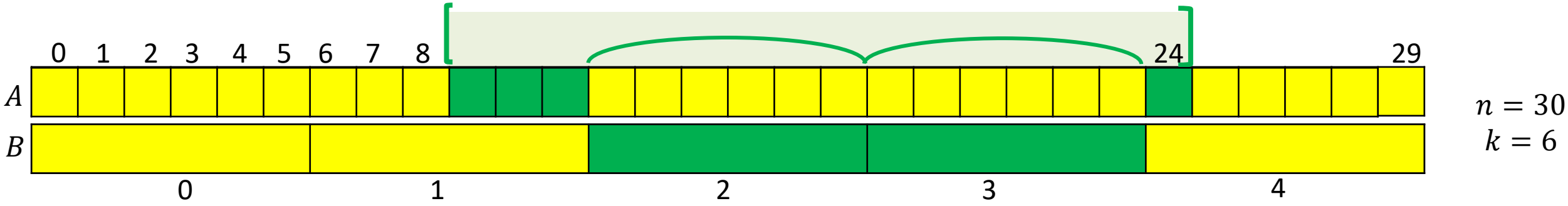


Модификация	$O(1)$	$O(n)$
Сумма	$O(n)$	$O(1)$

RSQ. Блоки

Выполним предподсчёт

- Можно просчитать заранее суммы для блоков определённого размера k (последний блок может быть меньше)
- При суммировании в цикле можно будет «перепрыгивать» блоки

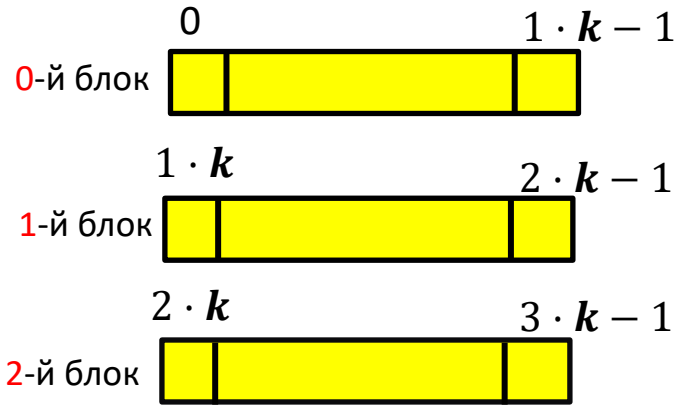


Для этого кроме исходного массива A, создадим массив B размера $\left\lceil \frac{n}{k} \right\rceil$ для хранения сумм по блокам.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

RSQ. Блоки

Удобно нумеровать блоки с нуля (на рис. размер блока $\boldsymbol{k = 6}$)



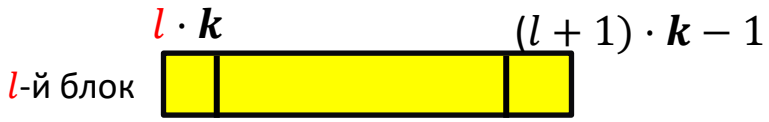
Пусть i индекс элемента в массиве A, в какой блок b_l он попадёт?

$$l \cdot k \le i < (l+1) \cdot k$$

$$\frac{i}{k} - 1 < l \le \frac{i}{k}$$

$$\frac{i}{k} - 1 < \left\lfloor \frac{i}{k} \right\rfloor \le \frac{i}{k}$$

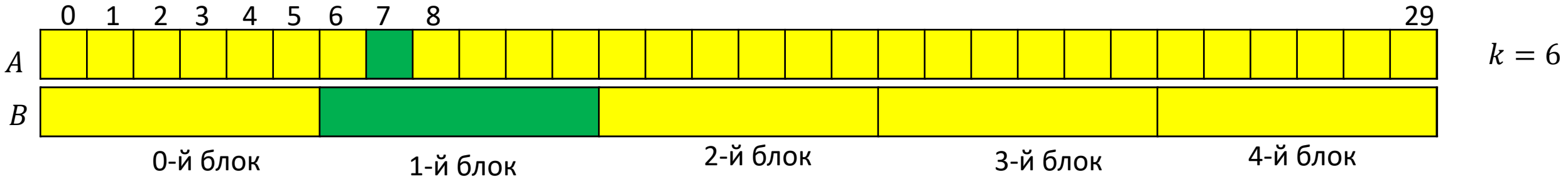
$$x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$$



Элемент массива A с индексом i попадает в блок $b_{\lfloor i/k \rfloor}$.

Выполнение операций

При **модификации** нужно изменить a_i и $b_{\lfloor i/k \rfloor}$.




```
def Add(self, i, x):
        self.a[i] += x
        self.b[i // self.k] += x
```

Время выполнения операции модификации — $O(1)$.

Выполнение операций

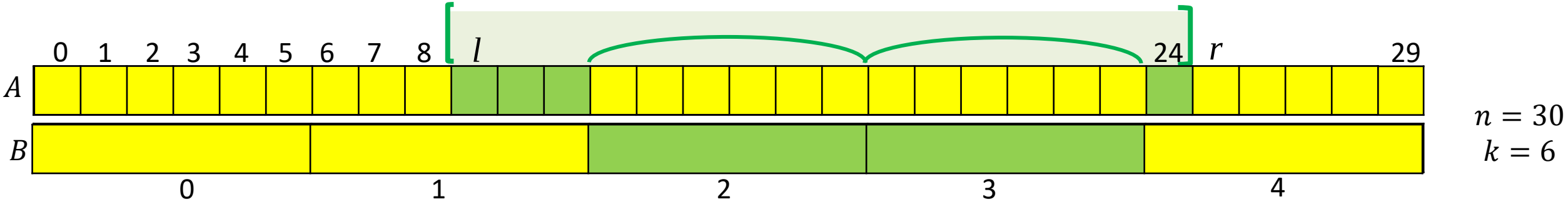
При **суммировании** на $[l, r)$:

Определяем номера блоков, к которым относятся l и r.

Суммируем от l до границы блока за $O(k)$ по массиву A.

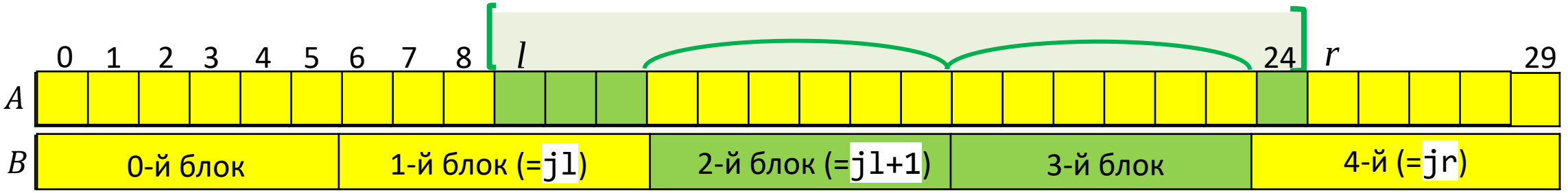
Суммируем блоки за $O(n/k)$ по массиву B.

Суммируем от границы блока до r за $O(k)$ по массиву A.



БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Выполнение операций

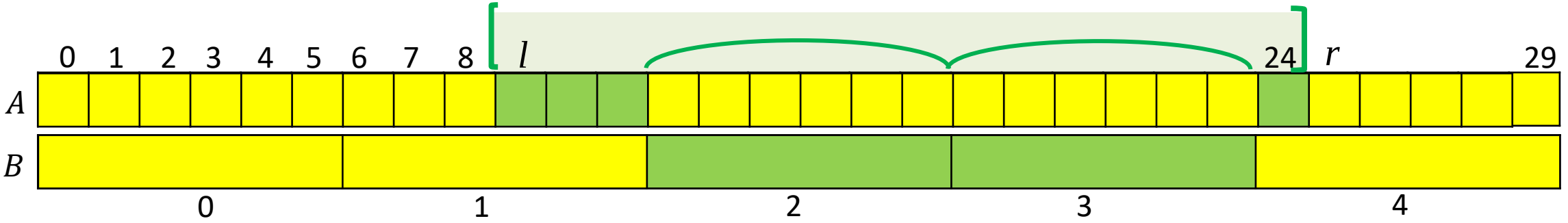


$n = 30$
$k = 6$

```
def FindSum(self, l, r):
    jl = l // self.k
    jr = r // self.k
    if jl == jr:  # same block
        return sum(self.a[l:r])
    else:
        return (
            sum(self.a[l:((jl + 1) * self.k)]) +
            sum(self.b[(jl + 1):jr]) +
            sum(self.a[(jr * self.k):r]
        )
```

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Выполнение операций



При **суммировании** на $[l, r)$общее время:

$$O(k) + O\left(\frac{n}{k}\right) + O(k) = O\left(k + \frac{n}{k}\right).$$

Каким лучше выбрать размер блока k?

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

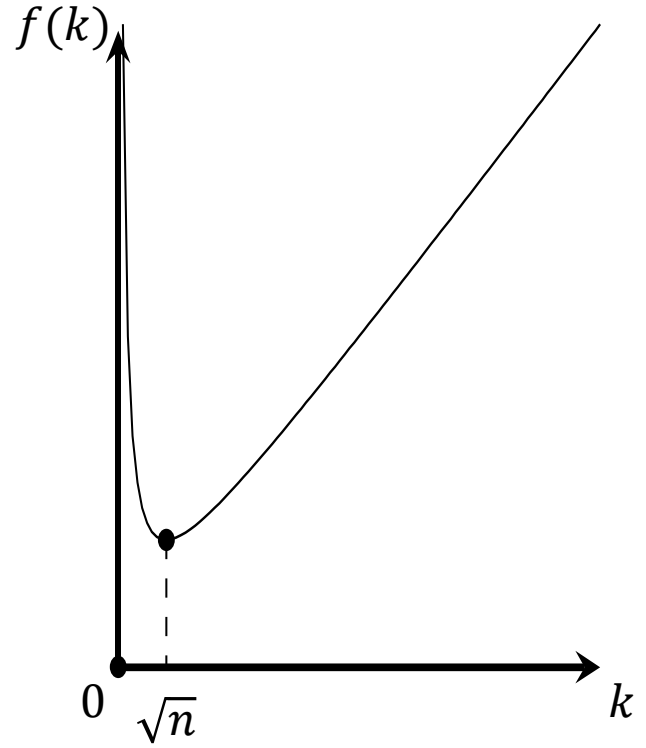
Оптимальный размер блока

Исследуем на экстремум функцию:

$$f(k) = k + \frac{n}{k}$$

$$f'(k) = 1 - \frac{n}{k^2}$$

$$f'(k) = 0 \Leftrightarrow k = \sqrt{\boldsymbol{n}}$$



Таким образом, оптимально разбивать на $\approx \sqrt{\boldsymbol{n}}$ блоков по $\approx \sqrt{\boldsymbol{n}}$ элементов.

Операция $\mathbf{FindSum}(\boldsymbol{l}, \boldsymbol{r})$ выполняется за время $O\left(k + \frac{n}{k}\right) = \mathbf{O}\left(\sqrt{\boldsymbol{n}}\right)$.

Приём называется ***sqrt-декомпозицией*** или ***корневой эвристикой.***

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Пример реализации

```
class Summator:
    def __init__(self, a):
        self.a = a
        self.k = floor(sqrt(len(a)))

        self.b = []
        for i in range(0, len(a), self.k):
            bsum = sum(self.a[i:(i + self.k)])
            self.b.append(bsum)

    def Add(self, i, x):
        self.a[i] += x
        self.b[i // self.k] += x
```

```
    def FindSum(self, l, r):
        jl = l // self.k
        jr = r // self.k
        if jl == jr:  # same block
            return sum(self.a[l:r])
        else:
            return (
                sum(self.a[l:((jl + 1) * self.k)]) +
                sum(self.b[(jl + 1):jr]) +
                sum(self.a[(jr * self.k):r]
            )
```

Оценки

	Обычный массив	Префиксные суммы	Sqrt-декомпозиция
Время на предподсчёт		$\Theta(n)$	$\Theta(n)$
Время на запрос модификации	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
Время на запрос суммы	$\Theta(n)$	$\Theta(1)$	$\Theta(\sqrt{n})$
Объем дополнительной памяти		$\Theta(1)$	$\Theta(\sqrt{n})$

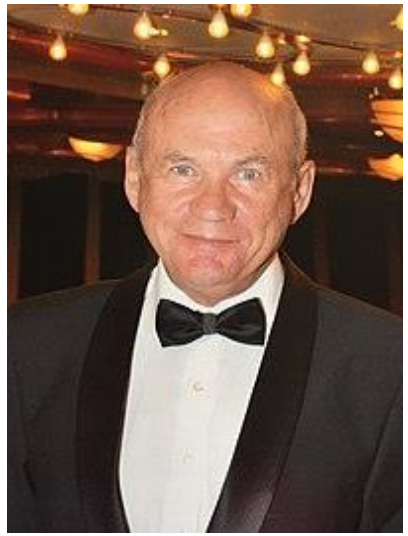
Дополнительный материал

Дерево Фенвика

англ. Fenwick tree, binary indexed tree

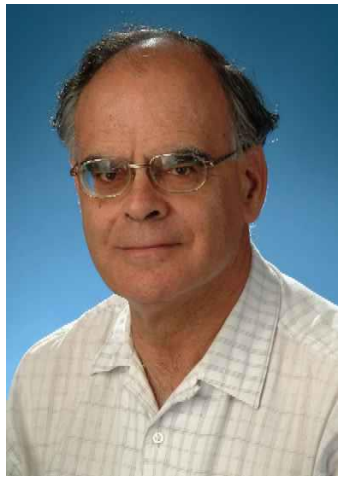
(двоично-индексируемое дерево)

1989 год



Рябко Борис Яковлевич,
Новосибирск

1994 год



Питер **Фенвик,**
Новая Зеландия

Дерево Фенвика

Применяется для выполнения интервальных запросов и запросов модификации для операций, которые являются:

- ассоциативными,
- коммутативными,
- обратимыми.

Например, операция суммы обладает всеми этими свойствами, а операция минимума – не обладает свойством обратимости (например, зная минимум двух чисел и одно из них, нельзя однозначно восстановить второе число).

Дерево Фенвика. Оценки

Для задачи RSQ предподсчёт для построения дерева Фенвика выполняется за время: $\mathrm{O}(n \cdot \log n)$.

Дополнительная память: $\mathrm{O}(n)$.

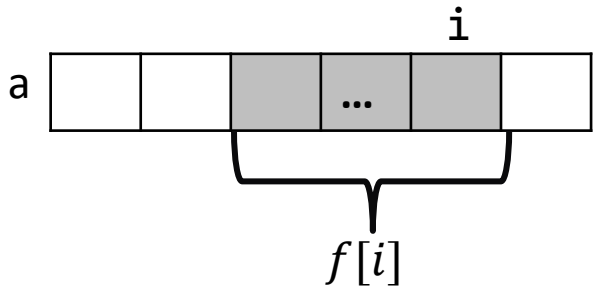
Операция суммы на отрезке и модификации выполняются за время: $\mathrm{O}(\log n)$.

Можно построить дерево Фенвика для задачи RMQ, но минимум можно найти только на префиксе.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево Фенвика. Идея

В дереве Фенвика элемент $t[i]$ отвечает за сумму элементов массива на отрезке длины $f[i]$, который заканчивается в элементе i.



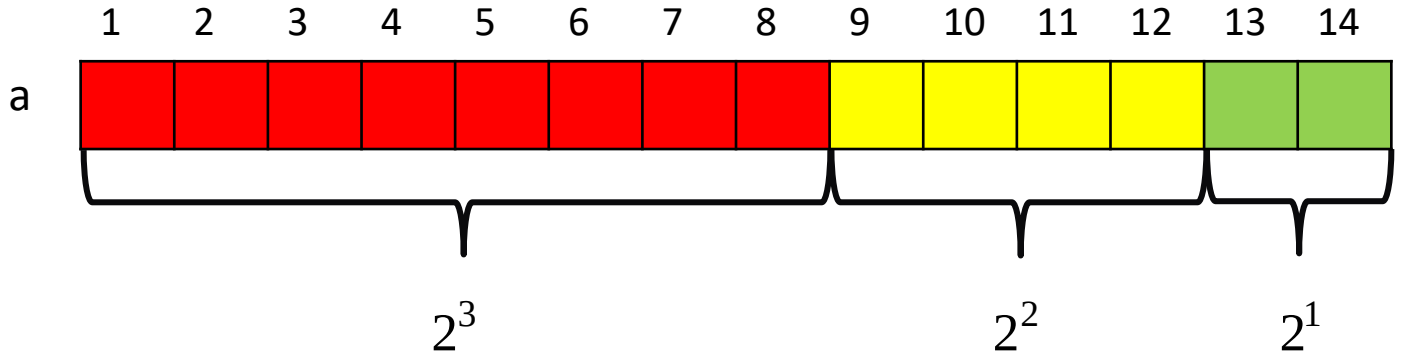
$$t[i] = \sum_{i-f[i]+1}^{i} a[i], \text{ где, например, } f[i] = i \wedge (-i)$$

$$i = 01110_2$$

$$(-i) = 10001_2 + 1_2 = 10010_2$$

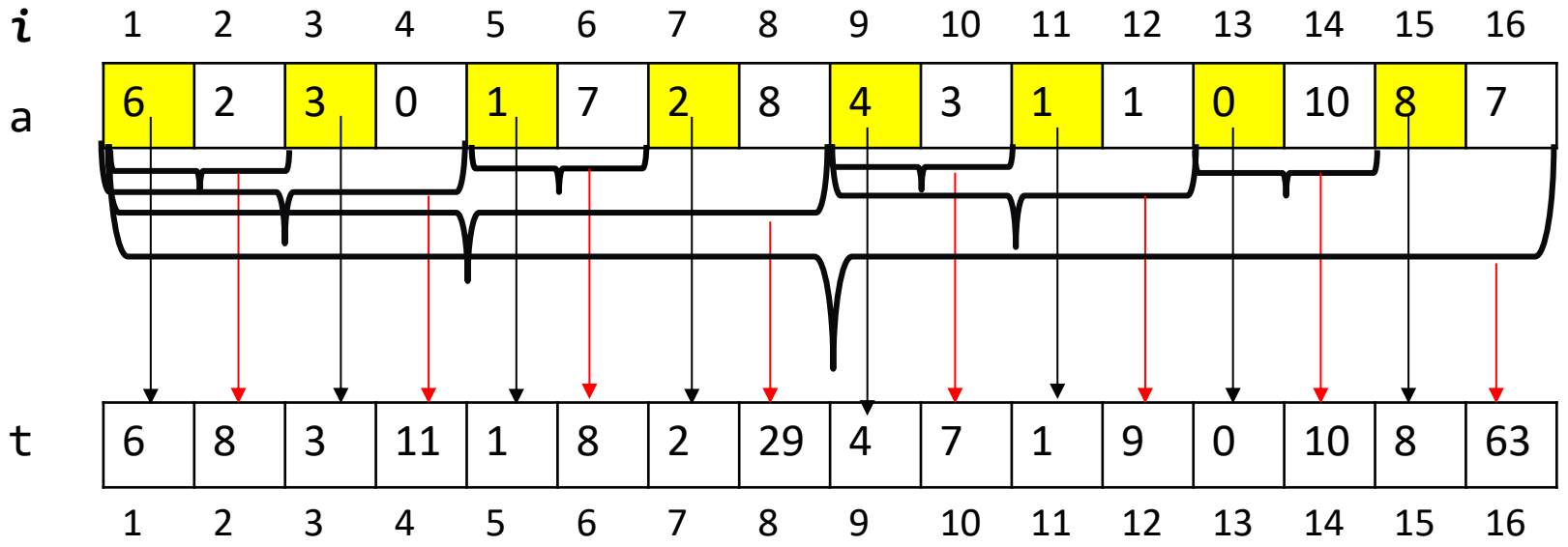
$$i \wedge (-i) = 01110_2 \wedge 10010_2 = 00010_2$$

$$n = 14 = 2^3 + 2^2 + 2^1 = 01110_2$$



БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево Фенвика. Построение



$1 = 2^0 = 00001_2$

$2 = 2^1 = 00010_2$

$3 = 2^1 + 2^0 = 00011_2$

$4 = 2^2 = 00100_2$

$5 = 2^2 + 2^0 = 00101_2$

$6 = 2^2 + 2^1 = 00110_2$

$7 = 2^2 + 2^1 + 2^0 = 00111_2$

$8 = 2^3 = 01000_2$

$9 = 2^3 + 2^0 = 01001_2$

$10 = 2^3 + 2^1 = 01010_2$

$11 = 2^3 + 2^1 + 2^0 = 01011_2$

$12 = 2^3 + 2^2 = 01100_2$

$13 = 2^3 + 2^2 + 2^0 = 01101_2$

$14 = 2^3 + 2^2 + 2^1 = 01110_2$

$15 = 2^3 2^3 + 2^2 + 2^1 + 2^0 = 01111_2$

$16 = 2^4 = 10000_2$

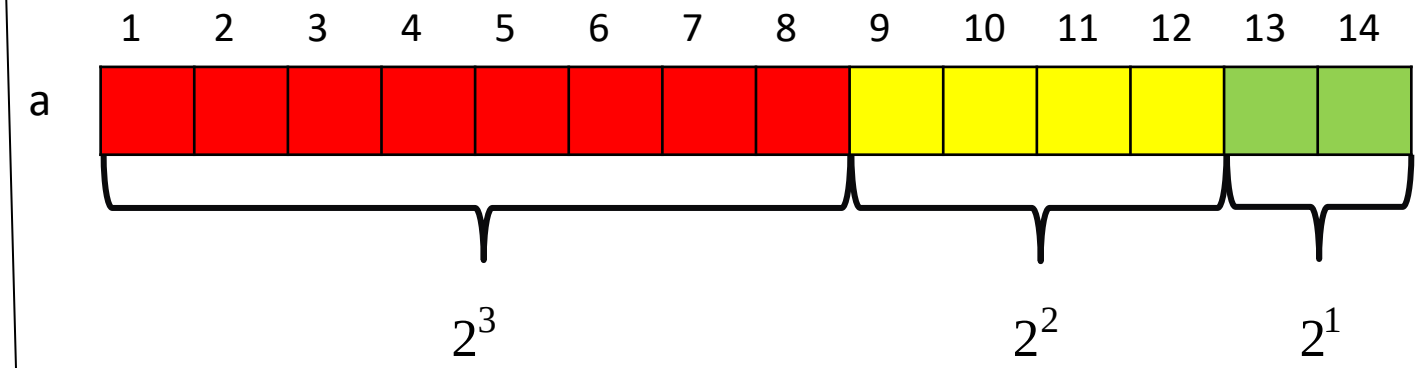
Дерево Фенвика. Код: сумма на префиксе

```
long long GetSum(int pos) {
    long long ot = 0;
    for(int i = pos; i > 0; i -= i &(-i)) {
        ot += t[i];
    }
    return ot;
}
```

Сумма на префиксе вычисляется за время $O(\log n)$

(за один шаг «стирается» крайняя правая единица в двоичном разложении числа pos).

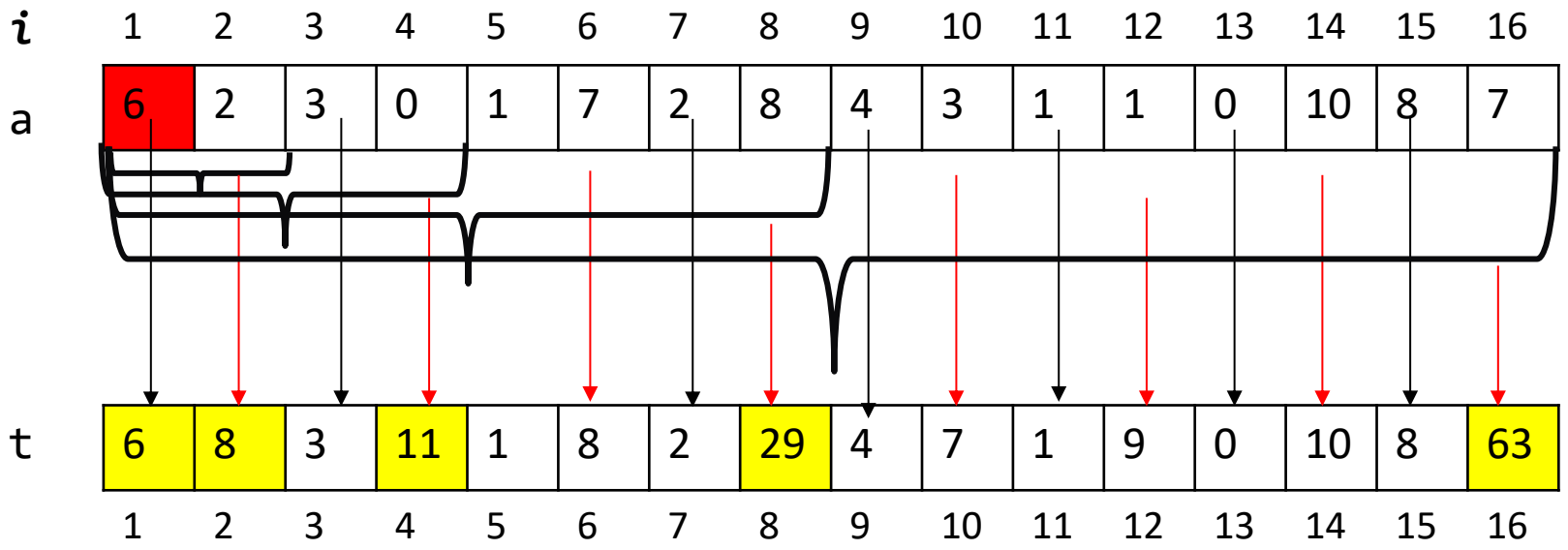
Сумма на префиксе длины $pos = 14$:



$$ot = t[14] + t[12] + t[8]$$

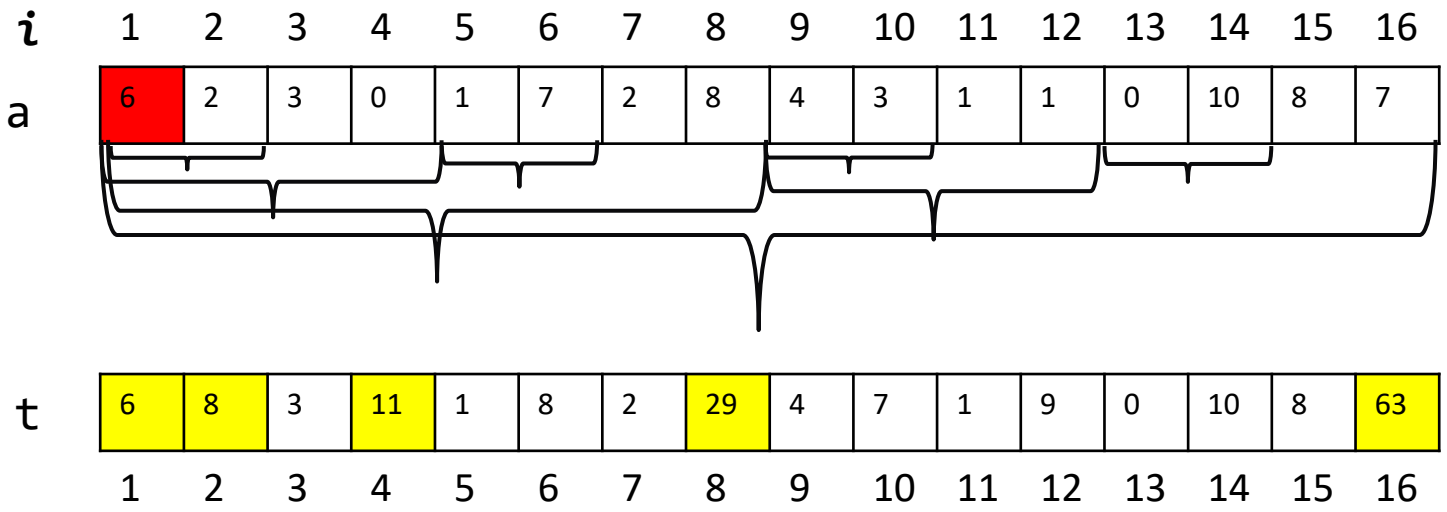
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

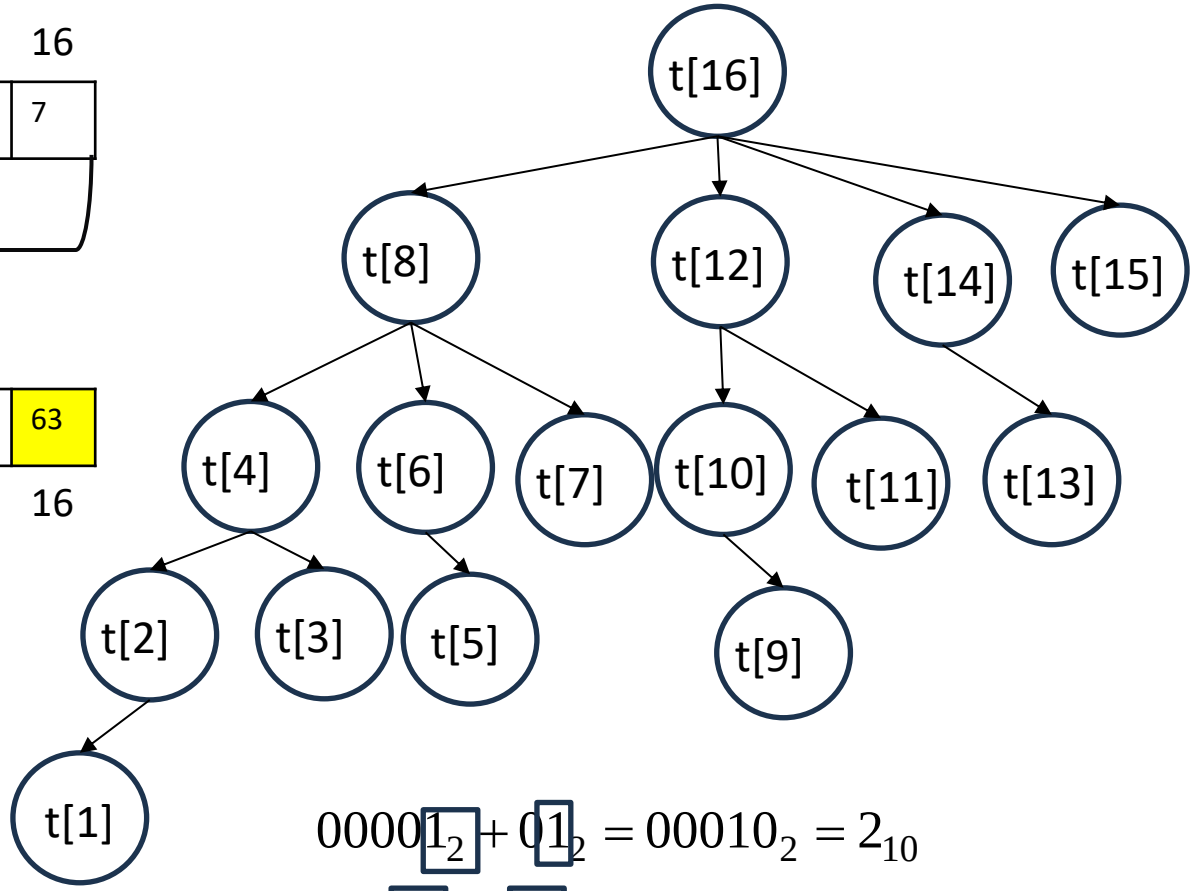
Дерево Фенвика. Модификация



БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево Фенвика. Модификация





Представим дерево Фенвика следующим образом. Вершинам дерева соответствуют отрезки. Сыновья вершины – максимальные по включению отрезки, входящие в этот отрезок. Высота дерева Фенвика не превосходит $O(\log n)$.

Другими словами, вершины дерева пронумерованы числами так, что номера (индексы) вершины-родителя вычисляются с помощью битовых операций над номером вершины-сына .

$$00001_2 + 01_2 = 00010_2 = 2_{10}$$
$$00010_2 + 10_2 = 00100_2 = 4_{10}$$
$$00100_2 + 100_2 = 01000_2 = 8_{10}$$
$$01000_2 + 1000_2 = 10000_2 = 16_{10}$$

Дерево Фенвика. Код: модификация

```
void Update(int pos, int x) {
    for(int i = pos; i < t.size(); i += i & (-i)) {
        t[i] += x;
    }
}
```

Модификация (подъем по дереву Фенвика от вершины к корню с модификацией всех вершин на пути) выполняется за время $O(\log n)$.

Используя функцию модификации, можно построить и дерево Фенвика за время $O(n \cdot \log n)$.

Оценки RSQ

	Обычный массив	Префиксные суммы	Sqrt-декомпозиция	Дерево Фенвика
Время на предподсчёт		$\Theta(n)$	$\Theta(n)$	$\Theta(n \cdot \log n)$
Время на запрос модификации	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$
Время на запрос суммы	$\Theta(n)$	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(\log n)$
Объем дополнительной памяти		$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков

Дальнейшим развитием идеи разбиения на блоки будет следующее:

- блоки разной длины организуем в виде дерева;
- в каждой вершине дерева содержится сумма элементов массива, индексы которых принадлежат соответствующему отрезку;
- корень соответствует всему массиву $[0, n)$ (т.е. в корне дерева хранится общая сумма всех элементов массива);
- у вершины, соответствующей $[t_l, t_r)$, два сына:

  $[t_l, m)$ и $[m, t_r)$, где $m = \left\lfloor \frac{t_l + t_r}{2} \right\rfloor$

Такую структуру будем называть ***деревом отрезков***

Терминология не устоялась, под *segment tree* и *interval tree* часто понимают другие структуры

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

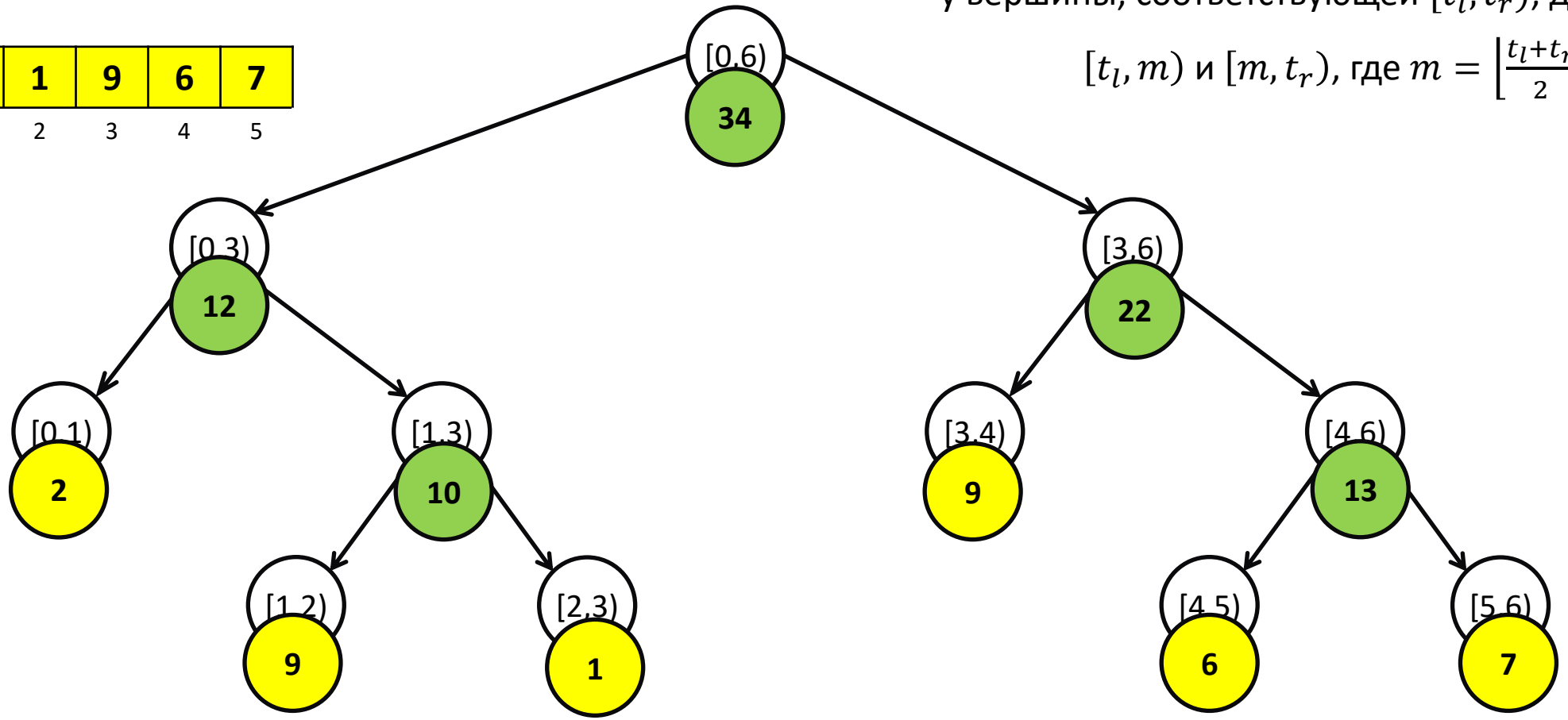
Пример дерева отрезков для задачи RSQ

$n = 6$

A	2	9	1	9	6	7
	0	1	2	3	4	5

у вершины, соответствующей $[t_l, t_r)$, два сына:

$[t_l, m)$ и $[m, t_r)$, где $m = \left\lfloor \frac{t_l + t_r}{2} \right\rfloor$



Дерево отрезков. Число вершин

Теорема

> Общее число вершин равно $2 \cdot n - 1$.

Доказательство.

По индукции.

Если $n = 1$, то есть одна вершина, $2 \cdot n - 1 = 1$ — верно.

Если $n = n_1 + n_2$, то строим два дерева и добавляем ещё одну вершину:

$$(2 \cdot n_1 - 1) + (2 \cdot n_2 - 1) + 1 = 2 \cdot (n_1 + n_2) - 2 + 1 = 2 \cdot n - 1$$

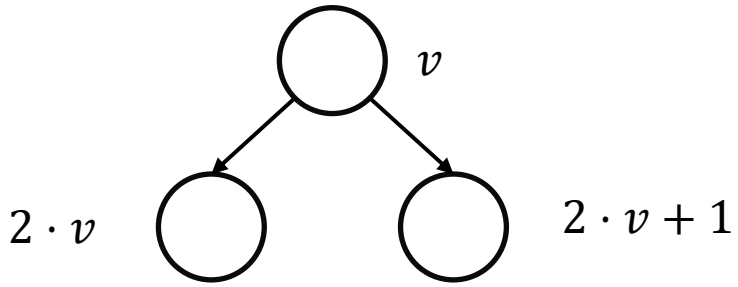
Таким образом, в дереве n листьев и $n - 1$ внутренняя вершина.

Высота дерева $O(\log n)$.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Хранение

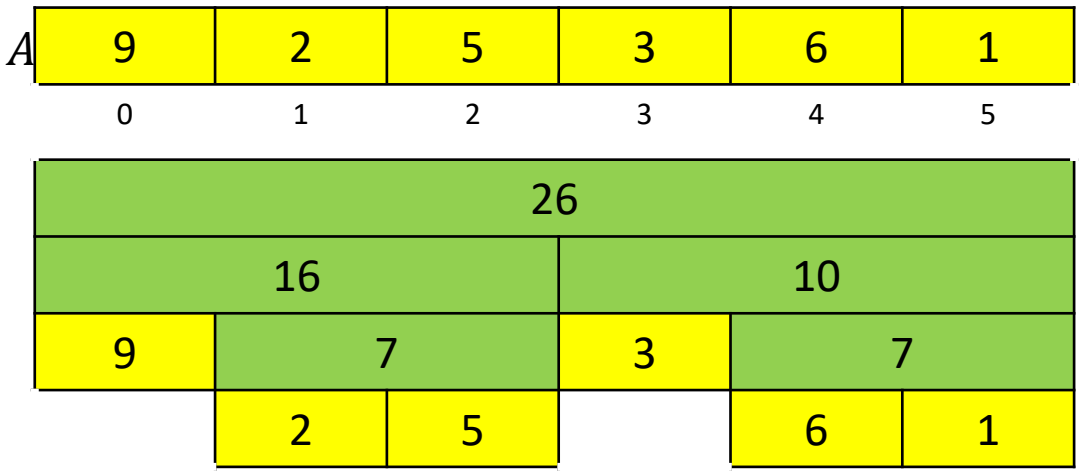
- Нет необходимости хранить указатели/ссылки
- Для хранения вершин дерева будем использовать массив
  (по аналогии с тем, как была реализована на массиве бинарная куча)
- Индексация в массиве с единицы (1 — корень)

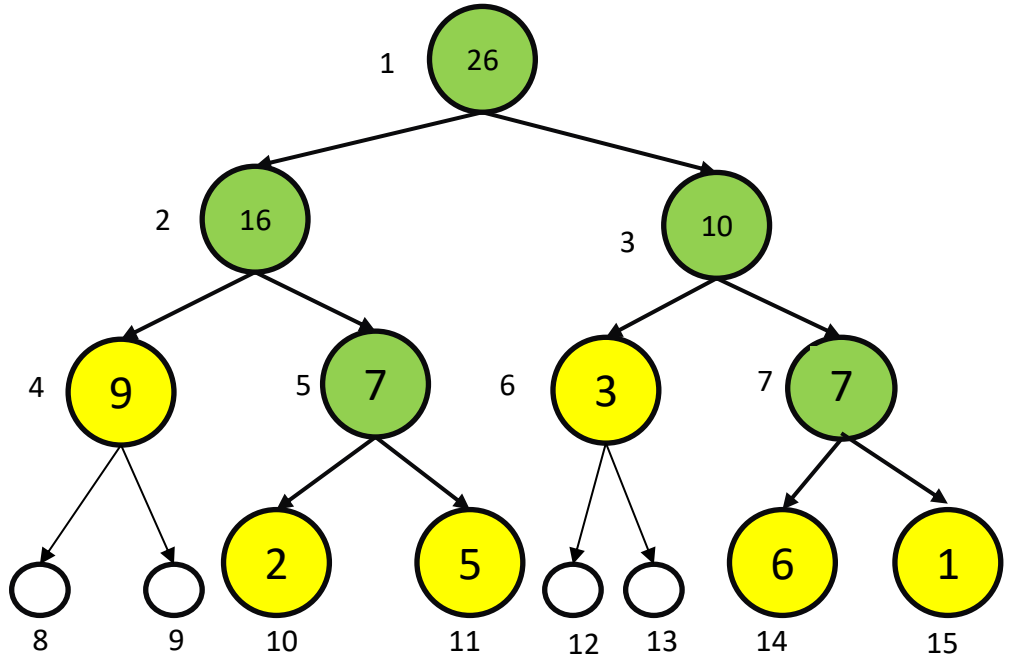


БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Хранение

$n = 6$





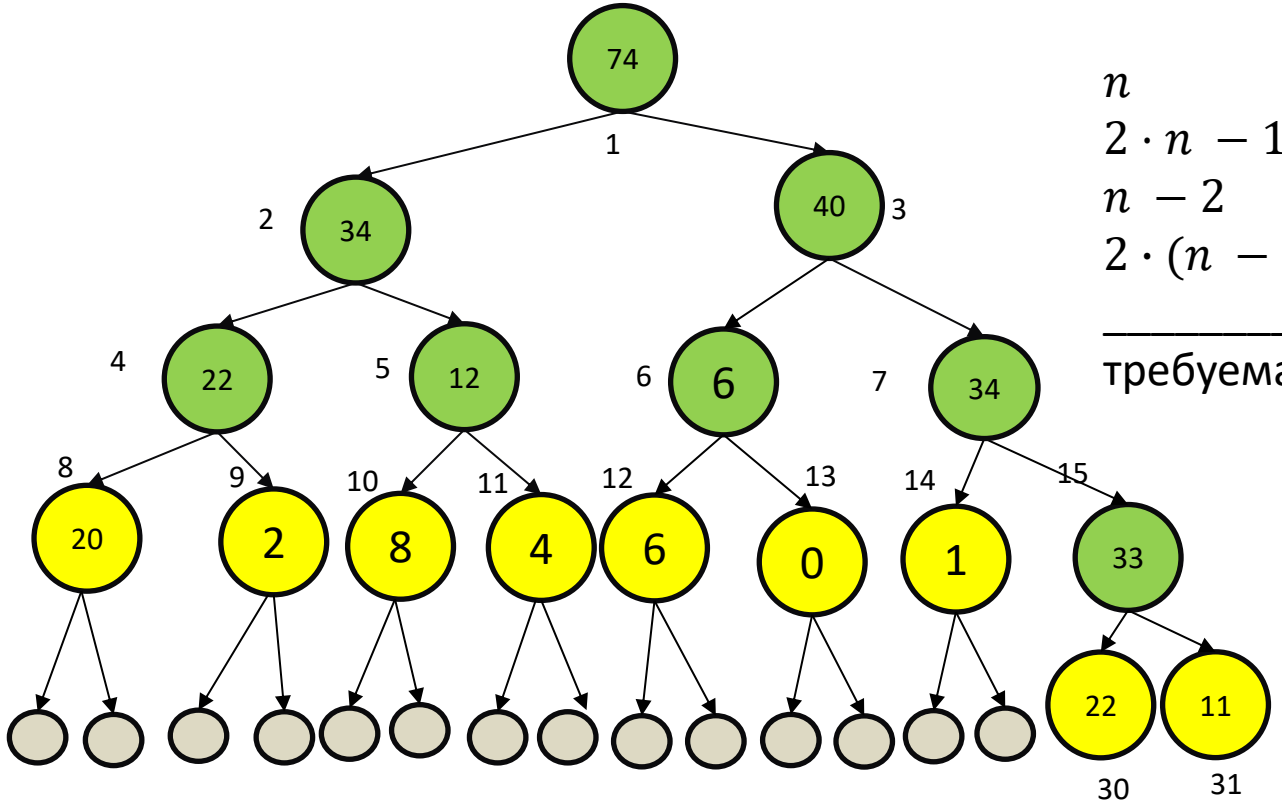
T	26	16	10	9	7	3	7	—	—	2	5	—	—	6	1
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Количество реально занятых ячеек: $2 \cdot n - 1 = 2 \cdot 6 - 1 = 11$.

Сколько всего ячеек надо зарезервировать в массиве T?

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Хранение



n число листьев в дереве (число элементов в массиве)

$2 \cdot n - 1$ число реально занятых ячеек массива T (по теореме)

$n - 2$ число листьев на предпоследнем слое

$2 \cdot (n - 2)$ число свободных элементов в массиве T

требуемая память для массива T :

$$\underbrace{(2 \cdot n - 1)}_{\text{используемая память}} + \underbrace{2 \cdot (n - 2)}_{\text{неиспользуемая память}} = 4 \cdot n - 5$$

T	74	34	40	22	12	6	34	20	2	8	4	6	0	1	33	-	-	-	-	-	-	22	11
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		...			29	30	31

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Хранение

В том случае, когда n не является степенью 2, дерево не является полным, из-за чего в массиве могут быть неиспользуемые ячейки.

Чтобы места гарантированно хватило, можно выставить размер массива $4 \cdot n$.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

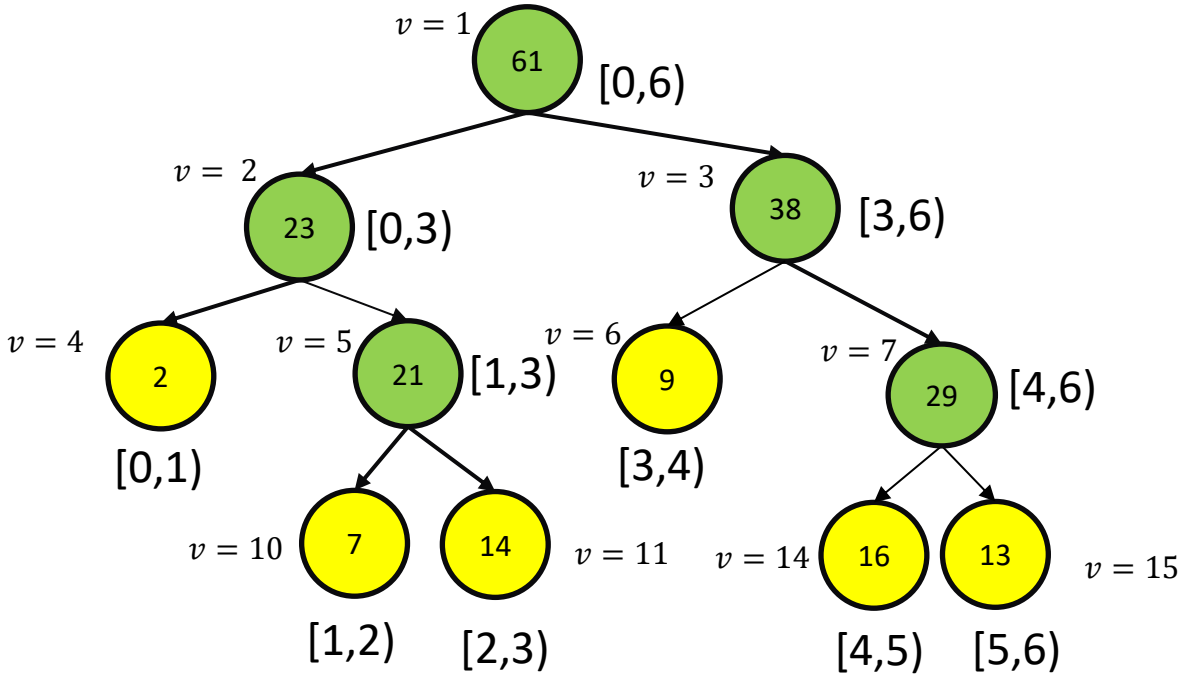
Дерево отрезков. Построение

```
def DoBuild(a, t, v, tl, tr):
    if tr - tl == 1:
        t[v] = a[tl]
    else:
        m = (tl + tr) // 2
        DoBuild(a, t, v=2*v, tl=tl, tr=m)
        DoBuild(a, t, v=2*v+1, tl=m, tr=tr)
        t[v] = t[2*v] + t[2*v+1]
```

```
def Build(a, n):
    t = [0] * 4*n
    DoBuild(a, t, v=1, tl=0, tr=n)
    return t
```

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Построение




```
def Build(a, n):
    t = [0] * 4*n
    DoBuild(a, t, v=1, tl=0, tr=n)
    return t
```

```
def DoBuild(a, t, v, tl, tr):
    if tr - tl == 1:
        t[v] = a[tl]
    else:
        m = (tl + tr) // 2
        DoBuild(a, t, v=2*v, tl=tl, tr=m)
        DoBuild(a, t, v=2*v+1, tl=m, tr=tr)
        t[v] = t[2*v] + t[2*v+1]
```

A

2	7	14	9	16	13
0	1	2	3	4	5

$n = 6$

T

61	23	38	2	21	9	29			7	14			16	13					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Дерево отрезков. Модификация

Add(i, x)

```
def DoAdd(t, v, tl, tr, i, x):
    if tr - tl == 1:
        t[v] += x
        return
    m = (tl + tr) // 2
    if i < m:
        DoAdd(t, v=2*v, tl=tl, tr=m, i=i, x=x)
    else:
        DoAdd(t, v=2*v+1, tl=m, tr=tr, i=i, x=x)
    t[v] = t[2*v] + t[2*v+1]
```

```
def Add(t, n, i, x):
    DoAdd(t, v=1, tl=0, tr=n, i=i, x=x)
```

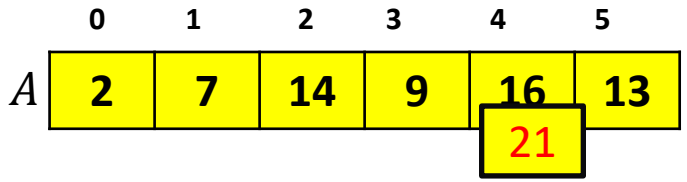
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

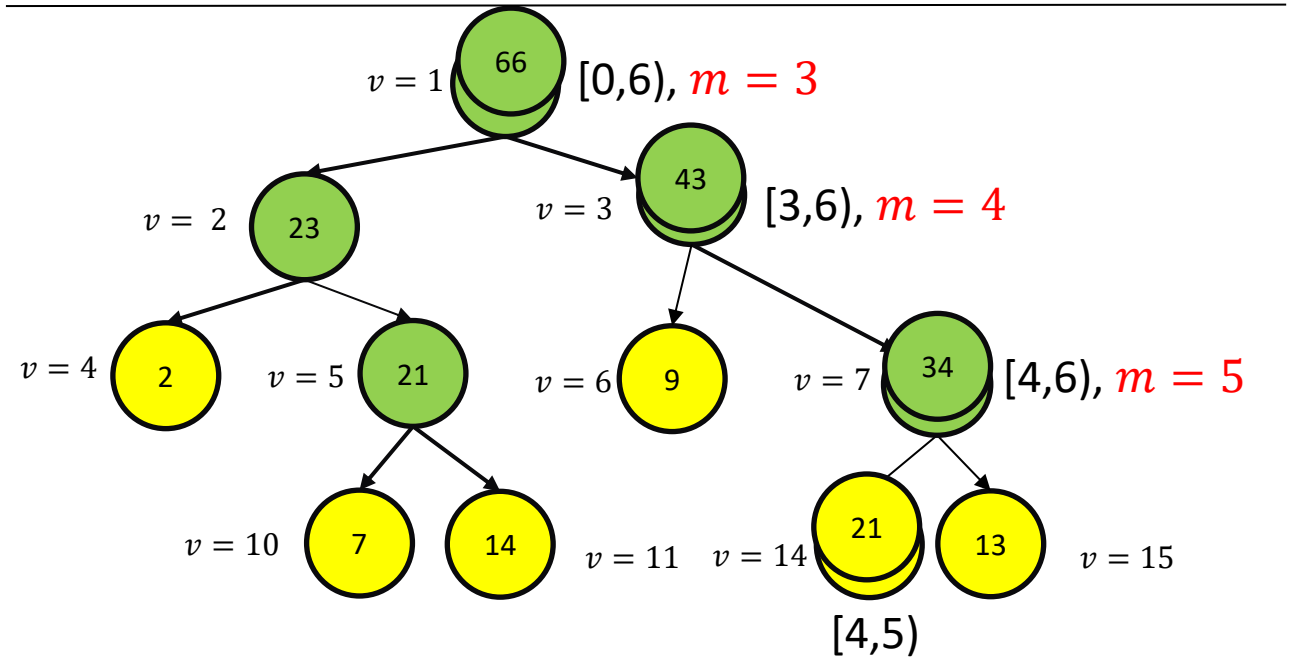
[4,5)

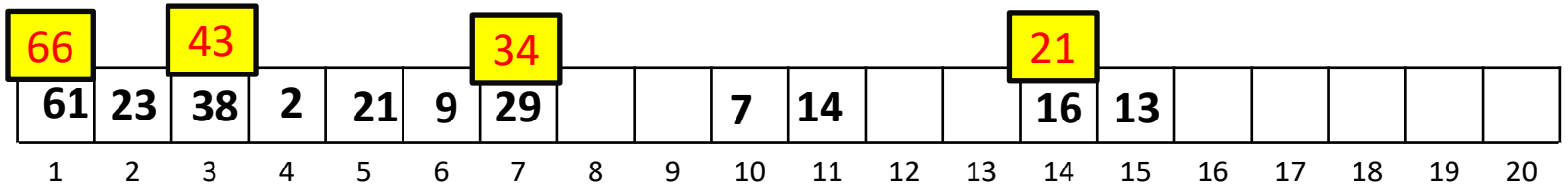
Дерево отрезков. Модификация

Add(4, 5)

$i = 4, x = 5$





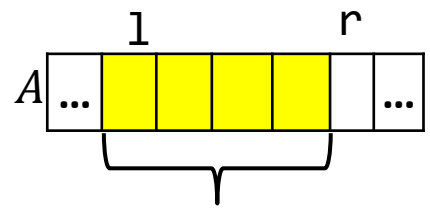


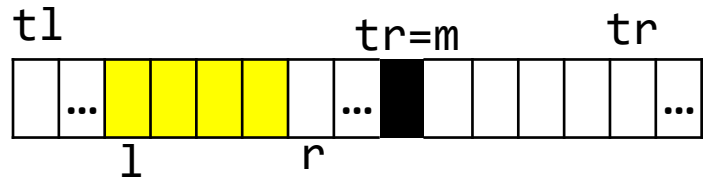

```
def Add(t, n, i, x):
    DoAdd(t, v=1, tl=0, tr=n, i=i, x=x)

def DoAdd(t, v, tl, tr, i, x):
    if tr - tl == 1:
        t[v] += x
        return
    m = (tl + tr) // 2
    if i < m:
        DoAdd(t, v=2*v, tl=tl, tr=m, i=i, x=x)
    else:
        DoAdd(t, v=2*v+1, tl=m, tr=tr, i=i, x=x)
    t[v] = t[2*v] + t[2*v+1]
```

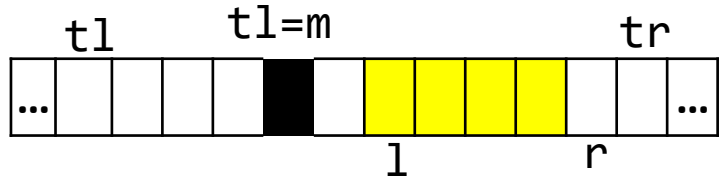
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

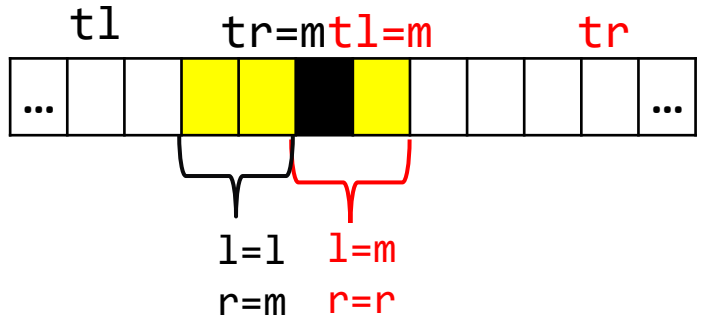
Дерево отрезков. Сумма

```
def DoFindSum(t, v, tl, tr, l, r):
    if l == tl and r == tr:
        return t[v]
    m = (tl + tr) // 2
    if r <= m:
        return DoFindSum(t, v=2*v, tl=tl, tr=m, l=l, r=r)
    if m <= l:
        return DoFindSum(t, v=2*v+1, tl=m, tr=tr, l=l, r=r)
    return (
        DoFindSum(t, v=2*v, tl=tl, tr=m, l=l, r=m) +
        DoFindSum(t, v=2*v+1, tl=m, tr=tr, l=m, r=r)
    )
```









БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дерево отрезков. Сумма. Пример

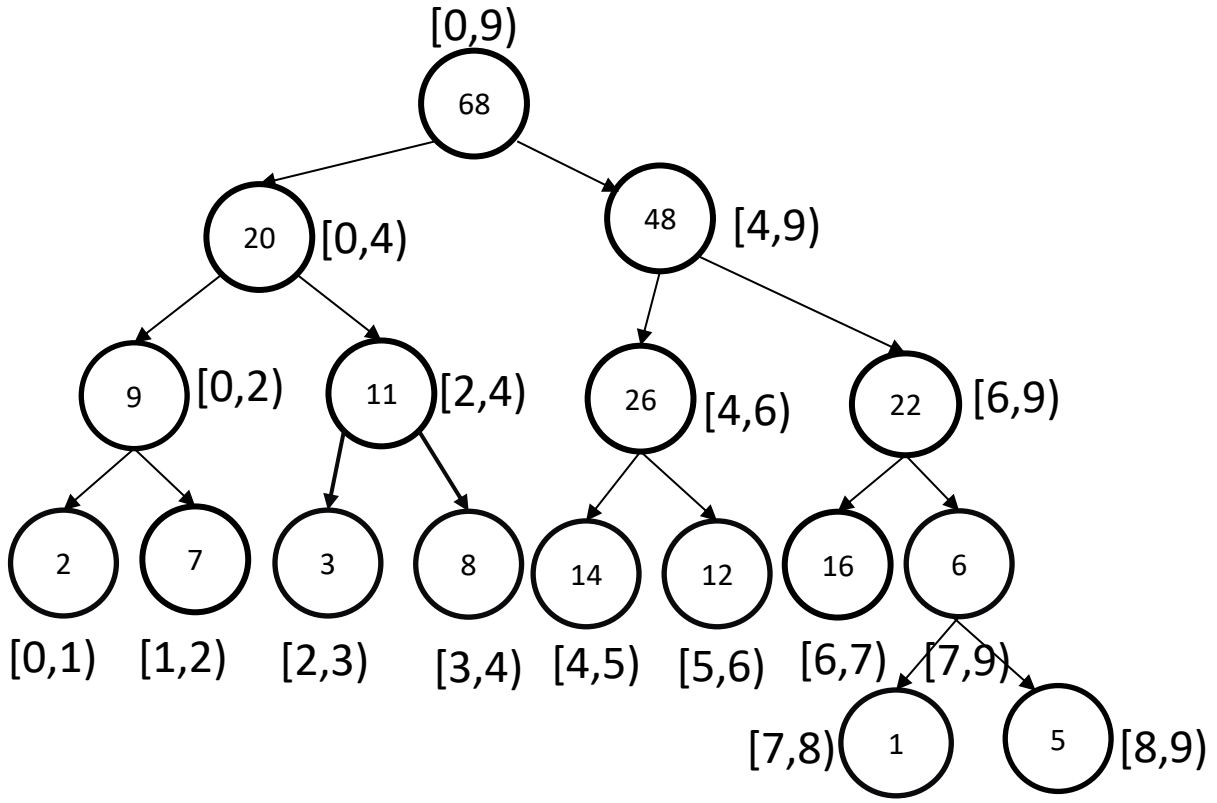
Найти сумму на отрезке [1,7).

	0	1	2	3	4	5	6	7	8
A	2	7	3	8	14	12	16	1	5

$n = 9$

2	7	3	8	14	12	16	1	5
2	7	3	8	14	12	16	1	5
2	7	3	8	14	12	16	1	5
	7					16		

Сумма на отрезке [1,7): 7+ 11+26+16 =60

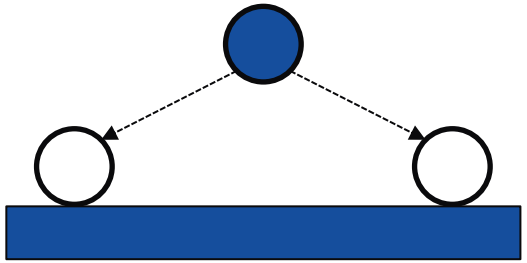


T	68 [0,9)	20 [0,4)	48 [4,9)	9 [0,2)	11 [2,4)	26 [4,6)	22 [6,9)	2 [0,1)	7 [1,2)	3 [2,3)	8 [3,4)	14 [4,5)	12 [5,6)	16 [6,7)	6 [7,9)	-	-	…	-	-	1 [7,8)	5 [8,9)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	28	29	30	31

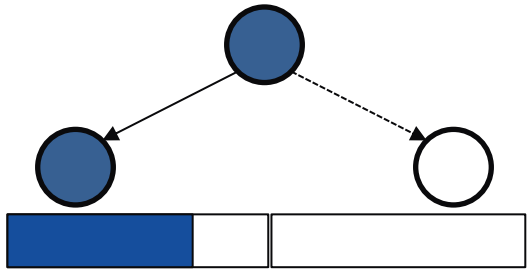
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Оценка времени работы

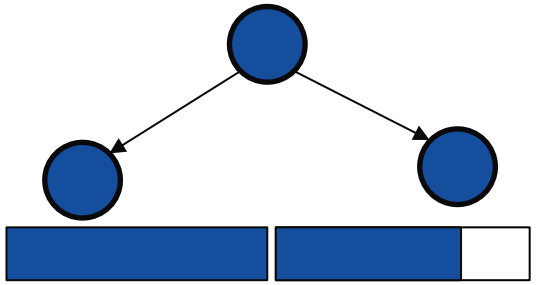
- При вычислении суммы рекурсия иногда уходит сразу в обе ветви
- Нужно доказать, что время работы — $O(\log n)$, не $O(n)$
- Пусть интервал имеет вид $[0, r)$
- Возможна одна из трёх ситуаций:



остановка



спуск только влево

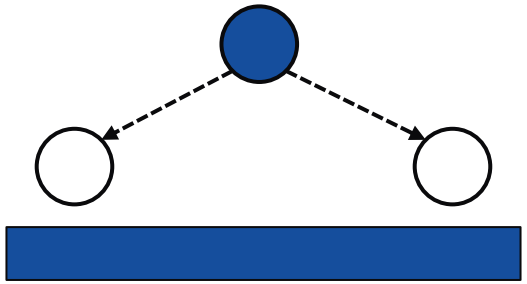


спуск влево и вправо,
но на следующем шаге левая
рекурсия сразу завершится

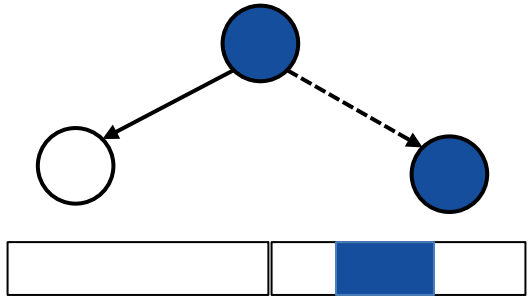
- Видим, что на каждом уровне активно работает только одна ветвь рекурсии

Оценка времени работы

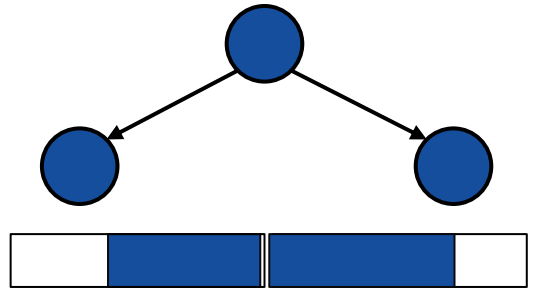
- Пусть теперь интервал $[l, r)$ произвольный
- Возможна одна из трёх ситуаций



остановка



спуск только вправо
(или только влево)



спуск в обе стороны

- В третьем случае задача сводится к ранее рассмотренной (после разделения на каждом уровне активно работают две рекурсии)

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Доработки

- Нерекурсивная реализация быстрее работает на практике

- Можно добавить поддержку операций на интервале:
  - ✓ $\mathrm{Add}(l, r, x)$ — прибавить x к каждому элементу $[l, r)$
  - ✓ $\mathrm{Set}(l, r, x)$ — установить элементы $[l, r)$ в значение x

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Оценка времени работы

	дерево отрезков
Время на предподсчёт	$\Theta(n)$
Время на запрос модификации	$\Theta(\log n)$
Время на запрос суммы	$\Theta(\log n)$
Объем дополнительной памяти	$\Theta(n)$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Оценки RSQ

	Обычный массив	Префиксные суммы	Sqrt-декомпозиция	Дерево Фенвика	Дерево отрезков
Время на **предподсчёт**		$\Theta(n)$	$\Theta(n)$	$\Theta(n \cdot \log n)$	$\Theta(n)$
Время на запрос **модификации**	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$
Время на запрос **суммы**	$\Theta(n)$	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(\log n)$	$\Theta(\log n)$
Объём **дополнительной памяти**		$\Theta(1)$	$\sqrt{n}$	n	$4 \cdot n$

Статическая задача RMQ

Статическая online задача

RMQ — **R**ange **M**inimum **Q**uery

(запрос минимума на отрезке)

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Статическая задача RMQ

Задана последовательность из n чисел:

$$A = [a_0, a_1, a_2, \dots, a_{n-1}].$$

Поступают запросы *минимума* $\mathbf{FindMin}(l, r)$ — найти минимум на полуинтервале $[\boldsymbol{l}, \boldsymbol{r})$.

Запросов модификации **нет.**

Статическая задача RMQ

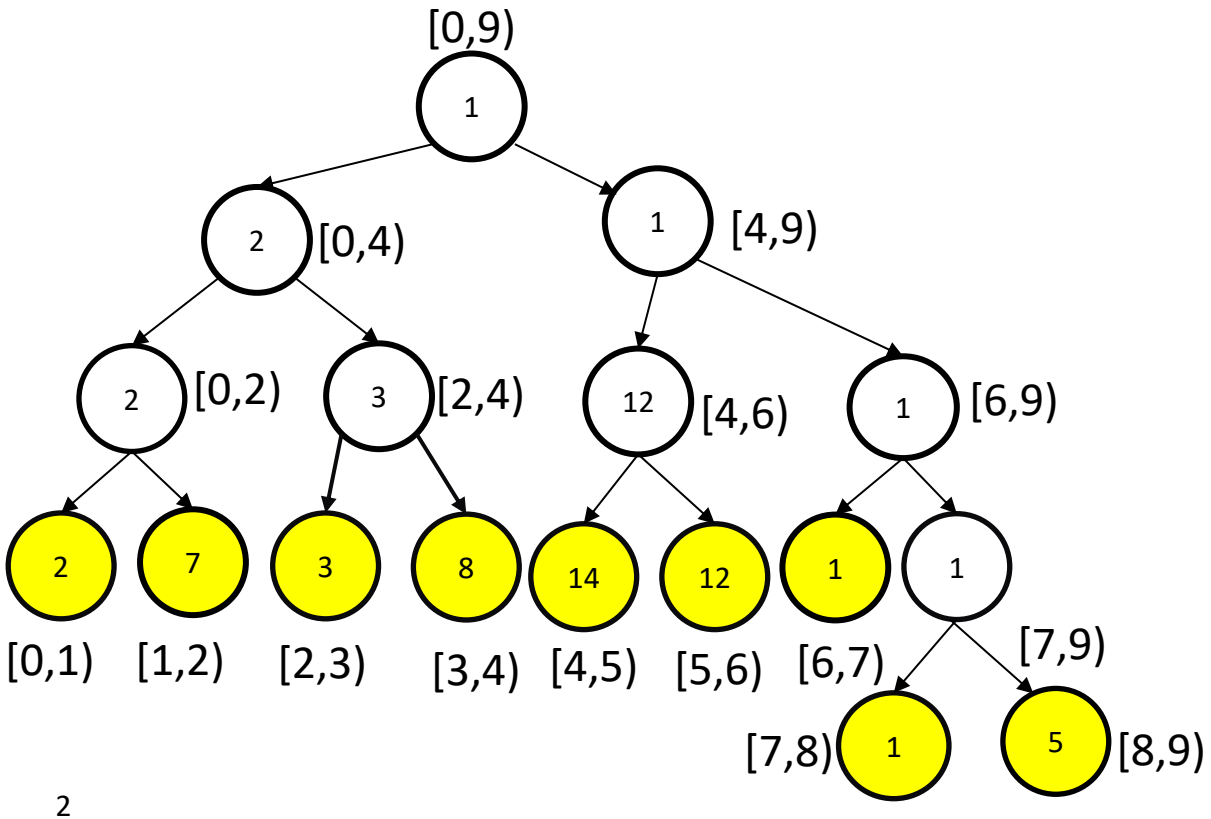
Можно решать корневой декомпозицией или деревом отрезков.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

	0	1	2	3	4	5	6	7	8
A	2	7	3	8	14	12	1	1	5

$n = 9$

Дерево отрезков для RMQ



	1	2																	31
T	1	2	1	2	3	12	1	2	7	3	8	14	12	1	1	-	…	-	1	5

$\mathbf{FindMin}(l, r) - \mathbf{O}(\log n)$

Корневая декомпозиция для RMQ

	0	1	2	3	4	5	6	7	8
A	2	7	3	8	14	12	1	1	5

	0	1	2
B	2	8	1

$\mathbf{FindMin}(l, r) - \mathbf{O}(\sqrt{n})$

Однако научимся отвечать на запрос $\mathbf{FindMin}(l, r)$ за время $\Theta(1)$.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Статическая задача RMQ

Сначала рассмотрим решение «в лоб»:

для всех пар (i, j) просчитаем минимум массива A на полуинтервале $[i, j)$ и занесём в таблицу T.

A

0	1	2	3	4	5
1	2	0	9	6	7

Так как каждый элемент таблицы по рекуррентному соотношению

$$t_{i,j} = \min\{t_{i,j-1}, a_{j-1}\}$$

может быть вычислен за константу, то время предподсчёта:

$$\Theta(n^2).$$

Затраты на дополнительную память (таблица T):

$$\Theta(n^2).$$

T

	0	1	2	3	4	5	6
0		1	1	0	0	0	0
1			2	0	0	0	0
2				0	0	0	0
3					9	6	6
4						6	6
5							7

$i = 1$

0	1	2	3	4	5
1	2	0	9	6	7

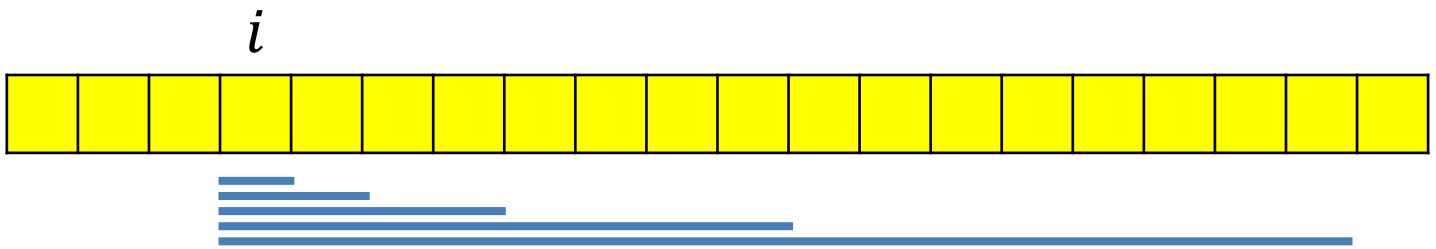
Разрежённая таблица

- Идея — разреди́ть таблицу, убрать часть информации
- Будем хранить минимумы для тех интервалов, длины которых являются *степенями двойки*
- Такая структура называется ***sparse table*** (рус. *разрежённая таблица*)

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Разрежённая таблица

- Иначе говоря, для каждой позиции i храним минимумы на интервалах длины $1, 2, 4, \ldots, 2^k$ вправо от i
- $t_{k,i} = \min\{a_i,\ a_{i+1},\ a_{i+2},\ \ldots, a_{i+2^k-1}\}$
- $\Theta(n \cdot \log n)$ памяти, $\Theta(n \cdot \log n)$ времени на построение



БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Sparse table. Пример

		2	9	1	9	6	7	5	2
k	2^k	0	1	2	3	4	5	6	7

$$\begin{cases} t_{0,i} = a_i \\ t_{k,i} = \min\{t_{k-1,i}, t_{k-1,i+2^{k-1}}\} \end{cases}$$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Разрежённая таблица

Предположим, что $2^{k-1} < n \leq 2^k$.

Разреженная таблица занимает $\Theta(n \cdot \log n)$ ячеек памяти.

0 (=2^0)	2	9	1	9	6	7	5	2
1 (=2^1)	2	1	1	6	6	5	2	
2 (=2^2)	1	1	1	5	2			
3 (=2^3)	1							

Оценим число «лишних» ячеек в таблице.

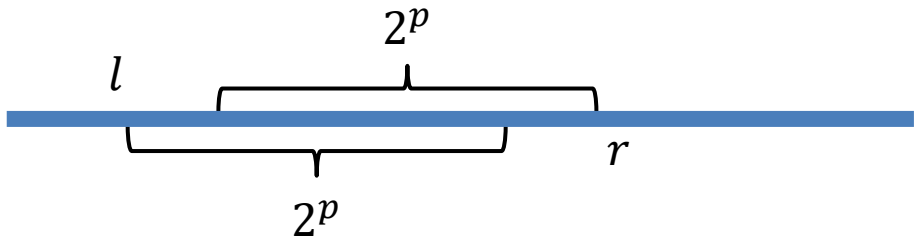
$$(2^1 - 1) + (2^2 - 1) + \cdots + (2^k - 1) = 2 \cdot (2^k - 1) - k < 4 \cdot n$$

$$(2^1 - 1) + (2^2 - 1) + \cdots + (2^k - 1) = 2 \cdot (2^k - 1) - k > 2 \cdot (n - 1) - \lceil \log_2 n \rceil - 1$$

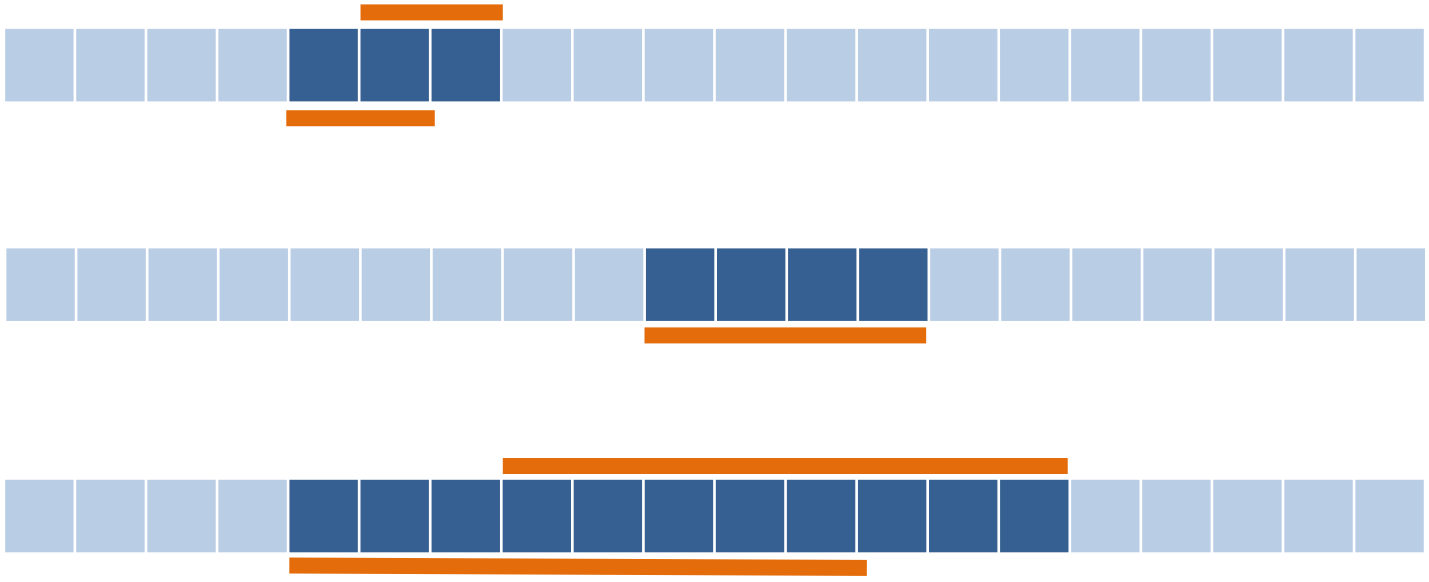
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Ответ на запрос

- Нужно найти минимум на $[l, r)$
- Пусть p — максимальная степень такая, что 2^p не превосходит длины интервала: $2^p \leq r - l < 2^{p+1}$
- Интервал $[l, r)$ покрывается не более чем двумя перекрывающимися интервалами длины 2^p (начало первого интервала фиксируется на левой границе l, а второго – на правой границе r)



$$\max\{p|2^p \leq r - l < 2^{p+1}\}$$



$r - l = 3$

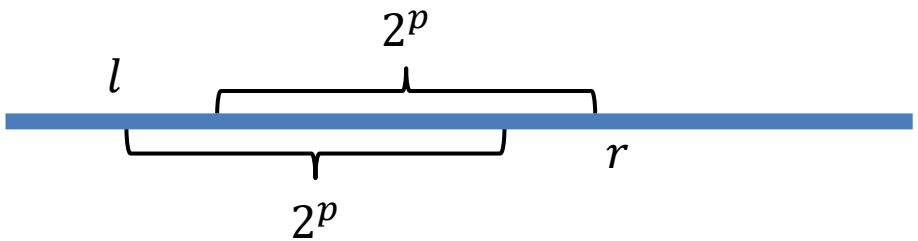
$r - l = 4$

$r - l = 11$

Ответ на запрос

Ответ на запрос минимума на полуинтервале можно вычислить по формуле:

$$\min\{t_{p,l},\ t_{p,r-2^p}\}$$



Формула является корректной, так как бинарная операция минимума обладает свойствами **ассоциативности, коммутативности и идемпотентность**.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Свойства бинарного отношения минимума

1. **Ассоциативность** (выполнять операции можно в произвольном порядке, т.е. допускается любой порядок расстановки скобок):

$$a \circ (b \circ c) = (a \circ b) \circ c$$

2. **Коммутативность** (можно располагать элементы в произвольном порядке, переставляя их):

$$a \circ b = b \circ a$$

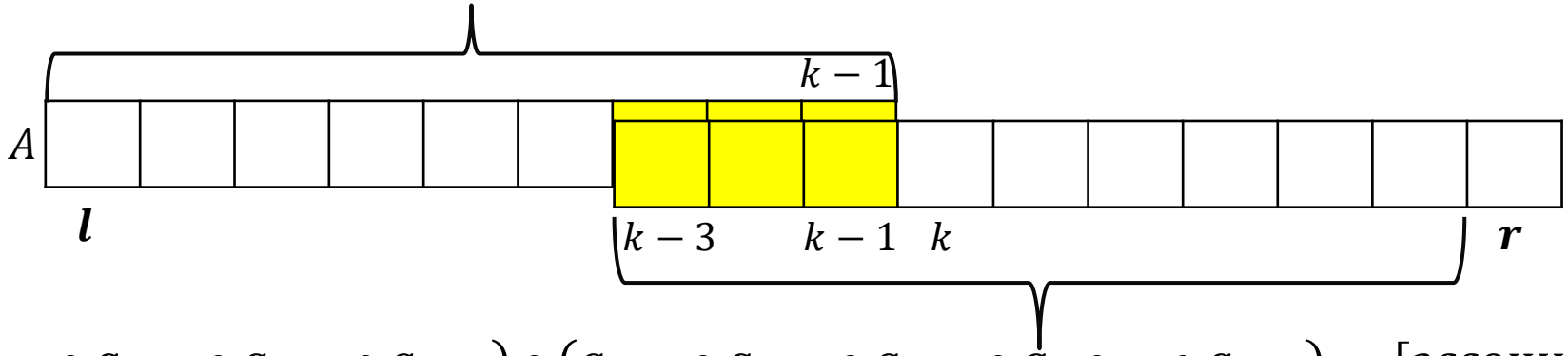
3. **Идемпотентность** (благодаря этому свойству мы сможем работать на пересекающихся отрезках):

$$a \circ a = a$$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Свойства бинарного отношения минимума

Например, покажем, что **минимум** на $[l, r) =$ **минимум** $\{$**минимум** на $[l, k)$, **минимум** на $[\, k-3, r)\,\}$



$$(a_l \circ a_{l+1} \circ \cdots \circ a_{k-3} \circ a_{k-2} \circ a_{k-1}) \circ (a_{k-3} \circ a_{k-2} \circ a_{k-1} \circ a_k \circ \cdots \circ a_{r-1}) = [\text{ассоциативность}]$$

$$= a_l \circ a_{l+1} \circ \cdots \circ a_{k-3} \circ a_{k-2} \circ a_{k-1} \circ a_{k-3} \circ a_{k-2} \circ a_{k-1} \circ a_k \circ \cdots \circ a_{r-1} = [\text{коммутативность}]$$

$$= a_l \circ a_{l+1} \circ \cdots \circ \boxed{a_{k-3} \circ a_{k-3}} \circ \boxed{a_{k-2} \circ a_{k-2}} \circ \boxed{a_{k-1} \circ a_{k-1}} \circ a_k \circ \cdots \circ a_{r-1} = [\text{ассоциативность}]$$

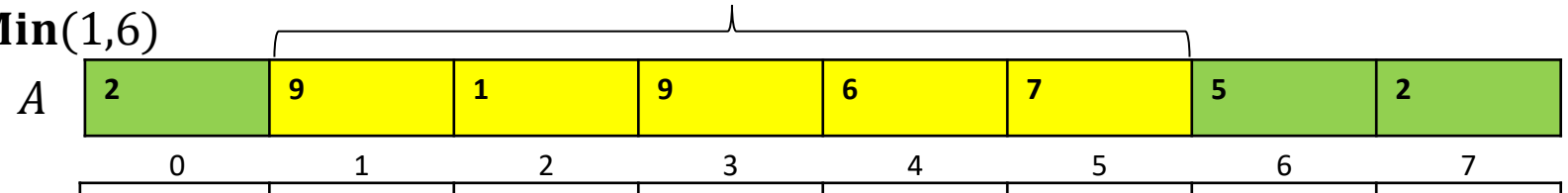
$$= a_l \circ a_{l+1} \circ \cdots \circ (a_{k-3} \circ a_{k-3}) \circ (a_{k-2} \circ a_{k-2}) \circ (a_{k-1} \circ a_{k-1}) \circ a_k \circ \cdots \circ a_{r-1} = [\text{идемпотентность}]$$

$$= a_l \circ a_{l+1} \circ \cdots a_{k-3} \circ a_{k-2} \circ a_{k-1} \circ a_k \circ \cdots \circ a_{r-1}$$

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Пример

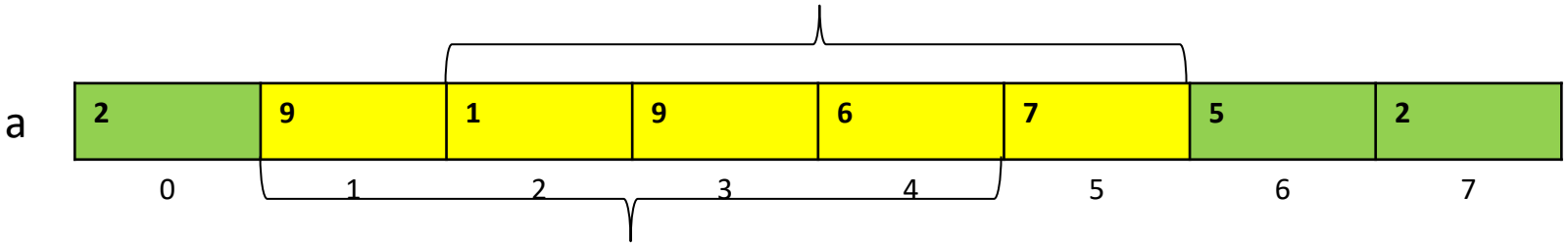
FindMin(1,6)



0 (=2^0)	2	9	1	9	6	7	5	2
1 (=2^1)	2	1	1	6	6	5	2	
2 (=2^2)	1	1	1	5	2			
3 (=2^3)	1							

$$t_{k,i} = \min\{a_i, a_{i+1}, \dots, a_{i+2^k-1}\}$$

$$\max\{p|2^p \le r-l < 2^{p+1}\} = \max\{p|2^p \le 6-1 < 2^{p+1}\} = \max\{p|2^p \le 5 < 2^{p+1}\} = 2$$



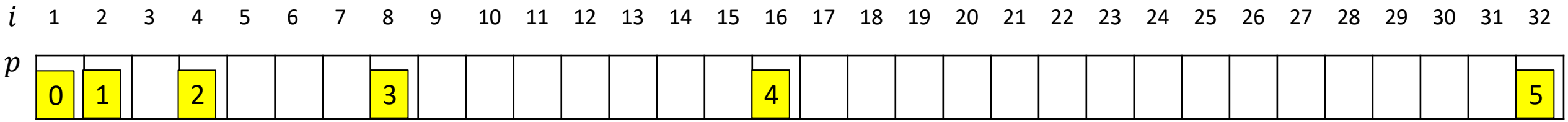
$$\textbf{FindMin}(1,6) = \min\{t_{p,l}, t_{p,r-2^p}\} = \min\{t_{2,1}, t_{2,6-2^2}\} = \min\{t_{2,1}, t_{2,2}\} = \min\{1,1\} = 1$$

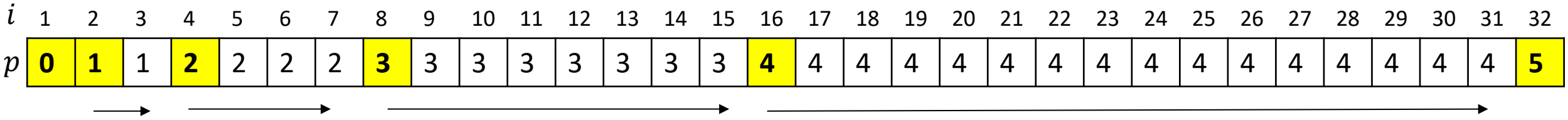
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Упражнение 1

- Дано целое число x ($1 \leq x \leq n$).
- Найти **наибольшее p такое, что $2^p \leq x$**, за константное время (для этого можно сделать предподсчёт).
- Подходы
  - Формула
  - Динамическое программирование

$$2^{p[i]} = i$$


i 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
p
0 1 2 3 4 5


i 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
p
0 1 1 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Упражнение 2

- Задана полоска из n клеток, клетки, некоторые клетки покрашены в чёрный цвет, остальные в белый

- Нужно уметь выполнять две операции:
  - $\mathrm{Invert}(l, r)$ — на интервале $[l, r)$ изменить цвет на противоположный
  - $\mathrm{GetColor}(i)$ — получить цвет i-й клетки

Общие задачи

iRunner

0.2 Задача о сумме (реализация структур для интервальных запросов - сумма на отрезке)

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Спасибо за внимание!