Интересные алгоритмы

Горденко Мария Константиновна

Префиксные суммы

Префиксными суммами массива $[a_0, a_1, a_2, \dots, a_{n-1}]$ называется массив $[s_0, s_1, s_2, \dots, s_n]$, определенный следующим образом:

$$\begin{cases} s_0 = 0 \\ s_1 = a_0 \\ s_2 = a_0 + a_1 \\ s_3 = a_0 + a_1 + a_2 \\ \dots \\ s_n = a_0 + a_1 + a_2 + \dots + a_{n-1} \end{cases}$$

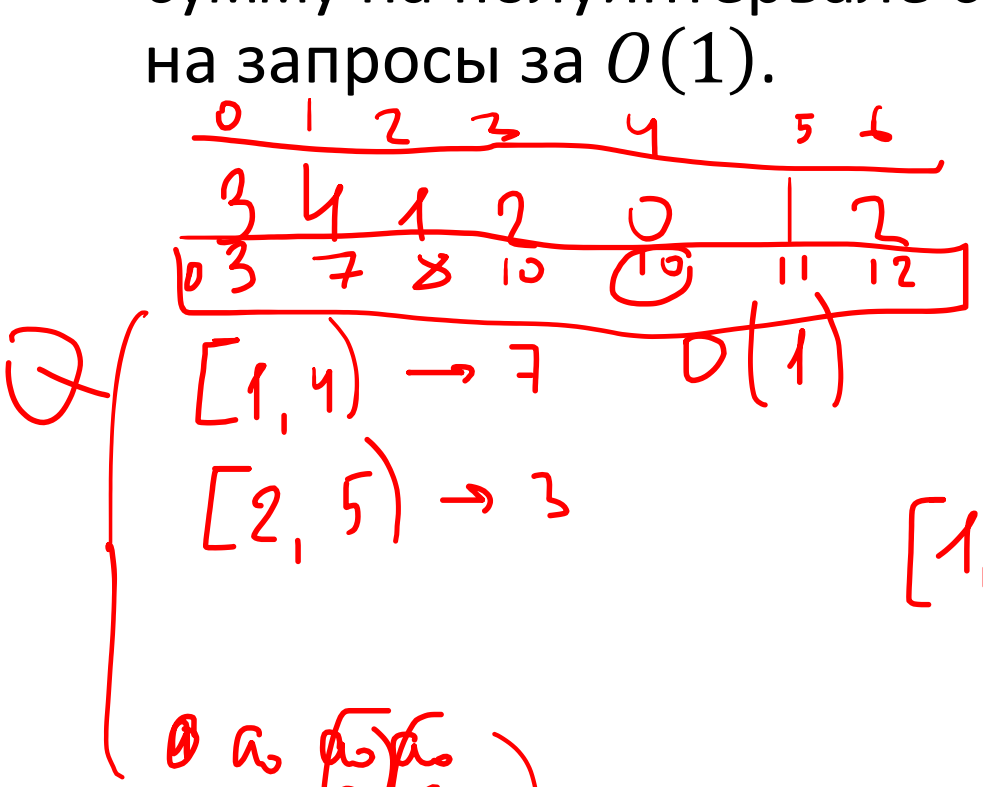
Обратите внимание, что в такой индексации

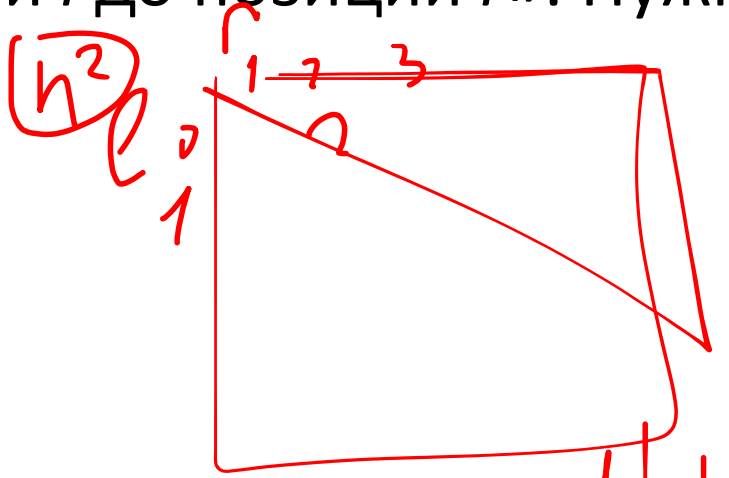
- s_k равен сумме первых *k* массива *a*, не включая a_k,
- длина *s* на единицу больше длины *a*,
- s_0 всегда равен нулю.

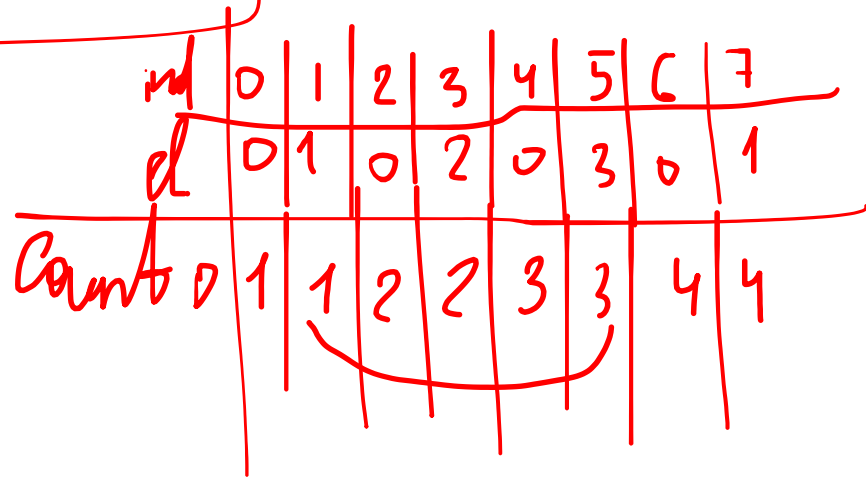
```
1.  int *s = new int[n + 1];
2.  s[0] = 0;
3.  for (int i = 0; i < n; ++i)
4.    s[i + 1] = s[i] + a[i];
5.  . . . .
6.  delete[] s;
```

Задача

Задача. Дан массив целых чисел, и приходят запросы вида «найти сумму на полуинтервале с позиции *l* до позиции *r*». Нужно отвечать на запросы за $O(1)$.







0	1	2	3	4	5
4	1	2	0	2	3

prefix: 4 | 4 | 8 | 1 | 2 | 6

count: 0 | 0 | 0 | 1 | 1 | 1

[1, 4) → 2 | 8 | 4 → 0

[2, 5) → 4 | 12 | 4 → 0 ← 4

[0, 2) → 4

[4, 5) → 2

if (count[l] == count[r]) p[r] | p[l]

else 0

Задача

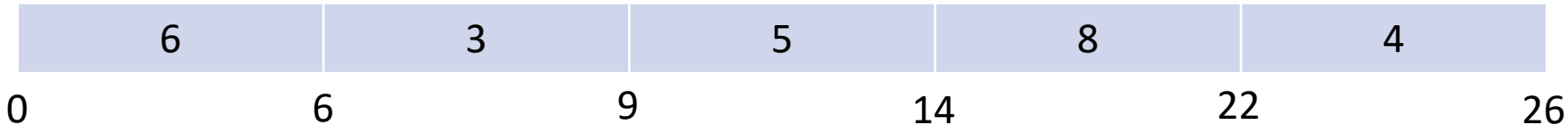
Произведем предварительный подсчет перед ответами на запросы: массив префиксных сумм для исходного массива. Тогда в случае, если бы во всех запросах левая граница s_l была бы равна нулю, то ответом на запрос была бы стандартная префиксная сумма s_r, однако как поступить, если эта граница не равна нулю?

В префиксной сумме s_r содержатся все нужные нам элементы, однако присутствуют и лишние, а именно $a_0, a_1, \dots, a_{r-1}$. Заметим, что такая сумма будет равна посчитанной префиксной сумме s_l.

Таким образом, выполняется тождество:

$$a_l + a_{l+1} + \cdots + a_{r-1} = s_r - s_l$$

Для ответа на запрос поиска суммы на произвольном полуинтервале необходимо вычесть друг из друга две известные префиксные суммы.



Другие операции

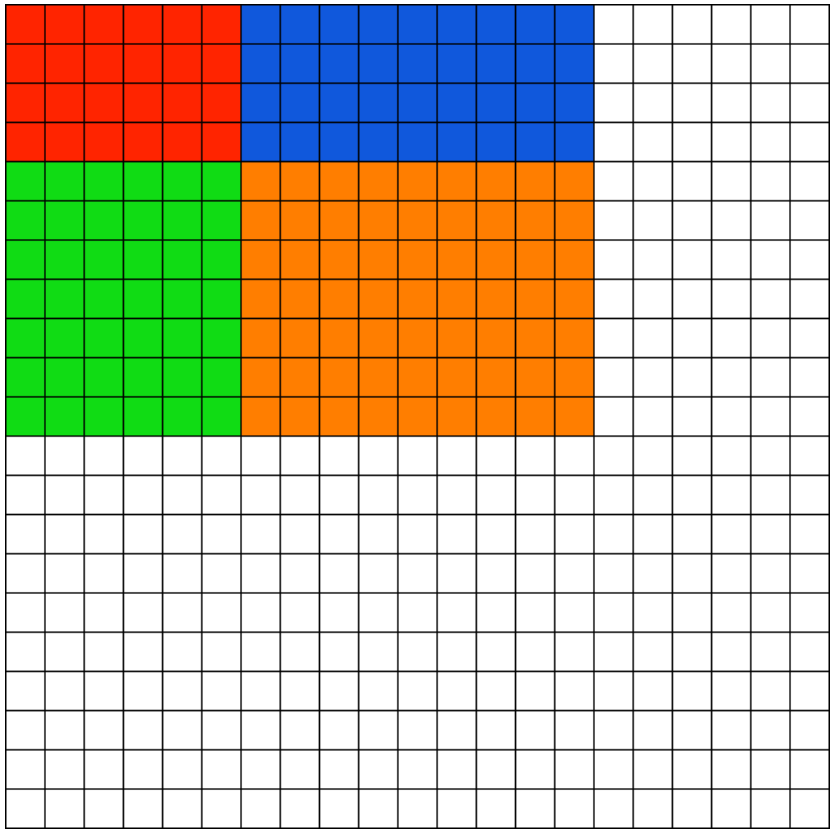
- побитовое исключающее «или», также известное как xor и обозначаемое $\oplus$,
- сложение по модулю,
- умножение и умножение по модулю (обратное — деление).

Префиксные суммы в матрицах

Задача. Необходимо отвечать на запросы вида «найти сумму в прямоугольнике (x_1, y_1, x_2, y_2)».

Разложим его на следующие запросы на «префиксах»:

$$\begin{cases} sum(x_1, y_1, x_2, y_2) = sum(0,0, x_2, y_2) - \\ sum(0,0, x_1 - 1, y_2) - sum(0,0, x_2, y_1 - 1) + \\ sum(0,0, x_1 - 1, y_1 - 1) \\ pref[0][0] = a[0][0] \\ pref[0][i] = pref[0][i-1] + a[0][i] \\ pref[i][0] = pref[i-1][0] + a[i][0] \\ pref[i][j] = pref[i-1][j] + pref[i][j-1] - \\ pref[i-1][j-1] + a[i][j] \end{cases}$$



Задача

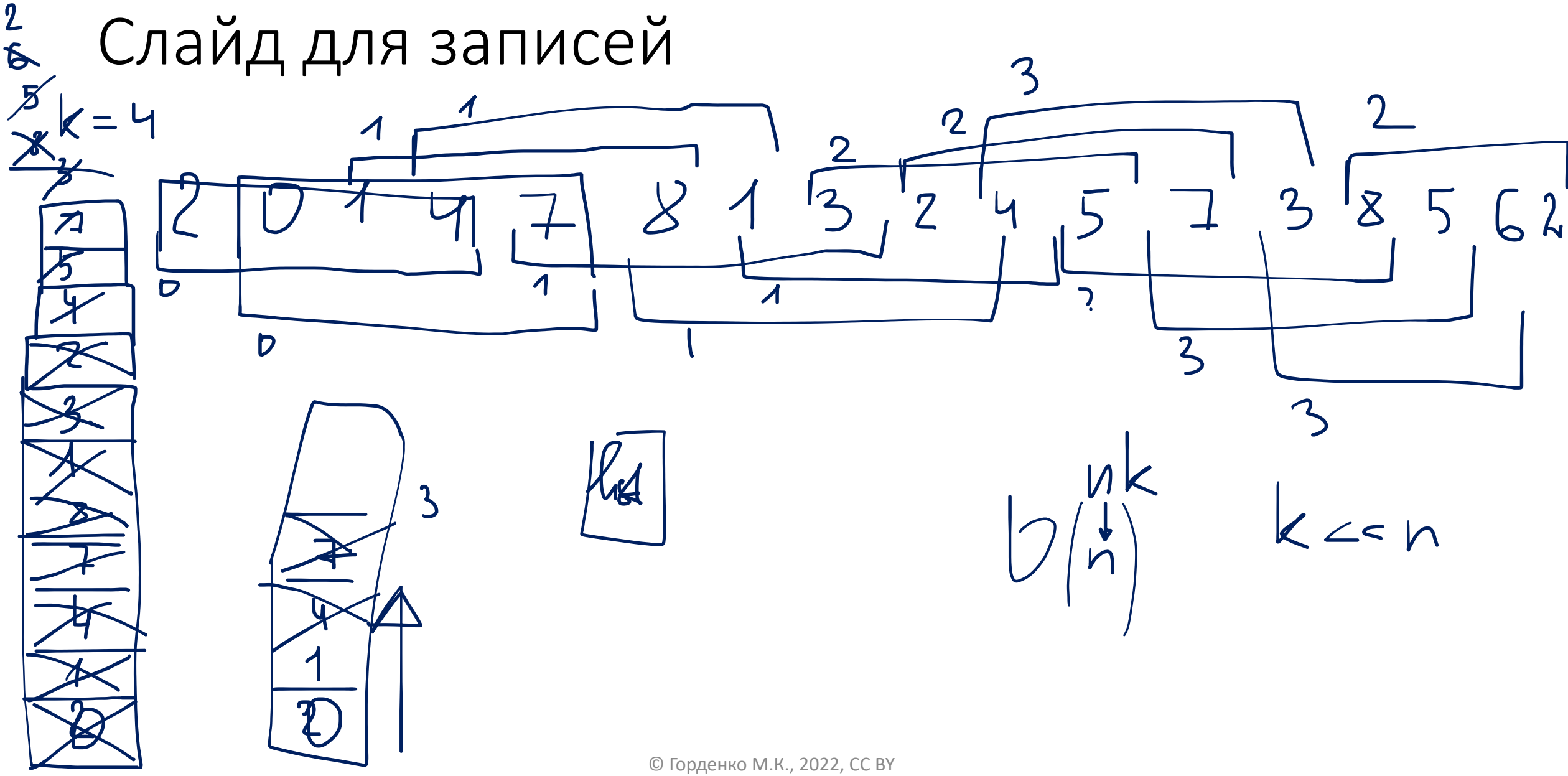
- **Задача.** Эффективно вычислять количество нулей в отрезке массива.

Слайд для записей

Задача

- **Задача.** Эффективно вычислять произведение на отрезке массива.

Слайд для записей



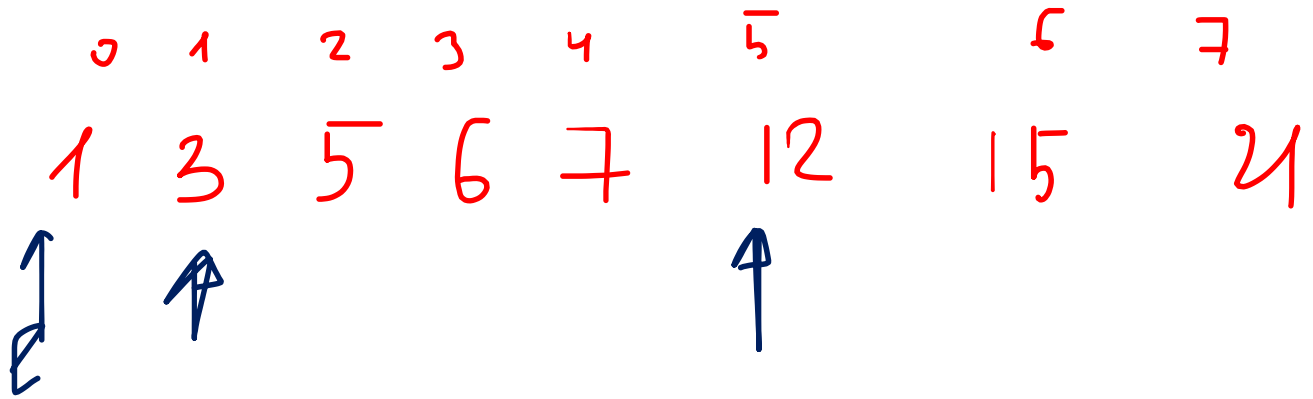
Два указателя

- Два указателя — это простой и эффективный метод, который обычно используется для поиска пар в отсортированном массиве.

Задача

O(n)

- **Задача.** Найти количество пар элементов *a* и *b* в отсортированном массиве, такие что $b - a > K$.




```
while (R < N):
    if (a[R] - a[L] > K)
        count += n - R
        L++
    else
        R++
```

Задача

- Наивное решение: бинарный поиск. Будем считать, что массив уже отсортирован. Для каждого элемента *a* найдем первый справа элемент *b*, который входит в ответ в паре с *a*. Нетрудно заметить, что все элементы, большие *b*, также входят в ответ.
  Итоговая сложность: $O(N * \log N)$.
- А можно ли быстрее? Да, давайте рассматривать **два указателя** – два индекса *first* и *second*. Будем перебирать *first* слева направо и поддерживать для каждого *first* такой первый элемент *second* справа от него, что $a[second] - a[first] > K$. Тогда в пару к $a = [first]$ подходят ровно $n - second$ элементов массива, начиная с *second*.

```
1.  int second = 0;
2.  int ans = 0;
3.  for (int first = 0; first < n; ++first) {
4.    while (second != n && a[second] - a[first] <= r)
5.      ++second;
6.    ans += n - second;
7.  }
```

Скользящее окно

- **Задача.** Для массива, состоящего из *N* целых чисел, найдите непрерывный подмассив заданной длины *k*, который имеет максимальное среднее значение. Нужно вывести максимальное среднее значение.

O(N)

Скользящее окно

- $n = 6, k = 3, nums = [4, 2, 6, 7, 6, 8]$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 4 + 2 + 6 = 12,$
$result = sum = 12$

Скользящее окно

- $n = 6, k = 3, nums = [4, 2, 6, 7, 6, 8]$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 4 + 2 + 6 = 12,$
$result = sum = 12$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 12 - 4 + 7 = 15,$
$result < sum, result = 15$

Скользящее окно

- $n = 6, k = 3, nums = [4, 2, 6, 7, 6, 8]$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 4 + 2 + 6 = 12,$
$result = sum = 12$

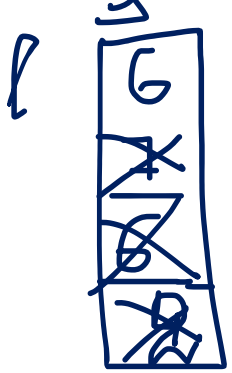
0	1	2	3	4	5
4	2	6	7	6	8

$sum = 12 - 4 + 7 = 15,$
$result < sum, result = 15$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 15 - 2 + 6 = 19,$
$result < sum, result = 19$

Скользящее окно



2 2 6 6

- $n = 6, k = 3, nums = [4, 2, 6, 7, 6, 8]$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 4 + 2 + 6 = 12,$
$result = sum = 12$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 12 - 4 + 7 = 15,$
$result < sum, result = 15$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 15 - 2 + 6 = 19,$
$result < sum, result = 19$

0	1	2	3	4	5
4	2	6	7	6	8

$sum = 15 - 6 + 8 = 17,$
$result > sum, result = 19$

Ответ: $\frac{result}{k} = \frac{19}{3}$

Скользящее окно

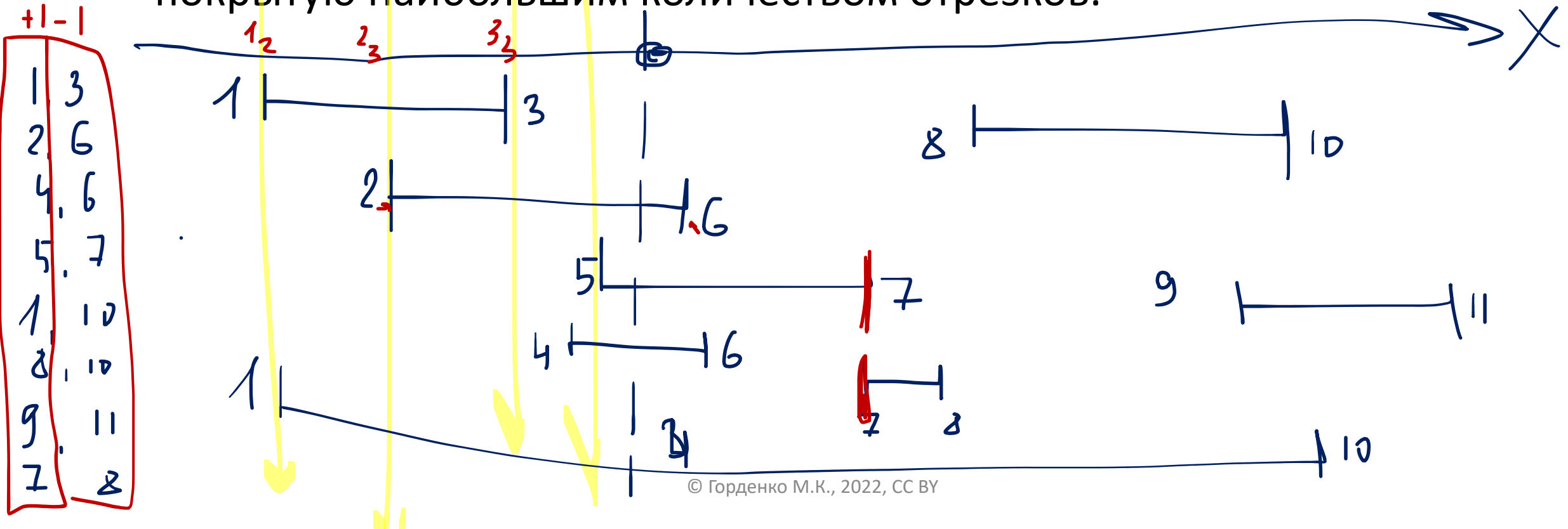
```
1.  int findMaxAverage(int nums[], int size, int k) {
2.    int sum = 0;

3.    for (int i = 0; i < k; ++i)    // Расчет изначальной суммы.
4.      sum += nums[i];

5.    int result = sum;

6.    for (int i = k; i < size; ++i) {      // Расчет и сдвиг суммы окон.
7.      sum += nums[i] - nums[i - k];
8.      result = (result < sum) ? sum : result;
9.    }

10.   return result / k;
11. }
```

Сортировка событий (сканирующая прямая)

- Метод сканирующей прямой (англ. *scanline*) заключается в сортировке точек или каких-то абстрактных *событий* (англ. *event*) и последующему проходу по ним.

Задача

- **Задача.** Дан набор из n отрезков на прямой, заданных координатами начал и концов $[li, ri]$. Требуется найти любую точку на прямой, покрытую наибольшим количеством отрезков.



Задача

```
1.   struct Event {
2.     int x;
3.     int type;
4.   };

5.   int scanLine(vector<pair<int, int>> segments) {
6.     vector<Event> events;
7.     for (auto [l, r] : segments) {
8.       events.push_back({l, 1});
9.       events.push_back({r, -1});
10.    }
11.    sort(events.begin(), events.end(),
12.         [](const Event &a, const Event &b) {
13.         return (a.x < b.x || (a.x == b.x && a.type > b.type));
14.    });
15.    int res = 0;
16.    int cnt = 0;
17.    for (const Event &event : events) {
18.      cnt += event.type;
19.      res = (res < cnt) ? cnt : res;
20.    }
21.    return res;
22.  }
```

Задача

- **Задача.** Дан набор из n отрезков на прямой, заданных координатами начал и концов $[li, ri]$. Требуется найти суммарную длину их объединения.

Задача

- **Задача.** Дан набор из *n* отрезков на прямой, заданных координатами начал и концов $[li, ri]$. Требуется найти суммарную длину их объединения.

```
1.  int res = 0;
2.  int cnt = 0;
3.  prev = -inf;
4.  for (auto &event : events) {
5.    cnt += event.type;
6.    if (prev != -inf && cnt > 0)
7.      res += prev - event.x;
8.    prev = event.x;
9.  }
```

Задача. Сложность

- Для анализа сложности выполнения обеих задач рассмотрим функцию $M(N)$, отражающую время выполнения некоторой сортировки на *N* элементах.
- Рассмотрим функцию работоспособности всего алгоритма и назовем ее $W(N)$. Рассмотрим построение «набора событий» - в случае заданных, это будет вектор точек начал и концов каждого отрезка, тогда для *N* отрезков он будет строиться за $O(2 * N) \sim O(N)$. Полученный набор событий сортируется за время $M(2 * N)$, и затем за линейное время $O(N)$ выполняется основная часть задания. Итоговая функция работоспособности $W(N)$ оценивается как $O(N + M(N) + N) \sim O\big(N + M(N)\big)$, тогда при использовании любой стандартной сортировок итоговая сложность будет приводиться к $O(N * \log N)$.