MUA

Maratones uniandes

SQUARE ROOT DECOMPOSITION

ISIS 2801

Finding queries

5	8	6	3	2	7	2	6	7	1	7	5	6	2	3	2

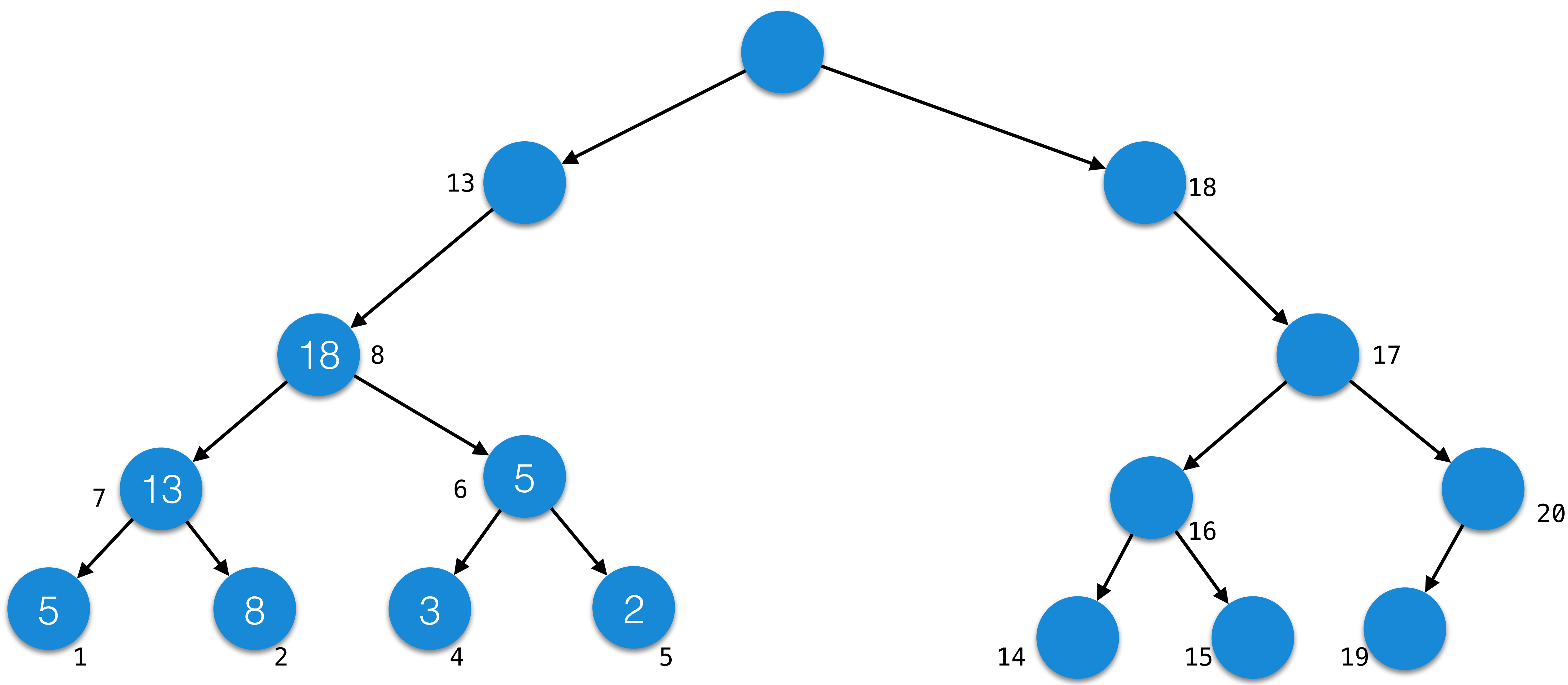
keep sum of element ranges

update elements

Finding queries

5	8	6	3	2	7	2	6	7	1	7	5	6	2	3	2

Binary indexed tree/fenwick tree



Finding queries

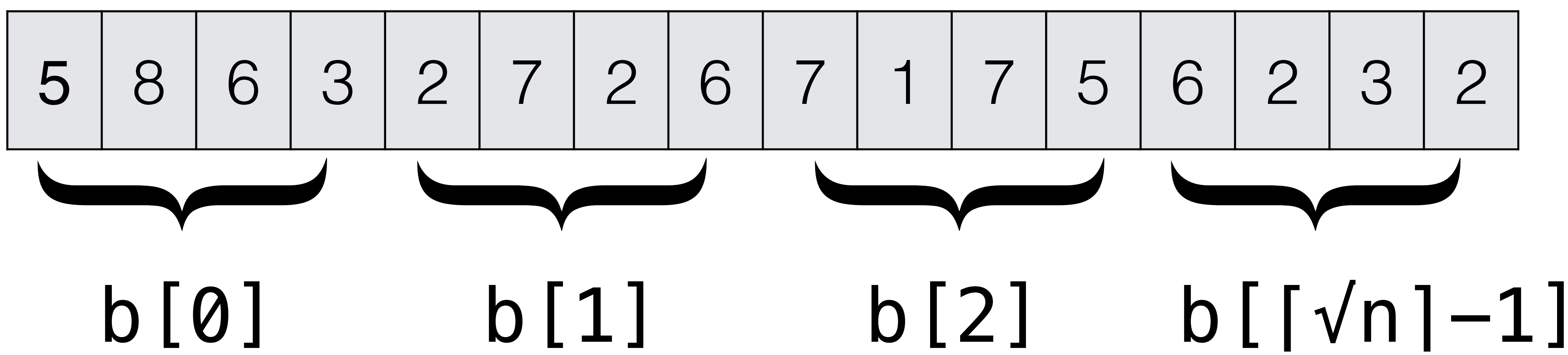
5	8	6	3	2	7	2	6	7	1	7	5	6	2	3	2

operate over element ranges

update elements

WE’LL DECOMPOSE THE STRUCTURE IN √ CHUNKS TO QUERY

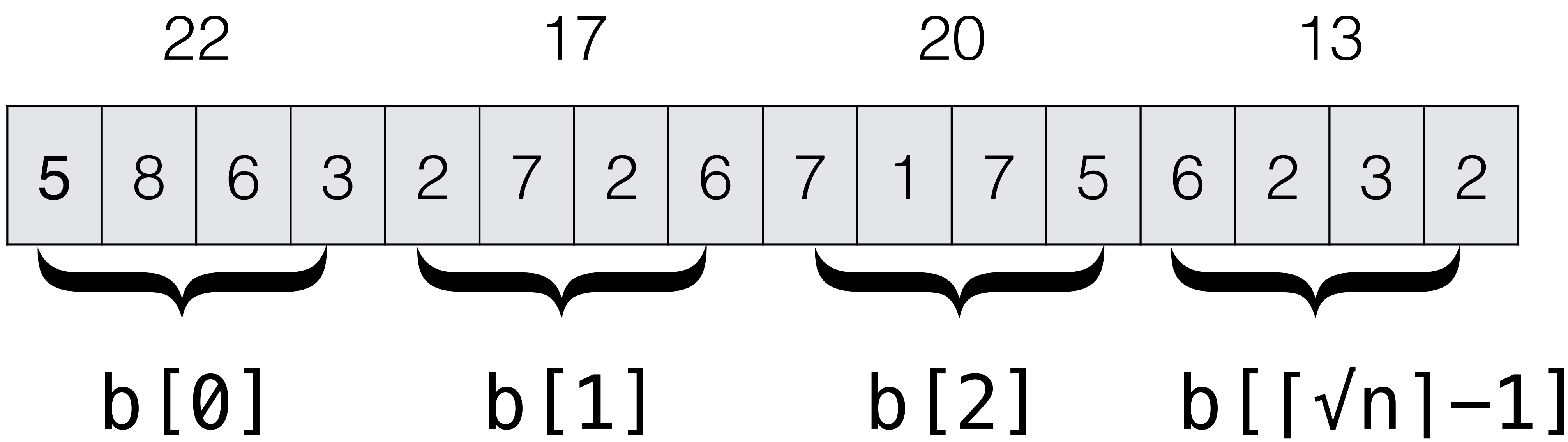
Range sum



1. divide the array in $\lceil\sqrt{n}\rceil$ chunks

number and **size** of the chunks

Range sum



1. divide the array in $\lceil\sqrt{n}\rceil$ chunks
2. sum the elements in the block

Range sum

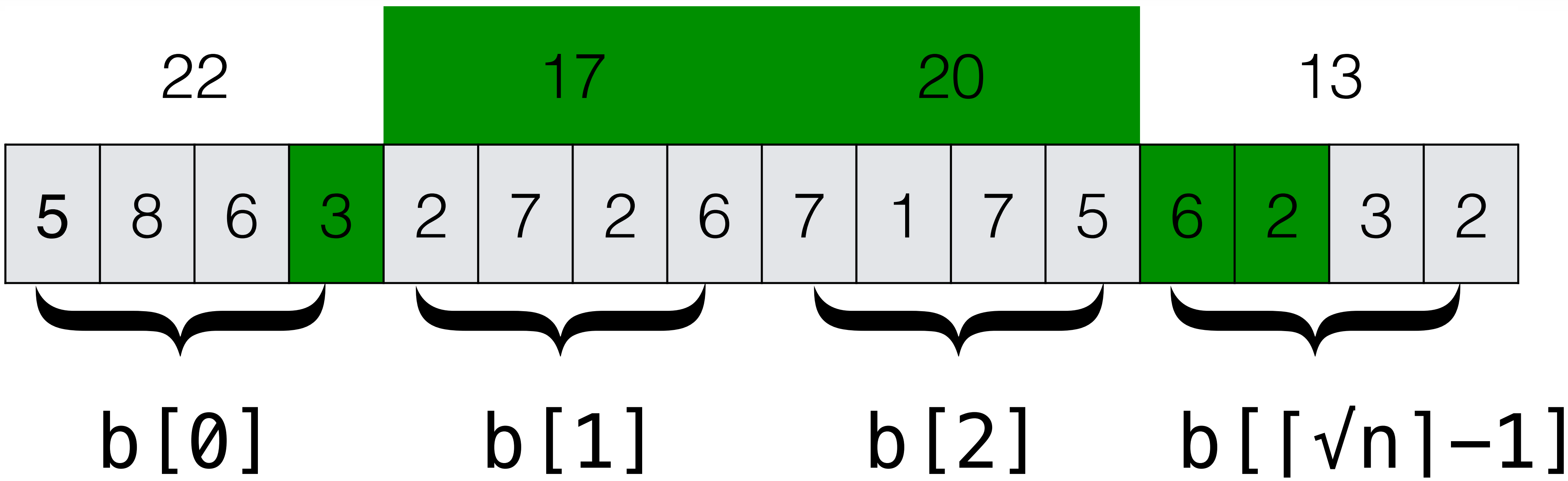
22				17				20				13			
5	8	6	3	2	7	2	6	7	1	7	5	6	2	3	2
b[0]				b[1]				b[2]				b[⌈√n⌉-1]			

1. divide the array in ⌈√n⌉ chunks
2. sum the elements in the block
3. calculate the sum of the range [l,r]

$$\sum_{l}^{r} a[i] = \sum_{l}^{(k+1)s-1} a[i] + \sum_{k+1}^{p-1} b[i] + \sum_{ps}^{r} a[i]$$

Range sum

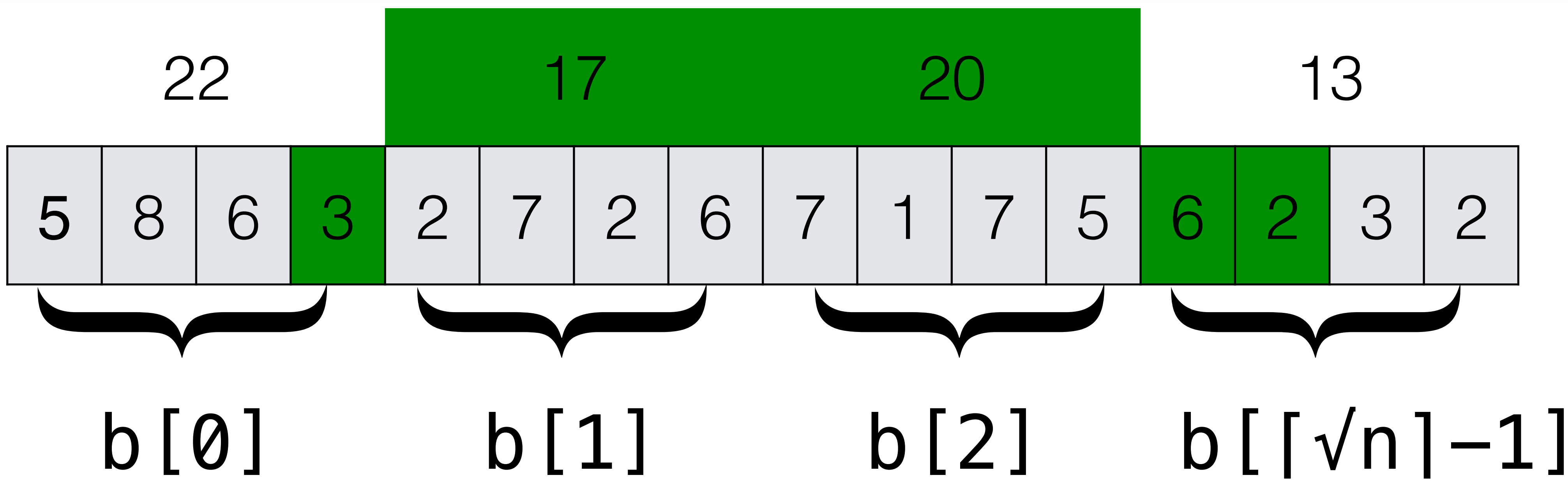
$$\sum_{3}^{13} a[i] = \sum_{3}^{3} a[i] + \sum_{1}^{2} b[i] + \sum_{12}^{13} a[i]$$




```
[l,r] := [3,13]
```

Range sum

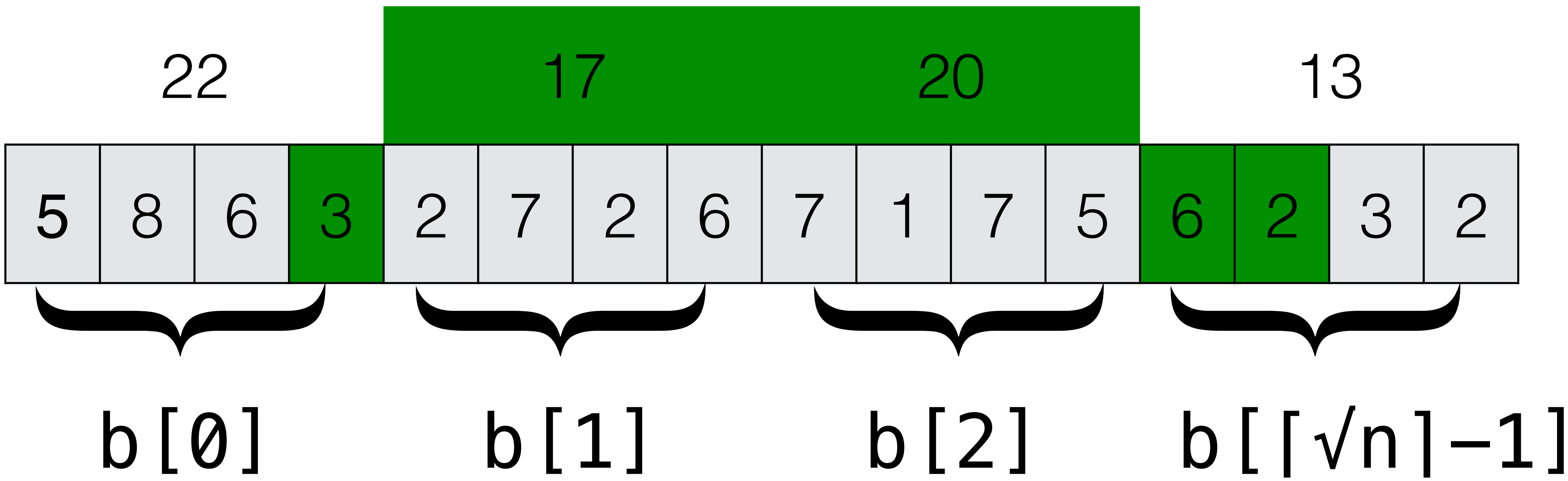
$$\sum_{3}^{13} a[i] = \sum_{3}^{3} a[i] + \sum_{1}^{2} b[i] + \sum_{12}^{13} a[i]$$




```
[l,r] := [3,13]
         3 + 17 + 20 + 6 + 2
         68
```

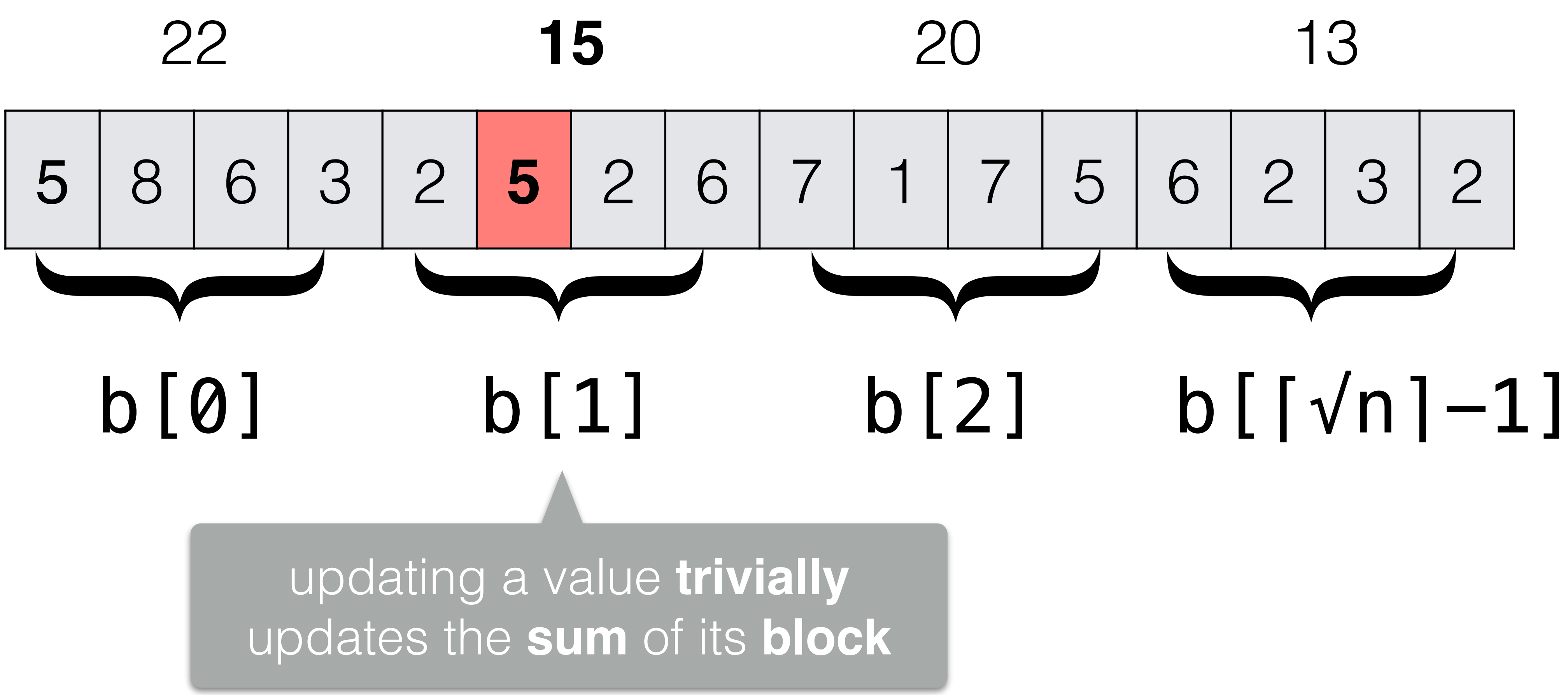
Range sum

$$\sum_{3}^{13} a[i] = \sum_{3}^{3} a[i] + \sum_{1}^{2} b[i] + \sum_{12}^{13} a[i]$$



- For the range `[l,r]`, there are maximum √n elements (per block) and √n blocks. Therefore the time complexity is **O(√n)**

Value update



FIRST IMPLEMENTATION

Sum range

block size

create the blocks

```
int s = (int) sqrt (n + .0) + 1;
vector<int> b (s);
for (int i=0; i<n; ++i)
    b[i / s] += a[i];

while(cin) {
    int l, r;
    cin >> l >> r;
    int sum = 0;
    for(int i=l; i<=r;)
        if (i % s == 0 && i + s - 1 <= r) {
            sum += b[i / s];
            i += s;
        } else {
            sum += a[i];
            ++i;
        }
}
```

sum blocks

sum tails

Sum range

```
int s = (int) sqrt (n + .0) + 1;
vector<int> b (s);
for (int i=0; i<n; ++i)
    b[i / s] += a[i];

while(cin) {
    int l, r;
    cin >> l >> r;
    int sum = 0;
    for(int i=l; i<=r;)
        if (i % s == 0 && i + s - 1 <= r) {
            sum += b[i / s];
            i += s;
        } else {
            sum += a[i];
            ++i;
        }
}
```

too much effort in calculating block indices

Sum range

```
int s = (int) sqrt (n + .0) + 1;
vector<int> b (s);
for (int i=0; i<n; ++i)
    b[i / s] += a[i];

while(cin) {
    int l, r;
     cin >> l >> r;
    int sum = 0;
    int c_l = l / s,   c_r = r / s;
    if (c_l == c_r)
      for (int i=l; i<=r; ++i)
         sum += a[i];
    else {
      for (int i=l, end=(c_l+1)*s-1; i<=end; ++i)
         sum += a[i];
      for (int i=c_l+1; i<=c_r-1; ++i)
         sum += b[i];
      for (int i=c_r*s; i<=r; ++i)
         sum += a[i];
    }
}
```

calculate block indices

MO's ALGORITHM

Value update

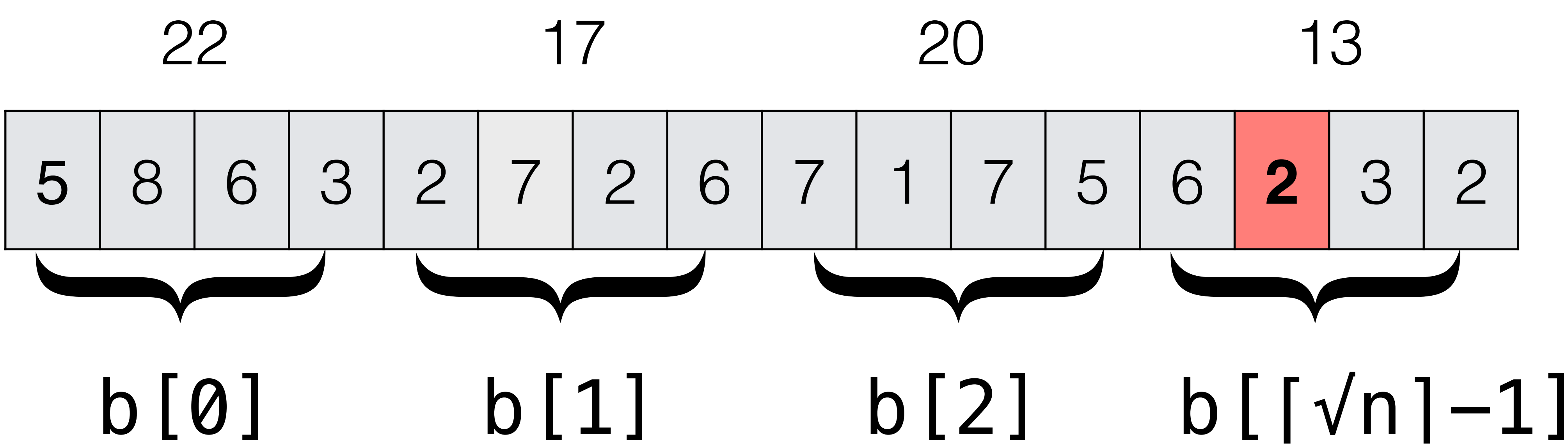
`mode(a) = 2`

22	17	20	13
5 8 6 3	2 7 2 6	7 1 7 5	6 2 3 2
b[0]	b[1]	b[2]	b[$\lceil\sqrt{n}\rceil$−1]

updating to calculate the mode

Value update

`mode(a) = 2`



have to take into account the number of times each number appears in each block

that **isn't efficient** anymore

Mo's algorithm

1. Order the queries (by left index)
2. Update the range by **adding** or **deleting** elements

> Queries need to be answered
> **offline!**

Mo’s algorithm

```
struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};
```

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(arr, cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(arr, cur_r);
        }
        …
    }
    return answers;
}
```

The data structure **always reflects** the range **[cur_l, cur_r]**

add missing elements left and right

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    …
    for (Query q : queries) {
        …
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

The data structure **always reflects** the range **[cur_l, cur_r]**

remove elements left and right of the range

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {

        …

        answers[q.idx] = get_answer();
    }
    return answers;
}
```

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());        O(Q logQ)
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {

        …

        answers[q.idx] = get_answer();
    }
    return answers;
}
```

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {

        …

        answers[q.idx] = get_answer();
    }
    return answers;
}
```

O(Q logQ)

$O((Q + N)\sqrt{N})$

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {

        …

        answers[q.idx] = get_answer();
    }
    return answers;
}
```

O(Q logQ)

$O((Q + N)\sqrt{N})$

$O(\sqrt{N})$

Mo’s algorithm

```
vector<int> mos_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    vector<int> arr(0);
    int cur_l = 0;
    int cur_r = -1;
    for (Query q : queries) {

        …

        answers[q.idx] = get_answer();
    }
    return answers;
}
```

O(Q logQ)

$O((Q + N)\sqrt{N})$

$O(\sqrt{N})$

$O((N+Q)\sqrt{N})$