Дерево отрезков

Задача

Имеется массив a, состоящий из N

($N \geq 100000$) элементов.

Требуется уметь выполнять 2 операции:

1. Найти максимальный среди элементов a_l, …, a_r
2. Изменить элемент a_{pos}

Вариации

Имеется массив a, состоящий из N

($N \geq 100000$) элементов.

Требуется уметь выполнять 2 операции:

1. Найти результат некоторой ассоциативной операции среди элементов a_l, …, a_r
2. Изменить элемент a_{pos}

Вариации

Имеется массив a, состоящий из N

($N \geq 100000$) элементов.

Требуется уметь выполнять 2 операции:

1. Найти результат некоторой ассоциативной операции среди элементов a_l, …, a_r
2. Изменить элемент a_{pos}

Основные ассоциативные операции: сумма,

произведение, НОД/НОК, максимум/минимум

Задача

Имеется массив a, состоящий из N

($N \geq 100000$) элементов.

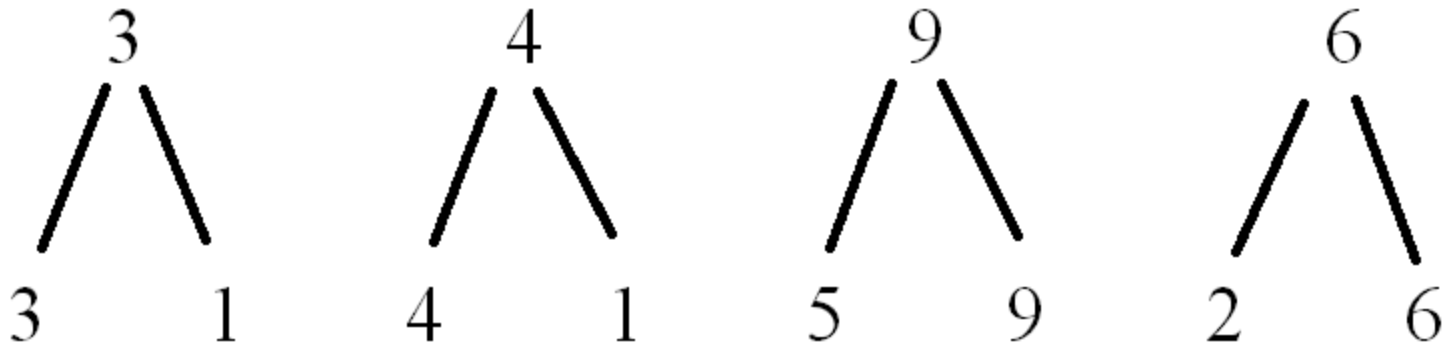
Требуется уметь выполнять 2 операции:

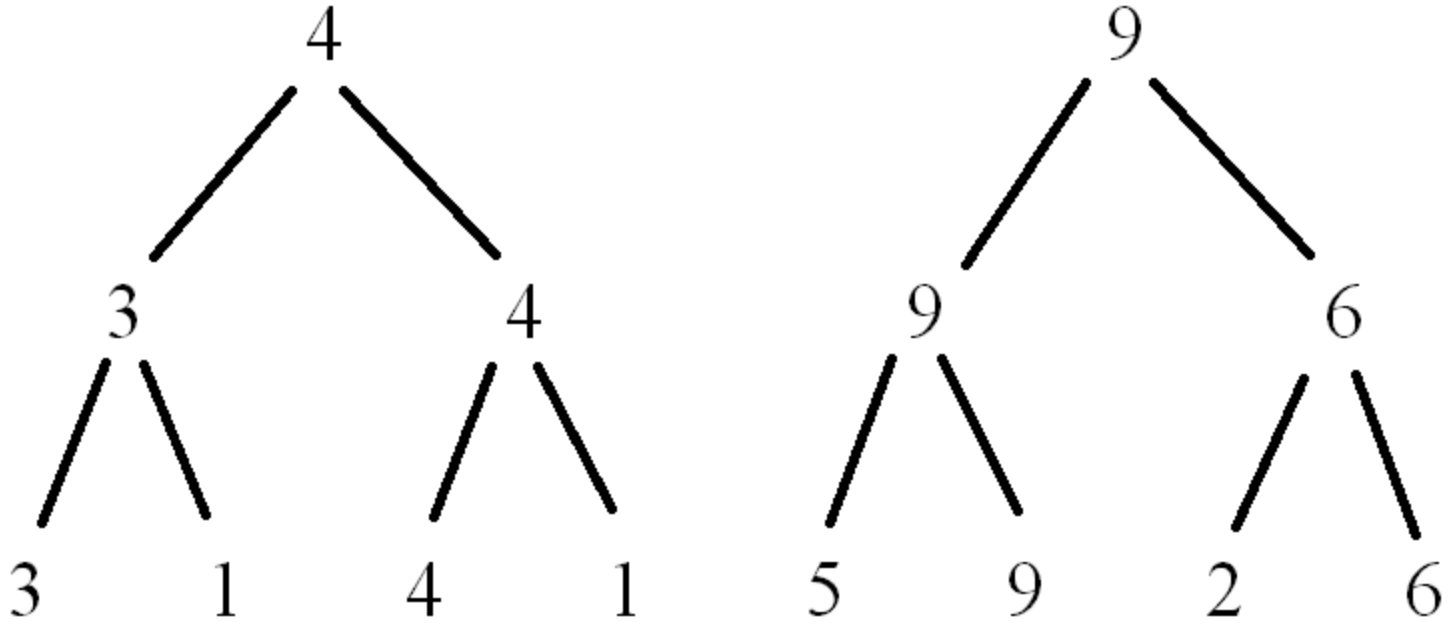
1. Найти максимальный среди элементов a_l, …, a_r
2. Изменить элемент a_{pos}

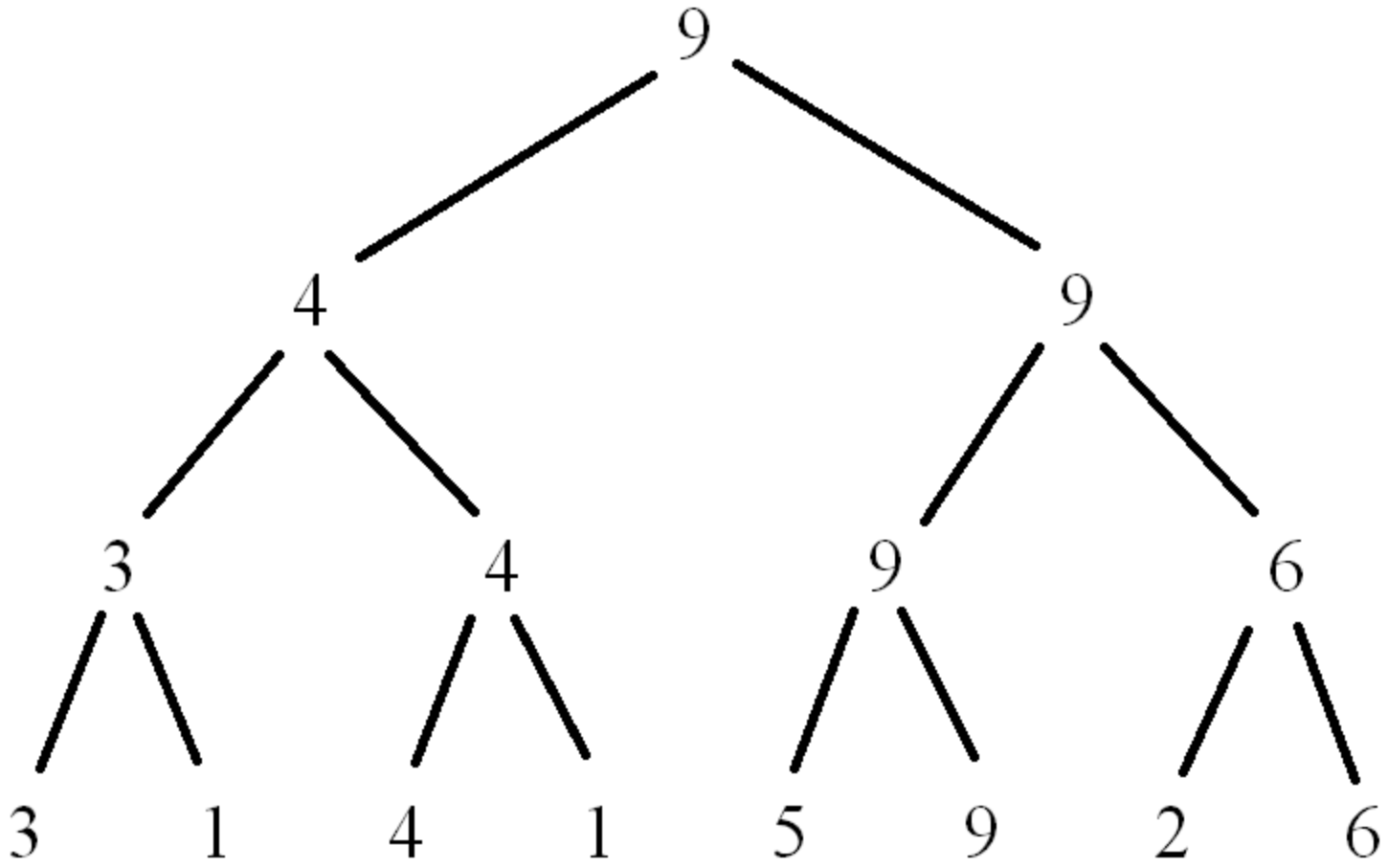
Отвечать на каждый запрос надо уметь за

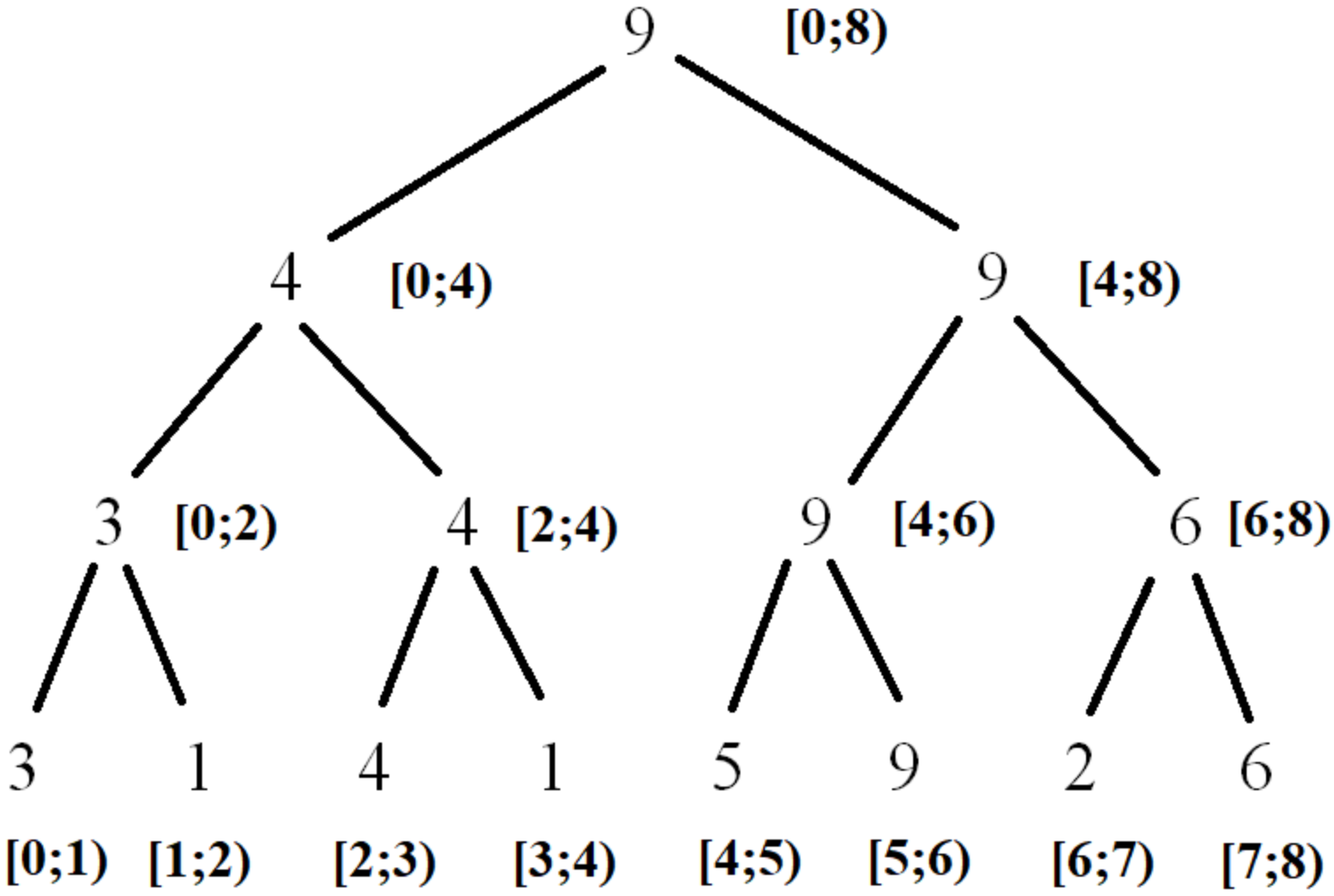
O(logN) времени

3 1 4 1 5 9 2 6


3
3
1
4
4
1
9
5
9
6
2
6


4
3
4
3
1
4
1
9
9
6
5
9
2
6


9
4
9
3
4
9
6
3
1
4
1
5
9
2
6

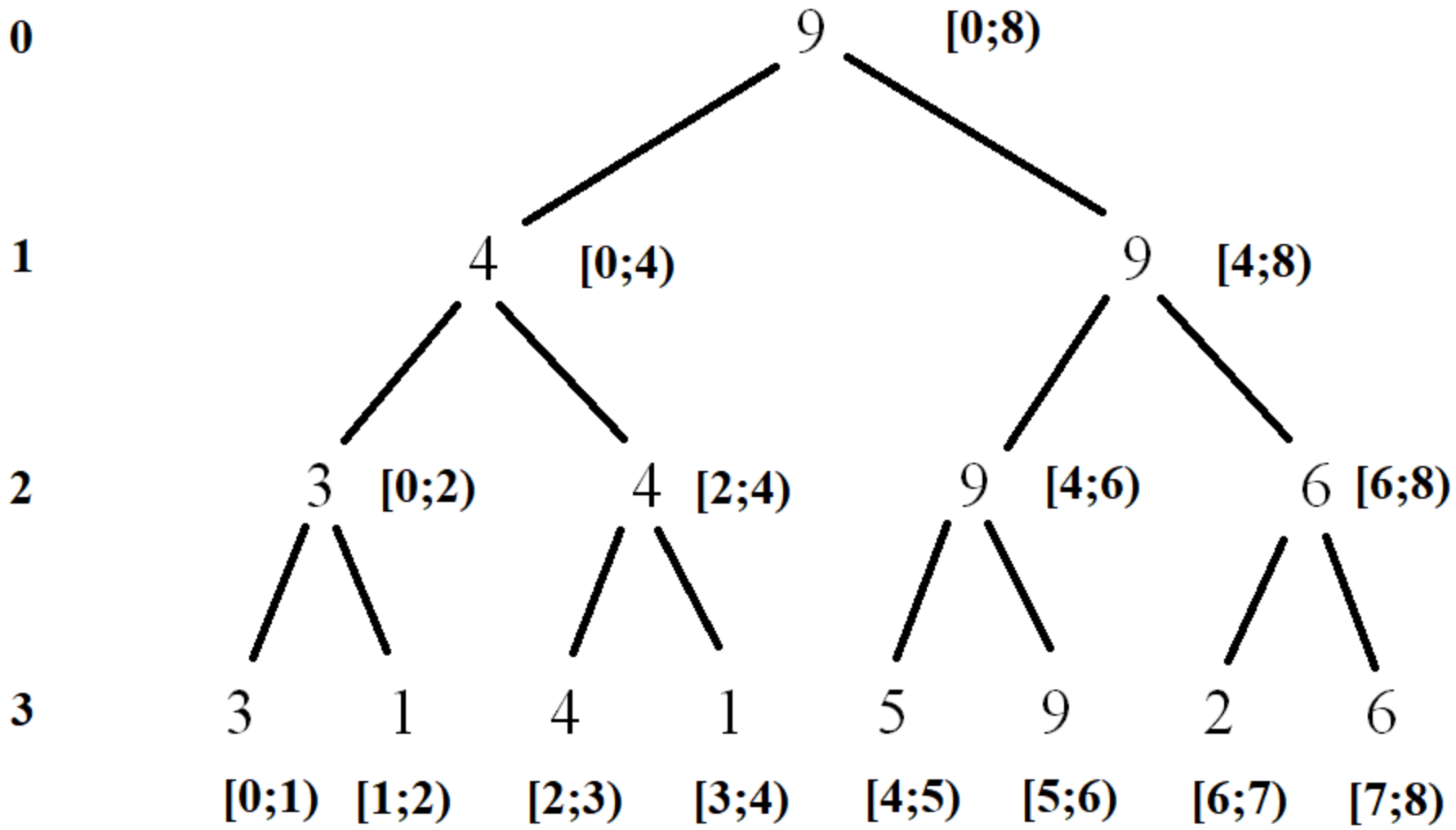

9 [0;8)
4 [0;4)
9 [4;8)
3 [0;2)
4 [2;4)
9 [4;6)
6 [6;8)
3 1 4 1 5 9 2 6
[0;1) [1;2) [2;3) [3;4) [4;5) [5;6) [6;7) [7;8)

Оценка требуемой памяти

Пусть $N = 2^k$.

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки.


0
9
[0;8)
1
4
[0;4)
9
[4;8)
2
3
[0;2)
4
[2;4)
9
[4;6)
6
[6;8)
3
1
4
1
5
9
2
6
[0;1)
[1;2)
[2;3)
[3;4)
[4;5)
[5;6)
[6;7)
[7;8)

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой. В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.

В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.

Всего вершин – $2^k + 2^{k-1} + \ldots + 1$.

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.

В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.

Всего вершин – $2^k + 2^{k-1} + \ldots + 1$.

$2^k + 2^{k-1} + \ldots + 1 =$

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.

В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.

Всего вершин – $2^k + 2^{k-1} + \ldots + 1$.

$2^k + 2^{k-1} + \ldots + 1 = 2^k * (1 – 1/2^{k+1}) / (1 – 1/2) = .$

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.

В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.

Всего вершин – $2^k + 2^{k-1} + \ldots + 1$.

$2^k + 2^{k-1} + \ldots + 1 = 2^k * (1 – 1/2^{k+1}) / (1 – 1/2) = 2^{k+1} * (1 – 1/2^{k+1}) = 2^{k+1} – 1$.

Оценка требуемой памяти

Пусть $N = 2^k$. Если это не так, дополним размер массива до ближайшей степени двойки. Всего в массиве будет (k + 1) слой.
В k-м слое будет 2^k вершин, (k – 1)-м слое – 2^{k-1}, …, в 0-м слое – 1 вершина.
Всего вершин – $2^k + 2^{k-1} + \ldots + 1$.
$2^k + 2^{k-1} + \ldots + 1 = 2^k * (1 – 1/2^{k+1}) / (1 – 1/2) = 2^{k+1} * (1 – 1/2^{k+1}) = 2^{k+1} – 1$.
Потребуется дополнительно 2N – 1 ячеек памяти.

Дополнение массива

```
int n;
cin >> n;
vector <int> a(n);
for(int i = 0; i < n; ++i){
    cin >> a[i];
}
int l = 1;
while (l < n){
    l *= 2;
}
// 0 .. n - 1 -> 0 .. l - 1
for(int i = n; i < l; ++i){
    a.push_back(-INF);
}
```

Дополнение массива

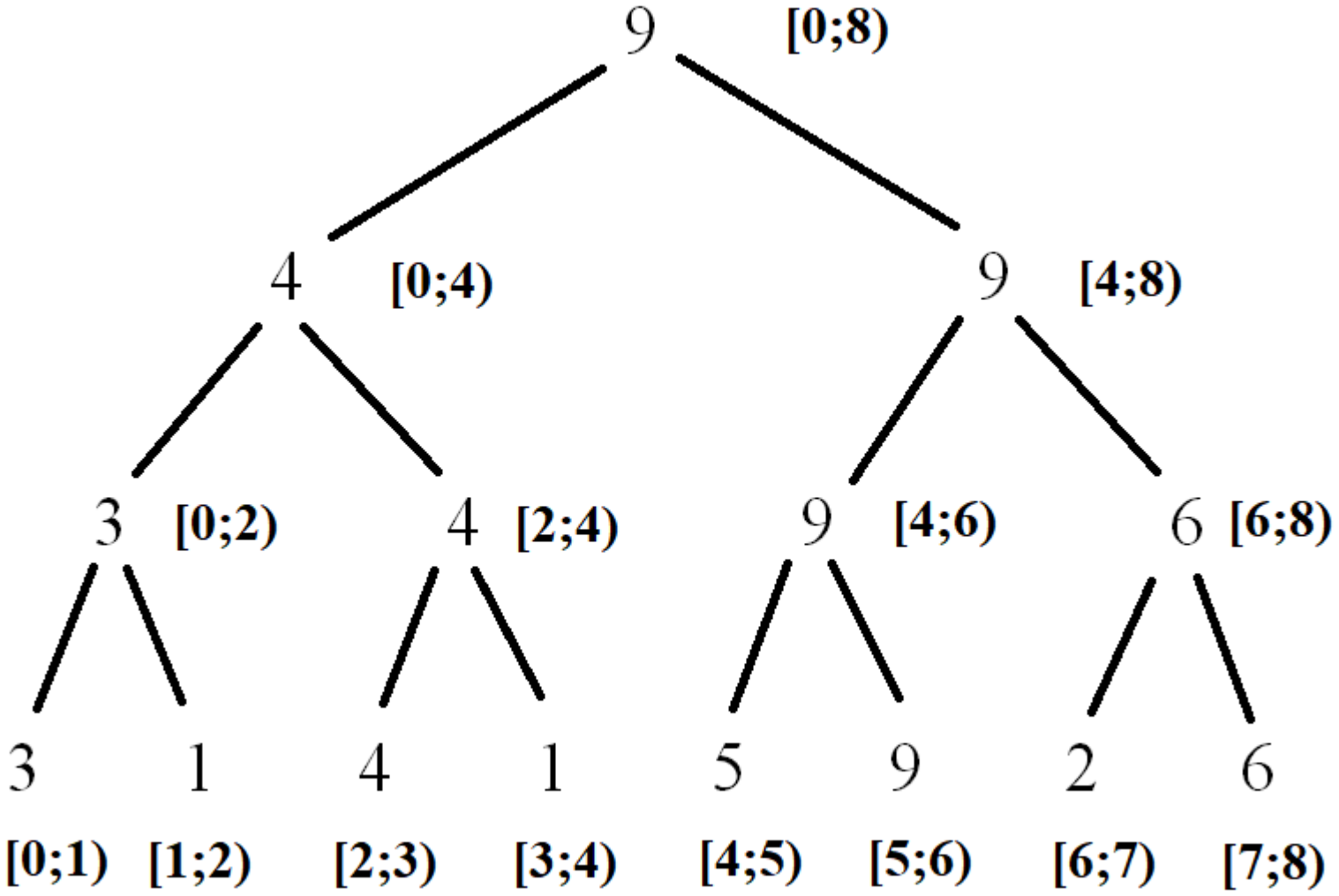
```
int n;
cin >> n;
vector <int> a(n);
for(int i = 0; i < n; ++i){
    cin >> a[i];
}
int l = 1;
while (l < n){
    l *= 2;
}
// 0 .. n - 1 -> 0 .. l - 1
for(int i = n; i < l; ++i){
    a.push_back(-INF);
}
```

Замечание: для нахождения минимума нужно будет добавить бесконечности, для суммы, НОД – 0, для произведения – 1.

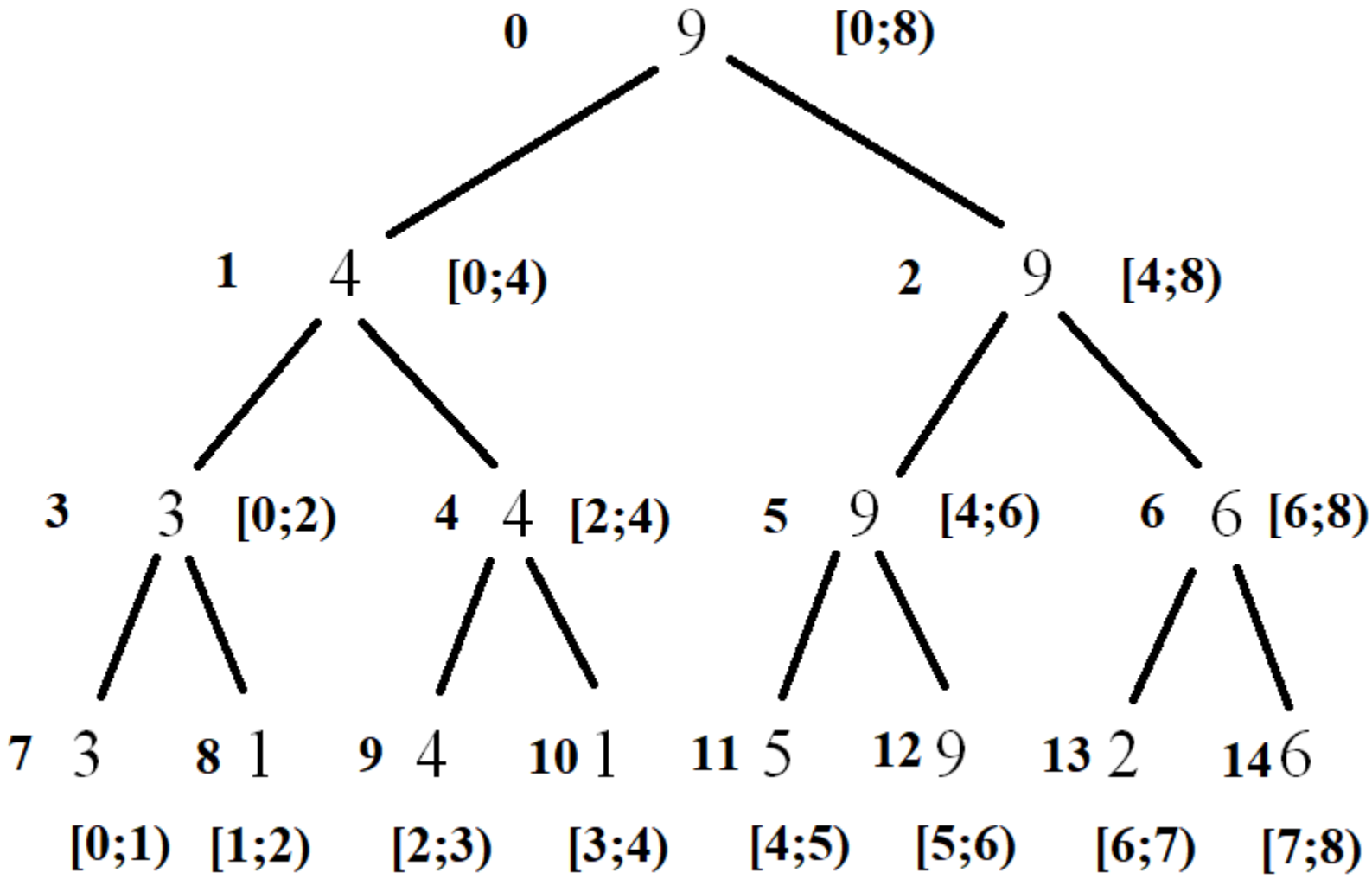
Если не хотим дополнять до ближайшей степени двойки?

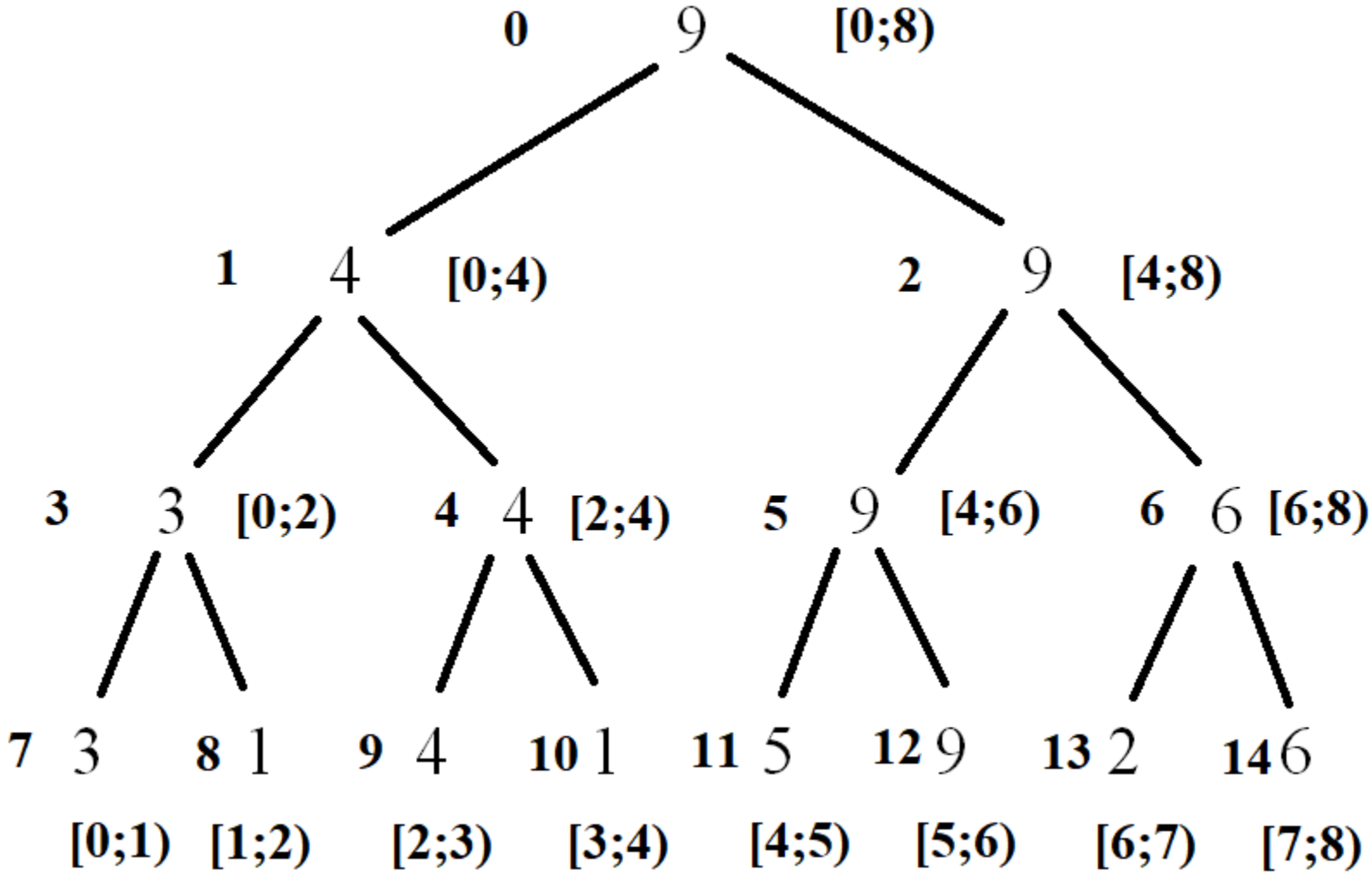
Объявляем массив на 4N ячеек памяти.

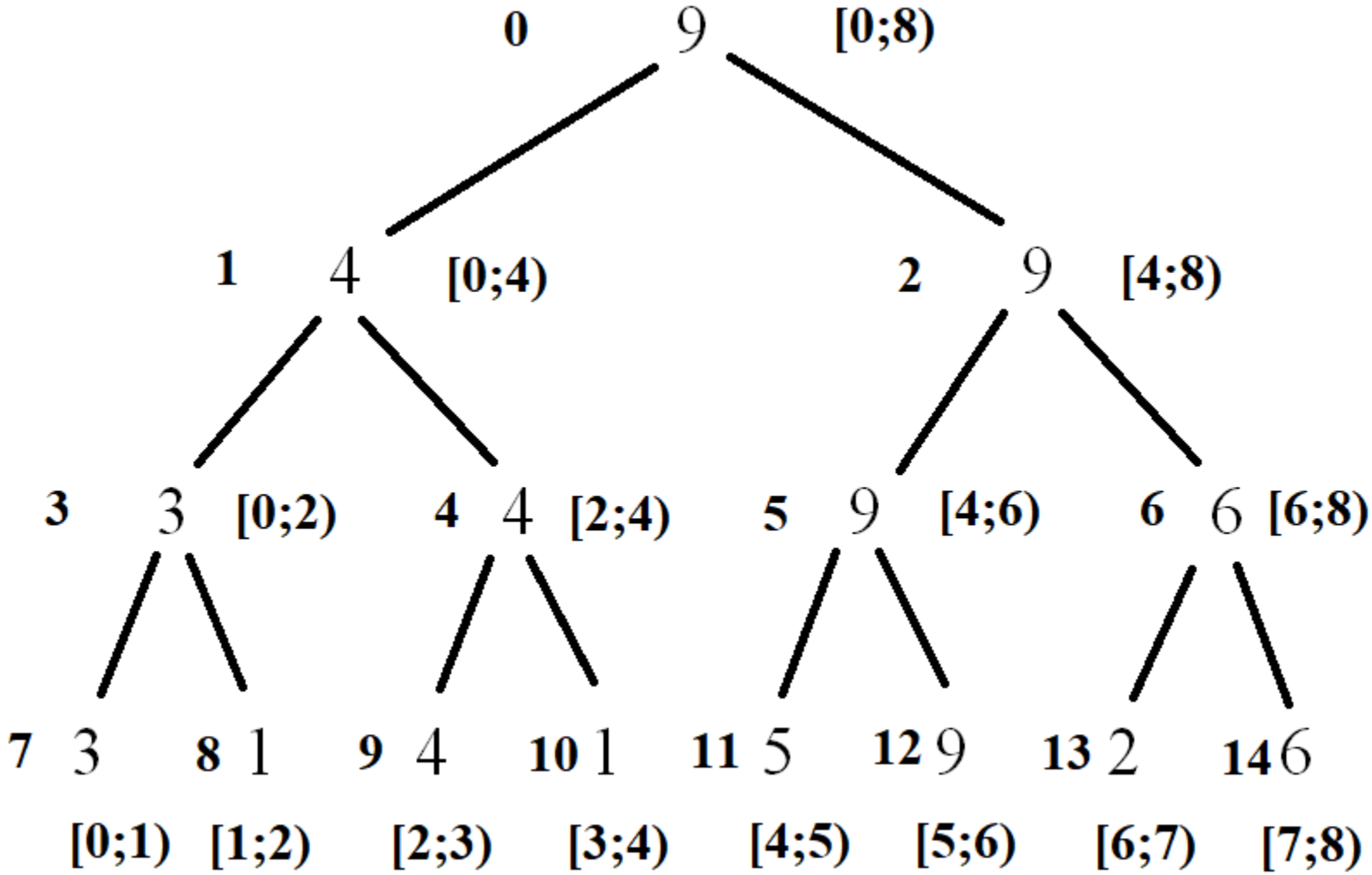
Нумерация вершин



Нумерация вершин




0 9 [0;8)
1 4 [0;4)
2 9 [4;8)
3 3 [0;2)
4 4 [2;4)
5 9 [4;6)
6 6 [6;8)
7 3 [0;1)
8 1 [1;2)
9 4 [2;3)
10 1 [3;4)
11 5 [4;5)
12 9 [5;6)
13 2 [6;7)
14 6 [7;8)



У вершины с номером i сыновьями будут вершины 2 * i + 1 и 2 * i + 2

Построение и запросы

Построение и запросы

Основные принципы:

1. Где я?
    1.1.* Раздать детям
2. Вызваться от детей
3. Пересчитать себя

Построение и запросы

Основные принципы:

1. Где я?
    1.1.* Раздать детям
2. Вызваться от детей
3. Пересчитать себя

Построение начинаем с корня (такое построение называется построением сверху).

Построение сверху

- Спустимся от корня до листьев

Построение сверху

- Спустимся от корня до листьев
- Лист соответствует одному элементу массива

Построение сверху

- Спустимся от корня до листьев
- Лист соответствует одному элементу массива
- Пересчитаем значения для каждого листа

Построение сверху

- Спустимся от корня до листьев
- Лист соответствует одному элементу массива
- Пересчитаем значения для каждого листа
- Вернёмся к родителю, пересчитаем его

Как узнать, что мы листе?

Как узнать, что мы листе?

- Листу соответствует отрезок [l; l + 1), то есть r – l = l + 1 – l = 1

Построение

```
vector <int> t(4 * MAXN), a;
void build(int v, int l, int r){
    if (r - l == 1){
        t[v] = a[l];
    }
    else{
        int m = (l + r) / 2;
        build(2 * v + 1, l, m);
        build(2 * v + 2, m, r);
        t[v] = max(t[2 * v + 1], t[2 * v + 2]);
    }
}
```

Построение

```
vector <int> t(4 * MAXN), a;
void build(int v, int l, int r){
    if (r - l == 1){
        t[v] = a[l];
    }
    else{
        int m = (l + r) / 2;
        build(2 * v + 1, l, m);
        build(2 * v + 2, m, r);
        t[v] = max(t[2 * v + 1], t[2 * v + 2]);
    }
}
```

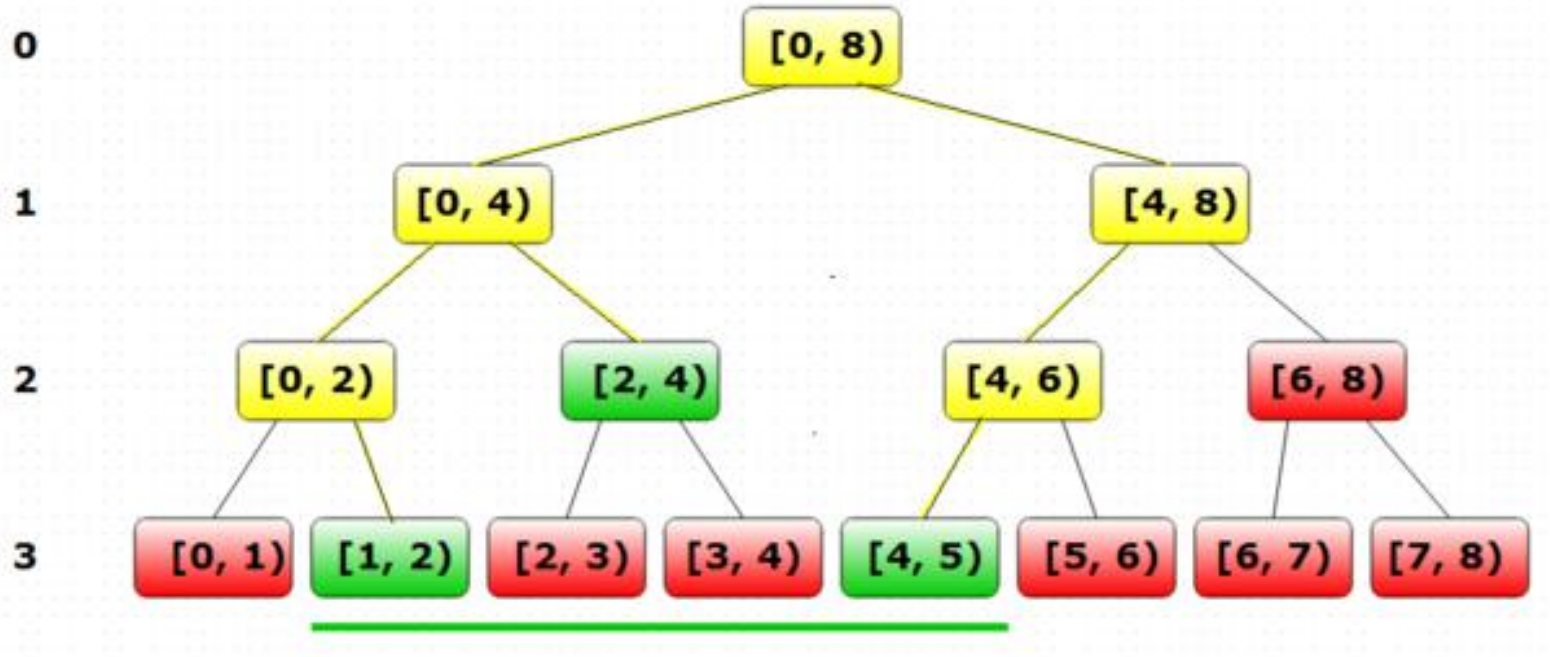
Вызов – build(0, 0, n);

Максимум на отрезке

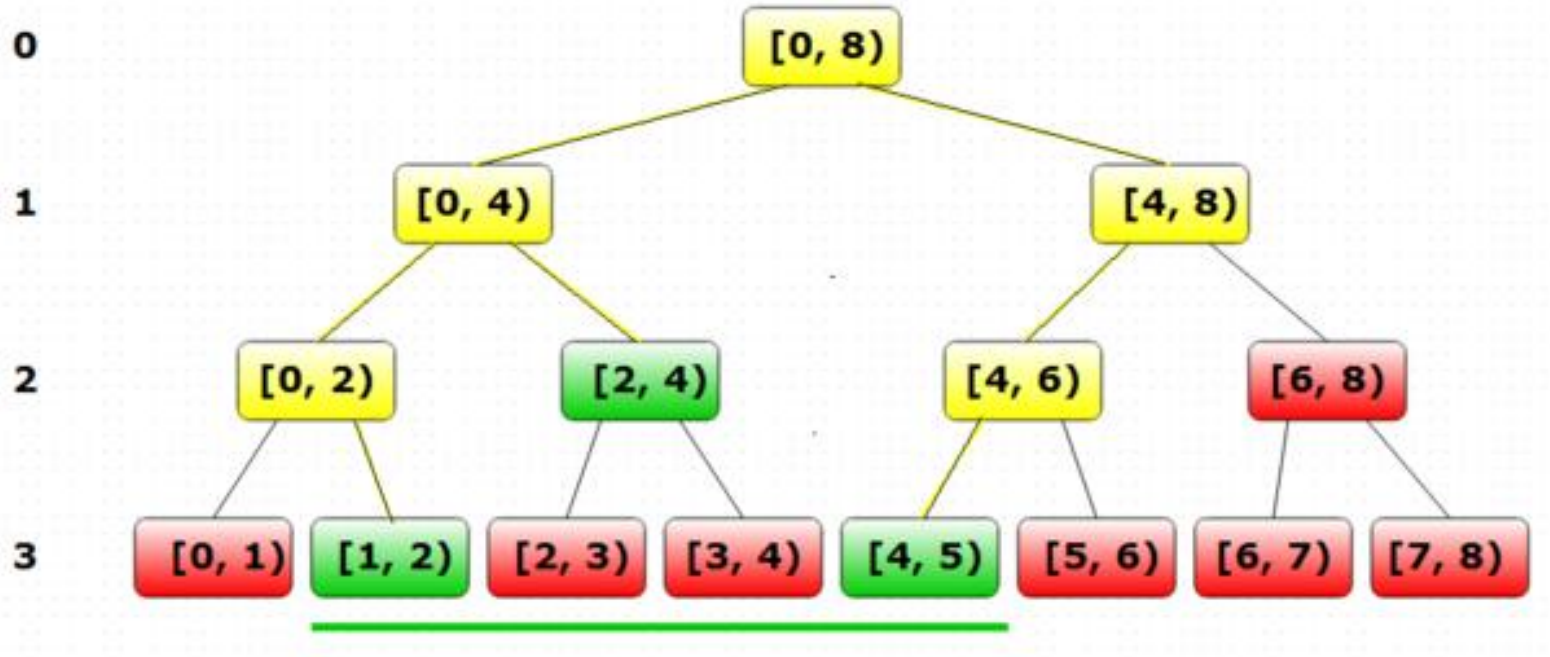
Отрезок [1; 4]

Отрезок [1; 4] (полуинтервал [1; 5))

Отрезок [1; 4] (полуинтервал [1; 5))

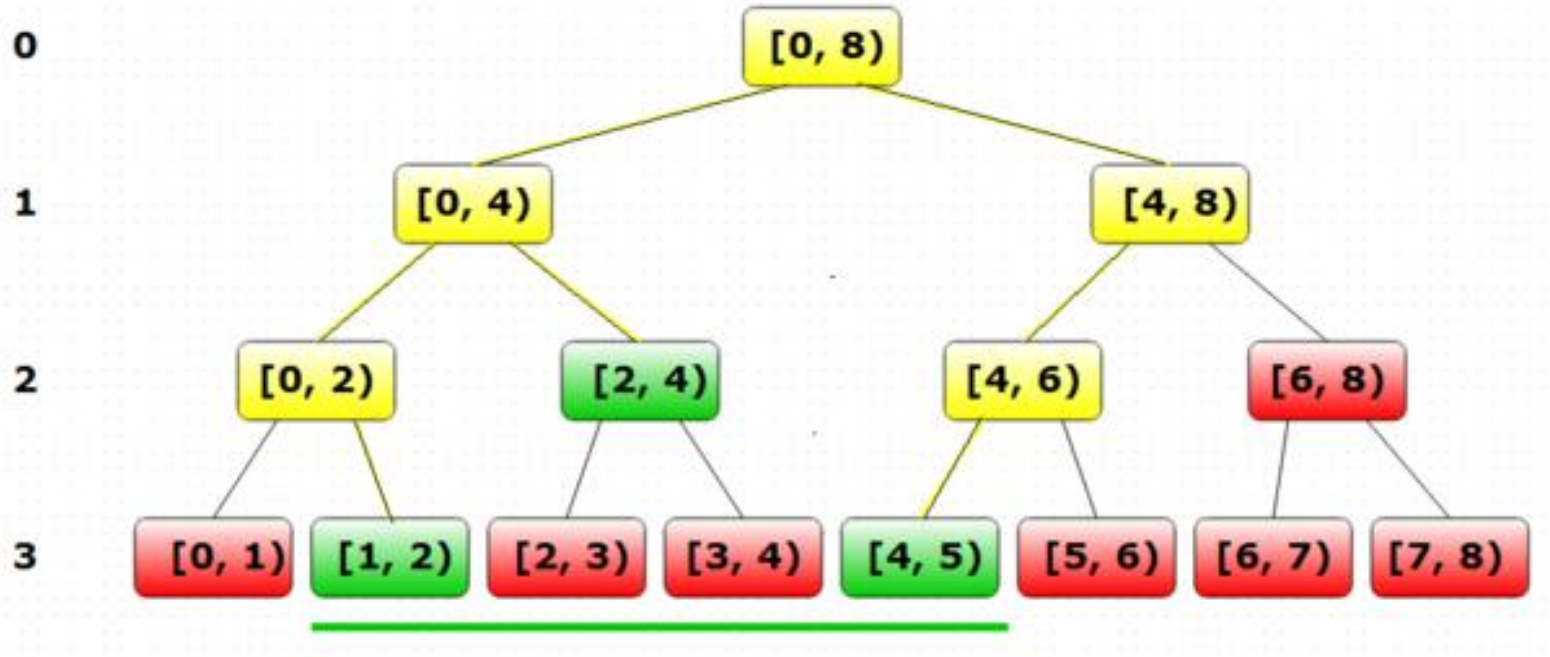


Отрезок [1; 4] (полуинтервал [1; 5))



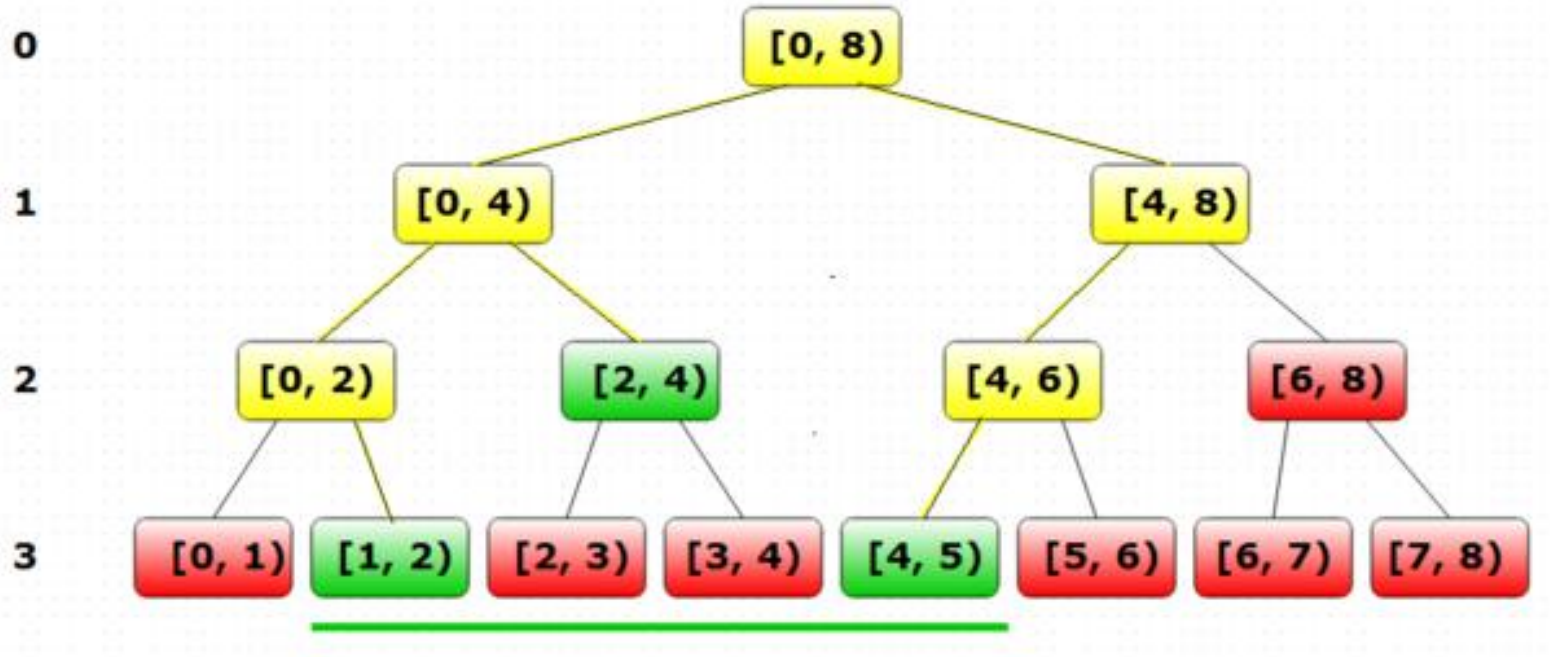
0 слой: [1;5) пересекается с [0;8)

Отрезок [1; 4] (полуинтервал [1; 5))



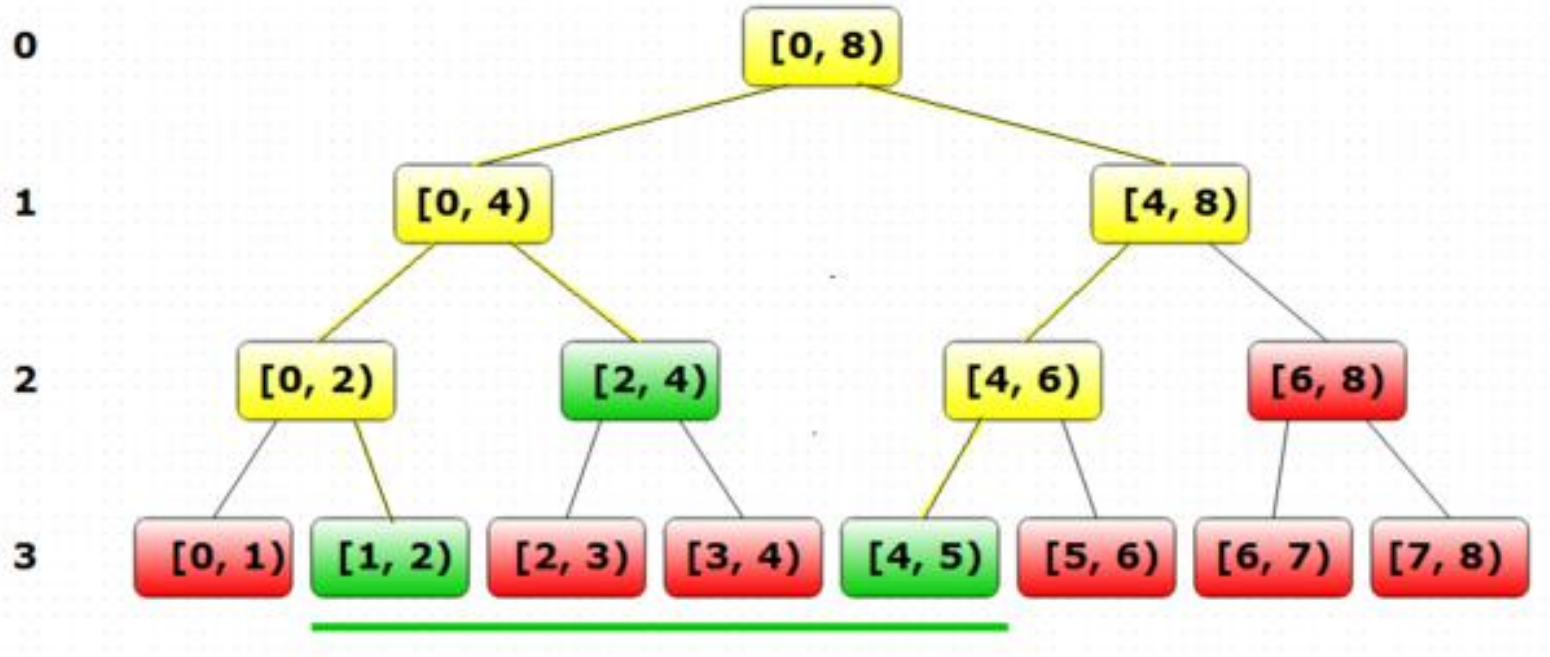
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)

Отрезок [1; 4] (полуинтервал [1; 5))



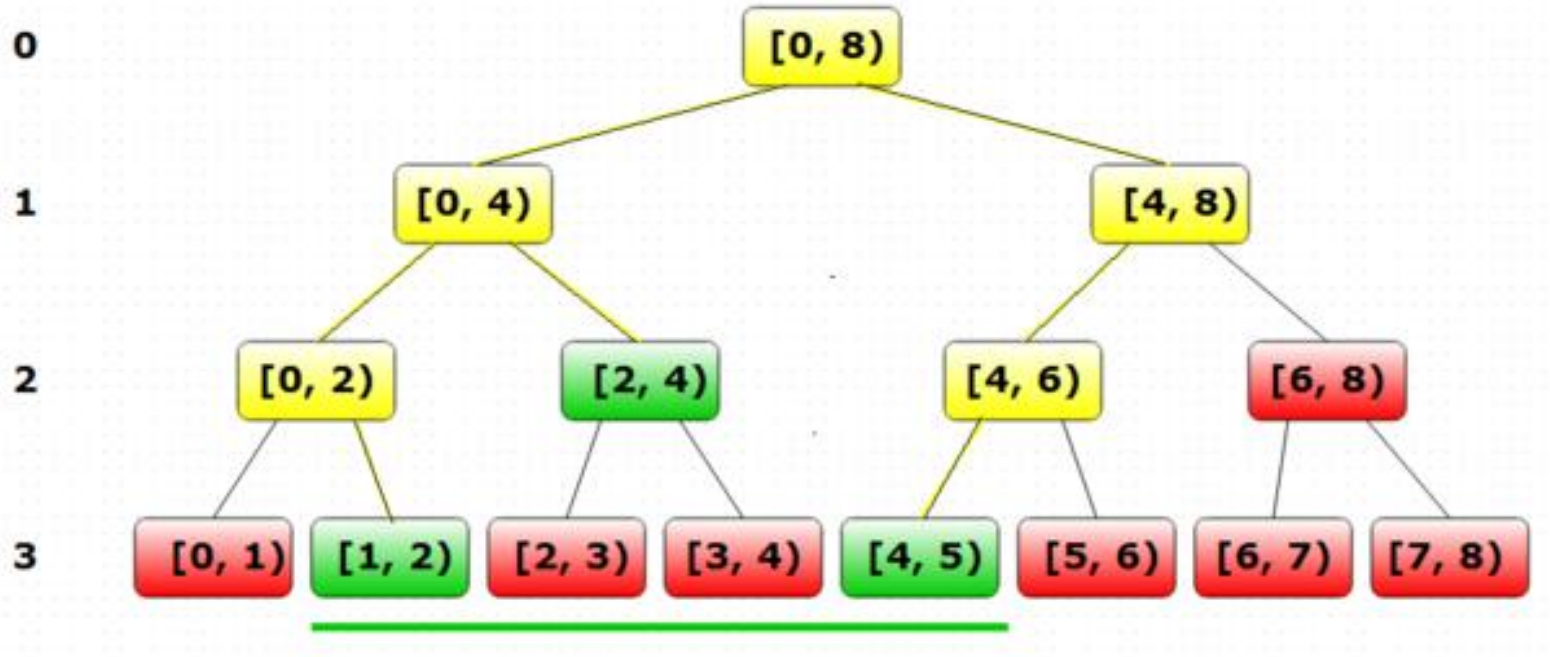
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8)

Отрезок [1; 4] (полуинтервал [1; 5))



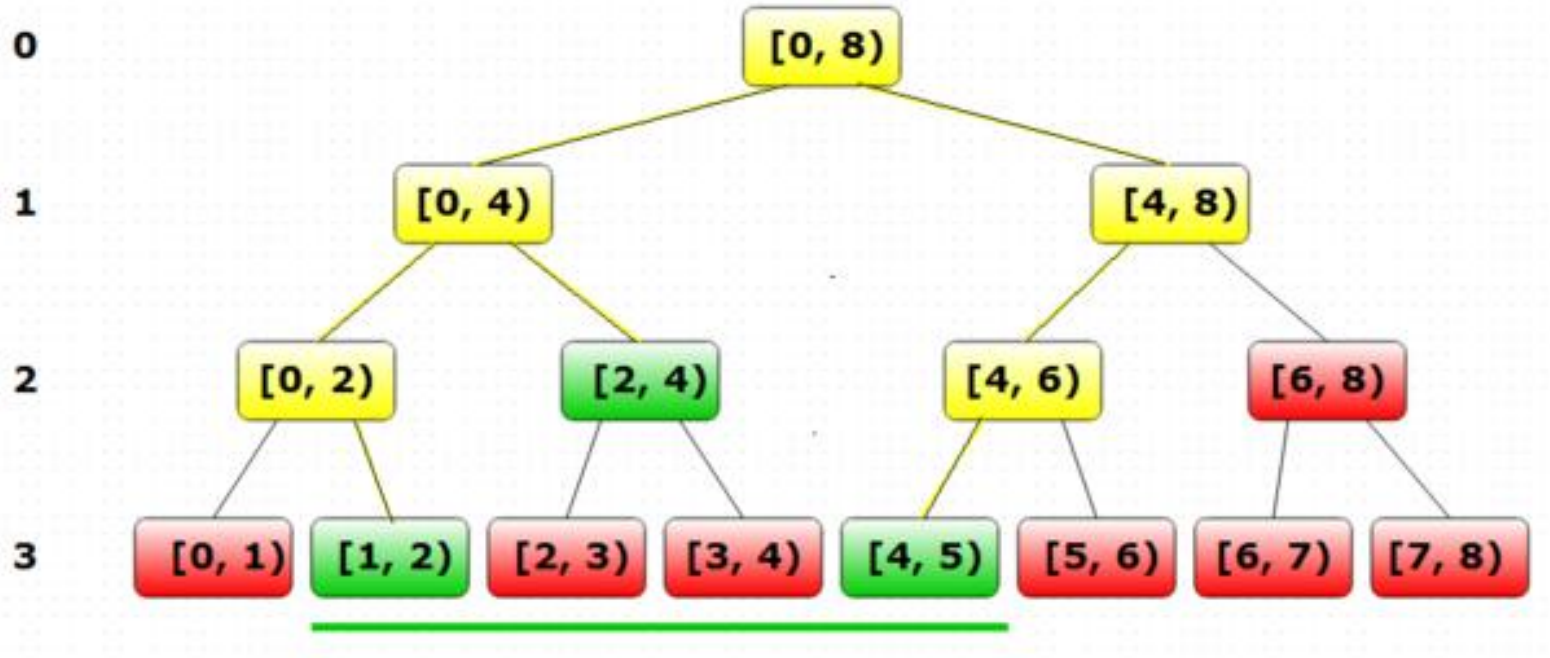
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)

Отрезок [1; 4] (полуинтервал [1; 5))



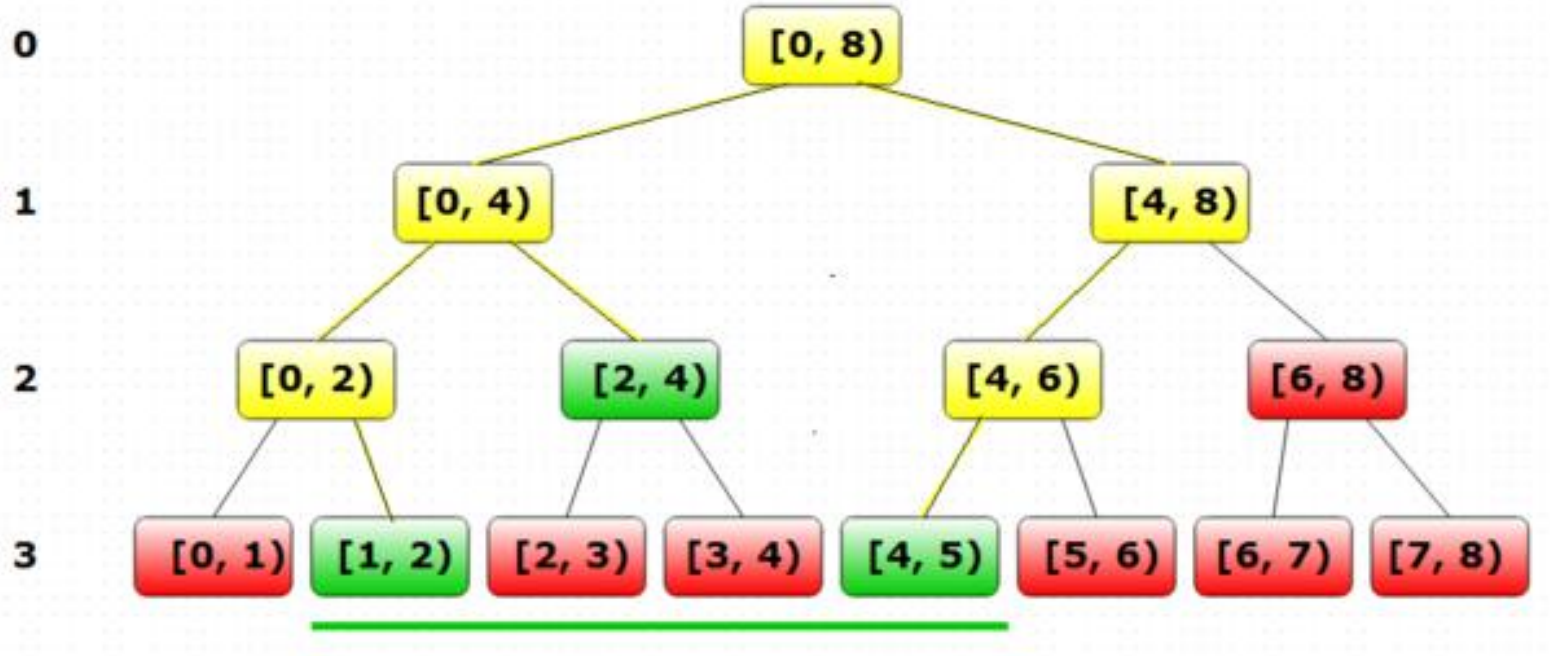
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5)

Отрезок [1; 4] (полуинтервал [1; 5))



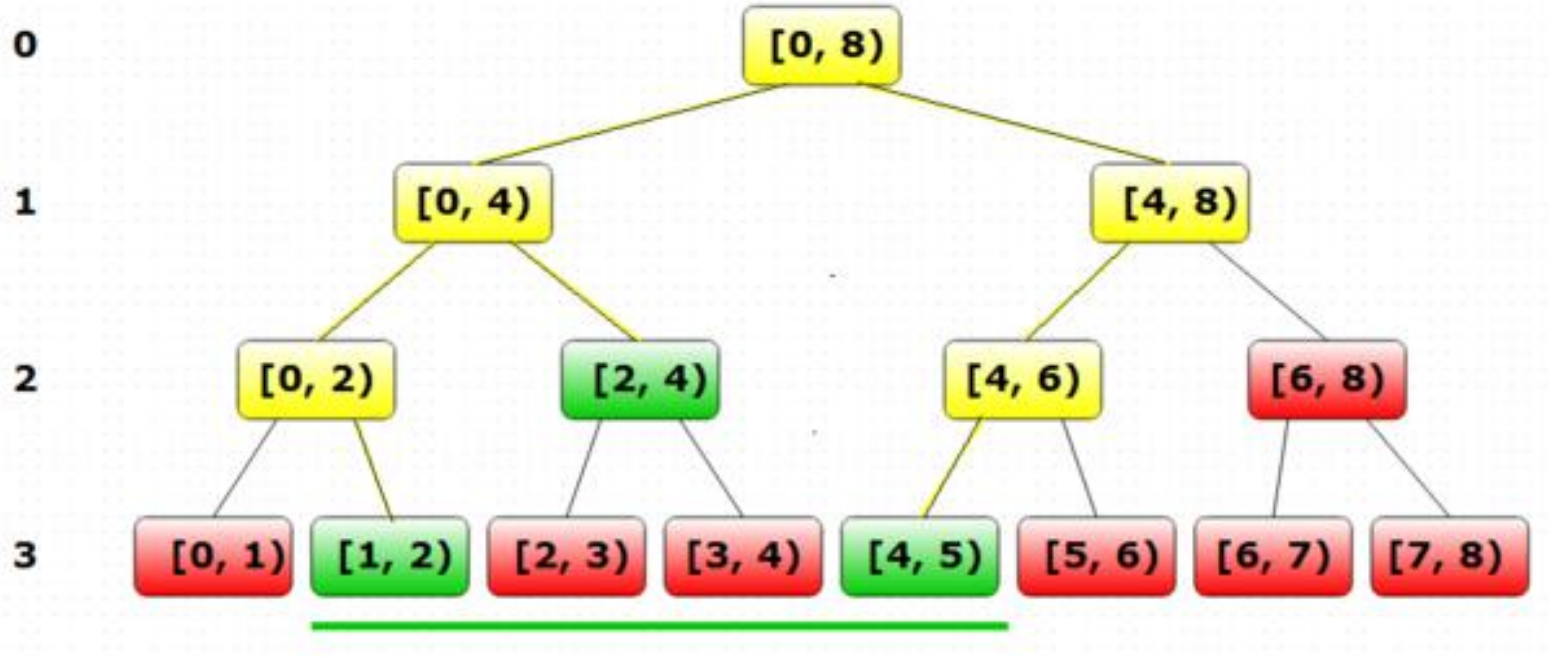
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)

Отрезок [1; 4] (полуинтервал [1; 5))



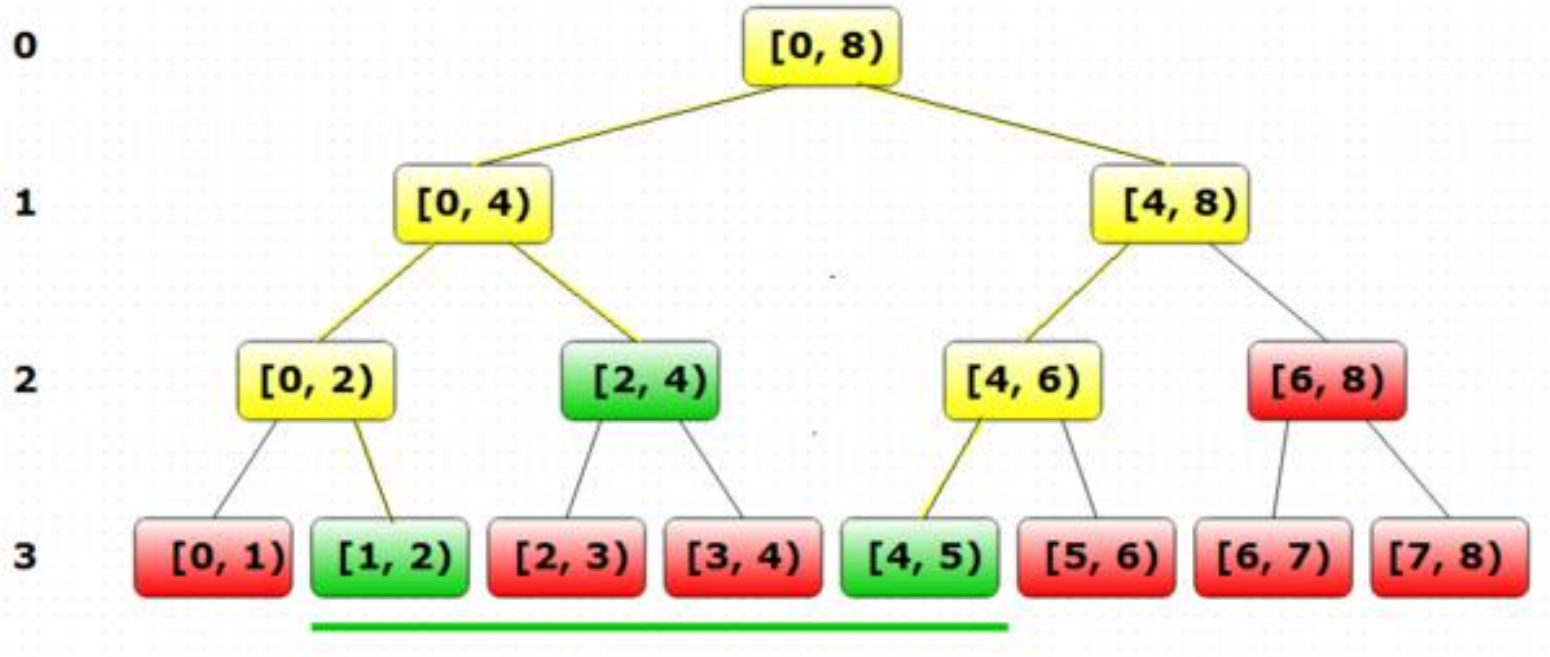
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5)

Отрезок [1; 4] (полуинтервал [1; 5))



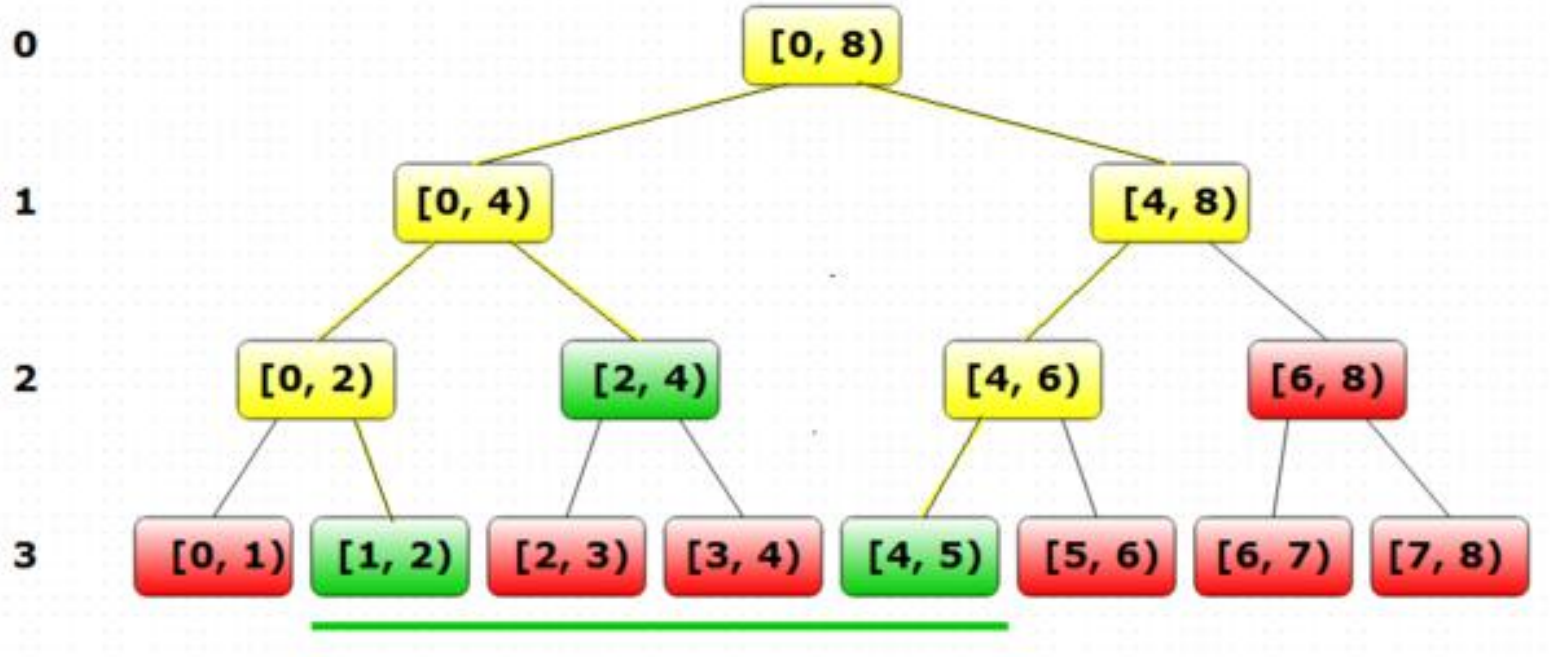
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)

Отрезок [1; 4] (полуинтервал [1; 5))



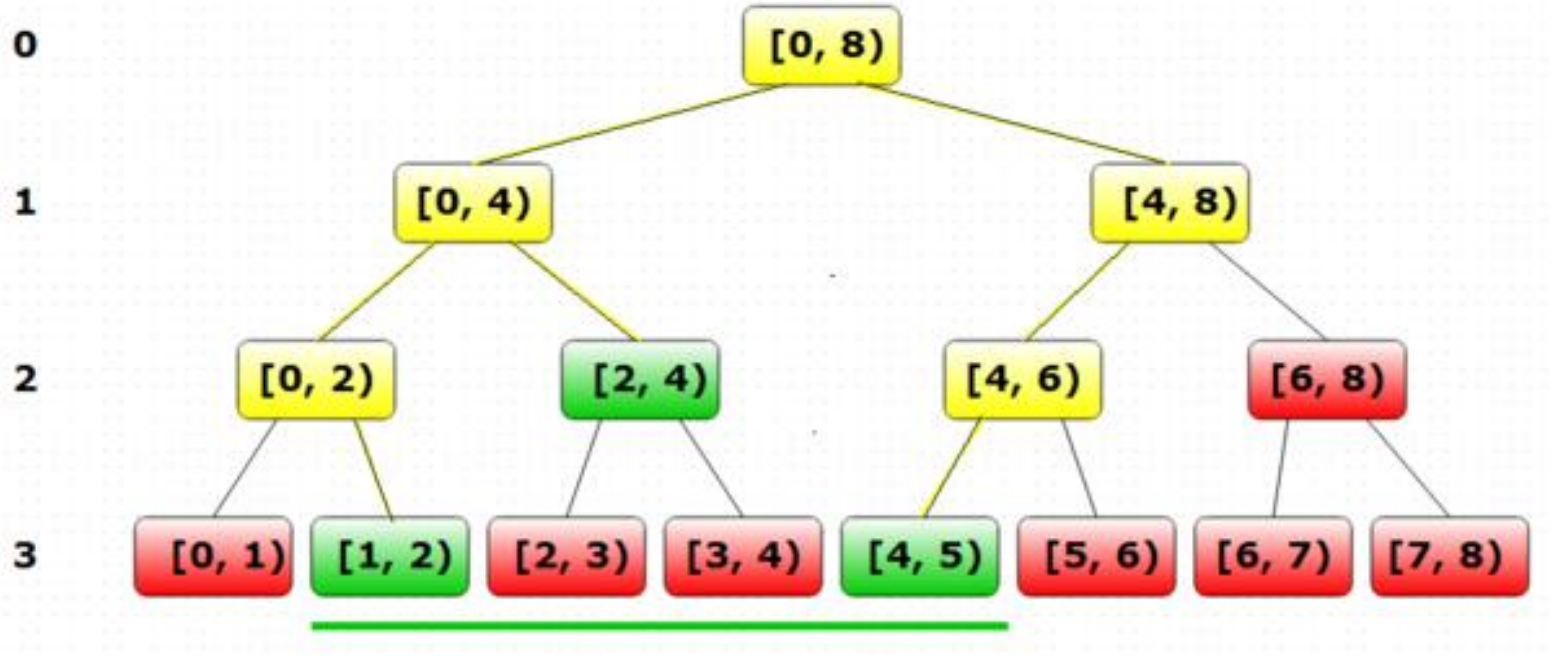
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5)

Отрезок [1; 4] (полуинтервал [1; 5))



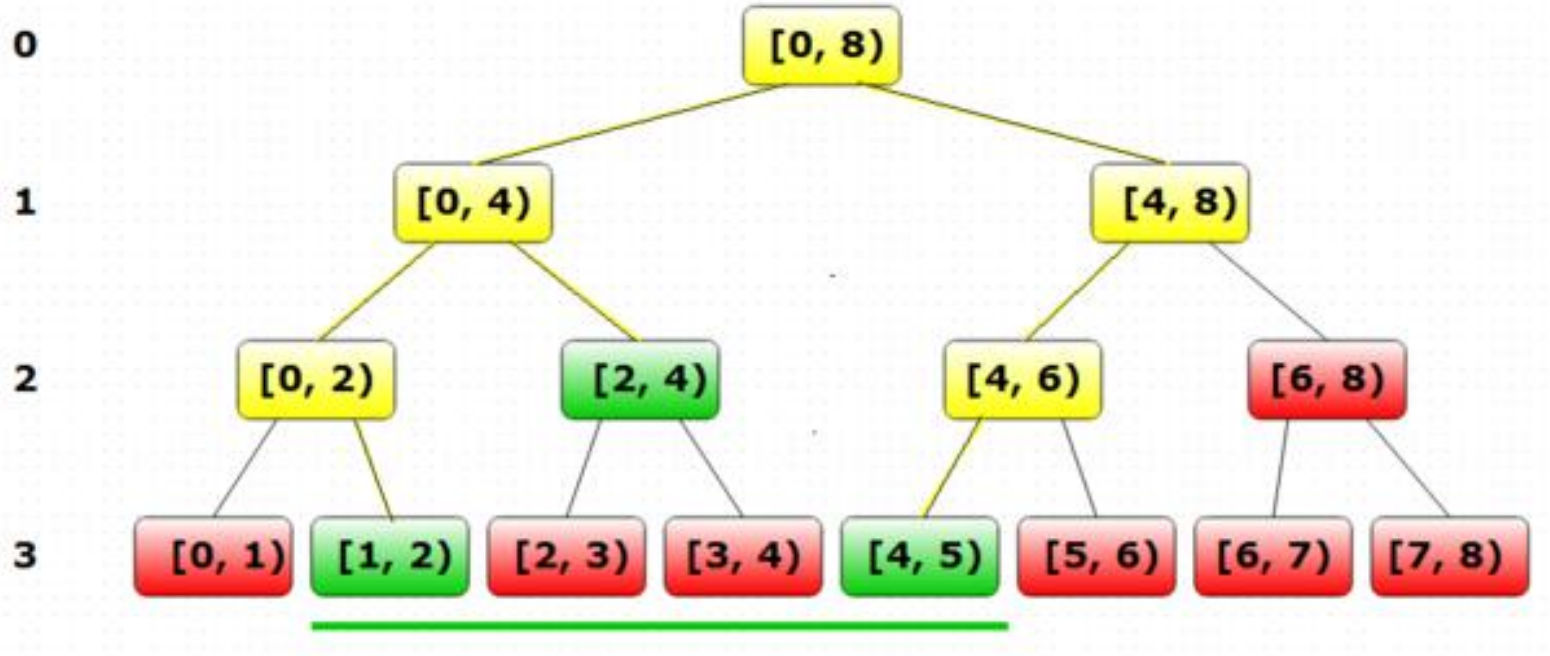
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5) => переходим в листья [4; 5) и [5; 6)

Отрезок [1; 4] (полуинтервал [1; 5))



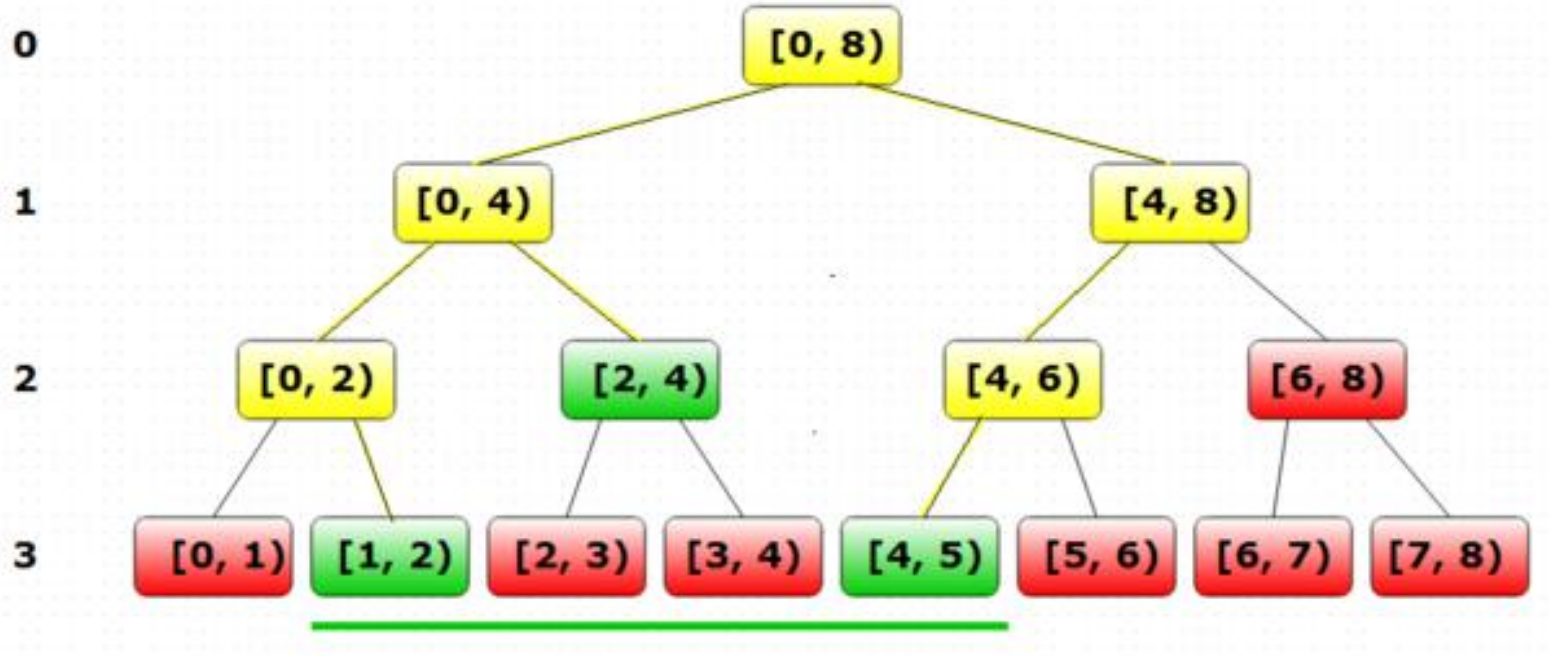
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5) => переходим в листья [4; 5) и [5; 6)

Отрезок [1; 4] (полуинтервал [1; 5))



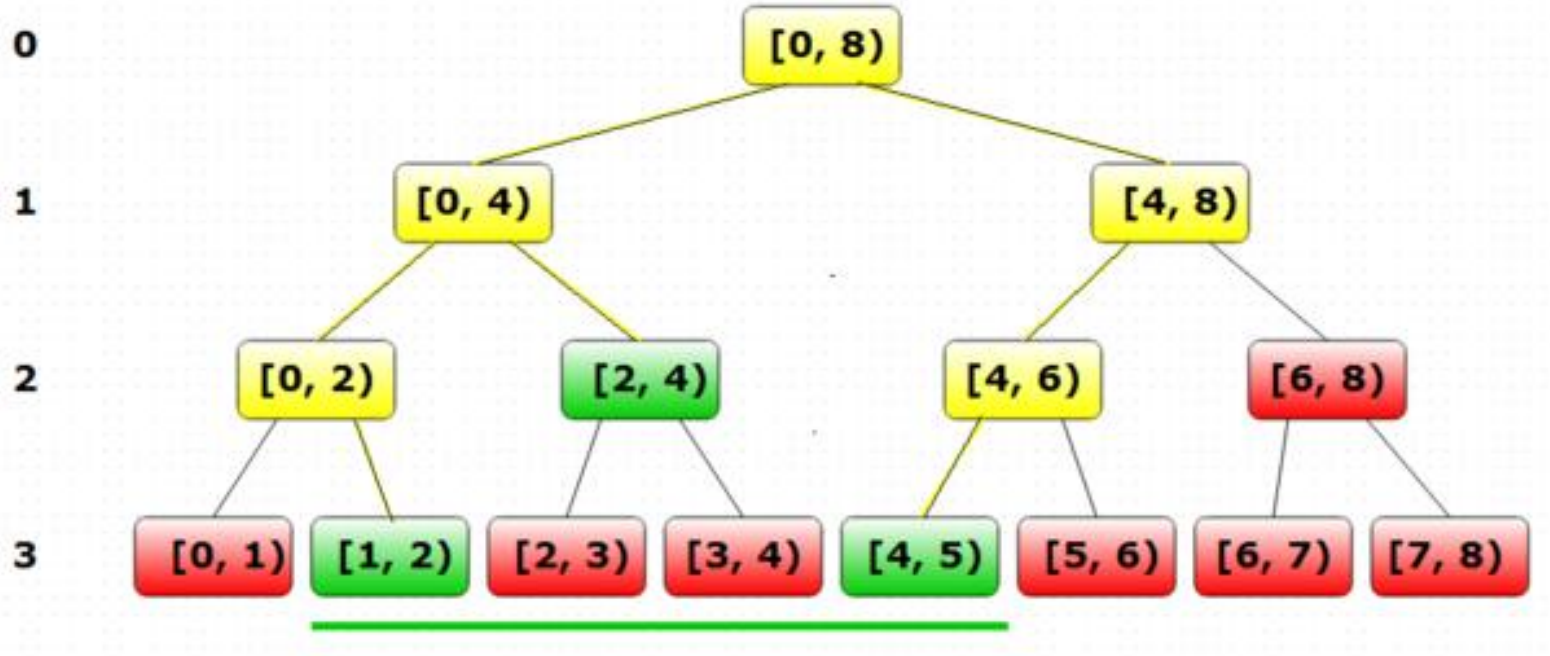
0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5) => переходим в листья [4; 5) и [5; 6)
3 слой: [1; 2), [4; 5) пересекаются с [1; 5)

Отрезок [1; 4] (полуинтервал [1; 5))



0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5) => переходим в листья [4; 5) и [5; 6)
3 слой: [1; 2), [4; 5) пересекаются с [1; 5) => возвращаем значения в этих листьях

Отрезок [1; 4] (полуинтервал [1; 5))



0 слой: [1;5) пересекается с [0;8) => переходим в детей [0; 4) и [4; 8)
1 слой: [1; 5) пересекается с [0; 4) и[4; 8) => переходим в детей [0; 2), [2; 4), [4; 6), [6; 8)
2 слой: [1; 5) пересекается с [0; 2), [2; 4) и [4; 6)
[0; 2) пересекается с [1; 5) => переходим в листья [0; 1), [1; 2)
[2; 4) лежит внутри [1; 5) => возвращаем значение в вершине [2; 4)
[4; 6) пересекается с [1; 5) => переходим в листья [4; 5) и [5; 6)
3 слой: [1; 2), [4; 5) пересекаются с [1; 5) => возвращаем значения в этих листьях
Для «красных» отрезков возвращаем -INF

Поиск максимума

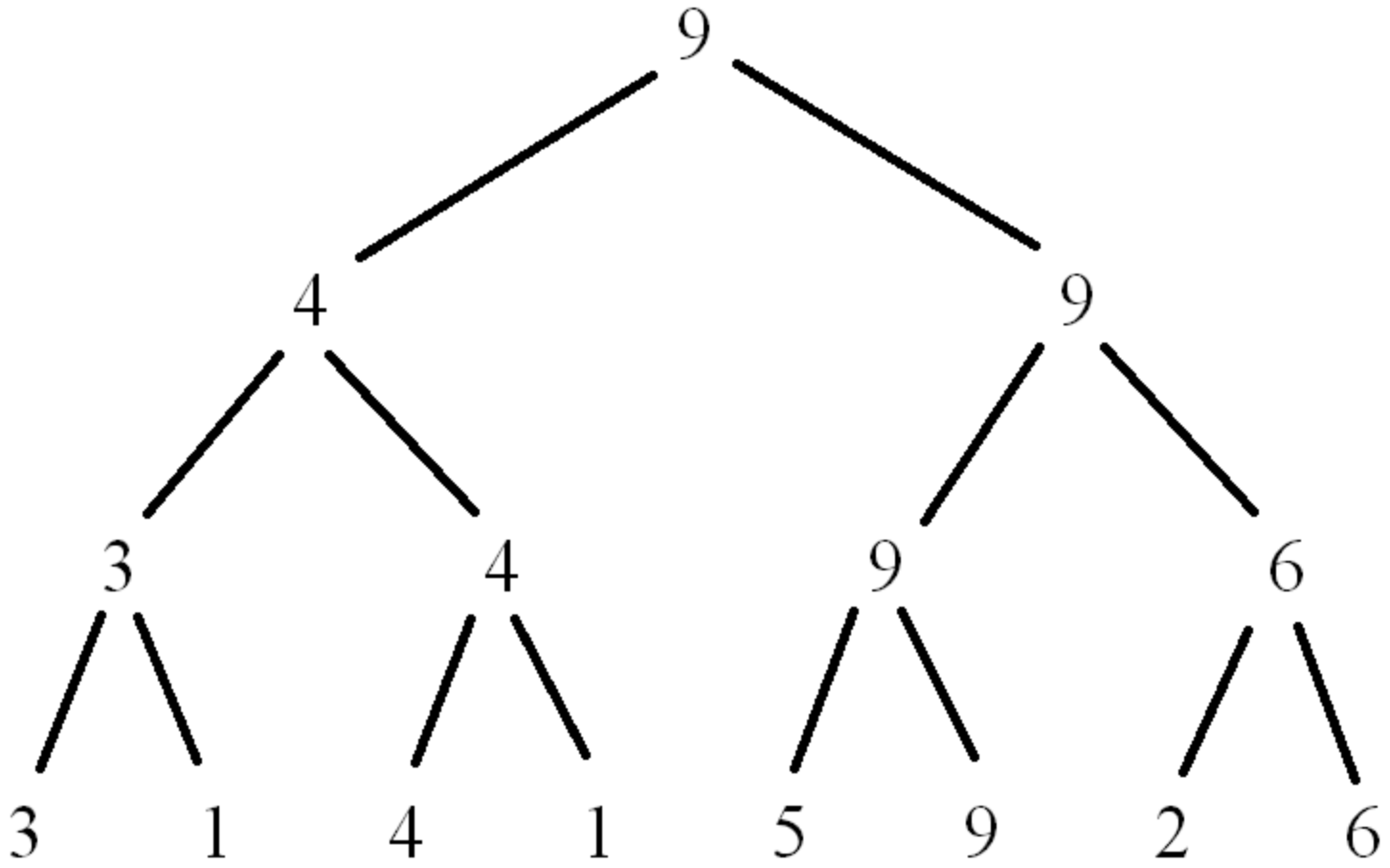
```
int get_max(int v, int l, int r, int a, int b){
    if (b <= l || r <= a)
        return -INF;
    if (a <= l && r <= b){
        return t[v];
    }
    int m = (l + r) / 2;
    int max_l = get_max(2 * v + 1, l, m, a, b);
    int max_r = get_max(2 * v + 2, m, r, a, b);
    return max(max_l, max_r);
}
```

Поиск максимума

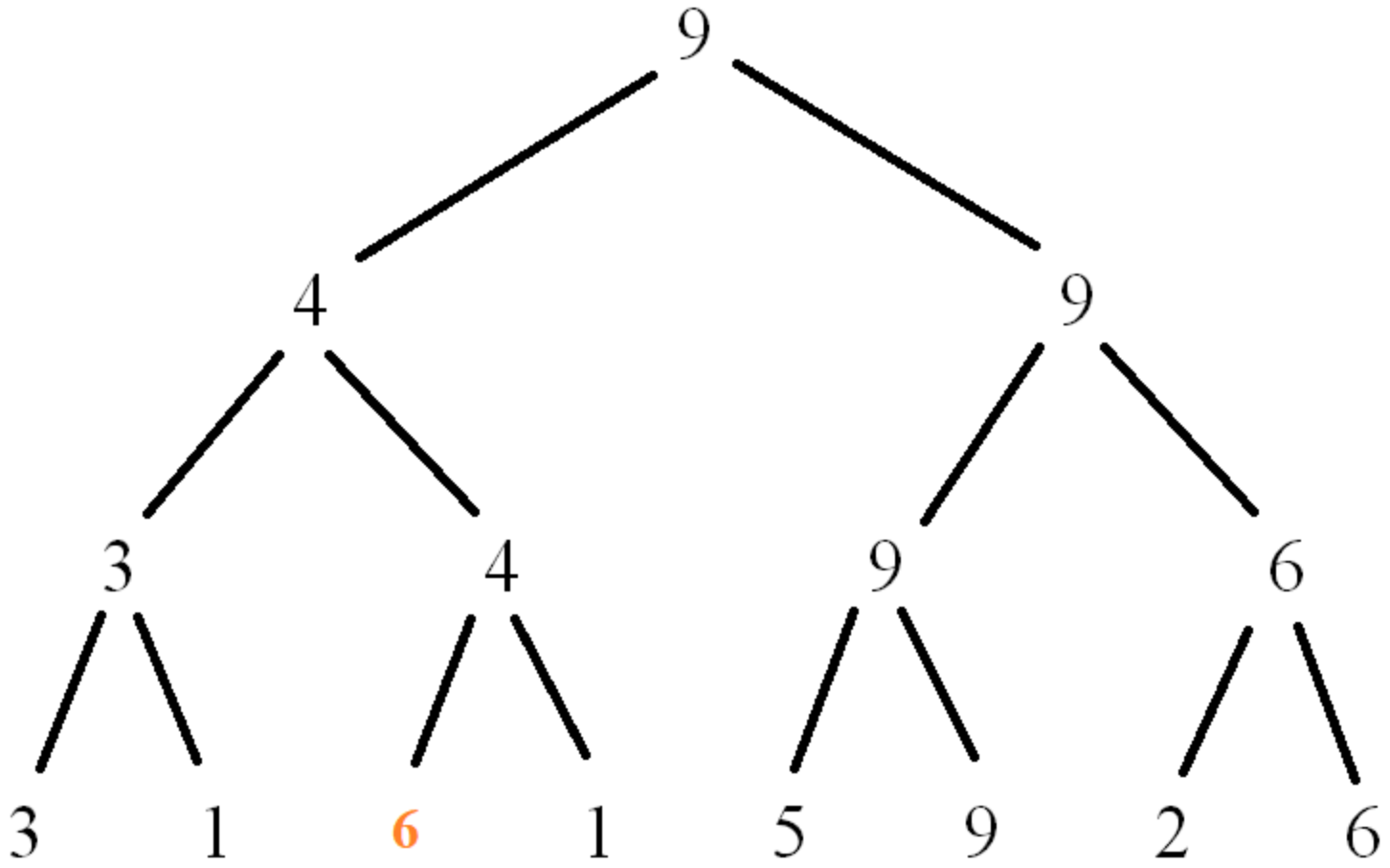
```
int get_max(int v, int l, int r, int a, int b){
    if (b <= l || r <= a)
        return -INF;
    if (a <= l && r <= b){
        return t[v];
    }
    int m = (l + r) / 2;
    int max_l = get_max(2 * v + 1, l, m, a, b);
    int max_r = get_max(2 * v + 2, m, r, a, b);
    return max(max_l, max_r);
}
```

Вызов – get_max(0, 0, n, l, r)

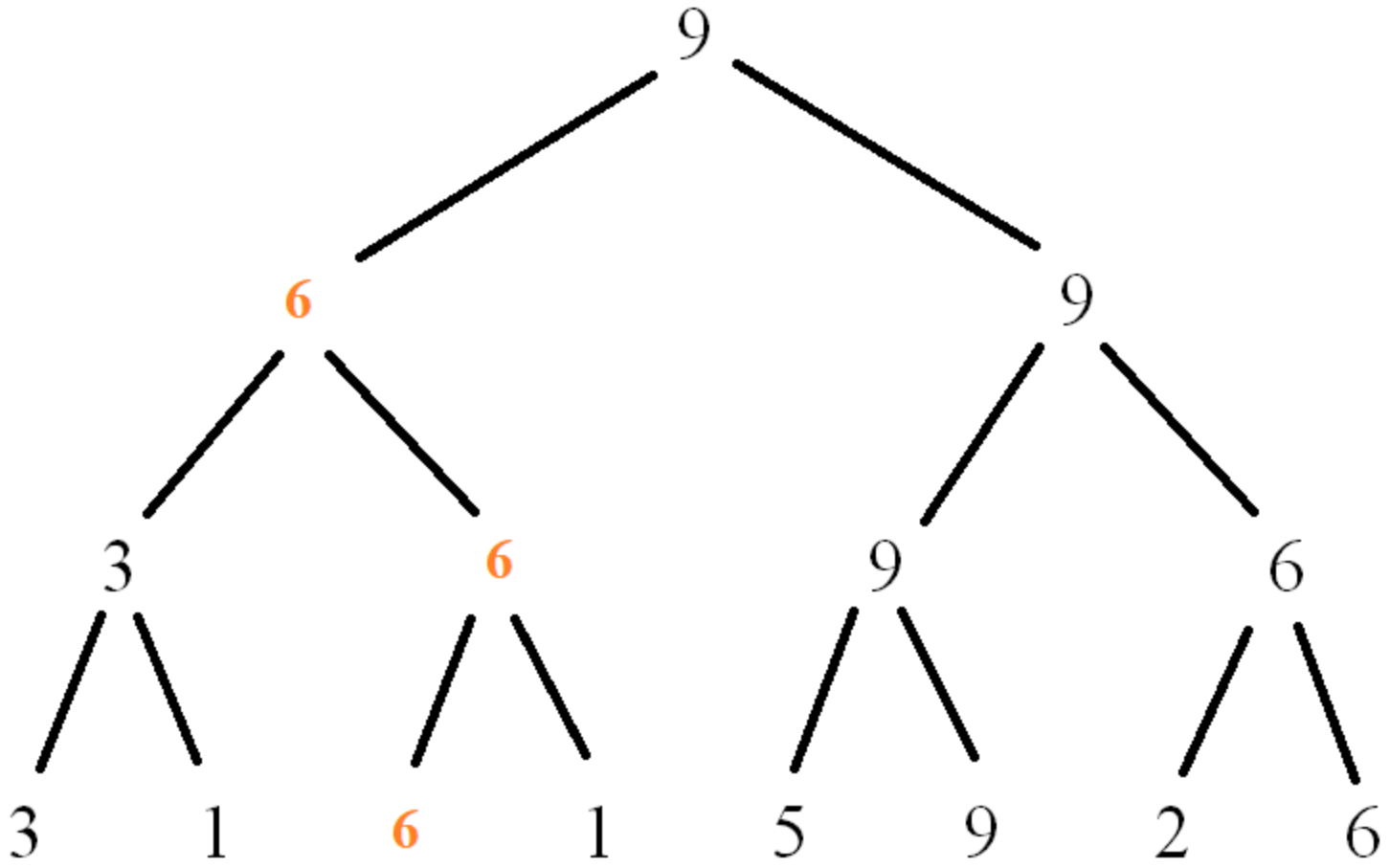
Изменение элемента



Изменение элемента



Изменение элемента



Алгоритм

- Находим лист, которому соответствует данный элемент

Алгоритм

- Находим лист, которому соответствует данный элемент
- Проталкиваем информацию предкам

Изменение элемента

```
void update(int v, int l, int r, int pos, int val){
    if (pos < l || pos >= r){
        return ;
    }
    if (r - l == 1){
        t[v] = val;
    }
    else{
        int m = (l + r) / 2;
        update(2 * v + 1, l, m, pos, val);
        update(2 * v + 2, m, r, pos, val);
        t[v] = max(t[2 * v + 1], t[2 * v + 2]);
    }
}
```

Изменение элемента

```
void update(int v, int l, int r, int pos, int val){
    if (pos < l || pos >= r){
        return ;
    }
    if (r - l == 1){
        t[v] = val;
    }
    else{
        int m = (l + r) / 2;
        update(2 * v + 1, l, m, pos, val);
        update(2 * v + 2, m, r, pos, val);
        t[v] = max(t[2 * v + 1], t[2 * v + 2]);
    }
}
```

Вызов – update(0, 0, n, pos, val)

Поиск k-го нуля

Дан массив из N чисел. Найти позицию k-го нуля.

Пример:

1 0 2 0 3 0

3-й ноль – 5

4-й ноль – -1

Как решать?

Построим новый массив, в котором на i-й позиции будет стоять 1, если a[i] = 0, и 0 в противном случае.

Как решать?

Построим новый массив, в котором на i-й позиции будет стоять 1, если a[i] = 0, и 0 в противном случае.

1 0 2 0 3 0 -> 0 1 0 1 0 1

Как решать?

Построим новый массив, в котором на i-й позиции будет стоять 1, если a[i] = 0, и 0 в противном случае.

1 0 2 0 3 0 -> 0 1 0 1 0 1

Тогда общее количество нулей в массиве – суммарное число единиц.

Как решать?

Построим новый массив, в котором на i-й позиции будет стоять 1, если a[i] = 0, и 0 в противном случае.

1 0 2 0 3 0 -> 0 1 0 1 0 1

Тогда общее количество нулей в массиве – суммарное число единиц.

Составим дерево отрезков для сумм на изменённом массиве.

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице.

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице. Спустимся к нему, используя информацию в вершинах дерева.

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице. Спустимся к нему, используя информацию в вершинах дерева. Для этого будем сравнивать k со значением в левом сыне (суммарное число нулей в нём).

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице. Спустимся к нему, используя информацию в вершинах дерева. Для этого будем сравнивать k со значением в левом сыне (суммарное число нулей в нём).

1) $k \le t[2 * v + 1]$ => k-й ноль в левом сыне

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице. Спустимся к нему, используя информацию в вершинах дерева. Для этого будем сравнивать k со значением в левом сыне (суммарное число нулей в нём).

1) $k \le t[2 * v + 1]$ => k-й ноль в левом сыне
2) $k > t[2 * v + 1]$ => k-й ноль в правом сыне

Как искать k-й ноль?

Найдём k-й по счёту лист, который равен единице. Спустимся к нему, используя информацию в вершинах дерева. Для этого будем сравнивать k со значением в левом сыне (суммарное число нулей в нём).

1) $k \le t[2 * v + 1]$ => k-й ноль в левом сыне
2) $k > t[2 * v + 1]$ => k-й ноль в правом сыне, причем там он $(k - t[2 * v + 1])$-й ноль

Реализация

```
vector <int> t(4 * MAXN), a;
void build(int v, int l, int r){
    if (r - l == 1){
        t[v] = (a[l] == 0);
    }
    else{
        int m = (l + r) / 2;
        build(2 * v + 1, l, m);
        build(2 * v + 2, m, r);
        t[v] = t[2 * v + 1] + t[2 * v + 2];
    }
}
```

Реализация изменения значения

```
void update(int v, int l, int r, int pos, int val){
    if (pos < l || pos >= r){
        return ;
    }
    if (r - l == 1){
        t[v] = (val == 0);
    }
    else{
        int m = (l + r) / 2;
        update(2 * v + 1, l, m, pos, val);
        update(2 * v + 2, m, r, pos, val);
        t[v] = t[2 * v + 1] + t[2 * v + 2];
    }
}
```

Поиск k-го нуля

```
int get_kth(int v, int l, int r, int k){
    if (k > t[v]){
        return -1;
    }
    if (r - l == 1){
        return l;
    }
    int m = (l + r) / 2;
    if (k <= t[v * 2 + 1]){
        return get_kth(2 * v + 1, l, m, k);
    }
    else { // k > t[v * 2 + 2]
        return get_kth(2 * v + 2, m, r, k - t[v * 2 + 1]);
    }
}
```

Поиск k-го нуля

```
int get_kth(int v, int l, int r, int k){
    if (k > t[v]){
        return -1;
    }
    if (r - l == 1){
        return l;
    }
    int m = (l + r) / 2;
    if (k <= t[v * 2 + 1]){
        return get_kth(2 * v + 1, l, m, k);
    }
    else { // k > t[v * 2 + 2]
        return get_kth(2 * v + 2, m, r, k - t[v * 2 + 1]);
    }
}
```

Вызов – get_kth(0, 0, n, k)