Lab8 Solution

YAO ZHAO

Lab8.A: Funny Fluffy Tuzi

- Given a finite sequence s of length 2^N of equally-spaced samples of a function, find the result of its Discrete Fourier Transform.

Sample 1 Input

2
0 1 0 -1

$x_0 = 0$
$x_1 = 1$
$x_2 = 0$
$x_3 = -1$

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn}$$

$$X_0 = x_0 * e^{-\frac{2\pi i}{4}*0*0} + x_1 * e^{-\frac{2\pi i}{4}*0*1} + x_2 * e^{-\frac{2\pi i}{4}*0*2} + x_3 * e^{-\frac{2\pi i}{4}*0*3}$$

$$X_1 = x_0 * e^{-\frac{2\pi i}{4}*1*0} + x_1 * e^{-\frac{2\pi i}{4}*1*1} + x_2 * e^{-\frac{2\pi i}{4}*1*2} + x_3 * e^{-\frac{2\pi i}{4}*1*3}$$

$$X_2 = x_0 * e^{-\frac{2\pi i}{4}*2*0} + x_1 * e^{-\frac{2\pi i}{4}*2*1} + x_2 * e^{-\frac{2\pi i}{4}*2*2} + x_3 * e^{-\frac{2\pi i}{4}*2*3}$$

$$X_3 = x_0 * e^{-\frac{2\pi i}{4}*3*0} + x_1 * e^{-\frac{2\pi i}{4}*3*1} + x_2 * e^{-\frac{2\pi i}{4}*3*2} + x_3 * e^{-\frac{2\pi i}{4}*3*3}$$

$X_0 = 0 \quad X_1 = -2i \quad X_2 = 0 \quad X_3 = 2i$

magnitude of complex number
$X_i = a+bi, \quad z_i = \sqrt{a^2 + b^2}$

$z_0 = 0 \quad z_1 = 2 \quad z_2 = 0 \quad z_3 = 2$

Sample 1 Output

0 2 0 2

For more details, please refer to : FFT Supplementary Instruction.pdf

Lab8.B: Too easy

- Given an array $a_1, a_2, \dots, a_N$, find the median of them.
- The median of an array of length L is defined as the $\left\lfloor \frac{L+1}{2} \right\rfloor^{th}$ entry in its non-decreasing sorted version (the array is $1 - indexed$).

  "No." Satori thought, "This is too easy. No."
- She has Q queries (L_i, R_i) for you. For each query (L_i, R_i), she wants to know the median of $a_{L_i}, a_{L_i+1}, \dots, a_{R_i}$.

Sample Input

10 10
10 18 4 9 12 18 6 8 6 16
4 5
3 10
5 5
6 10
7 10
1 10
1 2
2 4
4 6
2 9

***Q*1:**

$\left\lfloor \frac{2+1}{2} \right\rfloor = 1$

non-decreasing: 9 12 → **9**

1	2	3	4	5	6	7	8	9	10
10	18	4	9	12	18	6	8	6	16

***Q*2:**

$\left\lfloor \frac{8+1}{2} \right\rfloor = 4$

non-decreasing: 4 6 6 8 9 12 16 18 → **8**

1	2	3	4	5	6	7	8	9	10
10	18	4	9	12	18	6	8	6	16

Sample Input

10 10
10 18 4 9 12 18 6 8 6 16
4 5
3 10
5 5
6 10
7 10
1 10
1 2
2 4
4 6
2 9

…

Q6:

$\left\lfloor \frac{10+1}{2} \right\rfloor = 5$

non-decreasing: 4 6 6 8 9 10 12 16 18 18 → **9**

1	2	3	4	5	6	7	8	9	10
10	18	4	9	12	18	6	8	6	16

…

Q10:

$\left\lfloor \frac{8+1}{2} \right\rfloor = 4$

non-decreasing: 4 6 6 8 9 12 18 18 → **8**

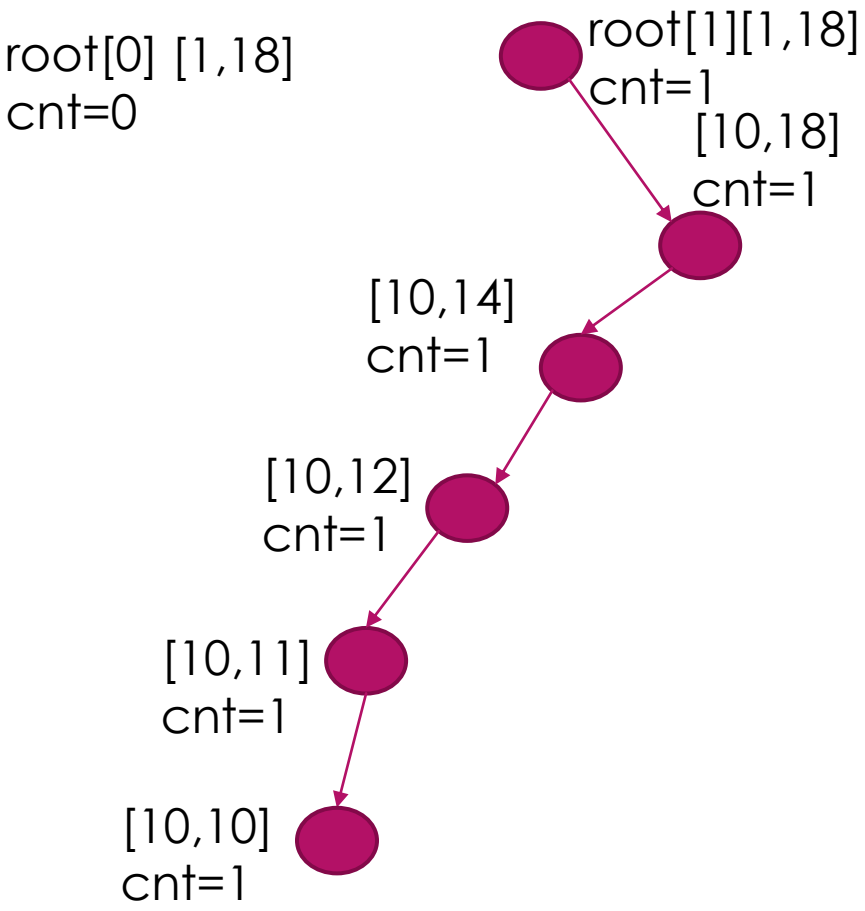
1	2	3	4	5	6	7	8	9	10
10	18	4	9	12	18	6	8	6	16

Sample Input	Sample Output
10 10	
10 18 4 9 12 18 6 8 6 16	
4 5	**9**
3 10	**8**
5 5	**12**
6 10	**8**
7 10	**6**
1 10	**9**
1 2	**10**
2 4	**9**
4 6	**12**
2 9	**8**

1	2	3	4
10	18	4	9

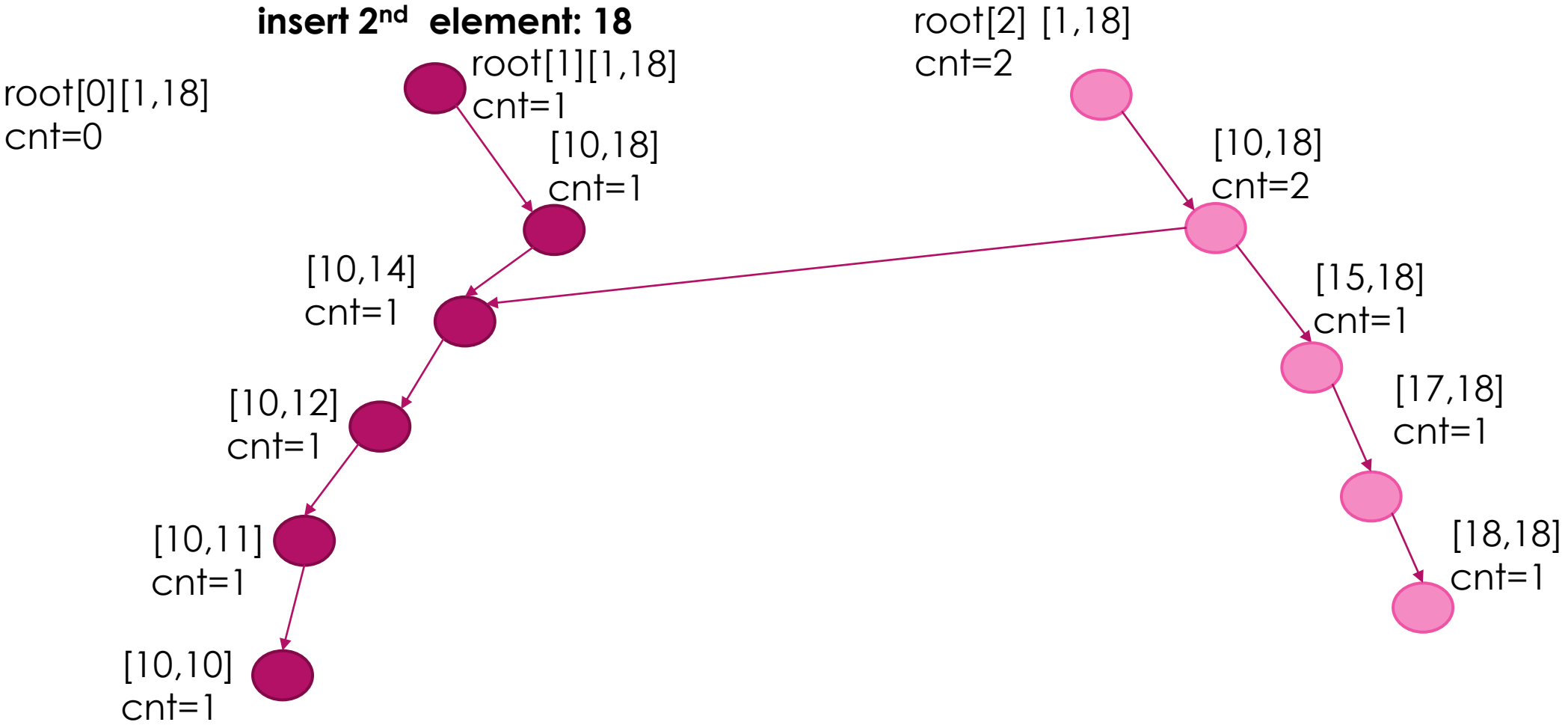
ai range: 1 ~18

insert 1st element: 10



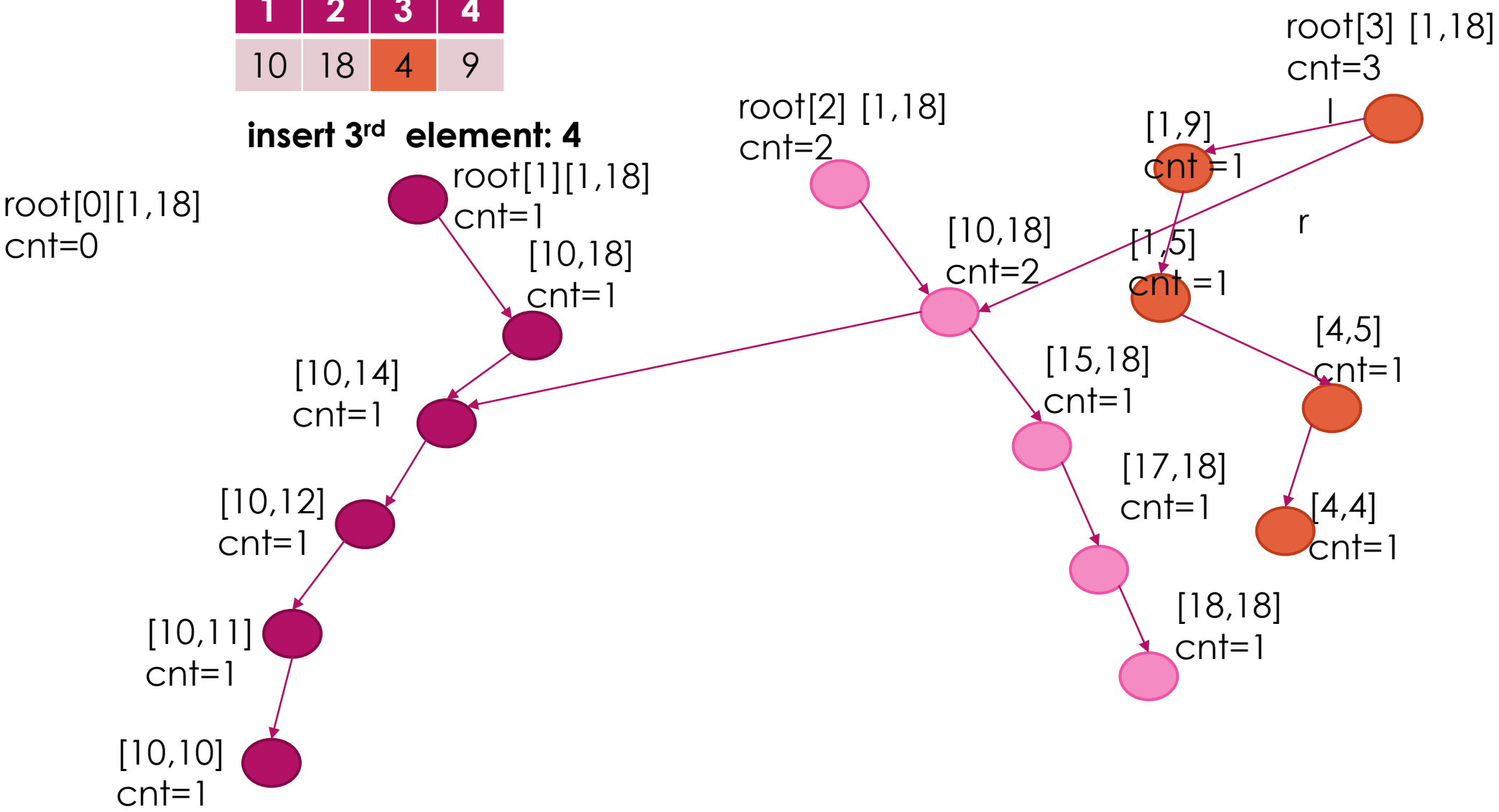
1	2	3	4
10	18	4	9

insert 2nd element: 18



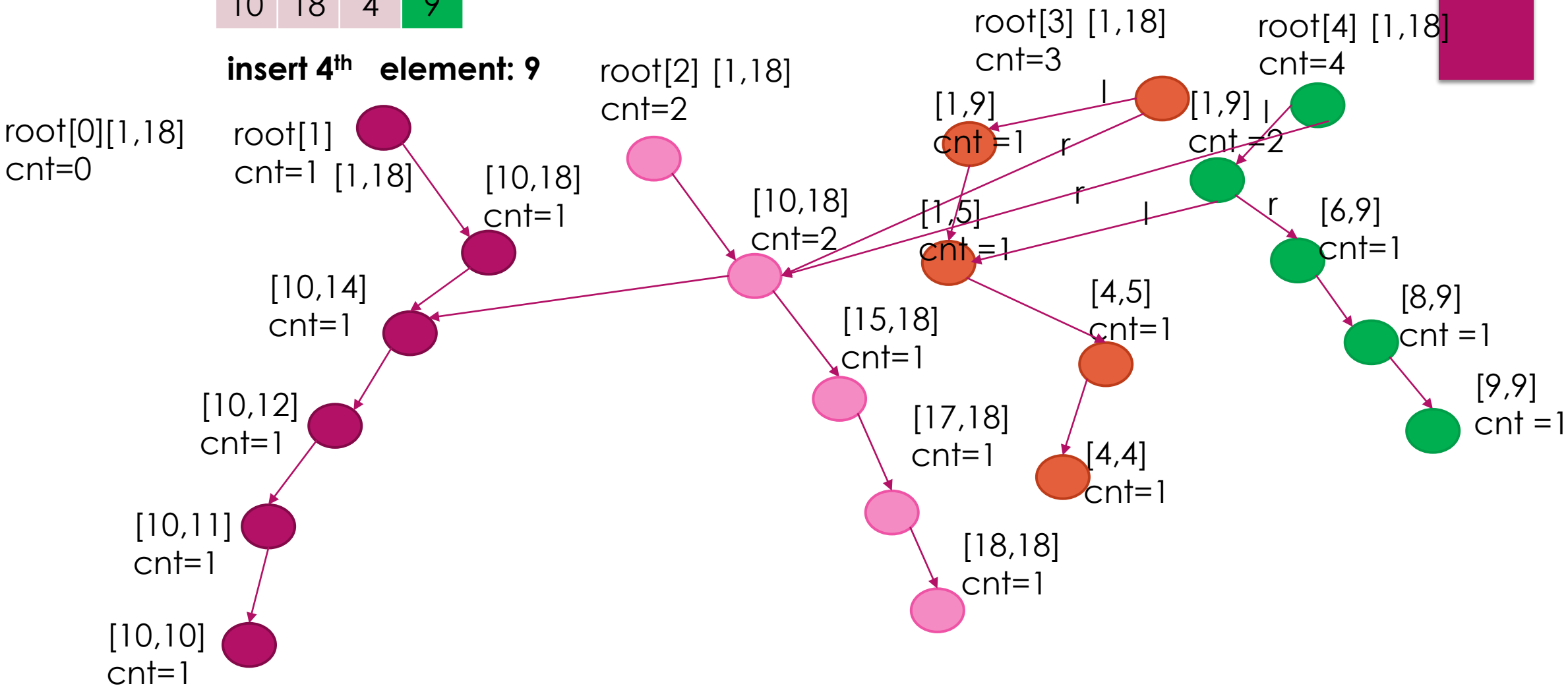
1	2	3	4
10	18	4	9

insert 3rd element: 4



1	2	3	4
10	18	4	9

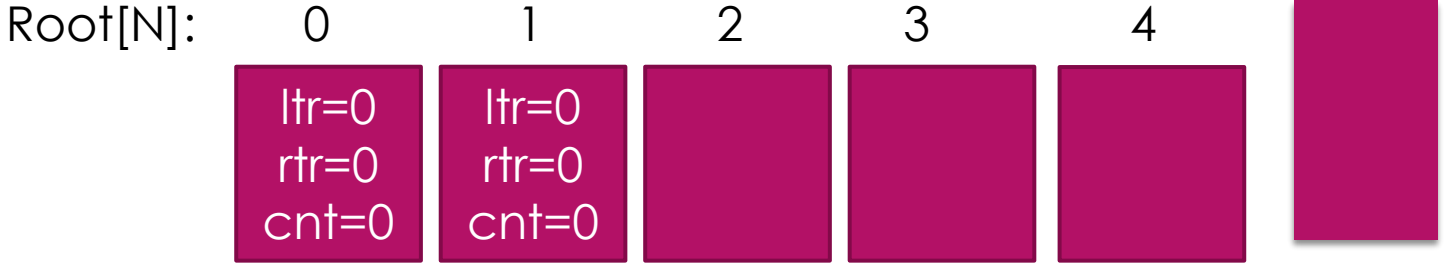
insert 4th element: 9

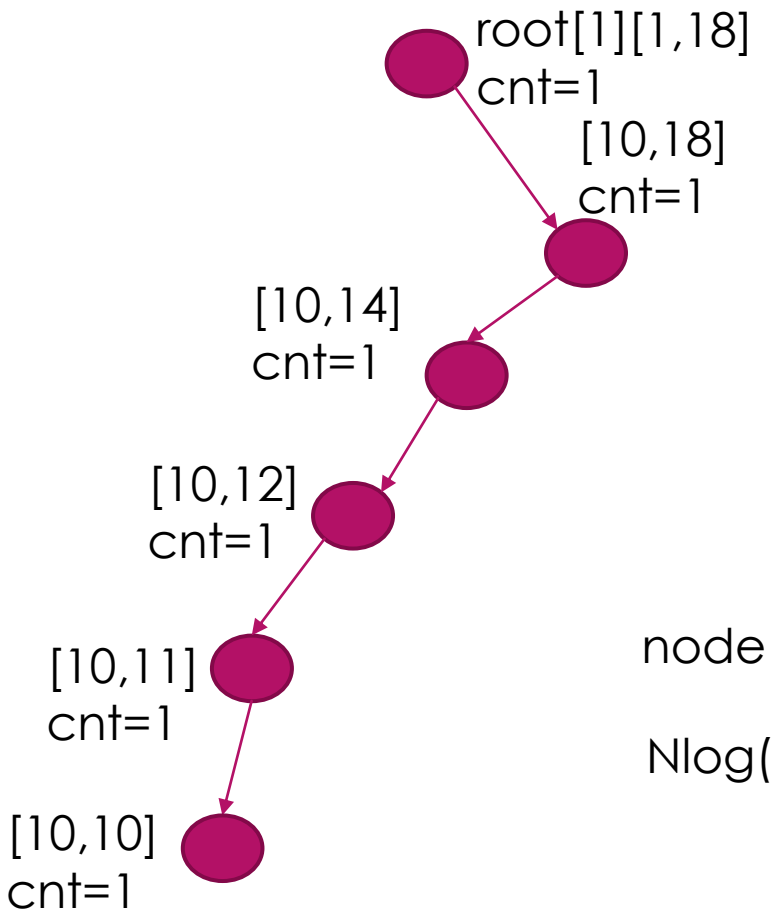


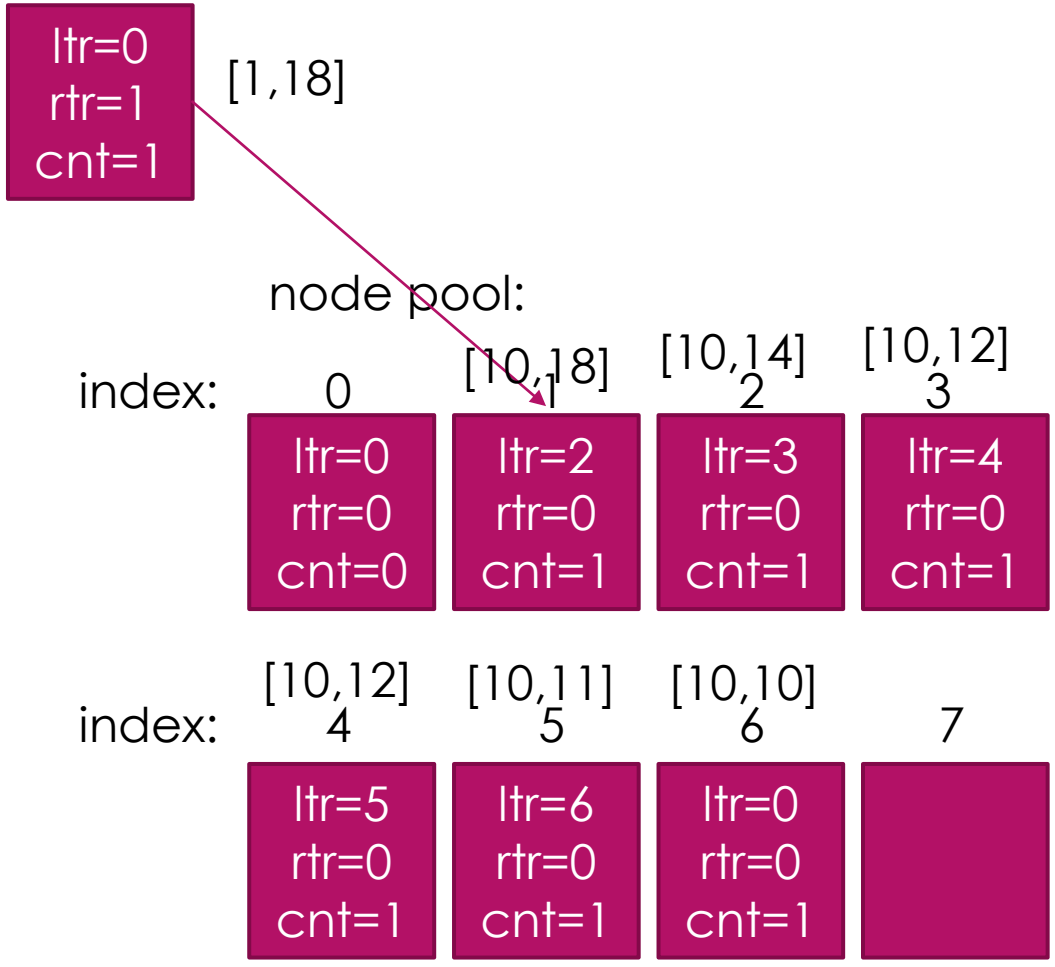
```
initial root[0] let l_tree = 0 r_tree = 0 cnt=0
for 1 to N:
        build(root[i], root[i-1], low_bound, high_bound, a[i])

void  build(Node new_node, Node pre_node low_bound, high_bound, value){
     copy pre_node to new_node
     new_node.cnt++
     if (low_bound = high_bound) return

     int mid = ⌊(low_bound + high_bound)/2⌋

     if (value ≤ mid)
              new_node.l_tree = apply a new node from node pool
              build(new_node.l_tree , pre_node.l_tree, low_bound, mid, value);
     if (value > mid)
              new_node.r_tree = apply a new node from node pool
              build(new_node.r_tree, pre_node.r_tree, mid+1, high_bound, value);
}
```

	1	2	3	4
a[]:	10	18	4	9

insert 3rd element: 4



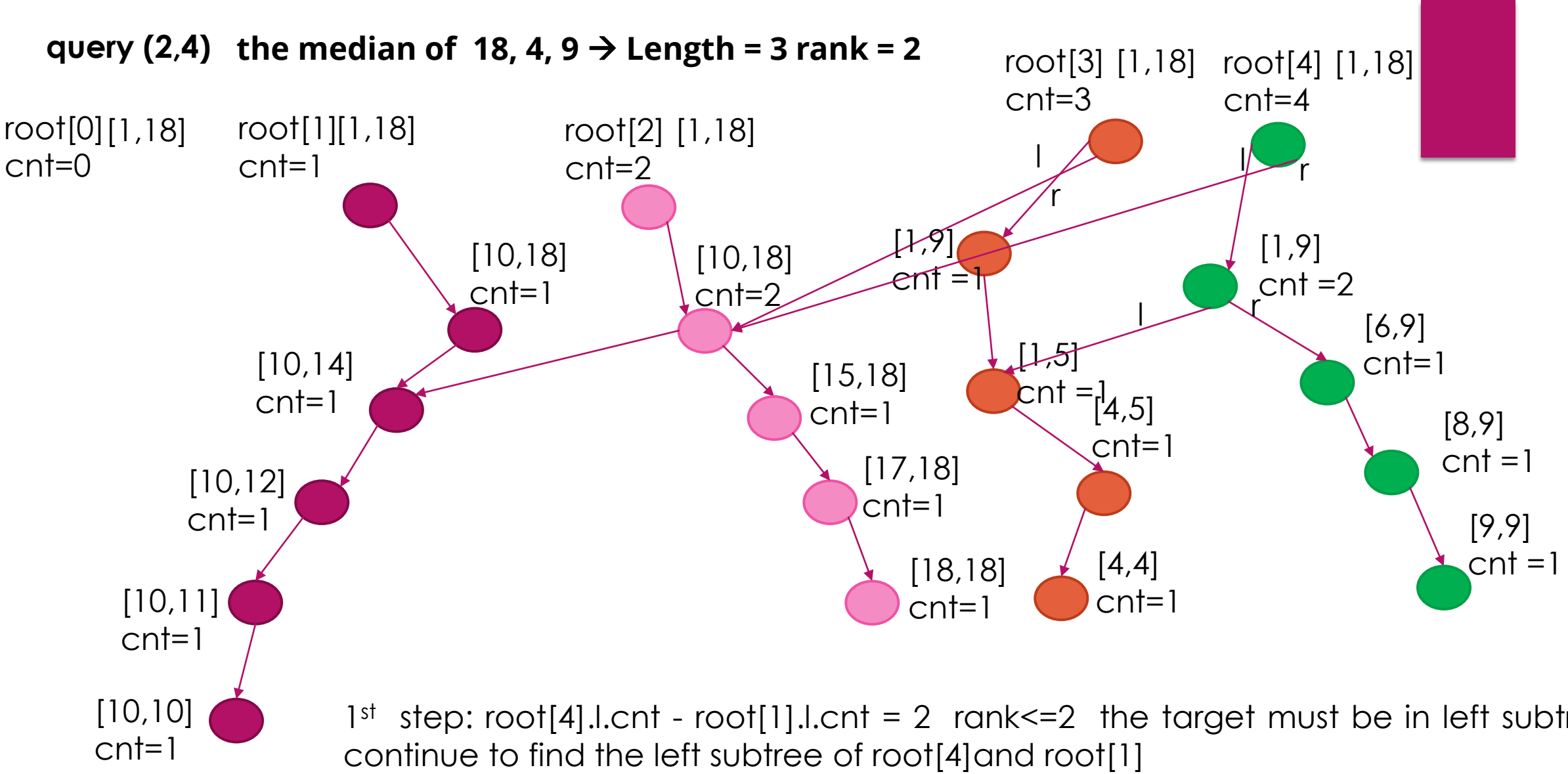




node pool size:

Nlog(max(a[])-min(a[])+1)

query (2,4) the median of 18, 4, 9 → Length = 3 rank = 2



1st step: root[4].l.cnt - root[1].l.cnt = 2 rank<=2 the target must be in left subtree, continue to find the left subtree of root[4]and root[1]

2nd step: left_diff = root[4].l.l.cnt - root[1].l.l.cnt =1 < rank the target must be in right subtree, continue to find the right subtree of root[4].l rank = rank – left_diff

….

repeat until leaf node, return the value of leaf node.

```
value  query(Node l_tree, Node r_tree, low_bound, high_bound, rank){
    if (low_bound = high_bound)
        return low_bound;
    int mid = ⌊(low_bound + high_bound)/2⌋
    int left_diff = r_tree.l.cnt – l_tree.l.cnt;
    if (rank ≤ left_diff) return query(l_tree.l, r_tree.l, low_bound, mid, rank);
    if (rank > left_diff) return query(l_tree.r, r_tree.r, mid+1, high_bound, rank - left_diff);
}
```