Persistent DS

Jonathan Irvin Gunawan
National University of Singapore

prerequisite

tau pointer

hayooo.. yang ngga tau berarti
ketauan ngga dengerin lecture Bu Inge
lecture pertama pisan

tau linked list

tau segment tree

In computing, a **persistent data structure** is a data structure that always preserves the previous version of itself when it is modified.

buat ngilustrasiin, pake soal ini aja

diberikan linked list, ada dua tipe operasi :

1. update x value pertama

2. print linked list setelah update ke-k

bruteforce :
bikin linked list baru tiap update,
simpen ke array untuk tiap “versi”

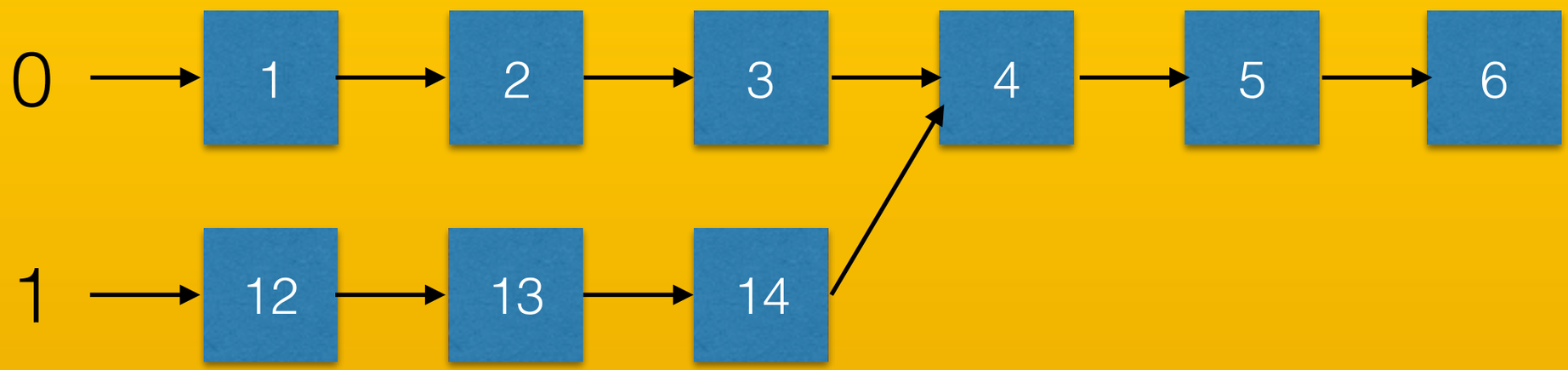
kalo x kecil, bisa manfaatin linked list sebelomnya

allocate cuma x space baru, terus elemen ke-x point ke element ke-(x+1) dari linked list sebelomnya

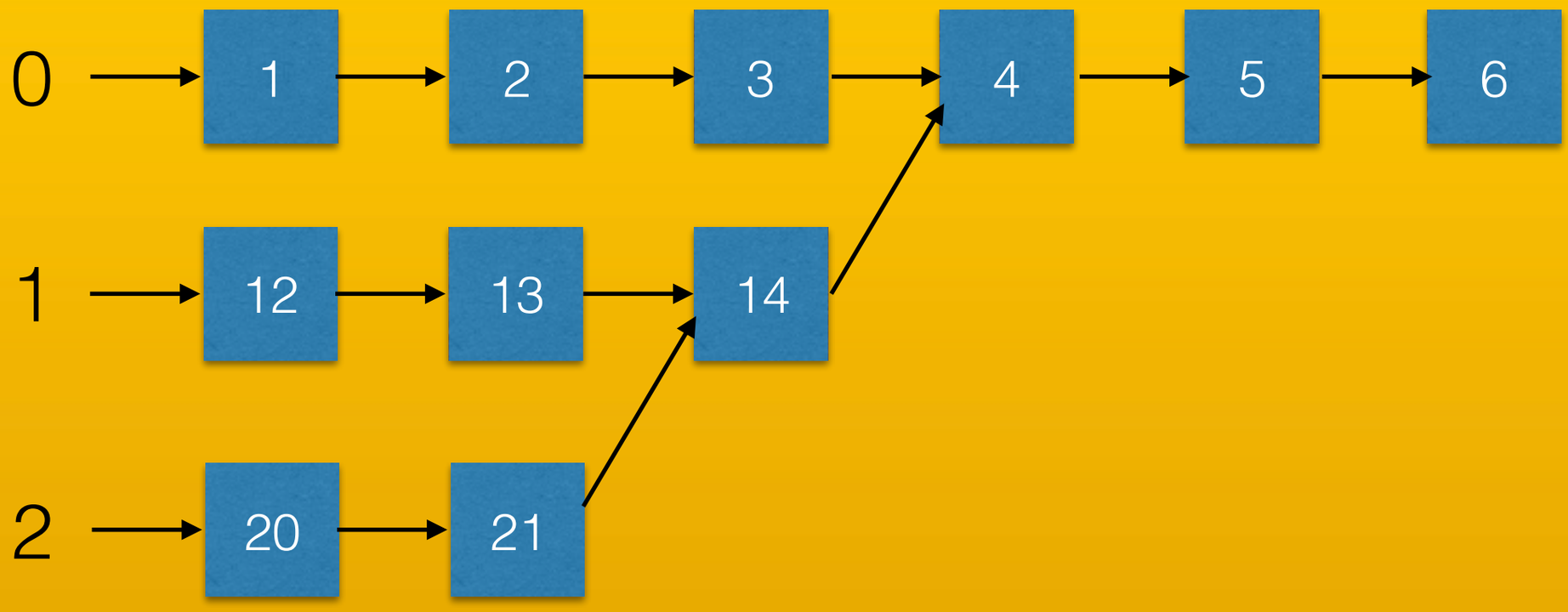
pasti ga ngerti

contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}

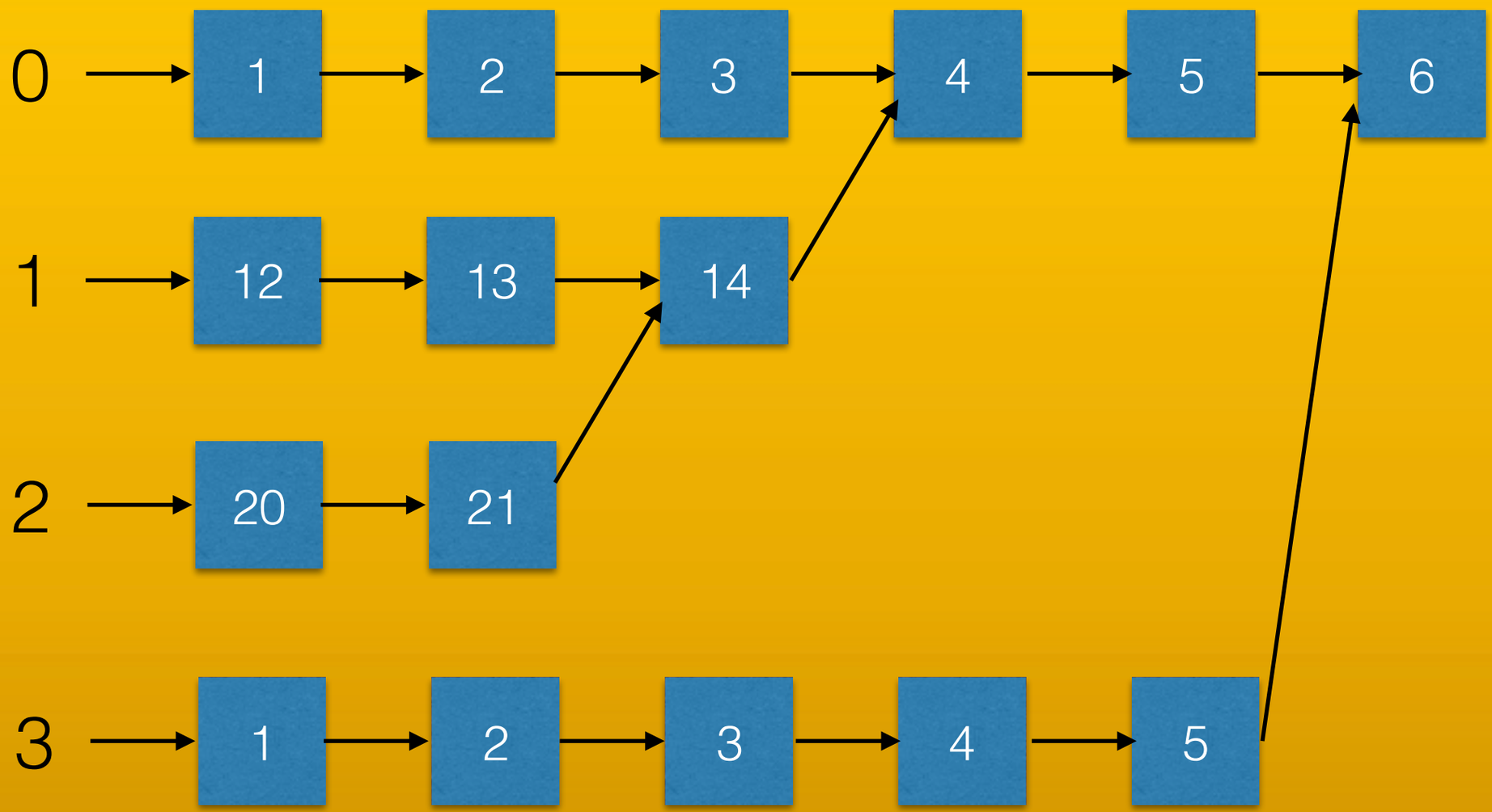
contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}



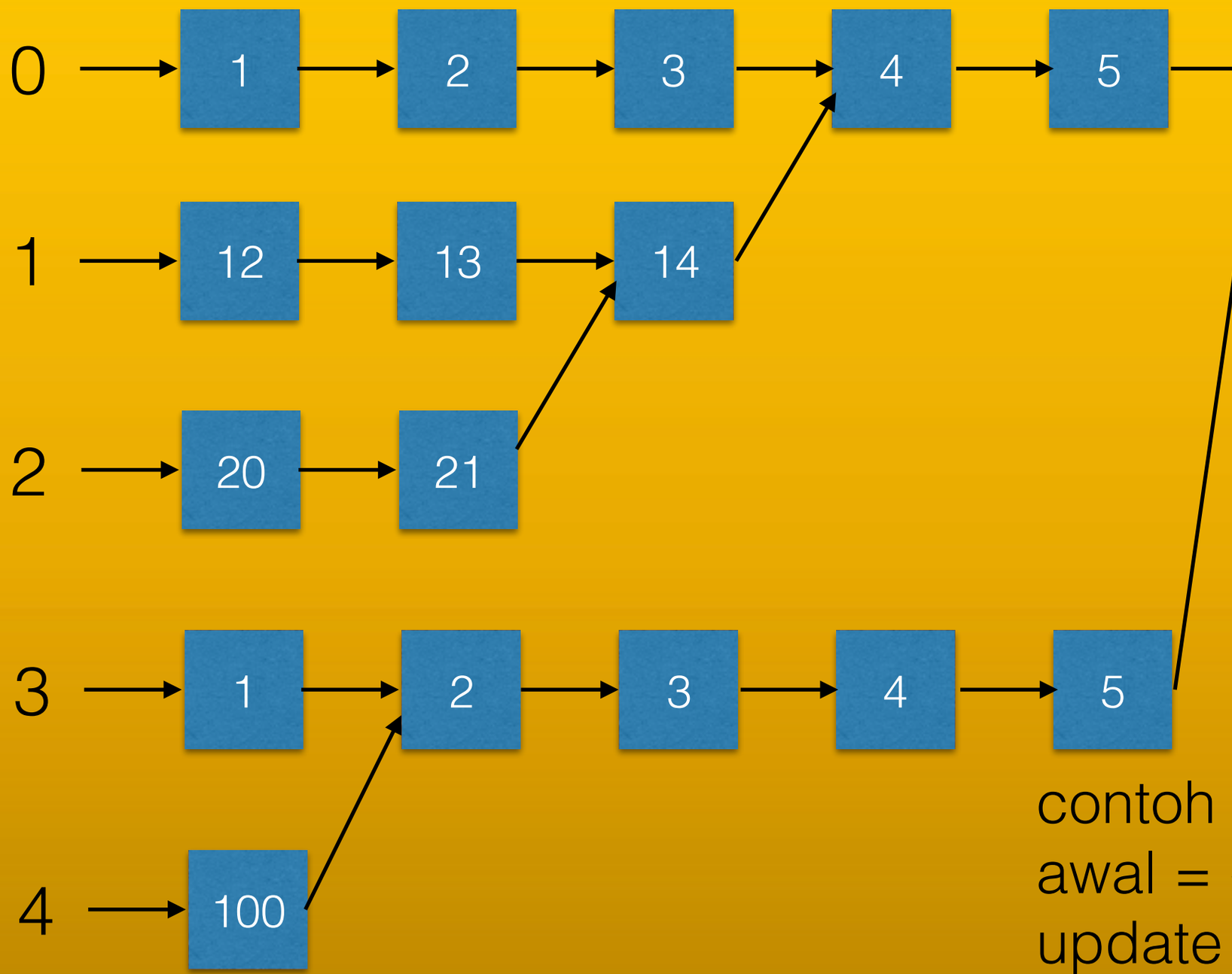
contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}



contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}



contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}



contoh
awal = {1, 2, 3, 4, 5, 6}
update 1 : x = 3, {12, 13, 14}
update 2 : x = 2, {20, 21}
update 3 : x = 5, {1, 2, 3, 4, 5}
update 4 : x = 1, {100}

bikin classnya dulu

```
class Node {
  int val;
  Node* next;

  Node(int val, Node* next): val(val), next(next) {}
}
```

```
Node* last[N];
Node* head[Q];

void init(vector<int> awal) {
  last[N - 1] = new Node(awal[N - 1], NULL);
  for (int i = N - 2; i >= 0; --i) {
    last[i] = new Node(awal[i], last[i + 1]);
  }
}

Node* insert(int index, vector<int> x) {
  if (index >= x.size()) {
    return last[index];
  }
  Node* now = new Node(x[index], insert(index + 1, x));
  return now;
}
```

kalo ada update vector x (katakan update ke-k) tinggal

```
head[k] = insert(0, x);
```

```
Node* last[N];
Node* head[Q];

void init(vector<int> awal) {
  last[N - 1] = new Node(awal[N - 1], NULL);
  for (int i = N - 2; i >= 0; --i) {
    last[i] = new Node(awal[i], last[i + 1]);
  }
}

Node* insert(int index, vector<int> x) {
  if (index >= x.size()) {
    return last[index];
  }
  Node* now = new Node(x[index], insert(index + 1, x));
  return last[index] = now; // jangan lupa diupdate last nya
}
```

kalo ada update vector x (katakan update ke-k) tinggal

```
head[k] = insert(0, x);
```

```
Node* last[N];
Node* head[Q];

void print(Node* now) {
   if (now == NULL) {
      return;
   }
   printf(“%d “, now->val);
   print(now->next);
}
```

kalo ada query print linked list ke-k tinggal

```
print(head[k]);
```

nah sekarang
persistent segment tree

kasih motivasi dulu

dikasih array A isinya N bilangan, dikasih Q query.

querynya itu dikasih tiga bilangan x,y,z. lu harus jawab ada berapa i yang memenuhi x≤i≤y, A[i] ≤ z

0 ≤ A[i] ≤ N, biar ga butuh kompres2an

range trees?

O(lg^2 N) per query?

gamau, maunya

O(lg N) per query

nih, misalkan kita punya infinite time dan memory buat precomputation sebelum query

kita bisa bikin N^2 segment tree untuk tiap interval (i,j)

kita bisa bikin N^2 segment tree untuk tiap interval (i,j)

tiap segment tree nyimpen ada berapa occurences untuk tiap bilangan HANYA pada range itu

```
int query(int ix, int L, int R, int z) {
  if (R == z) return tree[ix];
  int M = (L + R) >> 1;
  if (z <= M) return query(ix*2+1, L, M, z);
  else return tree[ix*2+1] + query(ix*2+2, M+1, R, z);
}
```

nah tapi preprocessingnya jadi O(N^3) kan

mahal parah

optimisasi 1 :

instead of N^2 segment tree, bisa kita optimize jadi N segment tree doang

node ke-k dari segment tree (i,j)

=

node ke-k dari segment tree (1,j)

-

node ke-k dari segment tree (1,i-1)

still, kita masih punya N segment tree, which means O(N^2) preprocessing

optimisasi 2 :

dari segment tree (1,i) ke segment tree (1,i+1), cuma log(N) node yang berubah

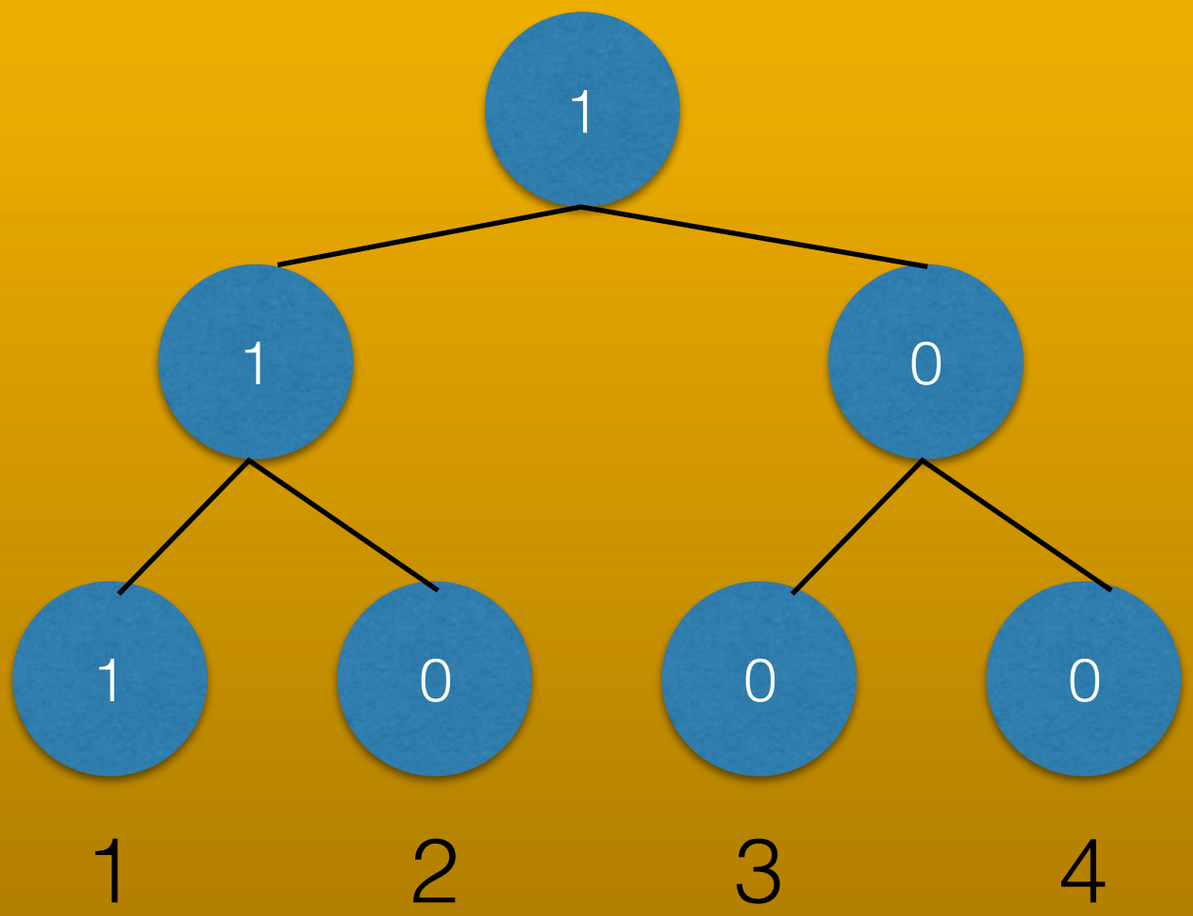
misal

$$A = \{1,2,3,1\}$$

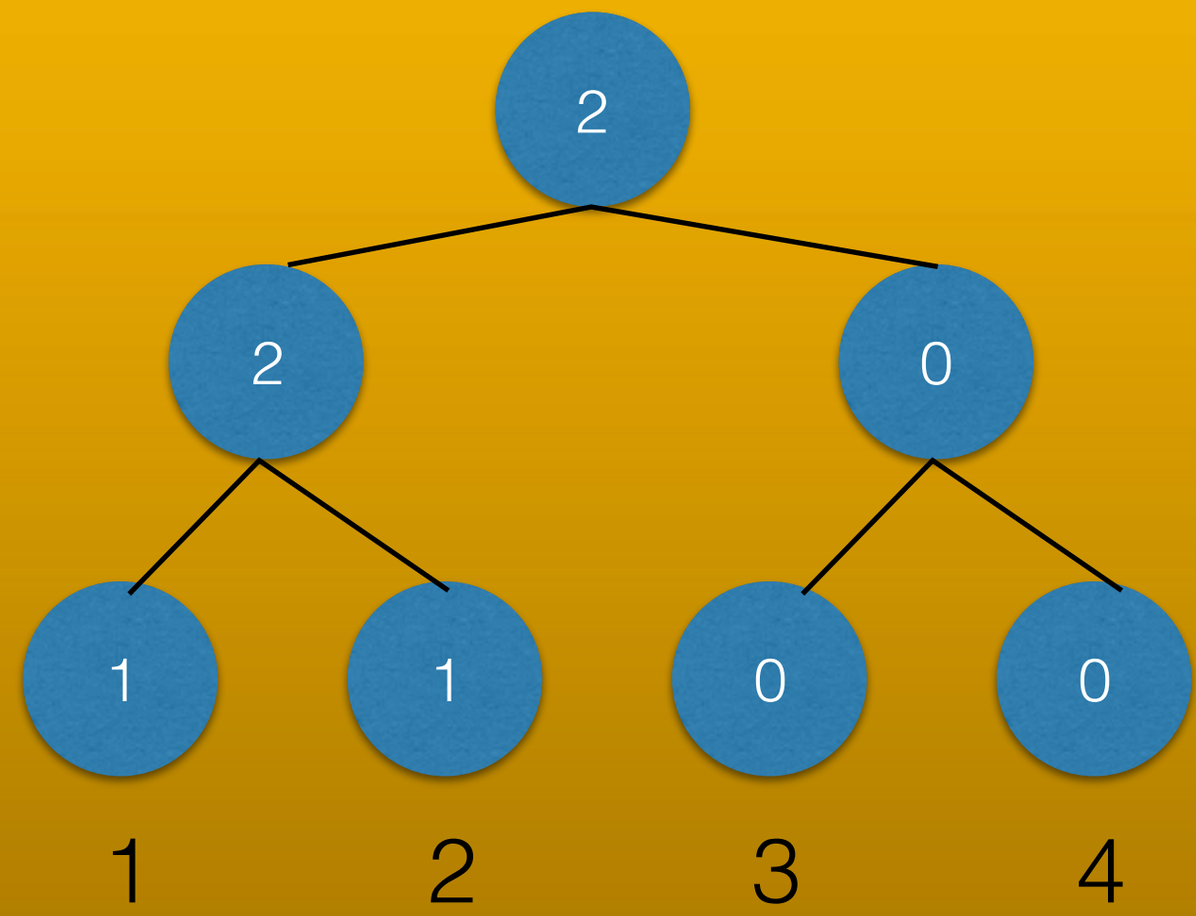
misal

A = {1,2,3,1}

segtree (1,1)



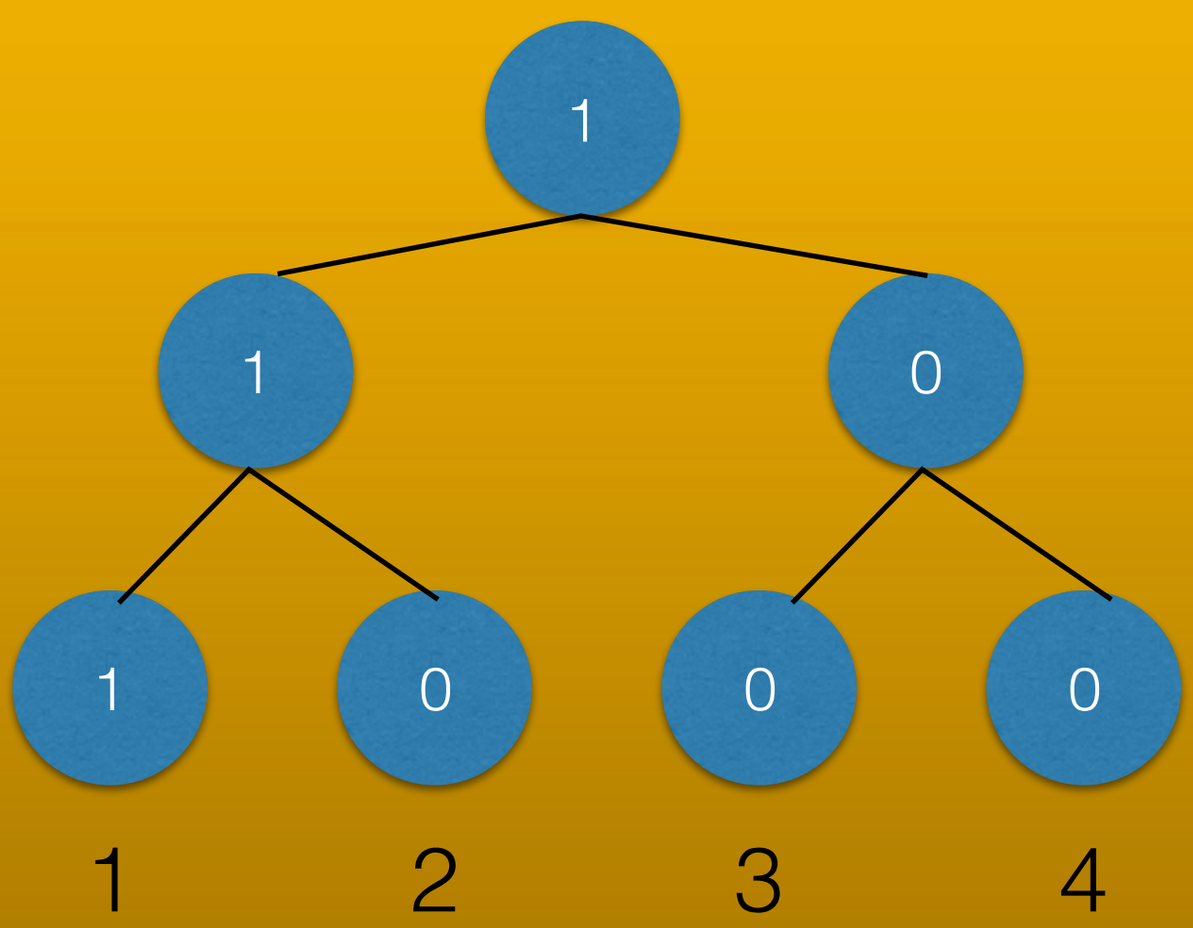
segtree (1,2)



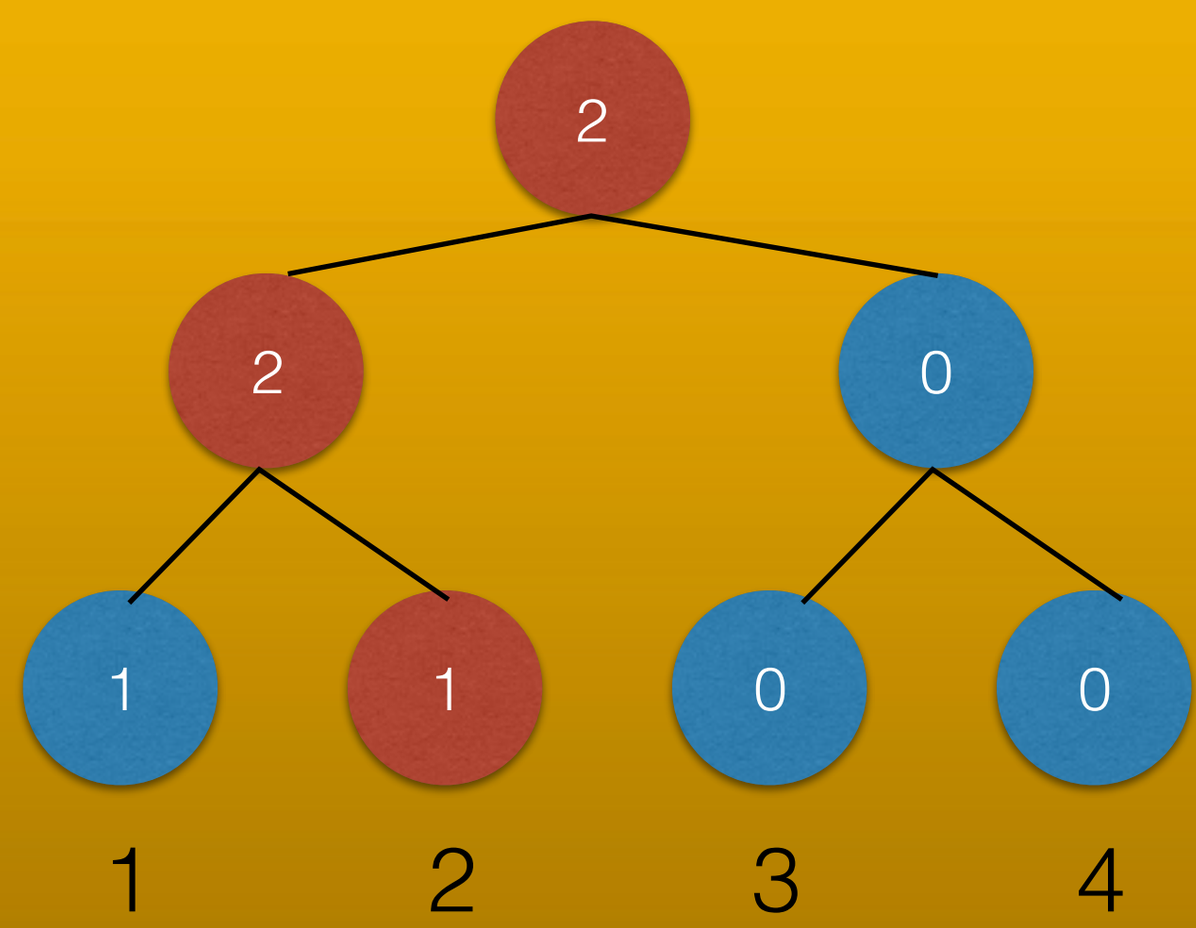
misal

A = {1,2,3,1}

segtree (1,1)



segtree (1,2)



pas compute segtree (1,2), kalo range sebuah node ga mencakup 2, pake node dari segtree (1,1)

bikin classnya dulu

```
class Node {
  int val;
  Node* left;
  Node* right;

  Node(int val, Node* left, Node* right):
    val(val), left(left), right(right) {}
}
```

```
Node* last[4 * N];
Node* root[N];

Node* insert(int ix, int L, int R, int z) {
  if (z < L || R < z) return last[ix];
  if (L == R) {
    Node* now = new Node(last[ix]->val + 1, NULL, NULL);
  } else {
    int M = (L + R) >> 1;
    Node* now = new Node(last[ix]->val + 1,
      insert(ix*2+1, L, M, z), insert(ix*2+2, M+1, R, z));
  }
  return last[ix] = now;
}
```

panggilnya `root[k] = insert(0,0,N-1,A[k]);`

querynya

```
Node* last[4 * N];
Node* root[N];

int query(Node* u, Node* v, int ix, int L, int R, int z) {
  if (L == R) return v->val - u->val;
  int M = (L + R) >> 1;
  if (z > M)
    return v->left->val - u->left->val +
           query(u->right, v->right, ix*2+2, M+1, R, z);
  return query(u->left, v->left, ix*2+1, L, M, z);
}


int x, y, z; // find how many integers < z in A[x..y]
int ans = query(root[x-1], root[y], 0, 0, N-1, z);
```

coba latihan

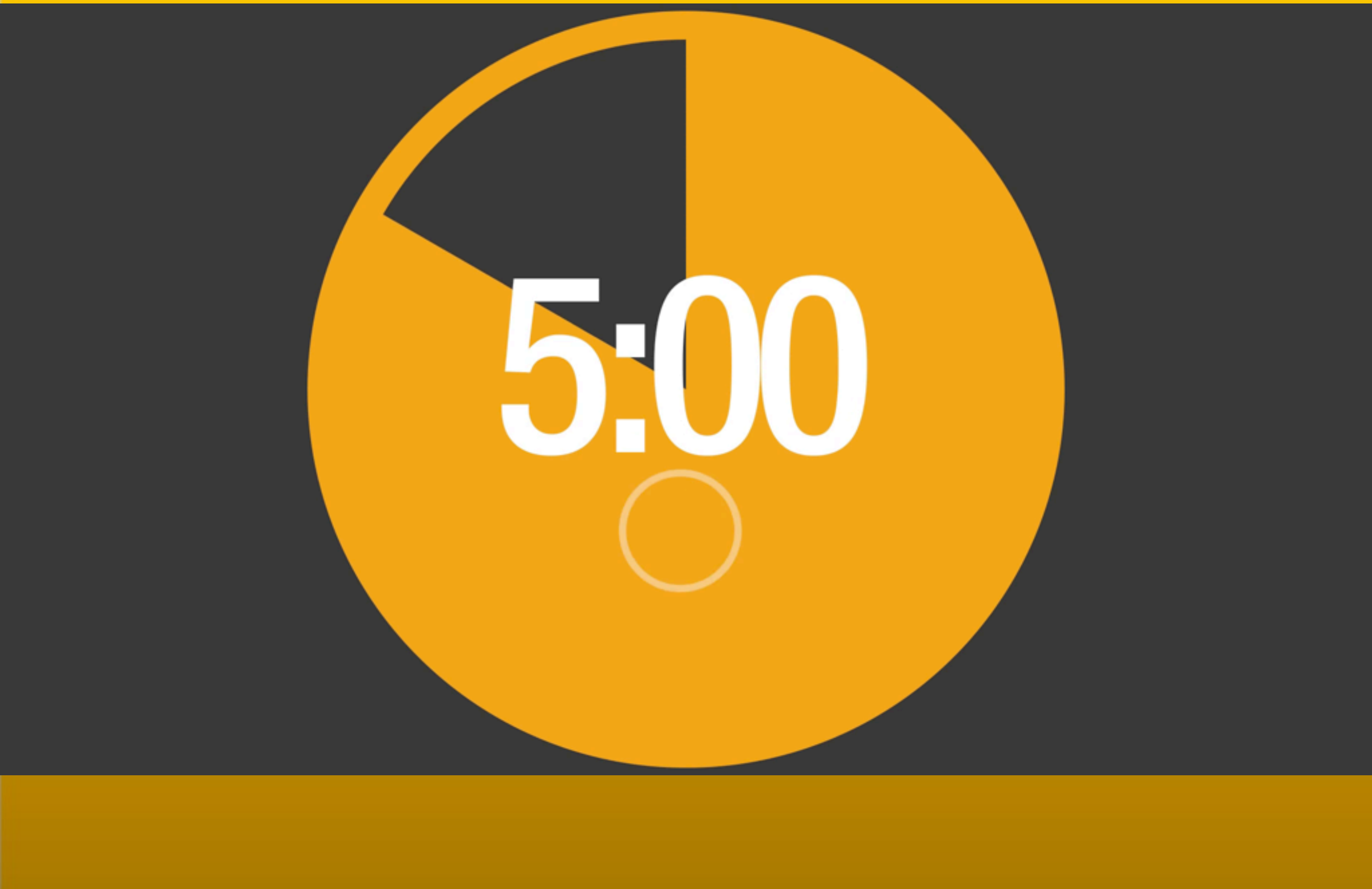
SPOJ MKTHNUM

gw cukup yakin ada yang udah pernah baca

dikasih array N dan Q query.
querynya dikasih tiga bilangan x,y,k.
Tentuin bilangan ke-k dari {A[x], A[x+1], A[x+2], …, A[y]} kalo disort

ada solusi pake range trees + binser, O(N lg^3 N)

tapi sekarang coba cari solusi O(N lg N) nya


5:00

EOF

Q&A?