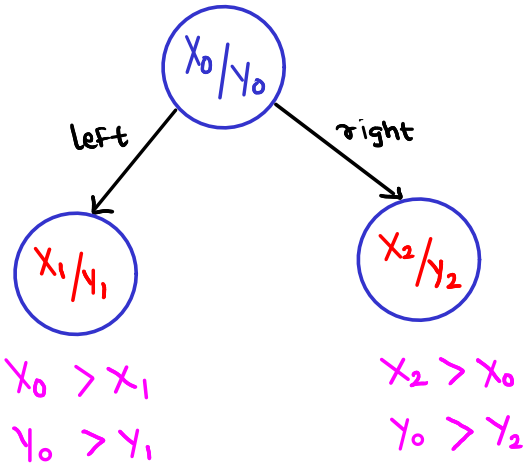
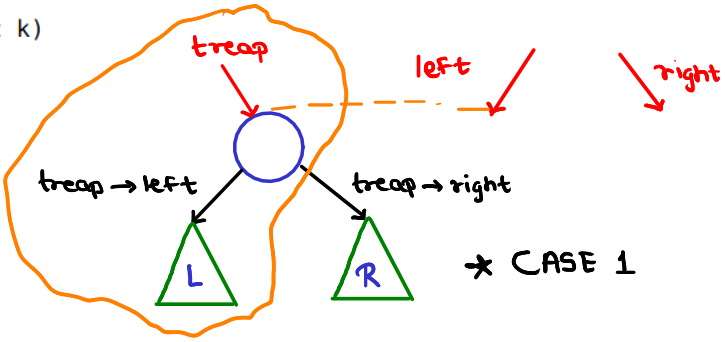
Treaps

TEJAS KHAIRNAR

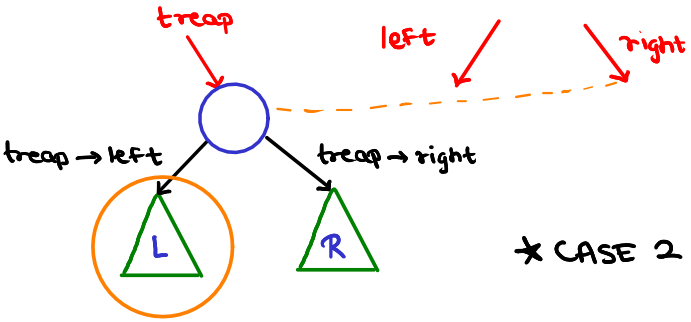
```
// Treap is a data structure that stores pairs (X, Y) in a binary tree.
// Binary search tree by X and a Binary heap by Y.
// Node contains values (X0, Y0)
// All nodes in the left  subtree have X < X0
// All nodes in the right subtree have X > X0
// All nodes in both left and right subtrees have Y < Y0.
```



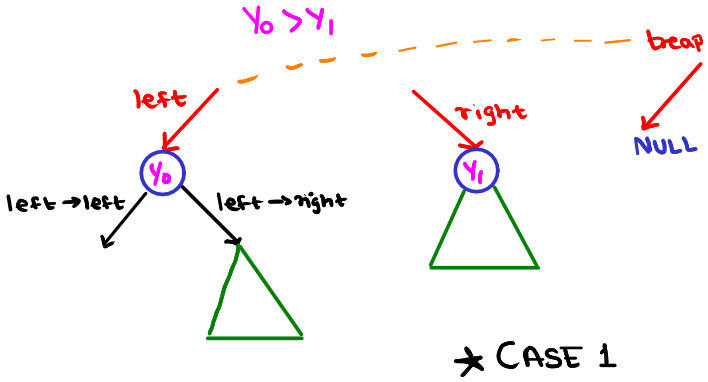

```
// The following function split implements the splitting operation.
// It recursively splits the treap treap into treaps left and right.
// Left treap contains the first k nodes and right treap contains the remaining nodes.

void split(node *treap, node *&left, node *&right, int k)
{
    if (treap == NULL)
    {
        left = right = NULL;
    }
    else
    {
        push(treap);
        if (size(treap->left) < k)
        {
            split(treap->right, treap->right, right, k - size(treap->left) - 1);
            left = treap;
        }
        else
        {
            split(treap->left, left, treap->left, k);
            right = treap;
        }
        treap->size = size(treap->left) + size(treap->right) + 1;
        combine(treap);
    }
}
```



Attach left treap of result @ treap → right

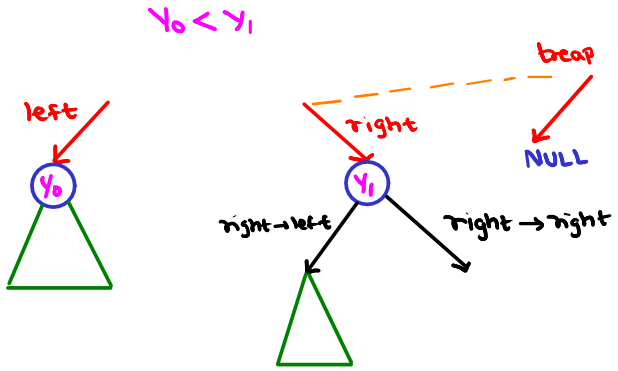


```
// The following function merge implements the merging operation.
// Creates a treap that contains first the nodes of the treap left and then the nodes of the treap right.

void merge(node *&treap, node *left, node *right)
{
    push(left);
    push(right);

    if (left == NULL)
        treap = right;
    else if (right == NULL)
        treap = left;
    else
    {
        if (left->prior > right->prior)
        {
            merge(left->right, left->right, right);
            treap = left;
        }
        else
        {
            merge(right->left, left, right->left);
            treap = right;
        }
        treap->size = size(treap->left) + size(treap->right) + 1;
        combine(treap);
    }
}
```



★ CASE 1

↳ Right half of recursion attached to right→left



★ CASE 2