```
void isset_print(const isset *s)
{
   isset_print_subtree(s->root);
}


void isset_print_subtree(const isset_node *n)
{
   if (n != NULL)
   {
      isset_print_subtree(n->left);
      printf("[%d-%d]\n", n->start, n->end);
      isset_print_subtree(n->right);
   }
}
```

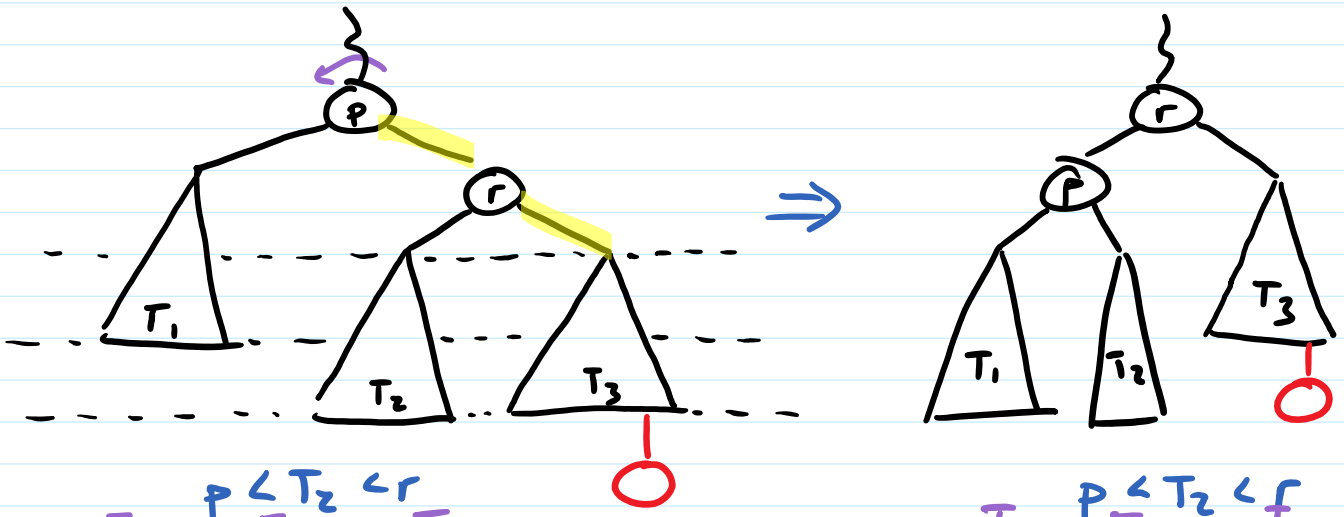
inorder traversal

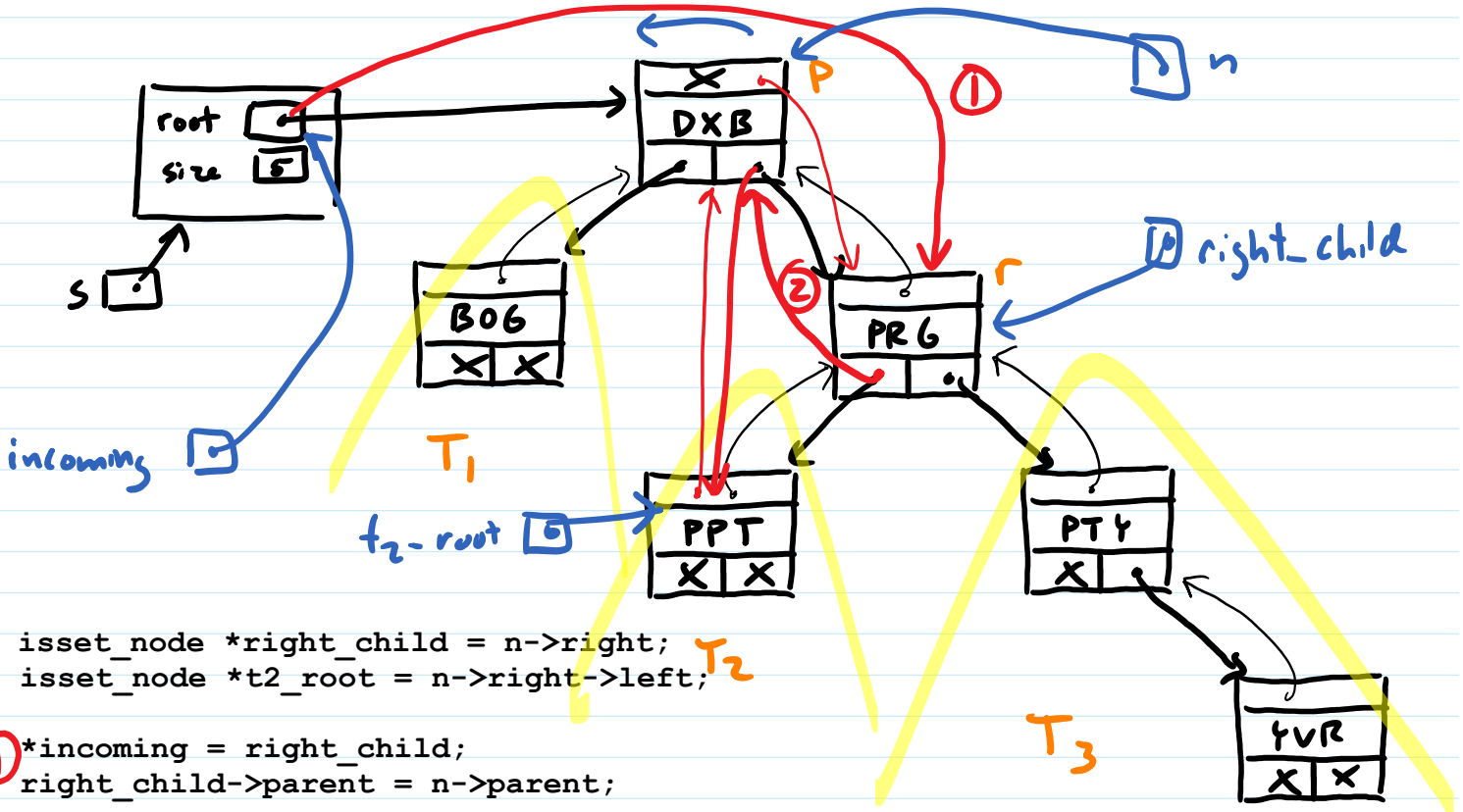
O(n)

```
void isset_destroy(isset *s)
{
   isset_destroy_subtree(s->root);
   free(s);
}


void isset_destroy_subtree(isset_node *n)
{
   if (n != NULL)
   {
      isset_destroy_subtree(n->left);
      iseet_destroy_subtree(n->right);
      free(n);
   }
}
```

postorder traversal

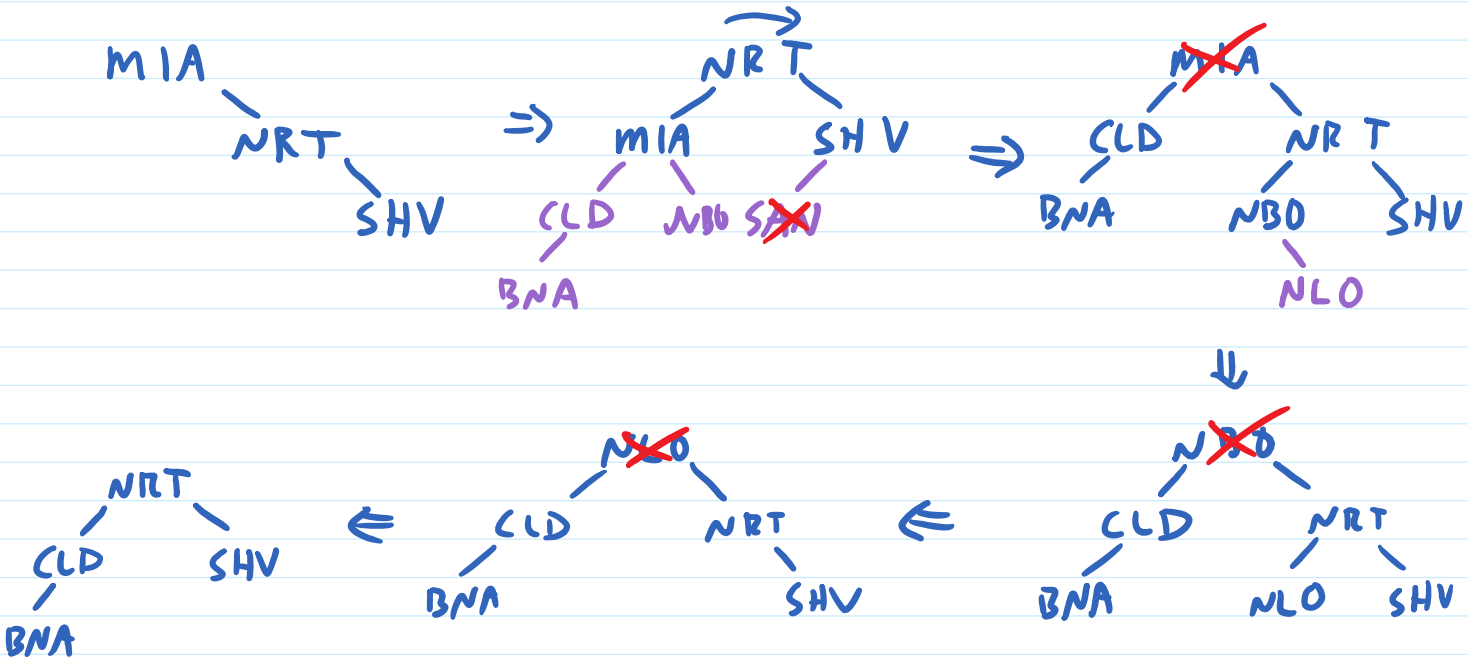
O(n)



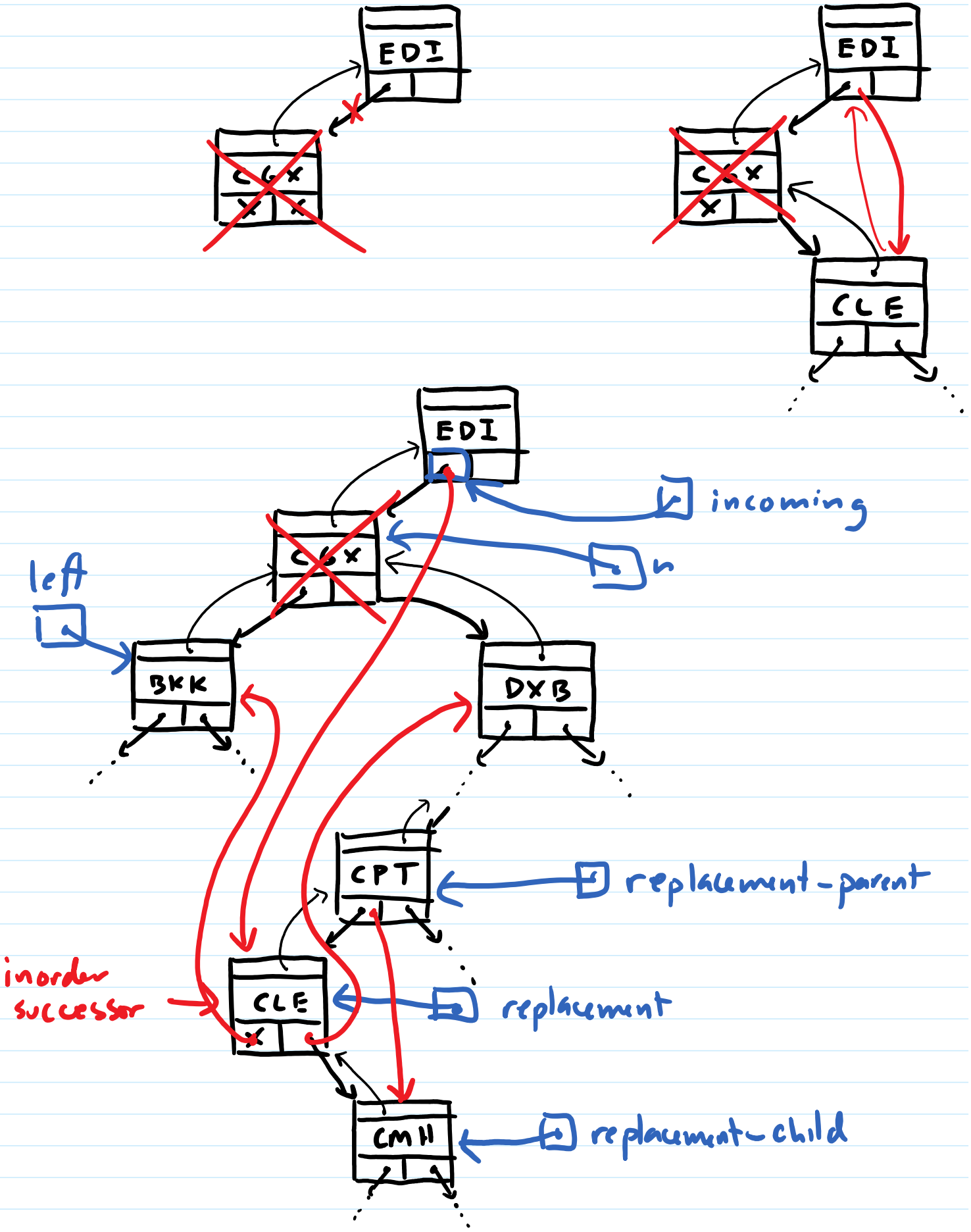


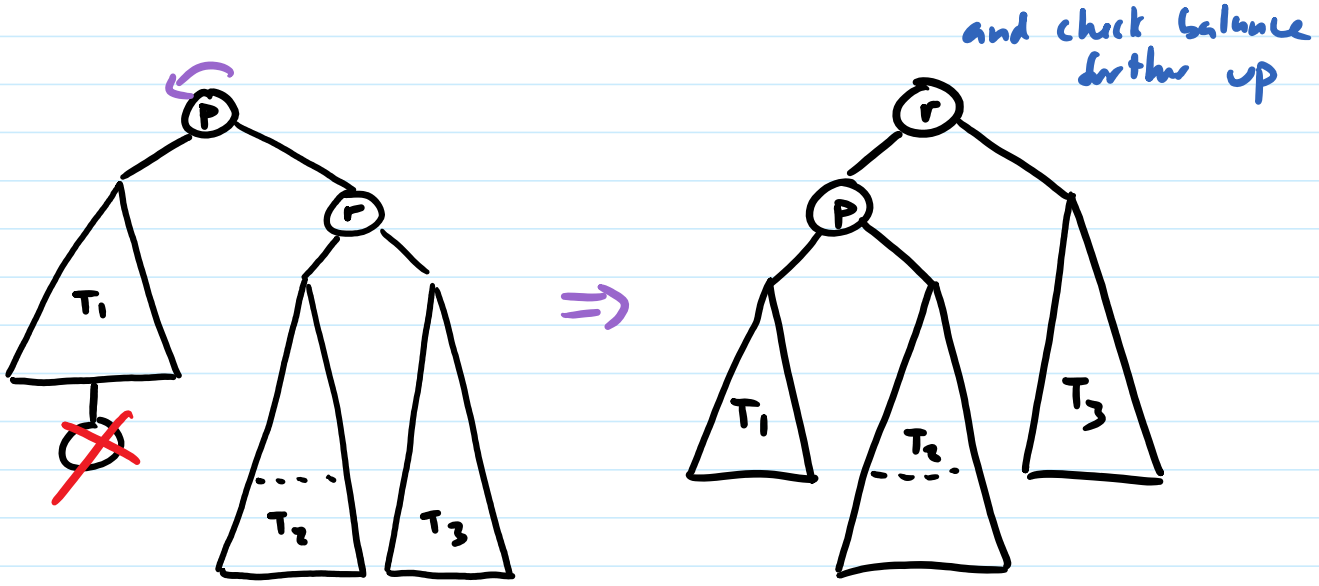

```
isset_node *right_child = n->right;
isset_node *t2_root = n->right->left;

1 *incoming = right_child;
  right_child->parent = n->parent;

2 right_child->left = n;
  n->parent = right_child;

  n->right = t2_root;
3 if (t2_root != NULL)
    {
      t2_root->parent = n;
    }
```

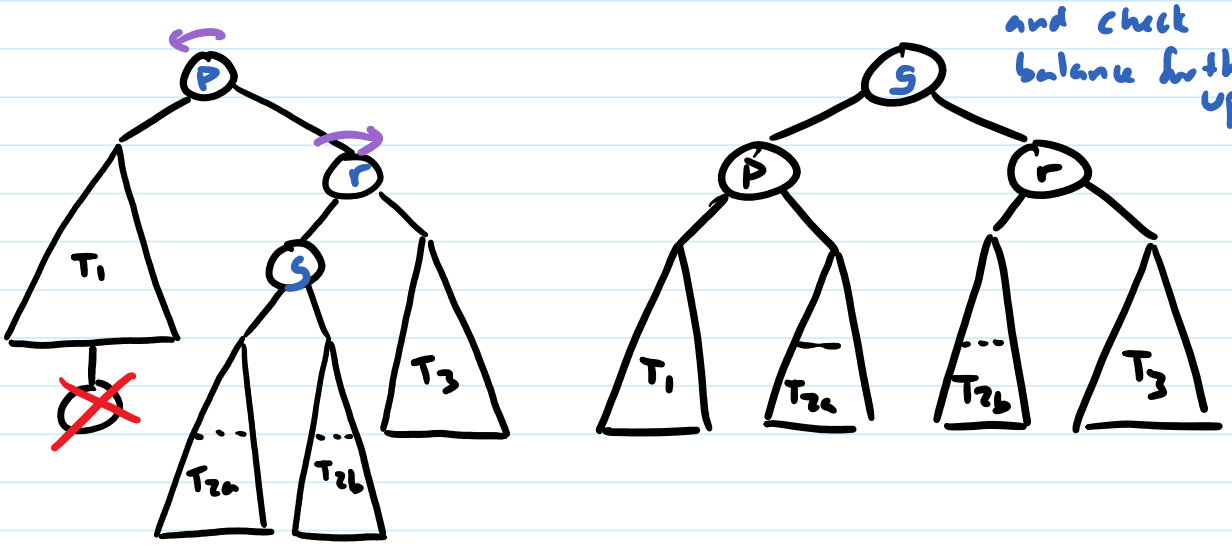
Example


MIA
NRT
SHV
NRT
MIA
SHV
CLD
NBO
BNA
CLD
NRT
BNA
NBO
SHV
NLO
NRT
CLD
SHV
BNA
CLD
NRT
BNA
SHV
CLD
NRT
BNA
NLO
SHV

Remove


EDI
CLE
EDI
CLE
EDI
CLE
incoming
n
left
BKK
DXB
CPT
replacement-parent
inorder successor
CLE
replacement
CMH
replacement-child

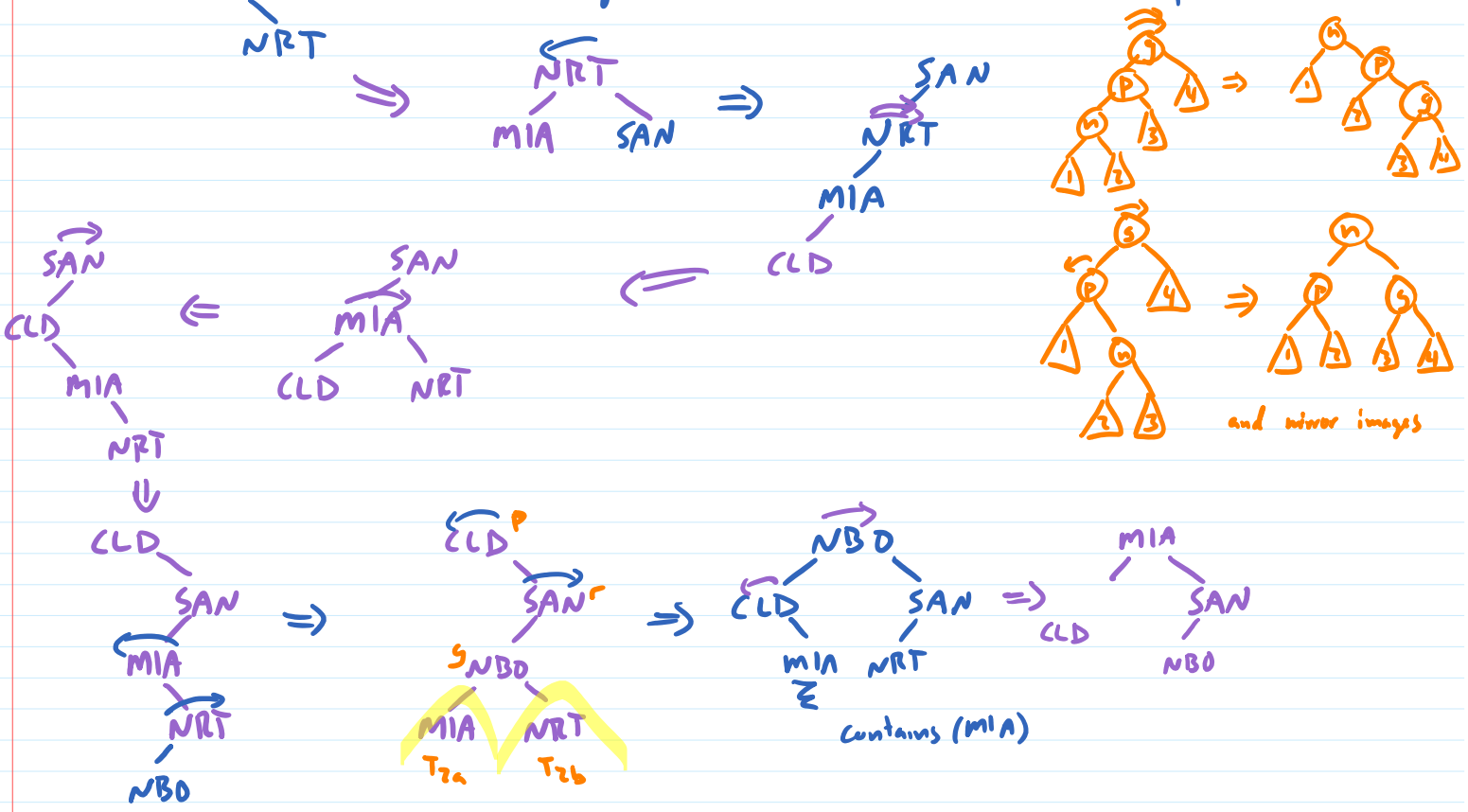

and check balance further up
p
r
T1
T2
T3
r
p
T1
T2
T3


and check balance further up
p
r
s
T1
T2a
T2b
T3
s
p
r
T1
T2a
T2b
T3

Splay Trees – after every operation, splay deepest node examined to top

↓

sequence of rotations to move two levels up at a time



$O(\log n)$ amortized time for add, remove, contains

so any sequence of n adds/removes/contains starting with empty tree takes $O(n \log n)$ time

(although a single operation may run in $\Theta(n)$ time)