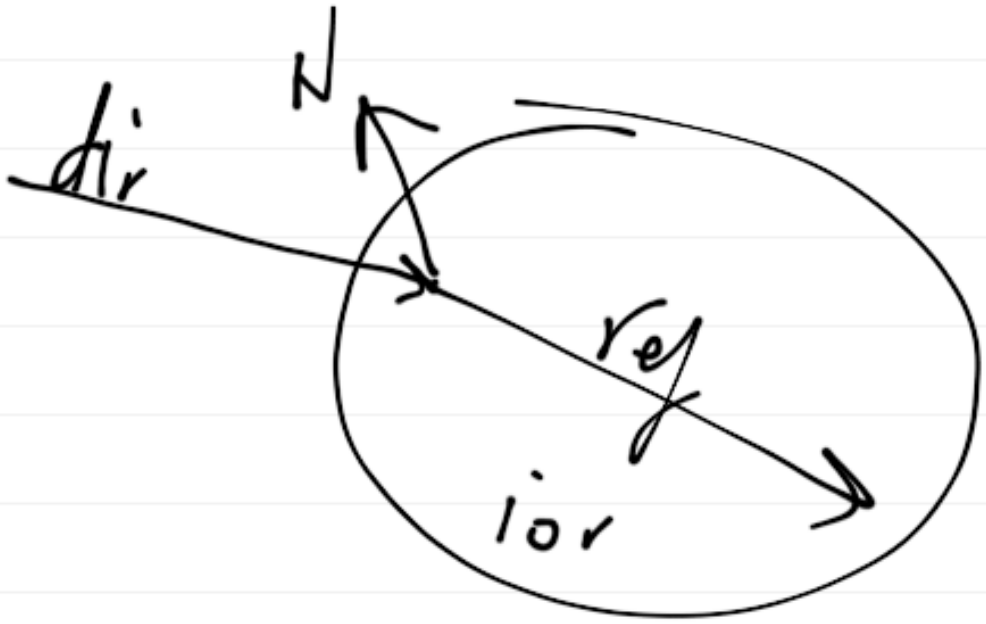


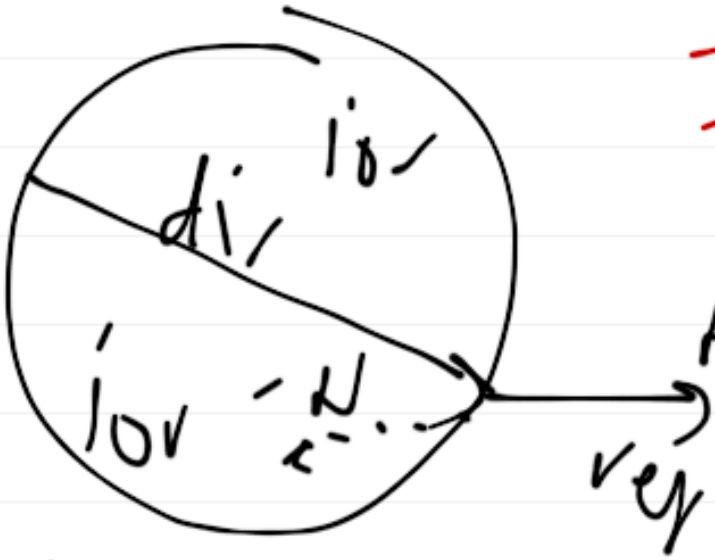
$\frac{n_1}{n_2}$: air

n_2

n_2

ref = dir.refract(N, ior)

Vec_t :: refract



$\frac{n_1}{1} = n_1$

$\frac{1}{n_2}$

n_2

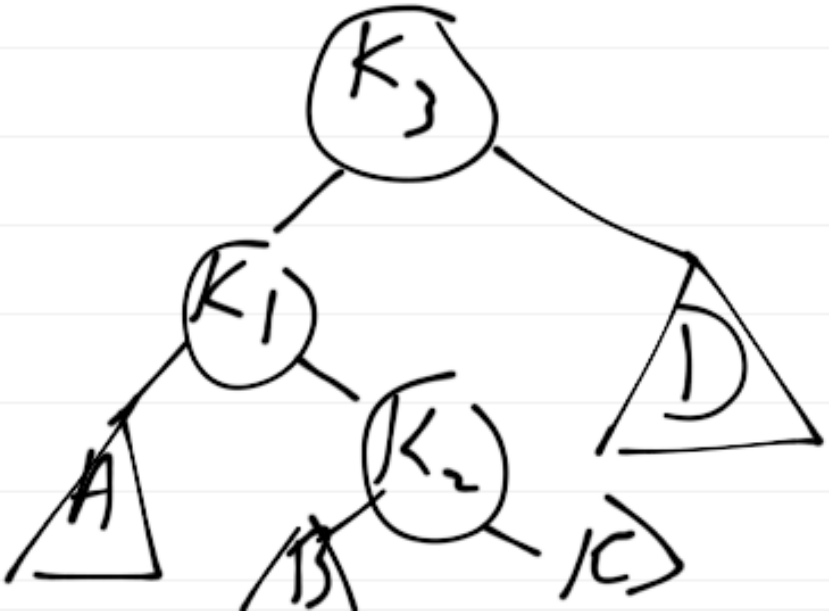
ref = dir.refract(-N, 1.0/ior)

double-left rotation is just the reverse of double-right rotation:

rotate_right

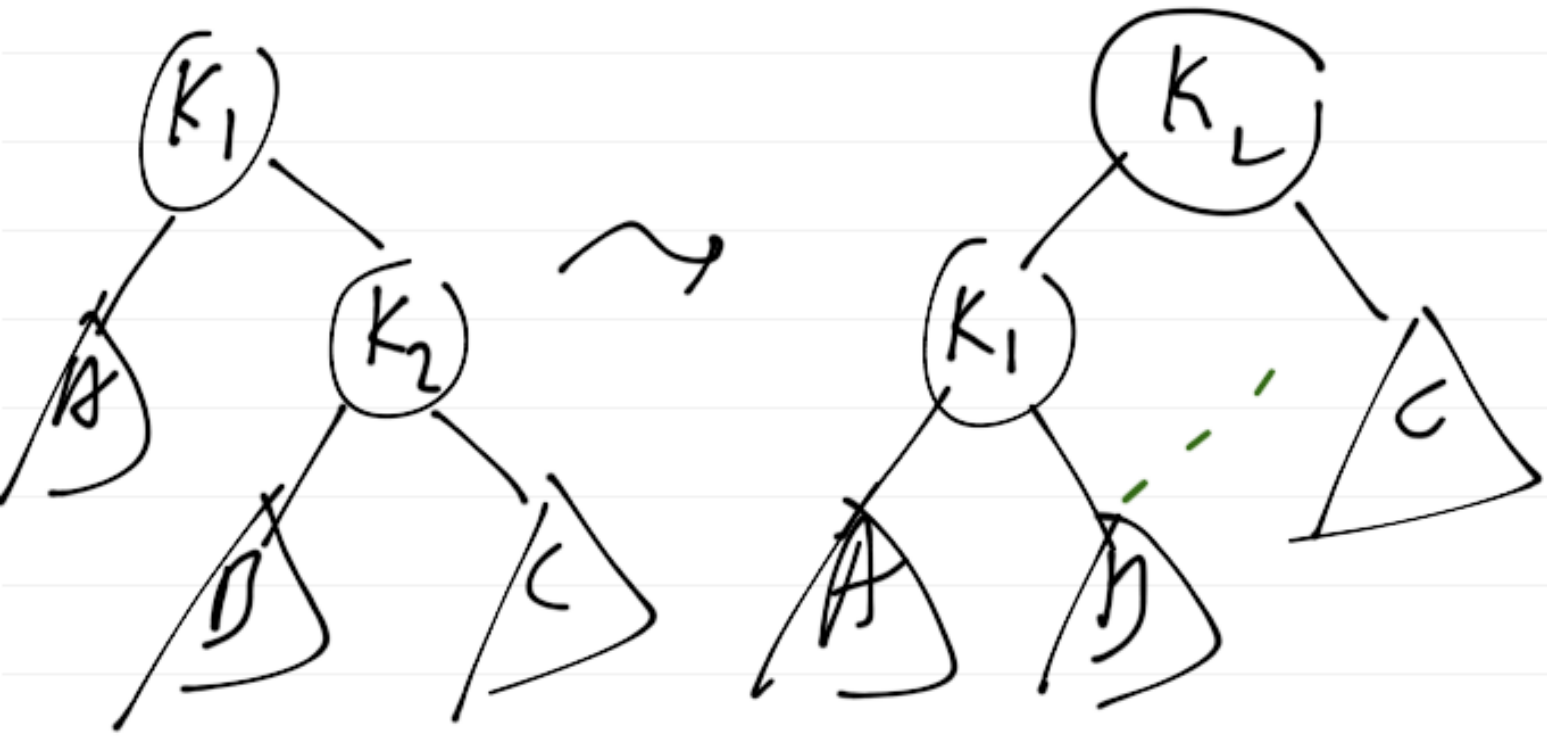
followed by

rotate_left.

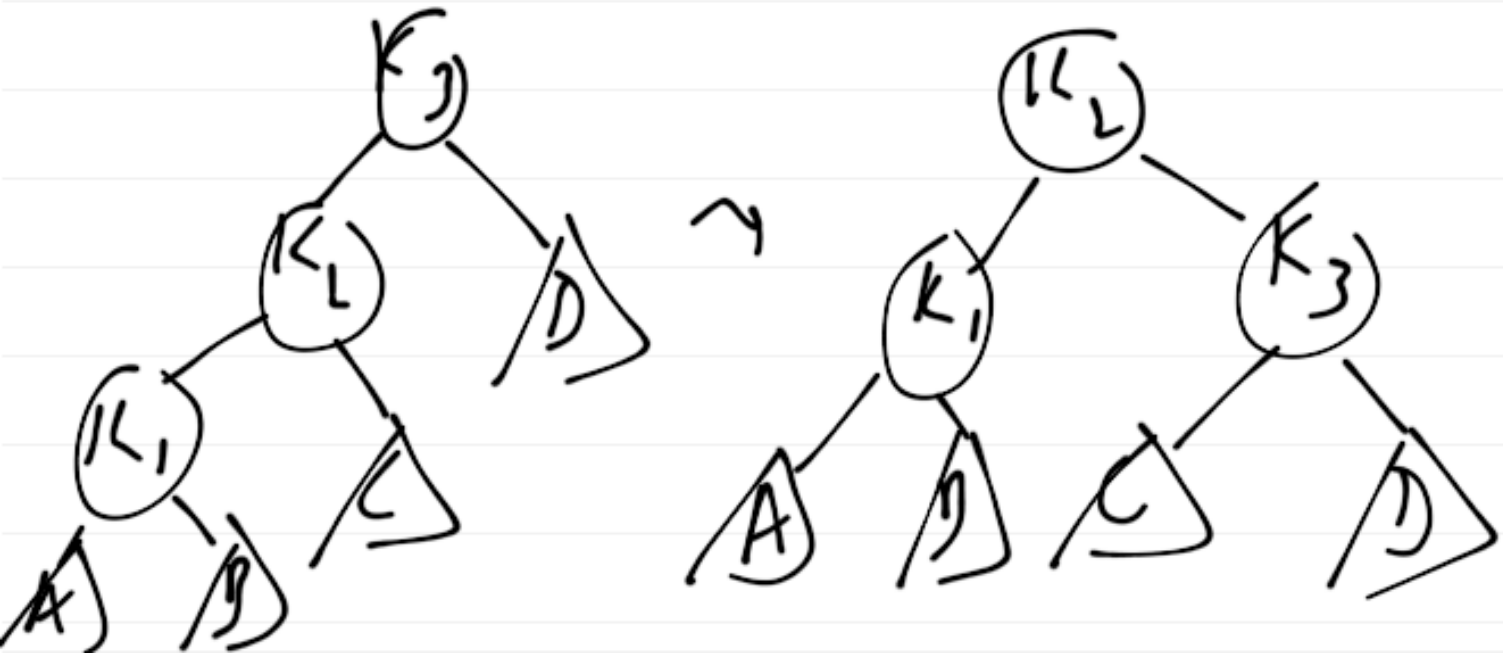


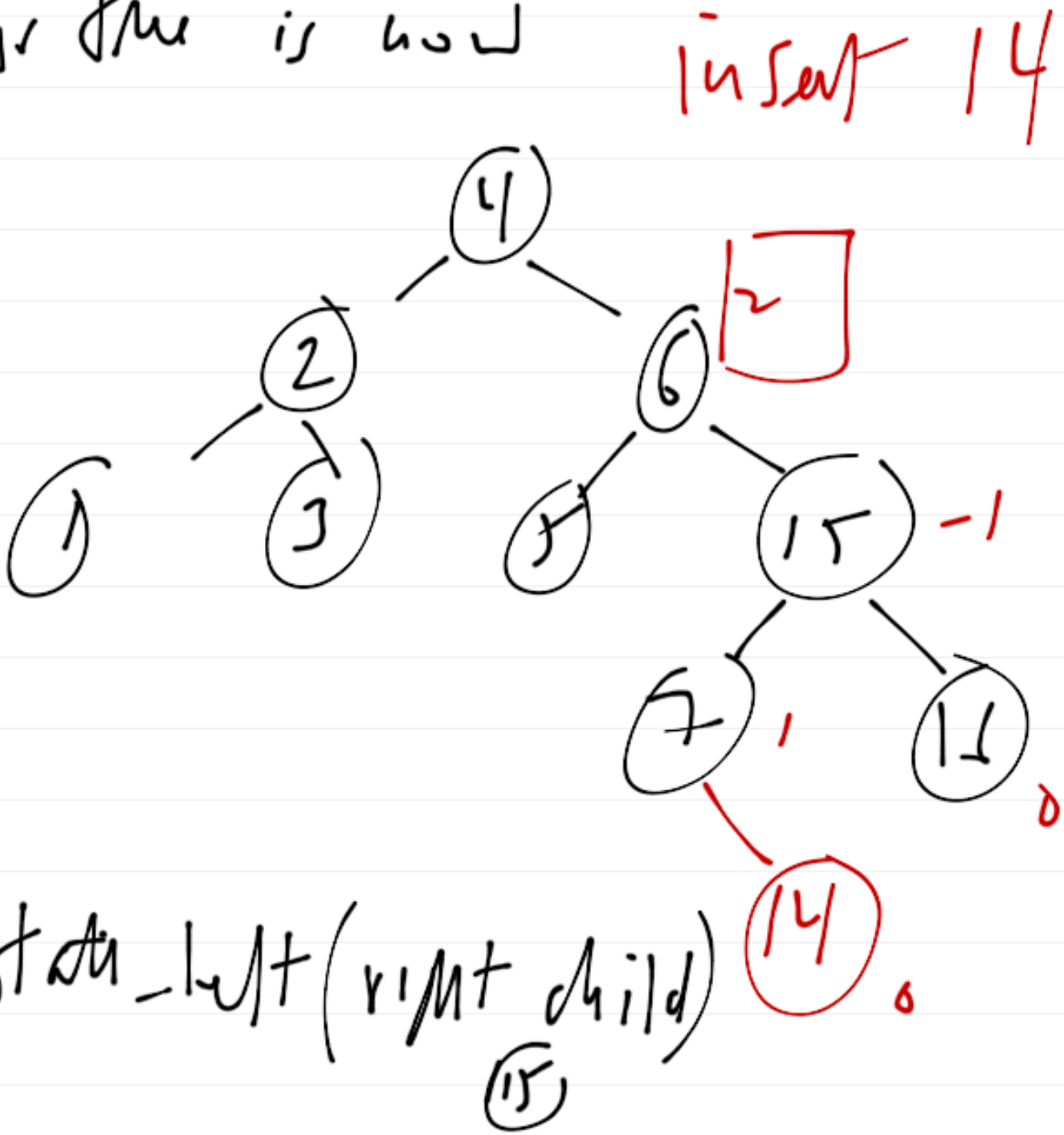
rotate_right (K_3 → left)

rotate_right($K_2 \rightarrow$ left)



rotate_left(K_3)





rotate_left(right child)

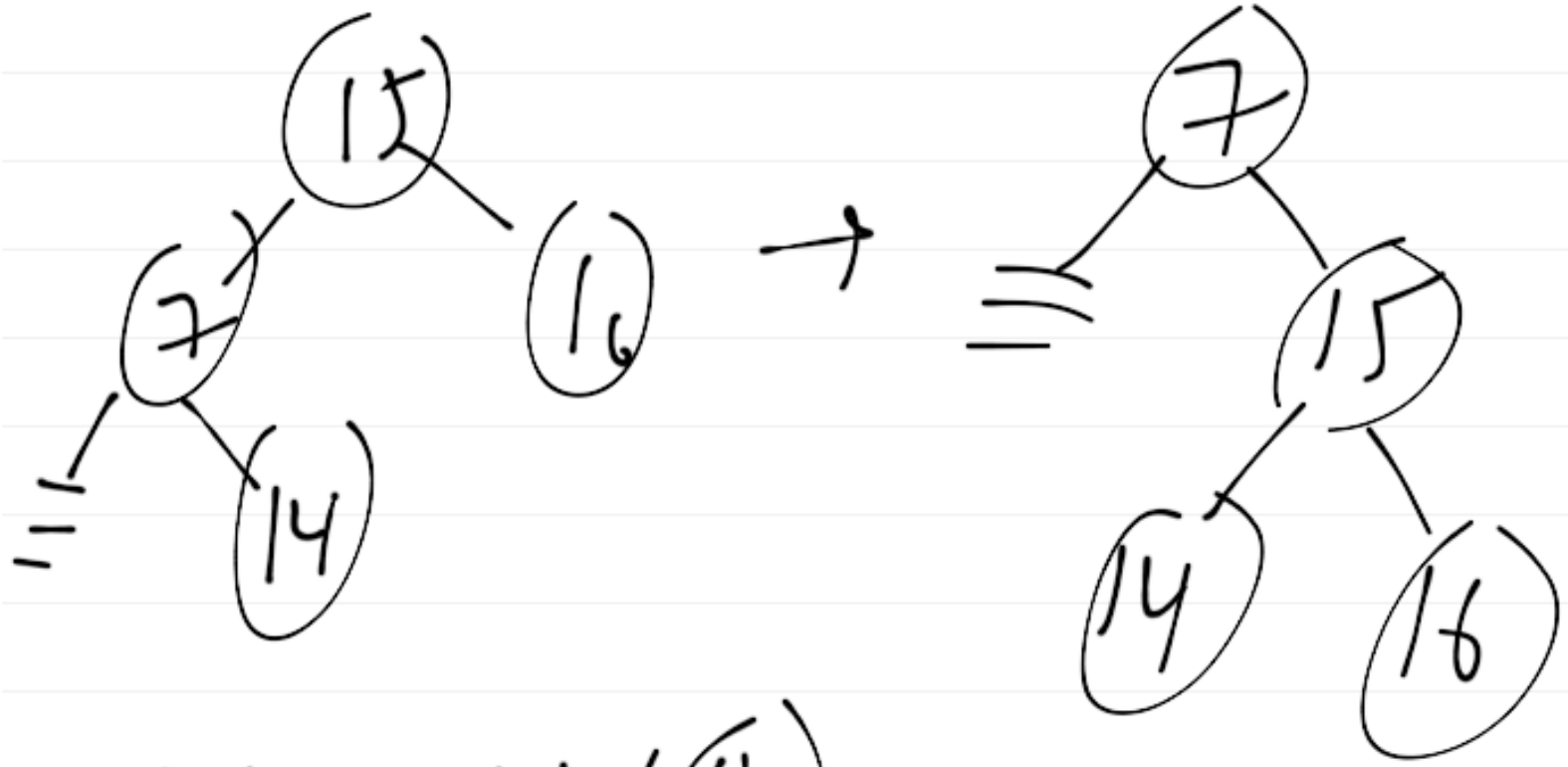
(15)

Then

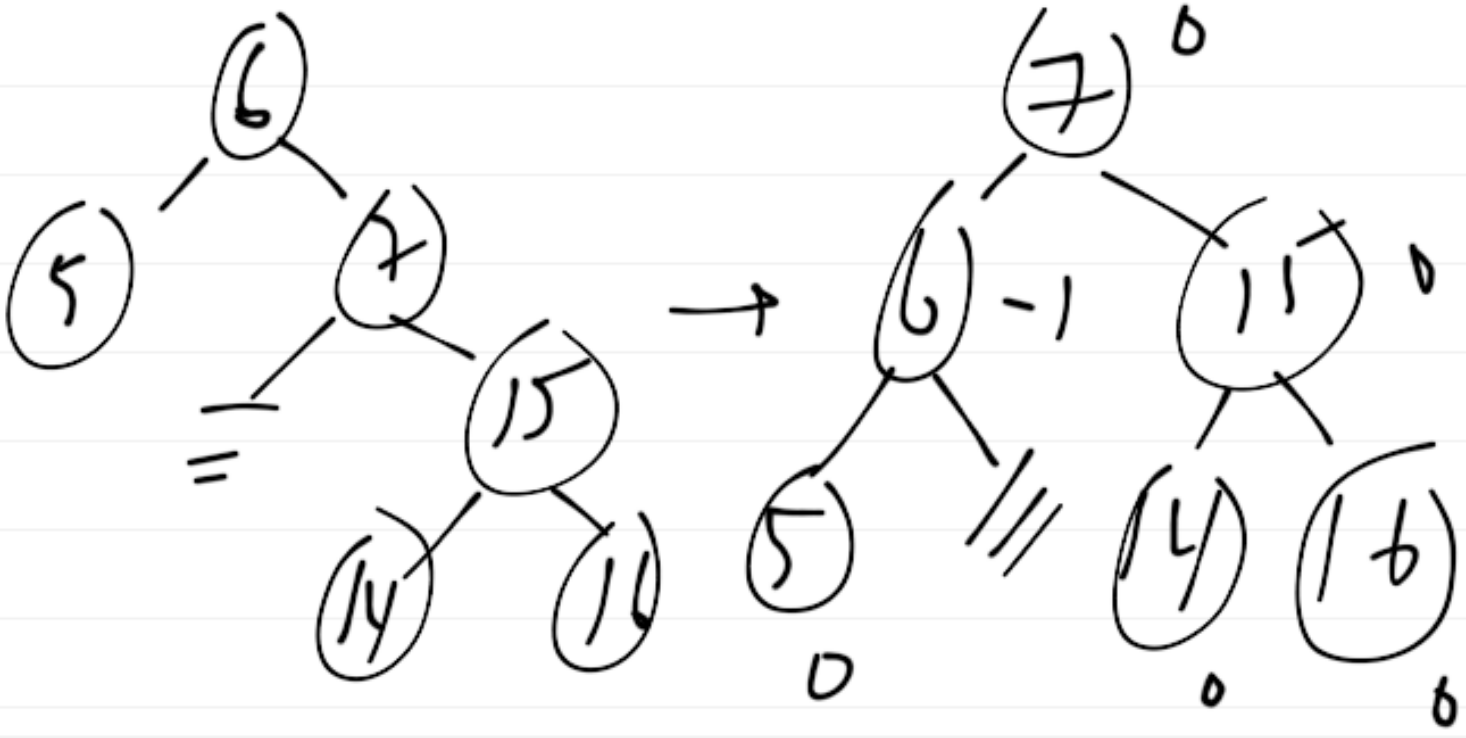
rotate_right(self)

(6)

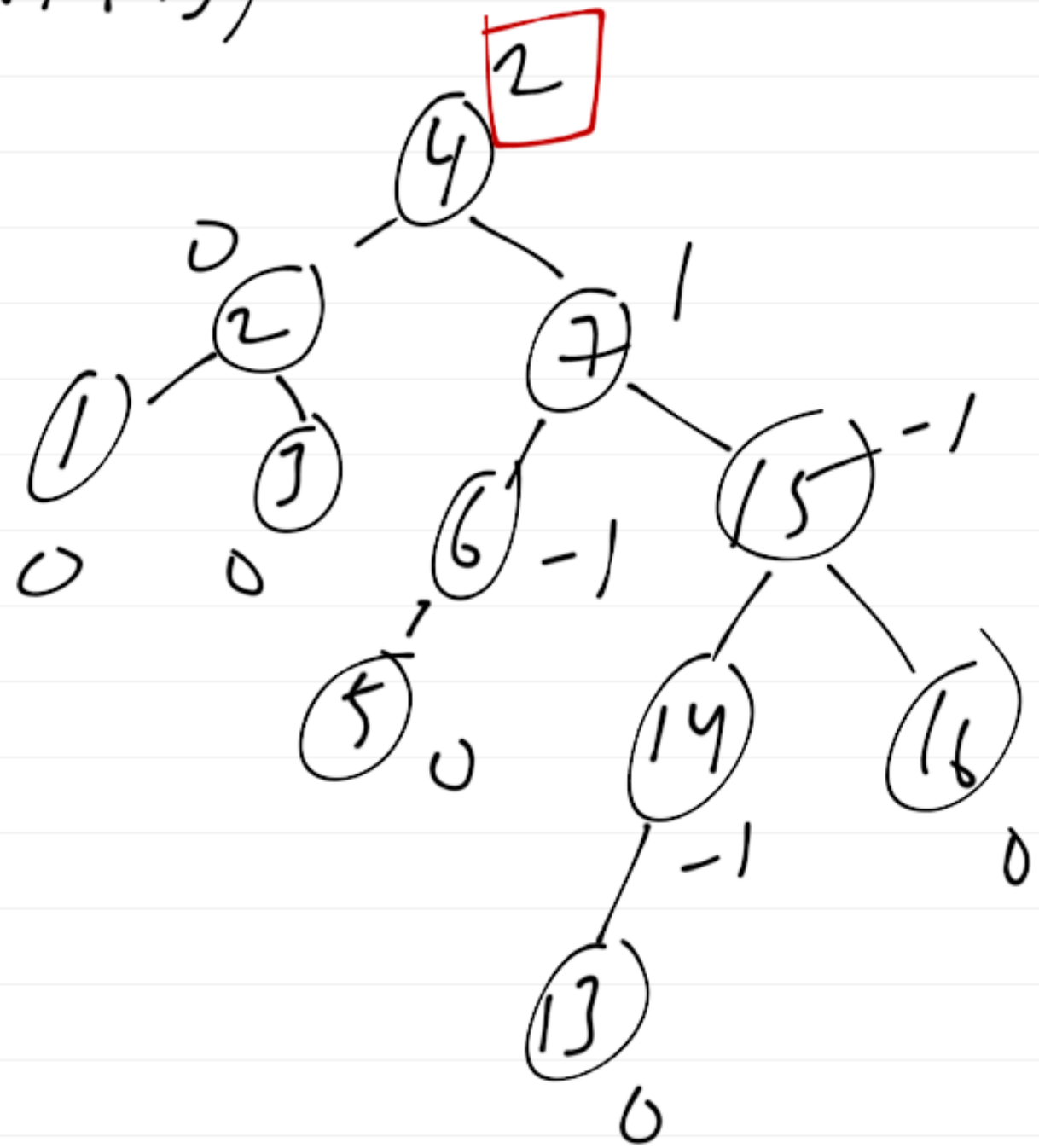
rotate_left((15))



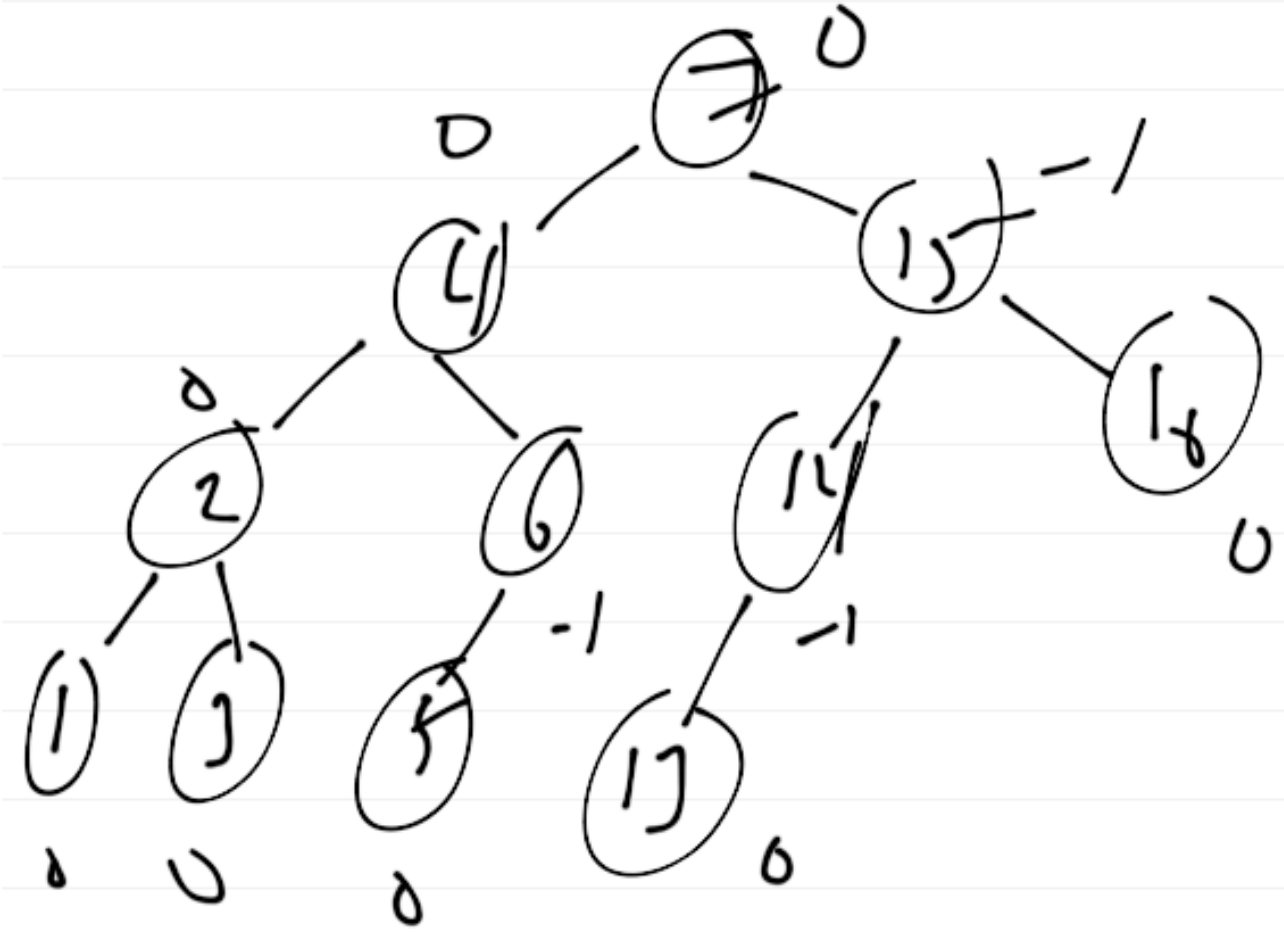
rotate_right((6))



insert(13)



single rotation



to implement AVL, just add additional int bal to BST node_t & set to 0 initially

```
insert(const T &x, node_t *&t)
  int Δh = 0 // delta h
  if (t == NULL) {
      t = new node_t(x,
                NULL, NULL);
      Δh = 1
  }
      :
      :
  return (Δh) // at end
}
```

```
else if (x < t->data) {
    // go down left subtree
    if (insert(x, t->left)) {
        t->bal--; // left subtree increased
        if (t->bal == -1)
            oh = 1;
        else if (t->bal == -2) {
            if (t->left->bal == 1)
                rotate_right(t->left)
            rotate_left(t)
        }
    }
}
```

left child "right-heavy"

```
else if (x > t->data) {
    if (insert(x, t->right)) {
        t->bal++;
        if (t->bal == 1)
            oh = 1;
        else if (t->bal == 2)
            if (t->right->bal == -1)
                rotate_left(t->right);
            rotate_right(t);
    }
} else; // no-op, x == t->data
```

// duplicate

right child ←

left heavy

What about delete?

(erase(... x, t))

Alg:

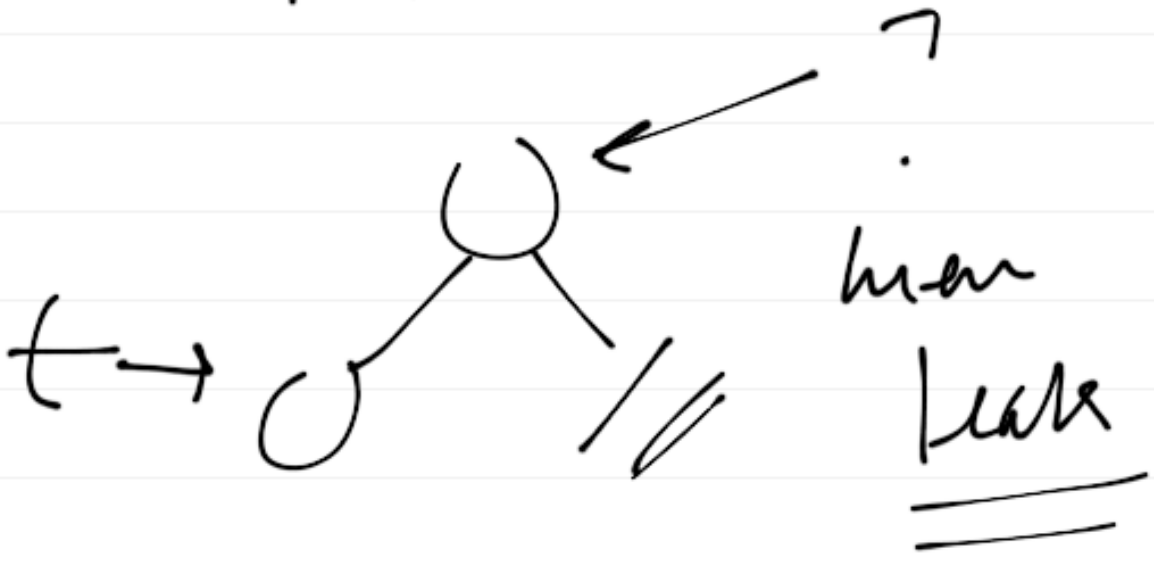
```
if (t == NULL)
    return ∅;
```

```
if (x < t->data) {
    if (erase(x, t->left)) {
        // almost like
        // insert(x, t->right)
        t->bal++; // height of
                  // left subtree
                  // increased
        if (t->bal == 0)
            dh = 1
        else if (t->bal == 2)
            if (t->right->bal == -1)
                rotate_left(t->right)
            rotate_right(t)
            if (t->bal == 0) dh = 1
```

```
else if (x > t->data)
  if (erase(x, t->right)) {
      // almost like
      // insert(x, t->left)
          .
          .
          .
      // for you to fill in
  }
}
```

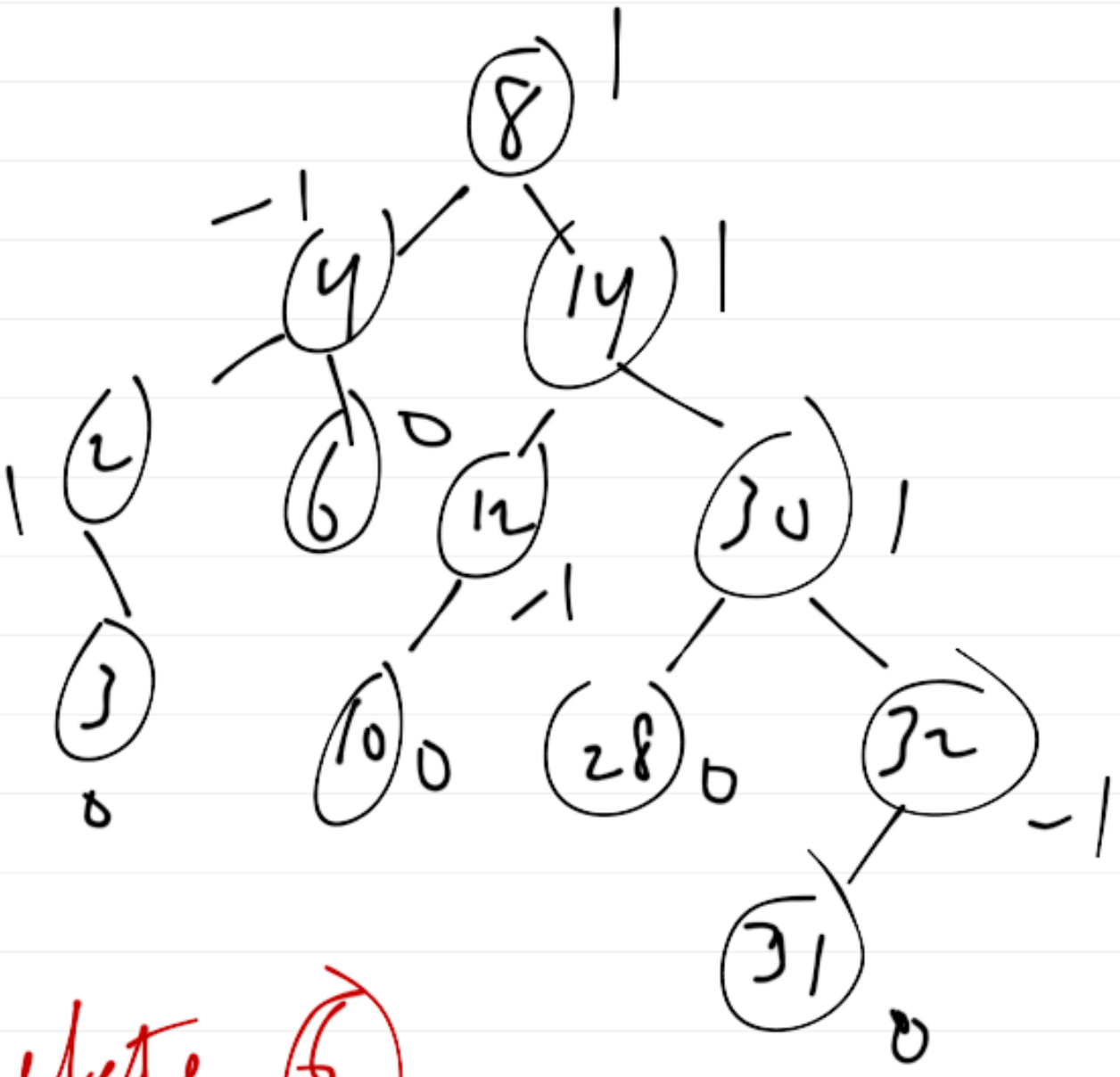
```
else if (t->left != NULL &&
         t->right != NULL)
  //x == t->data
  //want to delete t, t
  //has 2 children
  t->bal--;
  // almost like insert(x, t->left)
  if(t->bal == 0) dh = 1
  else if (t->bal == -2) {
      if(t->left->bal == 1)
          rotate_right(t->left)
      rotate_left(t)
  } else if(t->bal == 0) dh = 1
```

```
else { //one subtree NULL
    // set current node t
    // to non-NULL subtree
    node_t *old = t;
    t = (t->left != NULL) ?
        t->left : t->right;
    delete old; //calls node_t
                destructor
    dh = 1
}
return dh;
```

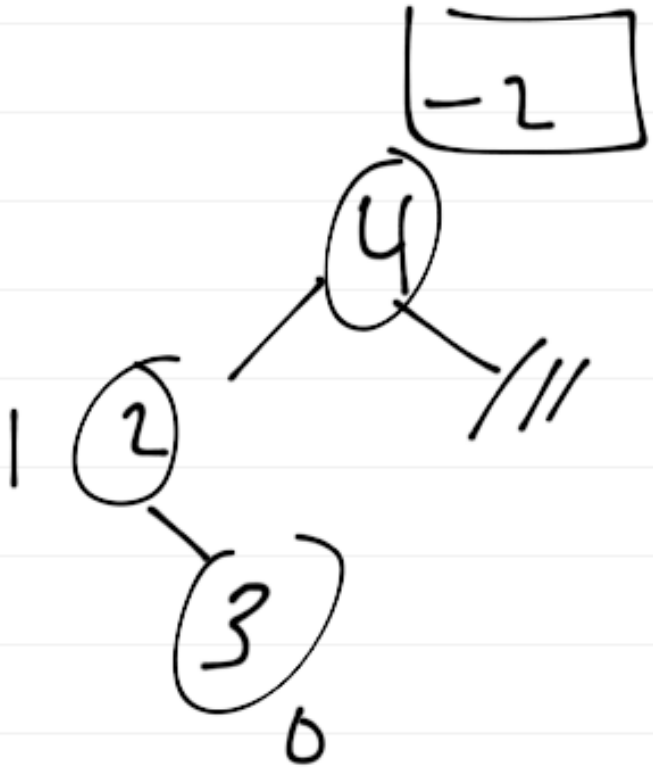


insert 6, 4, 2, 8, 10, 12, 14, 32,

30, 28, 1, 31

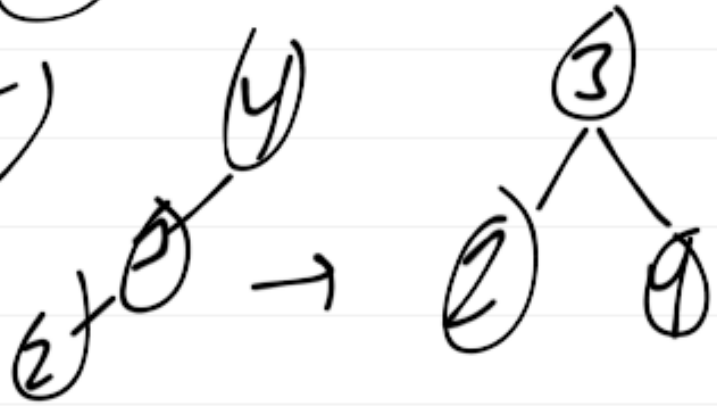


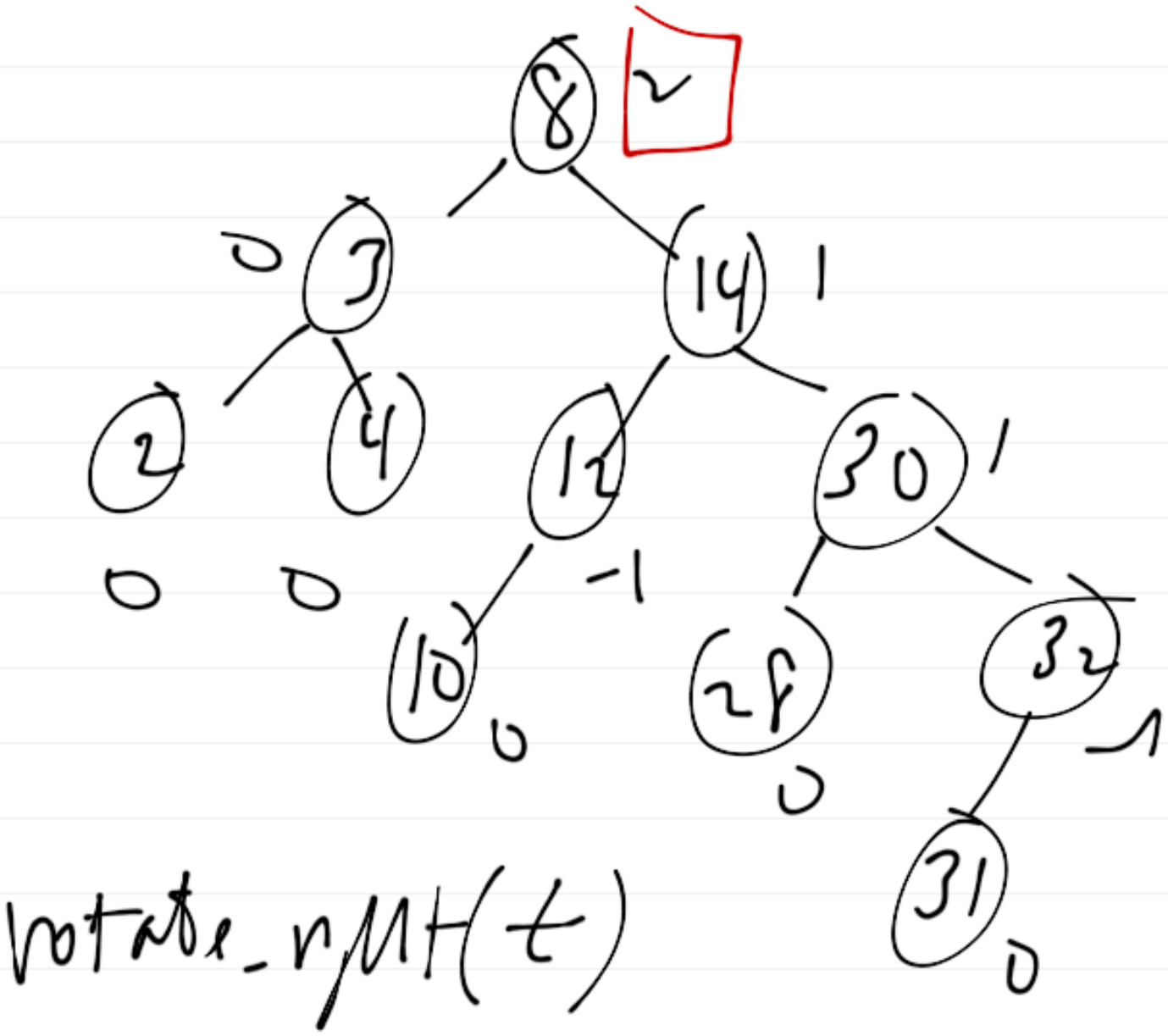
delete 6



rotate_right (t → Int)

rotate_left (t)





rotate_right(t)