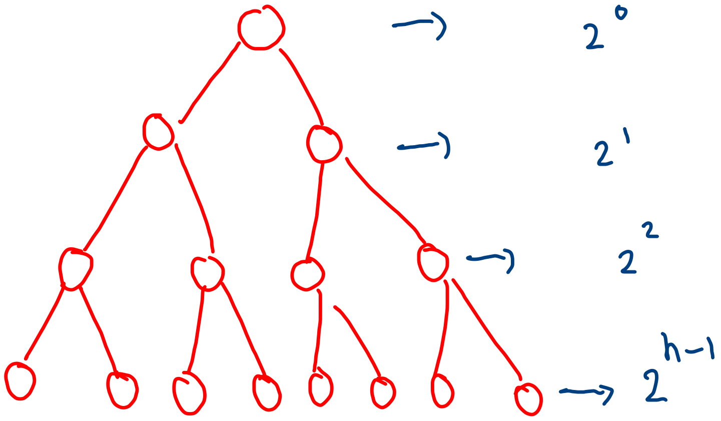
AVL: self-balancing BST



n: total nodes

h: height

gp 1, 2, 4, 8 . . . .

$a = 1, \quad r = 2$

$t = h$

$\text{sum} = \frac{a(r^t - 1)}{r - 1}$

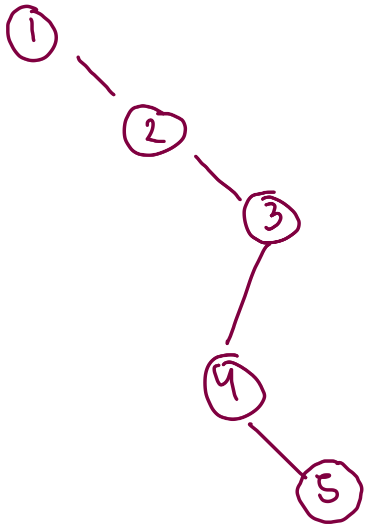
$n = 2^0 + 2^1 + 2^2 + \dots\dots 2^{h-1}$

$n = 1(2^h - 1)$

$n = 2^h - 1, \quad \log_2 n = \log_2 (2^h)$

$\log_2 n = h$

$\boxed{h \propto \log_2 n}$

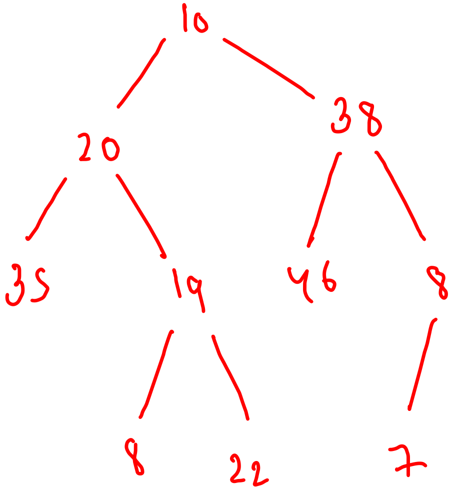


$h \approx n$

$$\boxed{\log_2 n \le h \le n}$$

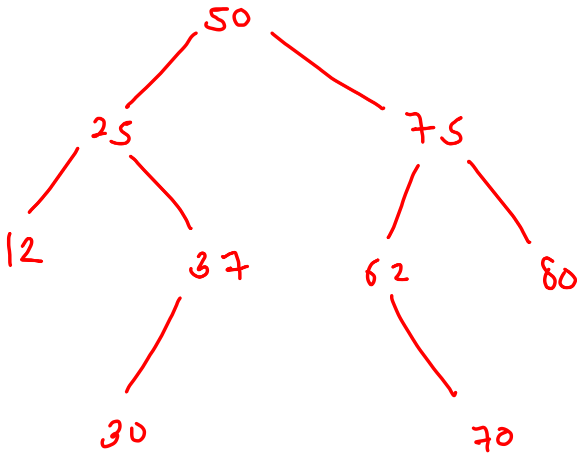
find (key)

Binary Tree



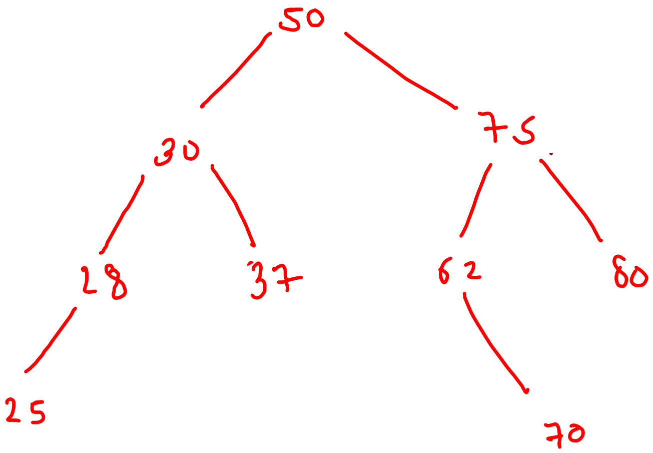
BT $T \propto o(n)$

Binary search tree



$T \propto o(h)$

$\log_2 n \leq h \leq n$



add(28)
remove(12)

$bf = lh - rh$

node is balanced

$-1 \le bf \le 1$

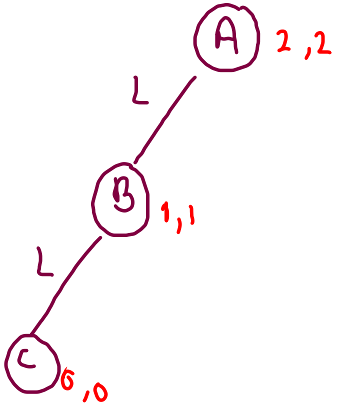
$-1, 0, 1$

ht: in terms of edges

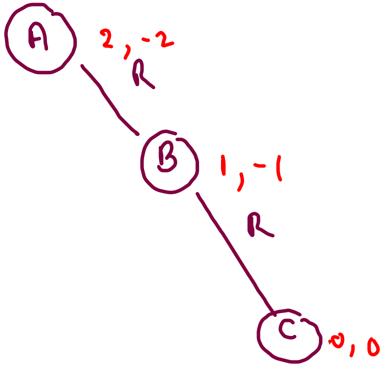
Problems:

(h,b)

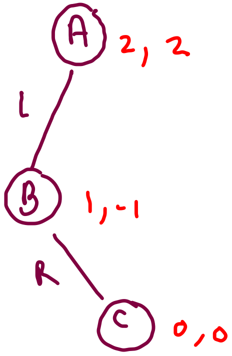
(i)



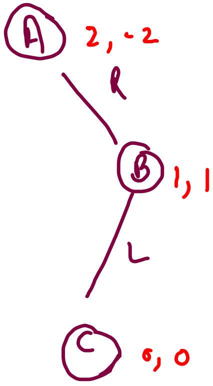
(ii)



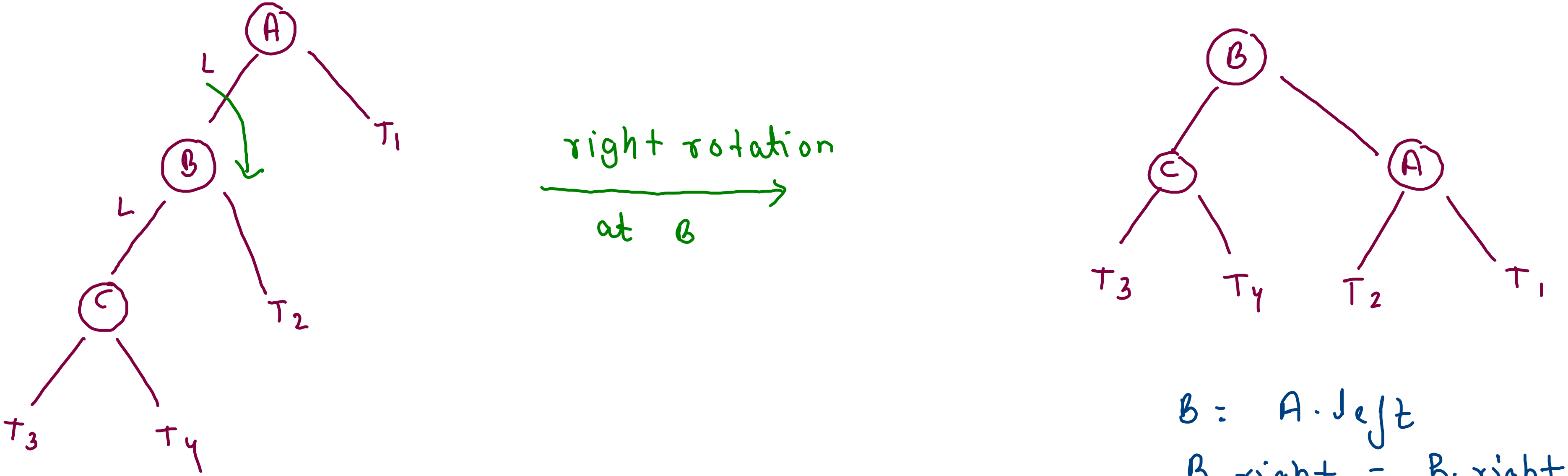
(iii)



(iv)



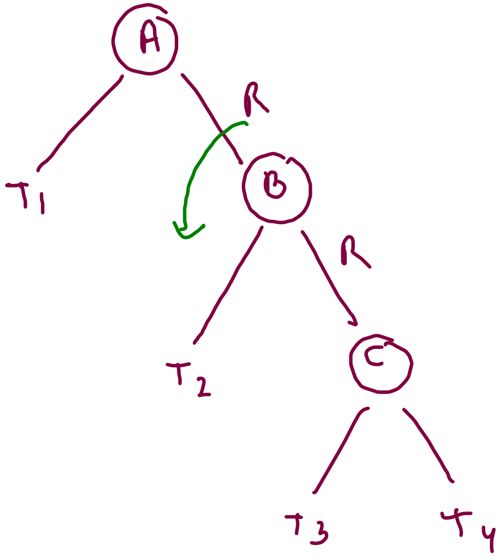
(i) LL



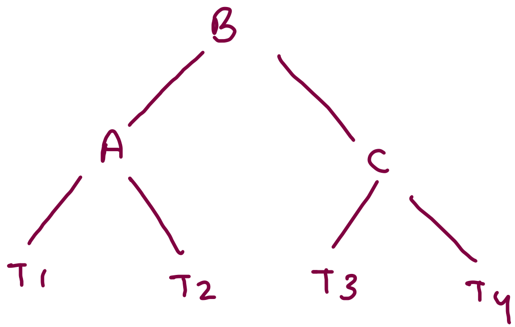
```
B = A.left
B_right = B.right
B.right = A
A.left = B_right
return B;
```

```
updateHt and Bal (A);  <——
updateHt and Bal (B);
```

(ii) RR

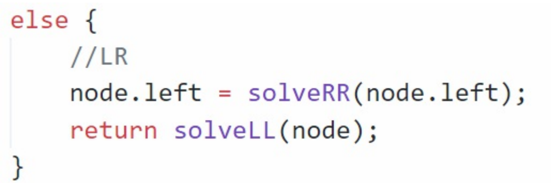


$\xrightarrow[\text{at B}]{\text{left rotation}}$

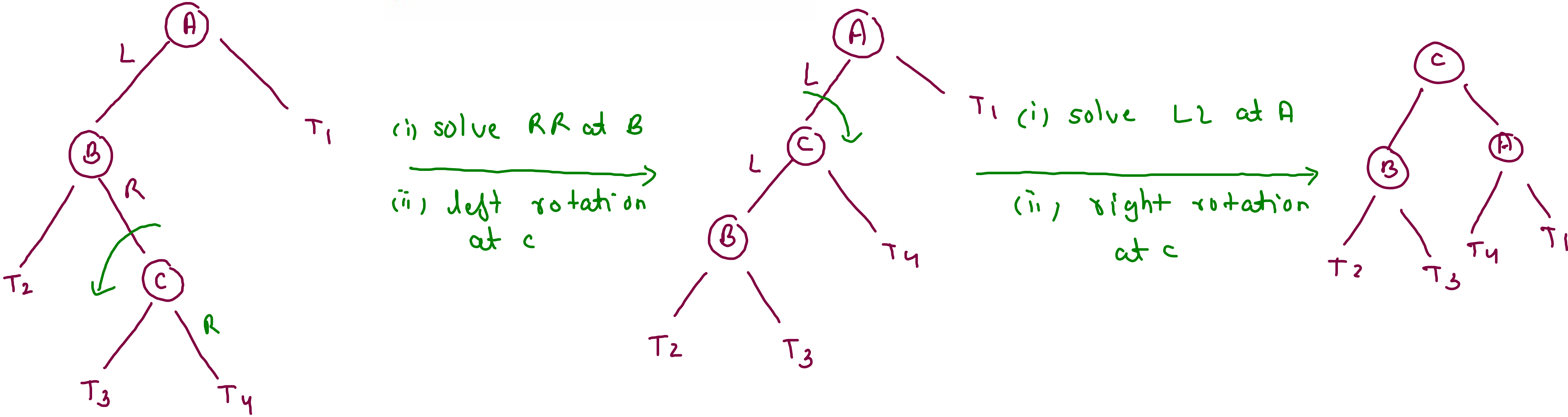


```
B = A.right
B_left = B.left
B.left = A
A.right = B_left

updateHTandBal (A);
updateHTandBal (B);

return B;
```

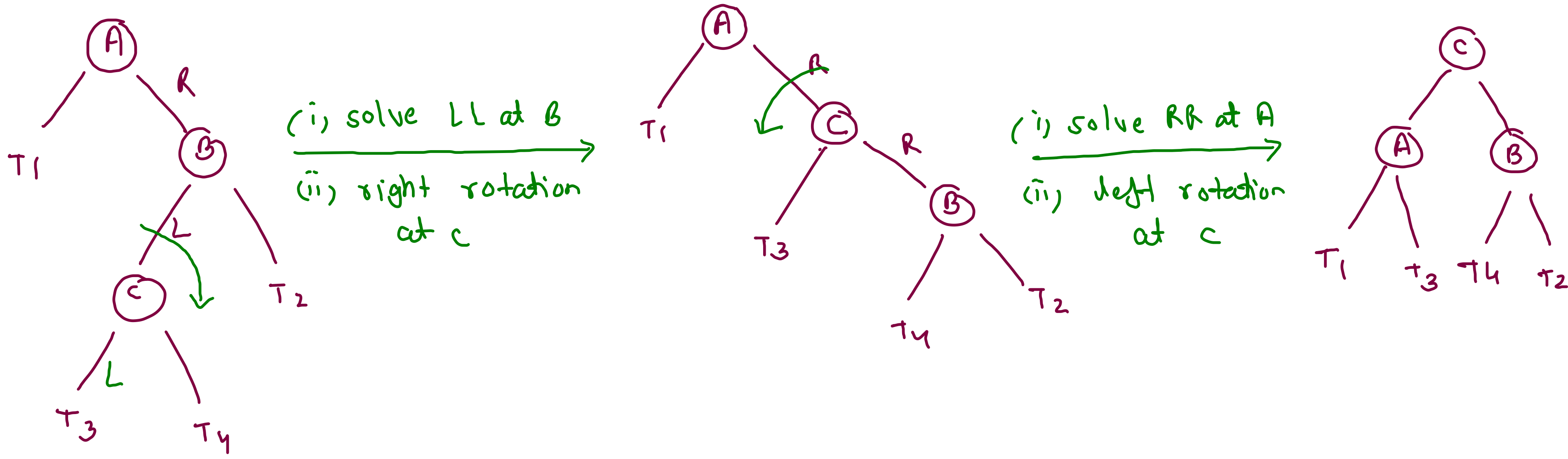
(iii) LR

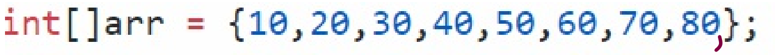


```
else {
    //LR
    node.left = solveRR(node.left);
    return solveLL(node);
}
```



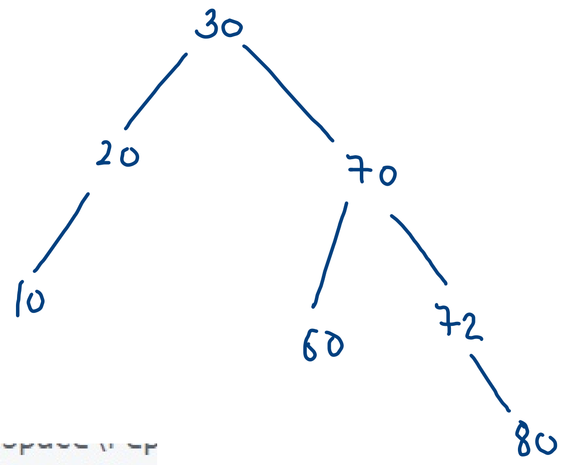
(iv) RL


A
R
T1
B
L
C
T2
L
T3
T4
(i) solve LL at B
(ii) right rotation at C
A
R
T1
C
R
T3
B
T4
T2
(i) solve RR at A
(ii) left rotation at C
C
A
B
T1
T3
T4
T2



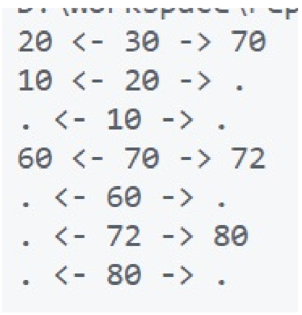
```java
int[]arr = {10,20,30,40,50,60,70,80};
```

, 72

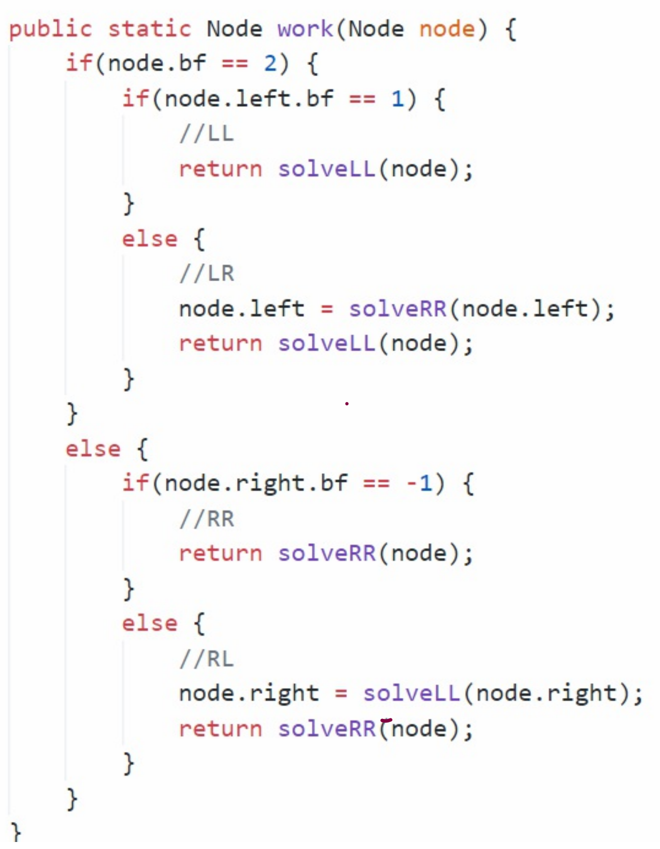


remove 40

remove 50



```
20 <- 30 -> 70
10 <- 20 -> .
. <- 10 -> .
60 <- 70 -> 72
. <- 60 -> .
. <- 72 -> 80
. <- 80 -> .
```



```java
public static Node work(Node node) {
    if(node.bf == 2) {
        if(node.left.bf == 1) {
            //LL
            return solveLL(node);
        }
        else {
            //LR
            node.left = solveRR(node.left);
            return solveLL(node);
        }
    }
    else {
        if(node.right.bf == -1) {
            //RR
            return solveRR(node);
        }
        else {
            //RL
            node.right = solveLL(node.right);
            return solveRR(node);
        }
    }
}
```