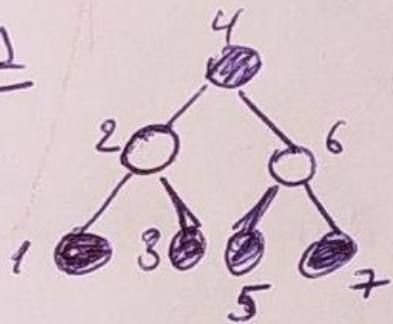


P.2

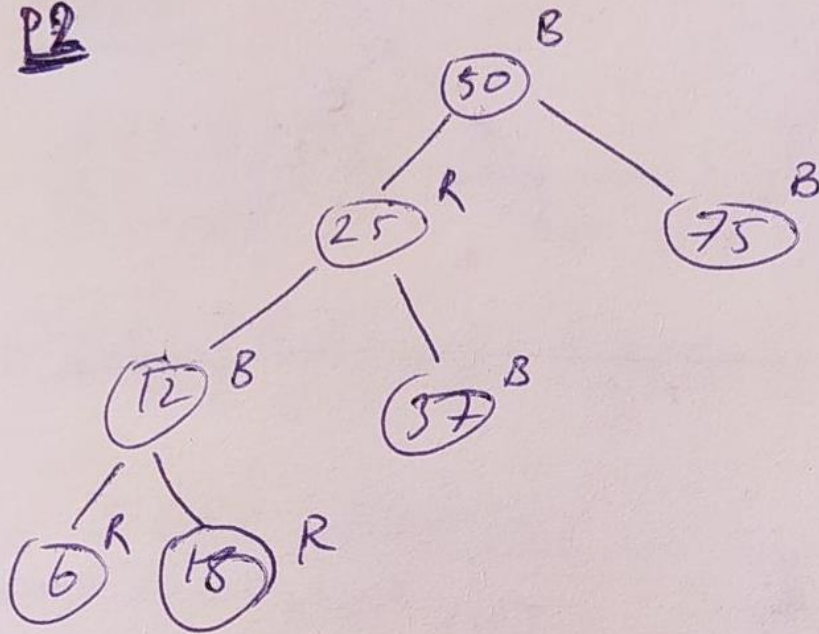


This Red-Black tree
is a valid tree but
we cannot obtain

this tree with any insertion sequence.

The last inserted node must be
a red ~~leaf~~ leaf node. But in the
tree above there is not a single red
leaf node.

P2



This is a valid Red-Black tree
from the lecture (lecture-6, P.25)
But it does not suffice the AVL
Tree requirements.

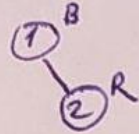
Prob. 3.

(a) 1, 2, 3, 4, 5, 6, 7, 8

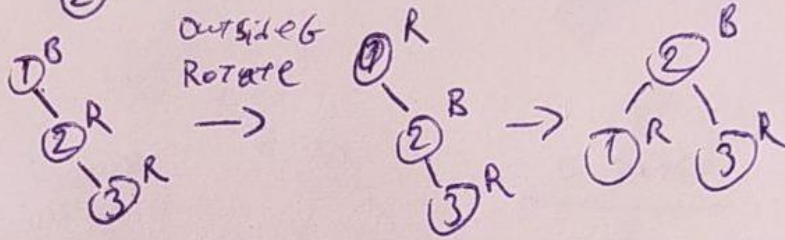
Insert-1:



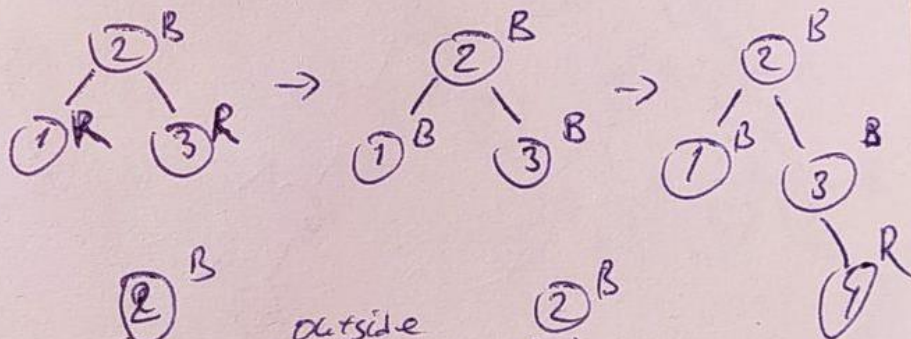
Insert-2:



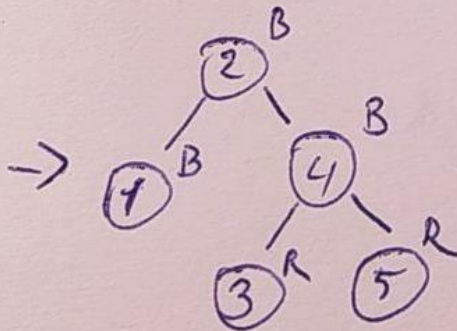
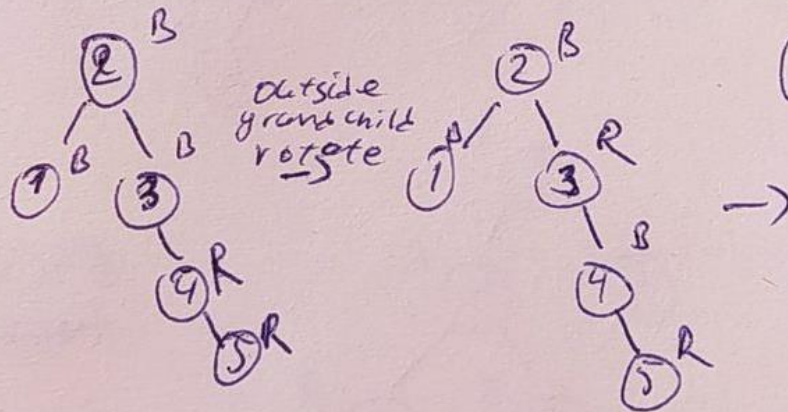
Insert-3:



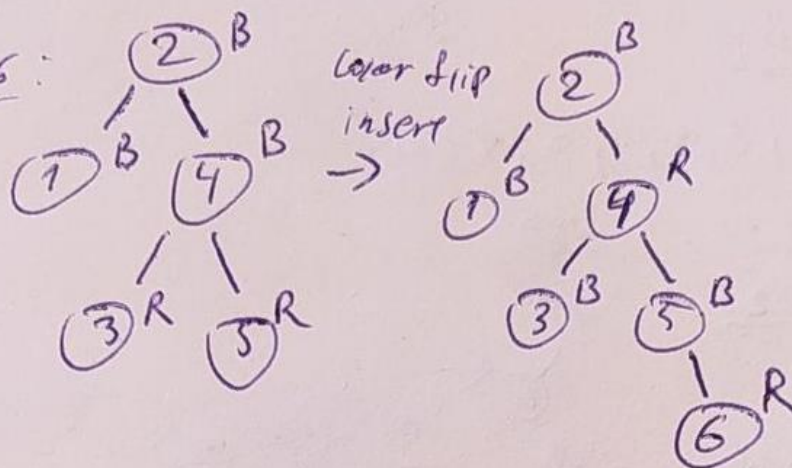
Insert-4:



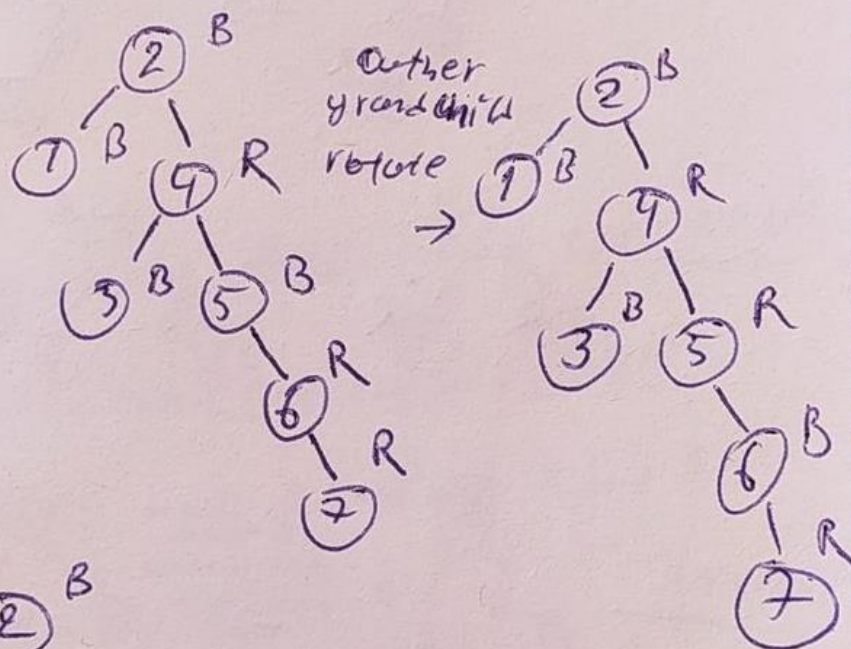
Insert 5:



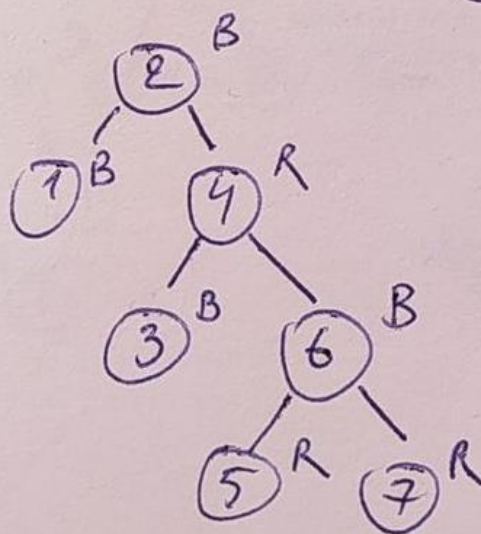
insert-6:



insert-7:

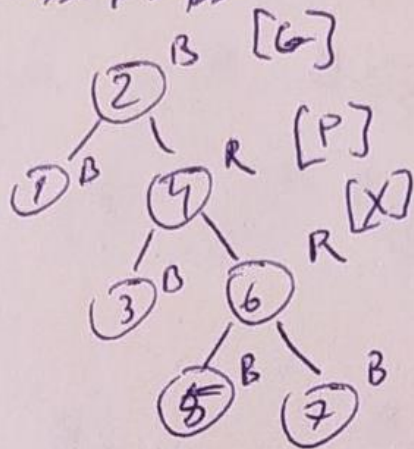


→

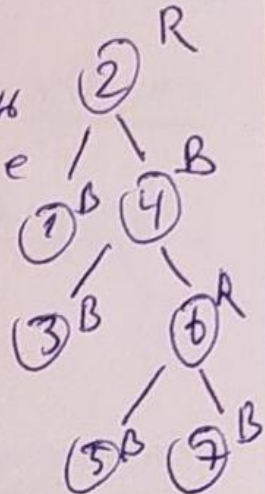


$Bh(n) = 2$
~~for every n~~

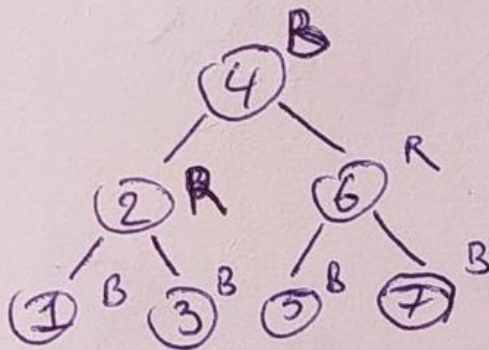
insert-8:



Outer grandchild rotate
→

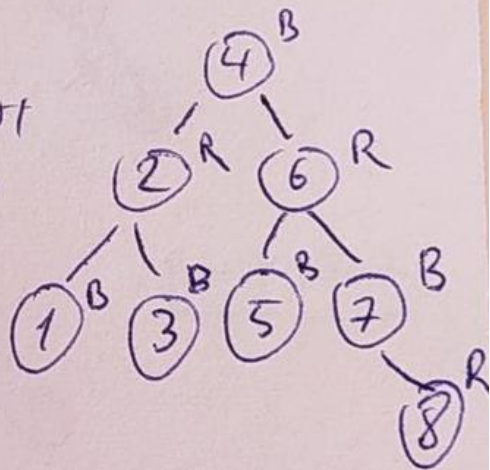


→



insert

→



(b) 3, 2, 1, 4, 5, 6

insert-3: (3)^B

insert-2:

```
graph TD
    2((2)) -- R --> 3((3))
    style 2 fill:#fff,stroke:#000,stroke-width:1px
    style 3 fill:#fff,stroke:#000,stroke-width:1px
```

insert-1:

```
graph TD
    2((2)) -- R --> 1((1))
    2 -- R --> 3((3))
    style 2 fill:#fff,stroke:#000,stroke-width:1px
    style 1 fill:#fff,stroke:#000,stroke-width:1px
    style 3 fill:#fff,stroke:#000,stroke-width:1px
```

insert-4:

```
graph TD
    2((2)) -- B --> 1((1))
    2 -- B --> 3((3))
    3 -- R --> 4((4))
    style 2 fill:#fff,stroke:#000,stroke-width:1px
    style 1 fill:#fff,stroke:#000,stroke-width:1px
    style 3 fill:#fff,stroke:#000,stroke-width:1px
    style 4 fill:#fff,stroke:#000,stroke-width:1px
```

insert-5:

```
graph TD
    2((2)) -- B --> 1((1))
    2 -- B --> 4((4))
    4 -- R --> 3((3))
    4 -- R --> 5((5))
    style 2 fill:#fff,stroke:#000,stroke-width:1px
    style 1 fill:#fff,stroke:#000,stroke-width:1px
    style 4 fill:#fff,stroke:#000,stroke-width:1px
    style 3 fill:#fff,stroke:#000,stroke-width:1px
    style 5 fill:#fff,stroke:#000,stroke-width:1px
```

insert-6:

(color flip) →

```
graph TD
    2((2)) -- B --> 1((1))
    2 -- R --> 4((4))
    4 -- B --> 3((3))
    4 -- B --> 5((5))
    5 -- R --> 6((6))
    style 2 fill:#fff,stroke:#000,stroke-width:1px
    style 1 fill:#fff,stroke:#000,stroke-width:1px
    style 4 fill:#fff,stroke:#000,stroke-width:1px
    style 3 fill:#fff,stroke:#000,stroke-width:1px
    style 5 fill:#fff,stroke:#000,stroke-width:1px
    style 6 fill:#fff,stroke:#000,stroke-width:1px
```


4. The algorithm that I'm proposing uses only a Binary search. Hence the running time of the algorithm is $O(\log N)$ which is $o(N)$

Algorithm Problem 4 (Arr):

input: Sorted array of distinct integers

output: is there any ~~integer~~ m that
 $arr[m] = m$

$L \leftarrow 0$

$R \leftarrow arr.length$

While $L < R$:

$mid \leftarrow (L + R) / 2$

if $arr[mid] == mid$ then
return true

if $arr[mid] < mid$ then

$L \leftarrow mid + 1$

else

$R \leftarrow mid - 1$

return false