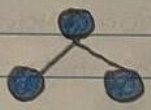
# Lab-6

problem 1 : derivable → if it is obtained from an insertion sequence of node.

→ Starting from an empty tree show that a given tree that has R-B Tree property but is not derivable.

Solution : a 3-node tree will all it's nodes black is not derivable from the Insertion algorithm.



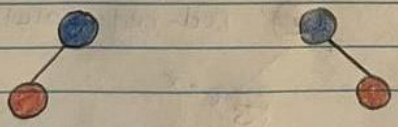
proof :

Adding one ~~node~~ element to tree

- root is black

Adding second element.

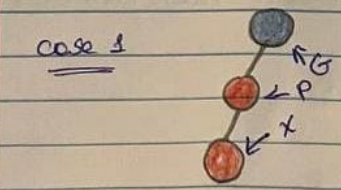
- new node is red.

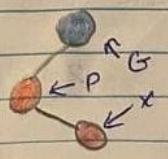


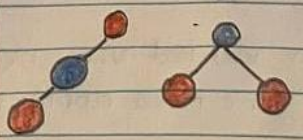
∴ So, this is correct.

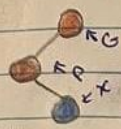
Adding 3rd element.

case 1



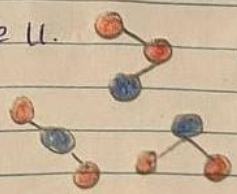



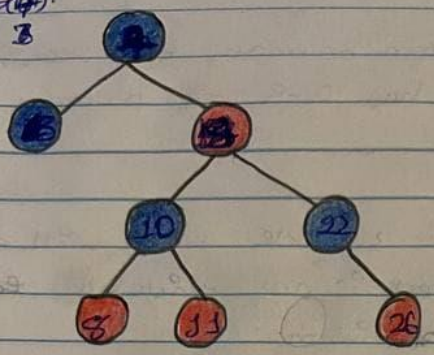



case II

case II.



problem 2: Create a red-Black tree that doesn't satisfy the AVL - Balance condition.




- This red Black Tree is a valid R-B Tree But not valid AVL - tree Because Left subtree is two levels shorter than the right subtree.

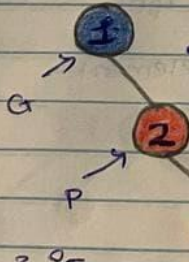
problem 3. use the insertion Algorithm for red-black trees to successively insert the following nodes, start with an empty tree

a. 1, 2, 3, 4, 5, 6, 7, 8.

1- insert (1)


2- insert (2)


3- insert (3)

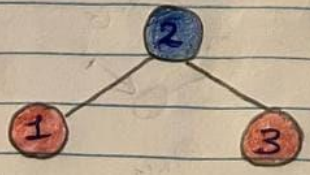



- here we encounter a red-red violation.

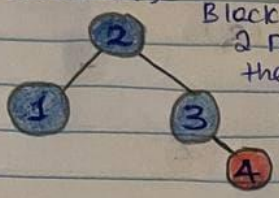
Since 3 is an outside child

step1: change G & P's color

step2: Rotate P.G in direction that lift up x

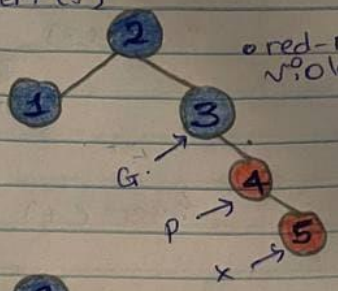


4- insert (4)



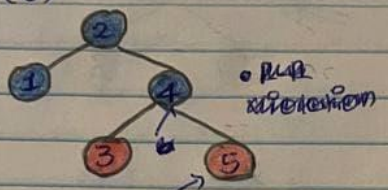

Black node with 2 red children the change color

5- insert (5)

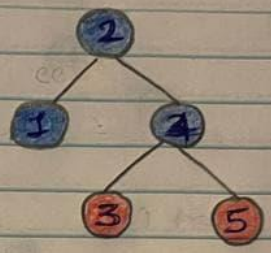



• red-red violation

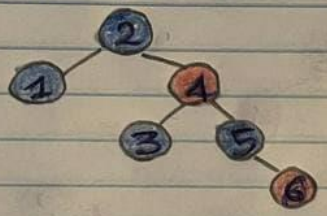
6 - insert (6)



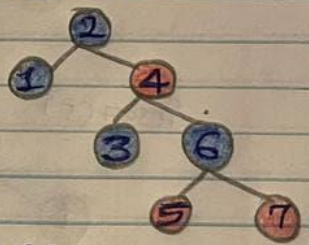

• Red violation



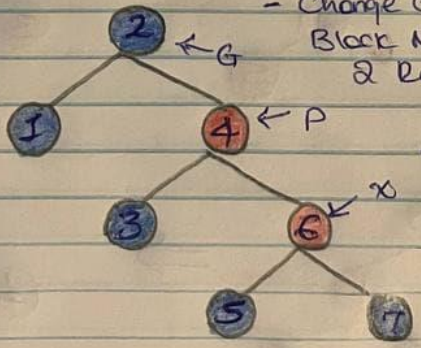

• Black node with ~~2 red~~ nodes will switch color.



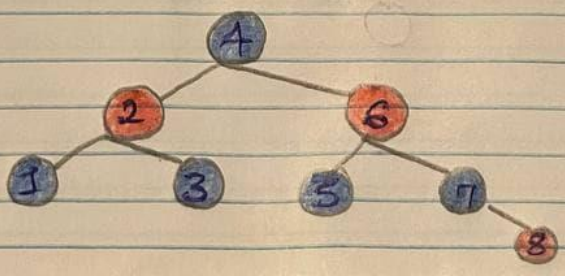

7- insert (7)




8- insert (8)




- Change color if Black Node has 2 Red children
- Red-Red violation happened.
- change color of G, P
- Rotate P, G left x.

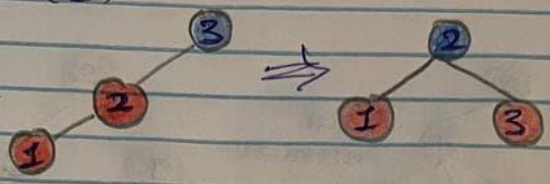


B - 3, 2, 1, 4, 5, 6.

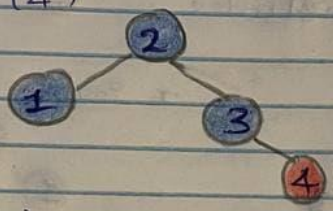
insert (3)


insert (2)

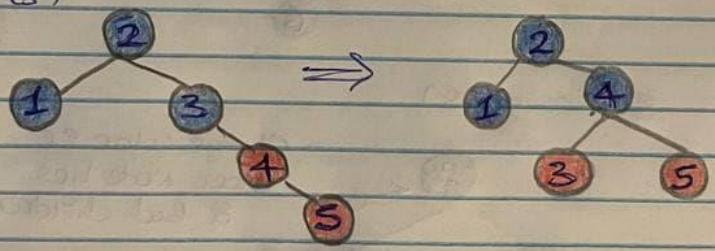

insert (1)



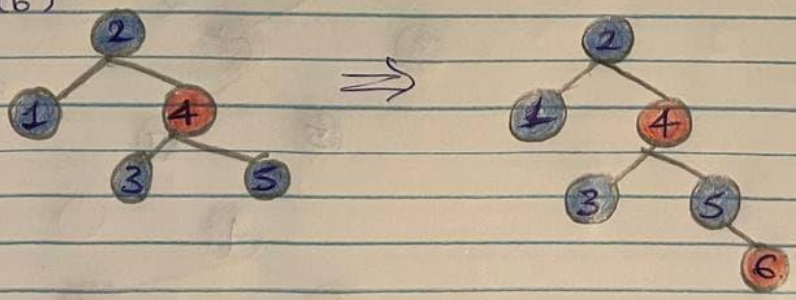

insert (4)




insert (5)




insert (6)



problem 4 : o(n) time to determine whether a sorted Array A contains element m for which A[m] = m

we can enhance and use Binary search Tree and we are guaranteed to have logn steps to find our result.

```
Algorithm binaryEnhanced ( A, low, high)
Input  Sorted Array of elements
output true if A[m] = m false otherwise
  if low <= high then
    mid ← (low+high)/2
    if mid = A[mid] then return true
    if mid > A[mid]
      return binaryEnhanced (A, mid+1, high)
    else
      return binaryEnhanced (A, low, mid-1)
  return false
```

proof that your algorithm runs in o(n) time.

Best case if A[mid] = mid the first trial.

```
if low <= high      +1
  mid ← (low + high)/2      +3

  if mid = A[mid] then      +2
    return true             +1
```

T(d) = 7

worest case if we didn't find element that satisfy this property.

if low <= high +1

mid ← $\frac{low + high}{2}$ +3

if mid = A[mid] then +2

return true

if mid > A[mid] +2

return binaryEnhanced(A, mid+1, high)

else

return binaryEnhanced(A, low, mid-1)

$T(n/2)$

$T(n) = T(n/2) + 8$

$$T(n) \begin{cases} 7 & \text{if found first at mid} \\ T(n/2) + 8 & \text{otherwise.} \end{cases}$$

$a = 1 \quad k = 0$

$b = 2 \quad d = 7 \quad c = 8$

$a = b^k \Rightarrow$ from master formula

this is $\Theta(n^k \log n)$

$n^0 \log n \Rightarrow \Theta(\log n)$

$\Theta(\log n)$ is $o(n)$ Because $\log n$ is strictly less than $n$

$\leq$