Master's Theorem:

$$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$$

$a \geq 1$, $b > 1$, $k \geq 0$, and p is any real no.

Case 1: If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

Case 2: If $a = b^k$ then

(a) if $p > -1$ then, $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

(b) if $p = -1$ then, $T(n) = \Theta(n^{\log_b a} \log\log n)$

(c) if $p < -1$ then, $T(n) = \Theta(n^{\log_b a})$

Case 3: If $a < b^k$ then

(a) if $p \geq 0$ then $T(n) = \Theta(n^k \log^p n)$

(b) if $p < 0$ then $T(n) = \Theta(n^k)$

$T(n) = 2T(n/2) + n/\log n$

$T(n) = aT(n/b) + \Theta(n^k \log^p n)$

$a=2,\ b=2,\ k=1,\ p=-1.\ ,\ b^k = 2^1 = 2.$

$a = b^k$ $\begin{cases} p<-1 \\ p>-1 \\ \boxed{p=-1} \end{cases}$ — $T(n) = \Theta(n^{\log_b a} \cdot \log\log n)$

$= \Theta(n^{\log_2 2} \cdot \log\log n)$

$= \Theta(n \log\log n)$

$T(n) = 2T(n/2) + n\log n =$

$a=2,\ b=2.\ \ n^{\log_b a} = n^{\log_2 2} = \boxed{n}$

$n^{\log_b a}$ $\boxed{\text{Comp}}$ $f(n)$ → polynomial (It does not contain log)

$\underline{n^{\log_b a}} = f(n)\ [n]$

$\boxed{n = n}$

$T(n) = \Theta(n^{\log_b a} \log n \cdot \log n)$

$= \underline{\Theta(n \log^2 n)}$

$$T(n) = 2T(\sqrt{n}) + 1 \longrightarrow (1)$$

We cannot apply master's theorem.

→ This question can be solved by method of changing variables.

Let $n = 2^m \Rightarrow m = \log_2 n$

$$T(2^m) = 2T(2^{m/2}) + 1$$

Let $S(m) = T(2^m)$

$$S(m) = 2S(m/2) + 1 \longrightarrow (2)$$

Master's theorem applies here

$a = 2$, $b = 2$, $m^{\log_b a} = m$

$$S(m) = \Theta(m)$$

$$\boxed{T(n) = \Theta(\log_2 n)}$$

$$T(n) = 2T(\sqrt{n}) + n$$

$$T(n) = 2T(\sqrt{n}) + \log n$$

How to identify time complexity of recursive & iterative Programs:-

```
retsum( )
{
    int sum:=0;
    for (i=0; i<=n; i++)
        sum = sum+i;
    return(sum);
}
```

1

n+2

n+1

1

$\boxed{2n+5}$

$\underline{\underline{\Theta(n)}}$

if for loop is given

↳ 1 [0 to n] → n

↳ 2 < 0 to n, 0 to n → n^2

```
A( )
{
    i=1;
    s=1;
    while (s≤n)
    {
        i++;
        s=s+i;
        printf("Hello");
    }
}
```

$\Theta(\sqrt{n})$

$i = 1, 2, 3, 4 \ldots k$

$s = 1, 3, 6, 10 \ldots \frac{k(k+1)}{2}$

$\frac{n(n+1)}{2}$

$\vdots$

$\frac{k(k+1)}{2} = n$

$k^2 + k = 2n$

$k^2 = n$ $\quad \boxed{k = \sqrt{n}}$

```
A( )
{
    for (i=1; i^2<=n; i++)
    {
        printf("Hello");
    }
}
```

$O(\sqrt{n})$

```
prog5( )
{
  int i,j,k,n;
  for(i=1; i<=n; i++)
  {
    for(j=1; j<=i; j++)
    {
      for(k=1; k<=100; k++)
      {
        printf("Hello students");
      }
    }
  }
}
```

$O(n^2)$

```
Prog6( )
{
  int i,j,k,n;
  for(i=1; i<=n; i++)
  {
    for(j=1; j<=i^2; j++)
    {
      for(k=1; k<=n/2; k++)
      {
        printf("1234");
      }
    }
  }
}
```

i → n times

j → n^2 times

k → $n/2$ times

$n * n^2 * n/2 = O(n^3)$

```
Prog 7( )
{ int i,j,k;
  for(i=n/2; i<=n; i++)          ---> n/2 times
  {
    for(j=1; j<=n/2; j++)        ---> n/2 times
    {
      for (k=1; k<=n; k=k*2)     ---> log n times
        { printf("Good Morning");
        }
    }
  }
}
```

$$T(n) = n/2 * n/2 * \log n$$
$$= (n^2 \log_2 n)$$

```
prog8( )
{ int i,j,k;
  for(i=1; i<=n/2; i++)          ---- n/2 times
  {
    for(j=1; j<=n; j=2*j)        ---> log n times
    {
      for(k=1; k<=n; k=k*2)      ---> log n times
        { printf("Good Morning");
        }
    }
  }
}
```

$$O(n(\log n)^2)$$

```
prog9( )
{
  while (n>1)
  {
    n = n/2;
    printf("Good Morning");
  }
}
```

$O(\log n)$

```
Prog10( )
{   int k=1;
    int n = 2^k;
    for (i = 1; i<= n; i++)
    {
      j= 2;
      while (j<= n)
      {
        j= j^2;
        printf("Good Morning");
      }
    }
}
```

K=1	K=2	K=3
$n=4$	$n=16$	$n=2^{2^3}=2^8$
$j=\underbrace{2,4}$	$j=\underbrace{2^1,2^2,2^4}$	$j=2^1,2^2,2^4,2^8$
$n*2$ time	$n*3$ time	$n*4$ time

$$\underline{\underline{n(K+1)}}$$

$$=O(n(\log\log n))$$

$$n = 2^{2^K} \Rightarrow \log n = 2^K$$

$$\log\log n = K$$

006, 005, 021, 020, 043, 013, 019, 046, 053, 5009, 070,
054, 267, 059, 038, 048, 65007.

Finding Time Complexity of recursive Programs.

```
A(n)
{
  if (n>1)
    return (A(n-1));
}
```

$\Rightarrow$ if $n=1$, $T(n)=1$

if $n>1$ $\quad T(n) = 1 + T(n-1)$

$$T(n) = \begin{cases} T(n-1)+1 & ; \text{ if } n>1 \\ 1 & ; \text{ if } n=1 \end{cases}$$

Back substitution Method:

$$T(n) = T(n-1)+1 \longrightarrow (1)$$

place $n-1$ in place of n

$$T(n-1) = T(n-2)+1 \longrightarrow (2)$$

place $(n-2)$ in place of n in eq^n (1)

$$T(n-2) = T(n-3)+1$$

$$T(n) = (T(n-2)+1)+1$$

$$T(n) = ((T(n-3)+1)+1)+1$$

$$T(n) = T(n-k) + \underbrace{1+1+1+\cdots 1}_{k \text{ times}}$$

$$1$$

let $n-k = 1$

$k = (n-1)$

$$T(n) = 1+1+1 - - \cdots n \text{ times}$$

$$= O(n)$$

$$T(n) = \begin{cases} T(n-1)+n & ; n>1 \\ 1 & ; n=1 \end{cases} \quad O(n^2)$$

Recursion Tree Method

$$T(n) = \begin{cases} 2T(n/2)+c & ; n>1 \\ c & ; n=1 \end{cases}$$

$$T(n) = \boxed{2T(n/2)} + c \quad ; \quad n > 1$$

$$T(n/n) = T(1) = c \quad ; \quad n = 1$$

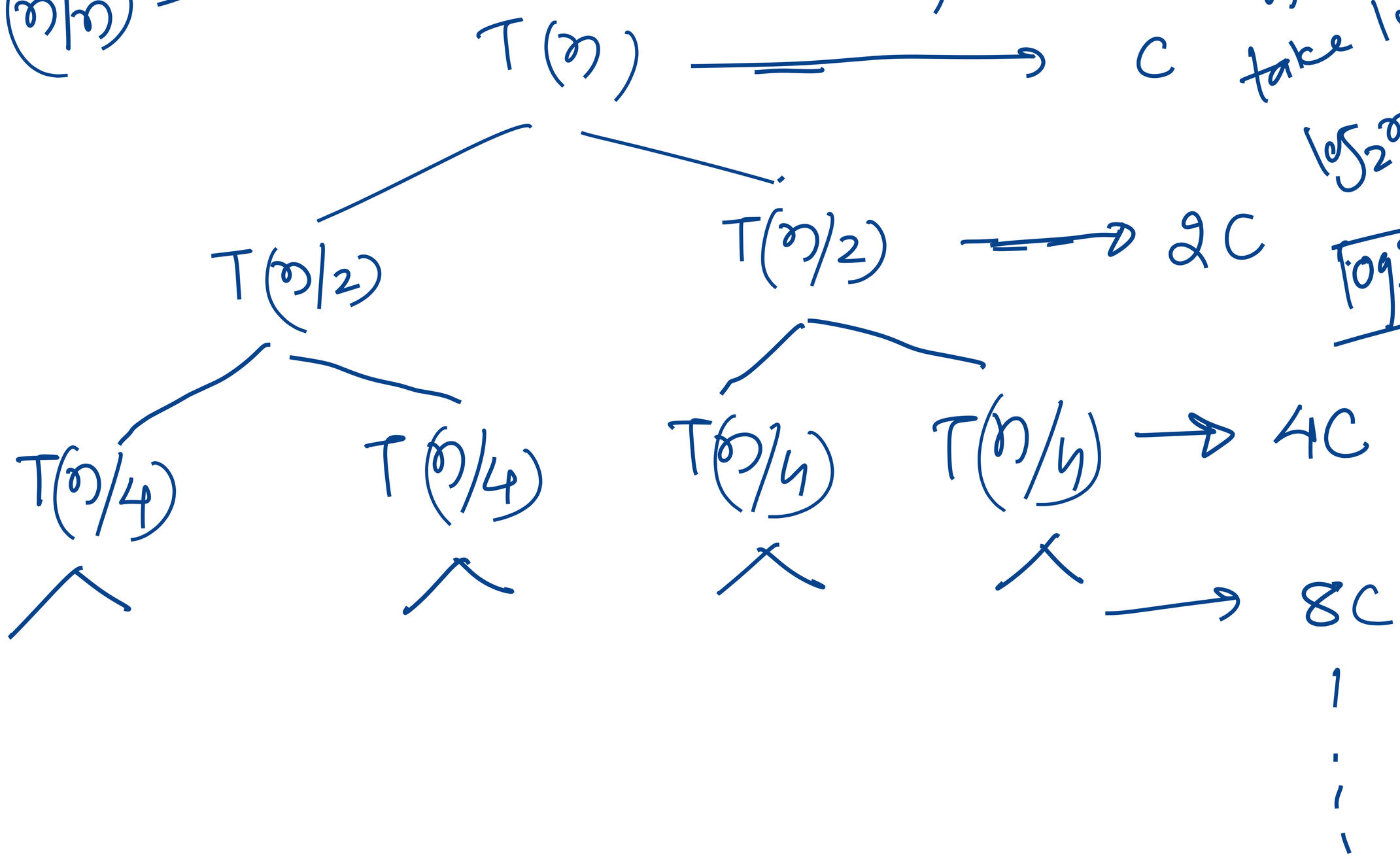
Let $n = 2^k$

take log both sides

$\log_2 n = \log_2 2^k$

$\boxed{\log_2 n = k \log_2 2}$

$\log_2 n = k$



$$C + 2C + 4C + 8C - - - - - C \cdot 2^k$$

$$C(2^0 + 2^1 + 2^2 + - - - - - - 2^k) \rightarrow \underline{\underline{GP}}$$

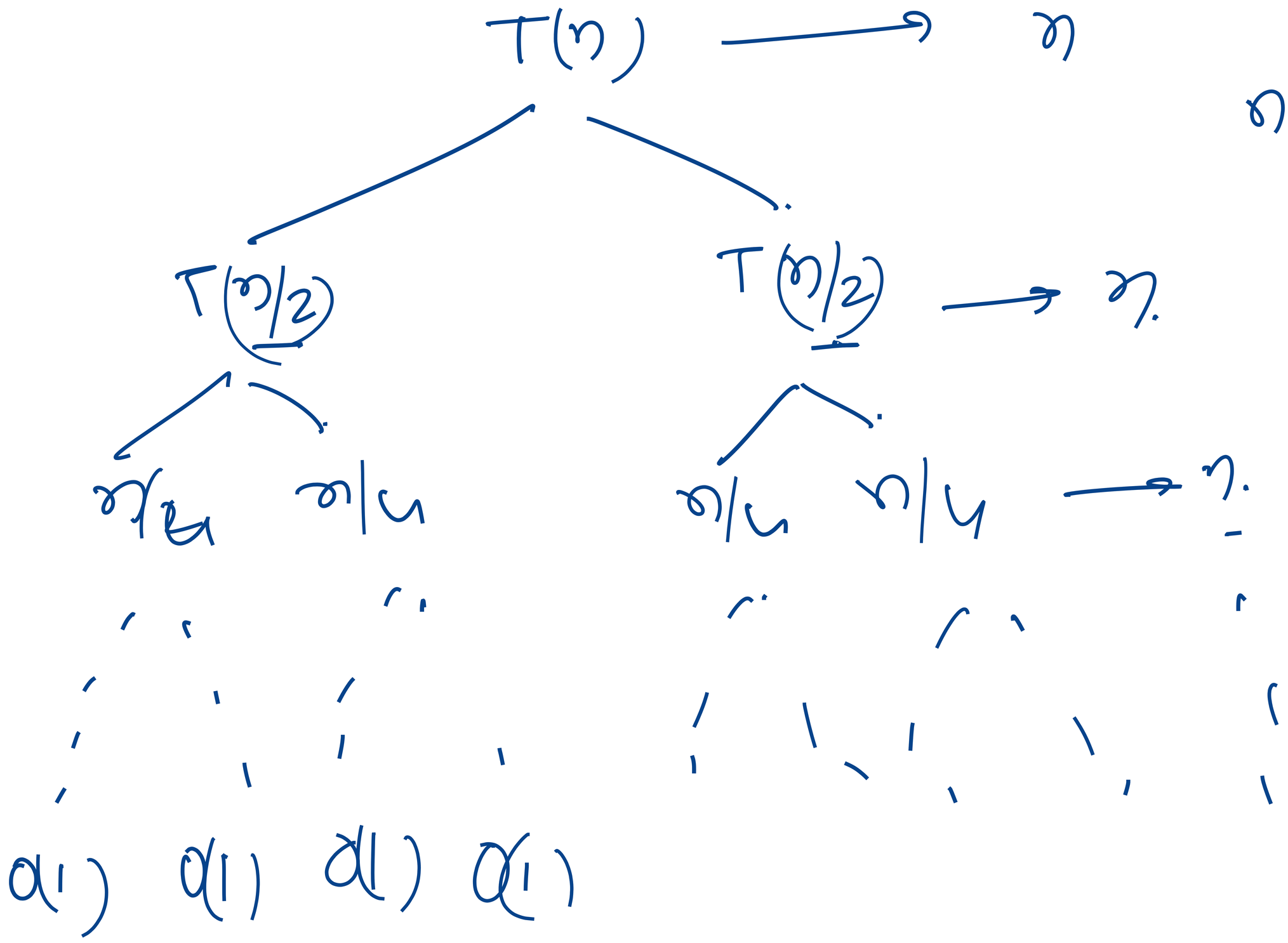
$$C\left(\frac{2^{k+1} - 1}{2 - 1}\right) = C\left(2^{k+1} - 1\right)$$

$$C = (2^k \cdot 2 - 1)$$

$$C = (n \cdot 2 - 1) = (2n - 1)$$

$$T(n) = \underline{\underline{O(n)}}$$

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



$n = 2^k$

$k = \log n$

$n(1+1+1 - \text{———} - - k \text{ time})$

$= n(\log n)$

$T(n) = O(n \log n)$.

$$T(n) = T(n/5) + T(4n/5) + n$$

$$\boxed{T(n) = O(n \cdot \log_{5/4} n)}$$

Comparison of various functions:

$f_1 \gtrless f_2$ $\qquad f(n) = 2^n \qquad g(n) = n^2$

Log method:

$f(n) \qquad g(n)$

taking log both sides

$\log f(n) \qquad \log g(n)$

$\log(2^n) \qquad \log n^2$

$n \cancel{\log_2 2}^{1} \qquad 2\log_2 n$

$n \quad > \quad \log_2 n$

$f_1(n) = 2^{\sqrt{n}} \qquad f_2(n) = n^{\log n}$

$f_1(n) = 2^n, \quad f_2(n) = n^{3/2}$

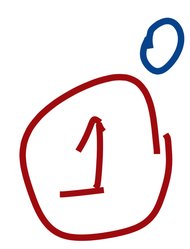
AVL Tree

↳ Binary Search Tree consists of some restrictions.

* Balance factor = Height of LST − Height of RST
* Balance factor can be either 0, 1 or −1.
* AVL Tree is also called as Height Balanced tree.

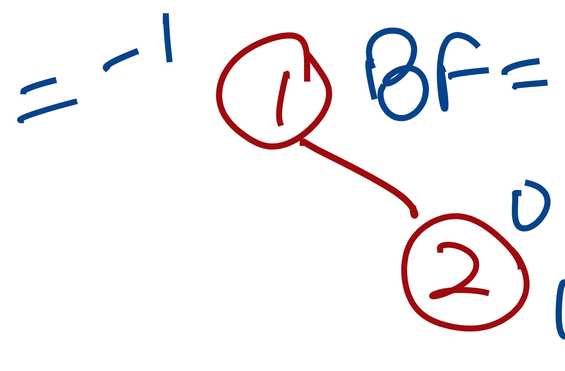
Ex: 1, 2, 3, 4, 5, 6, 7, 8, 9.

Insert 1.



BF = 0 − 0 = 0

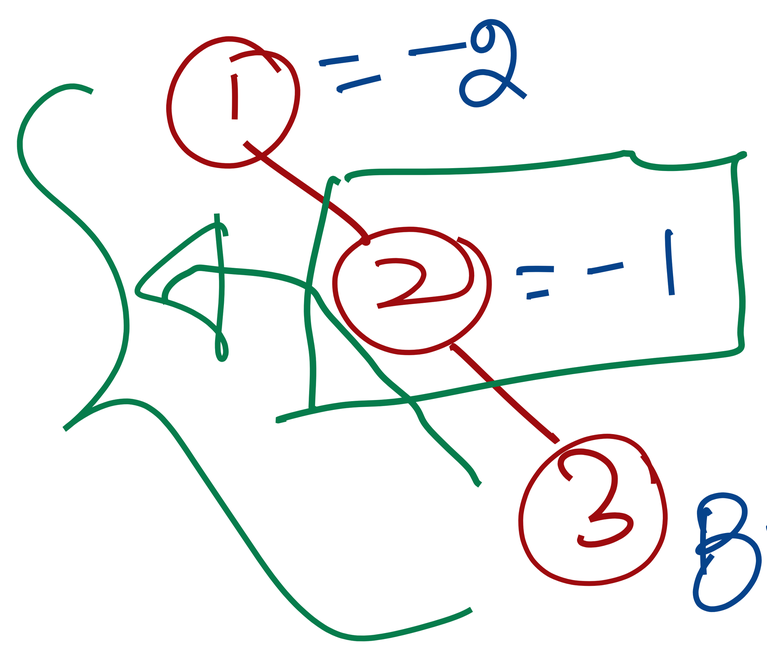
Insert 2.



BF = 0 − 1

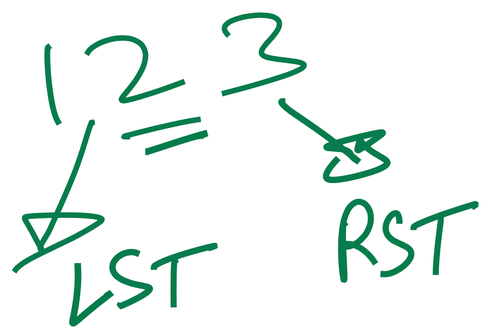
BF = 0 − 0 = 0

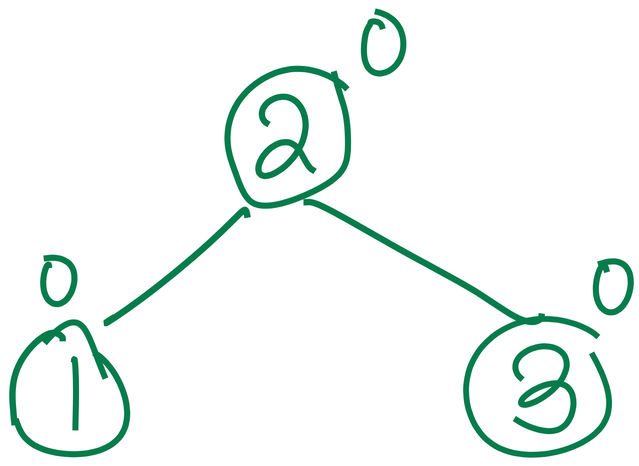
Insert 3.



Root Node

Imbalanced AVL Tree





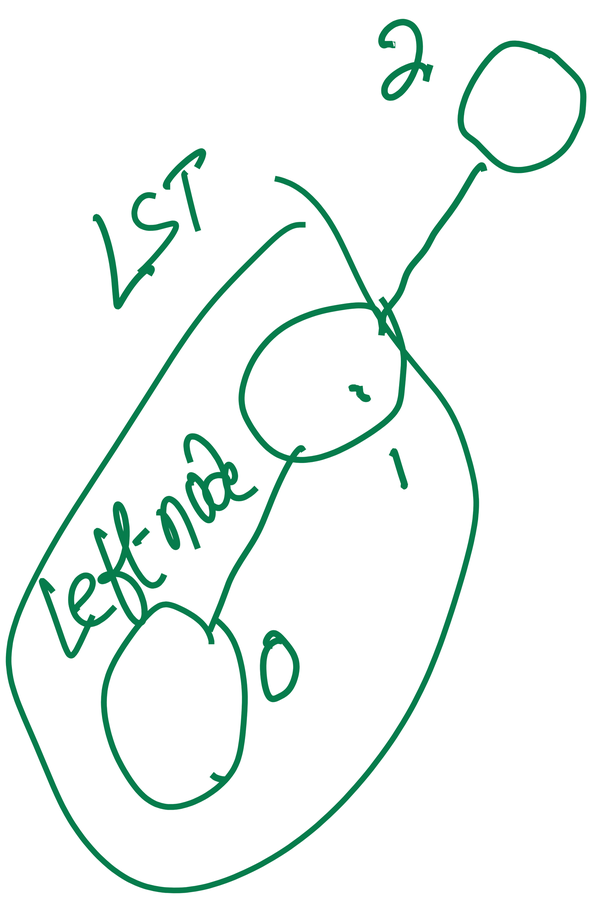
Inorder predecessor of BST

Rotation

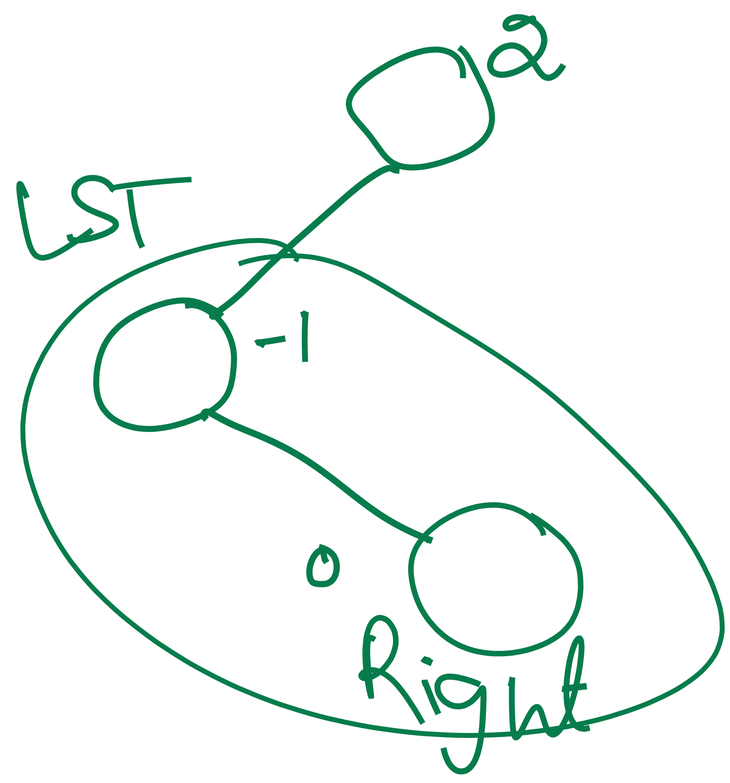
- Single Rotation
- Double Rotation

→ LL Rotation (Left node of a LST)
→ LR Rotation (Right node of a LST)
→ RL Rotation (Left node of a RST)
→ RR Rotation (Right node of a RST)

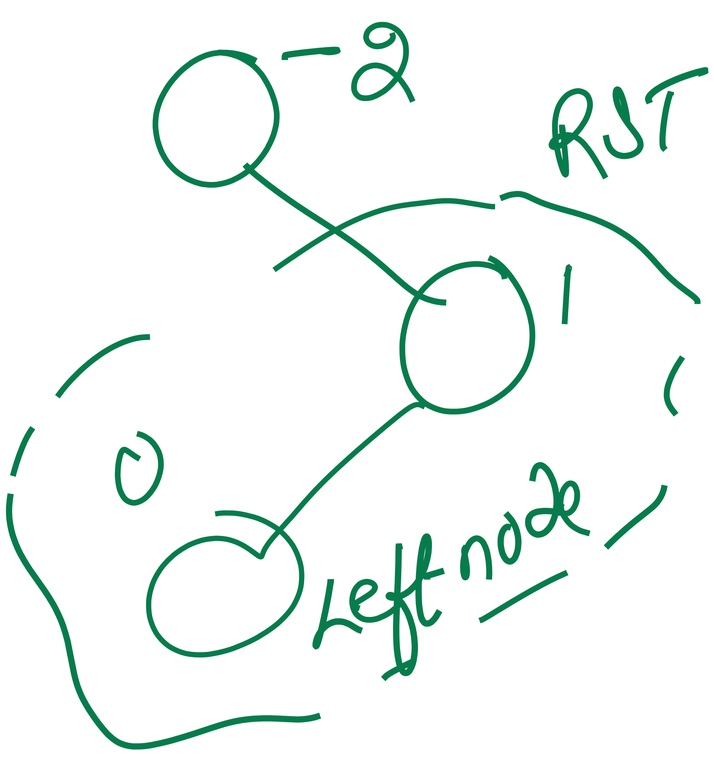
LL Rotation



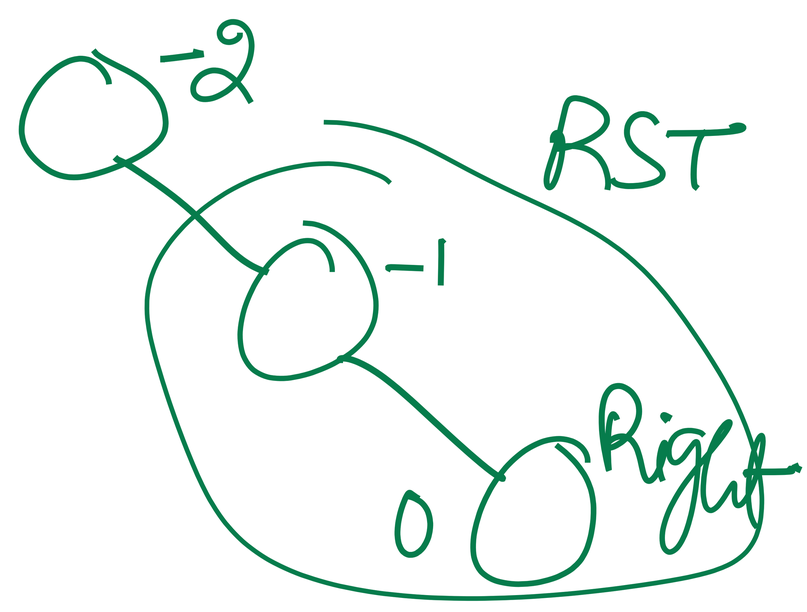
LR Rotation

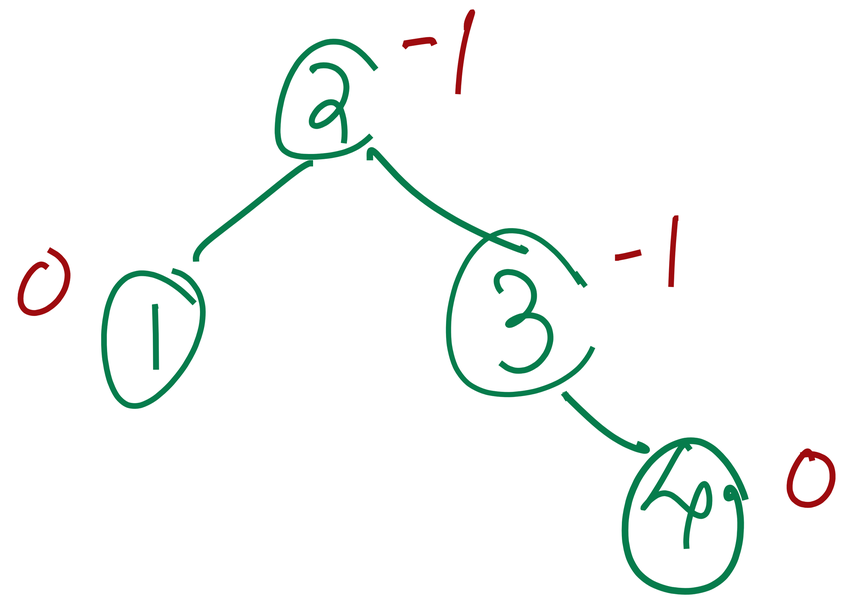


RL Rotation



RR Rotation

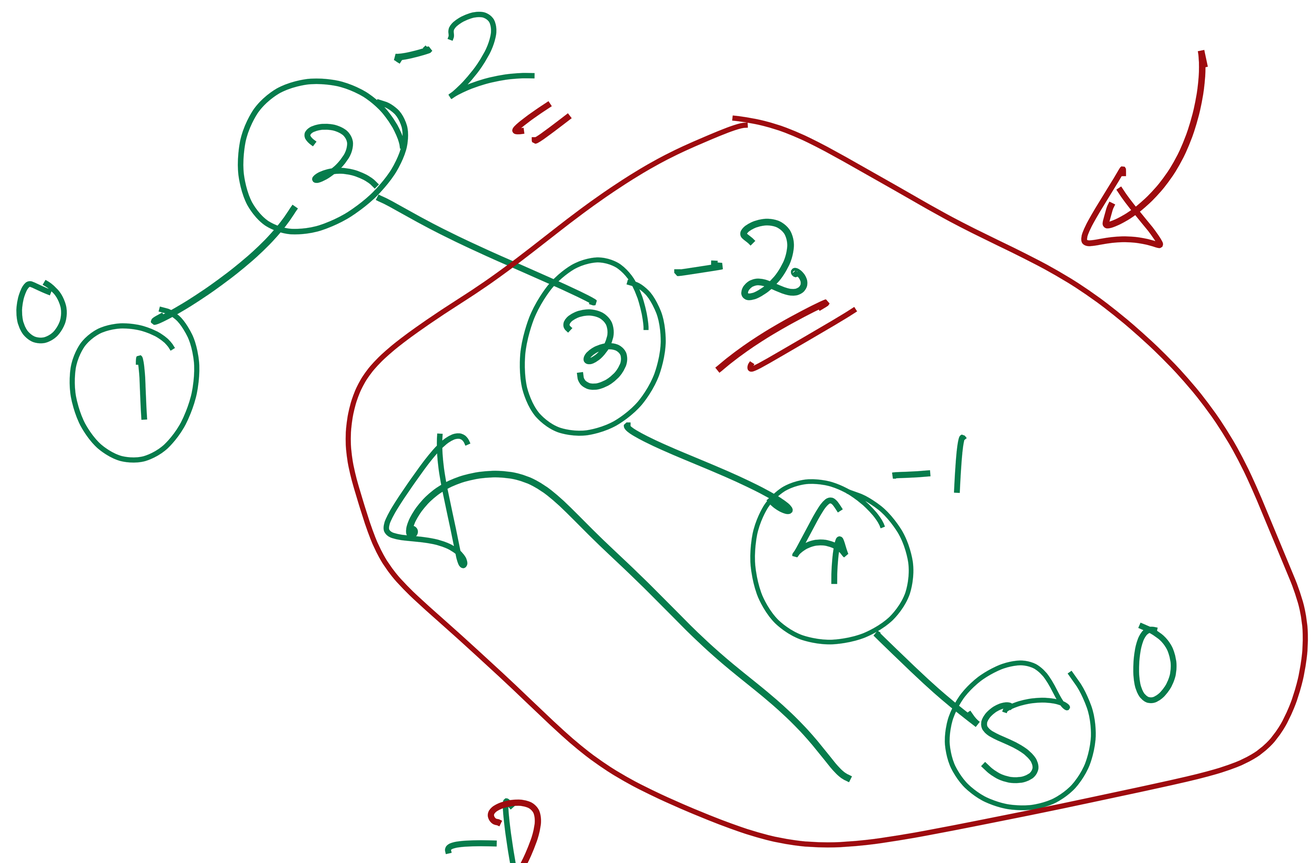


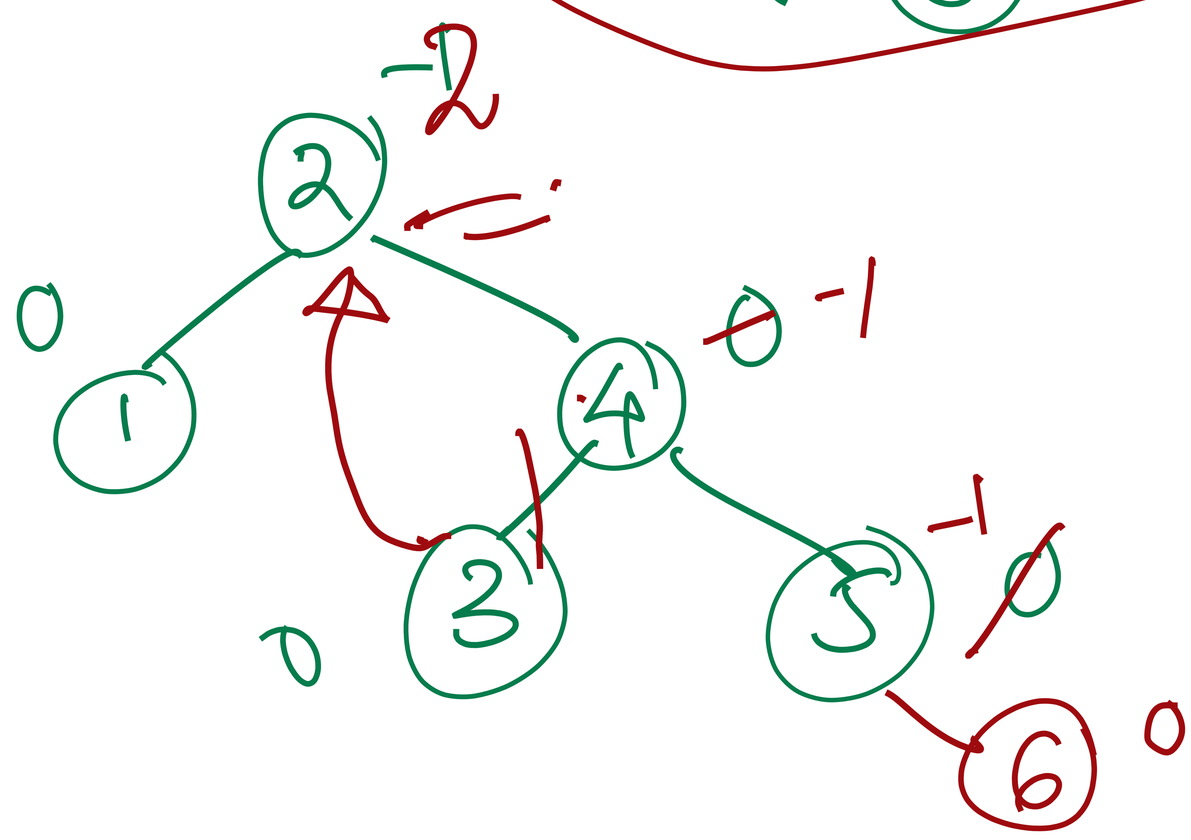


Insert 4:

Insert 5:

Apply RR Rotation

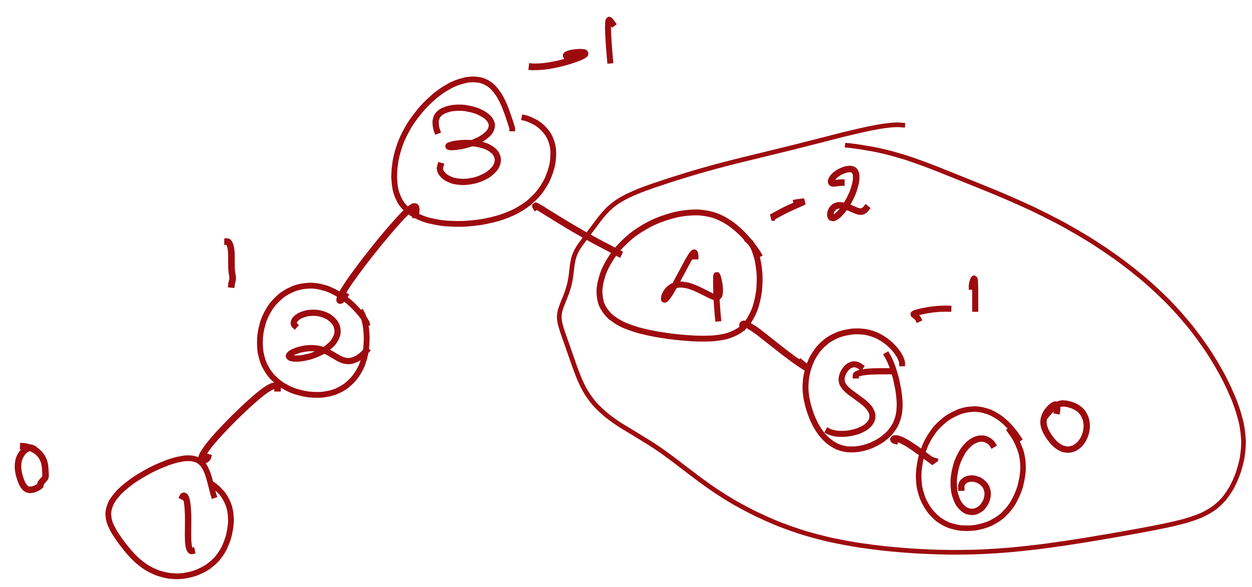


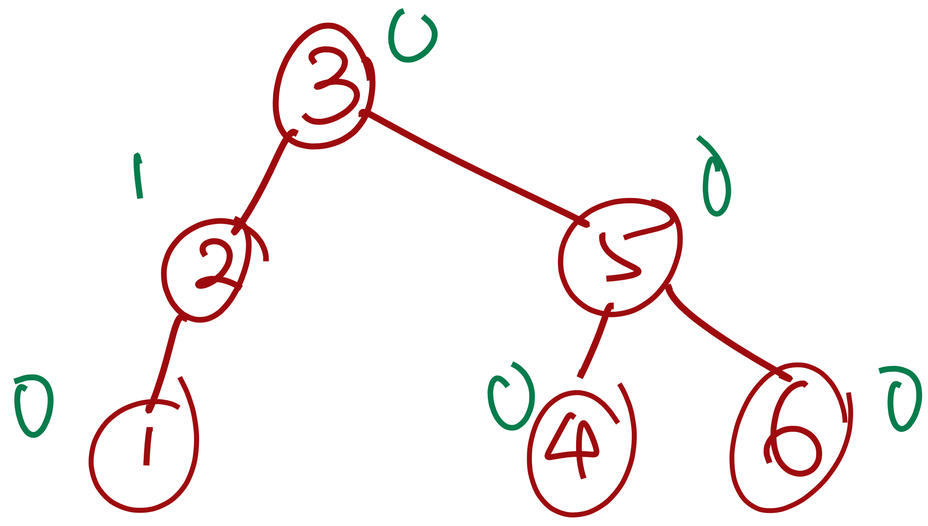


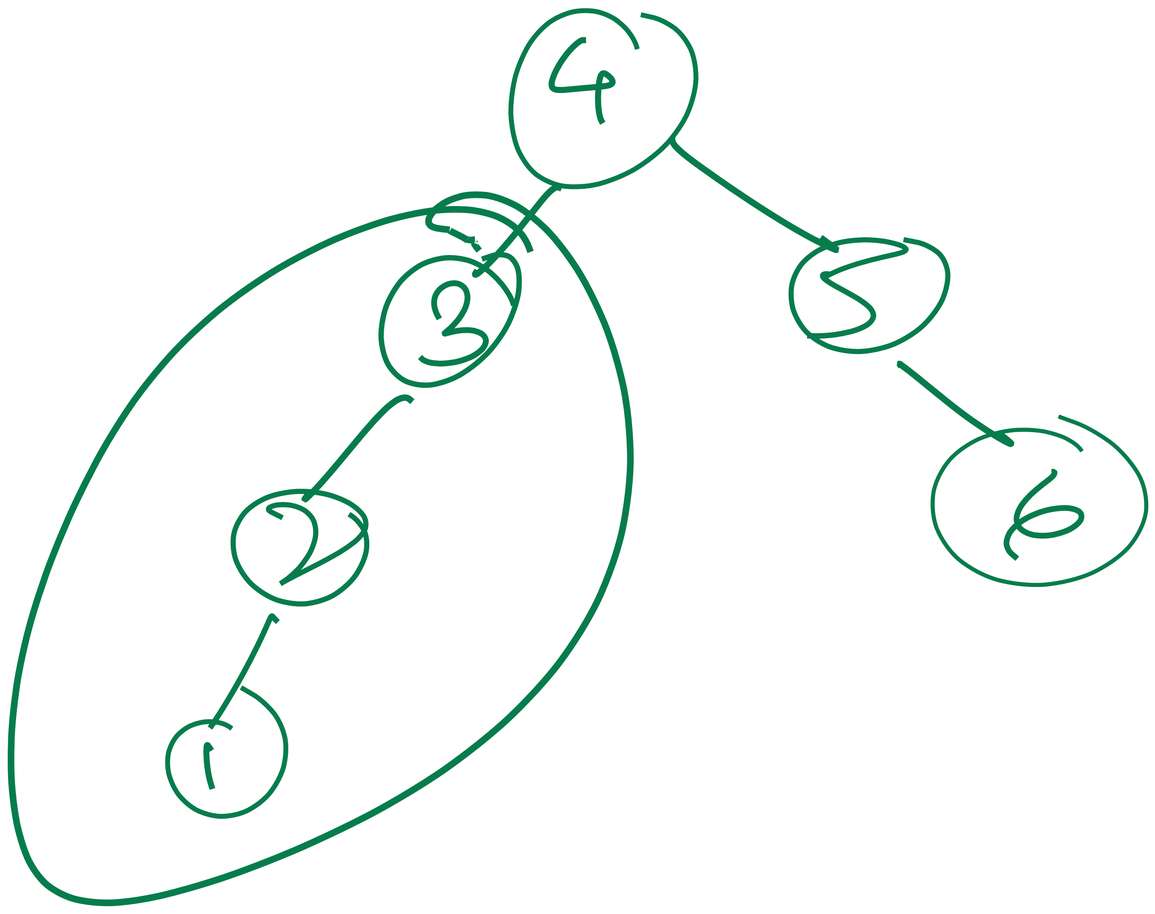
⇒ 1, 2, 3, 4, 5, 6

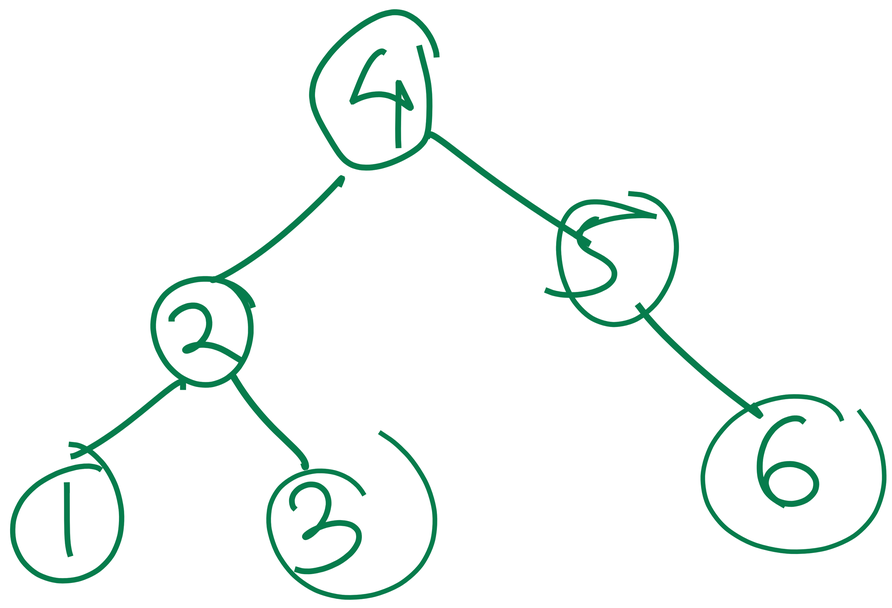
⇒ Case 1 Put 4 as root node

Case 2 Put 3 as root node









$\boxed{1, 4, 5, 6, 3, 2, 7, 9, 8}$

↳ How many single rotation & double rotations are there in building the AVL tree of the above list.

Complexity of AVL tree

Algorithm	Average	Worst Case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(\log n)$
Insertion	$O(\log n)$	$O(\log n)$
Deletion	$O(\log n)$	$O(\log n)$

Red Black Tree :- Self balancing BST.

↳ AVL Tree

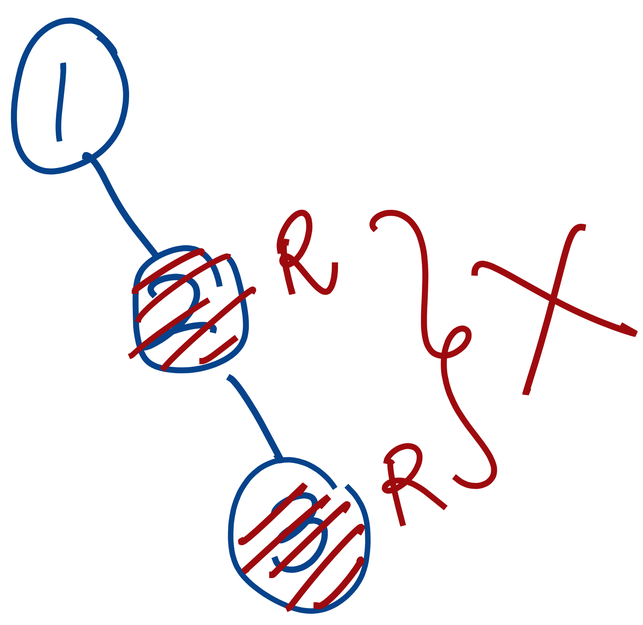
Suppose there is a skewed tree in AVL, then, we need to perform multiple no. of rotations.

In RB Tree: We must perform at most 2 rotations

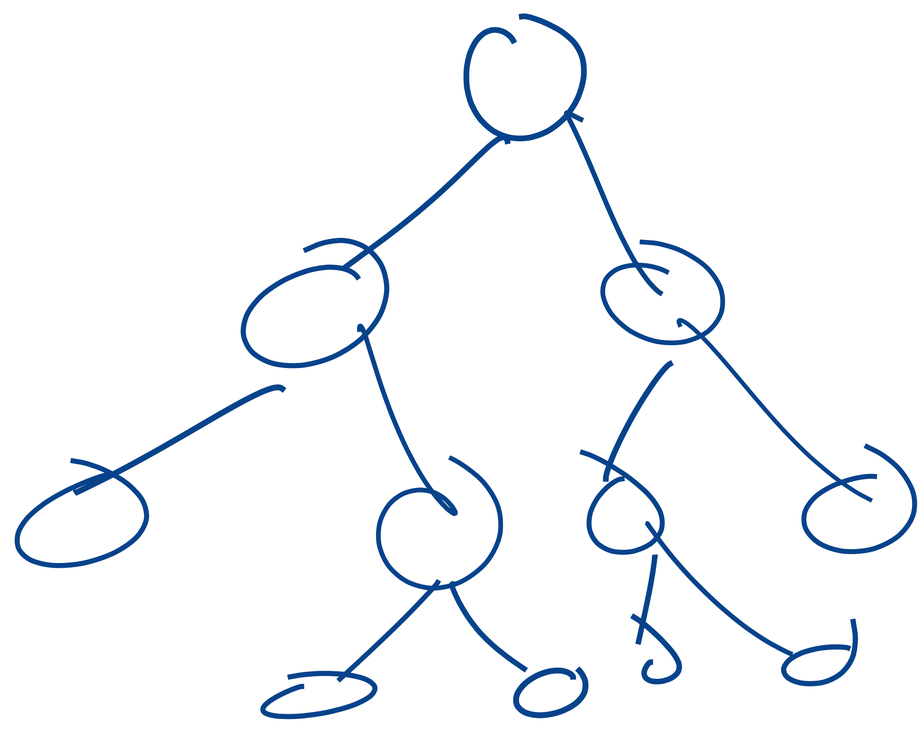
Properties of R-B Tree :-

→ The root node is always colored as Black.

→ No two adjacent nodes must be red in color.



→ Every path in tree must contains same amount of Black colored nodes.

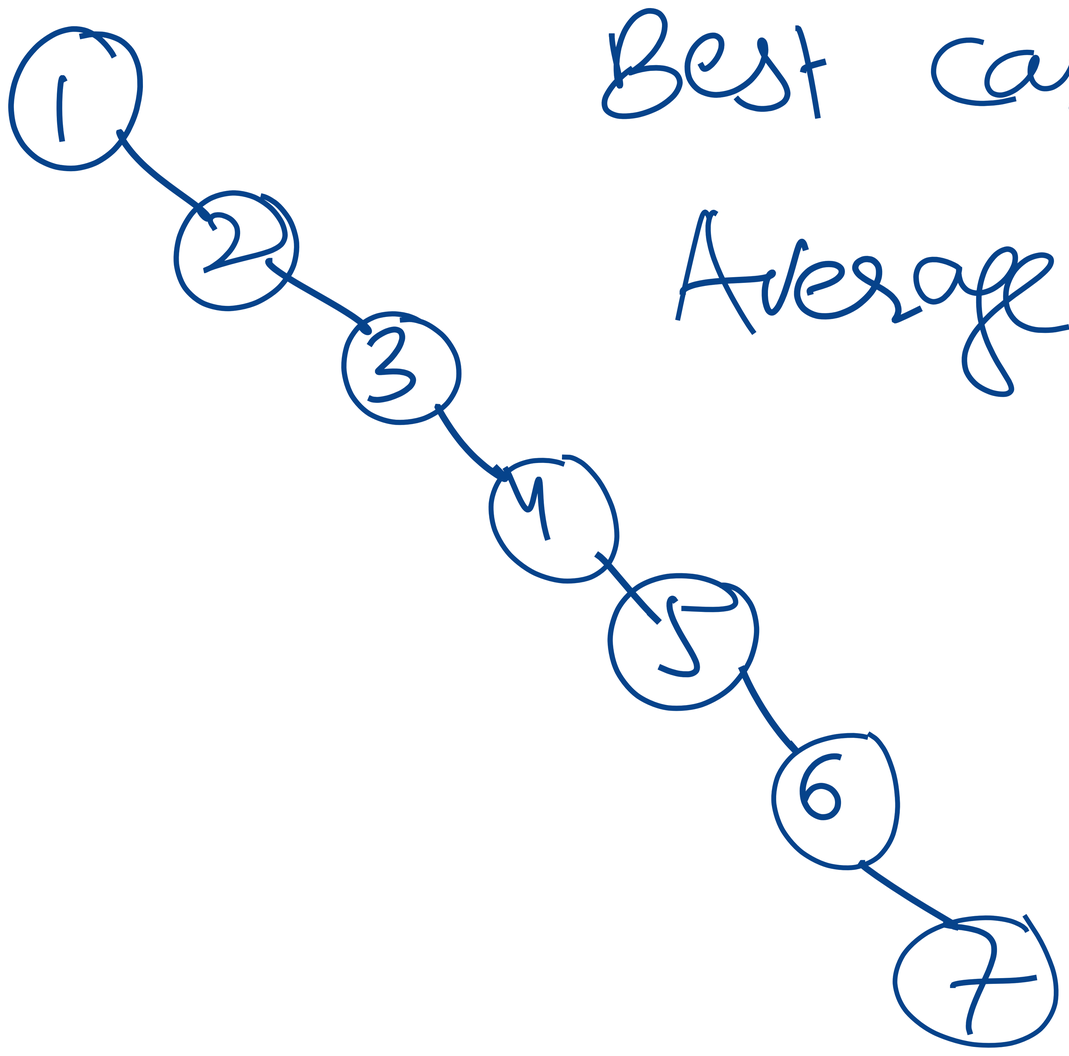


The time complexity of R.B. in all cases for insertion deletion & traversing is $O(\log n)$

BST = $\underline{O(n)}$ (worst case)

Best case ($O(1)$)

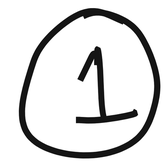
Average case ($O(\log n)$)



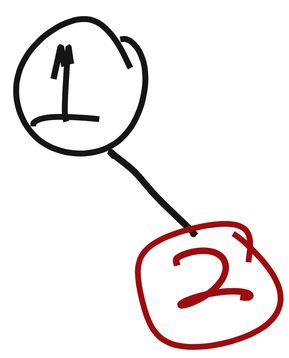
→ every node has a color either red or black.

⇒ Ex. 1, 2, 3, 4, 5, 6, 7.

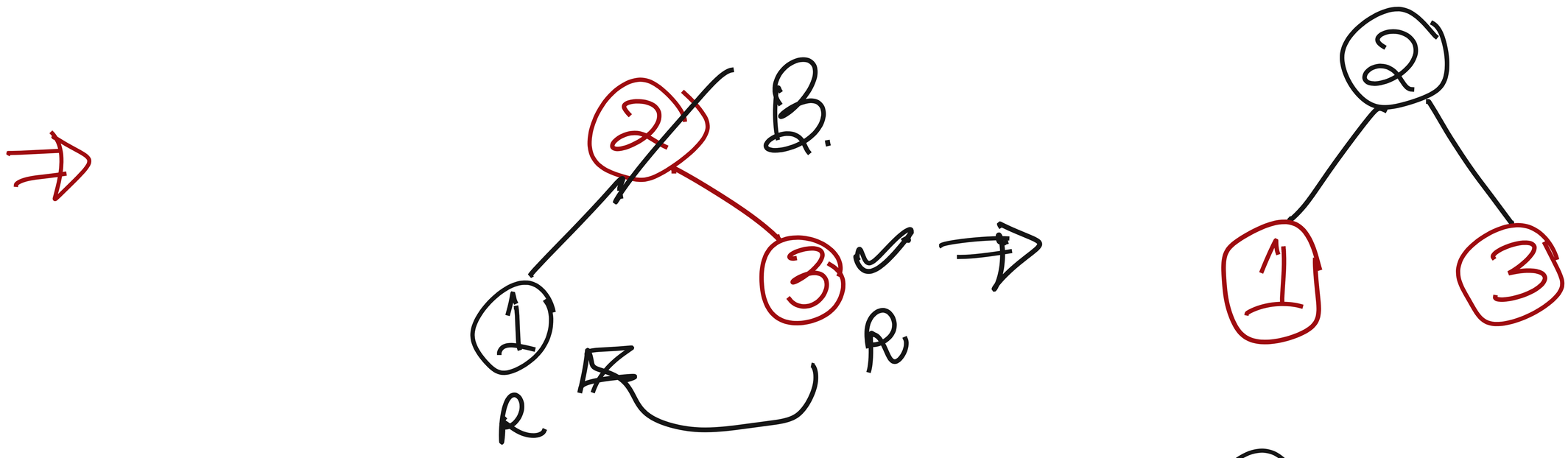
Insert 1:. (1 is root node)



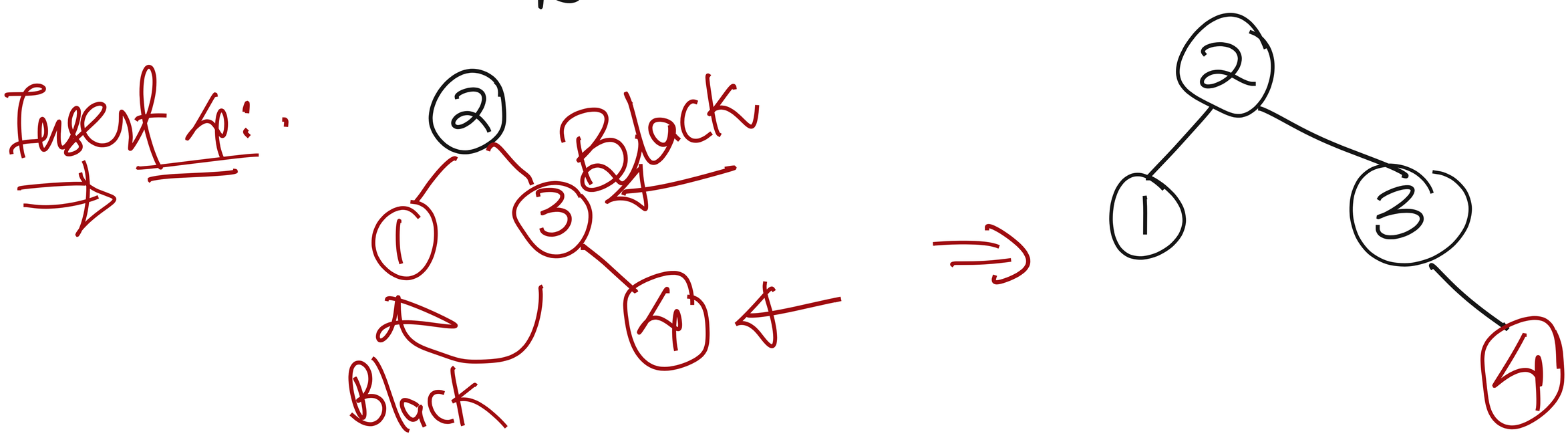
Insert 2: (As tree is not empty)



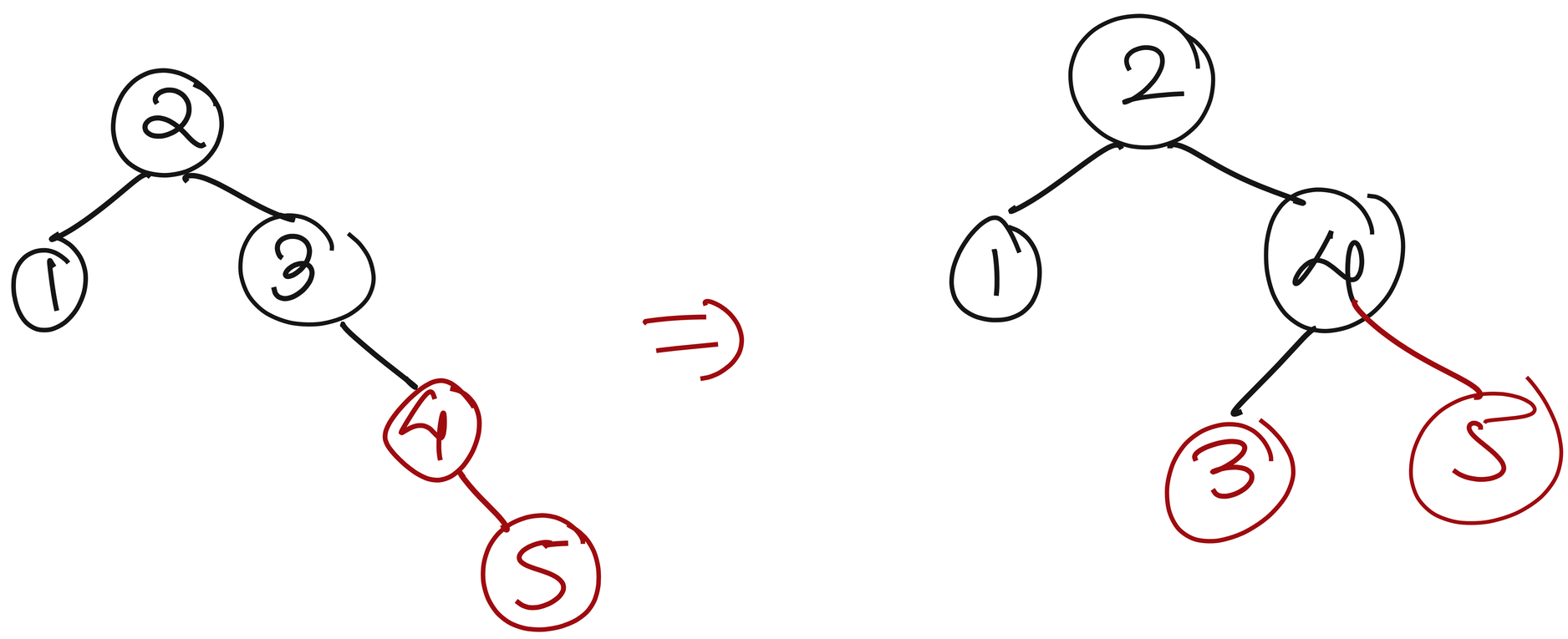
Insert 3:



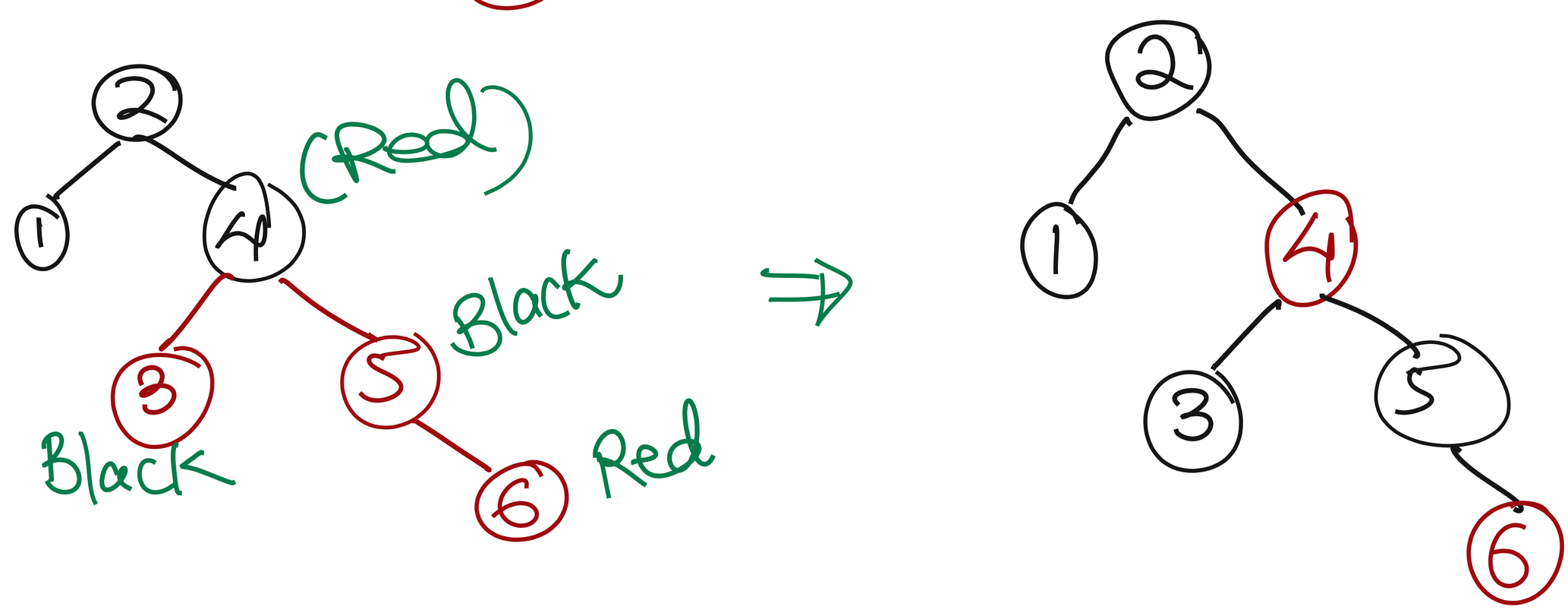
Insert 4:

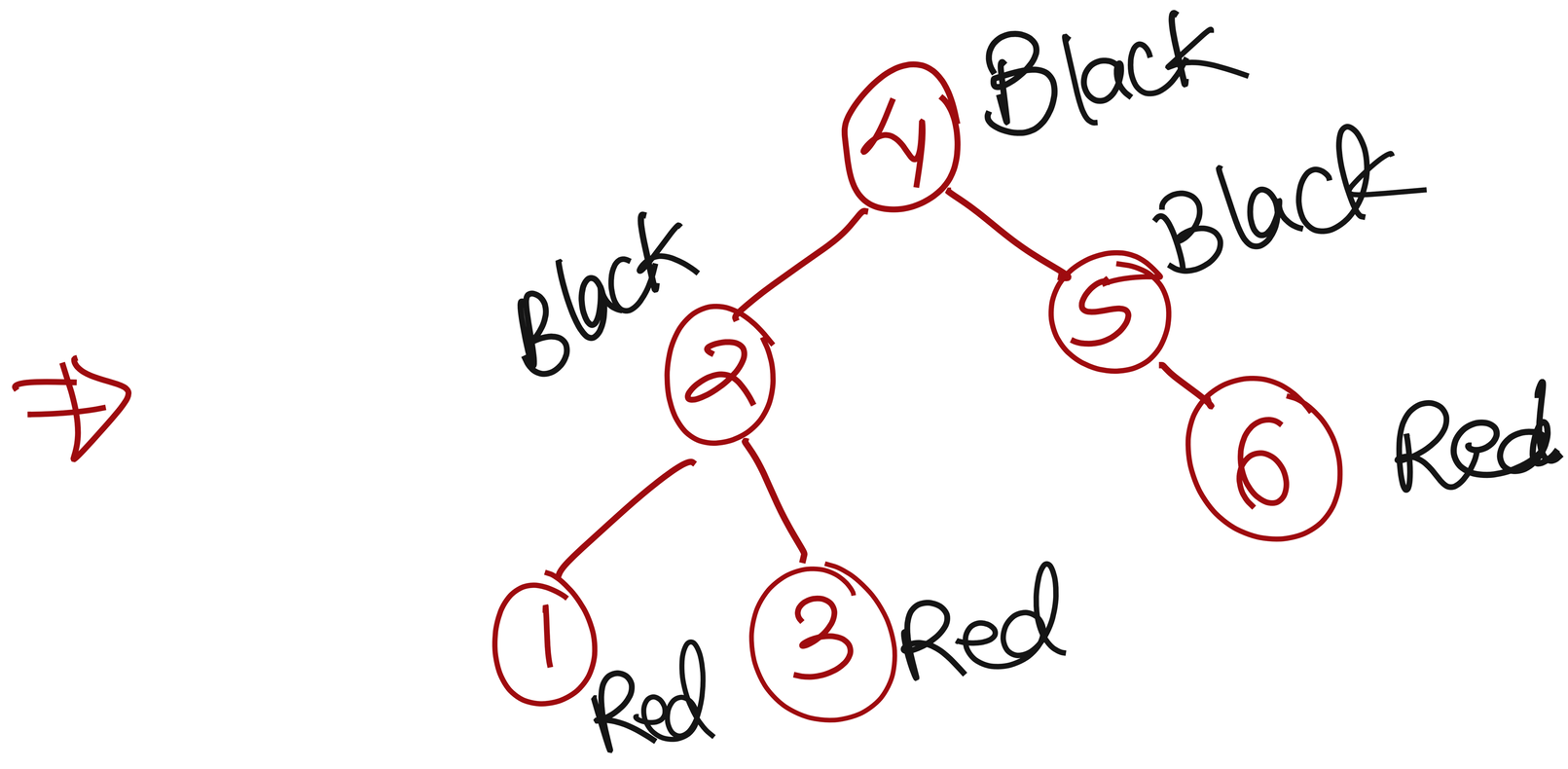


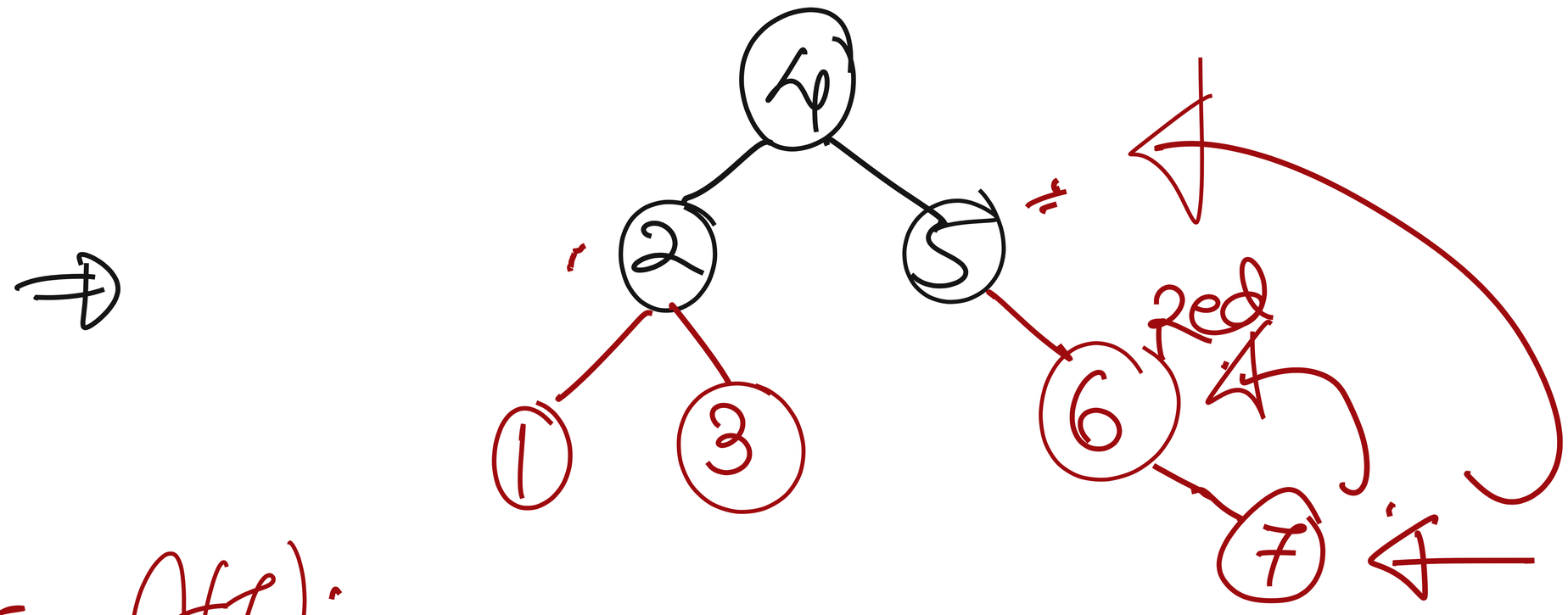
Insert 5:



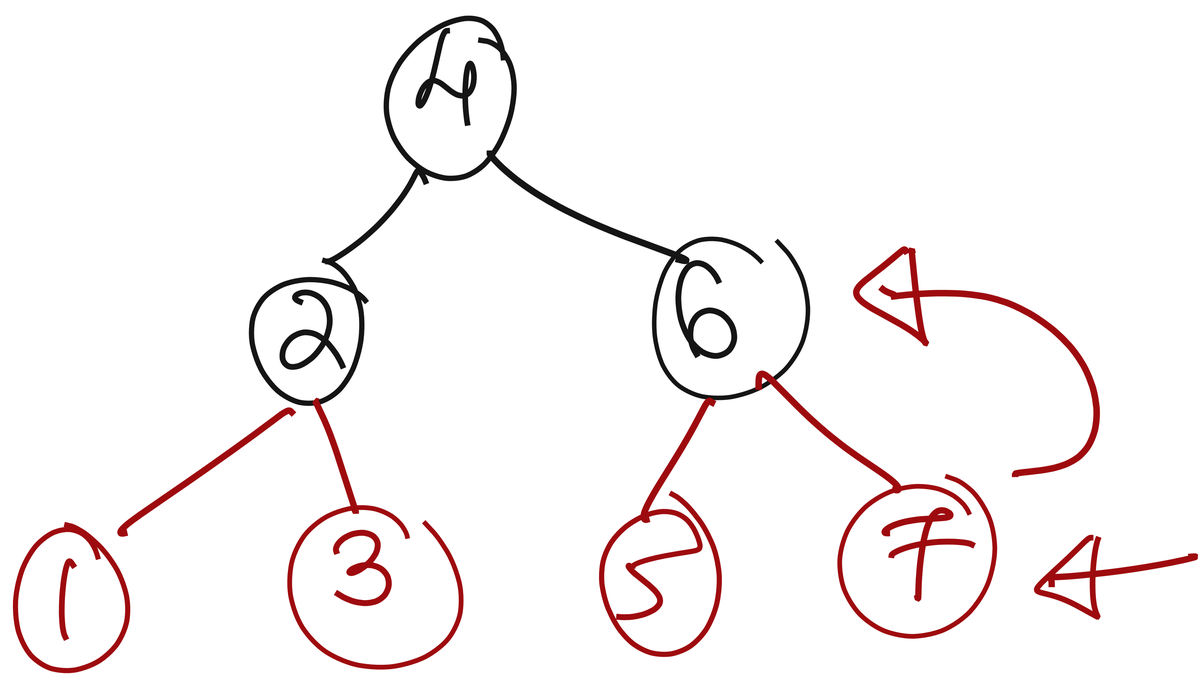
Insert 6:







Insert(7):



→ If the parent of a new node is Black; then exit.

→ If the parent of a new node is red, then we have to check the color of parent's sibling of a new node.

(a) If the color is black, then, perform the rotation & recoloring.

(b) If the color is red, then, we recolor the node. We also check whether parent's parent of a new node is the root node or not. If it is not a root node, we will recolor & recheck the node.

⇒ Perform insertion operation on R-B tree for following elements:

10, 18, 7, 15, 16, 30, 25, 40, 60.

AVL Tree → Insertion, deletion & Traversing.

Heap Tree → Insertion, Deletion & Traversing.

Red-Black Tree → Insertion, Deletion & Traversing.

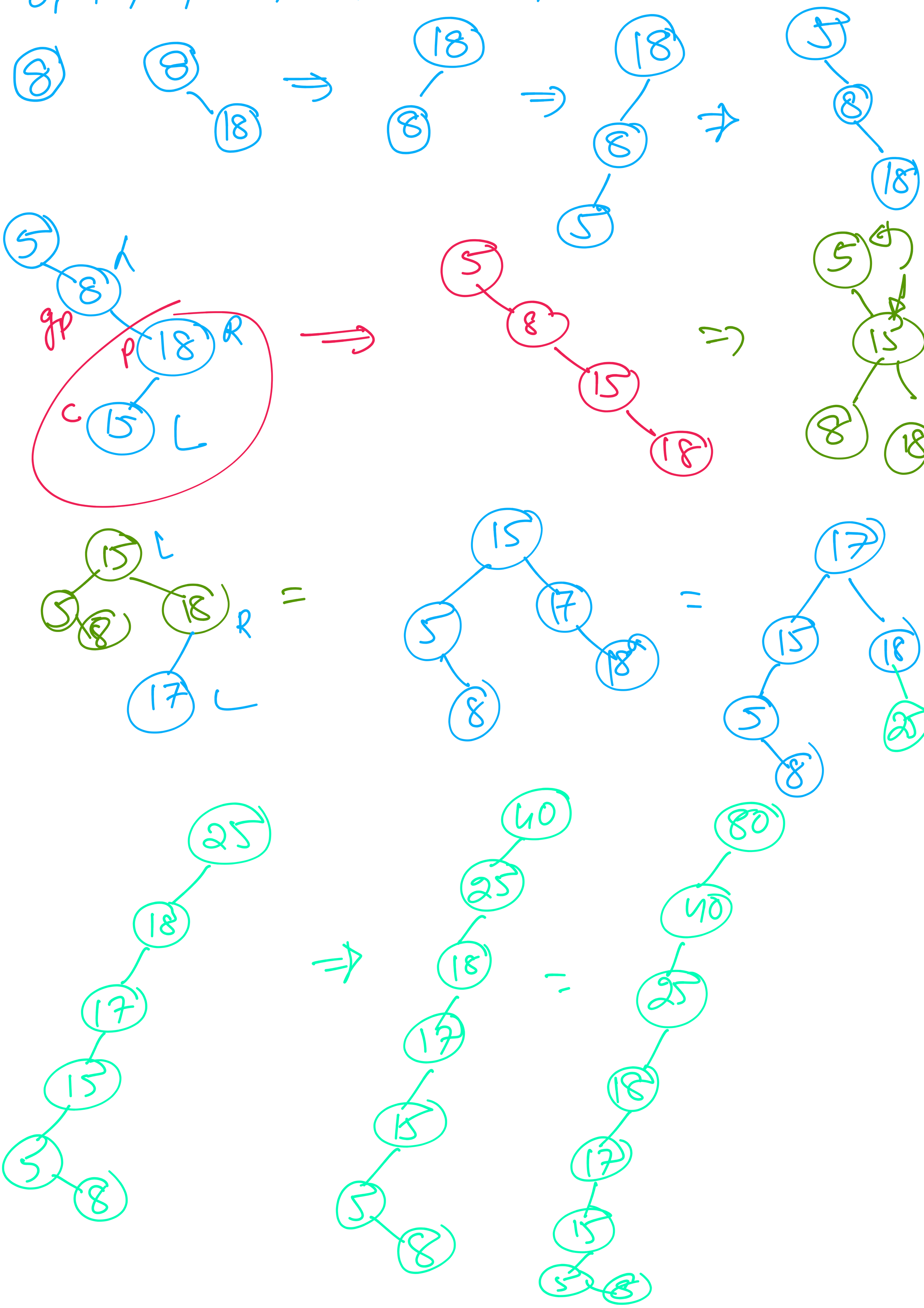
Splay Tree → Insertion, deletion & Traversing.

B Tree →

B^+ Tree →

Construct Splay Tree for the following list of elements:

8, 18, 5, 15, 17, 25, 40, 80.



B-Tree :- extension of Binary Search Tree :-

in place of two child, B-Tree contains more than two children also.

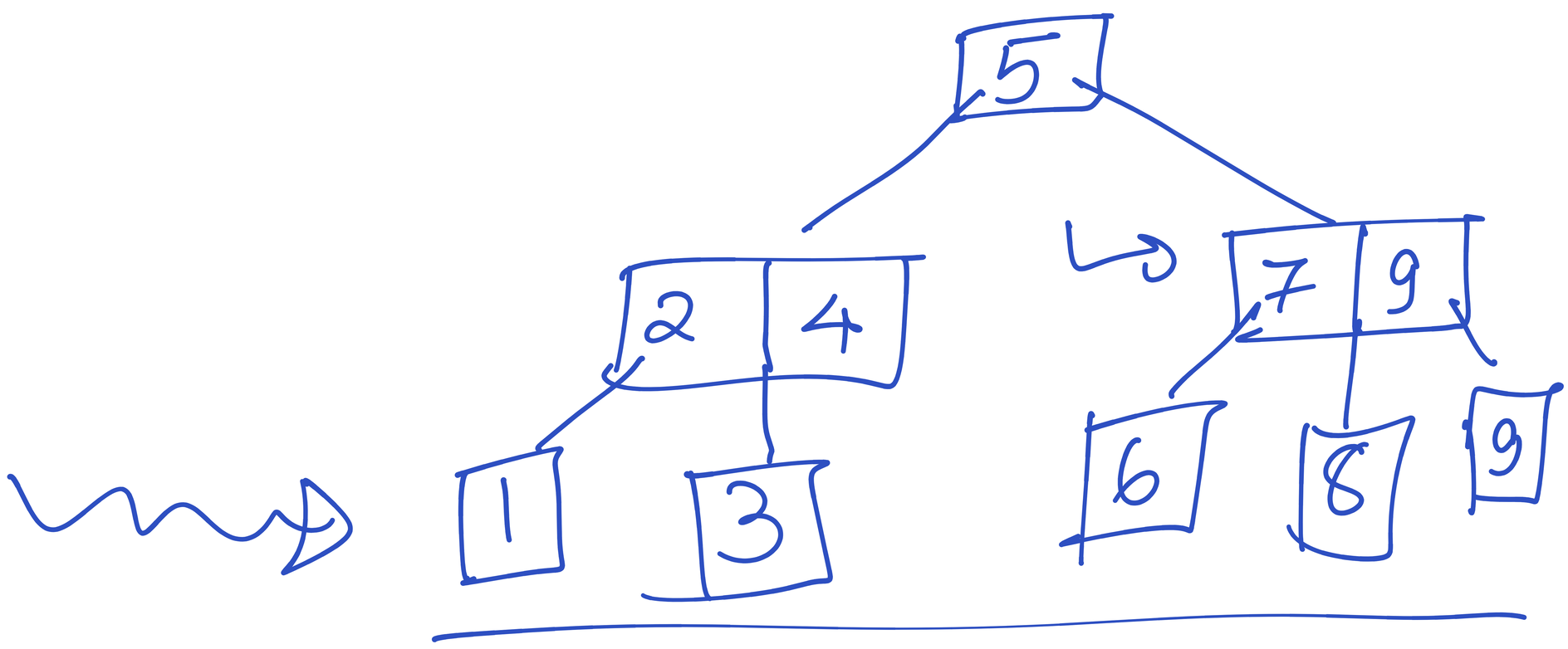
⇒ This is also called as m-way search Tree.

Properties :-

① All leafs are at same level.

② B-Tree is defined by the term minimum degree 't'. This 't' depends on the disk block size.

③ Every node, except root node contains atleast t-1 keys. Root node contains a minimum of '1' key.



④ All nodes (including root) contains atmost 2*t-1 keys.

⇒ $t = \boxed{2}$ $2*2-1 = \boxed{3}$

⑤ Number of children of a node is equal to the no. of keys in it plus 1.

⑥ All keys of a node are stored in increasing order.

⑦ B-Tree is also called as balanced multilevel search tree.

⑧ O(log n) – time complexity.

⑨ Insertion of node in B-Tree happens only at leaf node.

Applications

→ large databases to access the data stored on the disk.

→ Multilevel Indexing can be achieved.

→ Searching is easy.

→ Most of servers uses B-Tree approach.

Adv:

Adv.

→ self Balancing

→ O(logn) time complexing for insertion deletion & searching.

→ High concurrency & high throughput.

→ Efficient storage utilization.

Disadv.

→ High disk usage

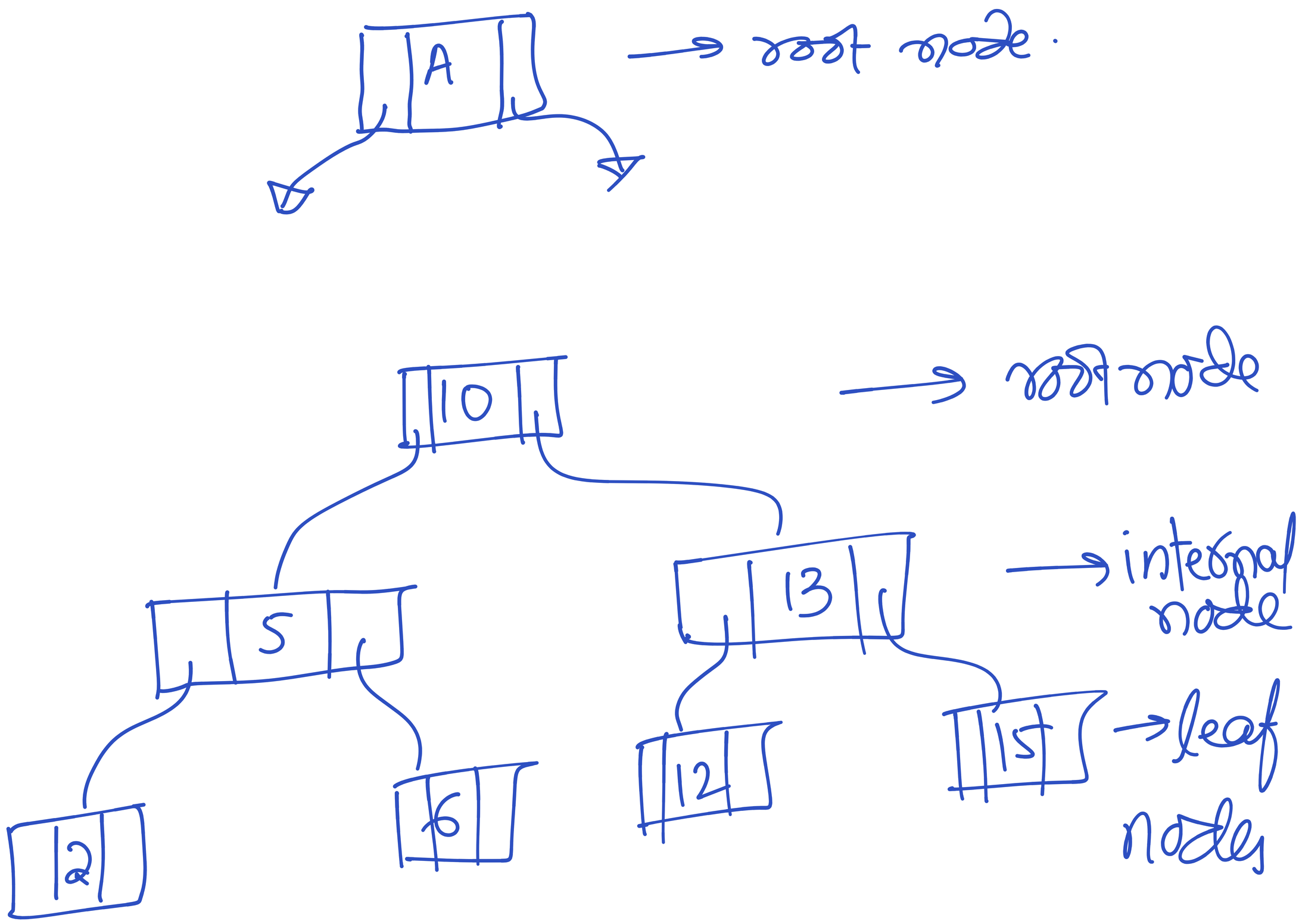
→ Not best for all cases

→ Slow in comparison to other DS.

Dt :- 16/02/2024. ⇒ 19, 13, 20, 5, 94, 123, 70, 45, 89, 39, 63, 30, 25, 35, 38, 59, 52, 54, 73, 55.

B-Tree :-

A



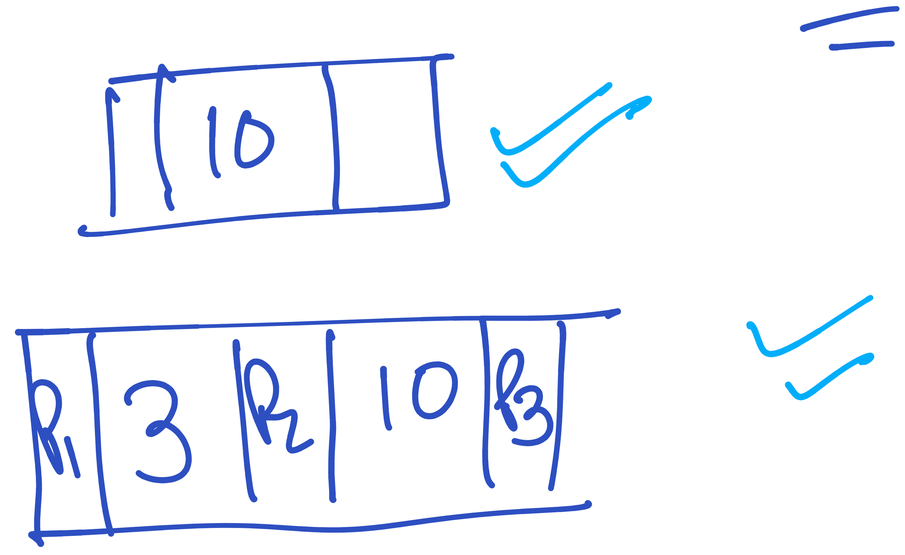
Insertion in B-Tree :-

① Search to determine which leaf node will hold a key.

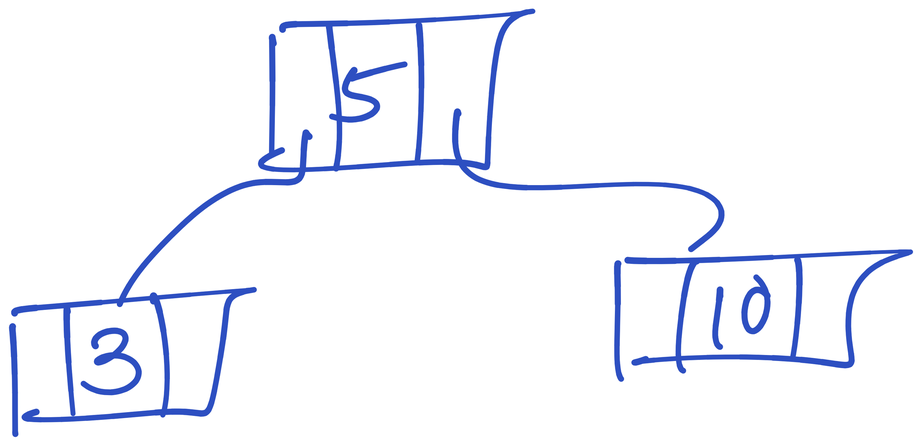
② If leaf node have a space, insert the key in ascending order.

order "t" = 3

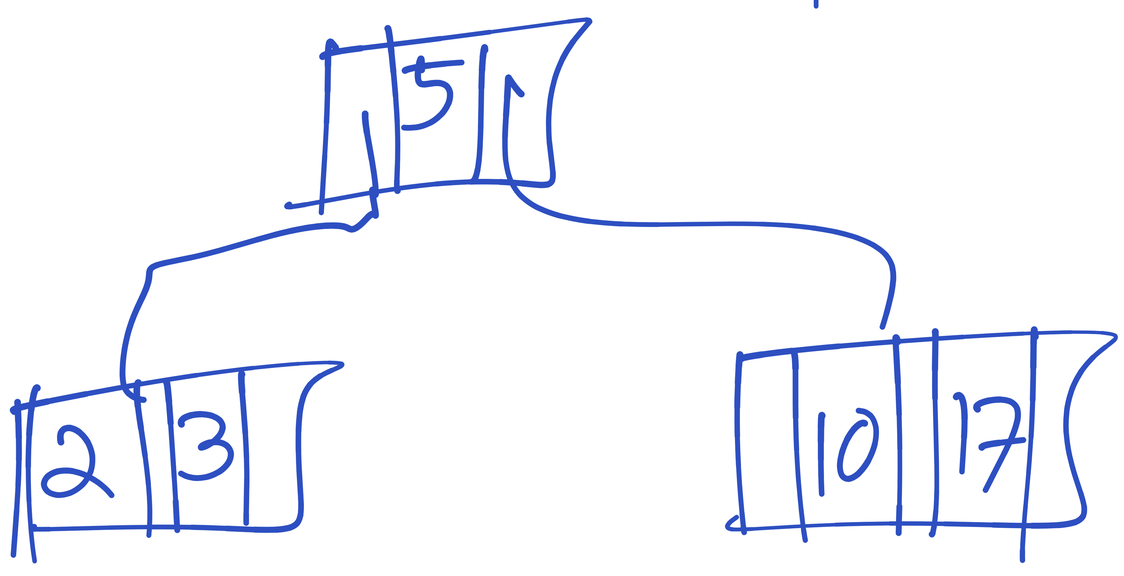
10, 3, 5, 2, 17



3 | 5 | 10 ✓

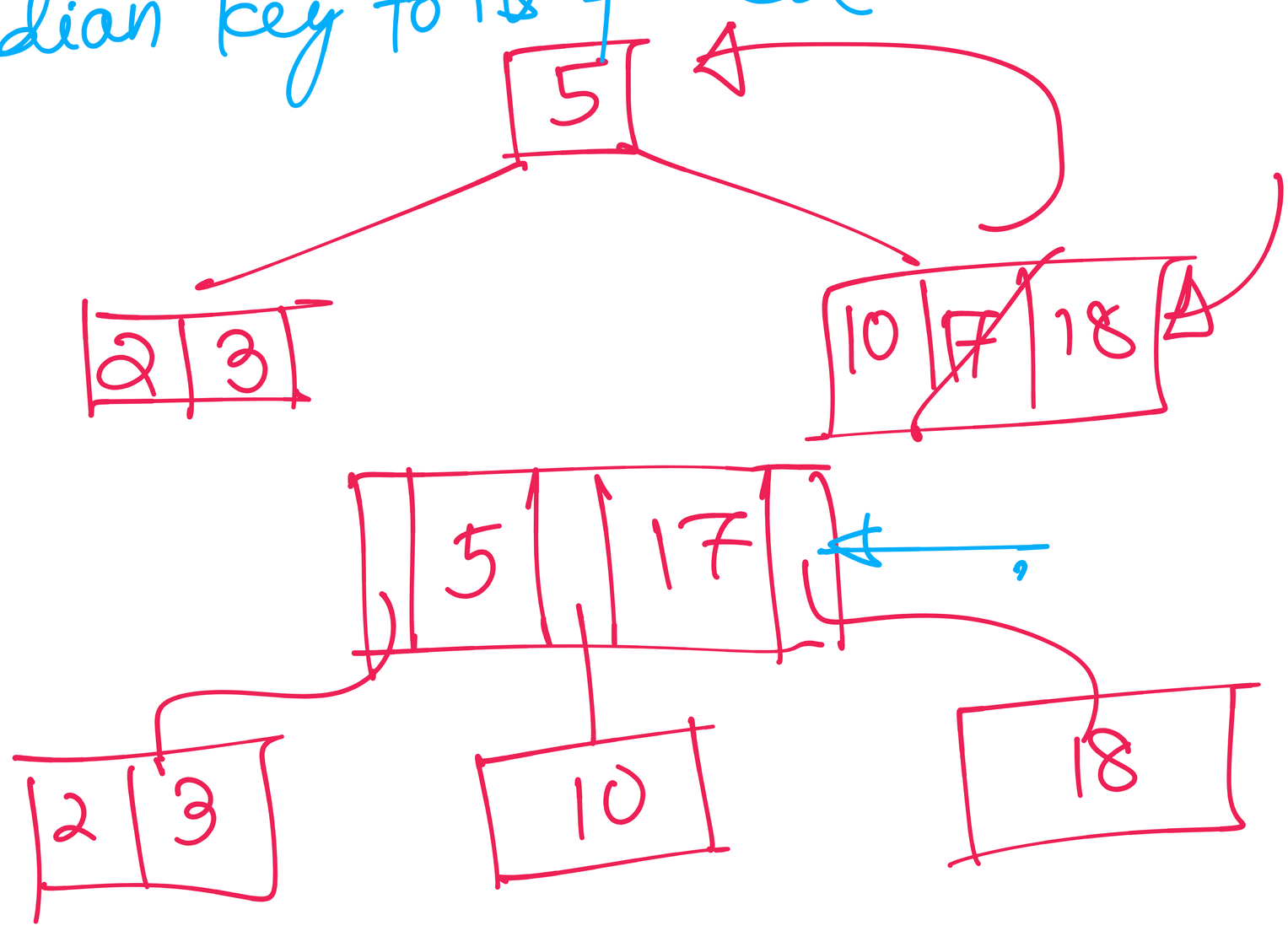


③ otherwise, split the node in two parts and shift the median key to the parent.



④ If the parent node is full, recursively split and promote median key to its parent.

Insert 18



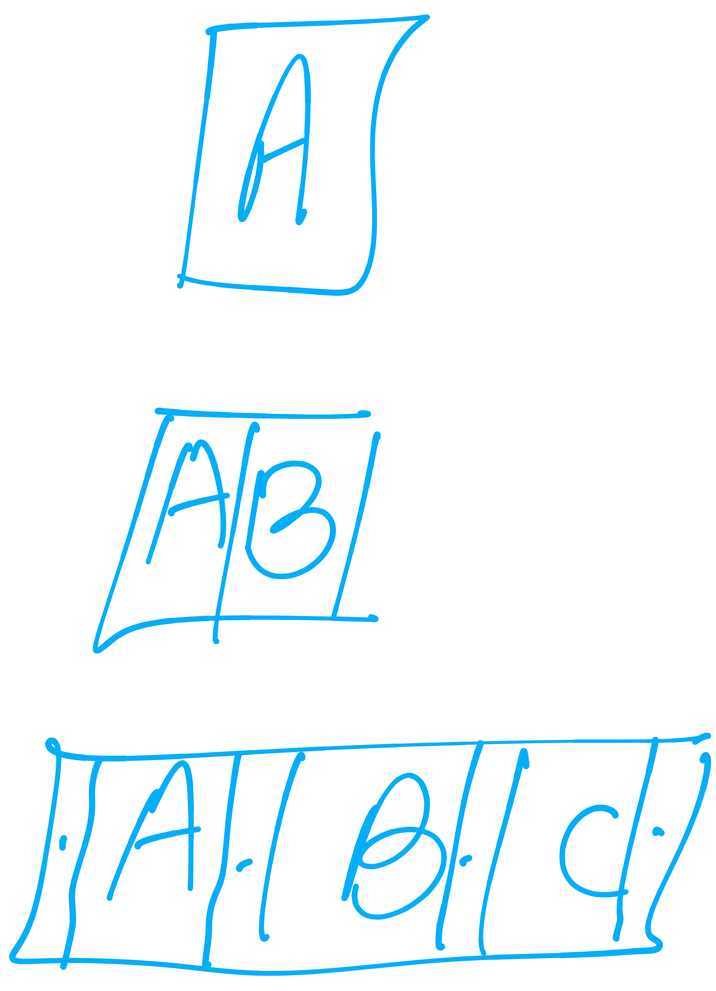
⑤ If a promotion is made to a full root node, split and create new root node holding the other nodes.

Insert A, B, C, D, E, F, G, H, I, J.

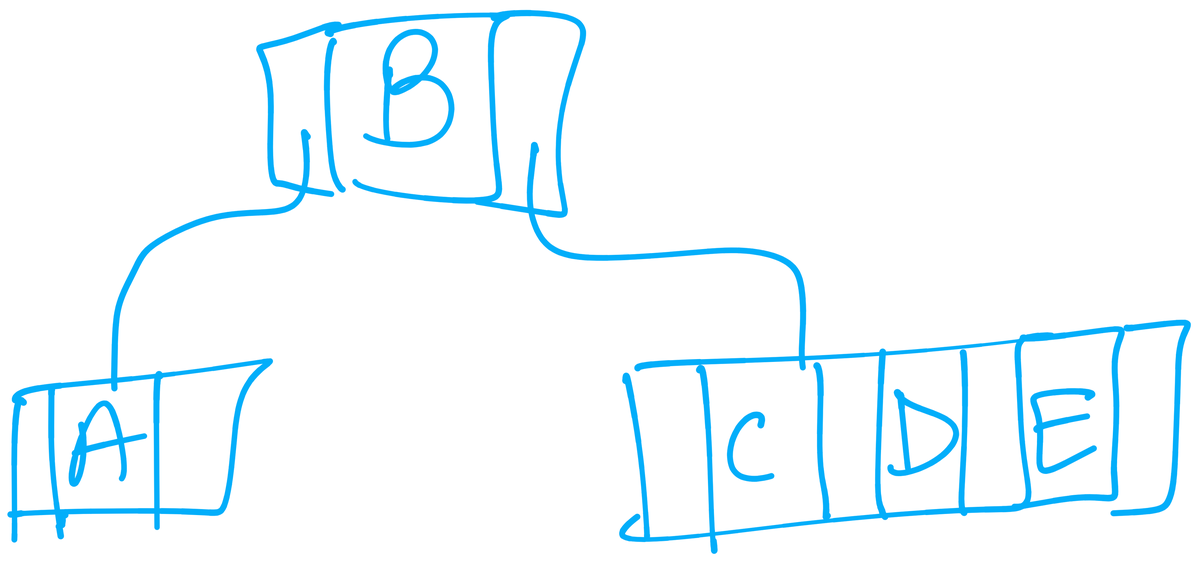
Consider the order of B Tree $= 4 = t$

max. no. of keys $= 3 = t - 1$

min no. of keys $= \lceil \frac{4}{2} \rceil - 1 = \lceil t/2 \rceil - 1$.



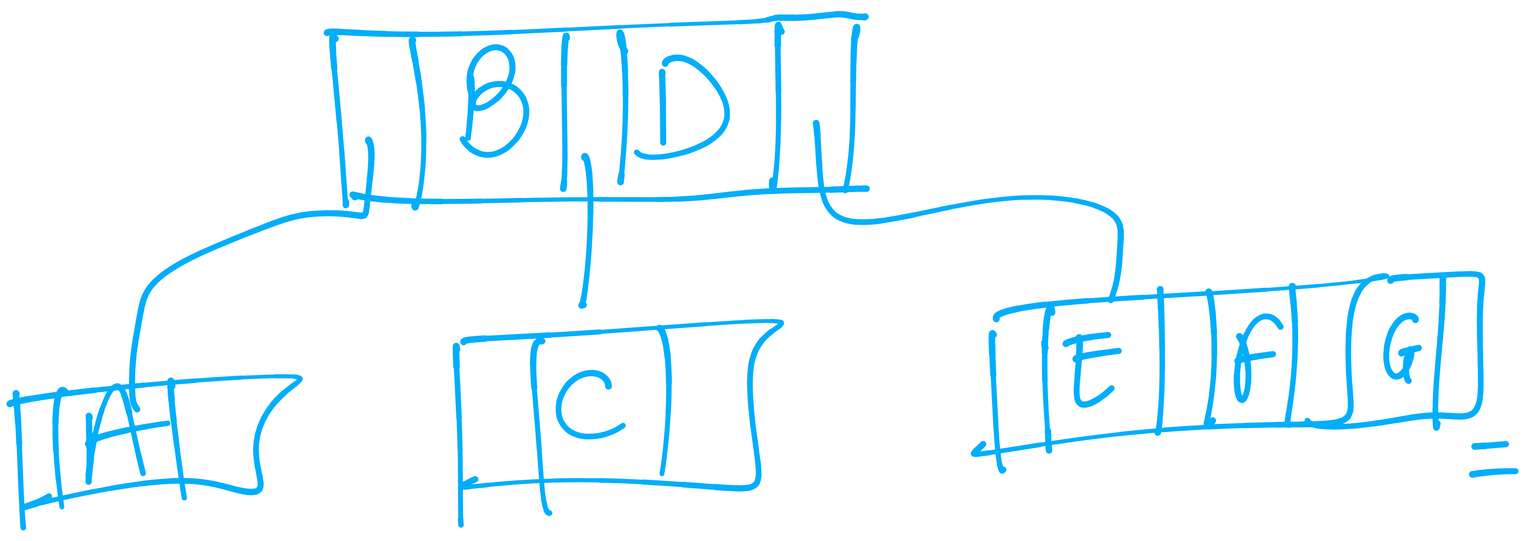
Insert D. (overflow)



(overflow)

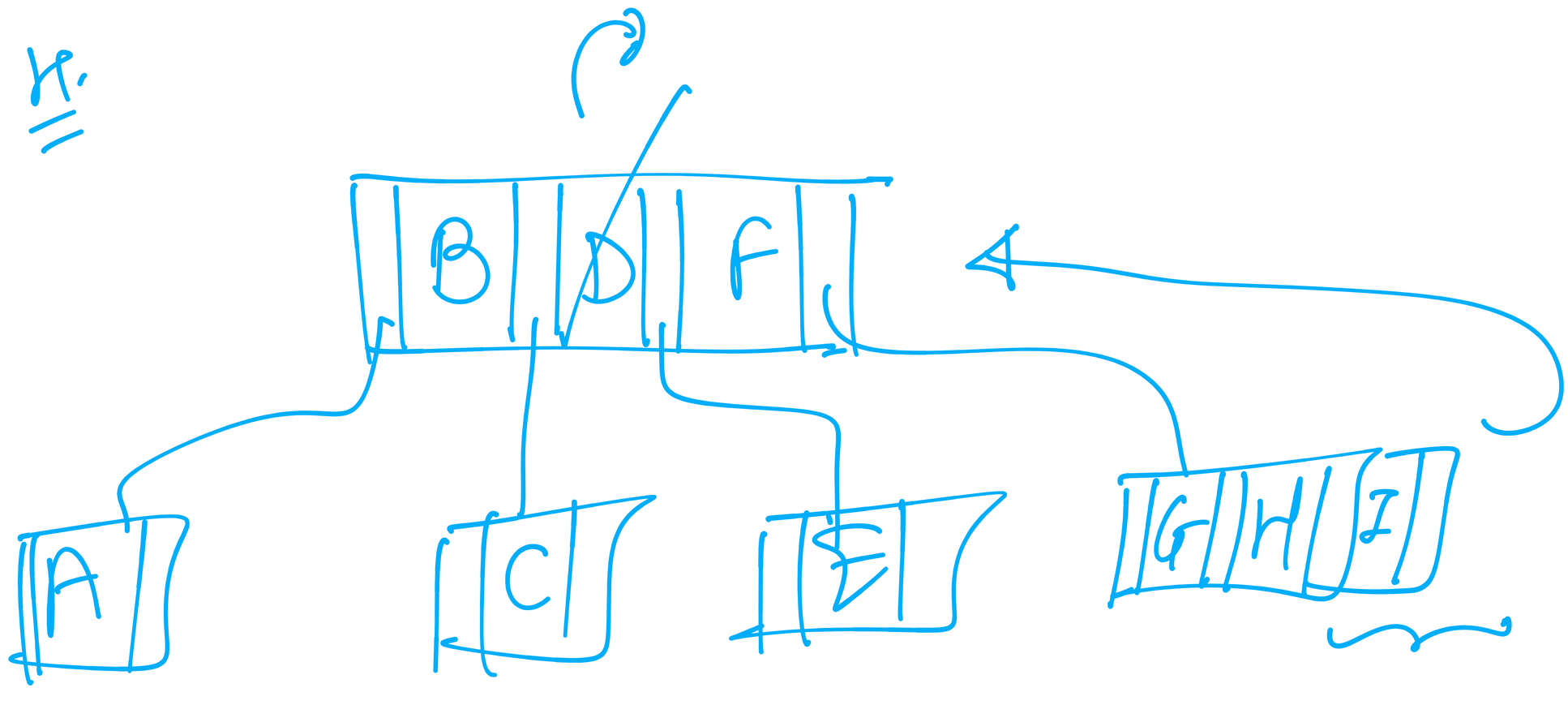
Insert E

Insert F



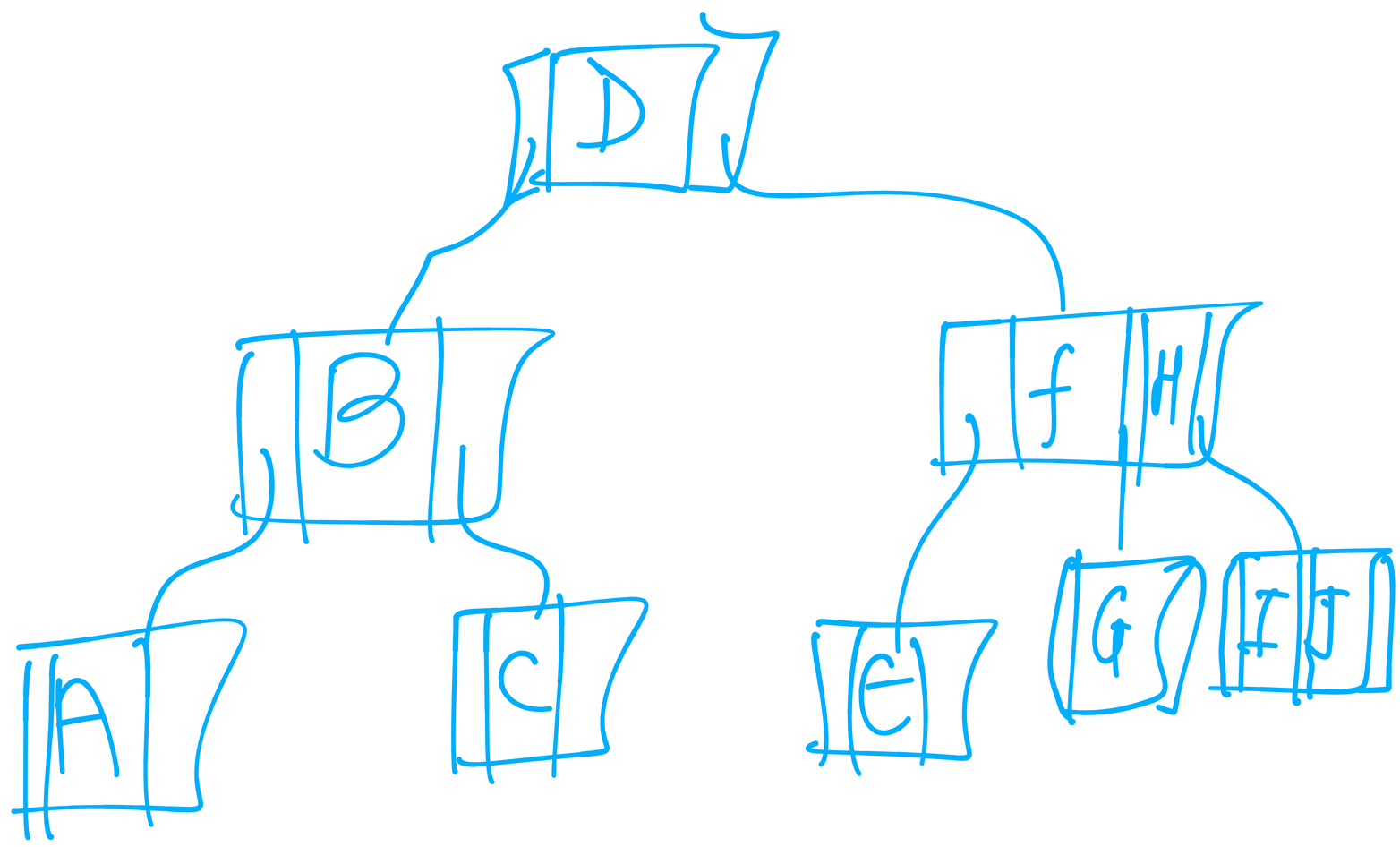
Insert G

Insert H.



Insert I.

Insert J



⇒ Delete D, B, H.

Insert the following keys in BTree of order 4.

Keys: 2, 5, 10, 1, 6, 9, 4, 3, 12, 18, 20, 25