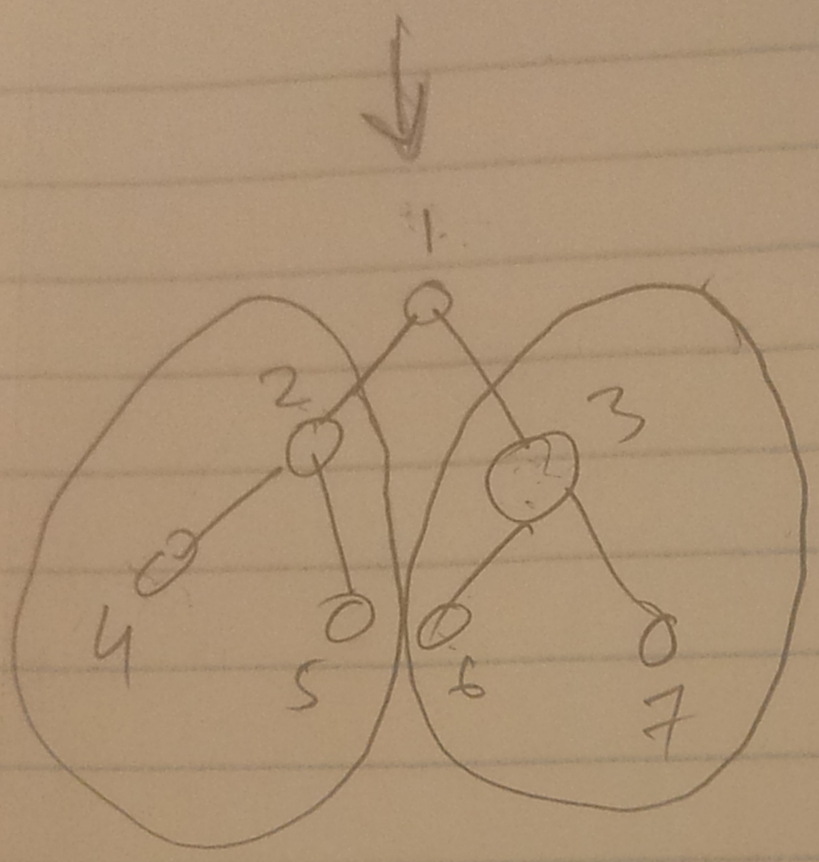


PreOrder : Root → left child → Right chi ld

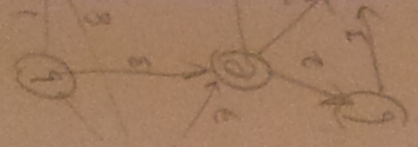
1 → 2 → 4 → 5 → 3 → 6 → 7

InOrder : Left → ~~Root~~ Root → Right

~~1~~ 4 → 2 → 5 → 1 → 6 → 3 → 7

post-order : Left → Right → root

4 → 5 → 2 → 6 → 7 → 3 → 1



Dynamic (4)

- technique to solve a problem by solving overlapping subproblems
- subproblems are dependent

$S_1 = \{VAGABOND\}$
$S_2 = \{MIXTAPE\}$] LCS?

Subproblems,

$S_1' = \{VAGA\}$
$S_2' = \{MIX\}$

$LCS(S_1', S_2') + LCS(S_1 - S_1', S_2 - S_2') \neq LCS(S_1, S_2)$

Divide & Conquer

- technique to solve a problem by diving the problem into independent subproblems and solving them
- subproblems are independent

Example:

$A_1 = [2, 1, 5, 9, 4]$ sorting ?

$A_1' = [2, 1, 5]$

$A_1'' = [9, 4]$

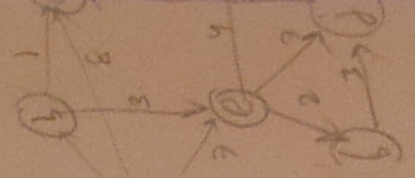
sort in A_1' and A_1'' and merging their solution gives same result as sort(A_1).

Approx

- algorithm that runs in polynomial time and outputs a solution close to the optimal solution
- can guarantee global (some sort of) optimization
  eg 2-approx algos can guarantee not-bad-than-twice-the-optimal solution

Greedy

- algorithm that produces an output by choosing locally optimal solution for each iteration
- cannot guarantee any sort of global optimization
  eg. local optimization might not lead to any sort of global optimization as it might miss solutions that are locally non-optimal but lead to global optimization



Dynamic

- technique to solve a problem by solving overlapping subproblems
- subproblems are dependent

$S_1 = \{VAGABOND\}$
$S_2 = \{MIXTAPE\}$
] LCS?

Subproblems,

$S_1' = \{VAGA\}$
$S_2' = \{MIX\}$

$LCS(S_1', S_2') + LCS(S_1 - S_1', S_2 - S_2')$
$\neq$
$LCS(S_1, S_2)$

Divide & Conquer

- technique to solve a problem by diving the problem into independent subproblems and solving them
- subproblems are independent

Example:

$A_1 = [2, 1, 5, 9, 4]$ sorting?

$A_1' = [2, 1, 5]$

$A_1'' = [9, 4]$

sort in A_1' and A_1'' and merging their solution gives same result as $sort(A_1)$.

Approx

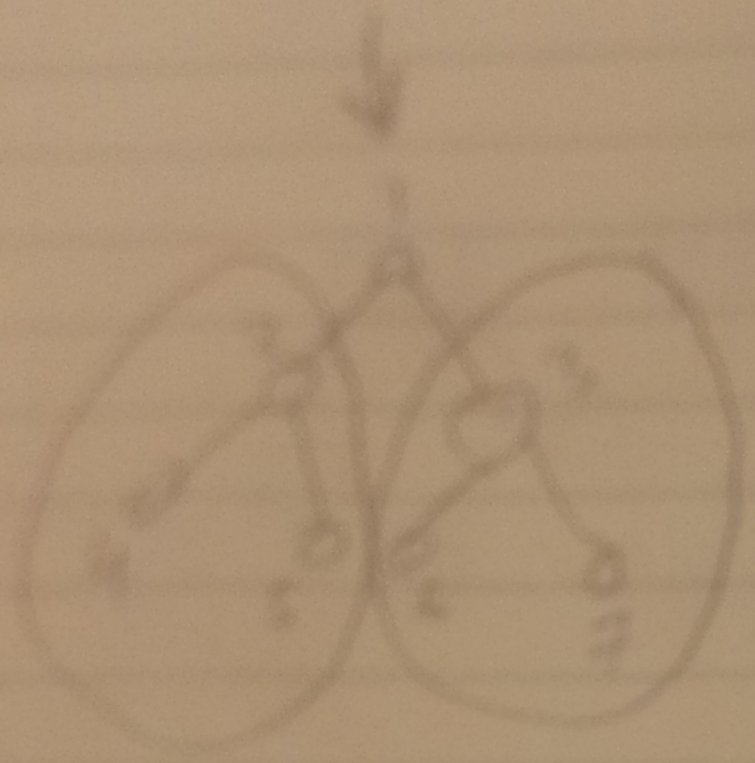
- algorithm that runs in polynomial time and outputs a solution close to the optimal solution
- can guarantee global / some sort of optimization

eg 2-approx algos can guarantee not-bad-than-twice-the-optimal solution

Greedy

- algorithm that produces an output by choosing locally optimal solution for each iteration
- cannot guarantee any sort of global optimization

eg. local optimization might not lead to any sort of global optimization as it might miss solutions that are locally non-optimal but lead to global optimization



Pre Order : Root → Left child → Right child

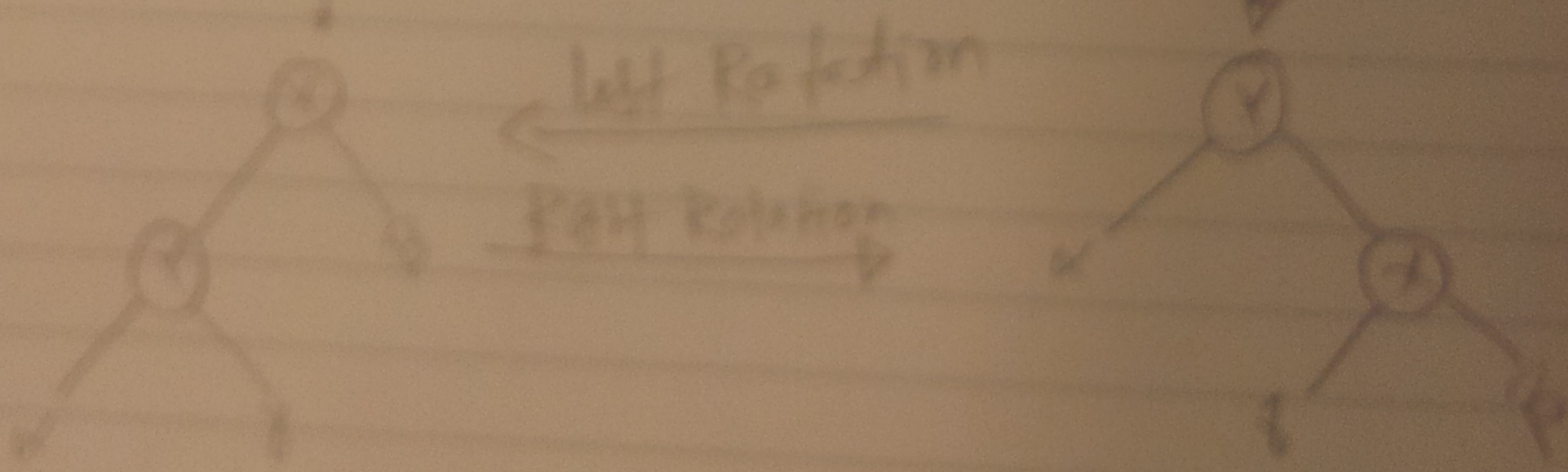
1 → 2 → 4 → 5 → 3 → 6 → 7

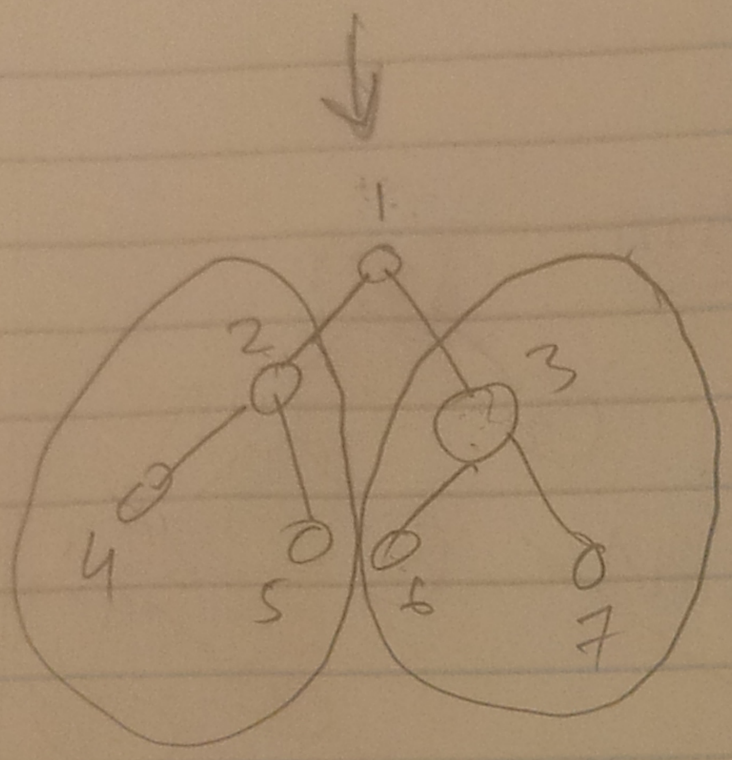
In Order : Left → Root → Right

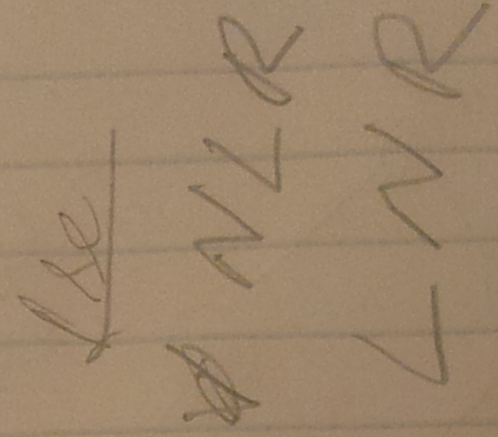
4 → 2 → 5 → 1 → 6 → 3 → 7

Post Order : Left → Right → Root

4 → 5 → 2 → 6 → 7 → 3 → 1







Pre Order : Root → left child → Right child

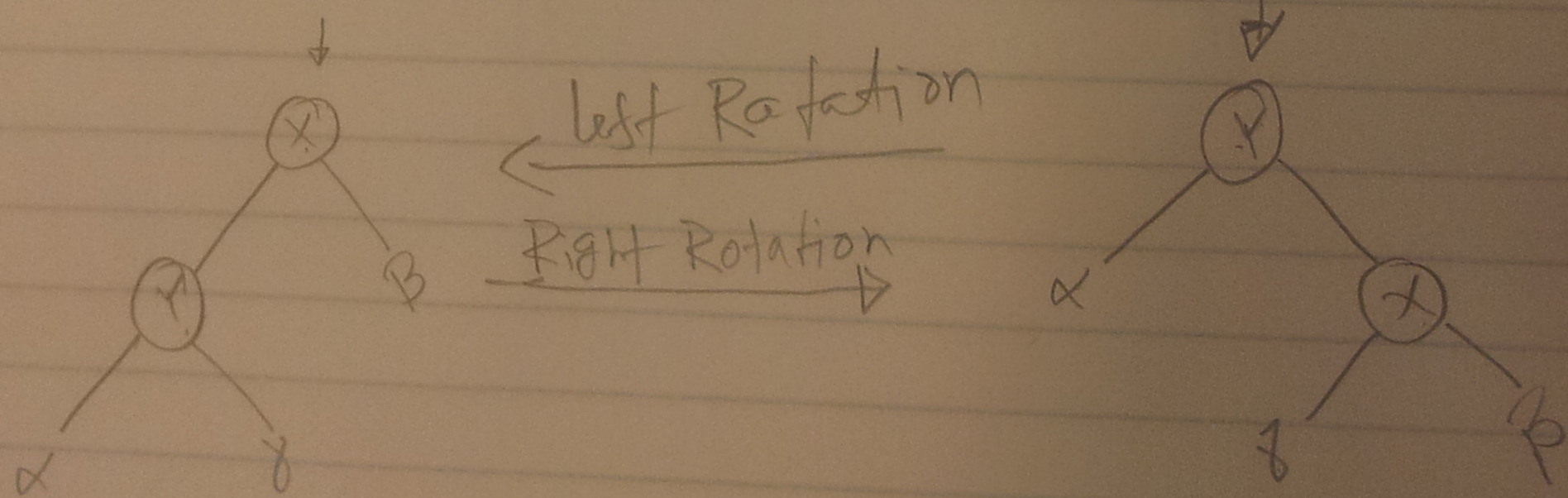
1 → 2 → 4 → 5 → 3 → 6 → 7

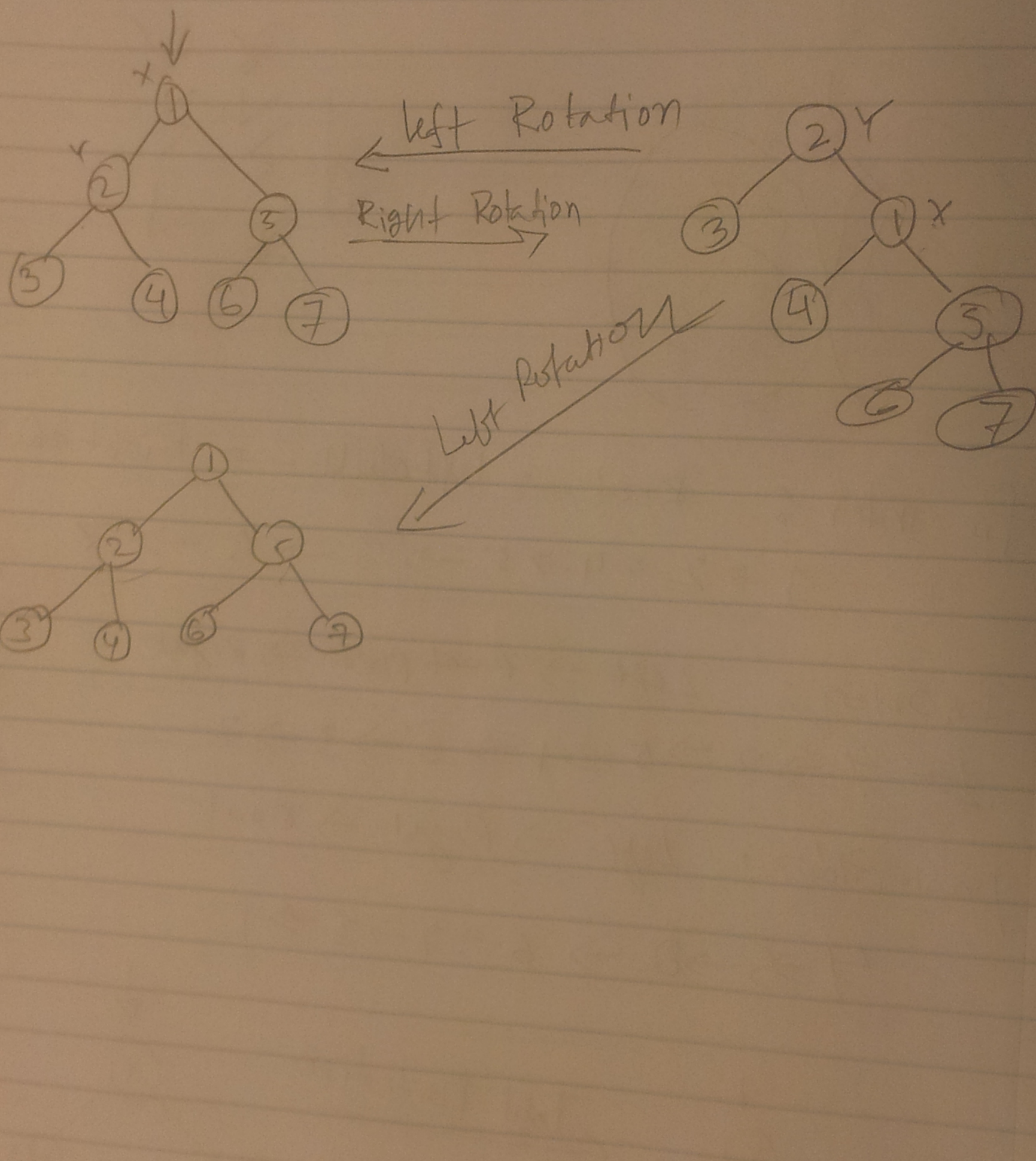
In Order : Left → ~~Root~~ Root → Right

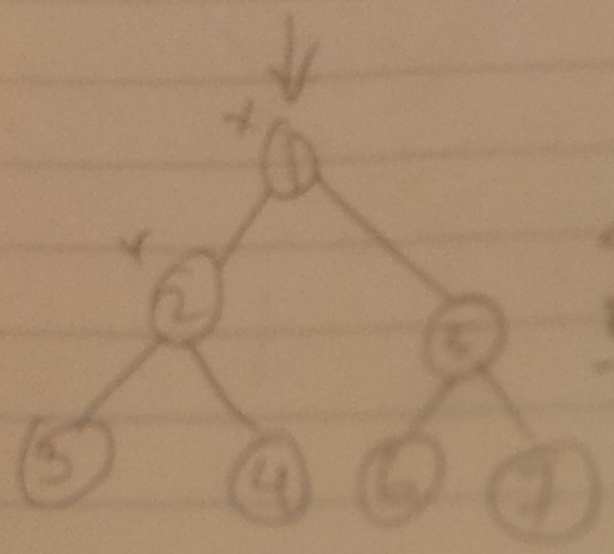
~~4~~ 4 → 2 → 5 → 1 → 6 → 3 → 7

post-Order : Left → Right → root

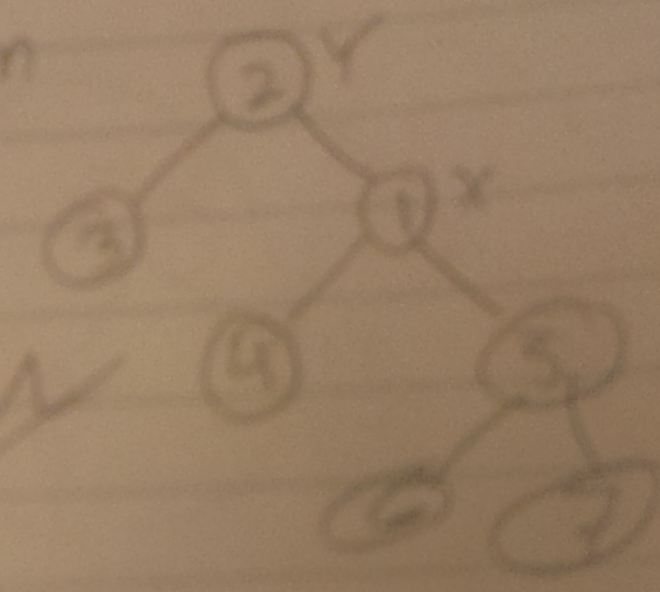
4 → 5 → 2 → 6 → 7 → 3 → 1



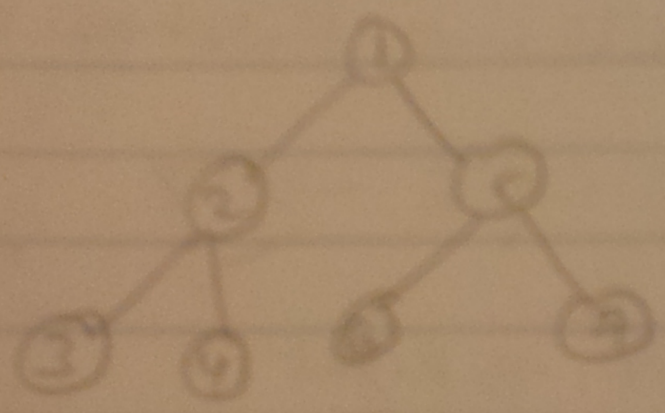

x
y
1
2
3
3
4
6
7
left Rotation
Right Rotation
2
y
3
1
x
4
5
6
7
Left Rotation
1
2
3
3
4
6
7

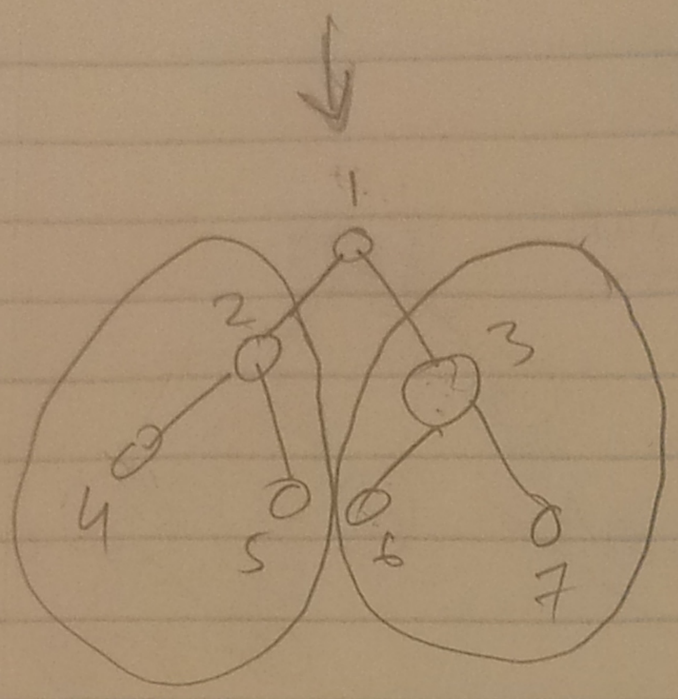


Left Rotation



Right Rotation





PreOrder : Root → left child → Right child

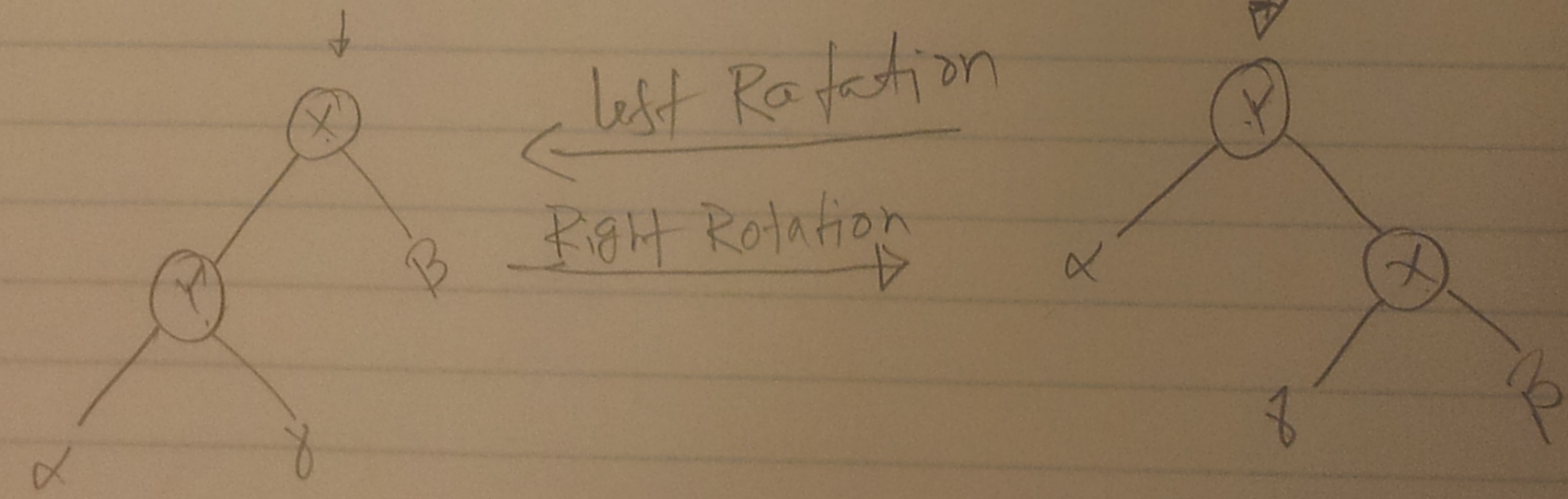
1 → 2 → 4 → 5 → 3 → 6 → 7

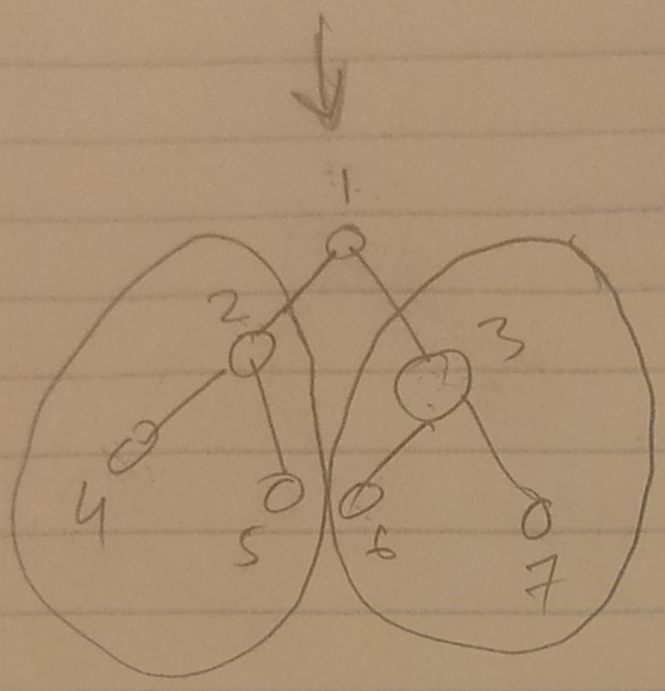
InOrder : Left → ~~Root~~ Root → Right

4 → 2 → 5 → 1 → 6 → 3 → 7

post-Order : Left → Right → root

4 → 5 → 2 → 6 → 7 → 3 → 1





Pre Order : Root → left child → Right child

1 → 2 → 4 → 5 → 3 → 6 → 7

In Order : Left → ~~Root~~ Root → Right

~~10~~ 4 → 2 → 5 → 1 → 6 → 3 → 7

post-Order : Left → Right → root

4 → 5 → 2 → 6 → 7 → 3 → 1