30-06-2025
Monday

* ANALYSIS OF ALGORITHMS *

*** Algorithm:**

Algorithm is a finite sequence of steps to be followed to complete a particular task.

*** Characteristics of an algorithm:**

1. Input   3. Finiteness   5. Effectiveness
2. Output   4. Definiteness

*** Performance of an algorithm:**

Can be analysed in two ways:

1. Time Complexity   2. Space Complexity.

*** Order of growth:**

Best Algorithm:

Consider two algorithms for a same problem with running times f(n) and g(n). which one is better?

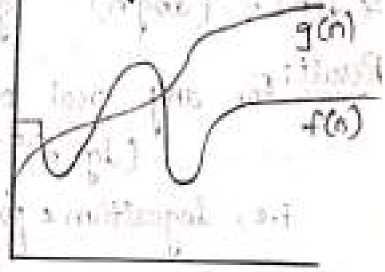
Compare f(n) and g(n) asymptotically (i.e, for large n).

if $f(n) \leq g(n)$ $(n \rightarrow \infty)$

f(n) is best choice

else

g(n) is best choice



	$f(n)=n$	$g(n)=n^2$
1		
2	2	4
3	3	9

- $f(n) \leq g(n) \quad \forall n \geq 1$
- $n^k \leq n^m$ if $k \leq m$

	$f(n)=n^2$	$g(n)=n^3$
1	1	
2	4	9

- $f(n)=n^2$ $g(n)=2^n$

1	1	2
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64

$n^2 \leq 2^n \quad \forall n \geq 4$

- $f(n)=n^3$ $g(n)=2^n$

1	1	2
2	8	4
3	27	8
4	64	16
5	125	32
6	216	64
7	343	128
8	512	256
9	721	512
10	1000	1024
11	1331	2048

$n^3 \leq 2^n \quad \forall n \geq 10$

* Mathematical preliminaries:

$a = b^x \Rightarrow x = \log_b a$ $\quad [\text{Eg:- } 2^x = n \Rightarrow x = \log_2 n]$

* Natural algorithm : $\ln a = \log_e a$

* Binary algorithm : $\lg a = \log_2 a$

<u>Result:</u> **VIP $\log(n!) \approx n \log n$

* $\log_y x = \frac{\log x}{\log y}$ Eg:- $\log_2 64 = \frac{\log 64}{\log 2} = 6$

* $\log x^m = m \log x$ Eg:- $\log_2 64 = \log_2 2^6 = 6 \cdot \log_2 2 = 6 \cdot 1 = 6$

* For $n \in z^+$, $1 + \frac{1}{2} + \frac{1}{3} + \cdots\cdots + \frac{1}{n} \approx \ln \cdot n = \log_e n$

* For all $p \geq 0$, $1^p + 2^p + 3^p + \cdots\cdots\cdots + n^p \approx \frac{1}{p+1} n^{p+1}$

* $f(n) = \log_2 n$ $\quad g(n) = n$

	$f(n)=\log_2 n$	$g(n)=n$
1	0	1
2	1	2
4	2	4
8	3	8

$\log n \leq n$

* $f(n) = (\log n)^2$ $\quad g(n) = n$

	$f(n)=(\log n)^2$	$g(n)=n$
1	0	1
2	1	2
4	$(\log_2 4)^2 = 2^2 = 4$	4
8	9	8
16	16	16
32	25	32
64	36	64

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

* $f(n) = (\log n)^2$ $\quad g(n) = \sqrt{n}$ $\Rightarrow$ $(\log n)^2 \leq \sqrt{n}$

<u>Result:</u> For any real constants $a>0, b>0, c>1$,

$(\lg n)^a < n^b < c^n$ $\rightarrow (\log n)^{100} \leq n^{0.2}, (\log n)^{10059} \leq n^2$

i.e, logarithm < polynomial < exponent $\rightarrow n^2 \leq 2^n$, $n^{200} \leq 2^n$, $n^{200000} \leq 3^n$

$n^{10000} \leq (1.00002)^n$

$22255 \leq \log n$

- $f(n) = 2^n$ $\quad g(n) = n!$

	$f(n)=2^n$	$g(n)=n!$
1	2	1
2	4	2
3	8	6
4	16	24
5	32	120
6	64	720

$2^n \leq n!$

$(22)^n \leq n!$

$(100)^n \leq n!$

- $f(n) = n!$
  $g(n) = n^n$ $\Rightarrow n! \leq n^n$

<u>Result:</u> For any constant $c>1$,

$c^n < n! < n^n$

For any constants $a>0, b>0, c>1$,

$(\lg n)^a < n^b < c^n < n! < n^n$

* Dominance ranking of basic functions:-

$\frac{1}{n^2}$ $\quad \frac{1}{n}$ $\quad 1$ $\quad 100$ $\quad 225$ $\quad \log(\log n)$ $\quad \log n$ $\quad n^{0.1}$ $\quad n^{0.2}$ $\quad \sqrt{n}$ $\quad n^{0.7}$ $\quad n$

$n.\log n$ $\quad n^2 \log(\log n)$ $\quad n^2 \log n$ $\quad n^{2.5}$ $\quad n^3$ $\quad n^{100}$ $\quad (1.2)^n$ $\quad 2^n$ $\quad 3^n$ $\quad (24)^n$

$(50)^n$ $\quad (100)^n$ $\quad n!$ $\quad n^n$

Q. $10, \sqrt{n}, n, \log_2 n, \frac{100}{n}$

$\frac{100}{n}, 10, \log_2 n, \sqrt{n}, n$

01-07-2025 * Frequently used functions:- e^n value = $(2.7)^n$

Tuesday

$1 < \log n < n < n\log n < n^2 < n^3 < 2^n$

Eg: 1) $(\log n)^{22} < n^{0.1}$ 3) $(254)^n < n!$

2) $n^{20022} < 2^n$ 4) $n! < n^n$

Q: Ⓐ $n^{1/3}$ Ⓑ e^n Ⓒ $n^{7/4}$ Ⓓ $n\log^9 n$ Ⓔ 1.0000001^n

A, D, C, E, B $n^{1/3}$ $n\log^9 n$ $n^{7/4}$ $(1.0000001)^n$ e^n

Q: $a(n) = 2^n$, $b(n) = n^{3/2}$, $c(n) = n\log_2 n$ $\underbrace{n\log^9 n \quad n \cdot n^{3/4}}$, $d(n) = n^{\log n}$

c b d a

$\log\square \mid \log\square$

$n\log_2 n$ $n^{3/2}$ $n^{\log n}$ 2^n

$n\log_2 n$ $n \cdot n^{1/2}$ 2^n $n^{\log_2 n}$

$n^{3/2}$ $n^{\log n}$ $\log 2^n$ $\log_2 n^{\log_2 n}$

$n\log_2 2$ $\log_2 n \cdot \log_2 n$

$n \cdot 1$ $(\log_2 n)^2$

→ base same powers lo 3/2 is constant so $n^{\log n}$ is biggest.

Q: $f_1 = 10^n$, $f_2 = n^{\log n}$, $f_3 = n^{\sqrt{n}}$

$\underline{n\sqrt{n}}$ $\underline{10^n}$

$n^{\log n}$ $n^{\sqrt{n}}$ 10^n $\log_{10} n^{\sqrt{n}}$ $\log_{10} 10^n$

f_2 f_3 f_1 $\sqrt{n}\log_{10} n$ $n\log_{10} 10$

Q: $f_1 = n^{3.5}$, $f_2 = \sqrt{2}n$, $f_3 = (\log n)^{55}$, $f_4 = n\log n$, $f_5 = 55^n$, $f_6 = n^3\log n$.

$n^{0.5} \cdot \log_{10} n$ n

$n^{0.5} \cdot \log_{10} n$ $n^{0.5} \cdot n^{0.5}$

$(\log n)^{55}$ $\sqrt{2}n$ $n\log n$ $n^3\log n$ $n^{3.5}$ 55^n

f_3 f_2 f_4 f_6 f_1 f_5

Q: Two alternative packages A & B are avaliable for processing a databay having 10^k records. Package A requires $0.0001n^2$ time units & B requires $10n\log_{10} n$ time units to process n records. what is the Smallest value of K for which package B will be prefered over A?

A) 12 B) 10 c) 6 D) 5

$\boxed{A}$ $\boxed{B}$

$0.0001\,n^2$ $10 \cdot n \cdot \log_{10} n$ 10^k records

$B \le A$

$$10 \cdot n \cdot \log_{10} n \le 0.0001\,n^2$$

$$10 \cdot 10^k \log_{10} 10^k \le 10^{-4}(10^k)^2$$

$$10^{k+1} \cdot k \cdot \log_{10} 10 \le 10^{-4} \cdot 10^{2k}$$

$$10^{k+1} \cdot k \cdot 1 \le 10^{2k-4}$$

$$k \cdot 10^{k+1} \le 10^{2k-4}$$

$$K \le \frac{10^{2k-4}}{10^{k+1}}$$

$$k \le 10^{2k-4-k+1}$$

$$k \le 10^{k-5}$$

$k \le 10^{k-5}$

$k=1$ $\quad 1 \le 10^{1-5}$

$1 \le 10^{-4}$ false

$k=2$ $\quad 2 \le 10^{2-5}$

$2 \le 10^{-3}$ false

$k=3$ $\quad 3 \le 10^{3-5}$

$3 \le 10^{-2}$ false

$k=4$ $\quad 4 \le 10^{4-5}$

$4 \le 10^{-1}$ false

$k=5$ $\quad 5 \le 10^{5-5}$

$5 \le 10^{0}$ false

$k=6$ $\quad 6 \le 10^{6-5}$

$6 \le 10^{1}$

$6 \le 10$ True

* Asymptotic time complexity:

1. Asymptotic time complexity:

Running time of an algorithm as a function of input size n for large n.

2. ~~Memory~~ Asymptotic space complexity:

Memory space requirement of an algorithm as a function of input size n for large n.

* Asymptotic Notation:-

- Asymptotic notation is a way to describe asymptotic complexity.
- Describe the running time / memory space requirement of an algorithm for large n $(n \to \infty)$
- Abstracts away low-order terms and constant factors.

$\underline{f}$

$$10n^3 + \cancel{20n^2} + \cancel{20000}$$

$$\cancel{10}\,n^3$$

$$\boxed{n^3}$$

$\underline{g}$

$$n(\log n)^{10} + 200n^2 + \frac{n}{[illegible]}$$

$$200\,n^2$$

$$\cancel{20}\ \boxed{n^2}$$

* Asymptotically positive:-

The function $f(n)$ is asymptotically positive if $\exists\, n_0$:

$$\forall n \ge n_0 , \quad f(n) > 0 .$$

02-07-2025 * Asymptotic Notation

Wednesday

1. Big-oh Notation (O) Akuva

2. Big-omega Notation (Ω) Takuva

3. Theta Notation (Θ) exact

4. Little-oh Notation (o) equal kantha koncham akuva

5. Little-omega Notation (ω) equal kantha koncham takuva

same ki O, Θ

Eg:1) $f(n) = 4n^2+100n+225$

$f(n) = 4n^2$

$f(n) = n^2$

① $f(n) = O(n^2)$, $f(n) = O(n^3)$, $f(n) = O(2^n)$

② $f(n) = \Omega(n^2)$; $f(n) = \Omega(n)$, $f(n) = \Omega(\log n)$

③ $f(n) = \Theta(n^2)$

④ $f(n) = o(n^3)$, $f(n) = o(n^{2.001})$

⑤ $f(n) = \omega(n)$, $f(n) = \omega(n^{1.899})$.

Eg:2) $f(n) = 2n^3+1524n^2+1000000$

$f(n) = 2n^3$

$f(n) = n^3$

① $f(n) = O(n^3)$, $f(n) = O(2^n)$, $f(n) = O(3^n)$

② $f(n) = \Omega(n^3)$, $f(n) = \Omega(n^2)$; $f(n) = \Omega(n)$ or $\Omega(1)$

③ $f(n) = \Theta(n^3)$

④ $f(n) = o(n^3)$, $f(n) = o(n^{10})$

⑤ $f(n) = \omega(n)$, $f(n) = \omega(\log n)$

Eg:3) $f(n) = 100n^2+2500n$; $g(n) = 24(\log n)^{100} + n^2 \log n$

$\underset{\text{small}}{f(n) = n^2}$ < $\underset{\text{big}}{g(n) = n^2 \log n}$

① $f = O(g)$, $f = o(g)$

② $f = \cancel{\Omega}(g)$, $g = \omega(f)$ ✓ $g = \Omega(f)$

Q: $f(n) = \underset{\text{big}}{2^n}$ and $g(n) = \underset{\text{small}}{n^5}$

① $f(n) = O(g(n))$ ② $f(n) = \Omega(g(n))$ ✓

Q: $f(n) = \underset{\text{big}}{n^2 \log n}$ & $g(n) = \underset{\text{small}}{n(\log n)^{10}}$

① $f(n) = O(g(n))$ ② $f(n) = \Omega(g(n))$ ✓

Q: $f(n) = n^{\log_2 2^n}$ $g(n) = n^n$

① $f(n) = O(g(n))$ ✓ ② $f(n) = \Omega(g(n))$ ✓

$n^{\log_2 2^n}$

$n^{n \log_2 2}$

n^n

Q: $f(n) = 3n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log_2 n}$, $h(n) = n!$

Ⓐ $h(n)$ is $O(f(n))$ Ⓒ $g(n)$ is not $O(f(n))$

Ⓑ $h(n)$ is $O(g(n))$ Ⓓ $f(n)$ is $O(g(n))$

$n^{\sqrt{n}}$ $2^{\sqrt{n}\log_2 n}$ $n!$

$\log_2 n^{\sqrt{n}}$ $\log_2 2^{\sqrt{n}\log_2 n}$ $\log_2 n!$

$\sqrt{n} \cdot \log_2 n$ $\sqrt{n}\log_2 n \cdot \log_2 2$ $n \log n$

$n^{0.5} \log_2 n$ $n^{0.5} \log_2 n$ $n \log_2 n$

$f(n)$ is $O(g(n))$

Q: $f(n) = 2^n$, $g(n) = n!$, $h(n) = n^{\overset{\text{big}}{\log n}}$

$\log_2 2^n$ $\log_2 n!$ $\log_2 n^{\log n}$

$n\log_2^2$ $n\log n$ $\log_2 n \; \log_2^n$

n $n\log_2 n$ $(\log_2 n)^2$

H_2 H_1 H_3

Ⓓ $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$;

Q: $f(n) = \underset{\text{big}}{n^2 \log n}$ $g(n) = \underset{\text{small}}{n(\log n)^{10}}$

A) $f(n) = O(g(n))$ & $g(n) \neq O(f(n))$

B) $f(n) \neq O(g(n))$ & $g(n) \neq O(f(n))$

✓ C) $g(n) = O(f(n))$ & $f(n) \neq O(g(n))$

D) $f(n) = O(g(n))$ & $g(n) = O(f(n))$

Q: $f(n) = \underset{\text{small}}{n}$ $g(n) = \underset{\text{big}}{n^2}$ MSQ Gate

A) $f(n) \in O(g(n))$ B) $f(n) \in \Omega(g(n))$

C) $f(n) \in o(g(n))$ D) $f(n) \in \Theta(g(n))$

07-07-2025

Monday

MICRO SYLLABUS

Course code: 23CS3301 Year: II Semester: I

Course category: Professional core course Branch: CSE Course Type: Theory

Credits: 3 L-T-P: 3-0-0 Prerequisite: Data Structures through c/object oriented programmi

Continous internal evaluation: 30 Semester End Evaluation: 70 Total marks: 100

COURSE OUTCOMES

CO1: Understand the fundamental concepts of algorithm analysis and design techniques.

CO2: Apply various algorithm design techniques for solving problem.

CO3: Apply concepts of trees and graphs for solving problems effectivel

CO4: Analyze the given scenario and choose appropriate algorith design for solving problem.

Syllabus Content

Unit-1. * Analysis of algorithms

* AVL-Tree

* B-Tree

Unit-2. * Heap / Trees

* Graphs

Unit-3. * Divide and conquer

* Greedy Method

Unit-4. * Dynamic programming

Unit-5. * Back tracking

techniques completed — * Branch and Bound

* P & NP (topic)

Books

computer algorithm in c++

Fundamentals of computer algorithm

↓

Rajasekaran

* Algorithm:

- An algorithm is a finite set of instructions that, if followed, accomplishes a task.
- An algorithm is composed of a finite set of steps, each of which may require one or more operations.

* Characterstics of an algorithm:

1. Input: takes zero or more inputs.
2. Output: Output will be atleast once [1 or more], result based on the provided inputs.
3. Definiteness: Each step or instruction must be clear, unambiguous and precisely defined. only one answer
4. Finiteness: It should terminate false or true. But process should be stop.
5. Effectiveness: Each instruction must be feasible. (perfect)

* Order of growth of functions:

$$1 < \log(\log n) < \log n < \sqrt{n} < n$$

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

✓ $< n^3 < 2^n < n! < n^n$

* How to analysis an algorithm? or write about performance algorithm? evaluation

- Performance of an algorithm is measured in 2 ways:

1. Time complexity:

The amount of time that is needed to execute an algorithm is called time complexity. In general, time complexity of an algorithm is measured in terms of the number of basic operations performed by an algorithm.

Eg:- factorial of a number.

Algorithm/Pseudo code	S/e	freq	total
Algorithm Fact(n)	0	–	0
{	0	–	0
f:=1;	1	1	1
for i:=1 to n do	1	n+1	n+1
f:=f*i;	2	n	2n
return (f);	1	1	1
}	0	–	0

Time complexity = 3n+3
$\therefore$ = O(n)

2. Space complexity:

- The amount of memory or space that is needed to execute an algorithm is called space complexity.
- The space complexity S(P) of any algorithm P can be written as S(P) = C+Sp(I)
  where C: constant that denotes fixed part.
  Sp(I): Variable part that depends on instance characteristics (I)

Eg:- Sum of numbers in an array.

Pseudo code

```
//a: Array of size n
Algorithm ArraySum (a,n)
{
    S:=0;
    for i:=1 to n do
        S:=s+a[i];
    return (s);
}
```

constant space:
1) Code : Assume 100 bytes
2) 3 variables: 3×4 = 12 bytes
Total: 112 bytes.

Variable space:
Sp(I) = 4 × n = 4n bytes
$\therefore$ Space complexity = C+Sp(I)
= 112+4n

* Explain the methods to calculate T.C?

- The amount of time that is needed to execute an algorithm is called time complexity.
- In general, time complexity of a algorithm is measured in terms of number of operations performed by algorithm.
- There are two methods to calculate the number of operations performed by a algorithm.

1) __By using a count variable:__

- In this method, we introduce a new variable count, into the program.
- This is a global variable with initial value 0.
- Statement to increament count variable by the appropriate amount are introduce into the program. This is done so that each time a statement in the original program is executed, count is incremented.

Eg: Factorial of a number.

Pseudo code:

```
Algorithm Fact(n)
{
    f:=1;
    for i:=1 to n do
        f:=f*i;
    return (f);
}
```

After introducting count variable:

```
Algorithm Fact(n)
{
    f:=1;
    Count = count+1; // For assignment
    for i:=1 to n do
    {
        count = count+1; // For for loop
        f:=f*i;
        count = count+2; // For assignment
    }
    count = count+1; // For last time of for loo
    count = count+1; // For return
    return (f);
}
```

2) __By using step count method:__

- In this method, to determine the step count of an algorithm, we build a table in which we list the total number of steps contributed by each statement. we use the following 3 quantities:

s/e : Number of operations per execution of the Statement.

freq : No: of time, the statement is executed.

total: s/e * freq

Eg: Factorial of a number.

Pseudo code	S/e	freq	total
Algorithm Fact(n)	0	←	0
{	0	—	0
f:=1;		1	1
for i:=1 to n do	1	n+1	n+1
f:=f*i;	2	n	2n
return (f);	1	1	1
}	0	—	0

Time Complexity = 3n+3
= O(n)

08-07-2025 *write about asymptotic notations.

Tuesday

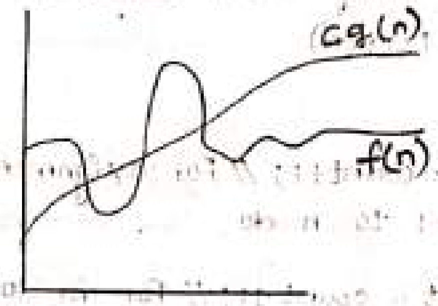
- Asymptotic time complexity is the running time of an algorithm as a function of input size n for large n.
- Asymptotic notation is a way to describe asymptotic complexity
- There are 5 asymptotic notations:-

1. **Big-oh Notation (O):**

f(n) is O(g(n)) means

there exist positive constants c and n_0, such that

$f(n) \leq cg(n) \ \forall \ n \geq n_0$



- g(n) has larger or equal order of growth as f(n).
- g(n) is an asymptotic upper bound for f(n)

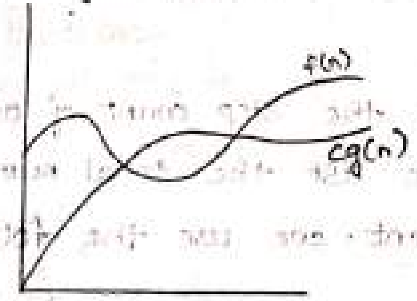
Eg: $2n^2+5n = O(n^2)$

2. **Big-omega Notation (Ω):**

f(n) ∈ Ω(g(n)) means

there exist positive constants c and n_0, such that

$0 \leq cg(n) \leq f(n) \ \forall n \geq n_0$



- g(n) has smaller or equal order of growth as f(n)
- g(n) is asymptotic lower bound for f(n)

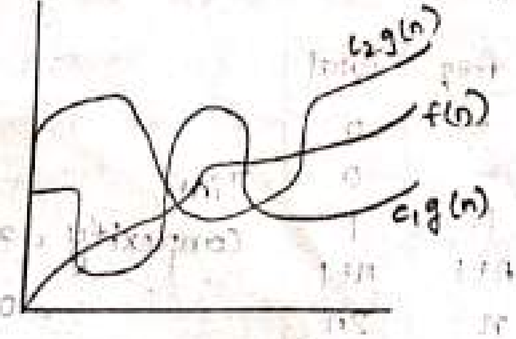
Eg:- $2n^2+5n = \Omega(n^2)$

3. **Theta Notation (θ):**

f(n) is θ(g(n)) means

there exist +ve constants c_1, c_2 & n_0, such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall n \geq n_0$



- f(n) grows with same order as g(n).
- g(n) is an asymptotically tight bound for f(n)

Eg:- $2n^2+5n = \Theta(n^2)$.

* Small Notation:-

o - notation:

$f(n)$ is $o(g(n))$ if $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Eg:- $2n^2+5n = o(n^3)$

ω - notation:

$f(n)$ is $\omega(g(n))$ if $\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = 0$

Eg:- $2n^2+5n = \omega(n)$

* Problems on notation:

1. Prove that $4n^2+20n+100 = O(n^2)$

* Def $f(n) = O(g(n))$ means $\exists\ c, n_0$ such that

$f(n) \le c \cdot g(n) \ \forall\ n \ge n_0$

* Suppose $f(n) = 4n^2+20n+100$

$n \ge 100$

$100 \le n$

$\therefore f(n) = 4n^2+20n+100$

$\le 4n^2+20n+n \quad \forall n \ge 100$ $\qquad n \ge 21$

$= 4n^2+21n \quad \forall n \ge 100$ $\qquad 21 \le n$

$\le 4n^2+n \cdot n \quad \forall n \ge \max\{100, 21\}$

$= 4n^2+n^2$

$= 5n^2 \quad \forall n \ge 100$

$\therefore f(n) \le 5n^2 \ \forall n \ge 100$

So $c=5$, $n_0 = 100$

Hence $4n^2+20n+100 = O(n^2)$

2. If $f(n) = 3n^2+5n+10$ then P.T $f(n) = O(n^2)$

$f(n) = 3n^2+5n+10$ $\qquad n \ge 10$

$\le 3n^2+5n+n \quad \forall n \ge 10$ $\qquad 10 \le n$

$= 3n^2+6n \quad \forall n \ge 10$ $\qquad n \ge 6$

$\le 3n^2+n \cdot n \quad \forall n \ge \max\{10, 6\}$ $\qquad 6 \le n$

$= 3n^2+n^2 \quad \forall n \ge 10$ $\qquad 6 \cdot n \le n \cdot n$

$= 4n^2 \quad \forall n \ge 10$

$\therefore f(n) \le 4n^2 \ \forall n \ge 10$

So $c=4$, $n_0 = 10$

Hence $f(n) = O(n^2)$

3. PT $5n^2+40n+20 = O(n^2)$

Suppose $f(n) = 5n^2+40n+20$ $\qquad n \ge 20$

$\le 5n^2+40n+n \quad \forall n \ge 20$ $\qquad 20 \le n$

$= 5n^2+41n \quad \forall n \ge 20$ $\qquad n \ge 41$, $41 \le n$

$\le 5n^2+n \cdot n \quad \forall n \ge \max\{20, 41\}$ $\qquad 41 \cdot n \le n \cdot n$

$\le 5n^2 + n\cdot n \quad \forall n \ge \max\{20, 41\}$

$= 5n^2 + n^2$ ~~$\forall n \ge 41$~~

$f(n) \le 6n^2 \quad \forall n \ge 41$

So $c = 6$, $n_0 = 41$

Hence $5n^2 + 40n + 20 = O(n^2)$

4. PT $4n^2 + 20n + 5 = O(n^3)$

$f(n) = 4n^2 + 20n + 5$

$\le 4n^2 + 20n + n \quad \forall n \ge 5$

$= 4n^2 + 21n \quad \forall n \ge 5$

$\le 4n^2 + n\cdot n \quad \forall n \ge 21$

$= 4n^2 + n^2$ $\boxed{n^2 \le n^3 \; \forall n \ge 1}$

$= 5n^2$

$\le 5n^3 \quad \forall n \ge \max\{21, 1\}$

$= 5n^3 \quad \forall n \ge 21$

$\therefore 4n^2 + 20n + 5 \le 5n^3 \; \forall n \ge 21$

So $c = 5$, $n_0 = 21$

Hence $4n^2 + 20n + 5 = O(n^3)$

09-07-2025
Wednesday

1) PT $4n^2 + 50n + 20 = O(2^n)$

$f(n) = 4n^2 + 50n + 20$

$f(n) \le 4n^2 + 50n + n \quad \forall n \ge 20$

$= 4n^2 + 51n \quad \forall n \ge 20$

$\le 4n^2 + n\cdot n \quad \forall n \ge \max\{20, 51\}$

$= 4n^2 + n^2 \quad \forall n \ge 51$

WKT

$= 5n^2 \quad \forall n \ge 51$ $\boxed{n^2 \le 2^n \; \forall n \ge 4}$

$\le 5\cdot 2^n \quad \forall n \ge \max\{51, 4\}$

$= 5\cdot 2^n \quad \forall n \ge 51$

$4n^2 + 50n + 20 \le 5\cdot 2^n \; \forall n \ge 51$

So $c = 5$, $n_0 = 51$

$\therefore 4n^2 + 50n + 20 = O(2^n)$

2) $5\cdot 2^n + 40n^2 + 3n = O(2^n)$

$f(n) = 5\cdot 2^n + 40n^2 + 3n$

$\le 5\cdot 2^n + 40n^2 + n^2 \quad \forall n \ge 3$

$= 5\cdot 2^n + 41n^2 \quad \forall n \ge 3$

$\le 5\cdot 2^n + n\cdot n^2 \quad \forall n \ge 41$

$= 5\cdot 2^n + n^3 \quad \forall n \ge 41$

$= 5\cdot 2^n + 2^n \quad \forall n \ge 41$

$= 6\cdot 2^n \quad \forall n \ge 41$

$f(n) \le 6\cdot 2^n \quad \forall n \ge 41$

So $c = 6$, $n_0 = 41$

3) P.T $5n^2 - 20n + 35 = O(n^2)$

$f(n) = 5n^2 - 20n + 35$

$\le 5n^2 - 20n + n \quad \forall n \ge 35$

$= 5n^2 - 19n \quad \forall n \ge 35$

$\le 5n^2 \quad \forall n \ge 35$

$f(n) \le 5n^2 - 20n + 35 \le 5n^2 \quad \forall n \ge 35$

4) P.T $4n^2 + 10n - 22 = O(n^2)$

$f(n) = 4n^2 + 10n - 22$

$\le 4n^2 + 10n$ $\boxed{\begin{matrix} n \ge 10 \\ 10 \le n \end{matrix}}$

$= 4n^2 + n\cdot n \quad \forall n \ge 10$

$= 5n^2 \quad \forall n \ge 10$

$\therefore f(n) \le 5n^2 \; \forall n \ge 10$

So, $c = 5$, $n_0 = 10$

Hence $4n^2 + 10n - 22 = O(n^2)$

* Problems on omega notation:

1. PT $5n^2 + 2n + 30 = \Omega(n^2)$

w.k.T $n^2 \le n^2$

$\le 5\cdot n^2$

$\le 5n^2 + 2n$

$\le 5n^2 + 2n + 30$

$\therefore n^2 \le 5n^2 + 2n + 30 \; \forall n \ge 1$

So $c = 1$, $n_0 = 1$

$\therefore 5n^2 + 2n + 30 = \Omega(n^2)$

2. PT $4n^2 - 50n + 22 = \Omega(n^2)$

w.k.T $n^2 \le n^2$

$\Rightarrow n^2 \le 4n^2$

$\Rightarrow \boxed{n^2 \le 4n^2 - 50n} \; \forall n \ge 17$

$\Rightarrow n^2 \le 4n^2 - 50n + 22 \; \forall n \ge 17$

So $c = 1$, $n_0 = 17$

$\therefore 4n^2 - 50n + 22 = \Omega(n^2)$

$n^2 \le 4n^2 - 50n$

$n^2 - 4n^2 \le -50n$

$-3n^2 \le -50n$

$50n \le 3n^2$

$50 \le 3n$

$\frac{50}{3} \le n$

$16.6 \le n$

$17 \le n$

*

1.

(

*

* Problems on Theta Notation:

1. PT $3n^2+20n+100 = \Theta(n^2)$

①: $f(n) = 3n^2+20n+100$

$\le 3n^2+20n+n \quad \forall n \ge 100$

$\le 3n^2+21n \quad \forall n \ge 100$

$\le 3n^2+n.n \quad \forall n \ge \max\{100, 21\}$

$\le 4n^2 \quad \forall n \ge 100$

$\therefore \boxed{f(n) \le 4n^2 \quad \forall n \ge 100} \longrightarrow ①$

②: WKT $n^2 \le n^2 \quad \forall n \ge 1$

$n^2 \le 3n^2 \quad \forall n \ge 1$

$n^2 \le 3n^2+20n \quad \forall n \ge 1$

$n^2 \le 3n^2+20n+100 \quad \forall n \ge 1$

$\boxed{1.n^2 \le f(n) \quad \forall n \ge 1} \longrightarrow ②$

from (1) & (2);

$1.n^2 \le f(n) \le 4n^2 \quad \forall n \ge \max\{100, 1\}$

$1.n^2 \le f(n) \le 4n^2 \quad \forall n \ge 100$

so, $c_1 = 1$, $c_2 = 4$, $n_0 = 100$

Hence $f(n) = \Theta(n^2)$.

$\therefore 3n^2+20n+100 = \Theta(n^2)$

* Problems on Small notation:

1. PT $4n^2+30n+100 = o(n^3)$

Def we say $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Suppose $f(n) = 4n^2+30n+100$

$g(n) = n^3$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4n^2+30n+100}{n^3}$$

$$= \lim_{n\to\infty} \frac{n^2\left[4+\frac{30}{n}+\frac{100}{n^2}\right]}{n^3}$$

$$= \lim_{n\to\infty} \frac{1}{n}\left[4+\frac{30}{n}+\frac{100}{n^2}\right]$$

$$= \lim_{n\to\infty} \left(\frac{4}{n}+\frac{30}{n^2}+\frac{100}{n^3}\right)$$

$$= 0+0+0 = 0$$

[omega(ω)]

2. PT $5n^2+20n+10 = \omega(n)$

Def we say that $f(n) = \omega(g(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$

Suppose $f(n) = 5n^2+20n+10$

$g(n) = n$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n}{5n^2+20n+10}$$

$$= \lim_{n\to\infty} \frac{n}{n^2\left[5+\frac{20}{n}+\frac{10}{n^2}\right]}$$

$$= \lim_{n\to\infty} \frac{1}{n\left[5+\frac{20}{n}+\frac{10}{n^2}\right]}$$

Ex

$$= \lim_{n\to\infty} \frac{1}{n} \cdot \frac{1}{5+\frac{20}{n}+\frac{10}{n^2}}$$

$$= 0 \cdot \frac{1}{5+0+0}$$

$$= 0$$

* Examples on Time complexity:-

- Example 01: Addition of two numbers

Algorithm/Pseudocode	S/e	freq	total
Algorithm Sum(m,n)	0	-	0
{	0	-	0
S:=m+n;	2	1	2
return(s);	1	1	1
}	0	-	0

Time complexity = 3
= O(1)

- If there is no loops, TC is order of 1 [TC = O(1)]

Example 02: Sum of numbers in an array

Algorithm/Pseudocode	S/e	freq	total
//a: Array of size n	0	-	0
Algorithm ArraySum(a,n)	0	-	0
{	0	-	0
S:=0;	1	1	1
for i:=1 to n do	1	n+1	n+1
S:=S+a[i];	2	n	2n
return (S);	1	1	1
}	0	-	0

Time complexity = 3n+3
= O(n)

- If there is a single ~~too~~ for loop ∴ Time complexity = O(n)

Example 03: Reversing an array

Algorithm / Pseudocode	s/e	freq	total
//a: Array of size n	0	–	0
Algorithm ReverseArray(a,n)	0	–	0
{	0	–	0
for i:=1 to n/2 do	1	n/2+1	n/2+1
{	0	–	0
temp := a[i];	1	n/2	n/2
a[i]:= a[n-i+1];	1	n/2	n/2
a[n-i+1] = temp;	1	n/2	n/2
}	0	–	0
}	0	–	0

$$\therefore TC = 2n+1$$
$$= \Theta(n)$$

Example 04: Matrix addition

Algorithm/ Pseudocode	s/e	freq	total
//A,B: Matrices of order m×n	0	–	0
//c: Matrix of order m×n to store A+B	0	–	0
Algorithm MatrixAddition(a,b,c,m,n)	0	–	0
{	0	–	0
for i:=1 to m do	1	m+1	m+1
for j:=1 to n do	1	m(n+1)	mn+m
c[i,j]:= A[i,j]+B[i,j];	2	mn	2mn
}	0	–	0

$$\therefore TC = 3mn+2m+1$$
$$= \Theta(mn)$$

- for nested loop:

  loop 1 : m times
  loop 2 : n times $\rightarrow O(m,n)$

Example 05: Matrix addition of square matrices

Algorithm / Pseudocode	S/e	freq	total
//A,B: Matrices of order nxn	0	–	0
//C: Matrix of order nxn to store A+B	0		0
Algorithm MatrixAddition (a,b,c,n)	0	–	0
{	0	–	0
for i:=1 to n: do	1	n+1	n+1
for j:=1 to n do	1	n(n+1)	n^2+n
c[i,j]:=A[i,j]+B[i,j];	2	n^2	$2n^2$
}	0	–	0

$$TC = 3n^2+2n+1$$
$$= \Theta(n^2)$$

- If there is nested loop, $TC = O(n^2)$

Example 06: Linear Search

Algorithm / Pseudocode	S/e	freq	total
//a: Array of size n	0	–	$\Theta(0)$
//x: Elements to be searched	0	–	$\Theta(0)$
Algorithm LinearSearch (a,n,x)	0	–	$\Theta(0)$
{	0		$\Theta(0)$
for i:=1 to n do	1	$\Theta(n)$	$\Theta(n)$
if (a[i]=x)	1	$\Theta(n)$	$\Theta(n)$
return(i);	1	$\Theta(1)$	$\Theta(1)$
return(-1);	1	$\Theta(1)$	$\Theta(1)$
}	0	–	$\Theta(0)$

$TC = \Theta(n)$.

Example 07: finding maximum in an Array

Algorithm / Pseudocode	S/e	freq	total
//a: Array of size n	0	–	$\Theta(0)$
Algorithm MaxInArray (a,n)	0	–	$\Theta(0)$
{	0	–	$\Theta(0)$
max:=a[1];	1	$\Theta(1)$	$\Theta(1)$
for i:=2 to n do	1	$\Theta(n)$	$\Theta(n)$
if (a[i] > max)	1	$\Theta(n)$	$\Theta(n)$
max:=a[i];	1	$\Theta(n)$	$\Theta(n)$
return (max);	1	$\Theta(1)$	$\Theta(1)$
}	0	–	$\Theta(0)$

$T.C = \Theta(n)$

xample 08: Selection Sort

Algorithm / Pseudocode	s/e	freq	total
//a: Array of size n	0	-	$\Theta(0)$
Algorithm SelectionSort(a,n)	0	-	$\Theta(0)$
{	0	-	$\Theta(0)$
for i:=1 to n-1 do	1	$\Theta(n)$	$\Theta(n)$
{	0	-	$\Theta(0)$
min:=a[i]; min_pos:=i;	2	$\Theta(n)$	$\Theta(n)$
for j:=i+1 to n do	1	$\Theta(n^2)$	$\Theta(n^2)$
{	0	-	$\Theta(0)$
if (a[j]<min)	1	$\Theta(n^2)$	$\Theta(n^2)$
{	0	-	$\Theta(0)$
min:=a[j];min_pos:=j;	2	$\Theta(n^2)$	$\Theta(n^2)$
}	0	-	$\Theta(0)$
}	0	-	$\Theta(0)$
Swap (a[i],a[min_pos]);	3	$\Theta(n)$	$\Theta(n)$
}	0	-	$\Theta(0)$
}	0	-	$\Theta(0)$

$TC = \Theta(n^2)$

Example 09: Bubble Sort

Algorithm/ Pseudocode	s/e	freq	total
//a: Array of size n	0	-	$\Theta(0)$
Algorithm BubbleSort (a,n)	0	-	$\Theta(0)$
{	0	-	$\Theta(0)$
for i:=1 to n-1 do	1	$\Theta(n)$	$\Theta(n)$
{	0	-	$\Theta(0)$
count:=0;	1	$\Theta(n)$	$\Theta(n)$
for j:=1 to n-i do	1	$\Theta(n^2)$	$\Theta(n^2)$
{	0	-	$\Theta(0)$
if (a[j] > a[j+1])	1	$\Theta(n^2)$	$\Theta(n^2)$
{	0	-	$\Theta(0)$
swap (a[j],a[j+1]);	3	$\Theta(n^2)$	$\Theta(n^2)$
Count := Count+1;	2	$\Theta(n^2)$	$\Theta(n^2)$
}	0	-	$\Theta(0)$
}	0	-	$\Theta(0)$
if (count=0)	1	$\Theta(n)$	$\Theta(n)$
break;	1	$\Theta(1)$	$\Theta(1)$
}	0	-	$\Theta(0)$
}	0	-	$\Theta(0)$

$TC = \Theta(n^2)$

10) Ins

10-07-2025 * **Rules for time complexity:**
Thursday

Single Statement:

$\Theta(1)$

Consecutive Statements:

Add the time complexities of each Statement.

if-else Statements:

The test, plus the if part or the else part (whichever is the large

Loops:

Θ(Number of iterations in a loop * Number of statements inside the loop)

*1. only one for loop - $\Theta(n)$

2. Two for loops, not nested - $\Theta(n)$

3. Two for loops, nested - $\Theta(n^2)$

4. Three for loops, not nested - $\Theta(n)$

5. Three for loops, nested - $\Theta(n^3)$.

* **Types of loops:-**

1. **Linear Loop:**

```
for (i=1; i<=n; i=i+c)
        or
for (i=n; i>=1; i=i-c)
```

Time complexity:
$T(n) = \Theta(n)$

2. **Logarithmic Loop:**

```
for (i=1 ; i<=n ; i=i*c) ;
        or
for (i=n ; i>=1 ; i=i/c)
```

Time complexity:

$T(n) = \Theta(\log_c n) = \Theta(\lg n)$

3. **Log (logarithmic) loop**/ expo

```
for (i=1 ; i<=n ; i=i^c)
        or
for (i=n ; i>=k ; i= c√i)
```

Time complexity:

$T(n) = \Theta(\log_c \log_k n) = \Theta(\lg \lg n)$

Eg: What is the complexity of the following c code segment.

1)

```
x=0;
for (i=5 ; i<=n ; i=i^2)
    x=x+1;
```

$TC = \Theta($

2)

```
x=0;
for (i=n ; i>=1 ; i=√i)
    x=x+1;
```

$TC = \Theta($

Example 3)

```
x=5;
for (i=1 ; i<=n ; i=i+1)
    for (j=1 ; j<=n ; j=2*j)
        x=x+1;
```

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

4)

```
x=5;
for (i=1 ; i<=n ; i=5*i)
    for (j=1 ; j<=n ; j=2*j)
        x=x+1
```

$TC = \Theta(\log n)^2$

5)

```
x=5;
for (i=1 ; i<=n ; i=i+i)
    for (j=1 ; j<=n ; j=j+j)
        x=x+1;
```

$TC = \Theta(\log n)^2$

6)

```
x = 5;
for (i=1 ; i<=n ; i=i+1)
    for (j=1 ; j<=n ; j=j+j)
        for (k=1 ; k<=n ; k=k+1)
            x=x+1;
```

$TC = \Theta(n^2 \log n)$

11-07-2025
Friday

* **Recursion:** what is Recursion?

- Process in which a problem is defined in terms of itself.
- Recursion has two cases:

  ① Base case ② Recursion case

n+sum(n-1)

Eg: 1. Factorial of a number n

$$fact(n) = \begin{cases} 1 & \text{if } n=0 \text{ (Base case)} \\ n * fact(n-1) & \text{if } n>1 \text{ (recursion case)} \end{cases}$$

*what is recursive algorithm or function?

- A recursive algorithm is an algorithm that calls itself, one or more times, on smaller inputs.
- To prevent an infinite chain of such calls, there has to be a value of the input for which the algorithm doesn't call itself. *

Eg: Factorial of a number n.

```
Algorithm RFact(n)
{
    if(n=0)
        return(1);
    else
        return(n * RFact(n-1));
}
```

*what are phases of recursion? recursive algorithm

- There are mainly two phases:
  1. Winding phase
  2. Unwinding phase

1. Winding phase:

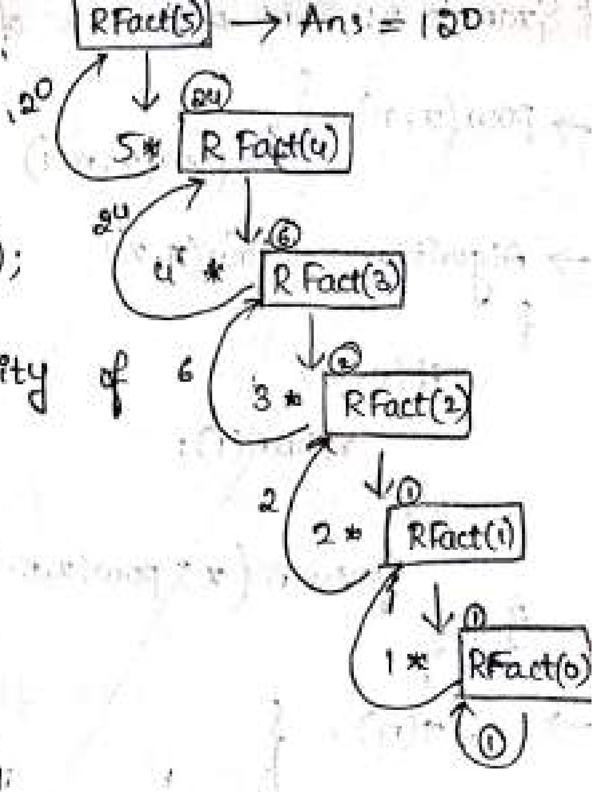
- Function keeps on calling itself and no return statements are executed in this phase.
- This phase terminates when the base case becomes true in a call.

2. Unwinding phase:

- All the recursive functions calls start retuing in reverse order till the first instance of function returns.
- In this phase control returns through each instance of a function.

Eg: Factorial of a number n.

```
Algorithm RFact( )
{
    if(n=0)
        return(1);
    else
        return (n* RFact(n-1));
}
```



* How to calculate time complexity of recursive algorithm?

```
Algorithm RFact(*)
{
    if(n=0)
        return(1);
    else
        return (n* RFact(n-1));
}
```

- To find the time complexity of recursive algorithm first we need to write recursence relation for time complexity.
- Recurrence relation for time complexity:

$$T(n) = \begin{cases} T(n-1) + 3 & \text{if } n>0 \\ 2 & \text{if } n=0 \end{cases}$$

$\underset{\text{if}}{1} + \underset{\text{return}}{1} + \underset{\text{multiplication}}{1} + T(n-1)$

Solution:

$$\begin{aligned} T(n) &= T(n-1)+3 \\ &= T(n-2)+3+3 \\ &= T(n-3)+3+3+3 \\ &\ \ \vdots \\ &= T(n-k)+3k \\ &= T(n-n)+3(n) \\ &= T(0)+3n \\ &= 2+3n \\ &= 3n+2 \end{aligned}$$

$T(n) = \Theta(n)$

$T(n-10)+3(10)$

$T(n-x)+3x$

$T(n-n)+3n$

$T(0)+3n$

$2+3n$

$\Theta(n)$

$T(0)$

$n-x=0$

$x=n$

* Example 3: Sum of first n Natural numbers:-

→ Algorithm RSum(n)

```
{
    if(n=1)
        return(1);
    else
        return(n+sum(n-1));
}
```

→ $sum(n) = \begin{cases} 1 & \text{if } n=1 \text{ (Base case)} \\ n+sum(n-1) & \text{if } n>1 \text{ (recursion case)} \end{cases}$

· Recurrence relation for T.C:-

→ $(n) = \begin{cases} T(n-1)+3 & \text{if } n>1 \\ 2 & \text{if } n=1 \end{cases}$

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

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

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

$= T(n-4)+3+3+3+3$

⋮

$= T(n-k)+3k \qquad \because k=n-1$

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

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

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

$= 2+3(n-1) \qquad \because T(1)=2$

~~= 2 + 3n-3~~ = 3n - 1

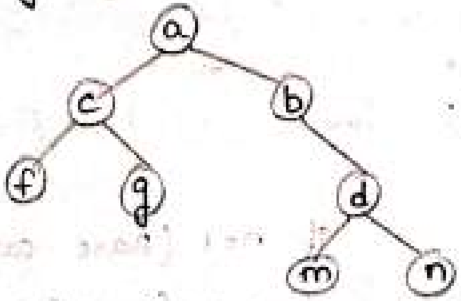
$= O(n)$

14-07-2025 * Binary Tree:

Monday

A binary tree is a hierarchical data structure in which every node has maximum 2 children, called the left child and right child.

Eg:-

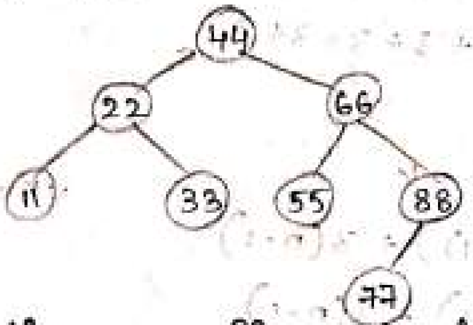


*Define binary search tree:

- A binary search tree (BST) is a binary tree that has a value associated with each of its nodes.
- The values satisfy Binary Search tree property:

1. The values in the non-empty left subtree of a node are Smaller than the value in the node.
2. The values in the non-empty right subtree of a node are greater than the value in the node;

Eg:-



*Major operations on Binary Search tree:

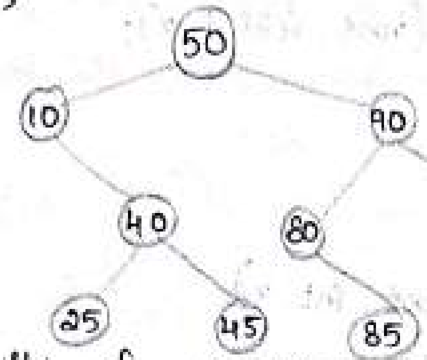
1. Search
2. Insert
3. Delete

*Explain the insertion process in Binary Search tree?

To insert a new node with a data value x

1. Start at the root node.
2. Compare the value x with the current node value
   → If x < current node value, go to left sub tree.
   → If x > current node value, go to right sub tree.
3. Repeat step 2 until we reach an empty slot and then insert the new node in that slot.

*Construct a binary search tree for the data : 50, 10, 90, 80, 99, 40, 85, 25, 45



*Time complexity of operations in BST:

Search:-

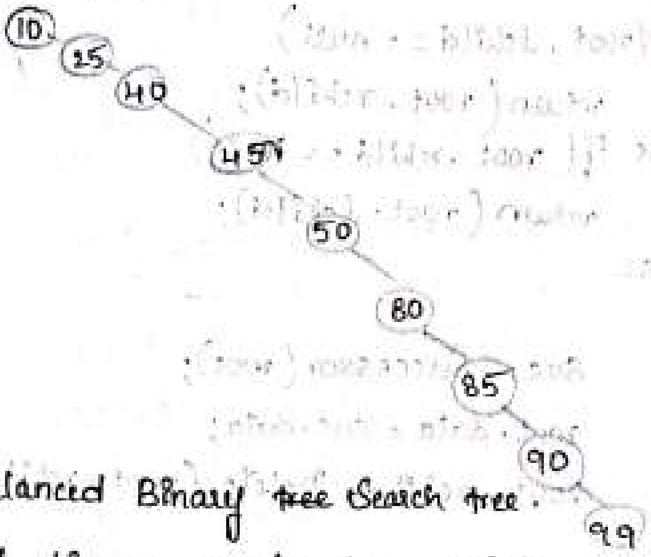
Best case = $O(1)$

worst case = $O(h)$

element not in tree = $O(h)$

Remark: In worst case, a binary search tree with n nodes have a height of $(n-1)$.

Eg: 10, 25, 40, 45, 50, 80, 85, 90, 99



*AVL Tree = Self Balanced Binary ~~tree~~ Search tree.

Note: For a balanced binary search tree with n nodes, the height will be $\Theta(\log_2 n)$

*Insertion:

```
Static Node Insert (Node root , int x)
{
    Node tmp;
    tmp = new Node(x);
    if(root == NULL)
        return(tmp);
    else if(x < root.data)
        root.lchild = Insert (root.lchild, x);
    else if(x>root.data)
        root.rchild = Insert (root.rchild, x);

      return(root);
}
```

*Deletion:

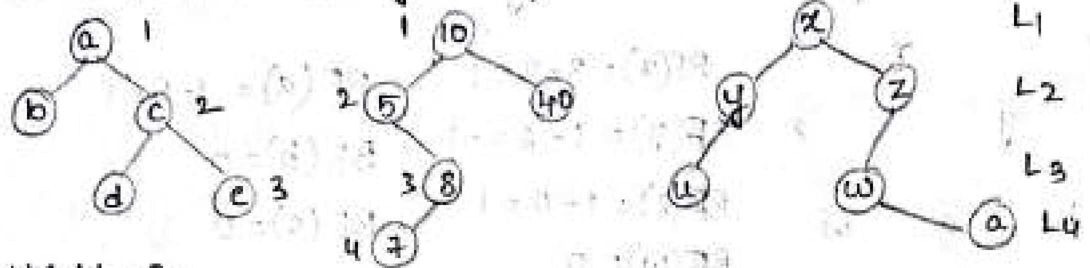
```
 Static  Node  Delete (Node root, int x)
 {
     Node suc;
     if (root = = NULL)
           return(root);
     if (x<root.data)
           root.lchild = Delete (root.lchild, x);
     else if (x>root.data)
           root.rchild = Delete (root.rchild, x);
     else
     {
         if (root.lchild = = null)
               return(root.rchild);
         else if( root.rchild = = null)
               return (root.lchild);
         else
         {
               suc = Successor(root);
               root.data = suc.data;
               root.rchild = Delete (root.rchild, suc.data);
         }
     }
     return (root);
 }
```

*Successor:

```
  Static  Node  Successor (Node root)
  {
      Node  tmp;
      tmp = root.rchild;
      while (tmp.lchild != null)
            tmp = tmp.lchild;
      return(tmp);
  }
```

15-07-2025 * Height of binary tree: (only for AVL Tree)
Tuesday The no.of nodes and the largest ~~pat~~ possible path from root to any leaf node.

Eg:-



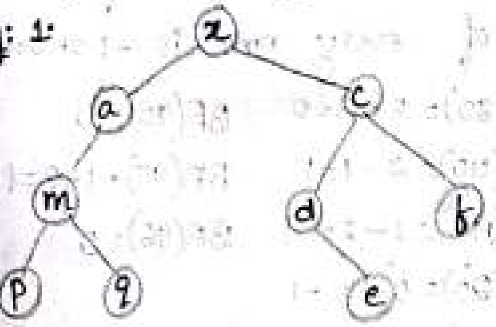
Height = 3 Height = 4 Height = 4

* Balancing factor of a node:

BF(node) = Height of (Left subtree)(node) – Height of (Rightsubtree)(node)

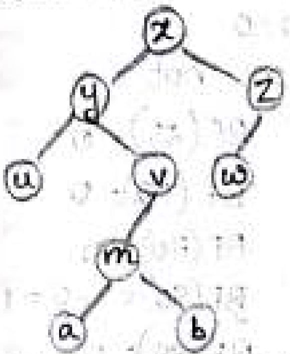
LST RST

Eg: 1:



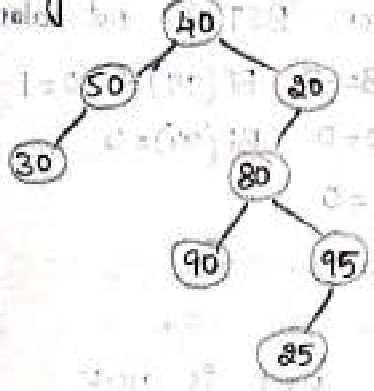
BF(x) = 3-3=0
Bf(a) = 2-0=2
BF(c) = 2-1=1
BF(m)=1-1=0
BF(d) = 0-1=-1
BF(f)=0-0=0
BF(P)=0-0=0
BF(q)=0-0=0
BF(e)=0-0=0

* Calculate the balancing factor of nodes in the following binary tree:



BF(x)=4-2=2
BF(y)=1-3=-2
BF(z)=1-0=1
BF(u)=0
BF(v)=2-0=2
BF(w)=0
BF(m)=1-1=0
BF(a)=0
BF(b)=0

* Calculate the balancing factor of nodes in the following binary tree:

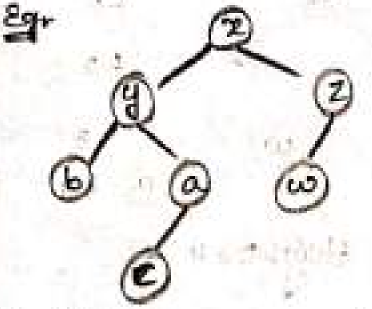


BF(40)= 2-4=-2
BF(50)= 1-0=1
BF(20)= 3-0 = 3
BF(30)= 0
BF(80)= 1-2=-1
BF(90)= 0
BF(95)= 1-0 =1
BF(25)= 0

* Balanced Binary ~~Search~~ tree:

- A binary tree is called balanced binary tree if the balancing factor of every node is -1 or 0 or +1

Eg:



BF(x) = 3-2 = 1 BF(a) = 1-0 = 1

BF(y) = 1-2 = -1 BF(b) = 0

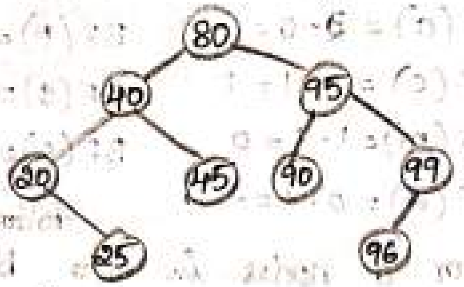
BF(z) = 1-0 = 1 BF(c) = 0

BF(w) = 0

* AVL Tree or balanced binary Search tree:

- A binary Search tree is called balanced binary Search tree or AVL tree if the balancing factor of every node is -1 or 0 or+

Eg:



BF(80) = 3-3 = 0 BF(90) = 0

BF(40) = 2-1 = 1 BF(99) = 1-0 = 1

BF(95) = 1-2 = -1 BF(96) = 0

BF(20) = 0-1 = -1

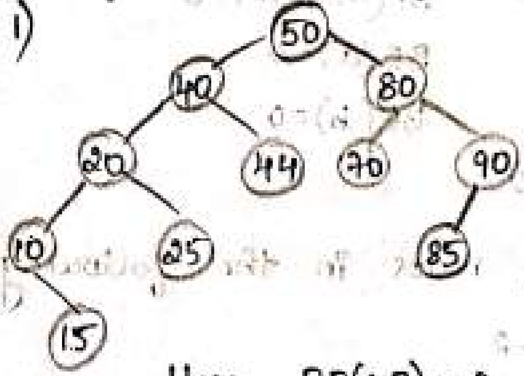
BF(25) = 0-0 = 0

BF(45) = 0-0 = 0

$\boxed{\text{Balanced}}$

* Verify the following BST is balanced or not.

1)



BF(50) = 4-3 = 1 BF(25) = 0

BF(40) = 3-1 = 2 BF(15) = 0

BF(80) = 1-2 = -1 BF(70) = 0

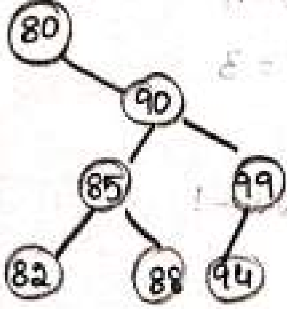
BF(20) = 2-1 = 1 BF(90) = 1-0 = 1

BF(44) = 0 BF(85) = 0

BF(10) = 0-1 = -1

Here BF(40) = 2, so the given BST is not balanc

2)



BF(80) = 0-3 = -3 BF(99) = 1-0 = 1

BF(90) = 2-2 = 0 BF(94) = 0

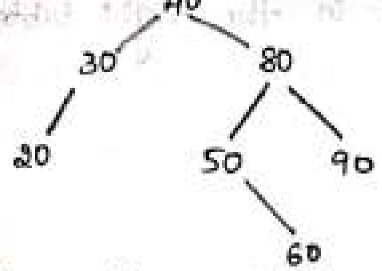
BF(85) = 1-1 = 0

BF(82) = 0

BF(88) = 0

Here BF(80) = -3, so the given BST is not balanced.

3)



BF(40) = 2 - 3 = -1
BF(30) = 1 - 0 = 1
BF(20) = 0
BF(80) = 2 - 1 = 1
BF(50) = 0 - 1 = -1
BF(90) = 0
BF(60) = 0

Here, the balancing factor (BF) of every node is -1, 0, +1, so the given BST is balanced and hence it is an AVL Tree.

16-07-2025 Wednesday *Operations in a ~~Vo~~ AVL Tree:-

1. Search operation - same as binary search tree
2. Insertion
3. Deletion

*Insertion operation in AVL Tree:

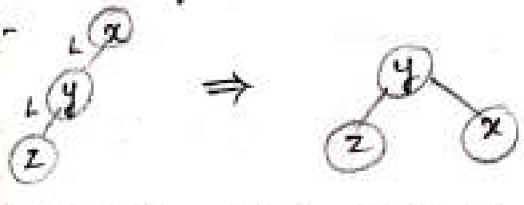
1. Follow the same ~~process~~ insertion process of BST
2. Recalculate the balancing factors of every node.
3. Verify whether there is any critical node or not.
4. Apply the suitable rotation.

*There are mainly 4 types of rotation:

1. LL rotation
2. LR rotation
3. RR rotation
4. RL rotation

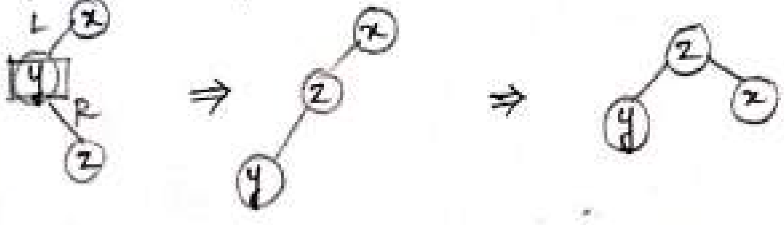
*LL Rotation: New node is inserted in the left sub tree of left child of critical node.

Eg:-



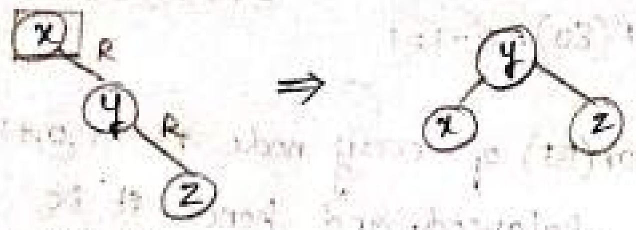
*LR Rotation: New node is inserted in the right subtree of left child of critical node.

Eg:-



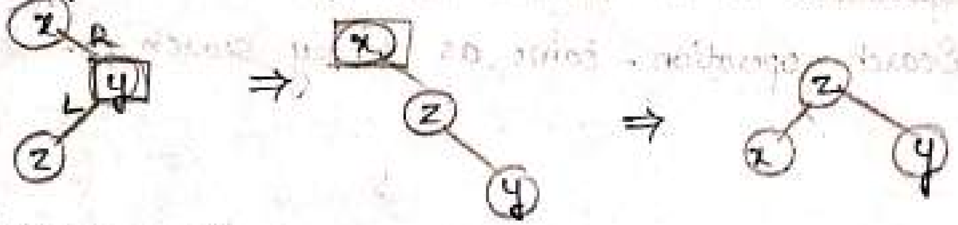
* **RR Rotation**: New node is inserted in the right subtree of right child of critical node.

Eg:



* **RL Rotation**: New node is inserted in the left subtree of right child of critical node.

Eg:



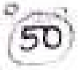
* **Critical node**: The lowest ancestor of newly inserted node for which the balancing factor is none of −1, 0, +1

empty space

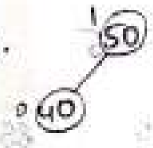
*Example for LL Rotation:

Input : 50, 40, 10, 8, 5

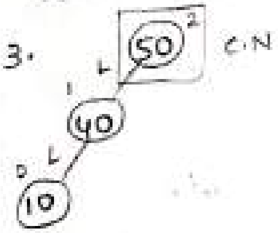
1.



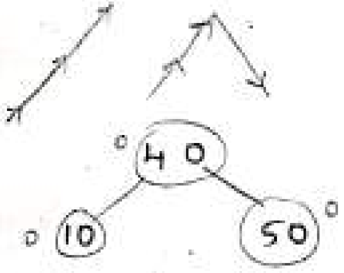
2.



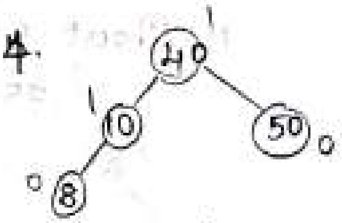
3.



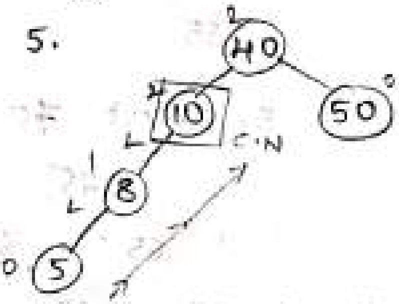
Apply LL Rotation



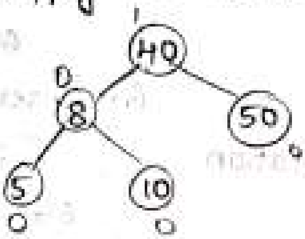
4.



5.



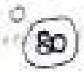
Apply LL Rotation



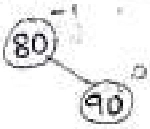
*Example for RR rotation:

Input : 80, 90, 100, 110, 120

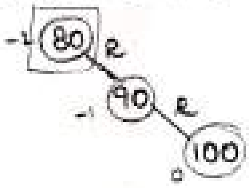
1. Insert 80



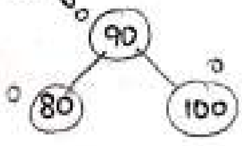
2. Insert 90



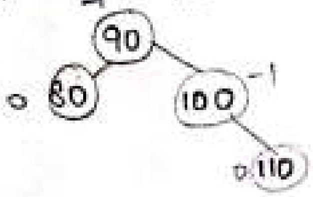
3. Insert 100



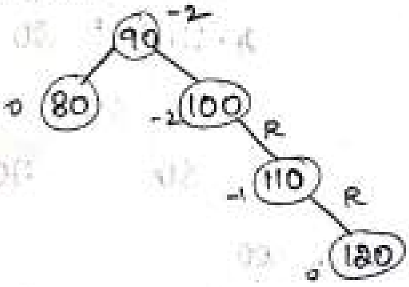
Apply RR rotation



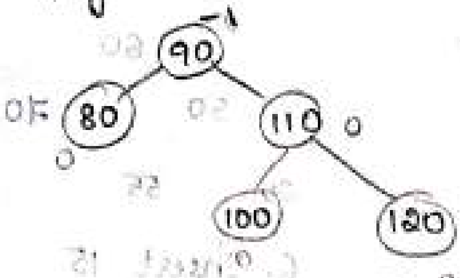
4. Insert 110



5. Insert 120



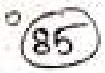
Apply RR rotation



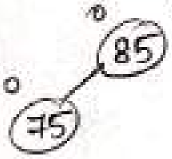
* Construct AVL Tree for following numbers:

① 85, 75, 65, 55, 70, 50

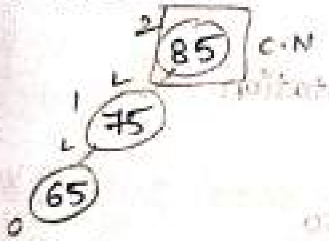
1. Insert 85



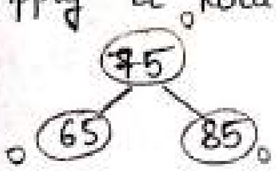
2. Insert 75



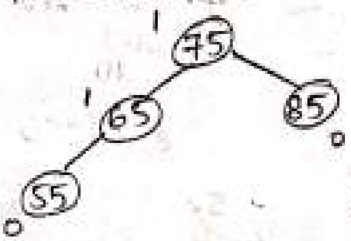
3. Insert 65



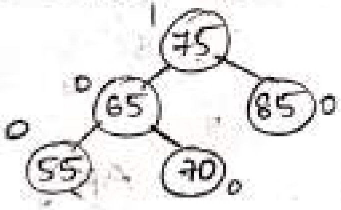
Apply LL Rotation



4. Insert 55

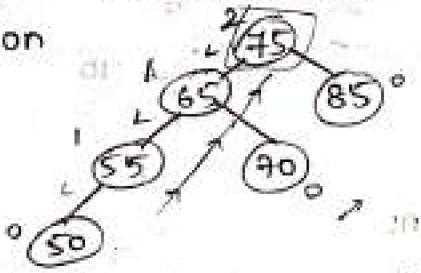


5. Insert 70

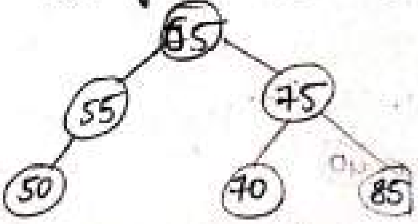


~~60~~

6. Insert 50

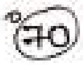


Apply LL Rotation

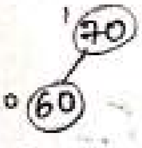


② 70, 60, 50, 20, 55, 15

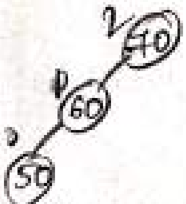
1. Insert 70



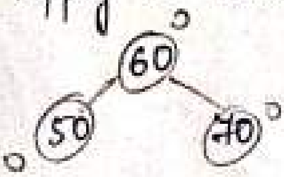
2. Insert 60



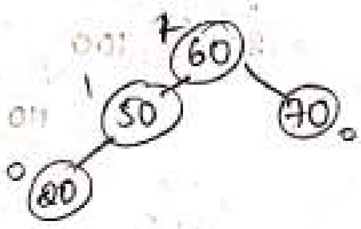
3. Insert 50



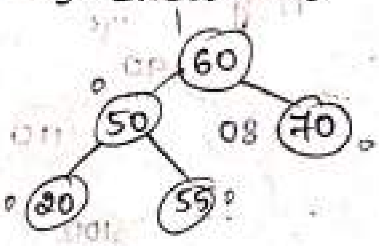
Apply LL Rotation



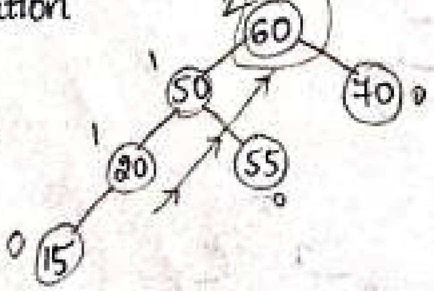
4. Insert 20



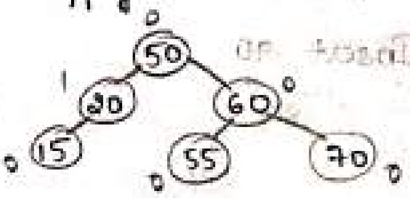
5. Insert 55



6. Insert 15

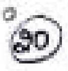


Apply LL Rotation

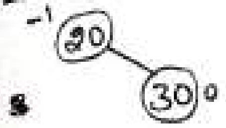


③ 20, 30, 80, 90, 70, 95

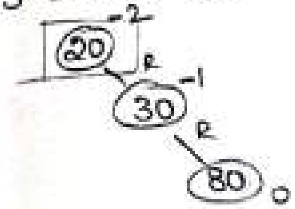
1. Insert 20



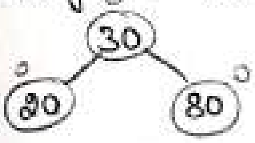
2. Insert 30



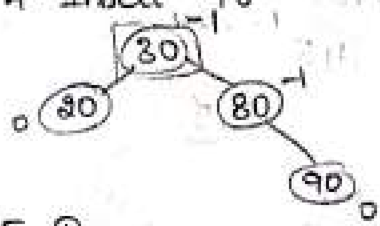
3. Insert 80



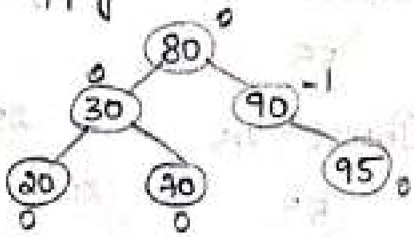
Apply RR rotation



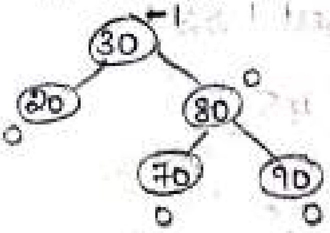
4. Insert 90



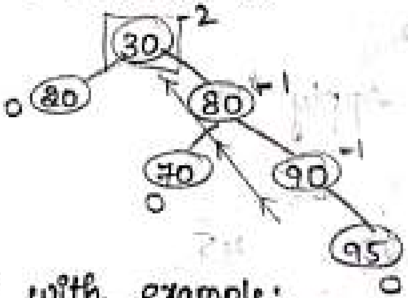
Apply RR rotation



5. Insert 70



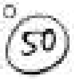
6. Insert 95



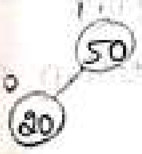
*Explain LR rotation with example:

1) Data : 50 20 40

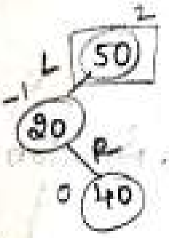
1. Insert 50



2. Insert 20

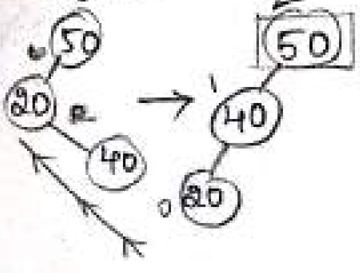


3. Insert 40

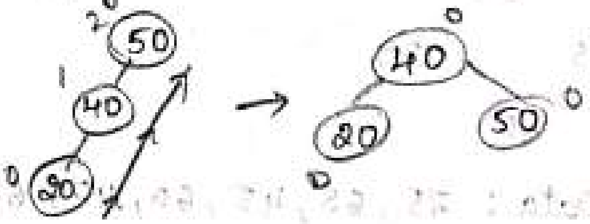


Apply LR rotation

→Apply RR Rotation

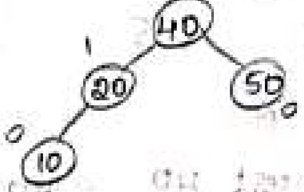


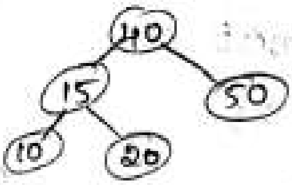
→ Apply ~~RR~~ rotation



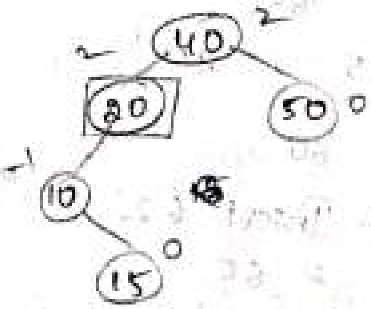
~~2) Data : 50 20 40 10 15~~

4. Insert 10 ~~5. In~~

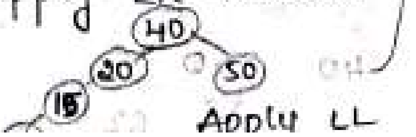




5. Insert 15



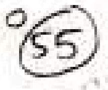
Apply LR rotation:



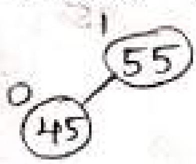
Apply LL

2) Data : 55 45 25 40 20 22

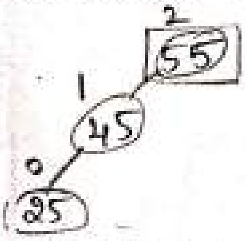
1. Insert 55



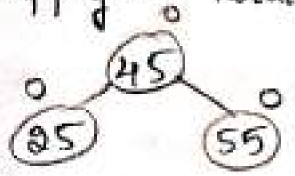
2. Insert 45



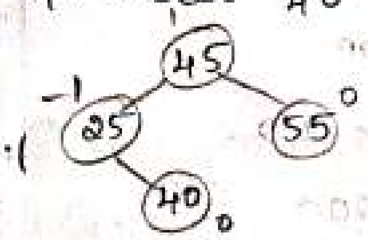
3. Insert 25



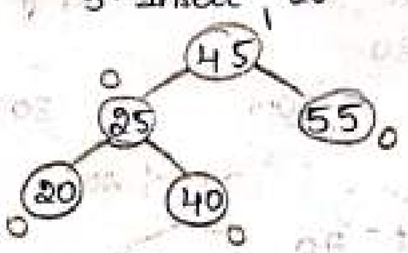
Apply LL Rotation



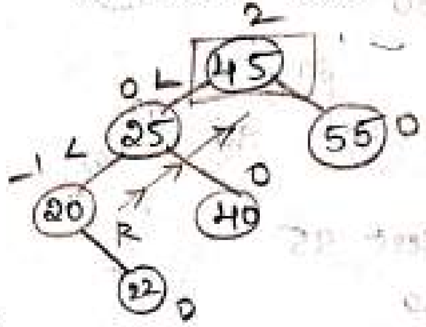
4. Insert 40



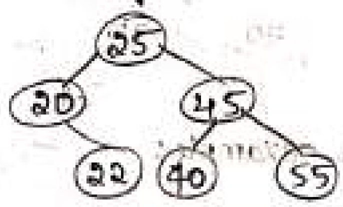
5. Insert 20



6. Insert 22

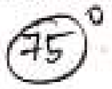


Apply LL

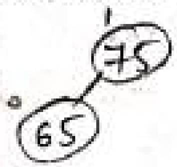


3) Data : 75, 65, 45, 60, 40, 62

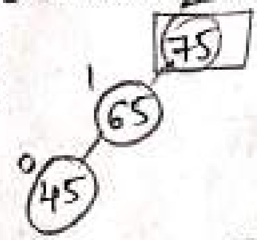
1. Insert 75



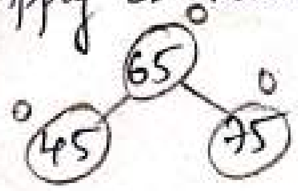
2. Insert 65



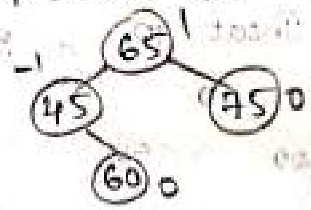
3. Insert 45



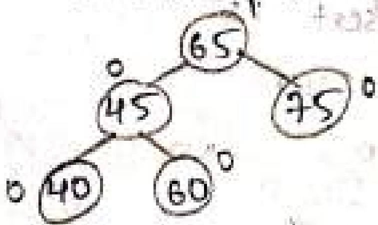
Apply LL Rotation



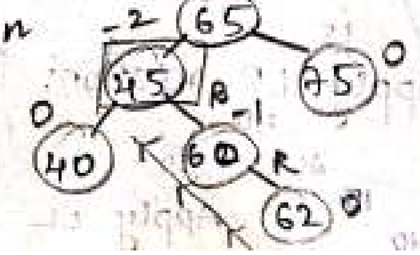
4. Insert 60



5. Insert 40

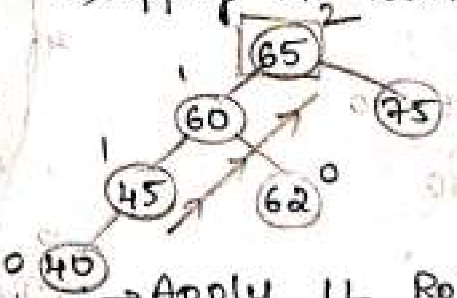


6. Insert 62

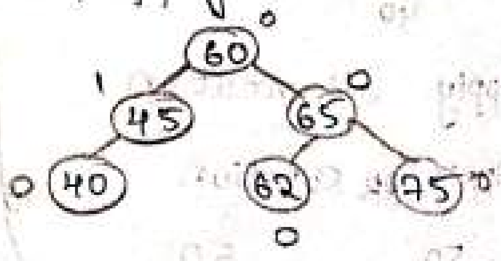


Apply LR rotation:

→Apply RR rotation



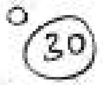
→Apply LL Rotation



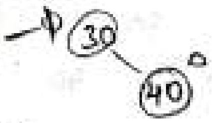
*Explain RL rotation with example:

1) Data: 30 40 50 70 65

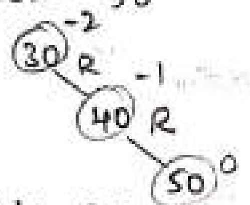
1. Insert 30



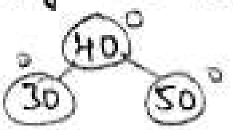
2. Insert 40



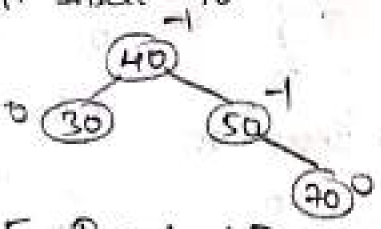
3. Insert 50



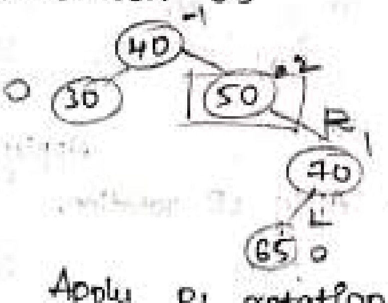
Apply RR rotation



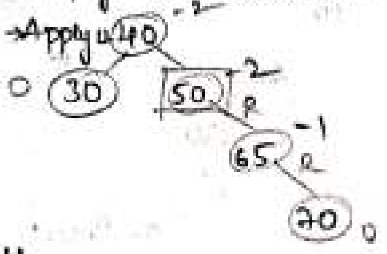
4. Insert 70



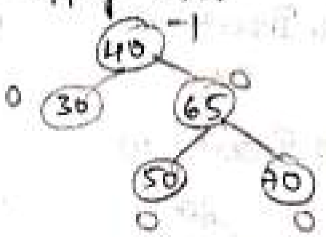
5. Insert 65



Apply RL rotation



Apply RR:



2) 11, 22, 15, 66, 33, 44

~~1. Insert 11~~

~~11~~

~~2. Insert 22~~

~~11 22~~

~~3. Insert 15~~

~~11 15 22~~

~~4 Insert 66~~ 15

~~11 15 22 66~~

5. Insert 33

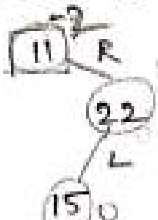
1. Insert 11

11

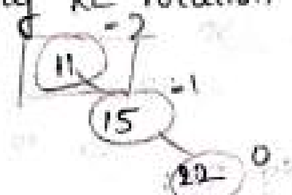
2. Insert 22

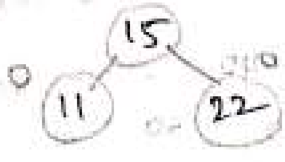
11 22

3. Insert 15

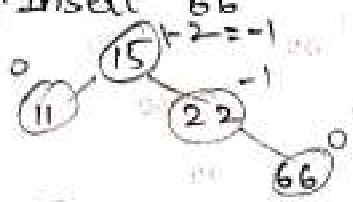


Apply RL rotation

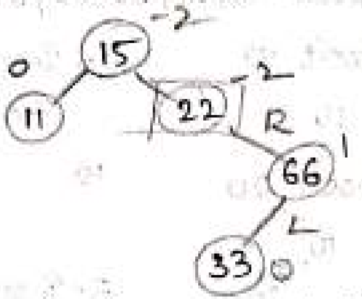




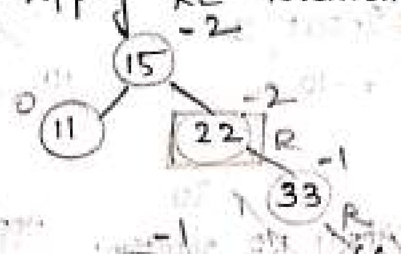
4. Insert 66

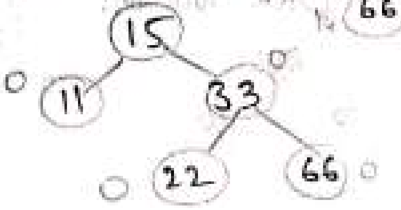


5. Insert 33



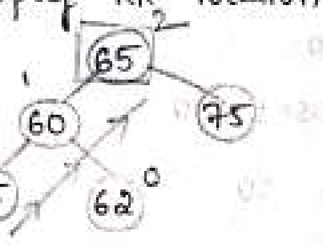
Apply RL rotation





ly LR rotation:

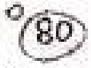
pply RR rotation



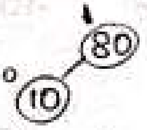
Apply LL Rotation

3) Data : 80, 10, 70, 30, 20, 60, 40, 100, 90

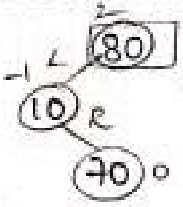
1. Insert 80



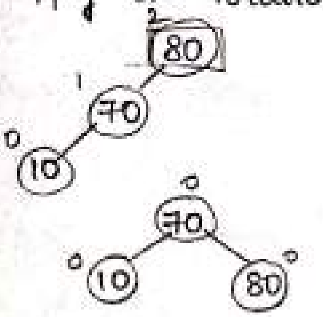
2. Insert 10



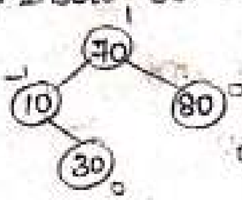
3. Insert 70

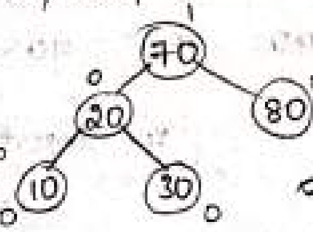


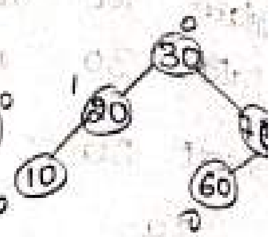
Apply LR rotation:



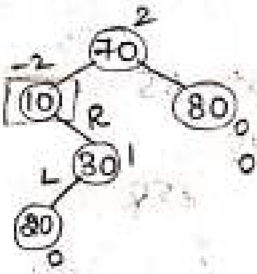
4. Insert 30



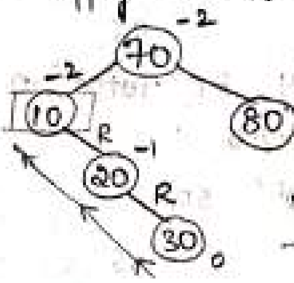




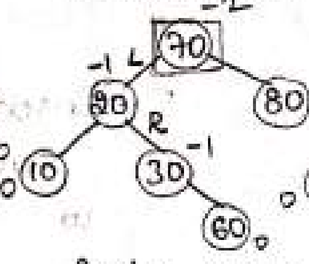
5. Insert 20



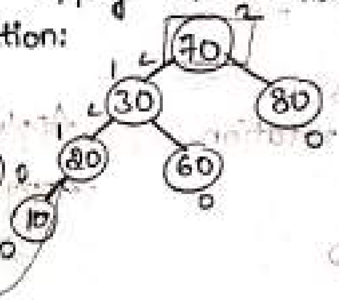
Apply LR rotation:



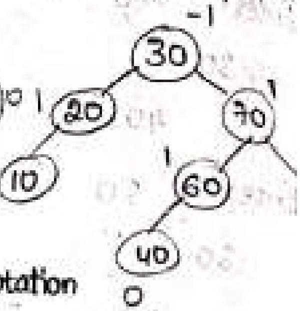
6. Insert 60



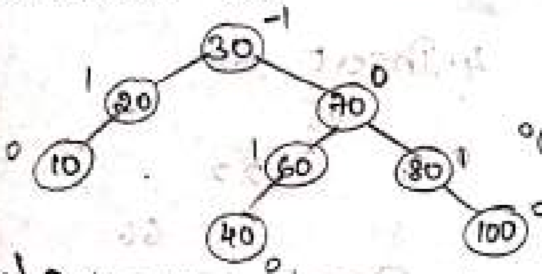
Apply LR rotation



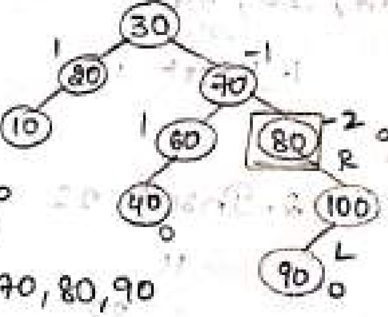
7. Insert 40



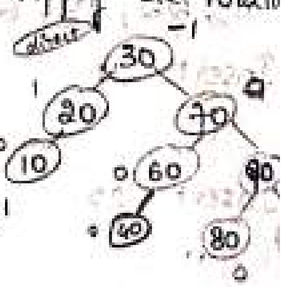
8. Insert 100



9. Insert 90

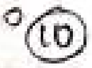


Apply LR rotation

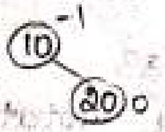


4) Data: 10, 20, 30, 40, 50, 60, 70, 80, 90

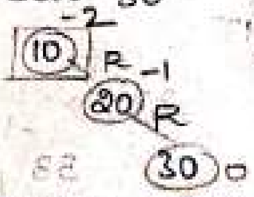
1. Insert 10



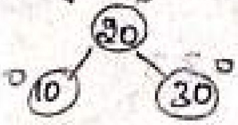
2. Insert 20



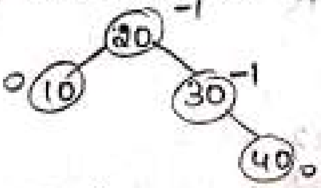
3. Insert 30



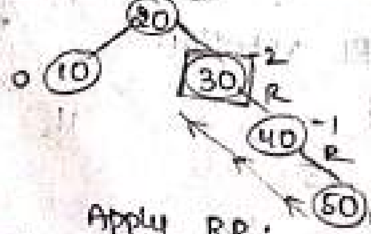
Apply RR rotation:



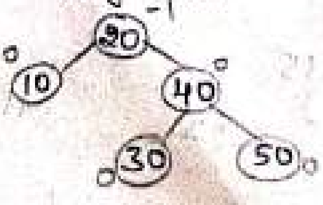
4. Insert 40



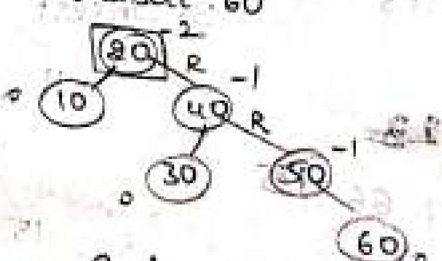
5. Insert 50



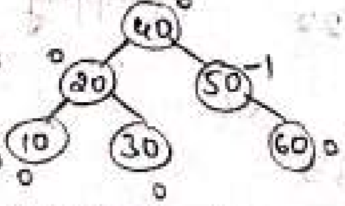
Apply RR:



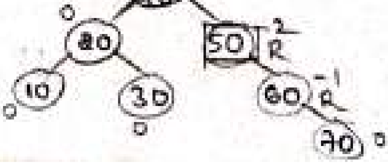
6. Insert 60



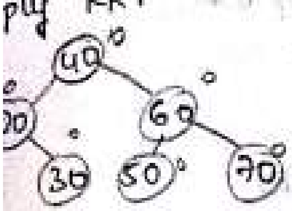
Apply RR:



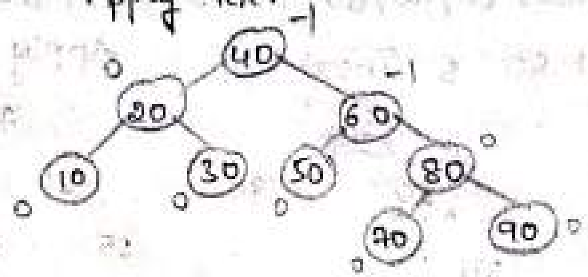
7. Insert 70



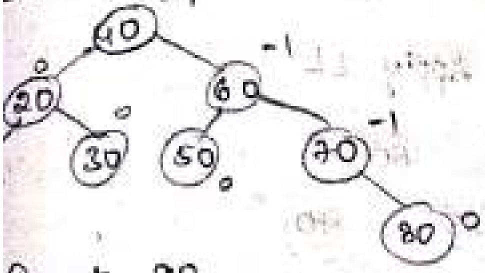
ply RR



Apply RR

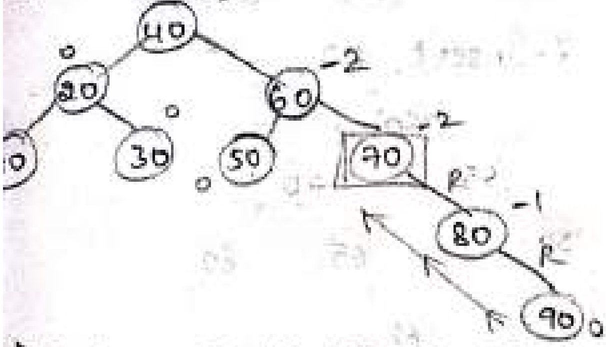


Insert 80



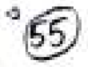
~~10. Insert~~

Insert 90

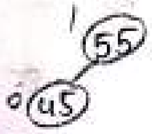


5) Data : 55, 45, 50, 80, 90, 85, 60

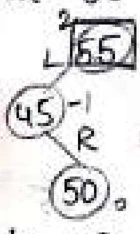
1. Insert 55



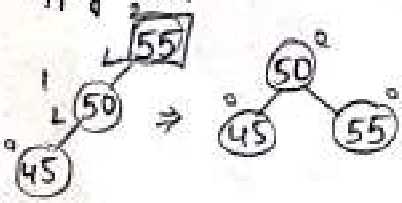
2. Insert 45



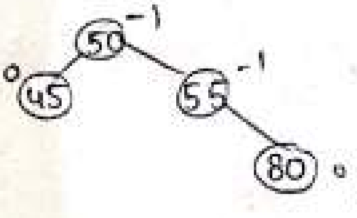
3. Insert 50



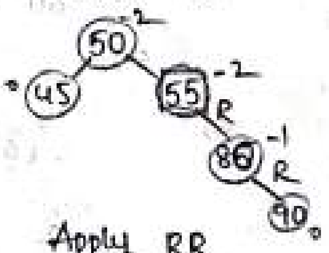
Apply LR:



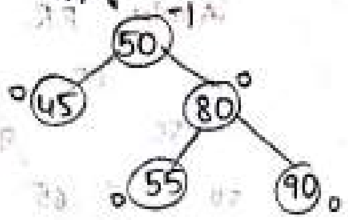
4. Insert 80



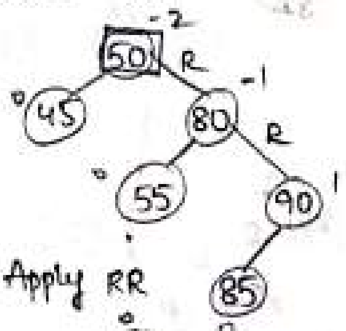
5. Insert 90



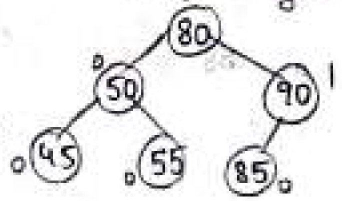
Apply RR



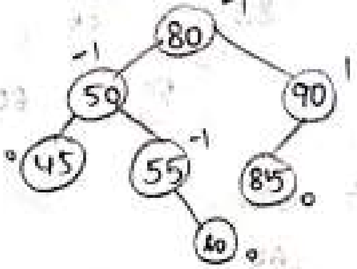
6. Insert 85



Apply RR



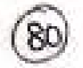
7. Insert 60



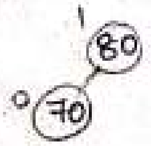
21-07-2025 * Data: 80, 70, 60, 50, 55, 65, 62, 85, 82

Monday

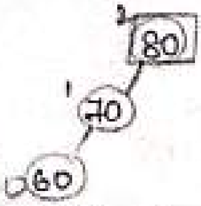
1. Insert 80



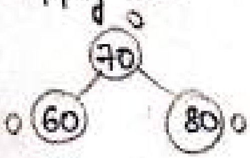
2. Insert 70



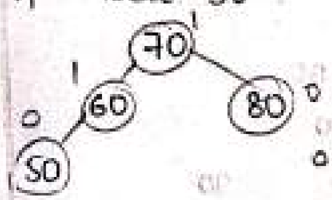
3. Insert 60



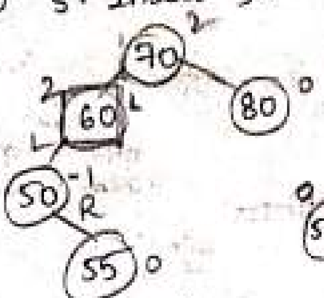
Apply LL



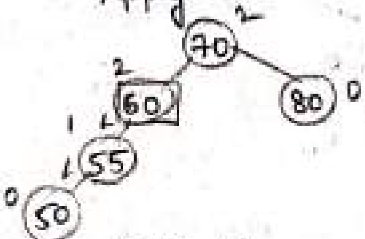
4. Insert 50



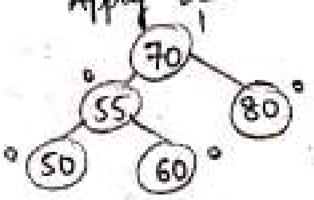
5. Insert 55



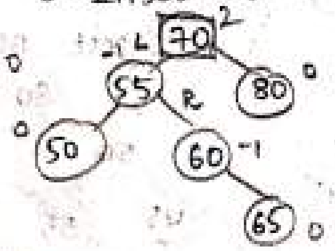
Apply LR



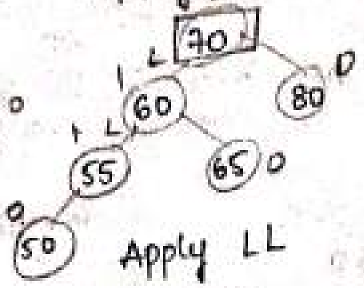
Apply LL



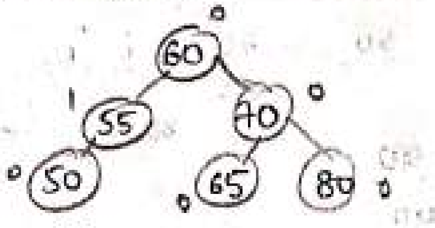
6. Insert 65



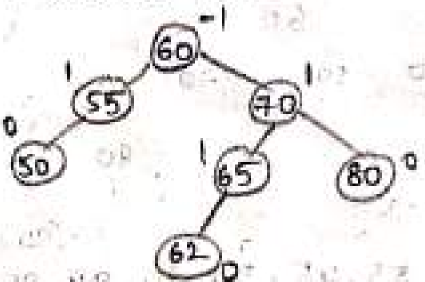
Apply LR



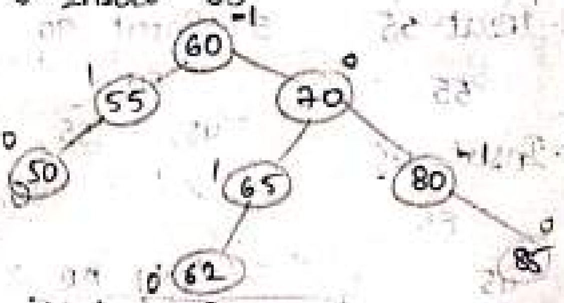
Apply LL



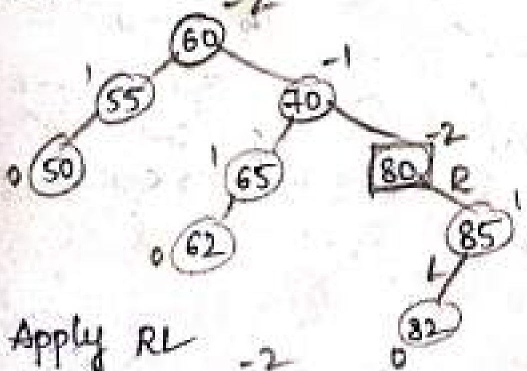
7. Insert 62



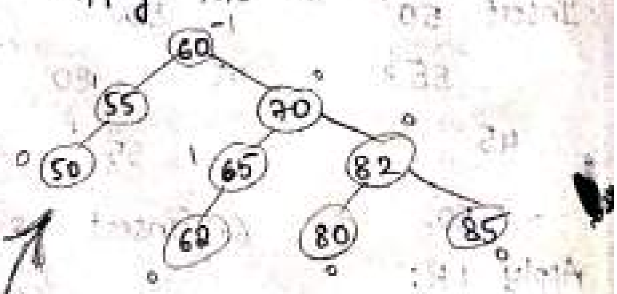
8. Insert 85



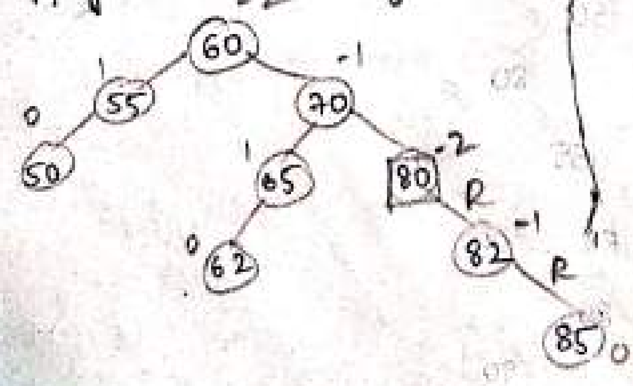
9. Insert 82



Apply RR

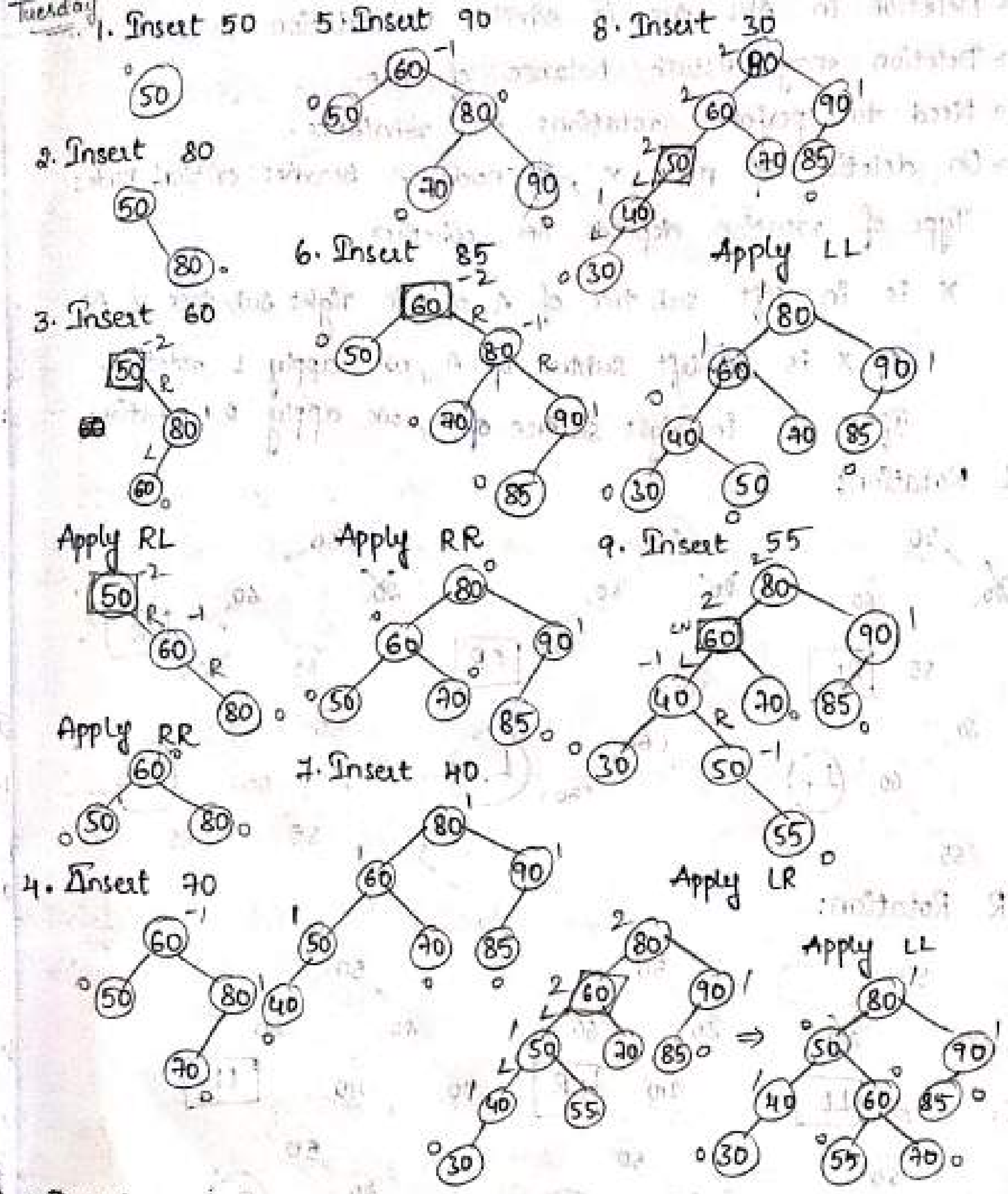


Apply RL



22-07-2025
Tuesday

Data: 50, 80, 60, 70, 90, 85, 40, 30, 55



* Deletion:

There are two types of rotations for deletion:

1) L Rotations (L_0, L_{-1}, L_1)

2)

2) R Rotations (R_0, R_{-1}, R_1)

* Deletion:-

→ Deletion in AVL tree is similar to deletion in BST.

→ Deletion may distuib balance of tree.

→ Need to perform rotations to rebalance.

→ On deletion of node x, if node A becomes critical node:

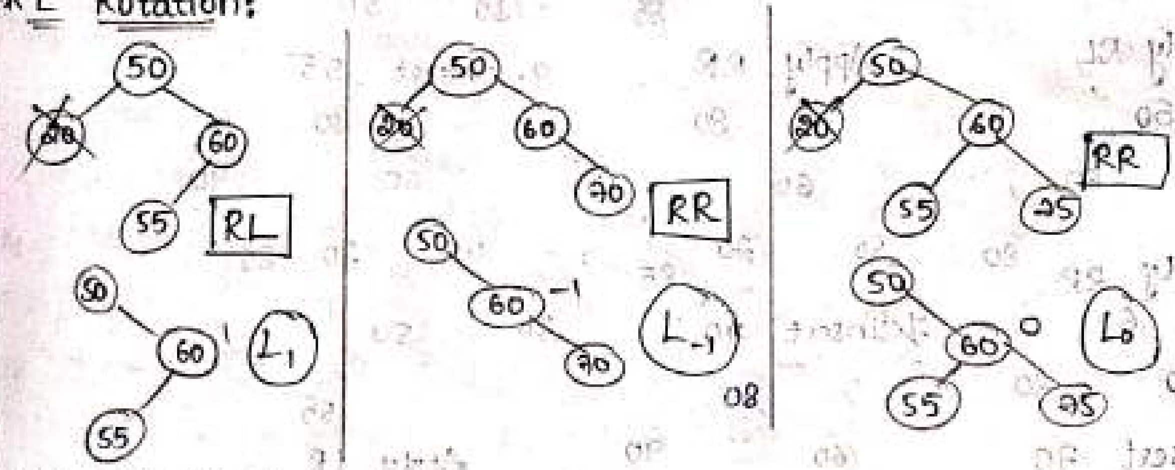
Type of rotation depends on whether

x is in left sub-tree of A or in right sub-tree of A.

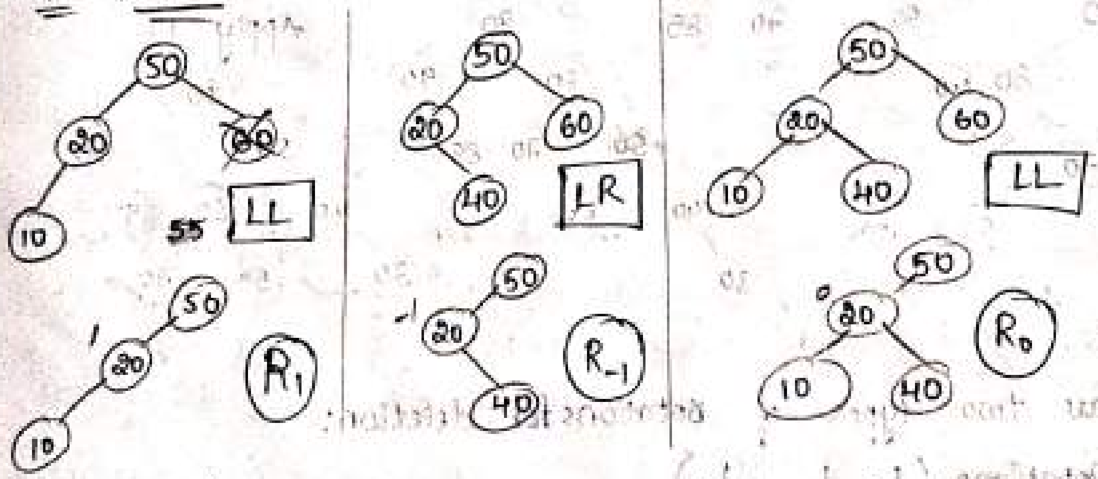
If x is in left subtree of A, we apply L rotation.

If x is in right subtree of A, we apply R rotation.

* L Rotation:



* R Rotation:



* L Rotations:-

L_0 rotation:

BF(critical node) = -2 and BF(critical node → rchild) = 0

Similar to RR rotation

L_1 rotation:

BF(critical node) = -2 and BF(critical node → rchild) = 1

Similar to RL rotation

L_2 rotation:

BF(critical node) = -2 and BF(critical node → rchild) = -1

Similar to RR rotation

*R Rotations:

R_0 rotation:

BF(node) = 2 and BF(node->lchild)=0

Similar to LL rotation.

R_1 rotation:

BF(node) = 2 and BF(node->lchild) = 1

Similar to LL rotation

R_{-1} rotation:

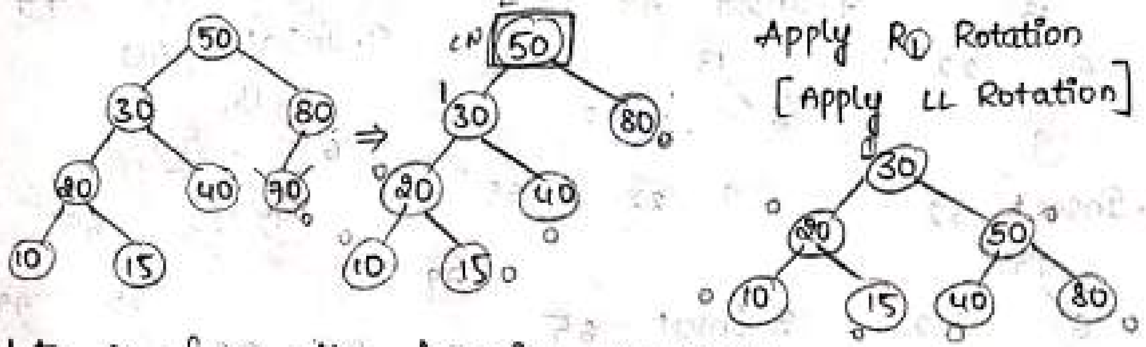
BF(node) = 2 and BF(node->lchild) = -1

Similar to LR rotation.

23-07-2025
Wednesday

*Delete 70 from the following AVL Tree:

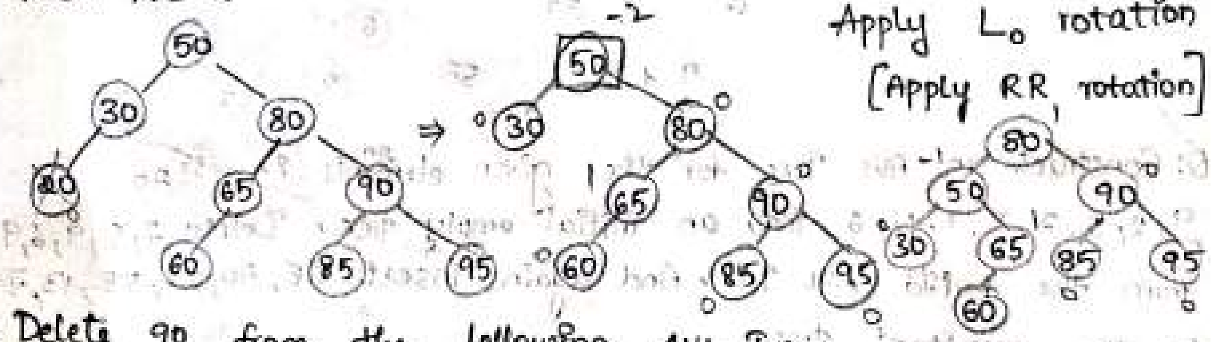
Given tree is



Apply R_1 Rotation
[Apply LL Rotation]

*Delete 20 from the following AVL Tree:

Given tree is

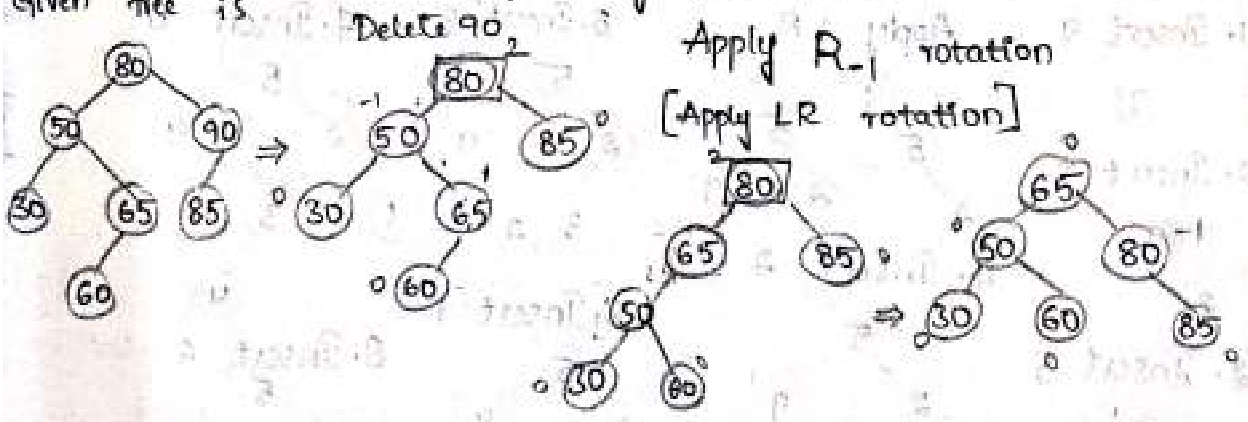


Apply L_0 rotation
[Apply RR_1 rotation]

*Delete 90 from the following AVL Tree:

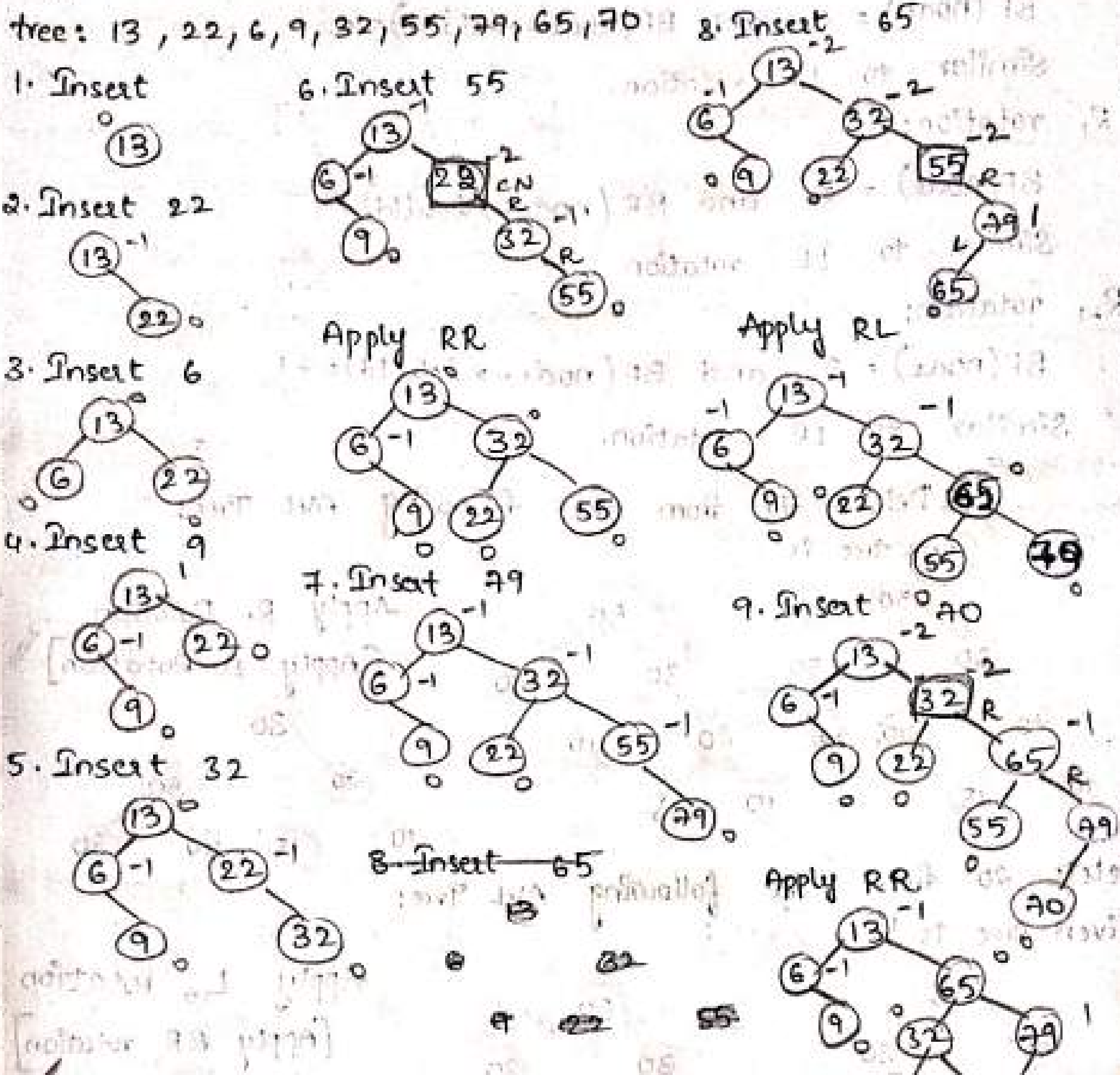
Given tree is

Delete 90,

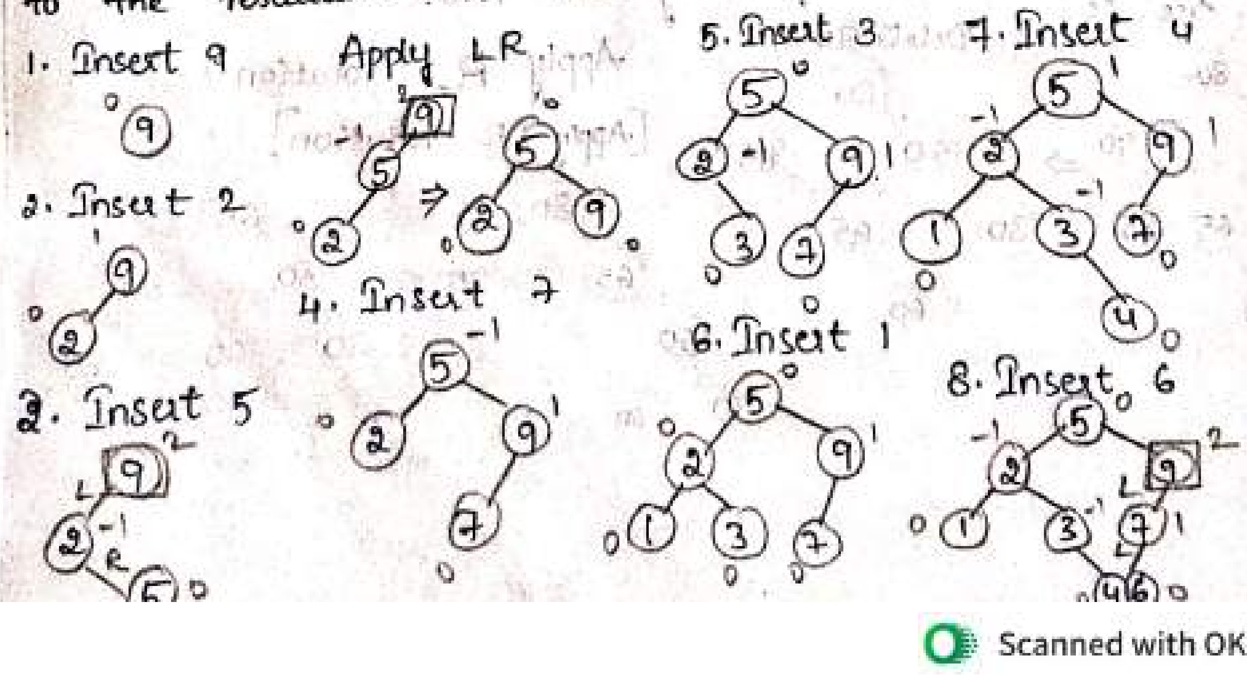


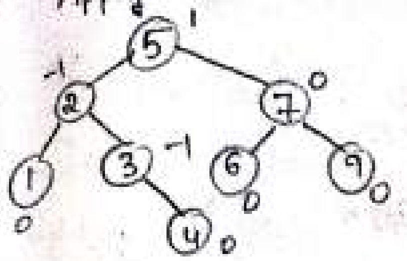
Apply R_{-1} rotation
[Apply LR rotation]

Q: Construct an AVL Tree using the following entries as a sequence set. Show the balance factors in the resulting tree: 13, 22, 6, 9, 32, 55, 79, 65, 70

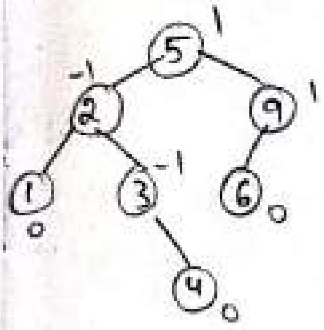


Q: Construct an AVL Tree for the given elements 9, 2, 5, 7, 3, 1, 4, 6 into an initial empty tree. Delete 7, 5, 9, 6, 9 from the bulild AVL Tree. And again insert 65, 70, 22, 55, 13, 79 to the resultant tree.

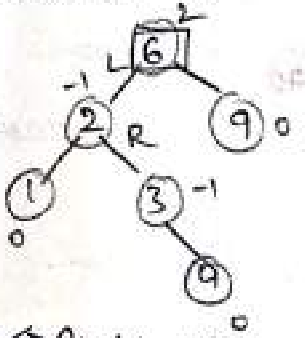




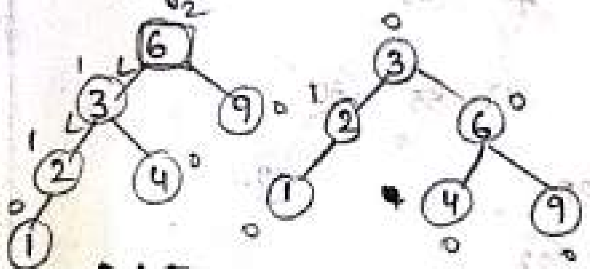
→Delete 7



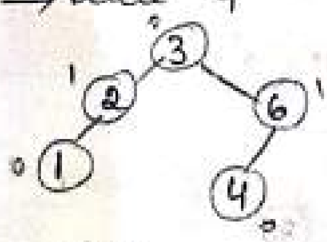
→Delete 5



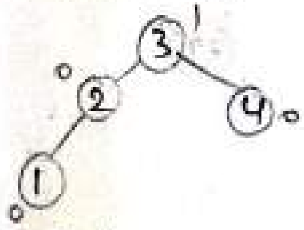
→ Apply LR



→ Delete 9



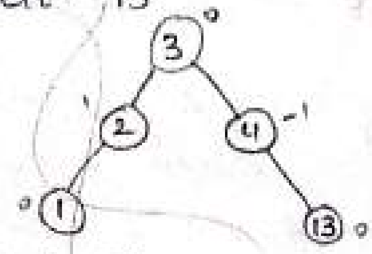
→ Delete 6



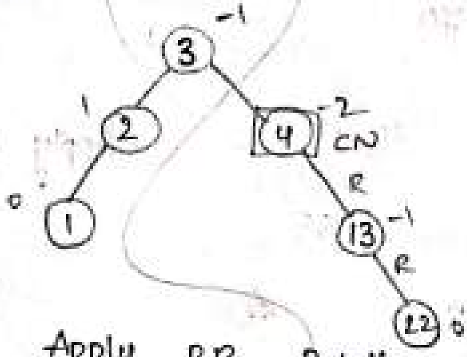
→ Delete 9

9 already deleted.

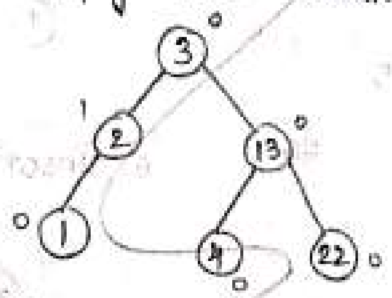
1) Insert 13



2) Insert 22

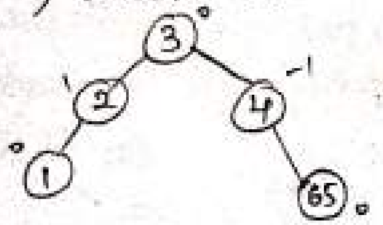


Apply RR Rotation

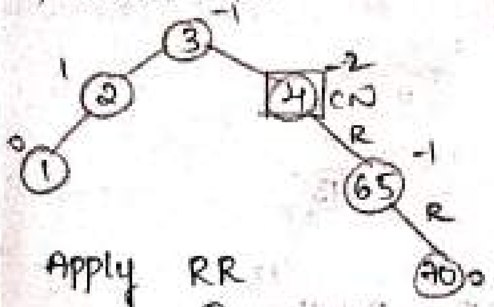


3) Insert 6

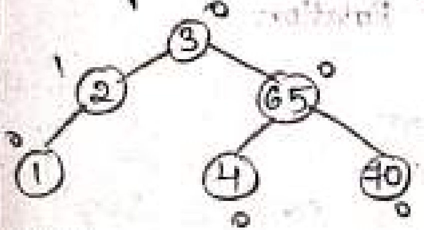
1) Insert 65



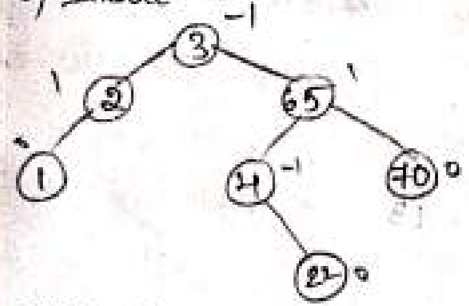
2) Insert 70



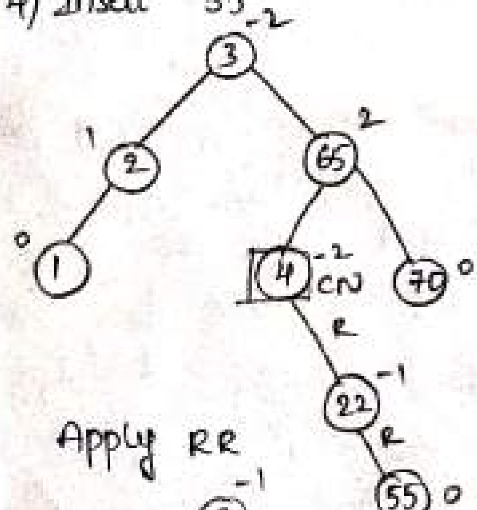
Apply RR



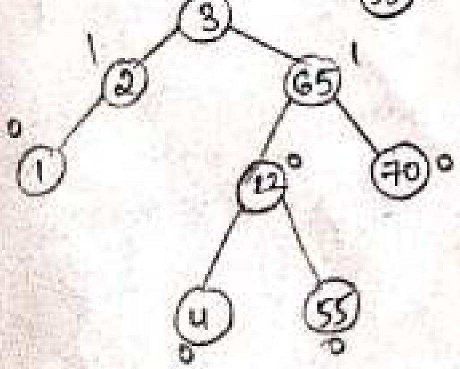
3) Insert 22



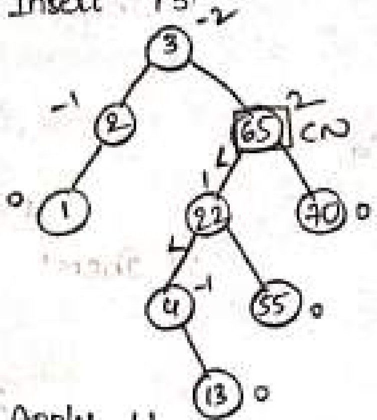
4) Insert 55



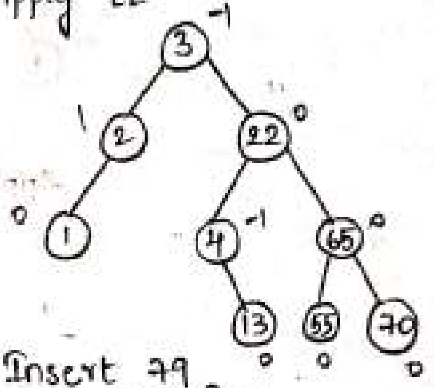
Apply RR



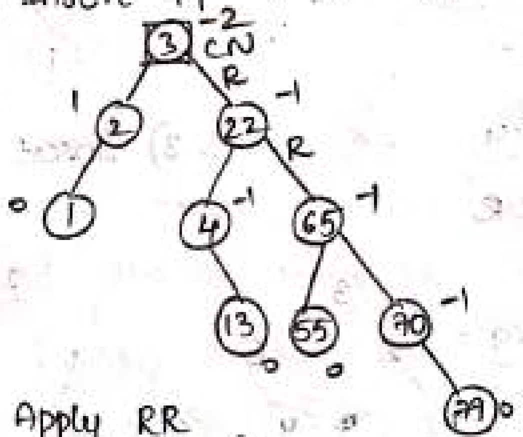
5) Insert 13



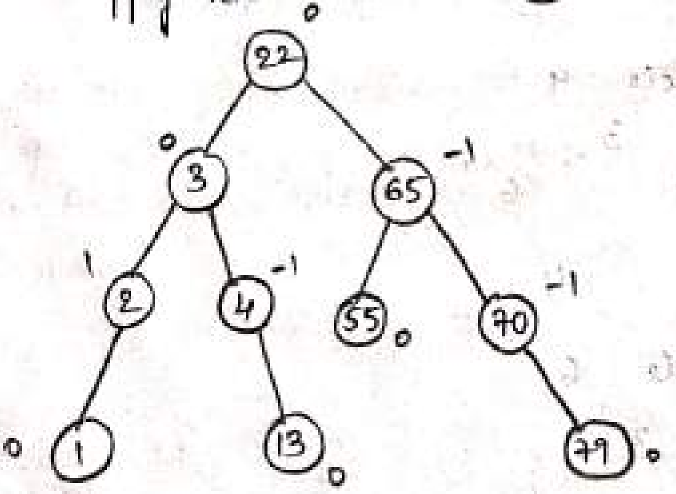
Apply LL



6) Insert 79

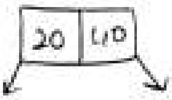
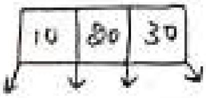


Apply RR



24-07-2025
Thursday

* B-Trees:-

• Binary tree → max 2 children → 1 data value 	Ternary tree → max 3 children → max 2 data values 	4-ary tree → max 4 children → max 3 data values 	m-ary tree → max m children → max (m-1 data valu
• Binary Search tree → max 2 children → 1 data value	Ternary Search tree → max 3 children → max 2 data values	4-way Search tree → max 4 children → max 3 data values	m-way Search tree
• Balanced Binary Search tree	B-Tree of order 3 2-3 Tree	B-Tree of order 4	B-Tree of order m

Q: write about m-way Search trees.

*what is B-Tree?

- B-Tree is a balanced m-way search tree.
- In a B-Tree of order m, every node has maximum m number of children and (m-1) $\overset{\text{data values}}{\text{keys}}$.

*Operations on B-Tree:-

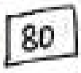
1. Search
2. Insertion
3. Deletion
4. Traverse

*Sample node structure for a B-Tree of order 5:-

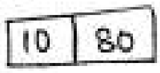
original view				our view			
10				10			
10	80			10	80		
5	10	80		5	10	80	
5	10	15	80	5	10	15	80

*Construct B-Tree of order 5 for the following keys:-
80, 10, 60, 50, 70, 120, 100, 20, 140, 30, 40, 85, 45, 48, 90, 75, 55,

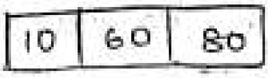
1) Insert 80



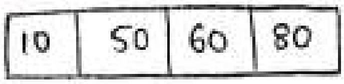
2) Insert 10



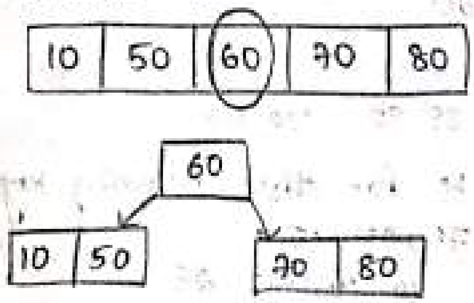
3) Insert 60



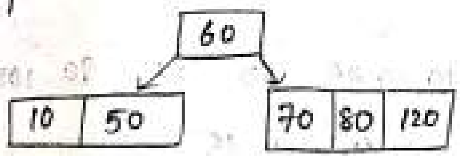
4) Insert 50



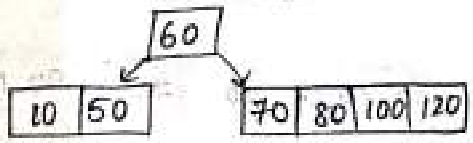
5) Insert 70



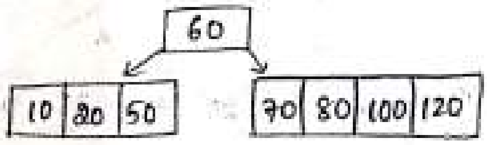
6) Insert 120



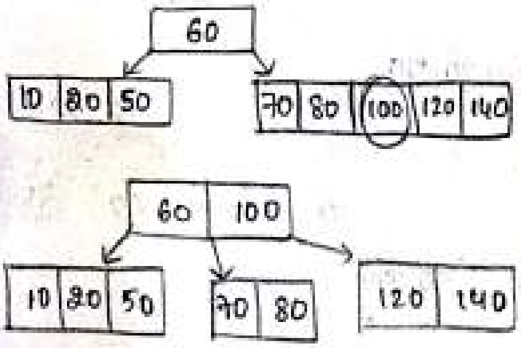
7) Insert 100



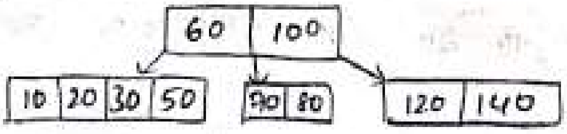
8) Insert 20



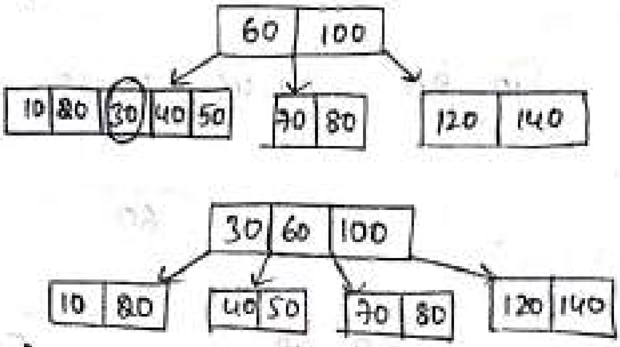
9) Insert 140



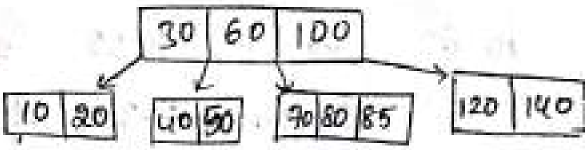
10) Insert 30



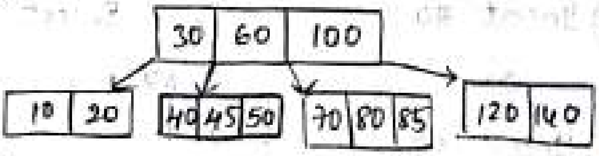
11) Insert 40



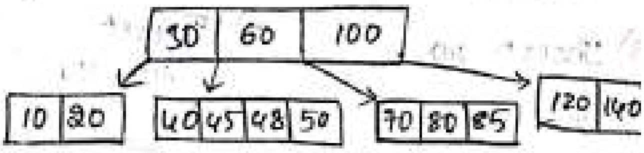
12) Insert 85



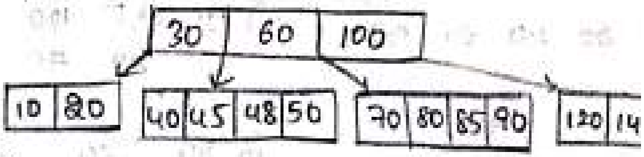
13) Insert 45



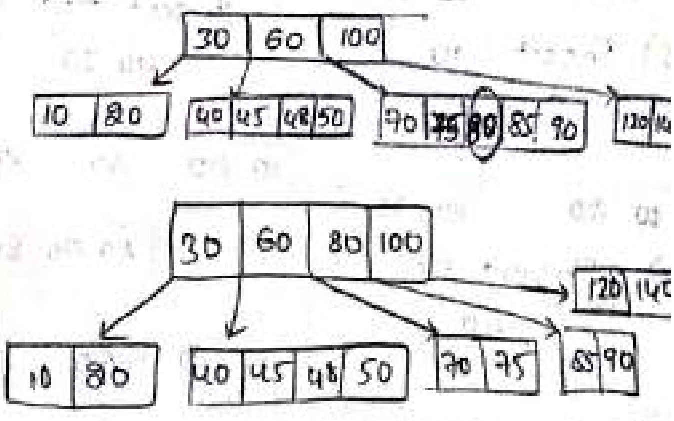
14) Insert 48



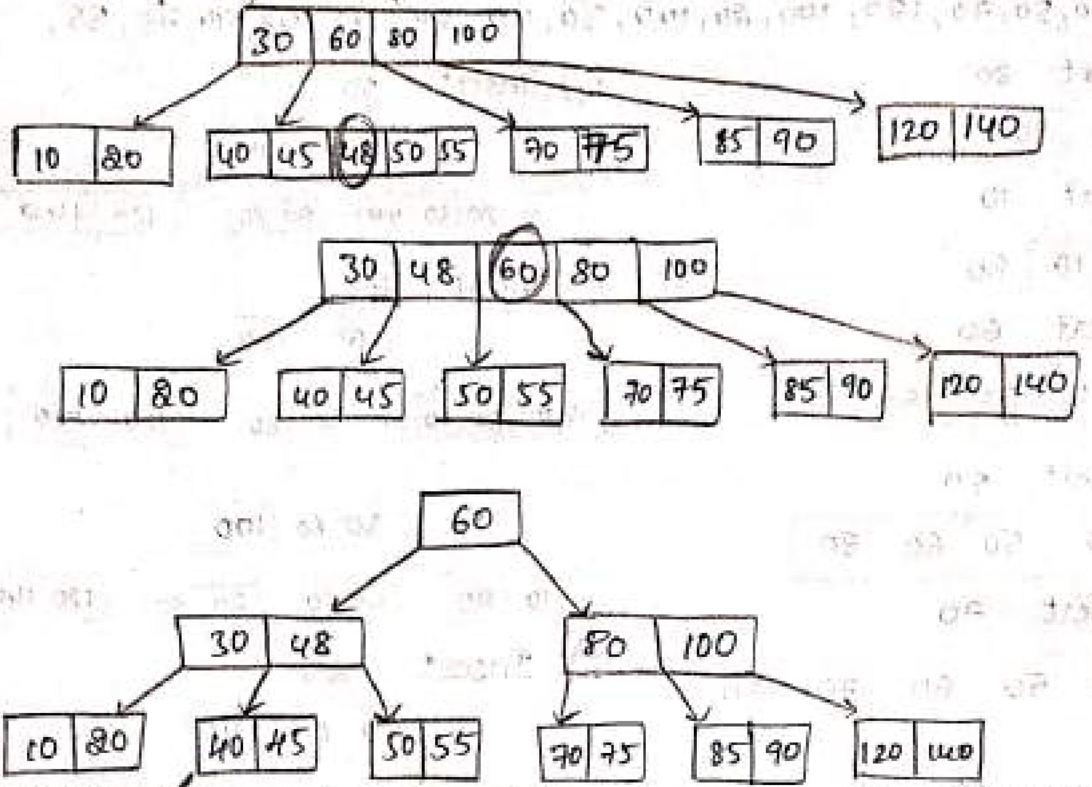
15) Insert 90



16) Insert 75



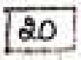
17) Insert 55



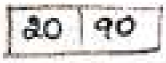
28-02-2025 Monday

* Construct B-Tree of order 4 for the following keys 20, 90, 40, 80, 10, 70, 60, 100, 75, 25, 15

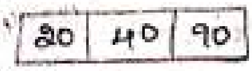
1) Insert 20



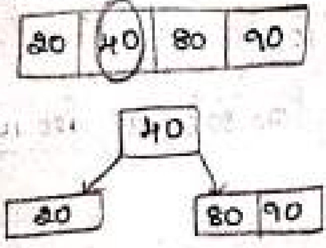
2) Insert 90



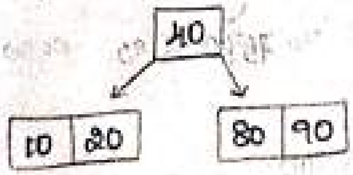
3) Insert 40



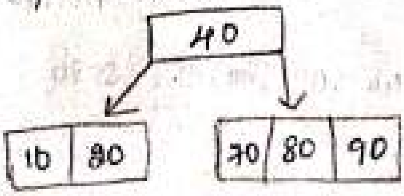
4) Insert 80



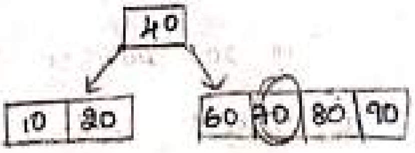
5) Insert 10



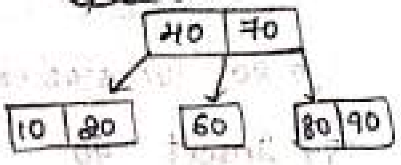
6) Insert 70



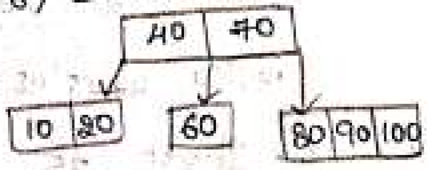
7) Insert 60



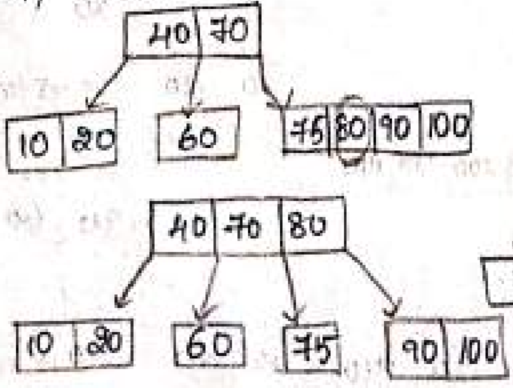
~~8) [illegible]~~



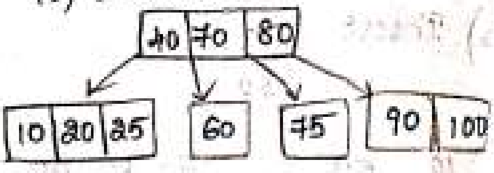
8) Insert 100



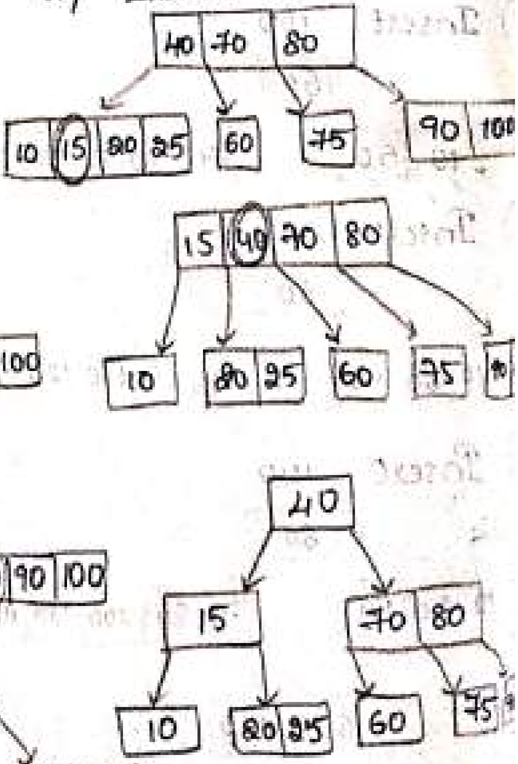
9) Insert 75



10) Insert 25



11) Insert 15



* **B-Tree:-**

- A B-Tree of order m is a m-way search tree with the following properties:

→ Every node in the B tree has at most m children

→ Every node in the B tree except the root node and leaf nodes has at least m/2 children.

→ The root node has at least two children if it is not a leaf node.

→ All leaf nodes are at the same level.

* **B-Tree: Insertion**

In a B-tree, all insertions are done at the leaf node level

Algorithm:-

Step 1: Do the search to determine which leaf node will hold a key.

Step 2: If leaf node have space, insert key in ascending order, otherwise split leaf node's keys into two nodes, and promote median key to the parent.

Step 3: If the parent node is full, recursively split and promote median key to its parent.

Step 4: If a promotion is made to a full root node, split and create a new root node holding only the promoted median key.

* **2-3 Tree:-**

- A B-tree of order 3 is called 2-3 Tree.
- In 2-3 tree, every node has either 2 children or 3 children

Note: Every B-tree of order-m has atleast $\underset{seal}{}\left\lceil \frac{m}{2} \right\rceil$ children

$\left\lceil \frac{m}{2} \right\rceil - 1$ Keys

$\frac{m}{2} = \frac{5}{2} = 2.5 \approx 3$

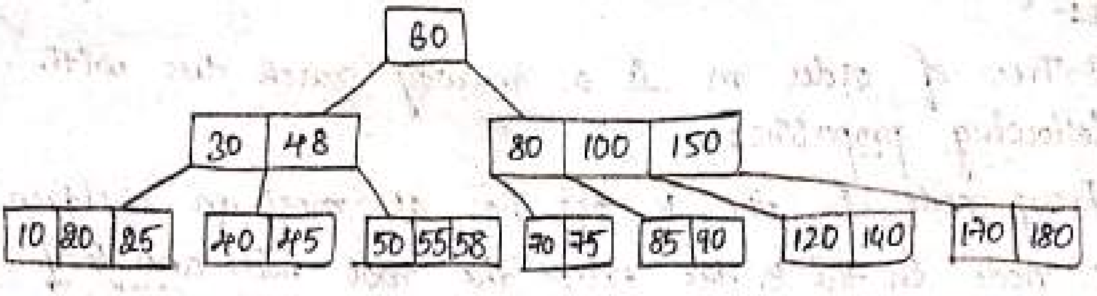
Here m = 5

$\frac{5}{2} - 1 = 3 - 1 = 2$

Every node must have 3 children and 2 keys.

* Explain the deletion process in B-Tree:

- Let us consider the following B-tree of order 5



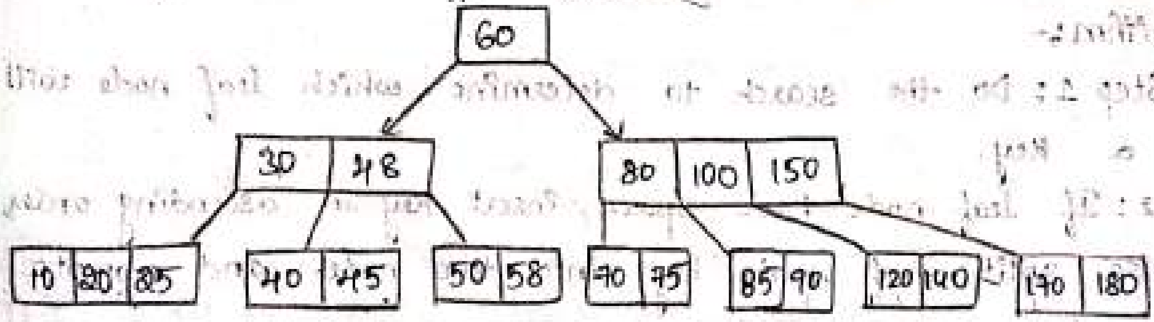
Case 1:

- The node is having more than minimum no. of trees
- let us delete 55:

→ The node 50|55|58 is having more than 2 Keys.

→ Simply delete 55 from the node.

→ After deletion of ~~55 the~~ tree looks like below:



Case 2:

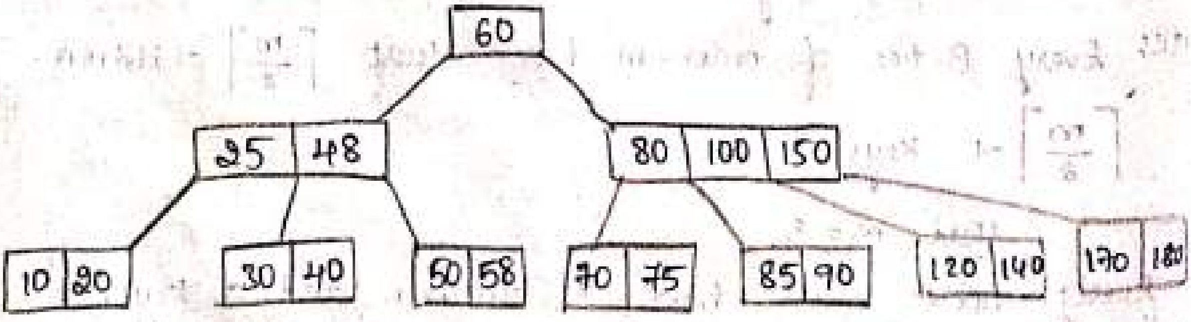
- The node is having exactly minimum no. of keys

Case 2.1: Get help ~~of~~ from siblings.

- let us delete 45

→ After deletion of 45:

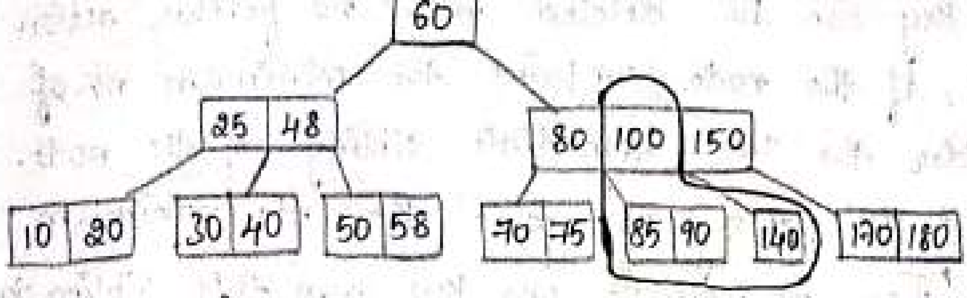
- The node 40 as less than 2 keys.
- Get the help from one of the siblings node 10|20|25



Case 2.2: The siblings of the deficient node has exactly minim no. of keys then get help from parent.

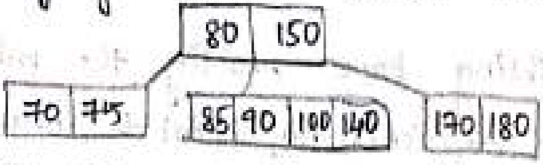
→ Let us delete 120 from the above tree:

After deletion of 120 the tree looks like



→ Get 100 from parent and merge with sibling | 85 | 90 |

→ After merging the tree looks like below:



Case 3: when case 2.2 leaves the parent node as deficient node we will apply same process for parent node.

Case 4: If the value to be deleted is present in internal node.

→ First replace the value with inorder successor and then delete successor from leaf node :-

Note :- → B-Tree of order m

1) Max no. of children = m

2) Max no. of Keys = m-1

3) Min no. of children = $\lceil \frac{m}{2} \rceil$

4) Min no. of Keys = $\lceil \frac{m}{2} \rceil - 1$

→ B-Tree of min degree t

1) max no. of children = 2t

2) max no of ~~children~~ keys = 2t-1

3) min no. of children = t

4) min no. of keys = t-1

Egr B-tree with min degree 2 = B-tree of order 4.

B-tree with min degree 3 = B-tree of order 6.

*B-Tree : Deletion

Algorithm:-

1) If the key to be deleted is not in a leaf, swap it with successor or predecessor under the natural order of the keys, then delete the key from the leaf.

2) If leaf node contains more than the minimum no. of

keys, then Key can be deleted with no further action

3) Otherwise, if the node contains the minimum no. of Keys, consider the two immediate siblings of the node.

4) If one of the sibling has more than the minimum no. of keys, then redistribute one key from this sibling to the parent node and the one key from the parent to the deficient node.

5) If both immediate sibling have exactly the minimum no. of keys, then merge deficient node with one of the sibling node, and one entry from the parent node. *

6) If this leaves the parent node with too few keys, then the process is propagated upward. *

*