ALGORITHM

- The word "Algorithm" comes from name of parsia author Abu, Jafar Hohammad in 825 AD.

- Algorithm is a step-by-step set of instructions that tells you how to solve a problem or do a task.

[redacted] Of an Algorithm:

- Clear and Unambigous :- easy to understand with no confusion.

- Input : Take input values to work with

- Output : outcome after processing the input

- Finiteness : An algorithm must have a limited number of steps.

- Effectiveness : simple enough for anyone to follow and perform

- Definiteness : Each step must be clearly defined.

** An algorithm must be seen to be believed **

Youtube Channel: Multi Atoms Plus

COMPLEXITY OF AN ALGORITHM

- The complexity of an algorithm computes the amount of time and space required by an algorithm for an input of size(n).

→ Complexity of An Algorithm can be of two types:-

- Space Complexity : This tells us how much memory (or space) an algorithm needs during its execution.

  Example : If an algorithm has to create new lists or tables

- Time Complexity :- This tells us how much time an algorithm takes during its execution.

  Example : Searching a list of 10 names is quicker than searching through 100 names.

Cases of Complexity :-

① Best Case : This is when the algorithm runs the fastest or uses the least amount of resources.

YouTube Channel Multi Atoms Dhs

Example: If you're searching for an item in a list and it's the very first item, $O(1)$.

- Worst case: This is when the algorithm runs the slowest or uses the most resource

Example: If you're searching for an item in a list and it's the very last item, $O(n)$ → no. of times

- Average case: This is when the algorithm performs in a typical or middle of the road situation.

Example: If you're searching for an item in a list and that somewhere in the middle $O(n/2)$.

⇒ Rate of Growth:-

The rate at which the running time increases as a function of input is called rate of growth.

Example: $O(1)$ — Constant Time
$O(n^2)$ — Quadratic Growth
$O(\log n)$ — Logarithmic Growth

YouTube Channel: Multi Atoms Pvt

Asymptotic Notations :-

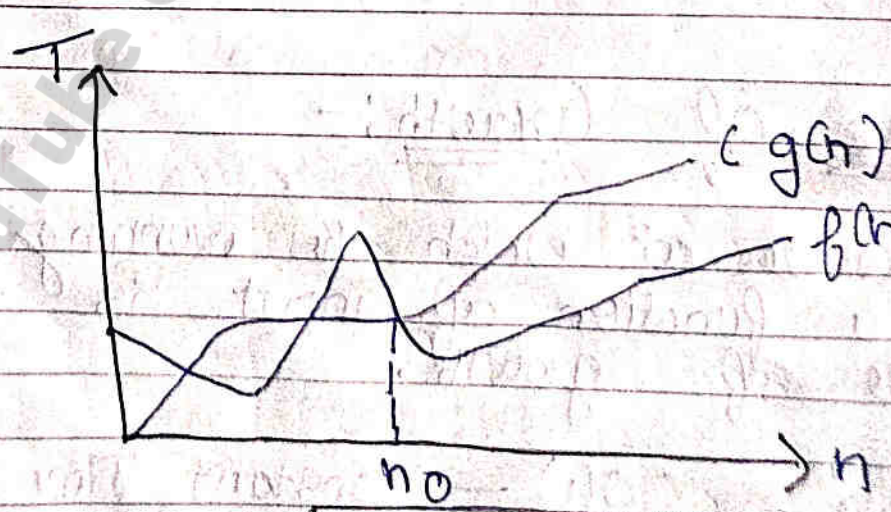
⇒ Asymptotic notation is a mathematical way of representing the time complexity

⇒ Asymptotic notation are of 3 - types :-

1. Big O Notation :-

- Describes the worst-case performance of an algorithm.
- It tells you the upper limit of how long an algorithm can take as the size increases.

$$\boxed{0 <= f(n) <= c * g(n) \text{ forall } N >= N_0}$$



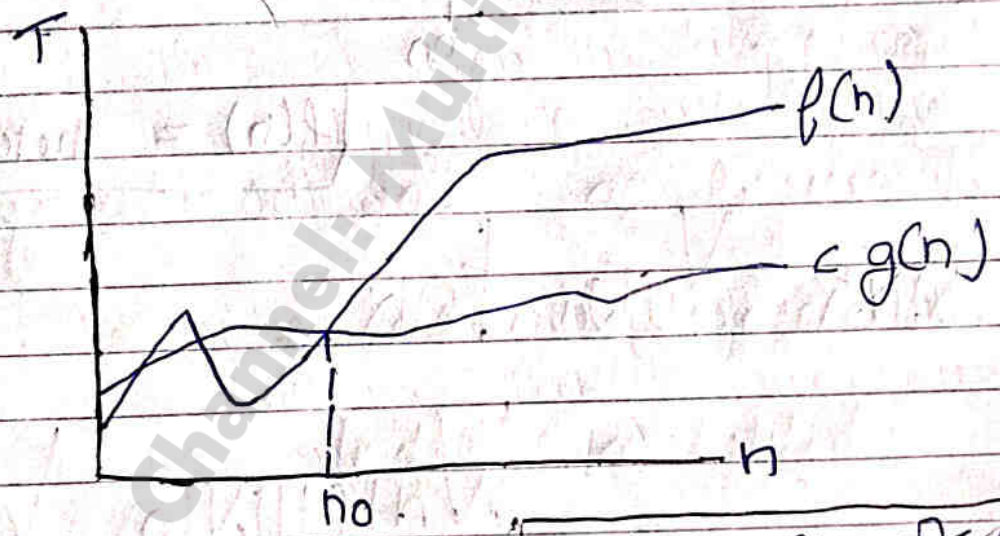
$$\boxed{f(n) = O(g(n))}$$

YouTube Channel - Multi Atoms Plus

2. Omega Notation (Ω):

- Describes the best-case performance of an algorithm
- It tells you the minimum time an algorithm can take.

$$\boxed{0 <= c * g(n) <= f(n) \text{ for all } n >= n_0}$$



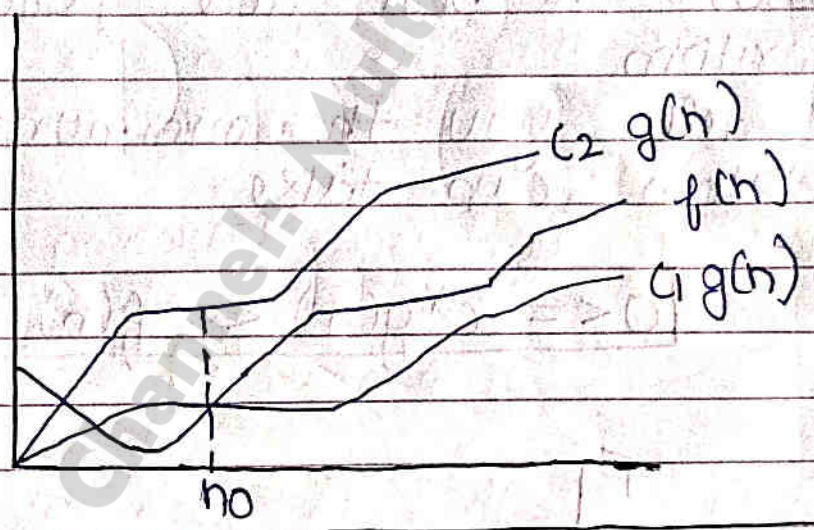
$$\boxed{f(n) = \text{Omega}(g(n))}$$

3. Theta Notation (Θ):

- Describe the average case or a tight bound, meaning it shows both the upper and lower limits of the algorithm
- It tells you the average time an algorithm can take.

YouTube Channel: Multi Atoms Plus

$$\boxed{0 <= c_1 * g(n) <= f(n) <= c_2 * g(n) \ \forall \ n >= n_0}$$



$$\boxed{f(n) = \text{theta } \Theta(g(n))}$$

YouTube Channel: Multi Arms Plus

QUESTION :-

Q.1 $f(n) = 2n^2 + 3n$

$= O(2n^2)$

$\Rightarrow O(n^2)$.

Q.2 $f(n) = \frac{N}{4}$

$= O(\frac{N}{4}) = O(N)$

Q.3

```
for (int i=0; i<n; i++)
{
    ------
}
```

$O(n)$, $i \to 0 \to n-1$

$j: 0 \to m-1$

```
for (int j=0; j<m; j++)
{
    ------
}
```

$O(m)$

$\Rightarrow O(n) + O(m)$

$\Rightarrow O(n+m)$

Q.4

```
for (int i=0; i<n; i++) {
    for (int j=n; j>i; j--)
    {
        ---
    }
}
```

$0 \to n-1$ $O(n)$

$n \to 0$ $O(n)$

$O(n*n)$

$\Rightarrow O(n) * O(n)$

$O(n^2)$

YouTube Channel: Multi Atoms Plus

least complexity = $O(1)$

$O(\log N)$

$O(\sqrt{N})$

$O(N)$

$O(n \log n)$

$O(n^2)$

$O(n^3)$

$O(2^n)$

$O(n!)$

Most complexity = $O(N^n)$

increase

<u>Home Work Question:-</u>

Q.1) $f(n) = 4n^4 + 3n^3$

Q.2)

```
for (int i=0 ; i<n; i++)
{
    for (int j=0; j<n; j++)
    {
    }
}
for (int i=0; i<n; i++)
{
}
```

YouTube Channel: MotiAtoms Plus

RECURRENCE RELATION

- Recurrence relation is an equation that expresses the time complexity of a recursive algorithm in terms of the input size.

Eg: Factorial of a Number

$$T(n) = T(n-1) + 1 \quad \text{for } n > 1; \quad T(0) = T(1) = 0$$

METHODS:

1. Substitution Method:

The Substitution method consist of two main steps:

- Guess the solution
- Use the mathematical induction to find the boundary condition and show that the guess is correct

YouTube Channel: Multi Atoms Pu

Examples :-

Q.1 $T(n) = \begin{cases} 3T(n-1) & , \text{if } n>0 \\ 1 & \text{otherwise, } n=0 \end{cases}$

Solution

$$T(n) = 3T(n-1) \quad \text{——} \; (1)$$

put $n \rightarrow n-1$

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

put $n \rightarrow n-2$ in eq (1)

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

Substitute

$$T(n) = 3T(n-1)$$
$$= 3\{3T(n-2)\}$$
$$\Rightarrow 3^2 T(n-2)$$
$$\Rightarrow 3^2\{3T(n-3)\}$$
$$\Rightarrow 3^3 T(n-3)$$
$$\vdots$$
$$\Rightarrow 3^k T(n-k)$$

Base condition $T(0) = 1$

$$T(n) = 3^n T(0) \quad \left\{ \begin{array}{l} \text{Assume} \\ n-k=0 \\ k=n \end{array} \right\}$$
$$\Rightarrow 3^n \cdot (1)$$
$$\boxed{T(n) = O(3^n)}$$

Answer

YouTube Channel: Multi Atom Plus

Q.2. $T(n) = \begin{cases} 2T(n-1)-1, & \text{if } n>0 \\ 1 & \text{otherwise } n=0 \end{cases}$

7 marks correct

$n=0$
$n<0$

Sol

$$T(n) = 2T(n-1) - 1 \quad \text{——} \quad (1)$$

put $n \longrightarrow n-1$

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

put $n \longrightarrow n-2$ in eq (1)

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

Substitute in eq (1)

$$T(n) = 2T(n-1) - 1$$
$$\Rightarrow 2\{2T(n-2)-1\} - 1$$
$$\Rightarrow 2^2 T(n-2) - 2 - 1$$
$$\Rightarrow 2^2\{2T(n-3)-1\} - 2 - 1$$
$$\Rightarrow 2^3 T(n-3) - 2^2 - 2^1 - 1$$

$$\vdots$$

$$2^k T(n-k) - 2^{k-1} - 2^{k-2} - \cdots - 2^2 - 2 - 1$$

Base case $T(0) = 1$

assume $n-k = 0$
$k = n$

$$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \cdots - 2^2 - 2 - 1$$
$$\Rightarrow 2^n T(0) - 2^{n-1} - 2^{n-2} - \cdots - 2^2 - 2^0$$
$$\Rightarrow 2^n - 2^{n-1} - 2^{n-2} - \cdots - 2^2 - 2^1 - 2^0$$
$$\Rightarrow 2^n - (2^0 + 2^1 + 2^2 - \cdots + 2^{n-2} + 2^{n-1})$$
$$\Rightarrow 2^n - \left\{\frac{2^0(2^n-1)}{2-1}\right\}$$
$$\Rightarrow 2^n - 2^n + 1$$
$$T(n) = O(1)$$

$$\left[S = \frac{a(r^n-1)}{r-1}\right]$$

a, r

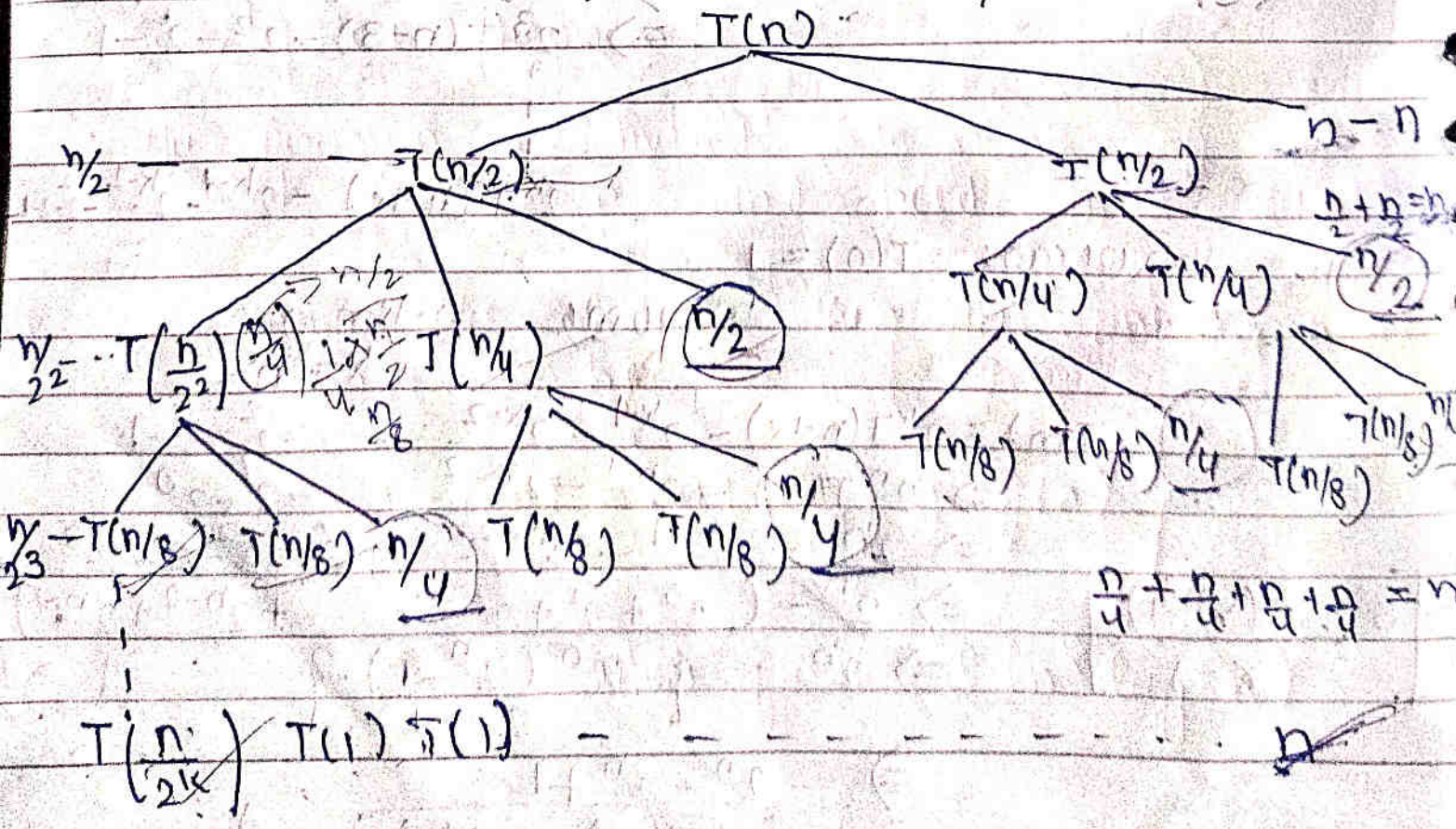
$\frac{2}{2} \Rightarrow \frac{8}{1}$

2. RECURSION TREE METHOD:

- In this method, we draw a recursion tree that breaks down the recurrence relations into smaller subproblems at each level.
- The total work is found by summing the work done at each level of the state.

Examples

Q $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n, & n > 1 \\ 1, & n = 1 \end{cases}$



assume

$$\frac{n}{2^K} = 1$$

$$n = 2^K$$

$$\log_2 n = K \log_2 2 \rightarrow 1$$

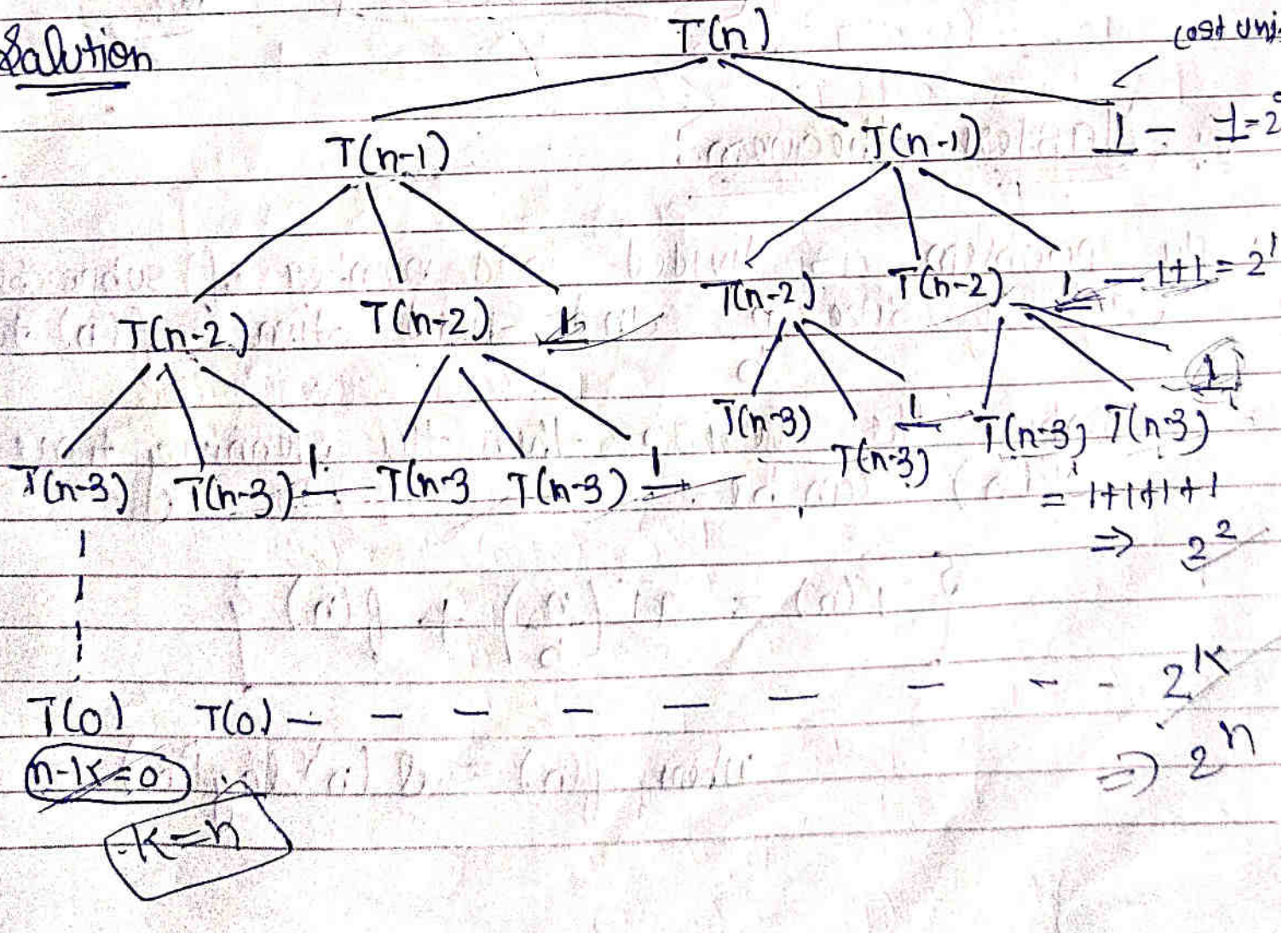
$$\boxed{K = \log n}$$

$$T.C = O(n \cdot K)$$

$$\boxed{T.C \Rightarrow O(n \log n)}$$ Ans

Q $T(n) = \begin{cases} 1 & , n = 0 \\ 2T(n-1) + 1 & , n > 0 \end{cases}$

Solution



assume

$$n-k=0$$

$$\boxed{k=n}$$

$$TC = 2^0 + 2^1 + 2^2 + \cdots + 2^n$$

$$Sum = \frac{a(r^n-1)}{r-1}$$

$$\Rightarrow \frac{2^0(2^{n+1}-1)}{2-1}$$

$$\Rightarrow 2^{n+1}-1$$

$$\Rightarrow 2^{n+1}$$

$$\Rightarrow 2^n \cdot 2$$

$$\boxed{T.C \Rightarrow O(2^n)}$$ Ans

3. Master theorem:

∴ The problem is divided into number of subproblem each of size $\frac{n}{b}$ and need time $f(n)$ to combine the solution then the running time $T(n)$ can be,

$$\left\{ T(n) = aT\left(\frac{n}{b}\right) + f(n) \right\}$$

where $f(n) = \Theta(n^k \log^p n)$

So it become,

$$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$$

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

case-1: if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

case-2: if $a = b^k$

a) if $p > -1$ then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
b) if $p = -1$ then $T(n) = \theta(n^{\log_b a} \log \log n)$
c) if $p < -1$ then $T(n) = \theta(n^{\log_b a})$.

case-3: if $a < b^k$

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

Example:-

Q. $T(n) = 2T\left(\frac{n}{2}\right) + 1$ using Master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$\Rightarrow aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

there $a = 2$, $b = 2$

$$f(n) = \Theta(n^k \log^p n) = \Theta(1)$$

$$\Rightarrow \Theta(n^0 \log^0 n) = \Theta(1)$$

$$\Rightarrow k = 0, \quad p = 0$$

Now

$$a = 2, \quad b^k = 2^0 = 1$$

Case-1 $a > b^k$

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

$$\Rightarrow \Theta(n^{\log_2 2})$$

$$\Rightarrow \Theta(n^1)$$

$$\boxed{T(n) \Rightarrow \Theta(n)}$$ Ans

Ques $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ using Master theorem

Sol $T(n) = aT\left(\frac{n}{b}\right) + \theta\left(n^k \log^p n\right)$

So, $a = 4$, $b = 2$, $f(n) = \theta(n^2 \log^0 n)$
and $k = 2$, $p = 0$

Now $a = 4$, $b^k = 2^2 = 4$, $a = b^k$ & $p > -1$

case-2 (a)

$$T(n) = \theta\left(n^{\log_b a} \log^{p+1} n\right)$$
$$= \theta\left(n^{\log_2 4} \log^{0+1} n\right)$$
$$= \theta(n^2 \log n)$$

$$\boxed{T(n) = \theta(n^2 \log n)}$$

Ques $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$ using Master theorem

Sol $T(n) = aT\left(\frac{n}{b}\right) + \theta\left(n^k \log^p n\right)$

$a = 2$, $b = 2$, $f(n) = \theta(n^1 \log^{-1} n)$
so, $k = 1$ and $p = -1$

Now, $a = 2$, $b^k = 2^1 = 2$, $a = b^k$ & $p = -1$

case-2 (b)

$$T(n) = \theta\left(n^{\log_b a} \log \log n\right)$$
$$= \theta\left(n^{\log_2 2} \log \log n\right)$$

$$\boxed{T(n) \Rightarrow \theta(n \log \log n)}$$

SORTING ALGORITHM

Sorting is defined as the process of arranging a collection of data in elements in a specific order, usually in ascending or descending order.

UNSORTED

170	45	75	90

SORTED

45	75	90	170

STABLE AND UNSTABLE SORT

① In a stable sorting algorithm data is sorted in a way that preserves the original order of element having equal keys

Eg (10 Blue) (20 Red) (20 green) (30 Yellow) (10 Pink)

Stable sort: (10 Blue) (Pink 10) (Red 20) (Green 20) (Yellow 30)

- In a unstable sorting algorithm data is sorted so that the order of element with equal key is not preserved after sorting.

Eg :

2	4	7	4	9
	A Pos		B Pos	

Unstable Sort :

2	4	4	7	9
	B Pos	A Pos		

<u>Stable Sort are</u> :- Merge sort, Insertion sort, Bubble Sort, Counting sort, Radix Sort,

<u>Unstable sort are</u> :- Quick sort, Heap sort, Shell sort,

BUBBLE SORT

$1 < 2$

Bubble sort is a simple sorting algorithm where we repeatedly go through a list, compare adjacent items and swap them if they are in wrong order.

→ This process is repeated until the list is sorted.

Algorithm :-

Step-1
- check if the first element in the input array is greater than the next element in the array.

Step-2
- If it is greater, swap the two numbers otherwise move the pointer forward in the array.

Step-3
- Repeat step-2 until we reach the end of the array.

Step-4
- check if the element are sorted; if not repeat the same process (step1 to step3) from the last element of the array to the first.

Q

5	6	1	3

Phase-1

Sol $i=0$

5	6	1	3

$5<6$ ignore (next)

$i=1$

5	6	1	3

$6 \not< 1$ (swap)

$i = 2$ | 5 | 1 | 6 | 3 |

$6 \not< 3$ (swap)

| 5 | 1 | 3 | 6 |

sorted

Phase-2

$i = 0$ | 5 | 1 | 3 | 6 |

$5 \not< 1$ (swap)

$i = 1$ | 1 | 5 | 3 | 6 |

$5 \not< 3$ (swap)

| 1 | 3 | 5 | 6 |

sorted

Phase-3

$i = 0$ | 1 | 3 | 5 | 6 |

$1 < 3$ true

| 1 | 3 | 5 | 6 |

sorted element

Time Complexity :- $T(n) = (n-1)*(n-1) * c$

$= cn^2 - 2cn + 1$

$\Rightarrow O(n^2)$ worst case

INSERTION SORT

- Insertion sort is a simple sorting algorithm where we build a sorted list one element at a time.

- It works by taking elements from an unsorted list and inserting them into their correct position in the sorted part of the list.

2	3	5	4

Steps:-

1) Start with the first element (which is already "sorted")

2) Pickup the next element and compare it to the element in the sorted part.

3) insert it in the correct position by shifting larger element to the right

4) Repeat for all elements until the entire list is sorted

Algo:-

```
for (int i=1 , i<n ; i++)
{
// fetch        int val = arr[i];
        for (int j=i-1 ; j>=0 ; j--)
        {
            if (arr[j] > val) {
// shift            arr[j+1] = arr[j]
            }
            else {
                break;
            }
        }
// copy         arr[j+1] = val;
}
```

Q Round-1

Sort

10	1	7	6	14	9

(A) value = 1 (B) 1<10 , true shift → 10

Round-2

1	10	7	6	14	9

(A) value = 7 (B) 10>7 , 1<7 , shift → 10, copy
true true

Round-3

1	7	10	6	14	9

(A) value = 6 (B) 10>6 , 7>6 , 1>6 , Shift 10, 7
true , true false , copy

Round 4

1	6	7	10	14	9

sorted

Round 5

(A) value = 14 (B) 10<14, 7<14, 6<14, 1<14 (C) nothi

1	6	7	9	10	14

Sort

Time - Complexity :- Best case - $O(n)$
Average case - $O(n^2)$
Worst case - $O(n^2)$

SELECTION SORT

- Selection sort is a simple sorting algorithm where we repeatedly find the smallest (or largest) element from the unsorted part of the list and swap it with the first unsorted element.

- We do this until the entire list is sorted.

Steps:-

Step1: Start with the first element

Step2: look through the rest of the list to find the smallest element.

3. Swap that smallest element with the first element.

4. Move to the second element and repeat the process for the rest of the list

5. Continue until the entire list is sorted.

Algo :-

```
int i, j, min, temp;
    for (i=0; i<=n-1, i++)
    {
        min = i;
        for (j=i+1; j<=n-1; j++)
        {
            if (arr[j] < arr[min])
                min = j;
        }
        temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
```

Swap

Time Complexity :-

Best case : $O(n^2)$
Average case : $O(n^2)$
Worst case : $O(n^2)$

Q

0	1	2	3	4	5
5	3	8	2	1	4

swap min = 1

1	3	8	2	5	4

sorted swap min = 2

1	2	8	3	5	4

sorted swap min = 3

1	2	3	8	5	4

sorted swap min = 4

1	2	3	4	5	8

sorted 5 < 8, true.

1	2	3	4	5	8

sorted

1	2	3	4	5	8

sorted

SHELL SORT :

- Shell sort is a highly efficient sorting sorting algorithm is based on insertion sort algorithm and we can code it easily.

- It roughly sort the data first, moving large elements towards one end and small elements towards the other.

- In shell short several passes over the data is performed.

- After the final pass, the data is fully sorted

Algorithm :

Step-1 - Start
Step-2 - Initialize the value of gap size, say n.

Step-3 - Divide the list into smaller sub-part. Each must have equal interval to n.

Step-4 - Sort these sub-lists using insertion sort.
Step-5 - Repeat this step 2 until the list is sorted
Step-6 - Print a sorted list.
Step-7 - Stop

Q 23, 29, 15, 19, 31, 7, 9, 5, 2

Initial Gap $= floor\left(\frac{N}{2}\right)$, $N = 8$, Gap $= floor\left(\frac{8}{2}\right)$

Gap = 4

Pass = 1

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2

i↑ j↑

i > j swap
i < j nothing

if (i > j) $23 \not> 31$ false
if (i < j) $23 < 31$ true nothing

23	29	15	19	31	7	9	5	2

i j swap

if (i > j) $29 > 7$ true (swap)

23	7	15	19	31	29	9	5	2

i j swap if (i > j) true

23	7	9	19	31	29	15	5	2

i j swap i > j true

23	7	9	5	31	29	15	19	2

i j swap i > j true

23	7	9	5	2	29	15	19	31

i j swap

2	7	9	5	23	29	15	19	31

i↑ j↑

Pass 2

$$gap = floor\left(\frac{gap}{2}\right)$$
$$= floor\left(\frac{4}{2}\right)$$
$$\Rightarrow 2$$

2	7	9	5	23	29	15	19	31

i j

i<j nothing

2	7	9	5	23	29	15	19	31

i j

i>j swap

2	5	9	7	23	29	15	19	31

i j

i<j nothing

2	5	9	7	23	29	15	19	31

i j

i<j nothing

2	5	9	7	23	29	15	19	31

i j

swap i>j

2	5	9	7	15	29	23	19	31

i j

swap i>j

2	5	9	7	15	19	23	29	31

i j

i<j nothing

Pass 3

$$gap = floor\left(\frac{gap}{2}\right)$$
$$\Rightarrow floor\left(\frac{2}{2}\right) = 1$$

2	5	9	7	15	19	23	29	31

i j i<j nothing

2	5	9	7	15	19	23	29	31

i j i<j nothing

2	5	9	7	15	19	23	29	31

i j i>j swap

2	5	7	9	15	19	23	29	31

i j nothing

2	5	7	9	15	19	23	29	31

i j nothing

2	5	7	9	15	19	23	29	31

i j nothing

2	5	7	9	15	19	23	29	31

i j

2	5	7	9	15	19	23	29	31

i j

2	5	7	9	15	19	23	29	31

gap $\Rightarrow \left(\frac{1}{2}\right)$ $\Rightarrow 0$ → sorted.

Time Complexity :-

Best case $\Omega(n \log n)$

Worst case $O(n^2)$

MERGE SORT

- Merge sort is a sorting algorithm that uses the idea of divide and conquer.
- It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.
- This process is repeated until the entire array is sorted

Algorithm:-

```
Merge_sort (A, lb, ub)
{
        if (lb < ub)
        {
            mid = (lb-ub)/2;
            merge_sort (A, lb, mid);
            merge_sort (A, mid+1, ub);
            merge (A, lb, mid, ub);
        }

merge (A, lb, mid, ub)
{
        i = lb;
```

```
j = mid+1;
k = lb;

while (i<=mid && j<=ub)
{
    if (a[i] <= a[j])
    {
        b[k] = a[i];
        i++;
        k++;
    }
    else {
        b[k] = a[j];
        j++;
        k++;
    }
}
if (i> mid)
{
    while (j<=ub)
    {
        b[k] = a[j]
        j++;
        k++;
    }
}
else {
    while (i<=mid)
    {
        b[k] = a[i];
        j++
        k++;
    }
}
```

```
for (k = lb ; k <= ub ; k++)
{
    arr[k] = b[k];
}
}
```

<u>Time complexity</u> :-

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Master theorem

$a = 2$
$b = 2$
$k = 1$
$P = 0$

$a = 2, \quad b^k = 2^1 = 2$

case :- 2 $\quad a = b^k \quad , P = 0$

(a) $\quad P > -1 \quad ,$

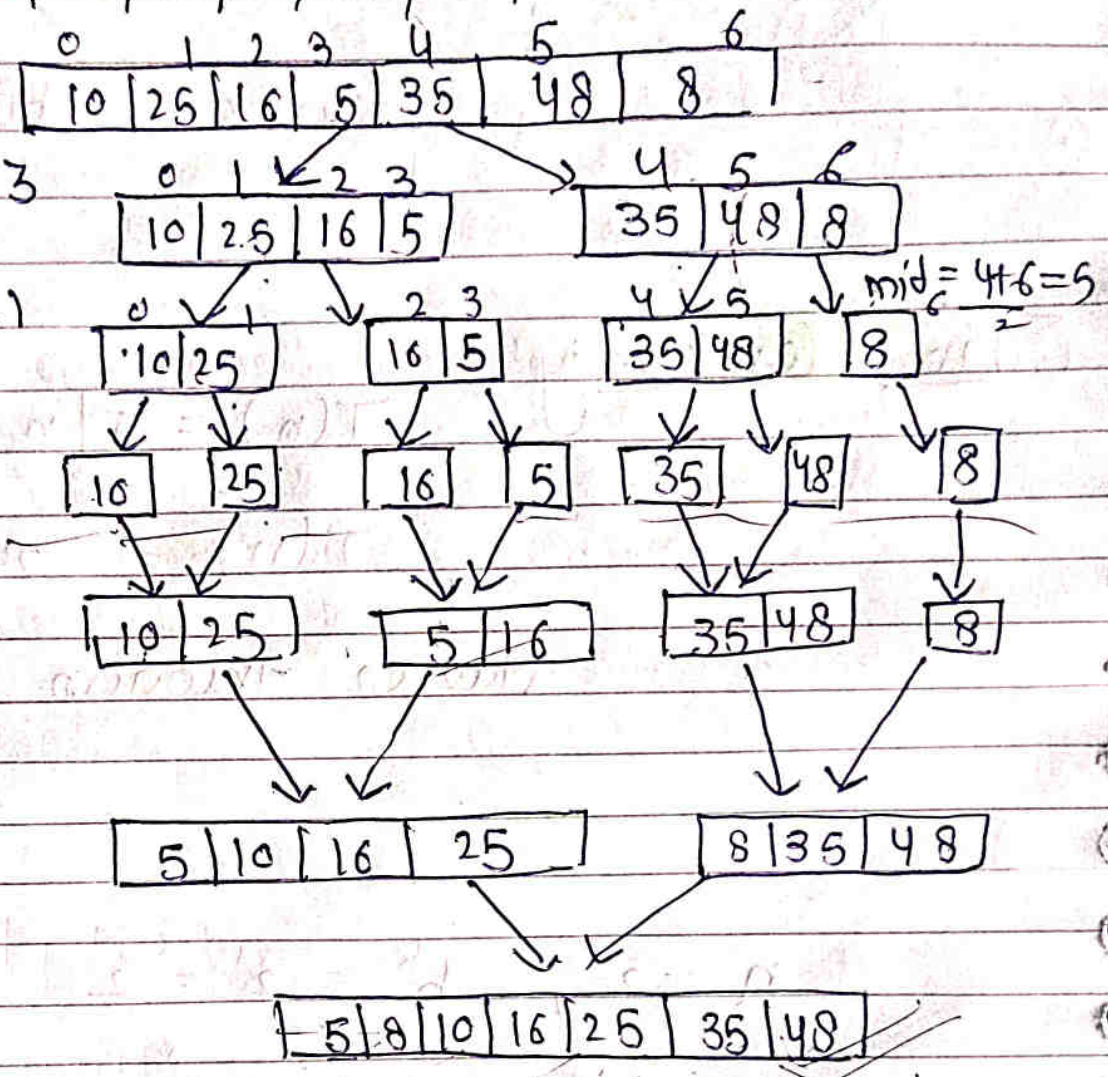
$$T(n) = \Theta(n^{\log_b a} \log^{P+1} n)$$

$$\Rightarrow \Theta(n^{\log_2 2} \log^1 n)$$

$$\boxed{T(n) \Rightarrow \Theta(n \log n)}$$

Q (Imp)

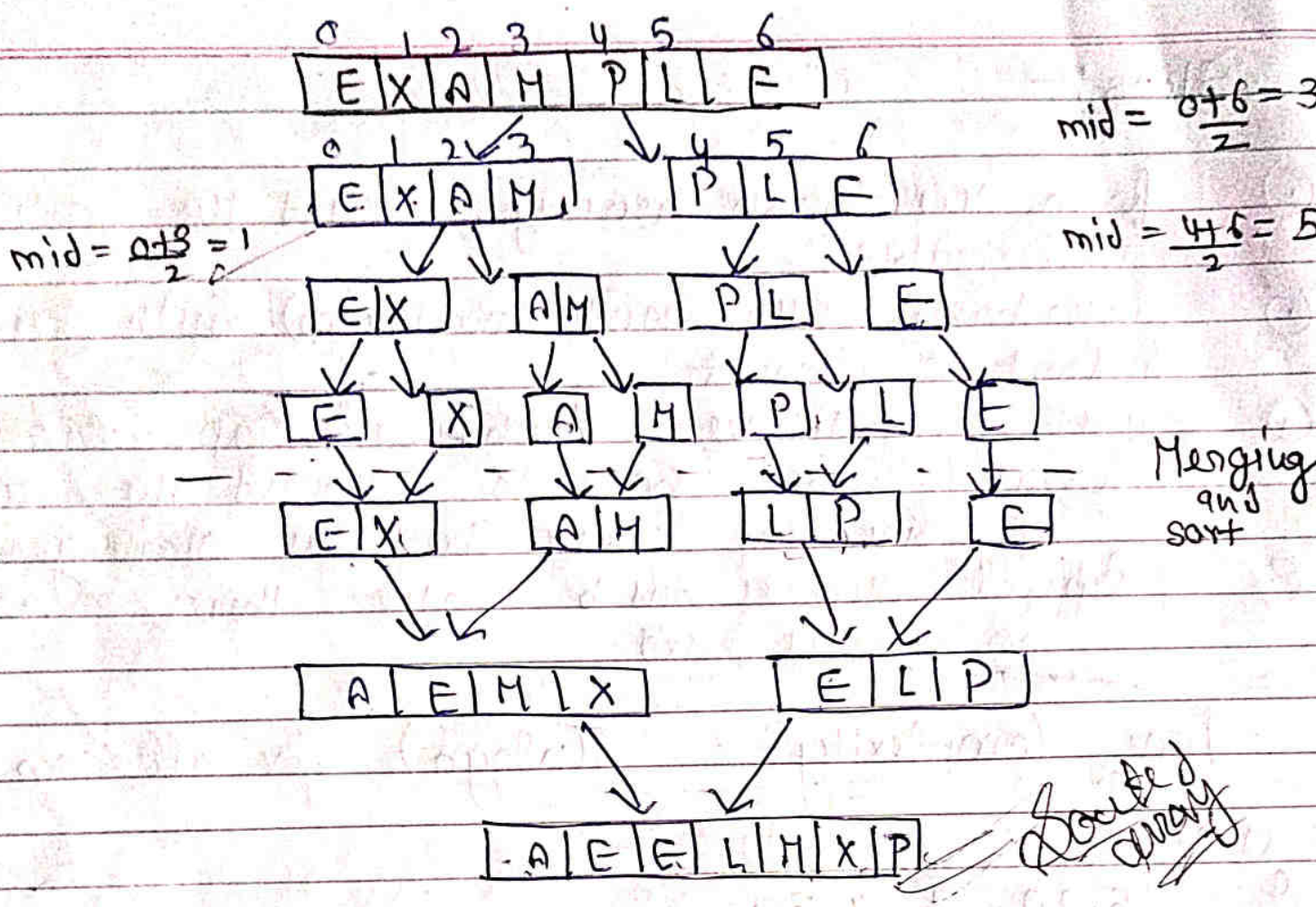
10, 25, 16, 5, 35, 48, 8



(Imp)

Q Among merge sort, insertion sort and quick sort which sorting technique is the best in worst case. there algorithm to sort the EXAMPLE in alphabetic order.

Sol Merge sort technique is best in worst case because of its time complexity $O(n \log n)$



HEAP SORT

- Heap sort is a comparison based sorting technique based on binary heap data structure
- Heap sort finds the largest element and puts it and end of array, then the second largest element found and this process is repeated for all the elements

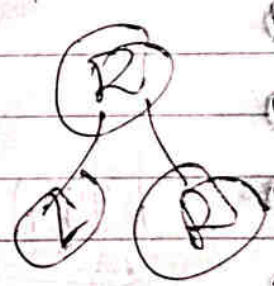
Steps :-

- from the given array, build the initial max heap
- interchange the root (maximum) with the last element
- use repititive downward operation from root node to rebuild the heap of size one less than the starting.
- Repeat step a and b until there are no element exist.

Time Complexity :- $O(n\log n)$ for all cases

(Imp)

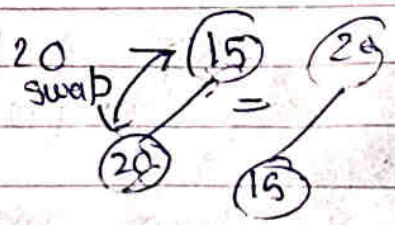
Q 16, 20, 7, 9, 30



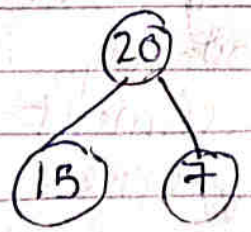
Max heap create

- insert 15

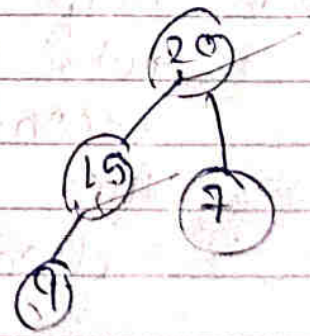
- insert 20



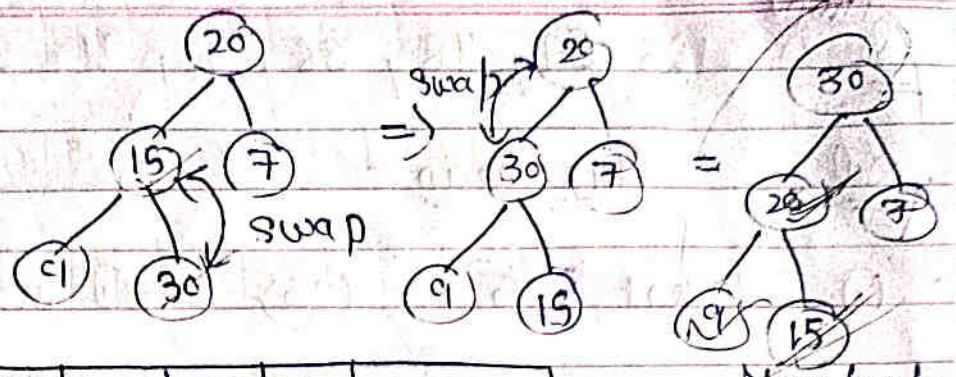
- Insert 7



- insert 9



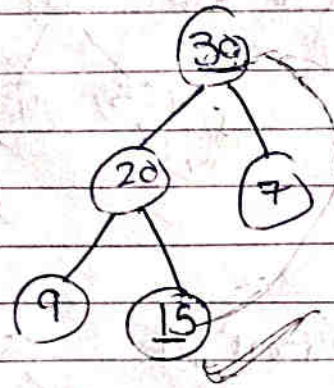
(1) insert 30



Array = | 30 | 20 | 7 | 9 | 15 |

Max heap

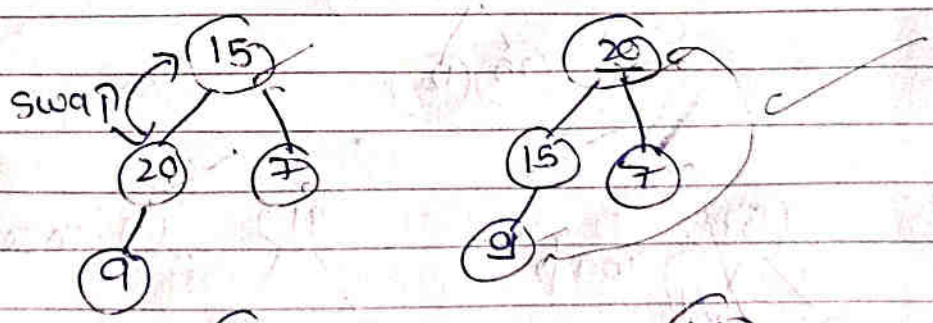
=>



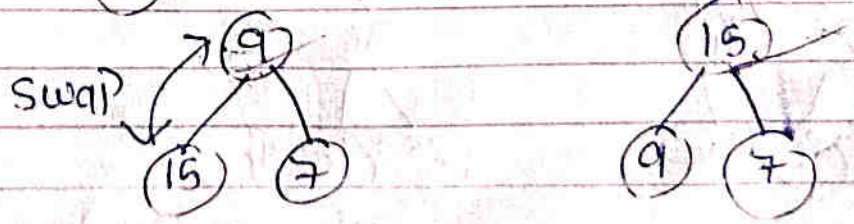
	a[0]	a[1]	a[2]	a[3]	a[4]
sorted heap	30	20	15	9	7

swap with last element

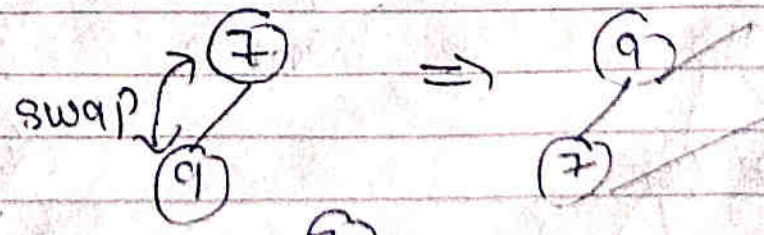
(1) Delete 30



(1) Delete 20



(1) Delete 15



(1) Delete 9

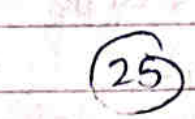
(1) Delete 7

7

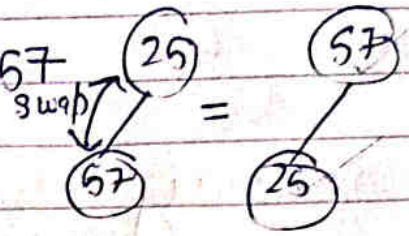
Q (Imp) 25, 57, 48, 36, 12, 91, 86, 32

Sol **Max heap:-**

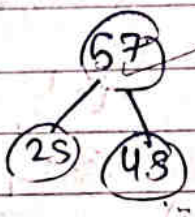
- Insert 25



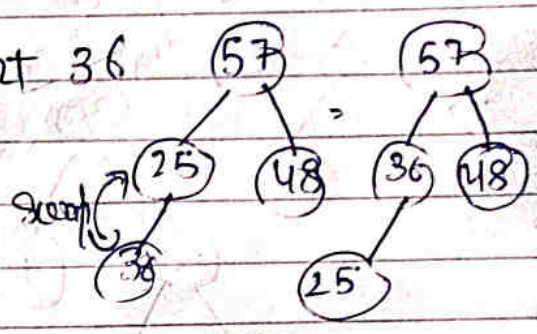
- insert 57



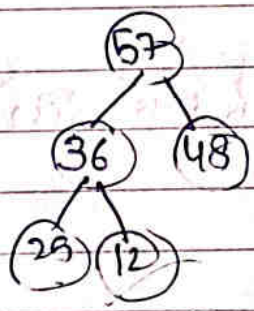
- Insert 48



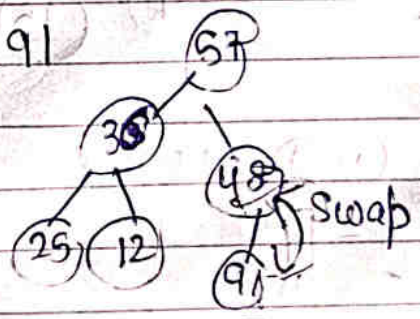
- insert 36

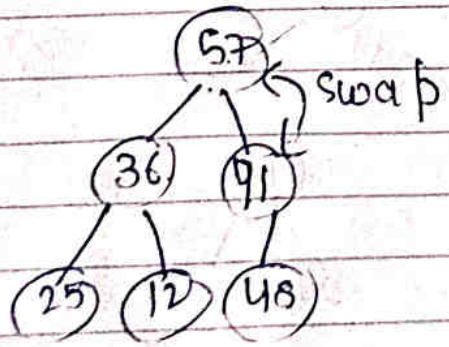


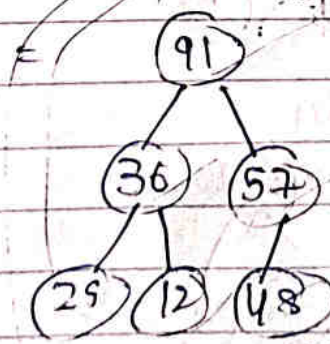
- insert 12



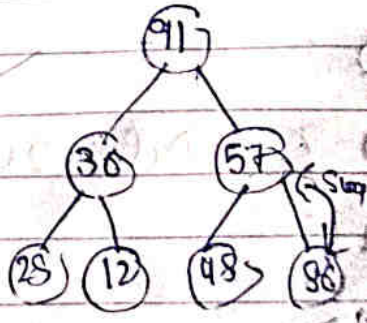
- insert 91

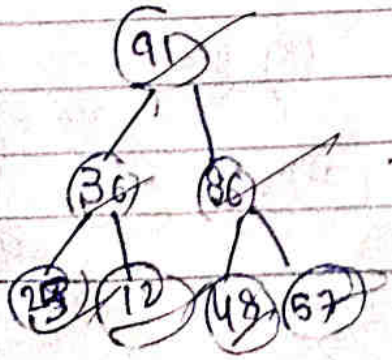




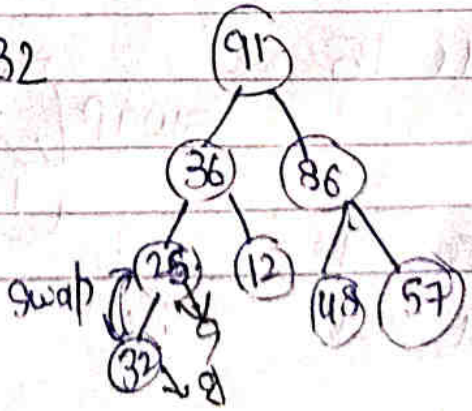


- insert 86

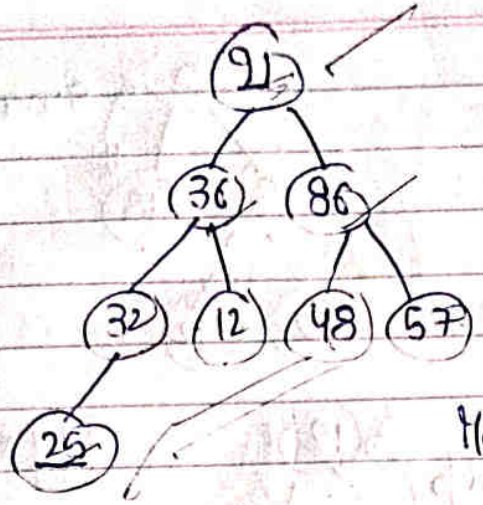




- insert 32



=>

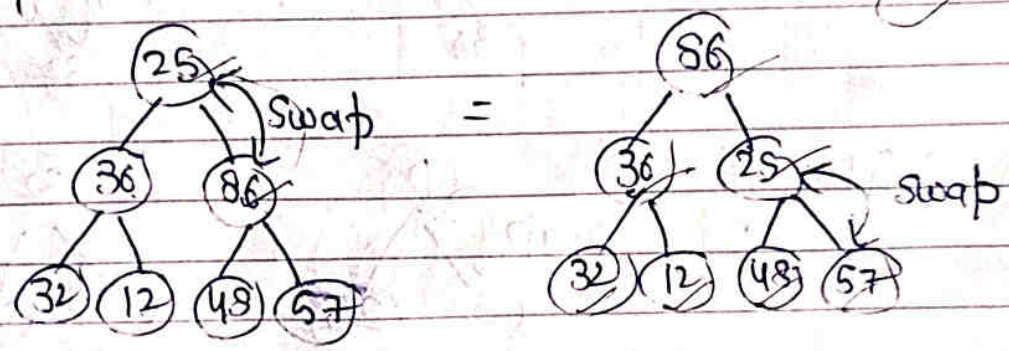


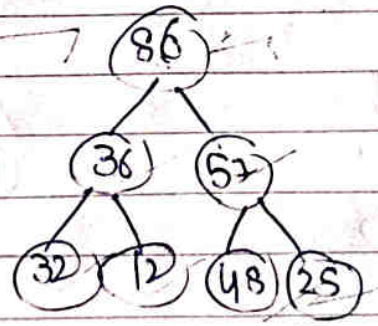
Max heap.

Heap Array

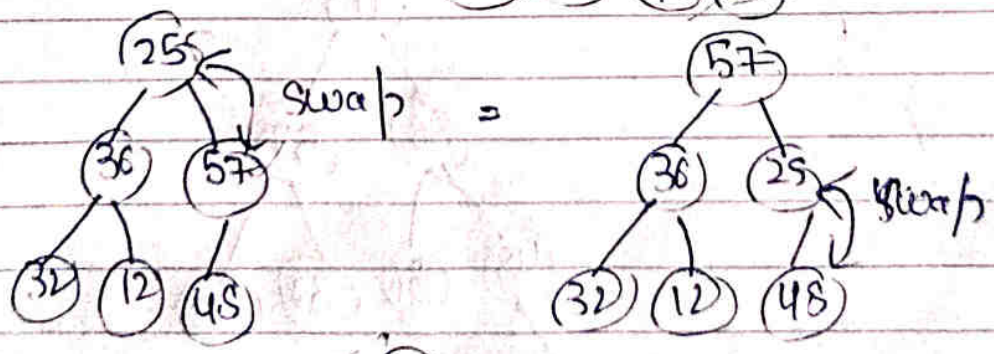
91	86	57	48	36	32	25	12

Delete 91

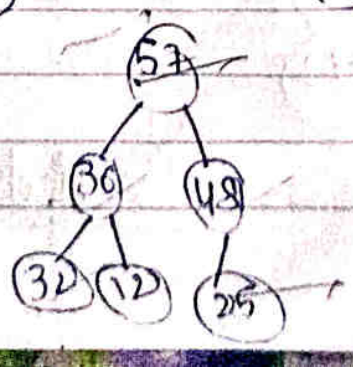




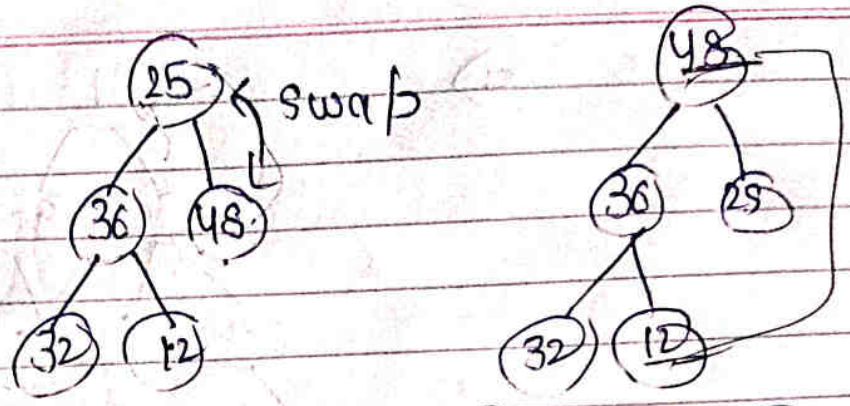
Delete 86



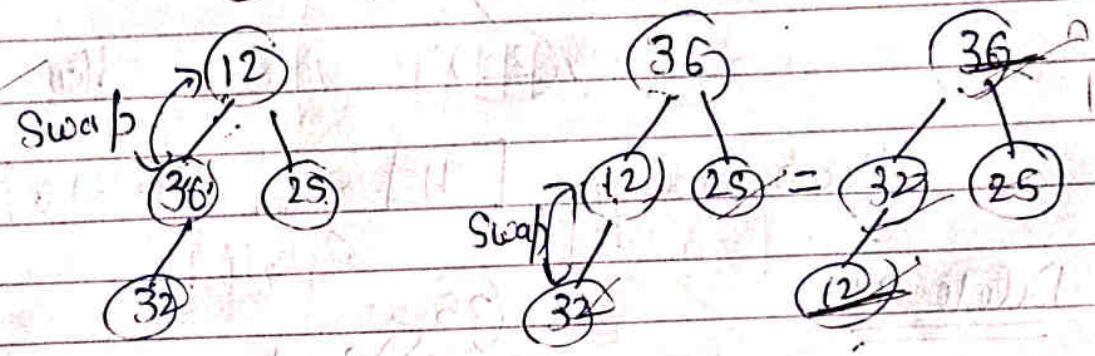
=>



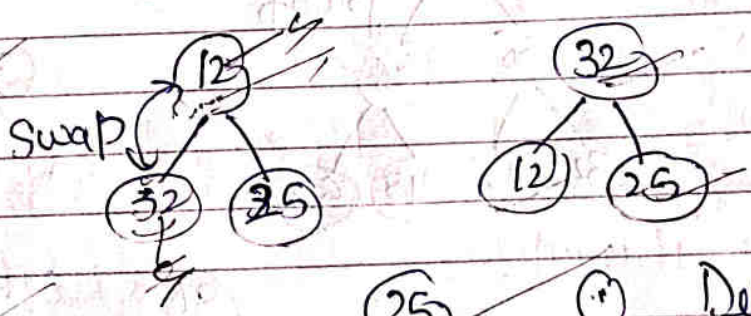
(i) Delete 57



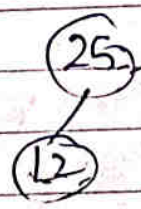
(i) Delete 48



(i) Delete 36

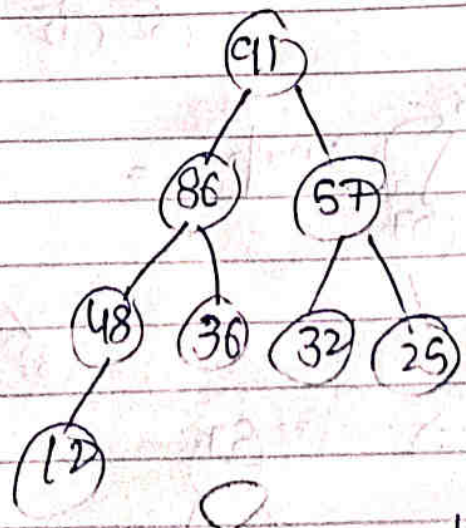


(i) Delete 32



(i) Delete 25

(i) Delete 12.



Sorted Max Heap

COMPARING OF SORTING ALGORITHM :-

Name	Avg case	Worst case	Best case	Stable	Method
① Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	NO	Selection
② Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n)$	Yes	Insertion
③ Shell sort	- - -	$O(n^2)$	$\Omega(n\log n)$	NO	Insertion
④ Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	Yes	Merging
⑤ Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	NO	Selection
⑥ Quick Sort	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	NO	Partition

SORTING IN LINEAR TIME :-

=> Sorting in linear time means arranging elements in a specific order (like ascending or descending) in a way that takes a time proportional to the number of elements, typically express as O(n).

Types :-

- Radix Sort
- Counting Sort
- Bucket Sort

RADIX SORT

⊙ Radix sort is a linear sorting algorithm that sort elements by processing them digit by digit.

⊙ It is an efficient sorting algorithm for integers or strings with fixed-size keys.

Algorithm :-

- find the largest element in the array, due to which we know how will loop is iterate.

For ex :- largest no = 802
3 digit
iterate three tim

- Sort the element based on the unit place digit. we use a stable sorting technique such as counting sort, to sort the digit at each significant place

- Perform Counting sort on the array based on the unit place digit.

- Sort the elements based on the tens place digit.

- Sort the elements based on the hundred place digit

- The array is now sorted in ascending order

Time Complexity :- $T(n) = O(d(n+k))$

d — no. of digit
n — no. of element
k → base of the number system being used.

Q 170, 45, 75, 90, 802, 24, 2, 66

Sol

Input	Ist Pass	IInd Pass	3rd Pass
170	170	802	002
045	090	002	024
075	802	024	045
090	002	045	066
802	024	066	075
024	045	170	090
002	075	075	170
066	066	090	802
			sorted

COUNTING SORT

- Counting Sort is a sorting method that doesn't compare elements directly.
- It works well when the range of number to be sorted is small.
- The main idea is to count how many times each number appears in the input list
- Then, it uses these counts to figure out where each number should go in the sorted list

Q. A = {3, 6, 4, 1, 3, 4, 1, 4}

Sol

0	0
1	0
2	0
3	0
4	0
5	0
6	0

0	0
1	0
2	0
3	~~0~~ 1
4	0
5	0
6	0

at(3)

0	0
1	0
2	0
3	1
4	0
5	0
6	1

at(6)

0	0
1	0
2	0
3	1
4	1
5	0
6	1

at(4)

0	0
1	1
2	0
3	1
4	1
5	0
6	1

at(1)

0	0
1	1
2	0
3	2
4	1
5	0
6	1

at(3)

0	0
1	1
2	0
3	2
4	2
5	0
6	1

at(4)

0	0
1	2
2	0
3	2
4	2
5	0
6	1

at(1)

0	0
1	2
2	0
3	2
4	3
5	0
6	1

at(4)

0	1	2	3	4	5	6	7
1	1	3	3	4	4	4	6

Sorted Array

Q Sort the following array by counting sort.
A = {2, 5, 3, 0, 2, 3, 0, 3}

Sol

0	0	0	0	0	~~0~~ 1
1	0	0	0	0	~~0~~ 0
2	0	1	1	1	1
3	0	0	0	1	1
4	0	0	0	0	0
5	0	0	1	1	1
		at (2)	at (5)	at (3)	at (0)

0	1	0	1	0	~~1~~ 2	0	2
1	0	1	0	1	0	1	0
2	2	2	2	2	2	2	2
3	1	3	2	3	2	3	3
4	0	4	0	4	0	4	0
5	1	5	1	5	1	5	1
	at (2)		at (3)		at (0)		at (3)

0	1	2	3	4	5	6	7
0	0	2	2	3	3	3	5

Sorted Array

⇒ Algo:-

- Find the range of number :- look at the smallest and largest number in the list.
- Make a count array :- to keep track of how many times each number appear in the list

- Count the numbers.
- Update the count arrays.
- Build the sorted lists.

Time Complexity = $O(n+k)$.

where

- n is the number of elements to be sorted
- k is the range of input values

⇒ BUCKET SORT

- Bucket sort or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets.
- Bucket sort works on floating point numbers between range 0.0 to 1.0.
- Input should be uniformly and independently distributed across [0,1] to get a running time of $O(n)$

Algo:-

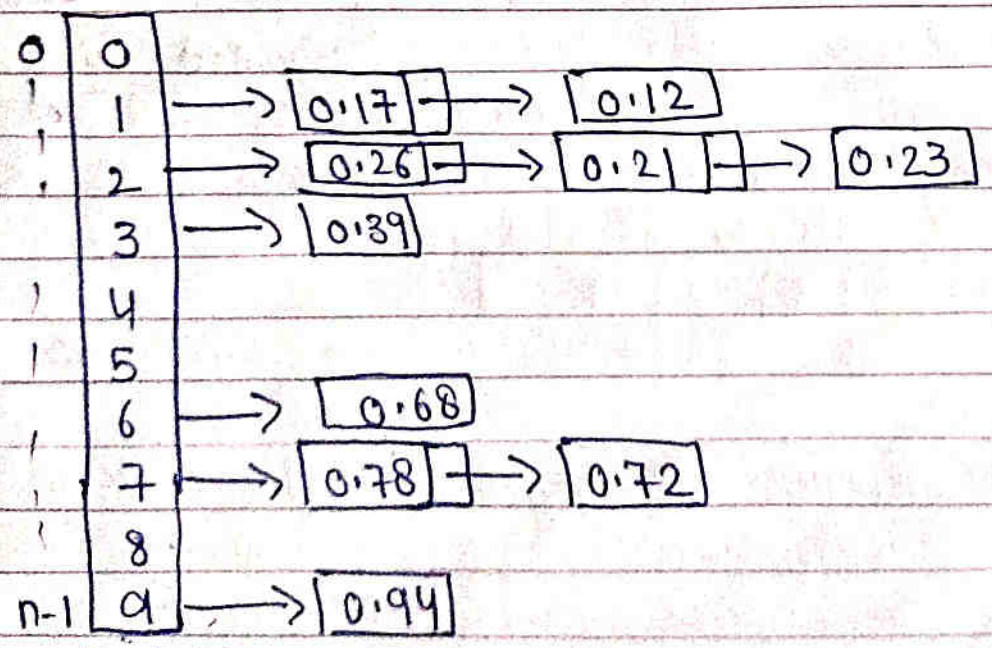
⇒ BUCKET - SORT (A)

Time Complexity :- $O(1) \times n$
$\Rightarrow O(n)$

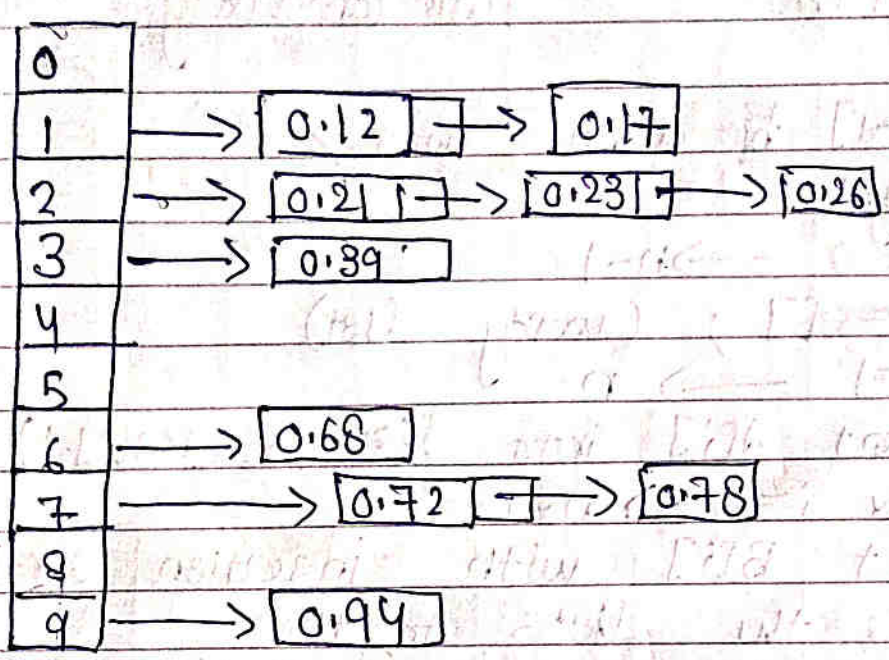
```
let B[0 ... n-1] be a new array
    n = A.length
    for i=0 ---> n-1
        B[i] = [] ; (empty list)
    for i=1 ---> n
        insert A[i] into list B[⌊nA[i]⌋]
        for i=0 -> n-1
    Sort list B[i] with insertion sort.
    Concatenate the lists together
        B[0], B[1] ---- B[n-1] (in order)
```

Q

0.78	0.17	0.39	0.26	0.72	0.94	0.21	0.12	0.23	0.68



Now sort each bucket individually using insertion sort



0.12	0.17	0.21	0.23	0.26	0.39	0.68	0.72	0.78	0.94

Sandeep Pandey

QUICK SORT

- The Quick Sort algorithm items to separate the list of element into two parts and then sort each part recursively.

- It use divide and conquer method.

- In this method, the partician of list is performed based on the element called pivot element

⇒ Algorithm :-

- choose the highest index value as pivot element
- Takes two variables to point left and right of the list excluding pivot.
- left point to low index.
- Right point to the right index.
- if while value at left < Pivot move right
- while value at right > pivot move left
- if both step (5) and (6) not match Pivot swap left and right.
- If left is less than or equal to right the point where they met is new pivot

Q

2	5	3	1	4

Pivot

2	5	3	1	4

Left Right

P=2, R=4, P<R Right move toward left

Pivot

2	5	3	1	4

Left Right

P=2, R=1, P<R false
(swap with Pivot)

Pivot

1	5	3	2	4

Left Right

left=1, P=2, P>L True
L move toward R

Pivot

1	5	3	2	4

Left Right

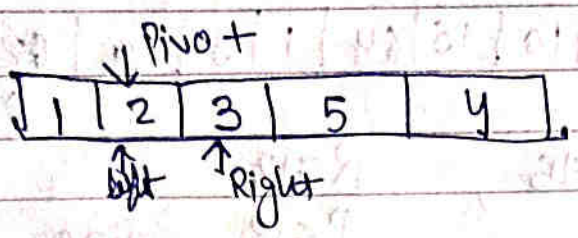
left=5, P=2 P>L false
swap with Pivot

Pivot

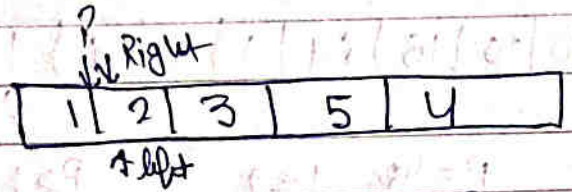
1	2	3	5	4

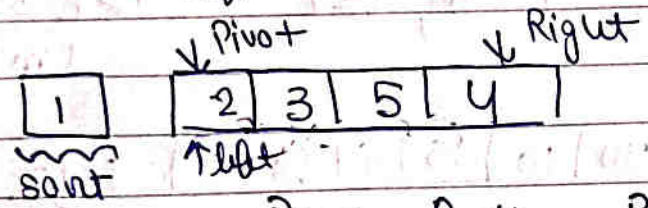
Left Right

P=2, right=5, P<R true.
(Right move toward L)

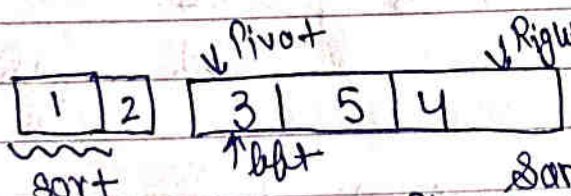


P=2, Right=3, P<R true
(Right move toward L)

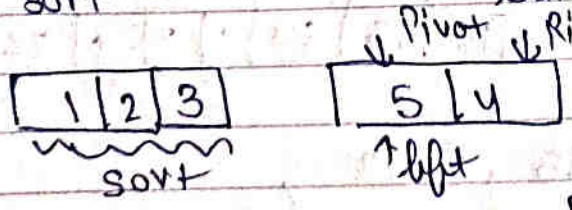




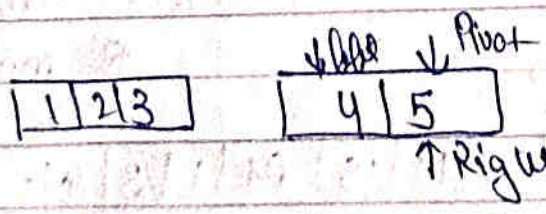
P=2, R=4, P<R true
R move towards L
after the repeat process



Same process above



P<R, false.
swap with Right



l=4, P=5, P>L true L
move toward R.

1 2 3 4 5 — Sorted Array

Q 15, 22, 30, 10, 15, 64, 1, 3, 9, 02

↓P 15	22	30	10	15	64	1	3	9	02
↑L									↑R

P=15 , R=12

P<R false

Swap with pivot

2	22	30	10	15	64	1	3	9	15
↑L									↑R↑P

P=15 , L=2 , P≥L true

L move towards R

2	22	30	10	15	64	1	3	9	15
	↑L								↑R↑P

P=15 , L=22

P≥L false

swap with Pivot

2	↓P 15	30	10	15	64	1	3	9	22
	↑L								↑R

P=15 , R=22 ,

P<R true

R move toward left

2	↓P 15	30	10	15	64	1	3	9	22
	↑L								↑R

P=15 , R=9

P<R , false swap with Pivot

2	9	30	10	15	64	1	3	15	22
	↑L							↑R	↓P

P=15 L=9

L<P

9<15 true. L move toward R

2	9	30	10	15	64	1	3	15	22
	↑L							↑R	↓P

L=30 , P=15

L<P false swap with Pivot

2	9	15	10	15	64	1	3	30	22
	↑L							↑R	↓P

P=15 R=30

P<R R move left

2	9	15	10	15	64	1	3	30	22
	↑L					↑R			↓P

P=15, R=3

P<R, false swap.

2	9	3	10	15	64	1	15	30	22
	↑L						R P		

L=3 , P=15

L<P true, L move to R

2	9	3	10	15	64	1	15	30	22
		↑L					↑R		↓P

L=10, P=15, L<P, true

L move to R

2	9	3	10	15	64	1	15	30	22

↓P ↑L ↑R

P=15, L=15

P>L true

L move toward R

2	9	3	10	15	64	1	15	30	22

↓P ↑L ↑R

P=15, L=64

P>L false

swap with Pivot

2	9	3	10	15	15	1	64	30	22

↓P ↑L ↑R

P=15, R=64, P<R true

R move to L

2	9	3	10	15	15	1	64	30	22

↓P ↑L ↑R

P=15, R=1, P<R false

swap with P

2	9	3	10	15	1	15	64	30	22

↓P ↑L ↑R

P=15, L=1, P>L true

L move to R

2	9	3	10	15	1	15	64	30	22

↓L ↓P ↑R

new pivot

2	9	3	10	15	1	15	64	30	22

1	2	3	9	10	15	15	22	30	64

1	2	3	9	10	15	15	22	30	64

THANK YOU FOR WATCHING

LIKE SHARE SUBSCRIBE

JOIN TELEGRAM FOR NOTES

@MULTIATOMS @MULTIATOMS PLUS

UNIT-1 COMPLETE

UNIT - 02

DESIGN ANALYSIS OF ALGORITHM

LECTURE - 01

Topics :-

- Red Black Trees
- Properties of Red Black Tree
- Rotation
- Algorithm
- Insertion of R-B tree

Subscribe :- @ MULTIATOMS

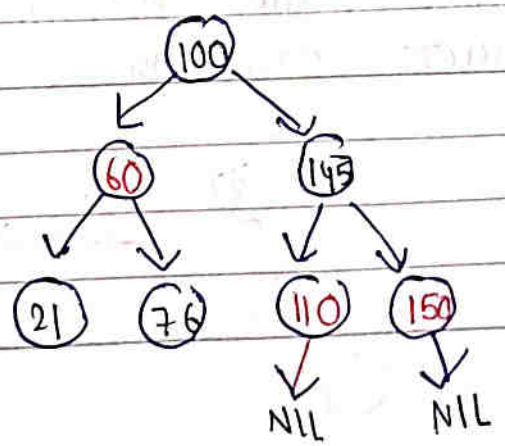
RED - BLACK TREES

- Red black Trees are another type of the balanced Binary Search Trees with two coloured nodes Red and Black.

- It is a self-balancing binary search tree that make use of there colours to maintain the balance factor during the inserting and deletion operation

Properties of Red-Black Tree

1. Node Color :- Each node is either red or black
2. Root Property :- The root of tree is always black
3. Red Property :- Red nodes cannot have red children (no two consecutive red nodes on any path)
4. Black Property :- Every path from a node to its descendant null nodes (leaves) has the same number of black nodes
5. Leaf Property :- All leaves (NIL nodes) are black



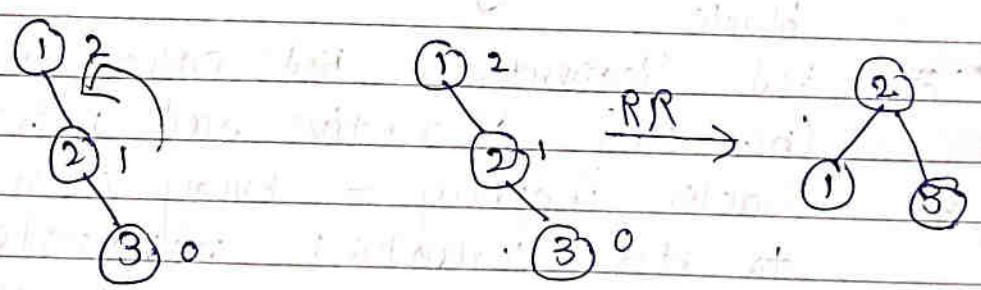
⇒ RED - BLACK TREE

1. **Rotation** :- Basically there are four type of rotation

- Left Rotation → LL
- Right Rotation → RR
- left Right Rotation → LR
- Right Left Rotation → RL

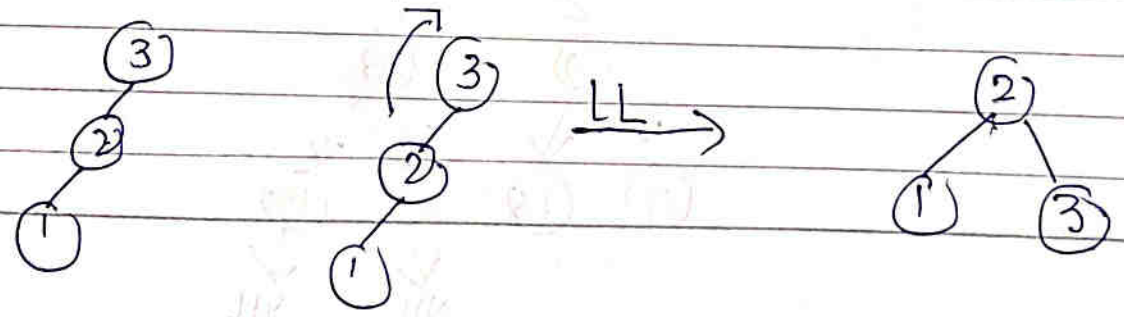
① **Right - Right rotation** :- In RR rotation every node move one position to left from the current position

Ex 1, 2 and 3



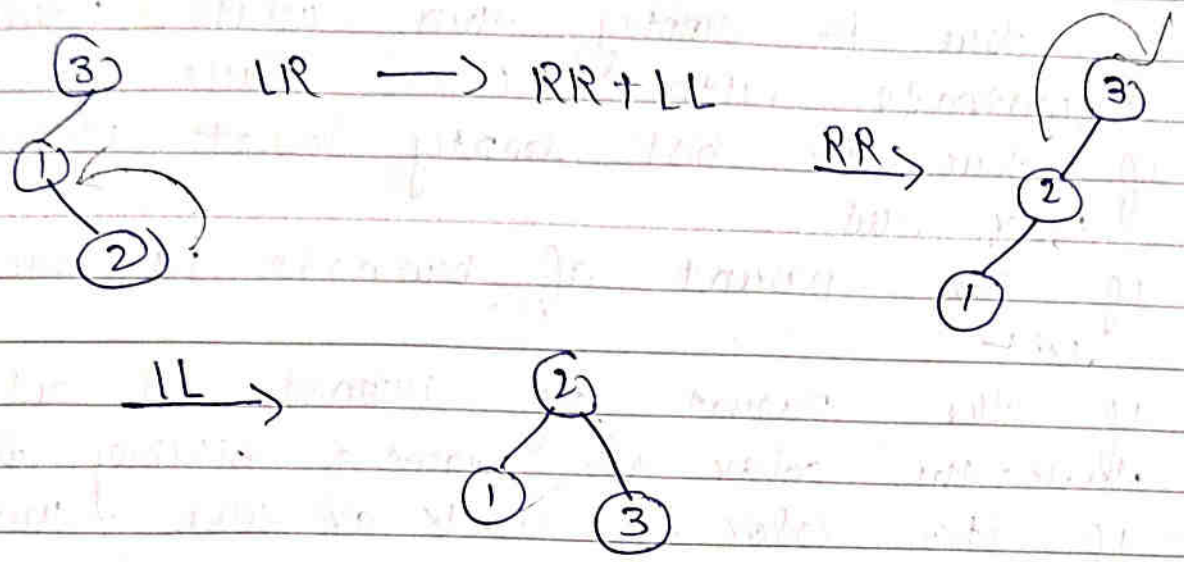
② **left - left rotation** :- In LL rotation every node move one position to right from the current position

Ex



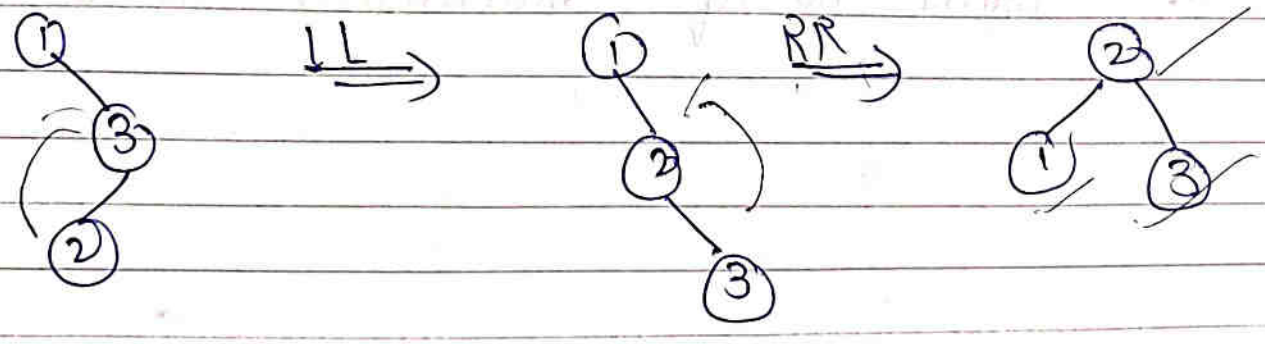
① Left - Right rotation :- In LR position first every node move one position to right then one position to left from the current possition

Ex



① Right - left rotation :- In RL rotation first every node move one position to left then one position to right from the current position

Ex



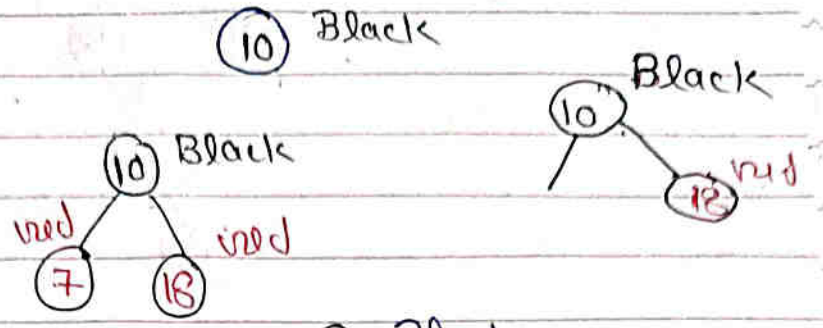
⇒ Red-black tree insertion :-

⇒ Algorithm :-

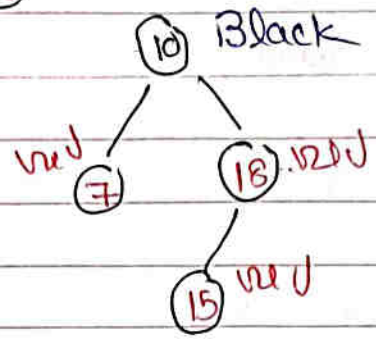
1. If tree is empty then create newnode as rootnode with color black
2. If tree is not empty insert newnode with color red
3. If the parent of new node is black then exit
4. If the parent of newnode is red then check the color of parent's sibling of newnode
5. If its color is black or NULL then make suitable rotation and recolor it.
6. If its color is red then recolor.
7. If a node is red then its children are black.
8. No two adjacent red nodes
9. Count no. of blacknodes in each path.

Q 10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, [redacted]

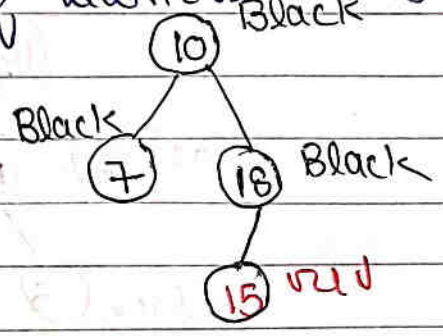
=> Insert 10
=> insert 18
=> Insert 7



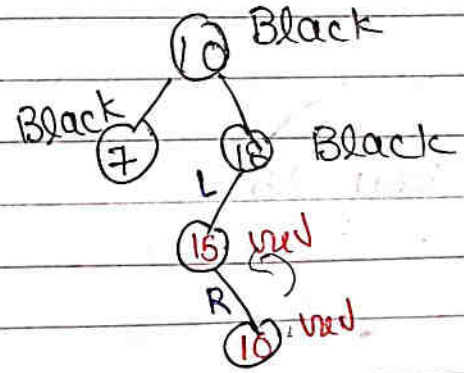
=> Insert 15



=> If the parent of newnode is red then check the color of parent's sibling of newnode, its color is red then recolor. check if parent's parent of newnode is not root node then recolor it

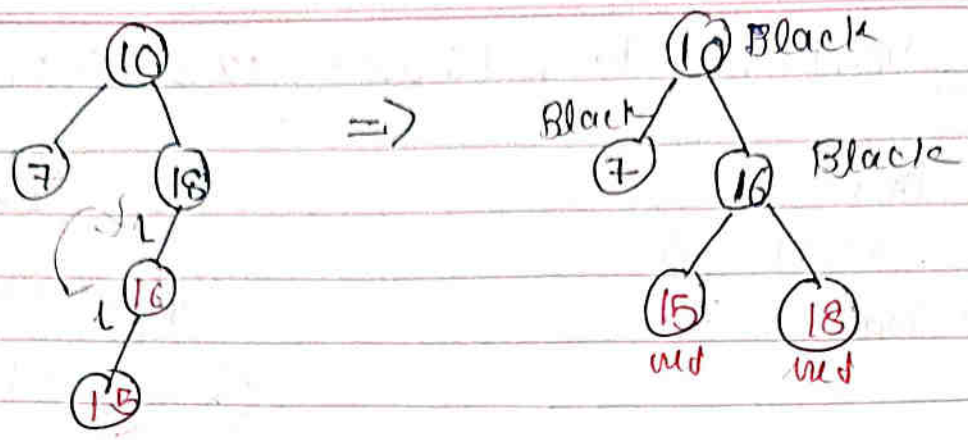


=> Insert 16

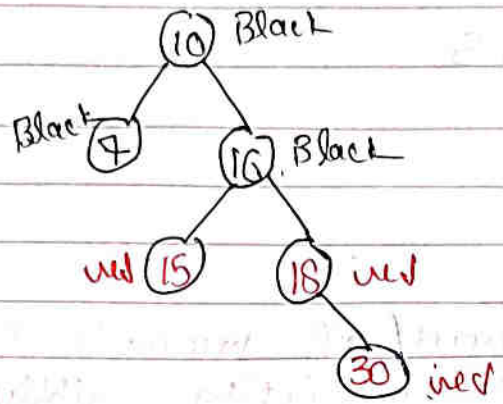


=> If the parent of newnode is red, the check the color of parent sibling of newnode, its color is black or null then perform rotation and recolor

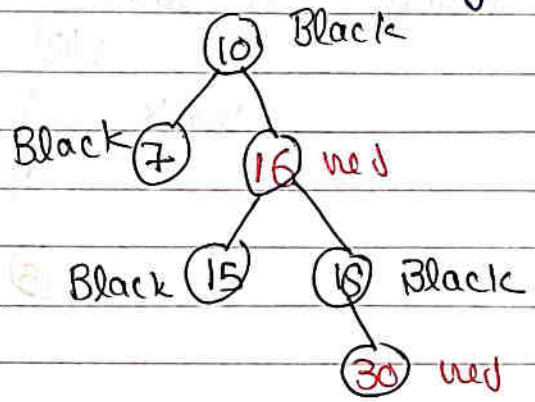
⇒



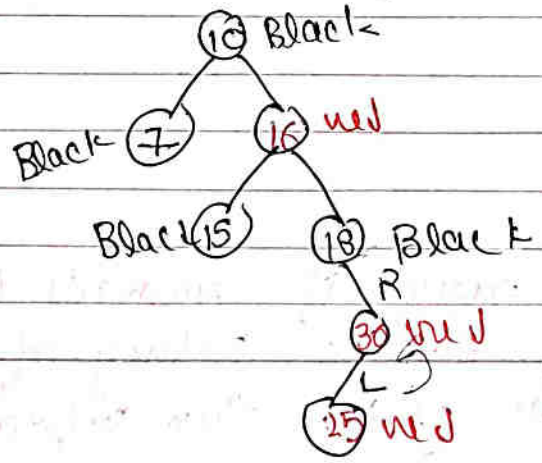
⇒ Insert 30



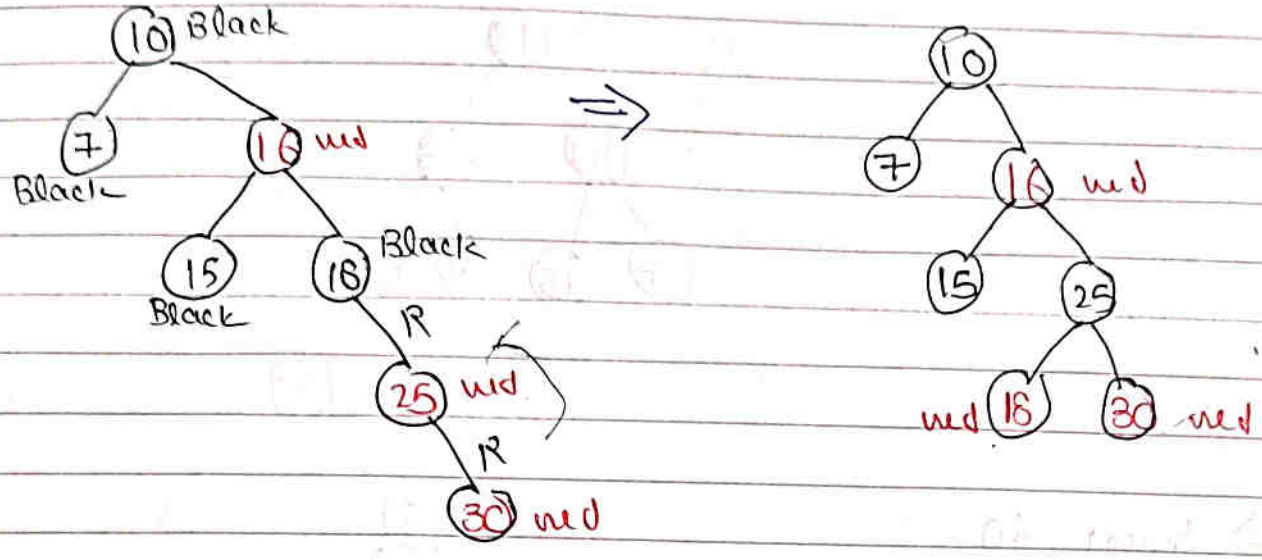
⇒ Sibling color is red, then simply recolor it



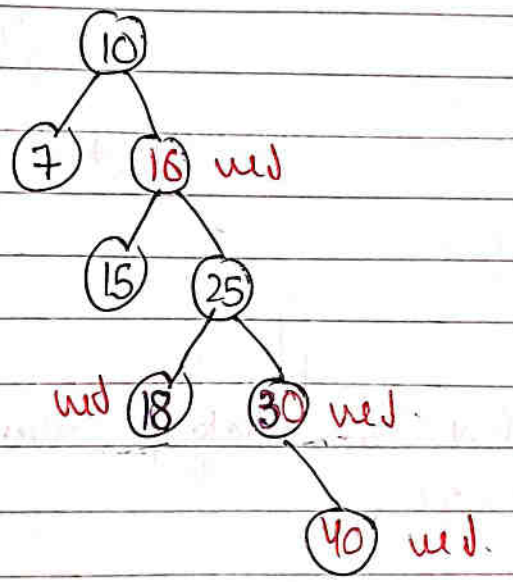
⇒ Insert 25



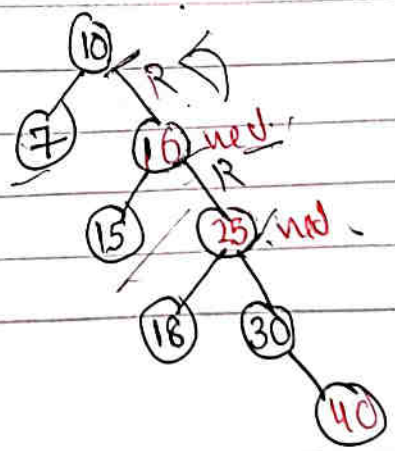
⇒ sibling color is black or null then simply perform rotation and recolor it



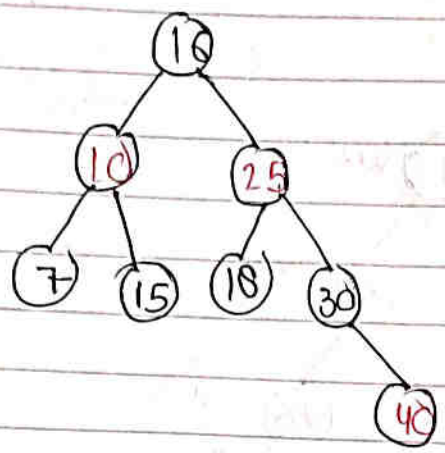
⇒ Insert 40



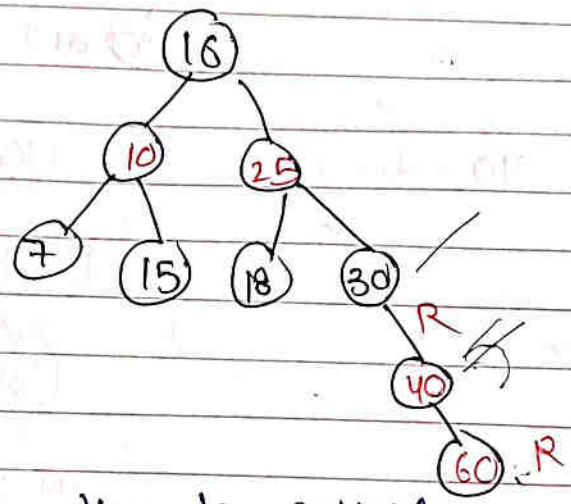
⇒ parent color is red then simply check sibling color is red then recolor it



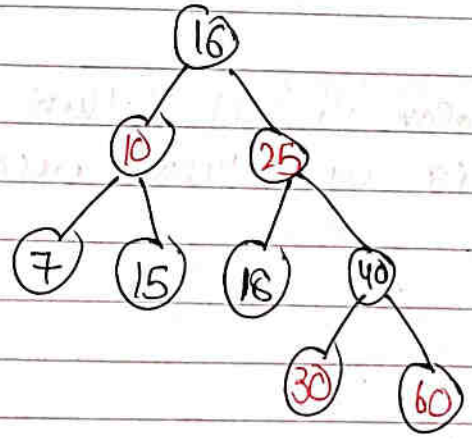
1 sibling color is black, then do suitable rotation then recolor



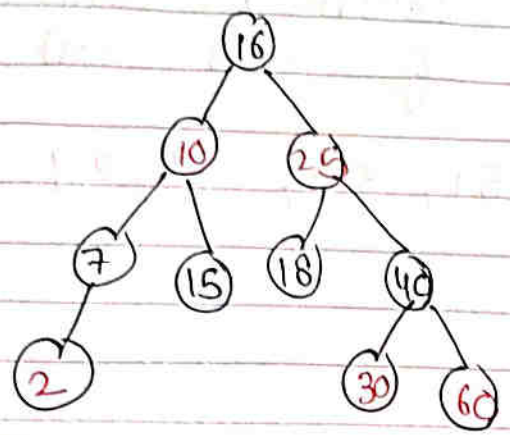
=> Insert 60



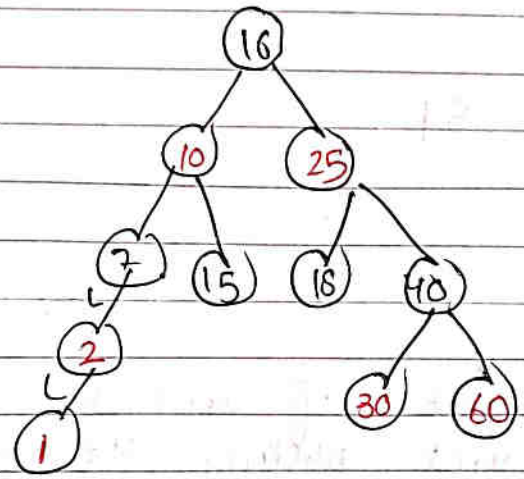
Sibling color is black, then do suitable rotation then recolor



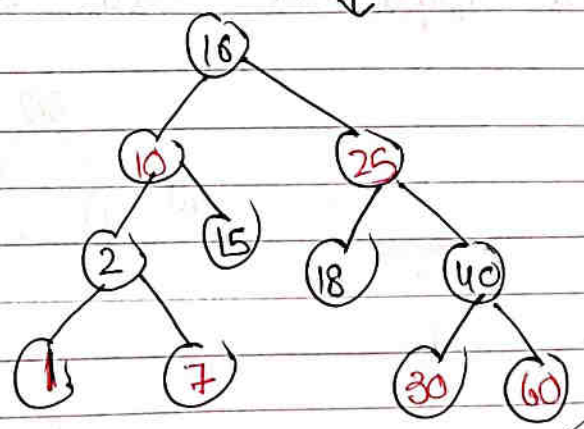
⇒ Insert 2



⇒ Insert 1



⇓



R.B Tree

Q Insert the following elements using the property of R-B tree

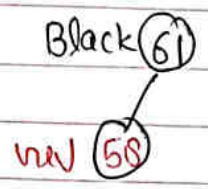
{AKTU 2023-2024}

=> 61, 58, 51, 32, 39, 29

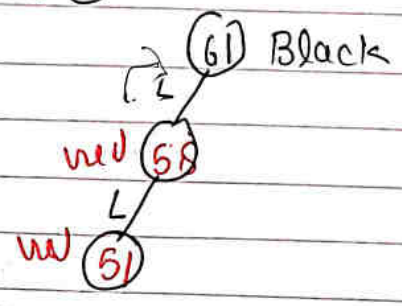
Sol

=> Insert 61

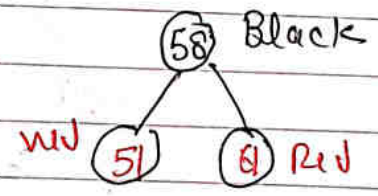
=> Insert 58



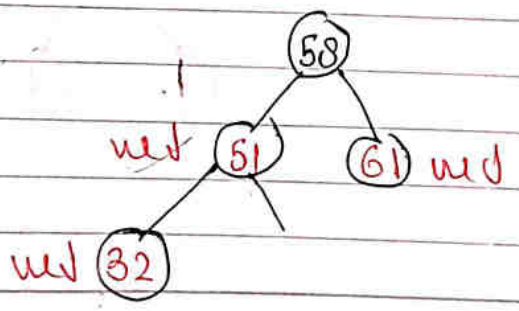
=> Insert 51



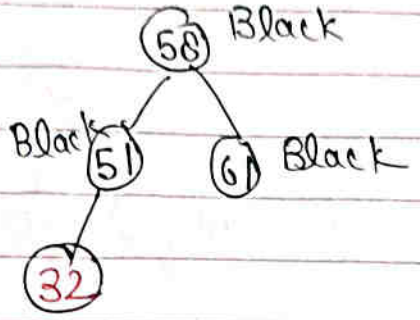
check parent of new node is red, then check it parent sibling its null or black then perform rotation and recolor it



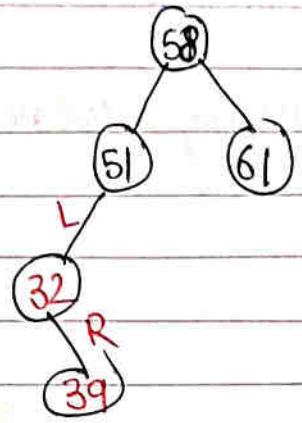
=> Insert 32



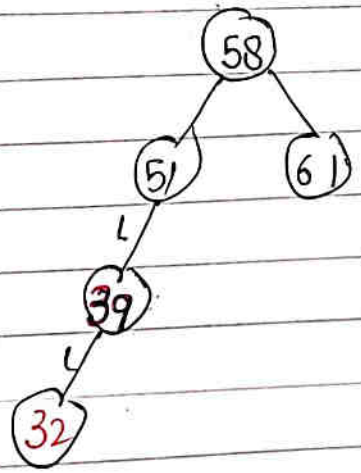
check parent sibling color is red then, recolor it



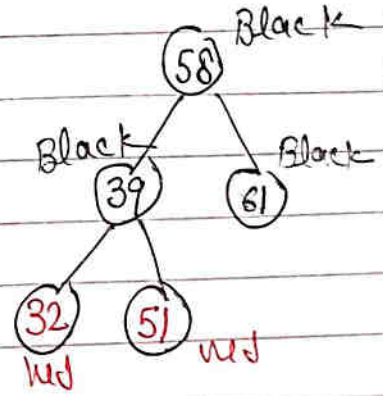
=> Insert 39



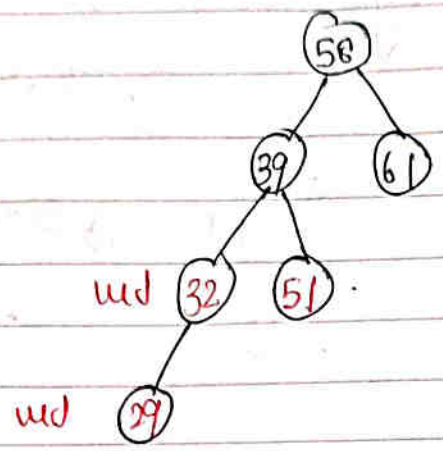
check parent sibling color is red, then perform rotation and recolor it



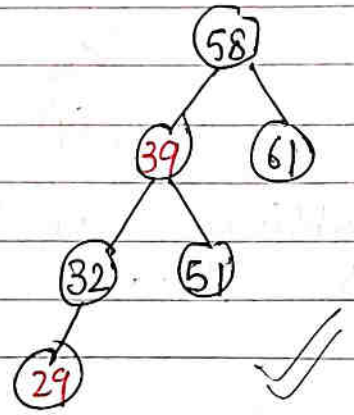
=>



⇒ Insert 29



check parent sibling color is red, then simply recolor it

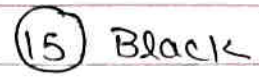


Ans

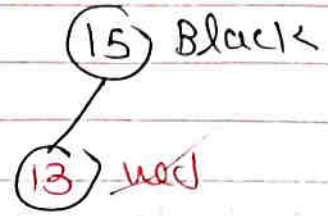
{ AKTU 2021 - 2022 }

Q Insertion 15, 13, 12, 16, 19, 23, 5, 8.

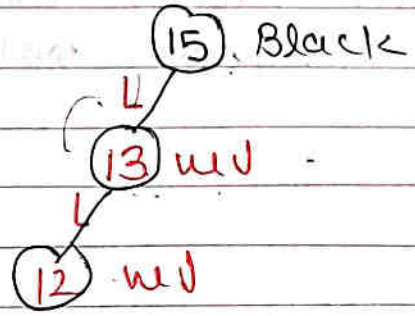
Sol. Insert 15

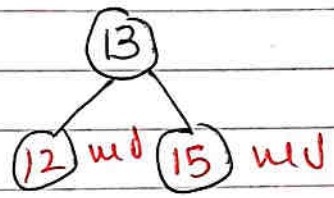


. Insert 13

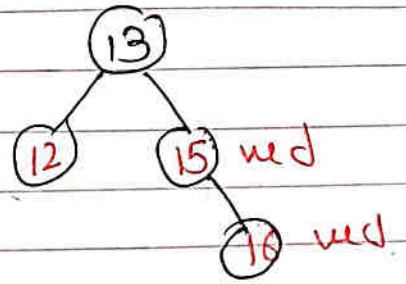


. Insert 12

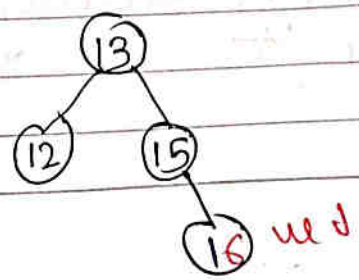




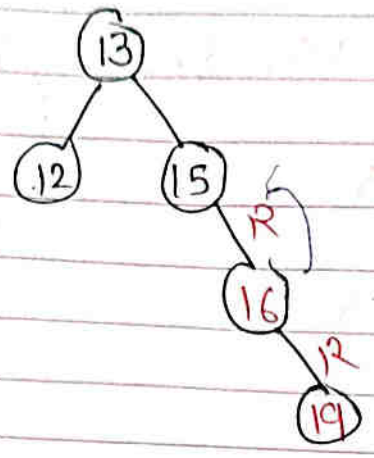
. Insert 16



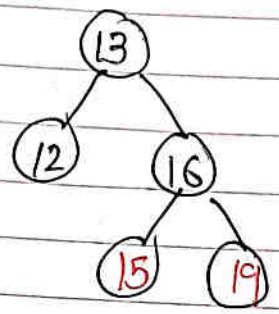
= parent sibling is red, then simply recolor it.



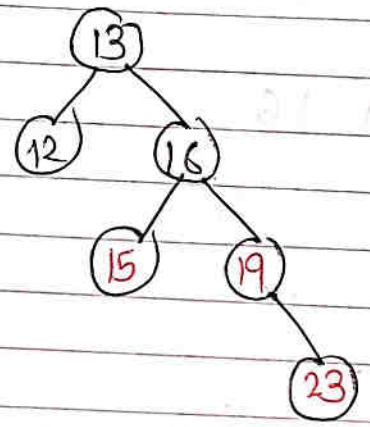
⊙ Insert 19



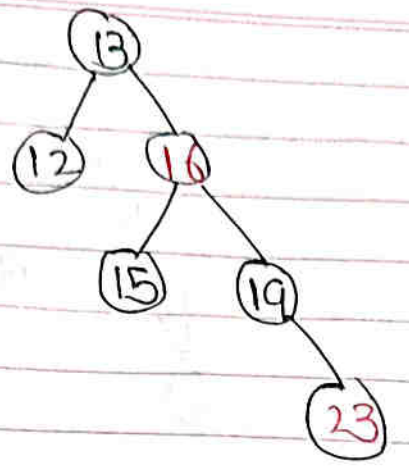
⇒ check parent sibling is black or null then perform rotation and recolor it



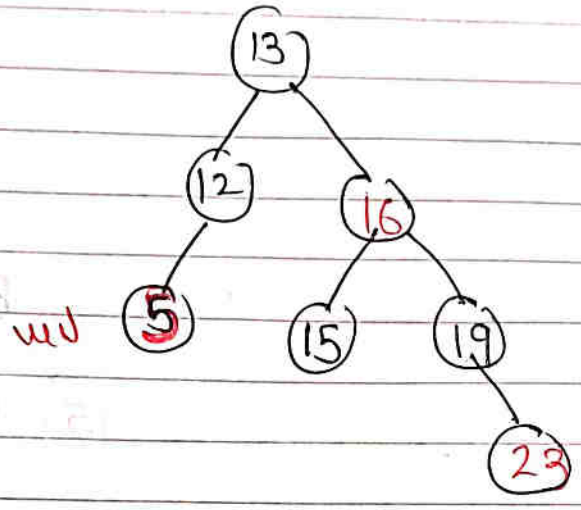
⇒ Insert 23



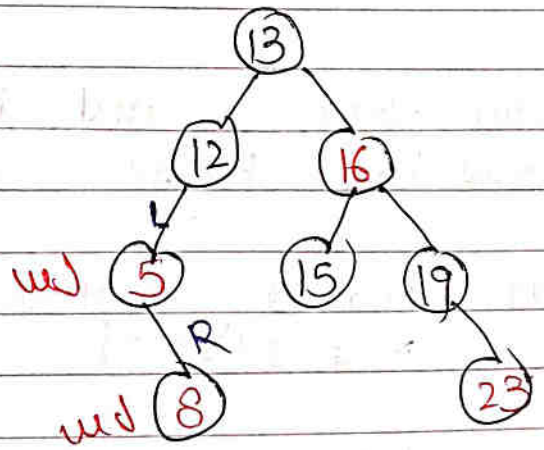
check parent sibling is red, then simply recolor it



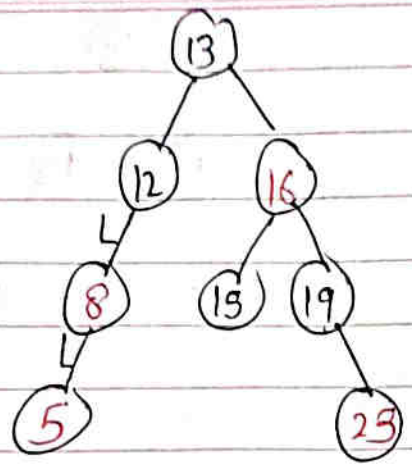
① Insert 5



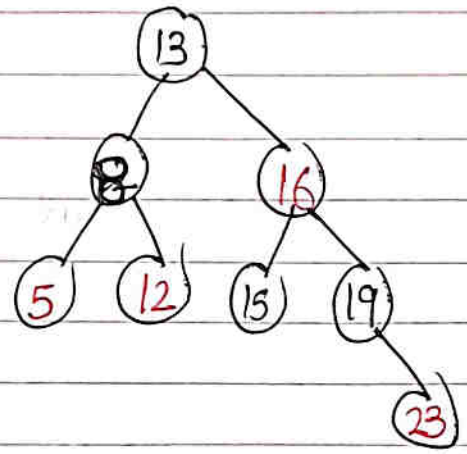
② Insert 8



⇒ check parent sibling is null or black then simply perform rotation and recolor it



$\Downarrow$



Ans

Q Show that a red black tree with n internal node has height at most $2\log(n+1)$ $h/2$

Sol The subtree rooted at any node x contains $\geq 2^{bh(x)}-1$ internal node. $\geq h/2$ $h/2$

$\Rightarrow$ A height-h node has black-height $\geq h/2$

$$n \geq 2^{h/2} - 1$$

$$n+1 \geq 2^{h/2}$$

$$\log(n+1) \geq h/2 \quad , \quad h \leq 2\log(n+1)$$

UNIT - 02

LECTURE - 02

DESIGN ANALYSIS OF ALGORITHM

Topics:-

- Red Black Tree : Deletion
- B - Tree properties
- B - Tree Insertion
- B - Tree Deletion

SUBSCRIBE ~ @MULTIATOMS
&
@MULTIATOMS+

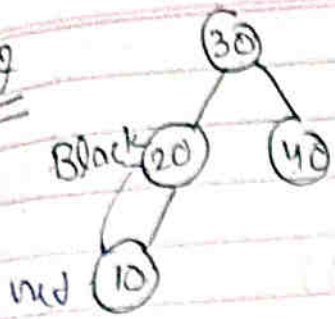
DELETION IN RED - BLACK TREE

=) Deleting a node from a Red- Black Tree involves multiple cases to ensure the tree remains balanced and adheres to the Red - Black Tree properties

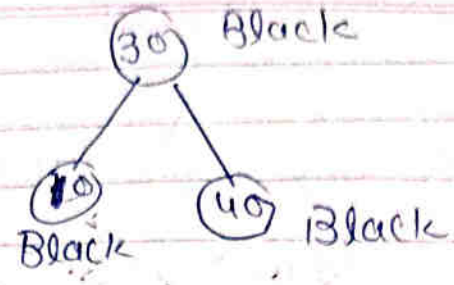
case	check Condition	Action
1.	If a node to be delete is a red leaf node	just remove it from the tree
2.	If DB node is root	Remove the DB and the root node becomes Black
3	(a) If DB's sibling is black and (b) DB's sibling children are black	- Remove the DB - Make DB's sibling red - If DB's parent are is black, make it DB also make it Black
4.	If DB's sibling is red	- swap color DB's parent with DB's sibling - perform rotation at parent node in the direction of DB node

		· check which case can be applied to this new tree and perform that action
5	DB sibling is black and DB sibling's child which is far from DB is black DB's sibling child which is near to DB is red	- Swap color of sibling with sibling's red child - perform rotation at sibling node in direction opposite of DB node. - Apply case 6
6	DB sibling is black, and DB's sibling far child is red	- Swap color of DB's parent with DB's sibling color - perform rotation at DB's parent in direction of DB - Remove DB sign and make the node normal black node - change colour of DB siblings's far red child to black

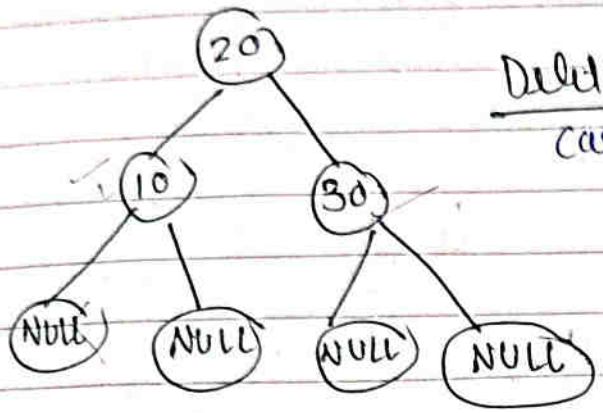
Q



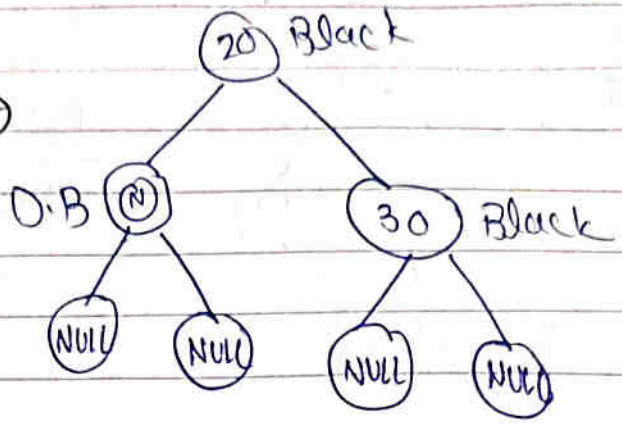
Delete 20 →

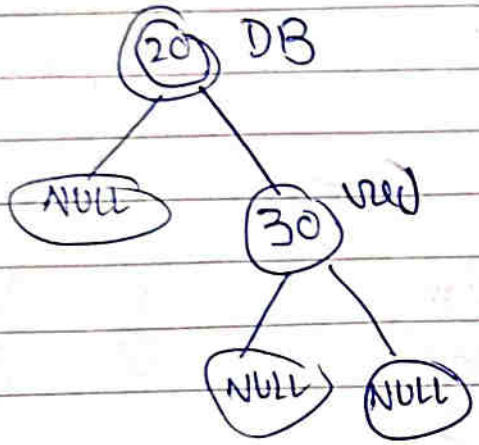


Q

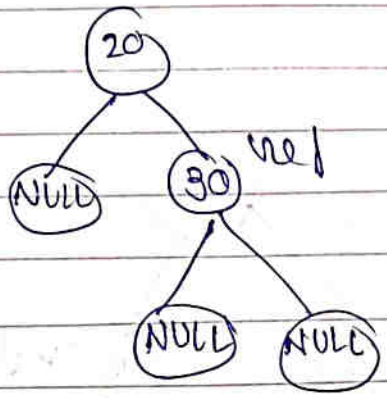


Delete 10 →
case-3

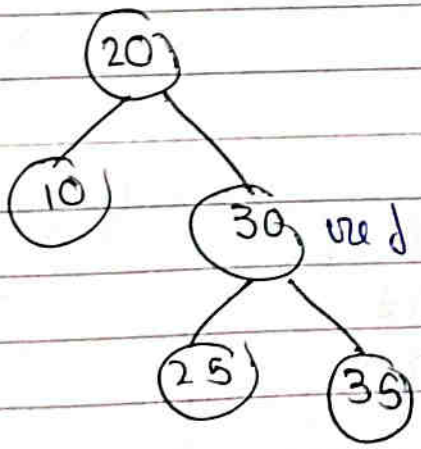




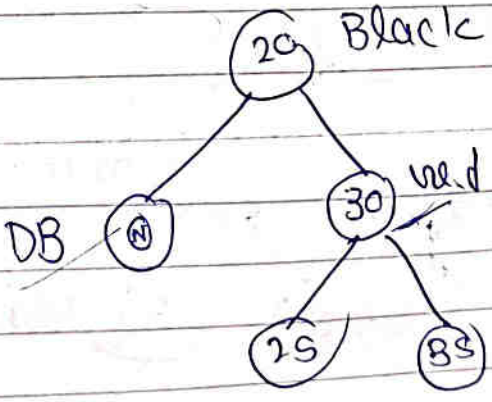
case-2 →

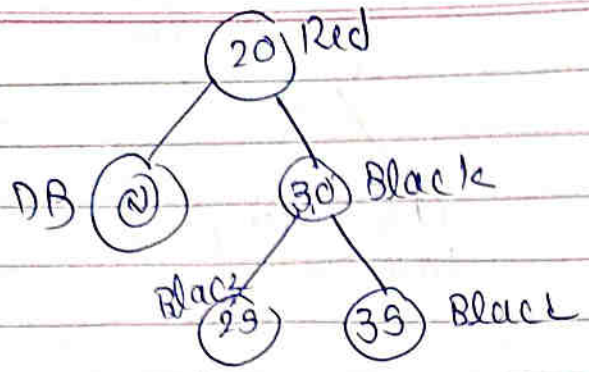


Q

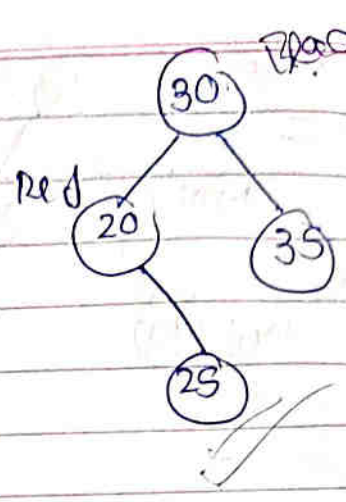


Delete 10 →
case-4

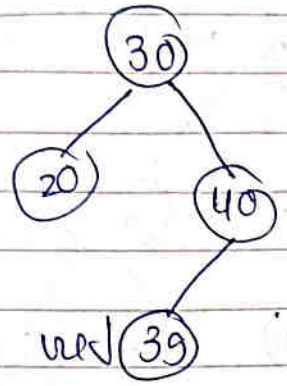




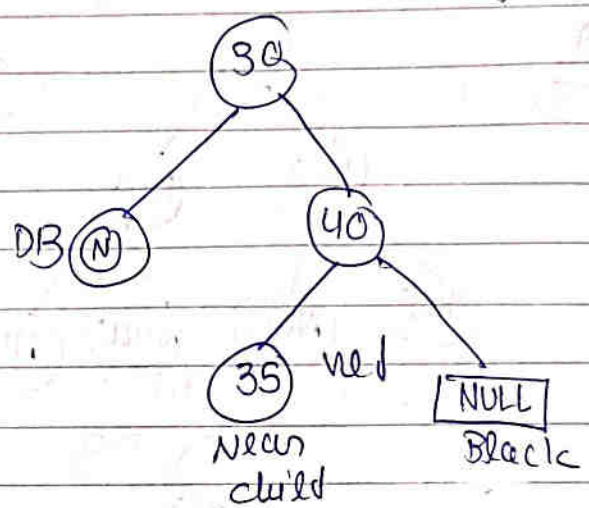
rotation → left rotation (Rotation toward DB)

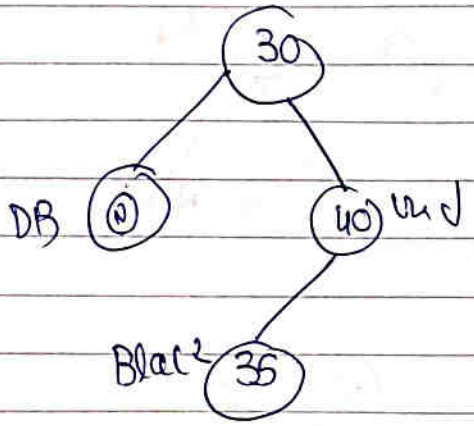


Q

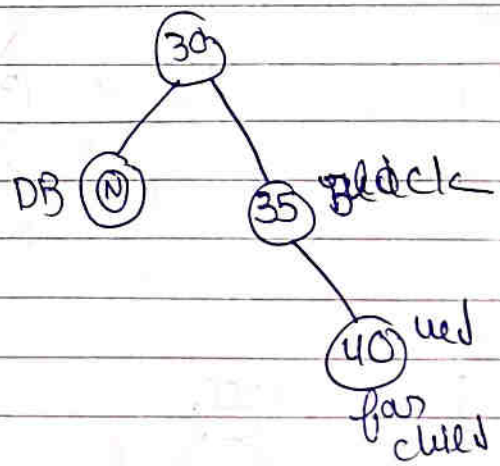


Dellete 20 → Case - 5

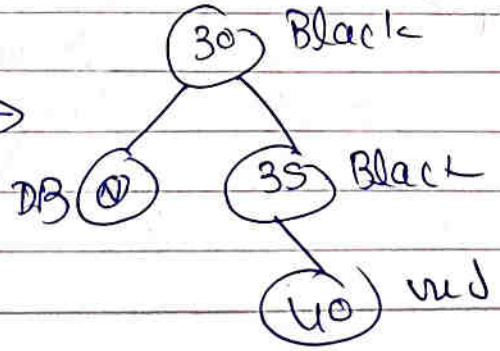




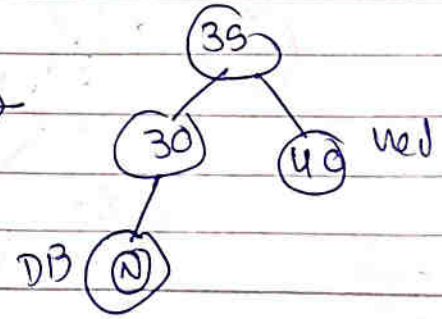
rotation opposite to DB →

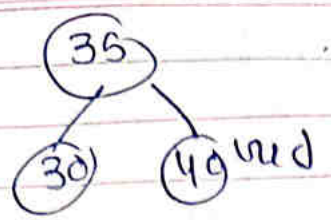


Case - 6 →

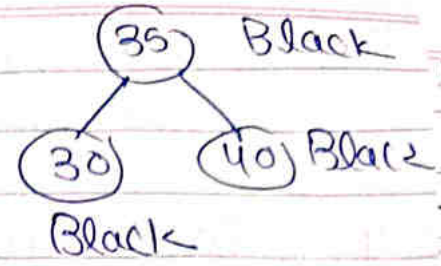


rotate toward to BB →



⇒ 

DB's sibling
for child
to red to
black



Q Explain B-Tree and its properties. Also write B-Tree deletion cases with example {AKTU 2023-2024}

⇒ A B-Tree is a self-balancing search tree designed for systems that read and write large blocks of data (like database or file systems)

⇒ It maintain sorted data and allows for efficient insertion, deletion and searches operations

Properties of B-Tree

1. Every Node Has a Fixed Range of Keys:-
→ Each node can store at least $\lceil \frac{m}{2} \rceil - 1$ keys and at most m-1 keys
→ m is order of the B-Tree

2. The root can have fewer keys (minimum 1 key).

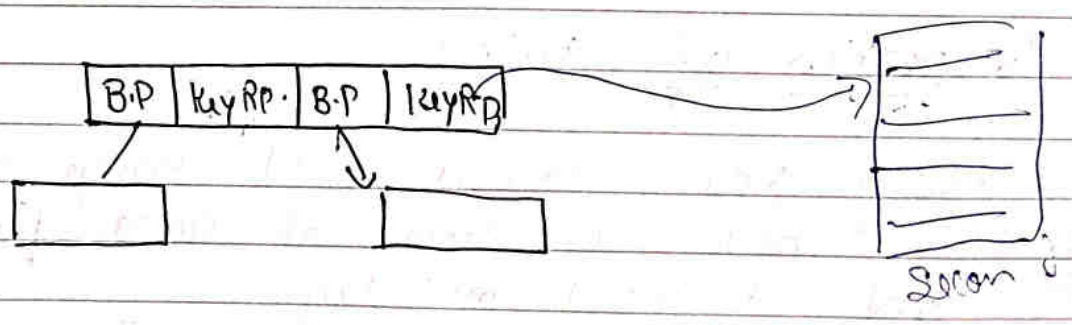
3. The keys inside a node are always kept in sorted order

4. Each key in a node acts like a divider for its children :-

→ keys in the left child are smaller than the first key.
→ keys in the middle child are between two keys
→ keys in the right child are larger than the last key.

5. All leaf node are at the same level.

6. Efficient Operations :- Searching, inserting and deleting keys take O(logn) time where n is total number of keys



Q 10, 20, 30, 40, 50, 60, 70, 80, 90, order = 4

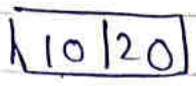
max key = 3

min = $\lceil \frac{4}{2} \rceil - 1$

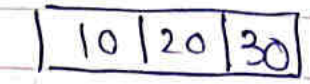
= 1

- Insert 10

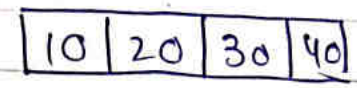
- Insert 20



- Insert 30

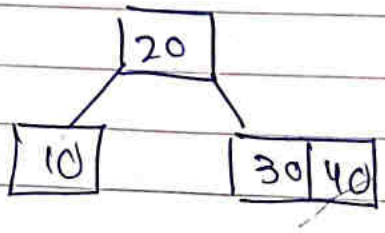


- Insert 40

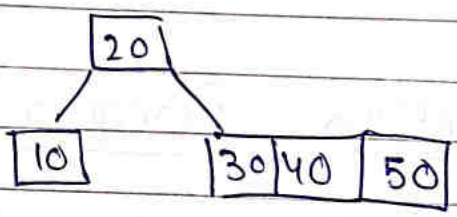


split

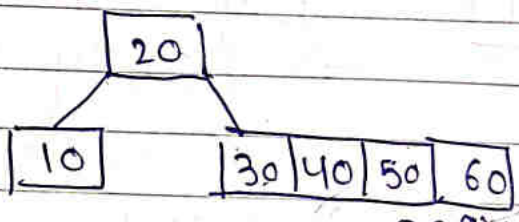
mid = $\frac{3}{2} = 1$



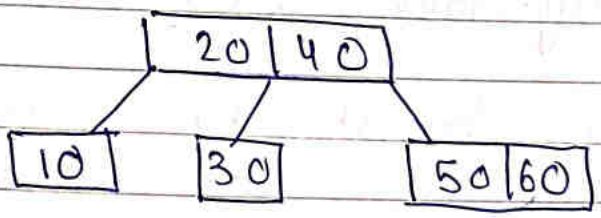
- Insert 50



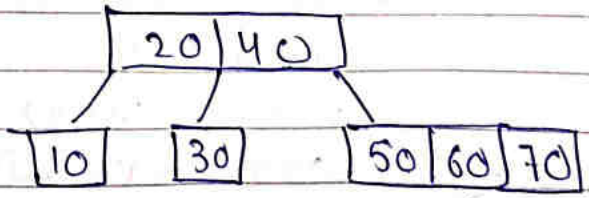
- Insert 60



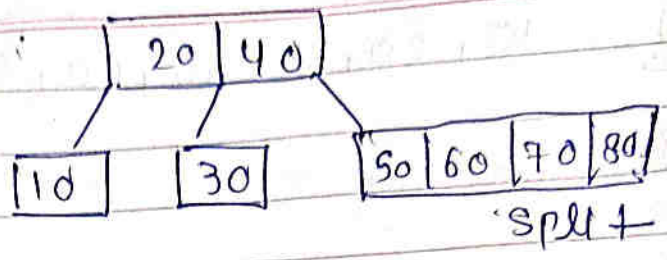
split

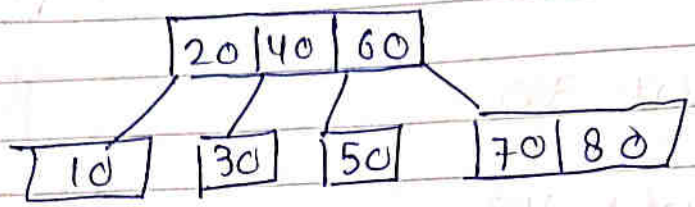


- Insert 70

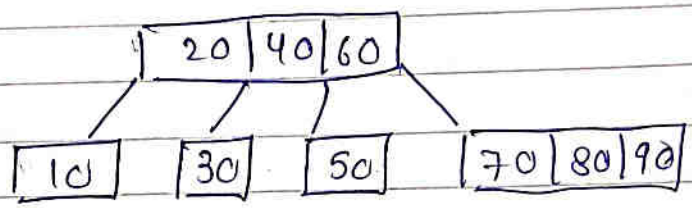


Insert 80





Insert 90



⇒ <u>Deletion Process</u>

<u>Case-1</u> Leaf Node has more than min key
→ simply delete the key

<u>Case-2</u> Lf. Node has minimum No. of keys

→ Borrow largest key from immediate left sibling (if left sibling has more key than minimum number) Transfer this key to the parent node, take the parent node key from the parent node and transfer it in the target node and replace it with the target key to be deleted

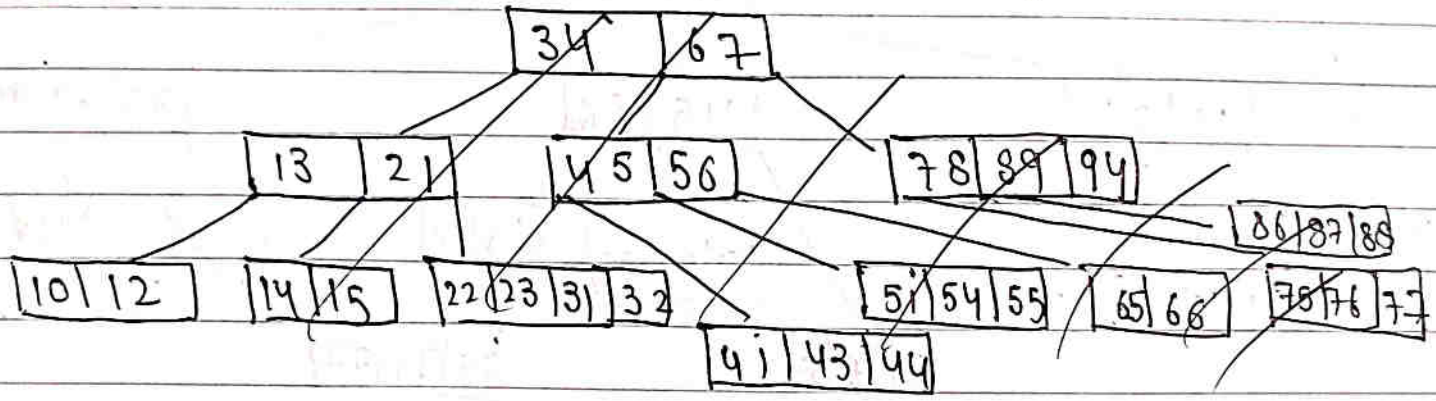
OR

→ Borrow smaller key from immediate right sibling (only if right sibling has more keys than minimum number) Transfer this keys to the parent node, take the parent node & transfer it in the target node and replace it with the target key to be deleted

OR

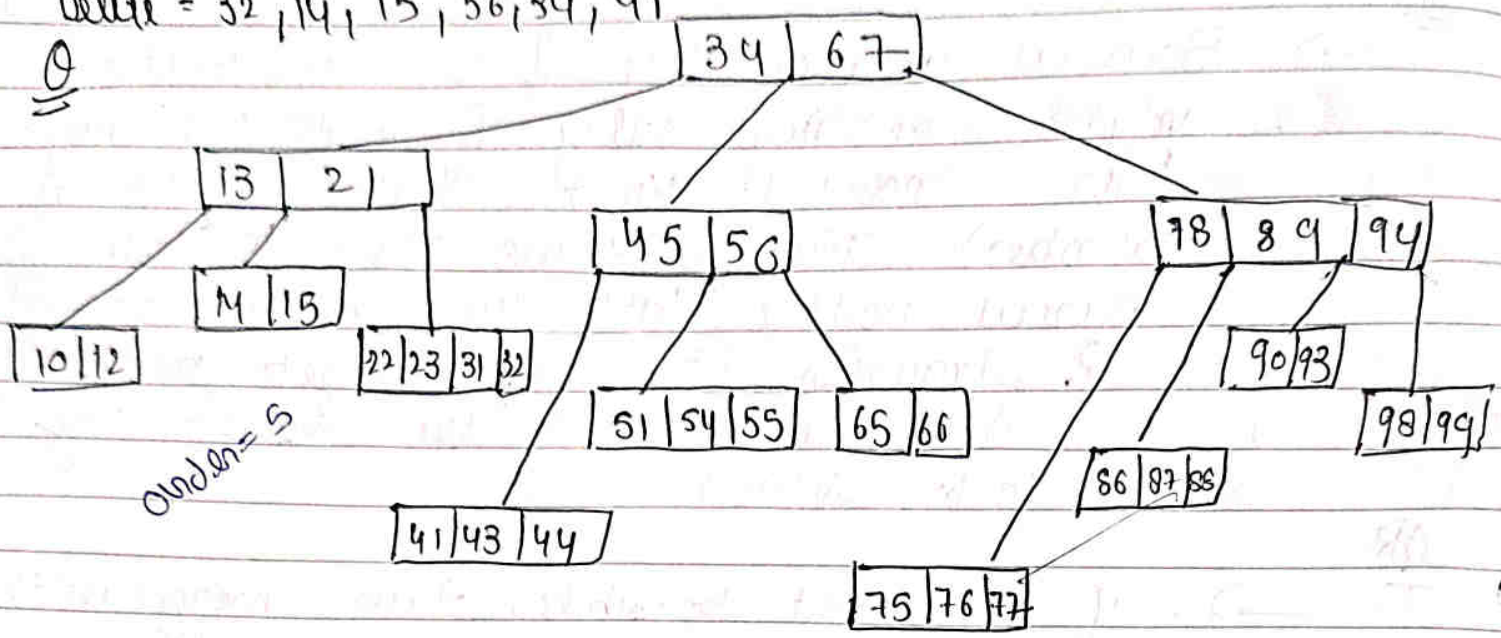
→ If both not possible, then merge with either left sibling or right sibling including the parent key. Then delete the target key

Q

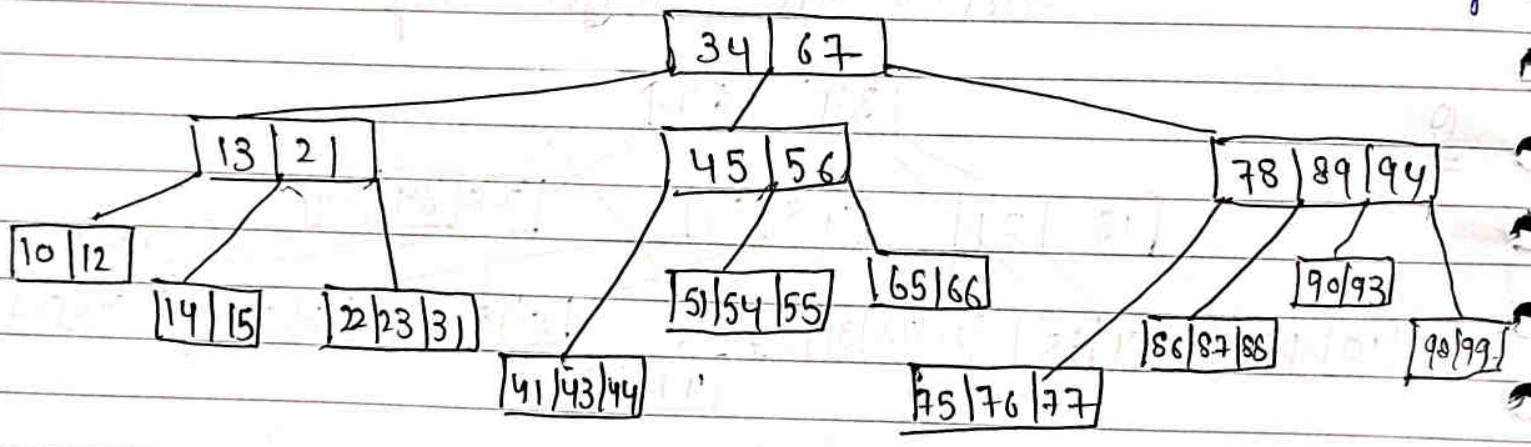


Delete = 32, 14, 15, 56, 34, 41

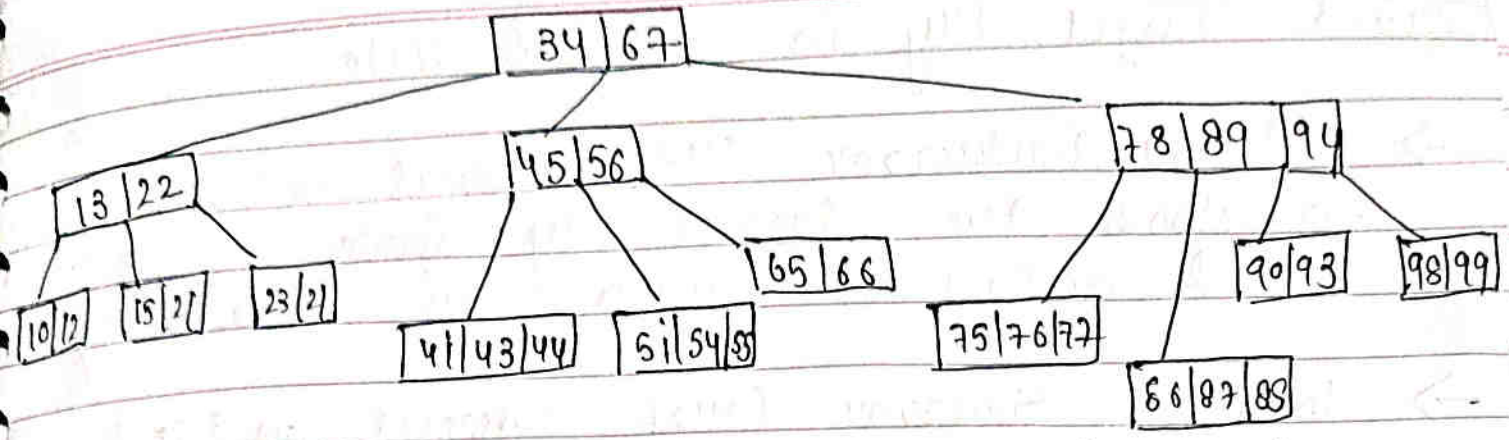
Q



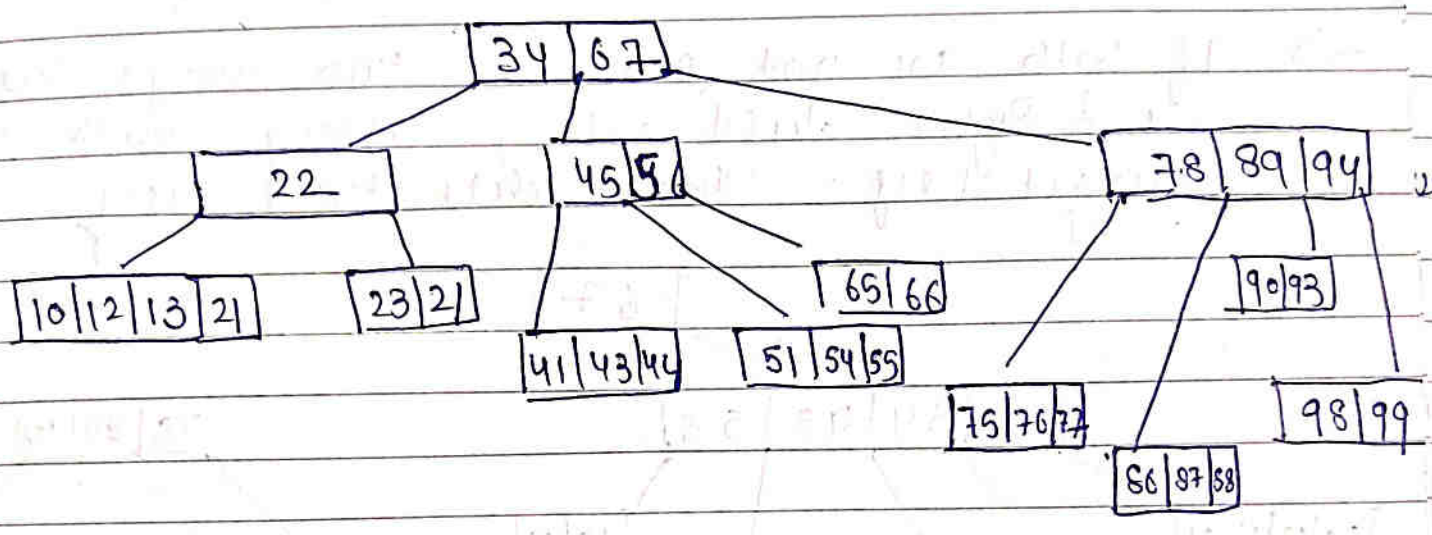
Delete = 32 → leave node has more than min no key

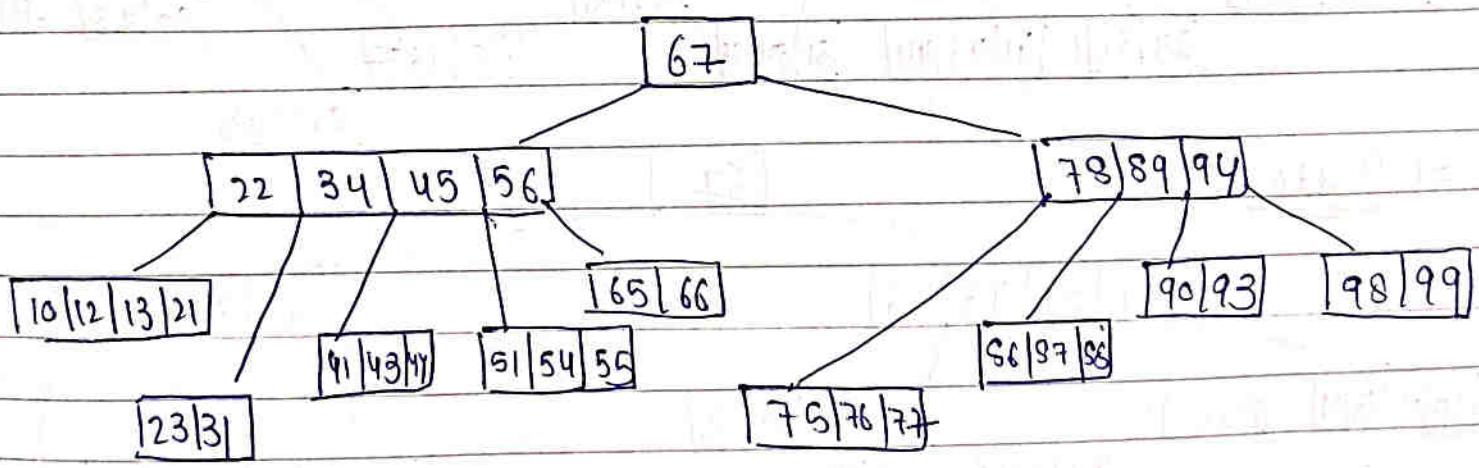


Delete = 14 → Borrow smallest key from the right sibling.



Delete=15 = merge with entire left sibling with parent node key



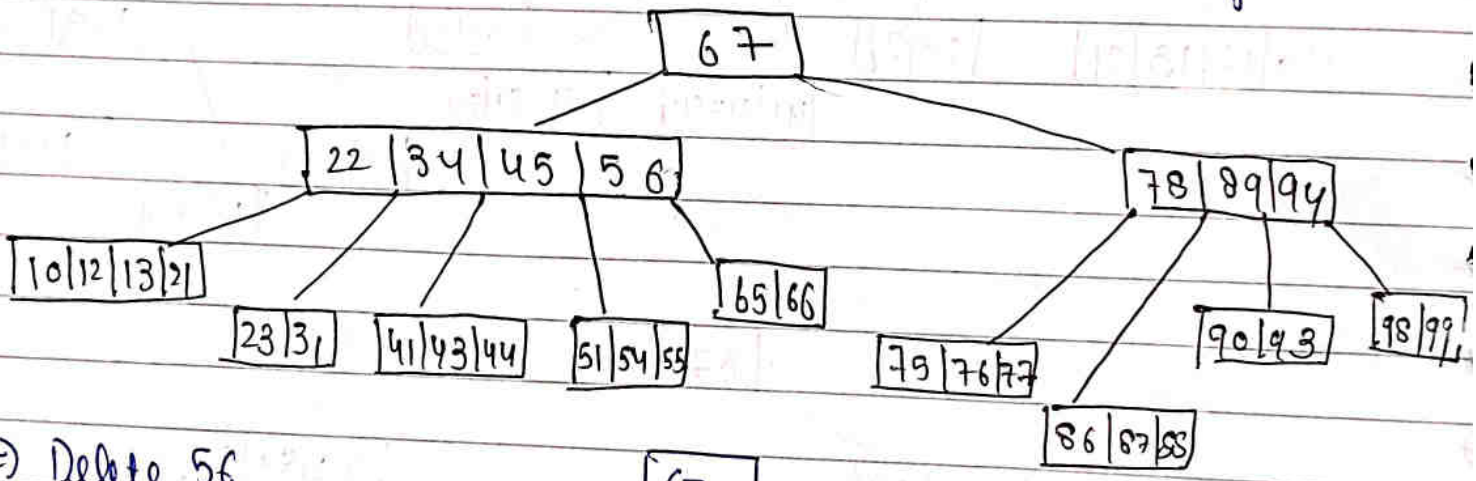


Case-3 Target key in Internal Node

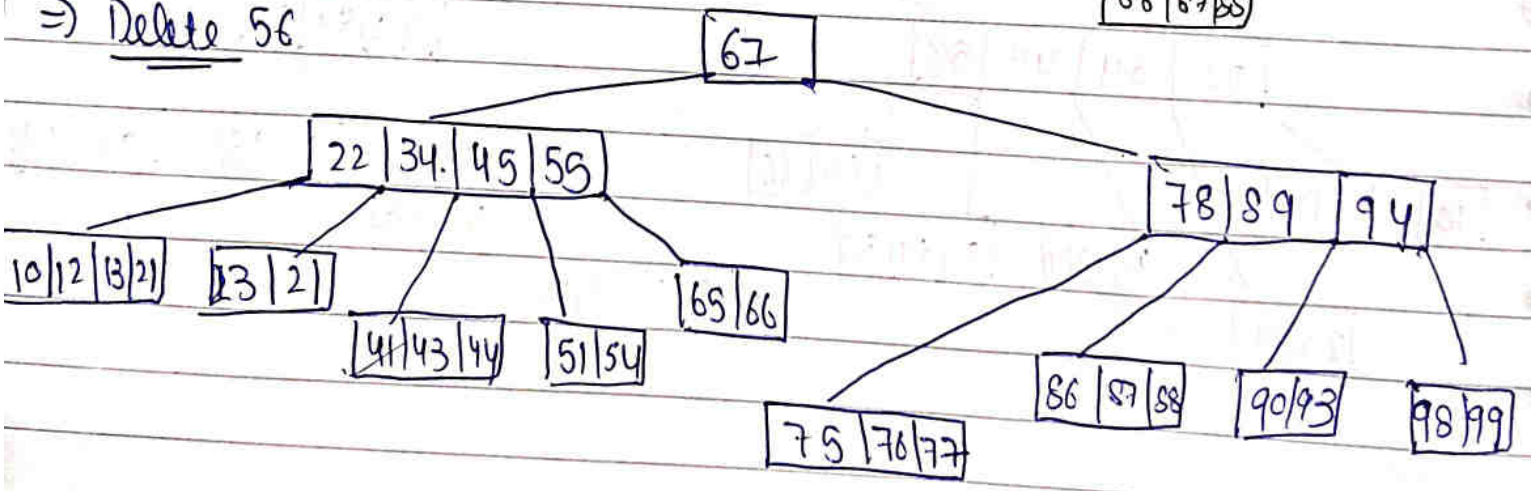
-> Inorder Predecessor (MAX element in LST)
  -> choose the largest key from the LST & replace it with the target key

-> Inorder Successor (MIN element in RST)
  choose the smallest key from the RST & replace it with the target key.

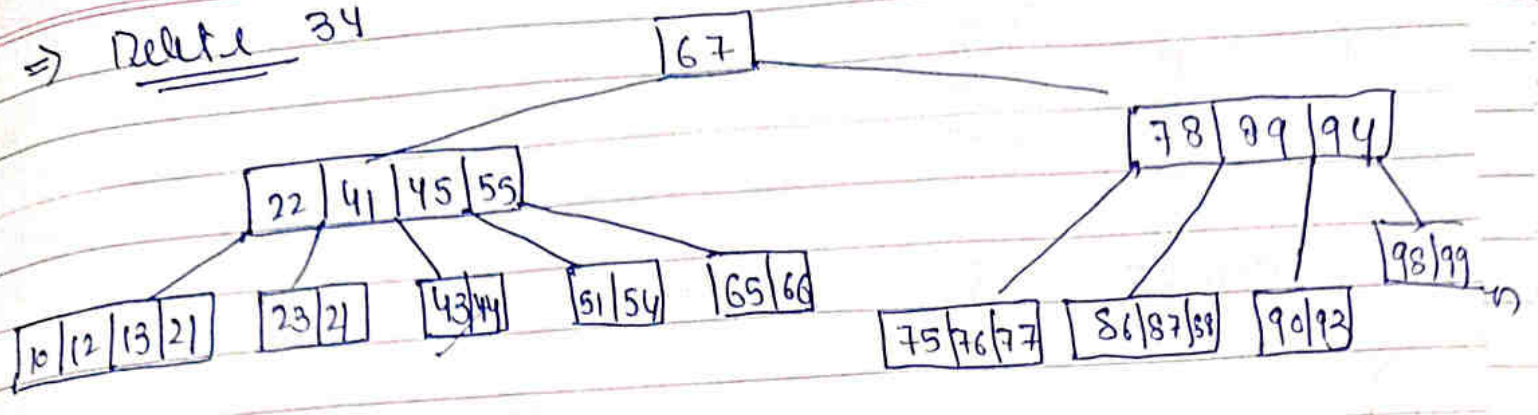
-> If both are not possible, then merge the left & Right child node, along with the target key. Then delete the key



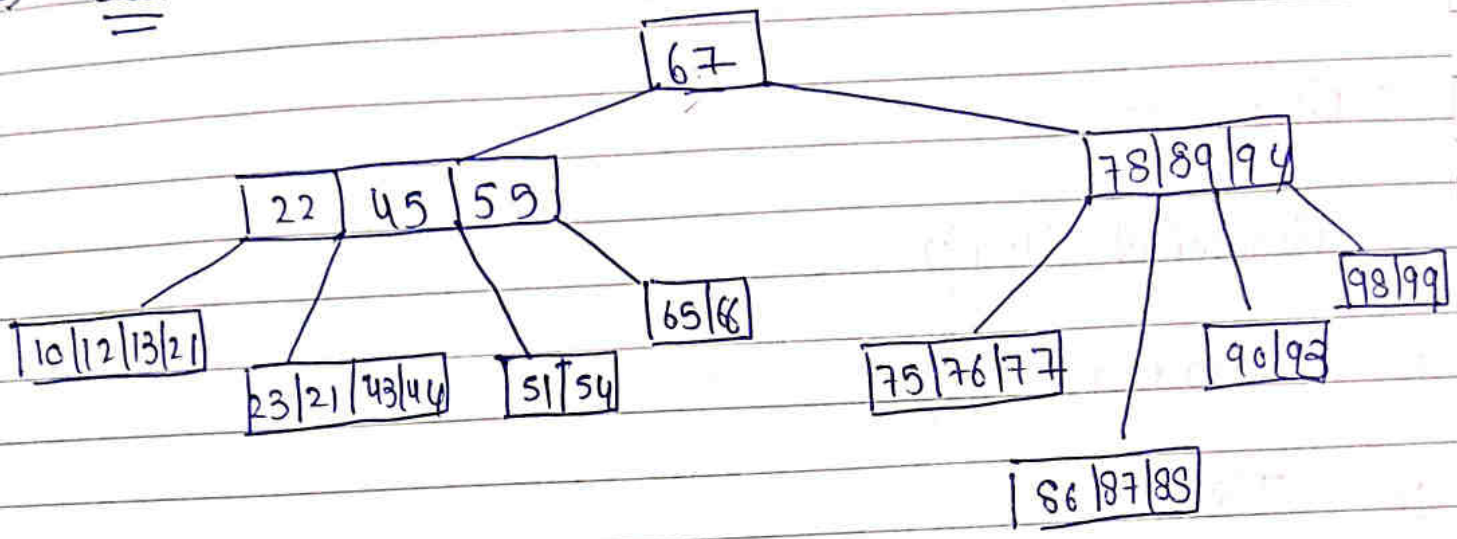
=> Delete 56



⇒ Delete 34



⇒ Delete 41



THANK YOU FOR WATCHING

LIKE SHARE SUBSCRIBE

SUBSCRIBE ~ @ MULTIATOMS

UNIT - 02

LECTURE - 03

DESIGN ANALYSIS OF ALGORITHM

Topics :-

- Binomial Heap
- Fibonacci Heap
- TRIE
- Skip list

SUBSCRIBE :- @MUNIATOMS & @MUNIATOMST

BINOMIAL HEAP

-> A binomial Heap can be defined as the collection of binomial trees that satisfies the heap properties i.e min-heap.

=> Min Heap in which each node has a value lesser than the value of its child.

=> Binomial Heap is used to implement a priority queue
=> It is an expansion of binary heap that give faster merge or union operation along with other operation provided by binary heap.

Properties of Binomial heap:-

① Every binomial tree in the heap must follow the min-heap property.

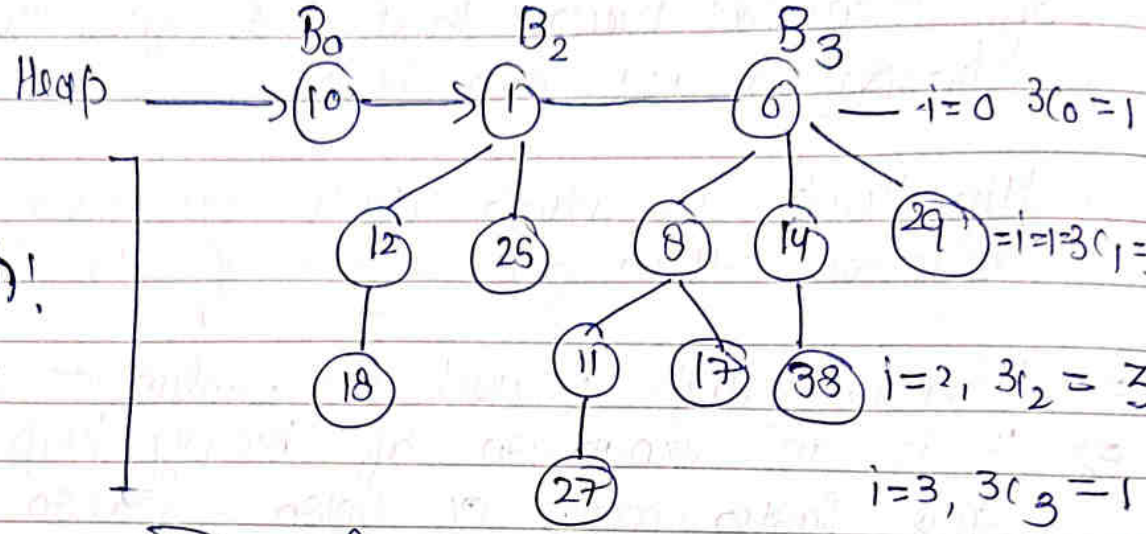
① For a non-negative integer K, there should be atleast one binomial tree in a heap where root has degree K.

① An n-node binomial Heap it consist of atmost It $\log_2 n$ binomial tree, here $\log_2$ is the binary algorithm

Examples No. of Node = 13 = binary Number $(1101)_2$

$B_3\ B_2\ B_1\ B_0$

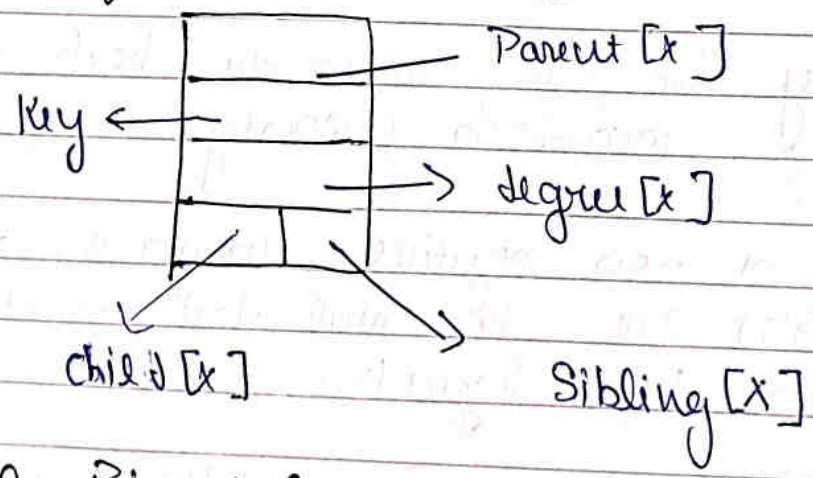
No. of Node $= 8+4+1 = 13$ node



$$^kC_i = \frac{k!}{i!(k-i)!}$$

Binomial min heap

Representation of binomial tree :-



Properties of Binomial tree :-

→ There are 2^k nodes

→ The height of the tree is k

→ There are exactly ^kC_i nodes at depth i for $i=0..k$

→ The root has degree k which is greater than that that of any other node

Operation of Binomial Heap :-

1. Creating a New Binomial Heap — O(1)

2. Insert :- Create a binomial heap with a single-node binomial tree for the new element — O(logn)

3. Find-Min (For a Min-Heap) :- find the smallest element in the heap — O(logn)

4. Delet-Min :— Remove the smallest element from the Heap. — O(logn)

5. Decrease-Key :- Reduce the value of a specific key in the Heap — O(logn)

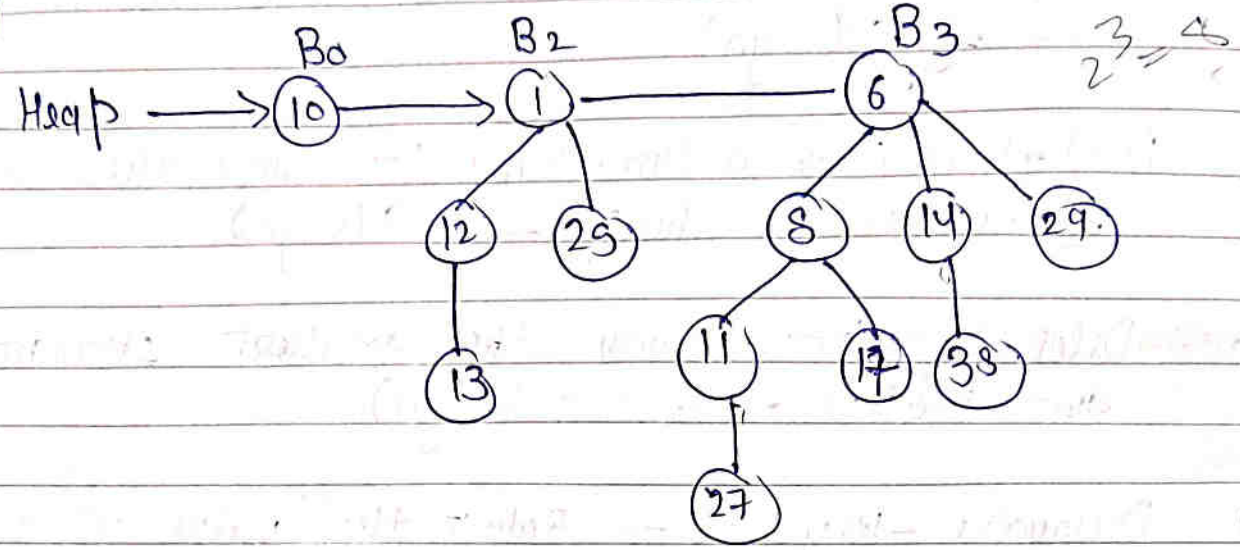
6. Delete — Remove an arbitrary element from the heap — O(logn)

7. Merge (union) :- Combine two binomial heaps into one. O(logn)

Q Prove all the properties of Binomial Tree
[AKTU 2023 - 2024]

Sol

Eg



Property 1 :- A binomial tree of order K has 2^K nodes

Order (0) (B_0) : 2^0 = 1 node
Order (2) (B_2) : 2^2 = 4 node
Order (3) (B_3) : 2^3 = 8 node

Property 2 :- A binomial tree of order K has height K.

→ Heigh of tree is the longest path from the root to a leaf.

- B_0 : 1 node → height - 0
- B_2 : 4 node → height - 2
- B_3 : 8 node → height - 3

Property-3 :- There are exactly $^{K}C_i$ node at depth for $i = 0, 1 \ldots K$ to find the node (at Depth).

B_3 = $i=0$ = $^{3}C_0$ = 1 node
= $i=1$ = $^{3}C_1$ = 3 node
= $i=2$ = $^{3}C_2$ = 3 node
= $i=3$ = $^{3}C_3$ = 1 node.

$$\boxed{^{K}C_i = \frac{K!}{i!(K-i)!}}$$

Property 4 :- The degree of the root is K.

order(3) — Degree of root (3).
order (2) — Degree of root (2)
order (0) — Degree of root (0)

Q. Explain and write an algorithm for union of two binomial Heap and write its time complexity. { AKTU 2022-2023 }

Sol The Union operation combines two binomial Heaps H_1 and H_2 into a single binomial Heap H.

=> The key idea is to merge their binomial tree while maintaining the heap properties

Algorithm :-

Step 1 Merge the root lists of binomial min Heap H_1 and H_2 into a single linked list that is stored in non-decreasing order of their degree

eg H_1 ① $B_0 \to B_1 \to B_2$
H_2 ② $B_0 \to B_2 \to B_4$
③ $B_0 - B_0 - B_1 - B_2 - B_2 - B_4$

Step 2 link the roots of equal degree until almost one root remains of each degree

④ $\underset{x}{B_0} - \underset{n-x}{B_0} - \underset{S-n-x}{B_1} - B_2 - B_2 - B_4$

a) if (degree[x] ≠ degree [next-x] or degree [x] = degree[next-x] = degree [sibling [next-x]])

⑦ $\underset{x}{B_2} - \underset{n-x}{B_2} - \underset{S-n-x}{B_2} - B_4$

b) if (degree [x] = degree [next-x] ≠ degree [sibling[next-x]])

i) if key (x) < key (next-x)
then make (next-x) as leftmost child of x

⑤ $\underset{x}{B_1} - \underset{n-x}{B_1} - \underset{S-n-x}{B_2} - B_2 - B_4$

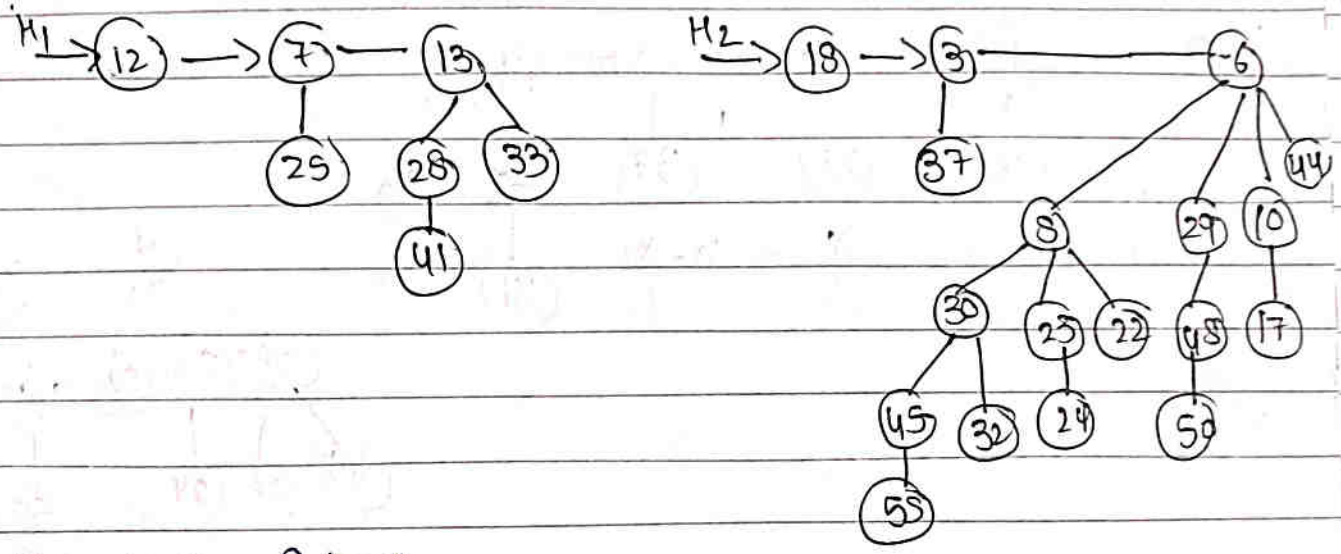
⑥ $\underset{x}{B_2} - \underset{n-x}{B_2} - \underset{S-n-x}{B_2} - B_4$

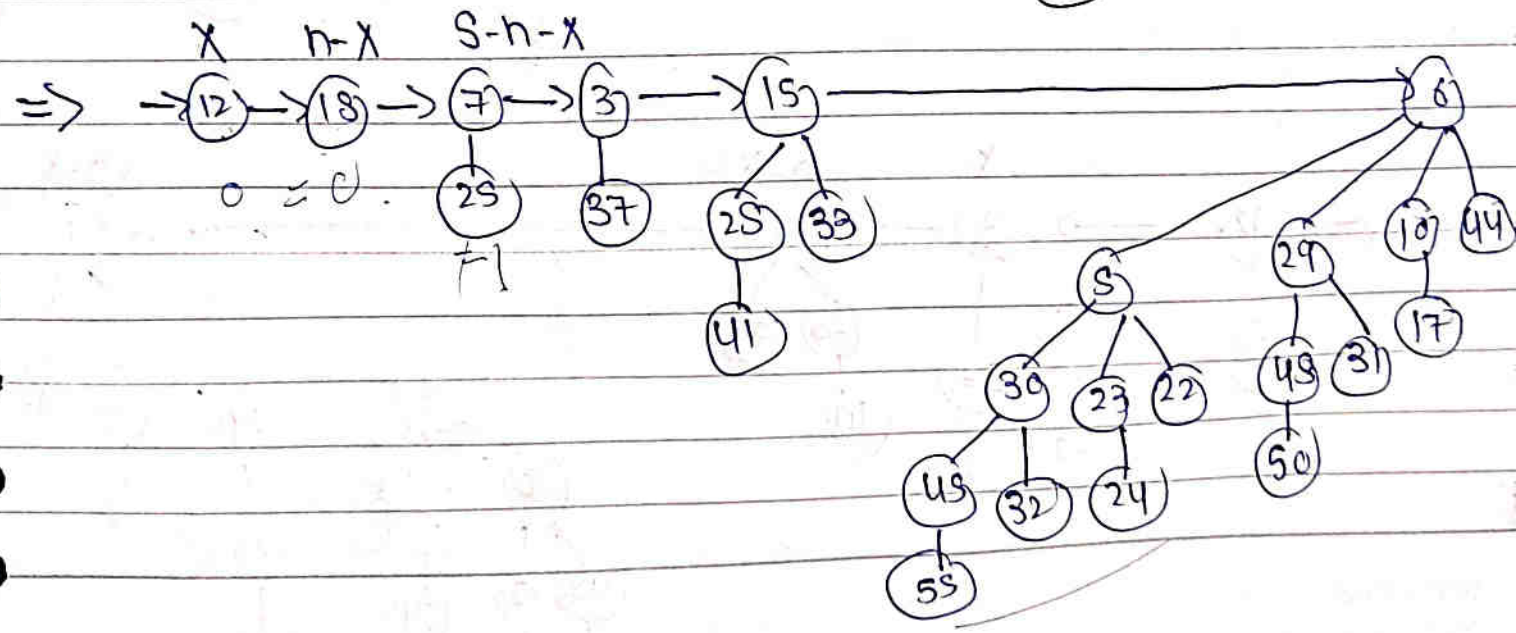
(ii) if key(x) > key (next-x)
then make X as leftmost child of (next-x)

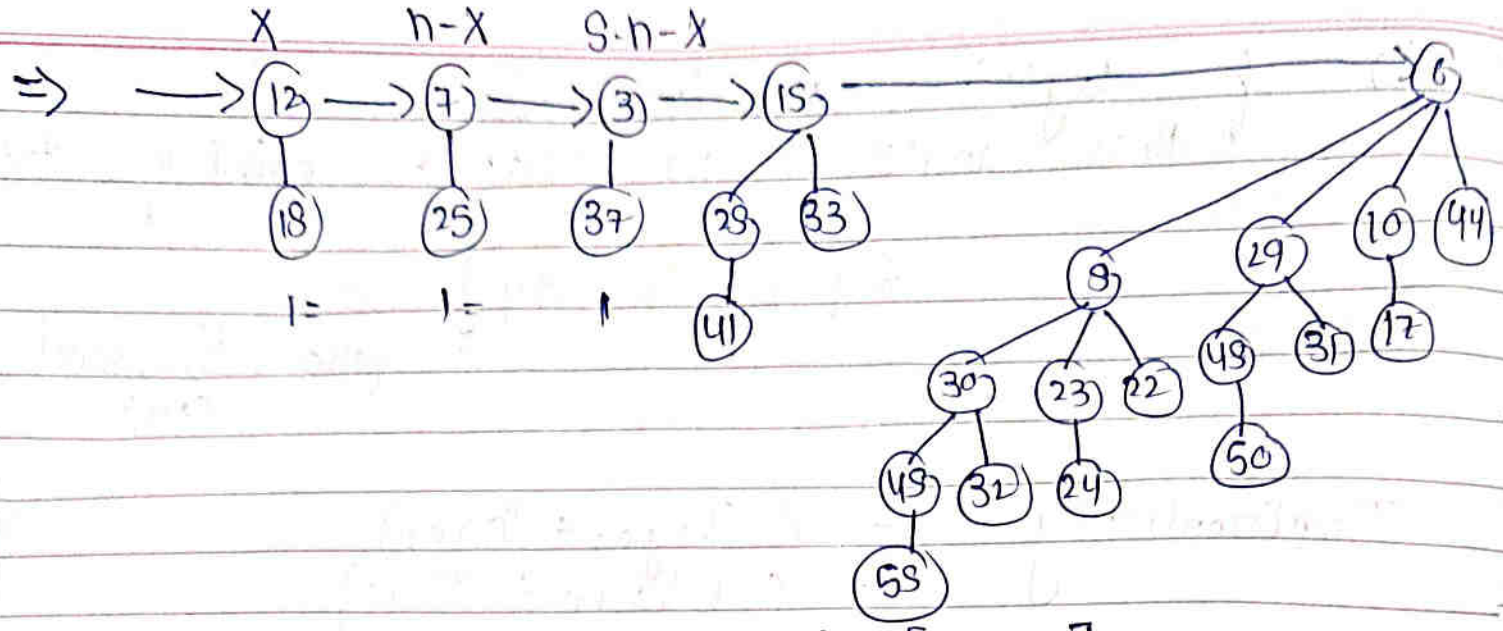
& $[B_2 - B_3 - B_4]$ final Binomial Heap.

Time Complexity $= O(\log n_1 + \log n_2)$
$= O(\log n + \log n)$
$= O(\log n)$

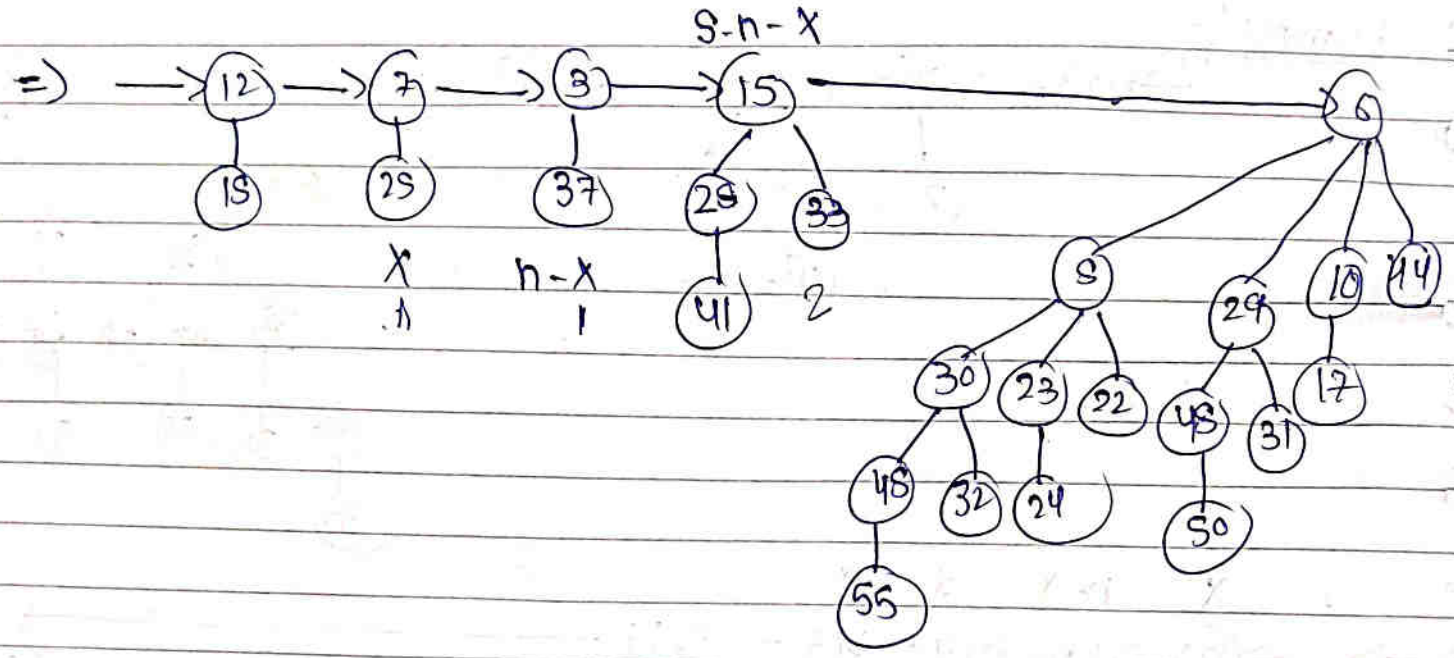
Example :-

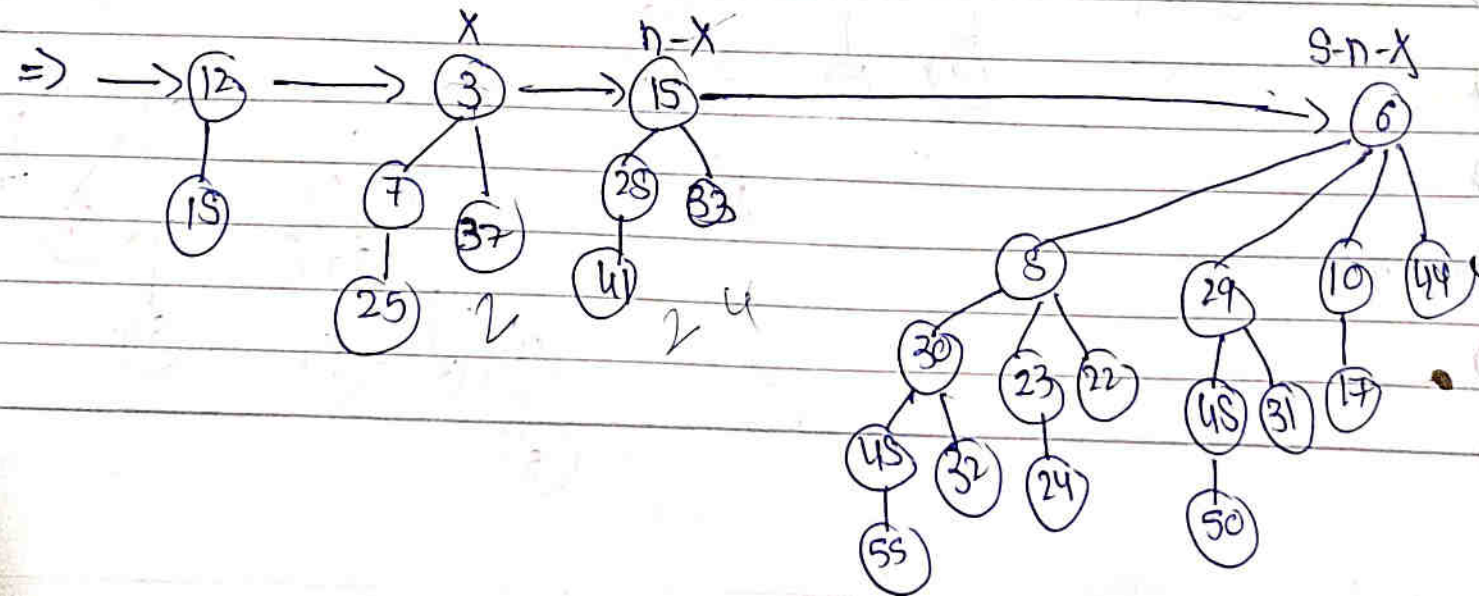


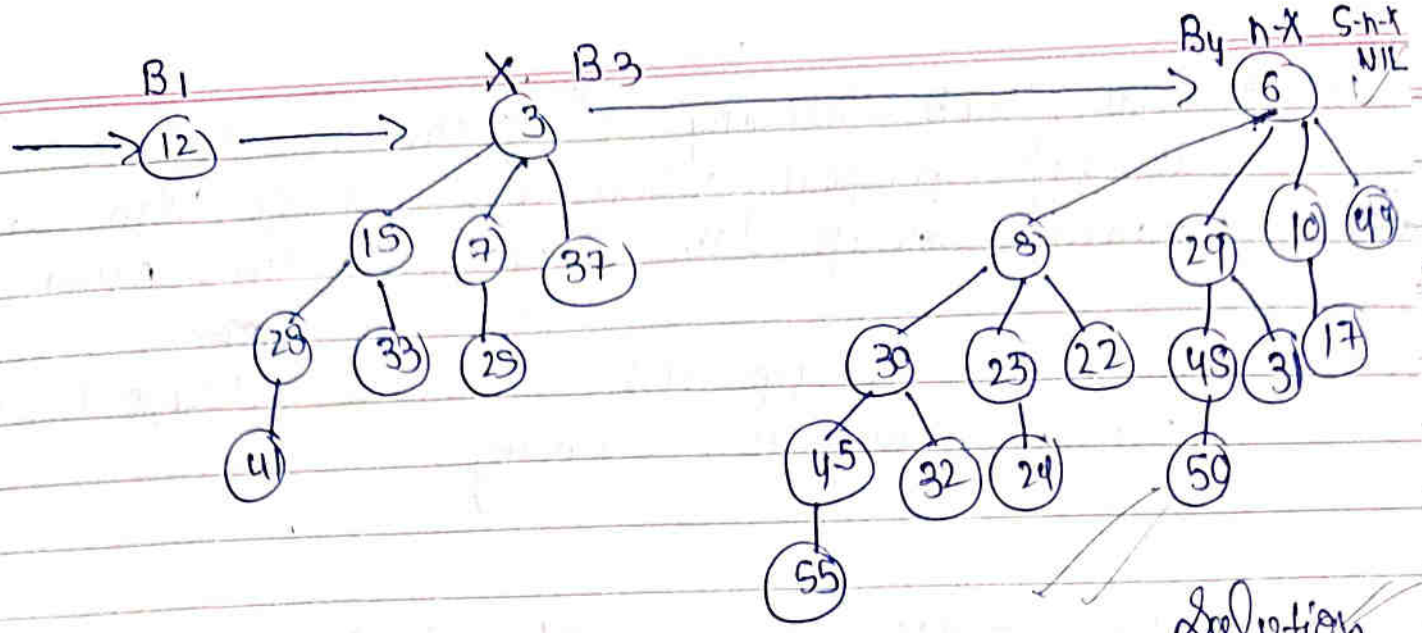




$$Deg[x] = Deg[n-x] = Deg[S-n-x]$$







Solution

Q Write down the algorithm for Decrease key Operation in Binomial Heap also write its time complexity

⇒ The decrease-key operation reduces the value of a specific key in a binomial heap to a smaller value while ensuring the heap properties are maintained.

Algorithm :-

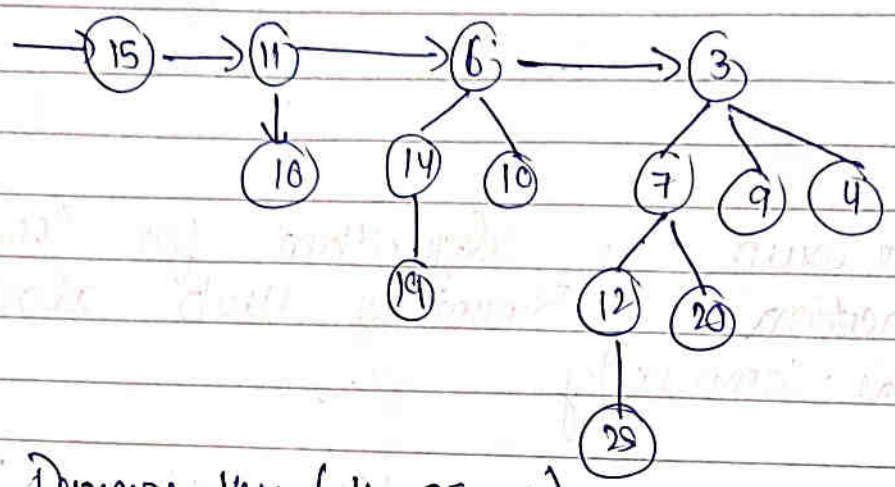
→ Input :→ • A binomial Heap H
• A pointer to the node where the key is to be decreased
• The new smaller key value newkey

→ Update the key → : • Replace the current key of the node with new key.
• Ensure newkey is smaller than the original key.

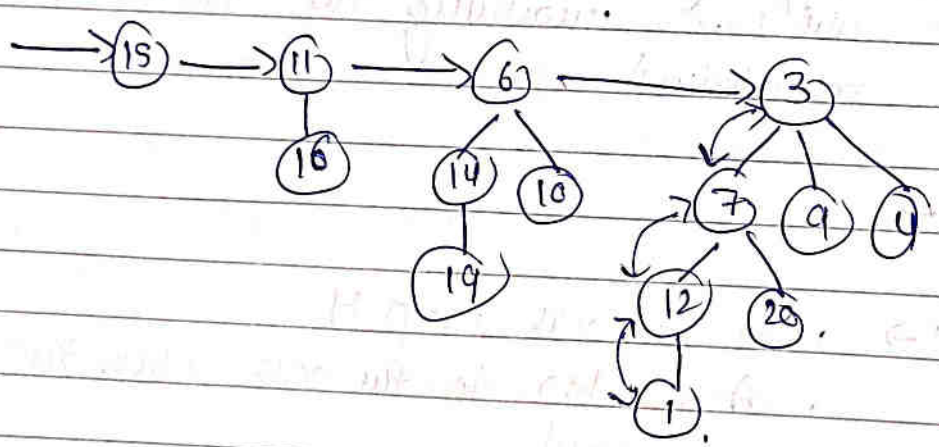
→ Restore Heap Property :- If the new key violates the heap property (i.e it is smaller than its parent) swap the key with the parent's key

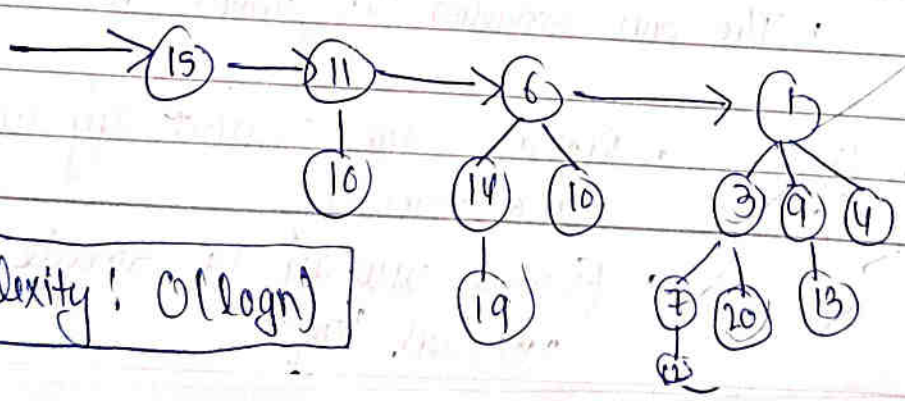
→ Output : The updated binomial heap H, which satisfies the heap property.

For eg :-



Heap Decrease key (H, 25, 1)





Time complexity : O(logn)

Q What are the various differences in Binomial and Fibonacci Heap? Explain {AKTU 2022-2023}

→ Binomial Heap: Simpler structure, but operations are slower compared to Fibonacci Heap.

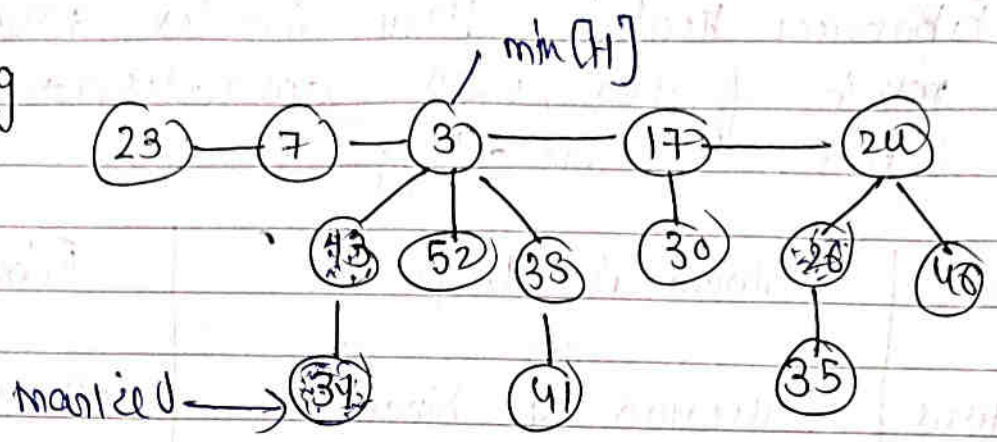
→ Fibonacci Heap: More complex structure, but much faster, used operations like insert and decrease-key

Aspect	Binomial Heap	Fibonacci Heap
Structure	Collection of binomial trees	Collection of trees with no specific structure
Tree shape	Trees are strict and follow binomial tree rules	Trees are more relaxed and can be unbalanced
Find-Min	O(logn): Traverse root node	O(1): Direct access
Insert	O(logn)	O(1)
Decrease-key	O(logn)	O(1)
Delete-Min operation	O(logn)	O(logn)
Merge Operation	O(logn)	O(1)
Efficiency	Less efficient	More efficient

Properties of Fibonacci Heap

→ The fibonacci Heap is a collection of unordered tree satisfying the heap property (min-heap or max-heap)

Eg



- H has a relaxed structure, allowing trees to be unbalanced for efficiency.
- The heap contain a pointer to the minimum node for find min-operation
- Operation like insert and merge are O(1), they don't require restructuring.
- Roots and sibling are circular doubly linked list
- Each node mark[x] either true or false newly created is unmarked
- Each node have it degree the number of children of node x have degree[x]

  Degree = no. of children

TRIE

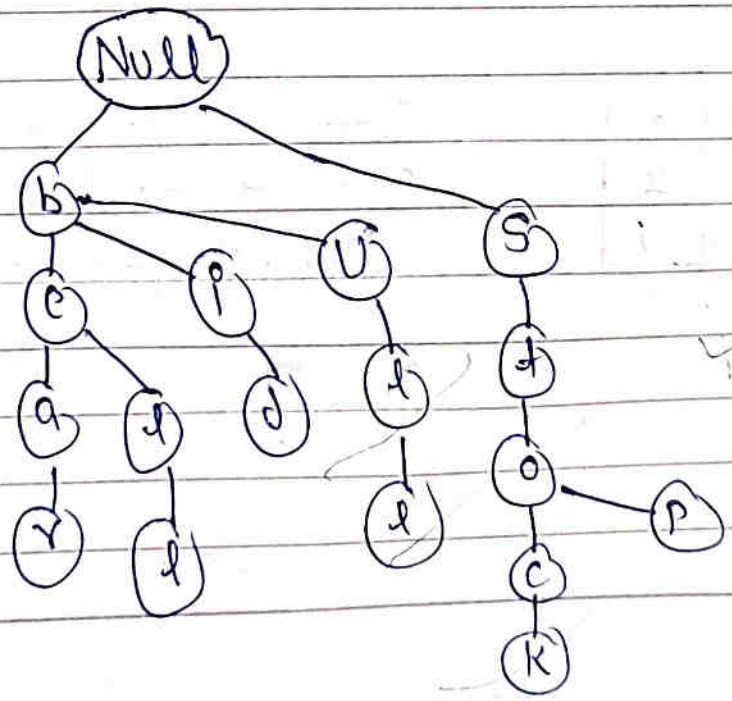
=> A Trie is a special tree-like data structure used to store and search strings efficiently

=> It is also called "prefix tree"

Properties

- Tries is multi-way tree
- Stores a set of string
- Every node consist at most 26 children
- Every node (except root) will store a letter in alphabet

Q Construct a Trie and compressed tries for set of string. S = {bear, bell, bid, bull, stock, stop}



SKIP LIST

⇒ A skip list is a data structure that allows fast search, insertion and deletion of elements in a sorted sequence.

⇒ It works by maintaining multiple levels of linked list, where each level skip over some elements to speed up operations

Structure of a skip list :-

⇒ **Levels :-** Higher levels act as "shortcuts" to skip over elements in the lower levels.

⇒ **Node :-** Each node contains a key and pointer to the next node in the same level or the levels below it.

eg level

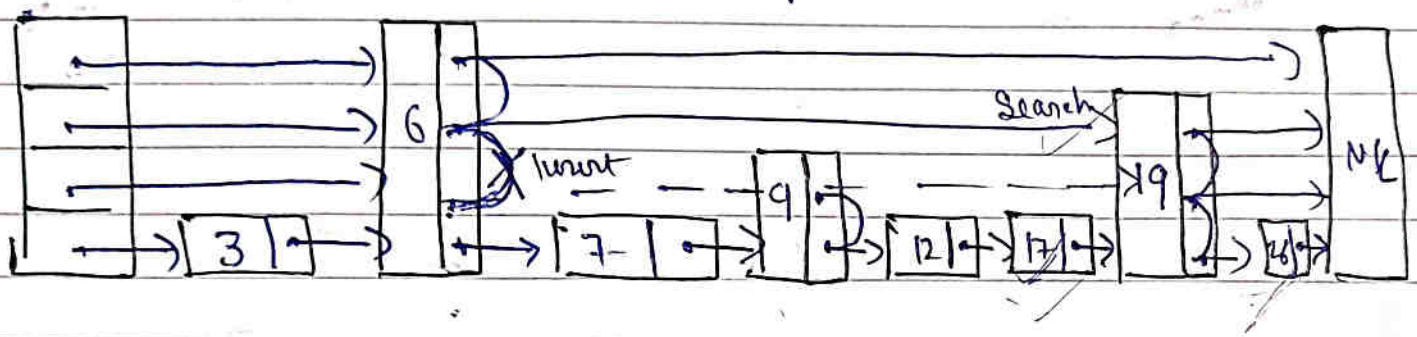
3
2
1

2 — — — — 8 — — — — 14
2 — — — 6 — — — 10 — — — 14
2 — — 4 — — 6 — — 8 — — 10 — — 12 — — 14

Algorithm for Search in Skip List :-

- Input : A skip list and the key to search for
- Start from the topmost, leftmost node.
- while the current level exists :-
  - Move forward until the next node's key is greater than the search key or null
  - Drop down to the next level if necessary
- At the lowest level, check if the key matches the search key.
- If found, return the key's position, otherwise return "not found"

Ex



UNIT-2 COMPLETED

THANK YOU FOR WATCHING

LIKE SHARE SUBSCRIBE

D.A.A

LECTURE - 01

UNIT - 3

© MULTI ATOMS PLUS

Topics :-

- Divide and Conquer
- Strassen's Matrix Multiplication
- Convex Hull Problem
- Sorting.

⇒ Divide and Conquer :-

⇒ Divide and Conquer Algorithm is a problem-solving technique used to solve problems by dividing the main problem into subproblems, solving them individually and then merging them to find solution to the original problem.

⇒ For ex:- Binary Search, Merge Sort etc.

Working of Divide and Conquer Algorithm :-

Divide and conquer Algorithm can be divided into three steps :-

1. Divide.
→ Break down the original problem into smaller subproblems
→ The goal is to divide the problem until no further division is possible.

2. Conquer :-
→ Solve each of the smaller subproblems individually.
→ The goal is to find solution for these subproblem independently

3. Merge :
→ Once the smaller subproblem are solved, we recursively combine their solutions to get the solution of larger problems

STRASSEN'S MATRIX MULTIPLICATION

- Strassen's algorithm developed by Volker Strassen in 1969.

- Strassen's Matrix Multiplication is an efficient algorithm for multiplying two square matrices.
- It reduces the number of multiplication compared to the standard approach, making it faster for large matrices $(O(n^3) \vee O(n^{2.81})$.

Algorithm

Given two $n \times n$ matrices A and B, Strassen's method divides them into four smaller $n/2 \times n/2$ matrices

1. Divide A and B into four submatrices
   - $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}_{2\times 2}$
   - $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}_{2\times 2}$

2. Compute 7 products using the submatrices
   - $P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
   - $P_2 = (A_{21} + A_{22}) B_{11}$
   - $P_3 = A_{11}(B_{12} - B_{22})$

- $P_4 = A_{22}(B_{21} - B_{11})$.
- $P_5 = (A_{11} + A_{12}) B_{22}$.
- $P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$.
- $P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$.

3. Combine these products to calculate the submatrices of the result C

- $C_{11} = P_1 + P_4 - P_5 + P_7$
- $C_{12} = P_3 + P_5$
- $C_{21} = P_2 + P_4$
- $C_{22} = P_1 - P_2 + P_3 + P_6$.

4. Reassemble C from $C_{11}, C_{12}, C_{21}, C_{22}$;

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

<u>Example</u> $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

<u>Sol</u> Divide A and B

$A_{11} = 1$, $A_{12} = 2$, $A_{21} = 3$, $A_{22} = 4$

$B_{11} = 5$, $B_{12} = 6$, $B_{21} = 7$, $B_{22} = 8$

⇒ Compute the 7 product :—

- $P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) = (1+4)(5+8) = 5 \times 13 = 65$
- $P_2 = (A_{21} + A_{22}) B_{11} = (3+4) \cdot 5 = 7 \times 5 = 35$
- $P_3 = A_{11}(B_{12} - B_{22}) = 1 \cdot (6-8) = 1 \times (-2) = -2$
- $P_4 = A_{22}(B_{21} - B_{11}) = 4 \cdot (7-5) = 4 \times 2 = 8$
- $P_5 = (A_{11} + A_{12}) B_{22} = (1+2) \cdot 8 = 3 \times 8 = 24$

- $P_6 = (A_{21} - A_{11})(B_{11} + B_{12}) = (3-1) \times (5+6) = 2 \times 11 = 22$
- $P_7 = (A_{12} - A_{22})(B_{21} + B_{22}) = (2-4) \times (7+8) = (-2) \times 15 = -30$

3. Compute $C_{11}, C_{12}, C_{21}, C_{22}$

$C_{11} = P_1 + P_4 - P_5 + P_7 = 65 + 8 - 24 - 30 = 19$
$C_{12} = P_3 + P_5 = -2 + 24 = 22$
$C_{21} = P_2 + P_4 = 35 + 8 = 43$
$C_{22} = P_1 - P_2 + P_3 + P_6 = 65 - 35 - 2 + 22 = 50$

4. Combine the results:

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Ans

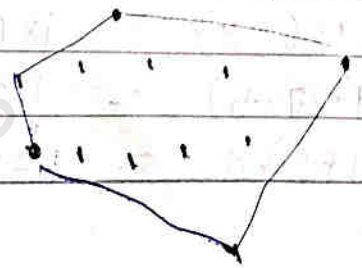
CONVEX HULL PROBLEM :-

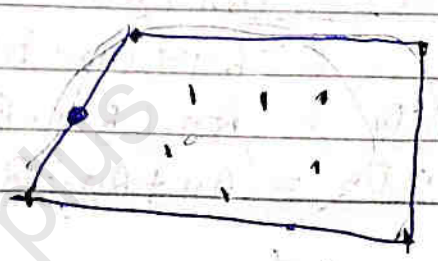
⇒ Imagine you have some points (dots) on a piece of paper.

⇒ you want to draw the smallest possible shapes that goes around all the points.

⇒ The problem of finding the smallest polygon, such that all the points of set, are either on the boundary, or inside is known as the convex hull problem.

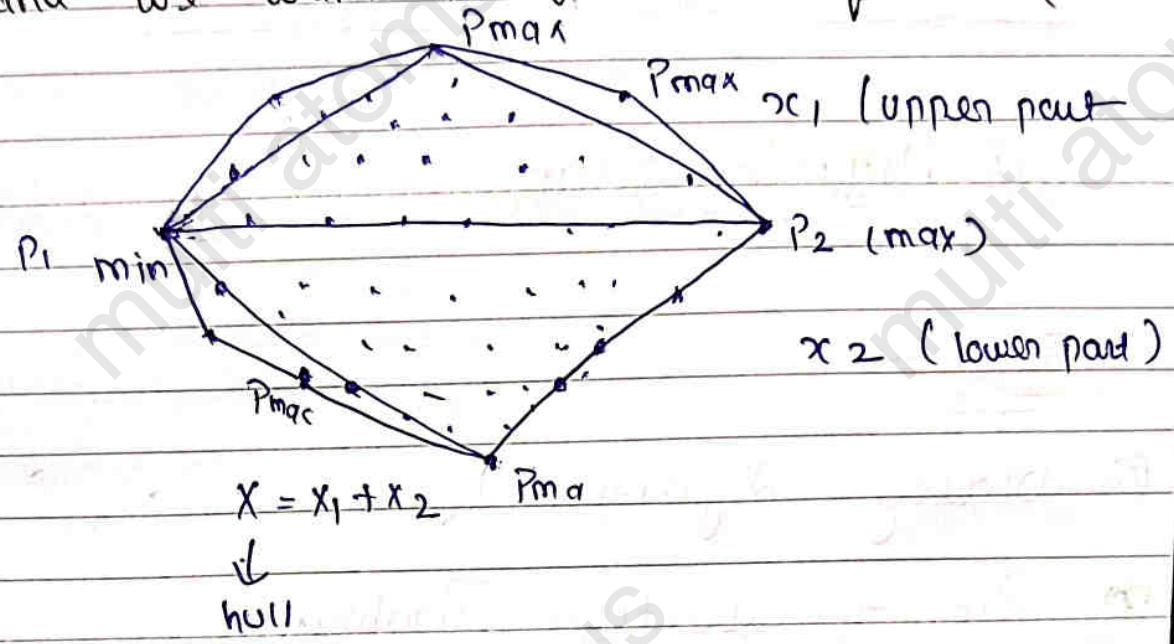
points





Convex Hull is a polyon that encloses all of the points

Q Give a set X at point $P_1(x_m, y_m)$, $P_2(x_m, y_m)$ is plane we want to find convex ~~poi~~ hull?



Sorting :-

Already done in Unit - 1

THANK YOU FOR WATCHING

LIKE SHARE SUBSCRIBE

D.A.A

UNIT-3 LECTURE-02

© MULTIATOMS PLUS

Topics:-

- Greedy Algorithm
- Activity Selection Problem
- Knapsack Problem {AKTU 2022-2023}
- 0/1 Knapsack Problem

GREEDY ALGORITHM

⇒ A greedy algorithm is a problem-solving technique that makes the best choice at each step, based on the current situation, without considering the overall problem.

- Feasible Solution :- A solution which satisfy constraints given with problem is known as feasible solution

- Optimal Solution :- A feasible solution is that solution that give either maximize profile or minimum cost a given objective function is called as optimal solution.

Psudocode for Greedy Algorithm

```
Greedy (a,n)
{
        // a[1:n] contains n inputs
    Solution = 0;
    for i=1 to n do
    {
        x = select(a);
    if ( feasible (solution x) then
                Solution = Solution + x }
            return Solution;
    }
```

Activity Selection Problem :-

The Activity Selection Problem is a mathematical optimization problem where the objective is to select the maximum number of activities from a set, such that no two activities overlap.

- Each activity is defined by a start time and an end time.

- Our goal is to select maximum number of non-conflicting activities that can be performed by a person

  of fin >= new start

- Any two activities are said to be non-conflicting if starting time of one activity is greater than or equal to the finishing time of other activity.

- A greedy algorithm is used to solve the problem efficiently by prioritizing activities that finish the earliest.

Q Activity Selection problem : The starting and finish time of all activities are given as follows: (2,3) (8,12) (12,14) (3,5) (0,6) (1,4) (6,10) (5,7) (3,8) (5,9) (8,11).

Sol Step-1 Sort the finishing time in increasing order

Sorted Activity	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Starting time	2	1	3	0	5	3	5	6	8	8	12
finish time	3	4	5	6	7	8	9	10	11	12	14

- We select first activity a_1 (2,3)
- check of activity a_2 starting time of a_2 $\geq$ finish time of a_1
  a_2 is not selected
- check for activity a_3 (3,5)
  starting time of a_3 $\geq$ finish time of a_1
  $\therefore$ a_3 is selected
- check for activity a_4
  starting time of a_4 $\geq$ finish time of a_3
  $\therefore$ a_4 is not selected
- check for activity a_5
  starting time of a_5 $\geq$ finish time of a_3
  $\therefore$ a_5 is selected
- check for activity a_6
  starting time of a_6 $\geq$ finish time of a_5
  $\therefore$ a_6 is not selected

- check for activity a_7
  starting time of $a_7 \geq$ finish time of a_3
  $\therefore a_7$ is not selected

- check for activity a_8
  starting time of $a_8 \geq$ finish time of a_5
  $\therefore a_8$ is not selected

- check for activity a_9
  starting time of $a_9 \geq$ finish time of a_5
  $\therefore a_9$ is selected

- check for activity a_{10}
  starting time of $a_{10} \geq$ finish time of a_9
  $\therefore a_{10}$ is not selected

- check for activity a_{11}
  starting time of $a_{11} \geq$ finish time of a_9
  $\therefore a_{11}$ is selected
  $[2 \geq 1]$

Therefore selected activities are:

a_1 : (2,3)
a_3 : (3,5)
a_5 : (5,7)
a_9 : (8,11)
a_{11} : (12,14)

Q What is Knapsack problem? Solve fractional Knapsack problem using greedy programming for the following four items with their weight $w = \{3, 5, 9, 6\}$ and values $P = \{45, 30, 45, 10\}$ with Knapsack capacity is 16

Sol Here Knapsack is like a container or a bag.

The Knapsack problem is a popular optimization problem in which you are given a set of items, each with a weight and a value.

-> The goal is to choose a subset of items to maximize the total value while staying within a given weight limit (capacity) of the Knapsack.

=> There are two types of Knapsack problems :-

- 0/1 Knapsack problem
- Fractional Knapsack Problem

=> 0/1 Knapsack problem :-

- The 0/1 Knapsack problem means that the item are either completely or no items are filled in a Knapsack.
- The 0/1 Knapsack problem is solved by the dynamic programming.

Fractional Knapsack Problem :-

- The Knapsack problem means that we can divide the item
- Solved using a greedy algorithm

Num :-

Objects :-	1	2	3	4
Profit (P) :-	3	5	9	5
Weight (w) :-				

$\boxed{N = 4 \quad W = 16}$

Objects :-	1	2	3	4
Profit (P) :-	45	30	45	10
Weight (w) :-	3	5	9	5
P_i / w_i :-	15	6	5	2

Method-1 Select object with maximum Profit

Objects	Profit (P)	Weight (w)	Remaining weight
1	45	3	$16 - 3 = 13$
3	45	9	$13 - 9 = 4$
2	$4 \times \frac{30}{5} \Rightarrow 24$	4	$4 - 4 = 0$

Total Profit $= 45 + 45 + 24$
$\Rightarrow 114$

Method-2 Select object with minimum weight

objects	Profit (P)	weight (w)	Remaining weight
1	45	3	16-3 = 13
2	30	5	13-5 = 8
4	10	5	8-5 = 3
3	$3 \times \frac{45}{9} = 15$	3	3-3 = 0

Total Profit = 45 + 30 + 10 + 15
= 75 + 25
= 100

Method-3 Select object with maximum P_i/w_i

objects :	1	2	3	4
Profit (P) :	45	30	45	10
Weight (w) :	3	5	9	5
P_i/w_i :	15	6	5	2

object	Profit (P)	weight (w)	Remaining weight
1	45	3	16-3 = 13
2	30	5	13-5 = 8
3	$8 \times \frac{45}{9} = 40$	8	8-8 = 0

Total Profit = 45 + 30 + 40
⇒ 115

optimum solution

0/1 Knapsack Problem :-

- 0/1 knapsack problem is based on Dynamic programming

- The most common problem being solved is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item Zero or One

- Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\}$$

Vacant

- Here x_i represent the number of instances of item i to include in the knapsack.

- The problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weight is less than or equal to the knapsacks capacity

Time Complexity :- $O(N * W)$, where N is the number of elements and W is capacity

formula

$$\boxed{m[i,w] = \max(m[i-1,w],\ m[i-1,\ w-w[i]]+P[i])}$$

Q Weights = {3, 4, 5, 6} $w = 8$

Profit = {2, 3, 4, [illegible]} $n = 4$

P_i	w_i	m \ w →	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
2	3	1	0	0	0	2	2	2	2	2	2
3	4	2	0	0	0	2	3	3	3	5	5
4	5	3	0	0	0	2	3	4	4	5	6
1	6	4	0	0	0	2	3	4	4	5	$\boxed{6}$

maximum profit

$6-4 \Rightarrow 2$

$2-2 \Rightarrow 0$

$X_1 = \{1, 0, 1, 0\}$

$m[4,7] = \max(m[3,7],\ m[3,7-6]+1)$

$\max(5,\ 0+1)$

$\max(5,1) \Rightarrow 5$

$m[3,7] = \max(m[2,7],\ m[2,7-5]+4)$

$\max(5,\ 0+4)$

$\max(5,4) \Rightarrow 5$

$m[3,8] = \max(m[2,8],\ m[2,8-5]+4)$

$\Rightarrow \max(5,\ 2+4)$

$= \max(5,6) = 6$

UNIT - 3

D.A.A

LECTURE - 3

© MULTIATOMS PLUS

MULTI ATOMS

Topics :-

- Spanning Tree
- Kruskal algorithm
- Prims algorithm
- Single Source Sortest Path
- Dijkastra Algorithm
- Bellman Ford Algorithm

Q. Write and explain the Kruskal algorithm to find the Minimum Spanning Tree of a graph with suitable example.

Sol **Spanning Tree :-**

A spanning tree is a subgraph of a connected (undirect) graph that includes vertices of the graph connected with the minimum number of edges. and contains no cycles

- **Minimum spanning Tree :-**

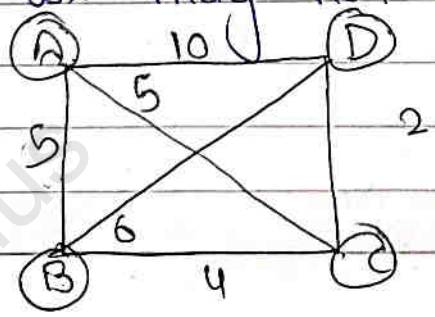
- A M.spanning tree is a spanning tree of a weighted where the sum of the weight of its edges is the smallest possible.

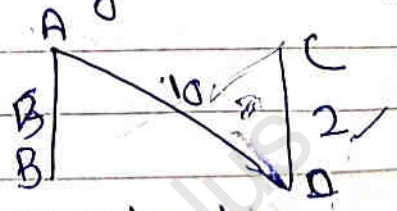
- It connects all vertices with the least total cost.

Points to remember

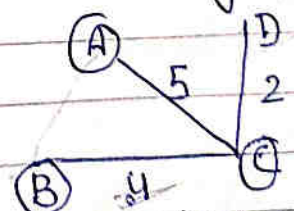
- We must use only edges with the graph
- We must use exactly n-1 edges
- We may not use edges that produce a cycle

Q





Spanning tree



MST

Kruskal's Algorithm

⇒ Kruskal algorithm is a minimum spanning tree algorithm that takes a graph as input and find the subset of the edges of that graph.

⇒ Algo :-

Step1: Select an edge of a graph, such that it has the smallest cost associated with it.

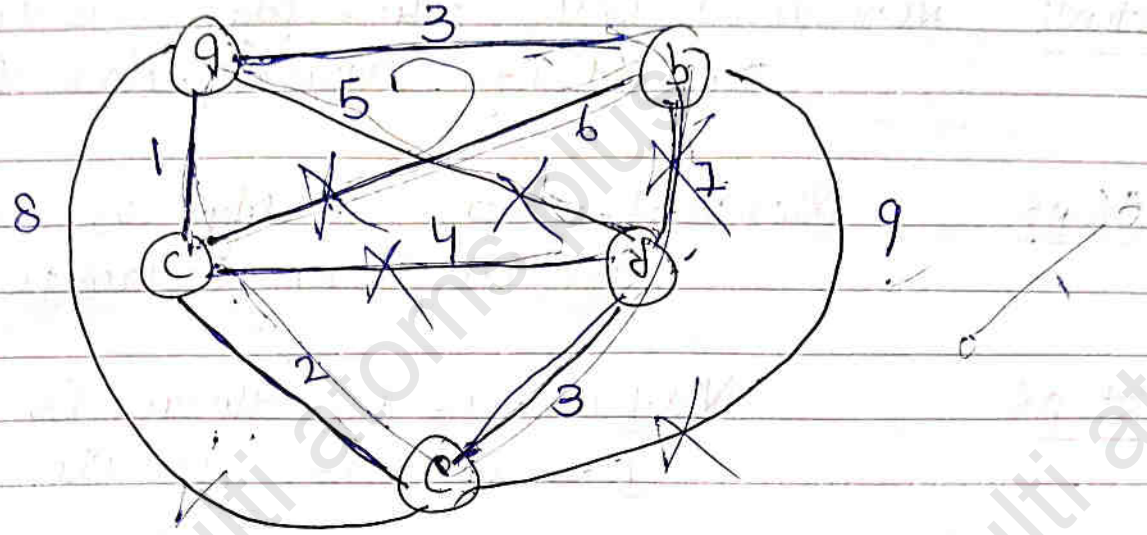
Step2 :- Select next edge for tree of a graph, having the least cost.

Step 3 Add this new edge only if inclusion of this edge does not form a closed path

Step4 Repeat Step 2 & 3 this all the nodes of graph are included in tree

$$\boxed{\text{Time Complexity} = O(v \log v)}$$

Q

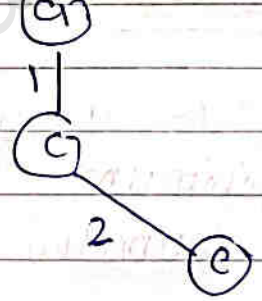


Step-1 Arrange all edges in their increasing order of weight

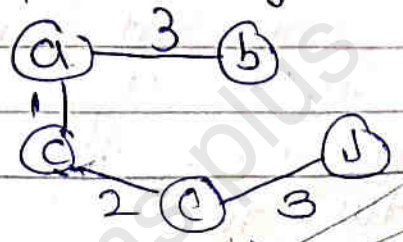
(a,c)	(c,e)	(e,d)	(a,b)	(c,d)	~~(d,a)~~	~~(c,b)~~
1	2	3	3	4	5	6

~~(d,b)~~	~~(a,e)~~	~~(b,e)~~
~~7~~	8	9

Step-2 Add the edges which has atleast weight edges



Step-3 Net cost is 3, the edges are e,d and a,b



Step4 Next cost is 4, the edge is c,d
Discard it create close loop

Step5 Next cost is 5 the edge is d,a
Discard it create close loop

Step6 Next cost is 6 the edge is a,b
Discard it create close loop

Step 7 Next cost is 7 the edge is d,b
Discard it create close loop

Step-8 Next cost is 8 the edge is a,e
Discard it create close loop

Step-9 Next cost is 9 which is discard.

Minimum path = 1+3+2+3 = 9 Ans

⇒ Prims Algorithm :-

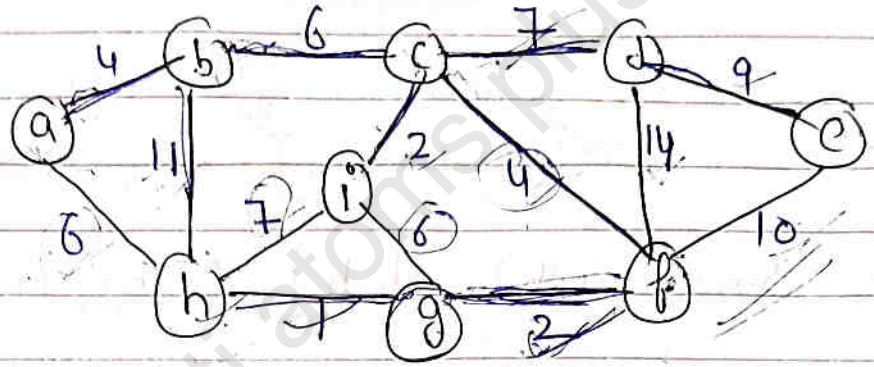
- Prims algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted connected graph.

Algo :-

- Initialize the minimum spanning tree with a vertex choose at random
- find all the edges that connect the tree to new vertices, find the minimum and add it to the tree

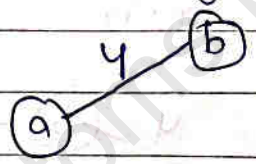
- Repeat step 1 and 2 until all vertices are marked visited.

Q.

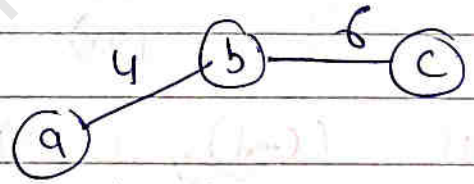


Step-1 Remove loops & parallel edges
→ If parallel edges exist, retain the one with the smallest weight

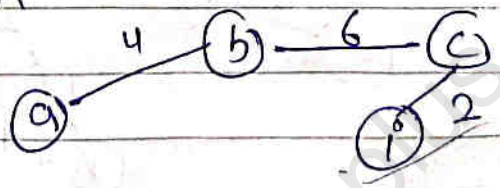
Step 2 Select any vertex as source node.
→ a be the source node, select edge (a,b) as distance between edge (a,b) is minimum



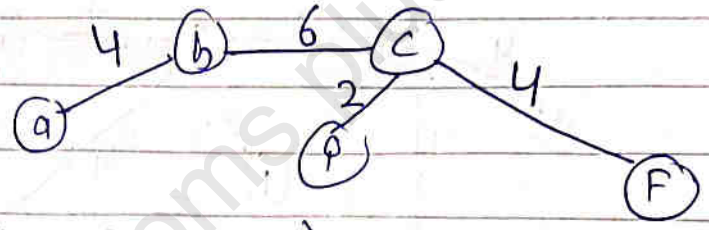
Step3 Now you have (a-h), (b-c), (b-h)
– select edge (b,c) as minimum weight



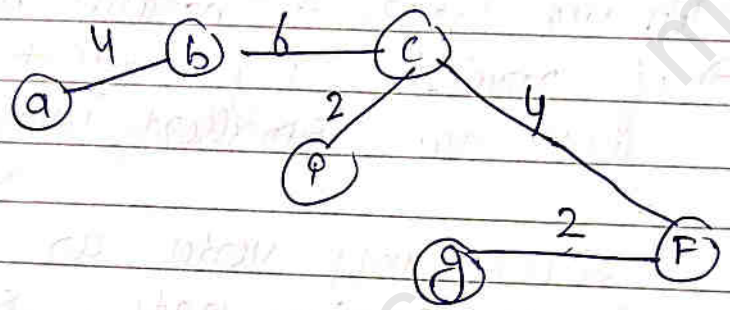
Step4 Now you have (a-h), (b-h), (c-d), (c,f) (c,i)
Select min



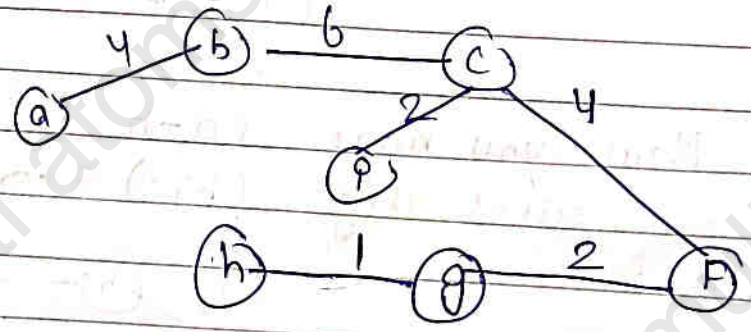
Now you have (a-h), (b-h), (i-h), (i-g), (c-d), (c-f)
– select min



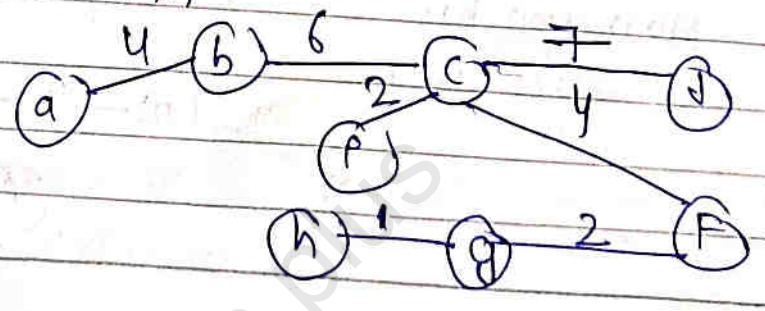
Now you have (a-h), (b-h), (i-h), (i-g), (c-d), (f-g), (f-c)
→ select min



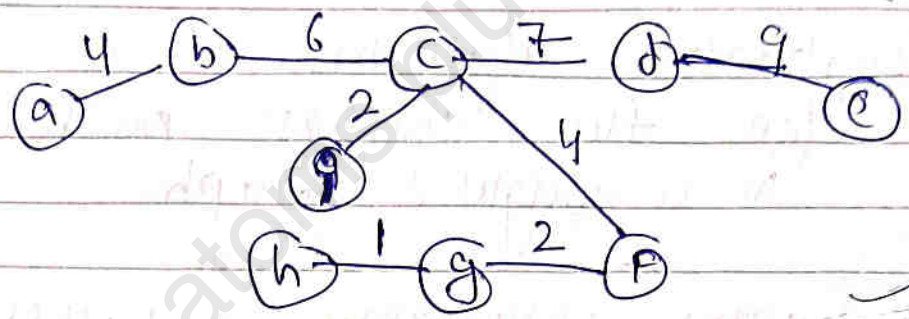
Now you have (a-h), (b-h), (c-d), (F-f), (g-h), (i-h)
→ select min



Now, you have (c-d), (F,c), (F-d)
→ select min



Now you have (d-e), (f-e)
→ select min



Total cost = 4+6+7+9+2+4+2+1
⇒ $\boxed{35}$ Ans

$\boxed{\text{Time Complexity} = O(E \log V)}$

⇒ **Single Source Shortest Path:-**

→ The shortest path problem is about finding the minimum distance or cost to travel between two nodes in a graph.

Algorith for Shortest Path:-

1] Dijkastra Algorithm

2] Bellman Ford Algorithm

DIJKASTRA ALGORITHM {AKTU 2022-2023}

- Dijkastra algorithm is an algorithm for find the shortest path between nodes in a weighted graph

- Dijkastra Algorithm is a well known greedy algorithm used to solve the single - source shortest path problem. It find the shortest path from a source vertex to all other vertices in a graph with non-negative weight.

Time Complexity :-

- Adjacent Matrix : $O(v^2)$

- Adjacent list : $O((V+E) \log V)$

Relaxation :-

$$\boxed{\begin{array}{l} \text{if } d(v) \geq d(u) + w(u,v) \\ \quad \text{then } d(v) = d(u) + w(u,v) \end{array}}$$

Algo :-

Step 1 : First, we will mark the source node with a current distance of 0 and set the rest of the nodes to Infinity (∞)

Step 2 : We will then set the unvisited node with the smallest current distance as the current node. Suppose X.

Step 3 :- For each neighbour N of the current node X; We will then add the current distance of X with the weight of the edge joining (X-N). If it is smaller than the current distance of N, set it as the new current distance of N.

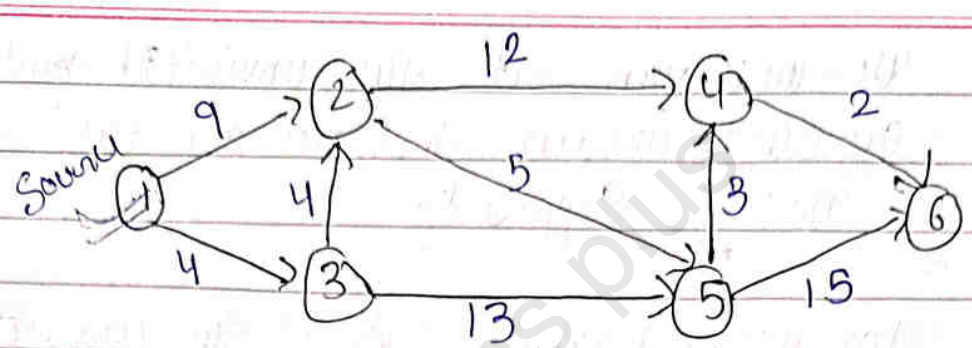
Step 4 : We will then mark the current node X as visited.

Step 5 :- We will repeat the process from 'Step 2' if there is any node unvisited left in the graph.

Application :-

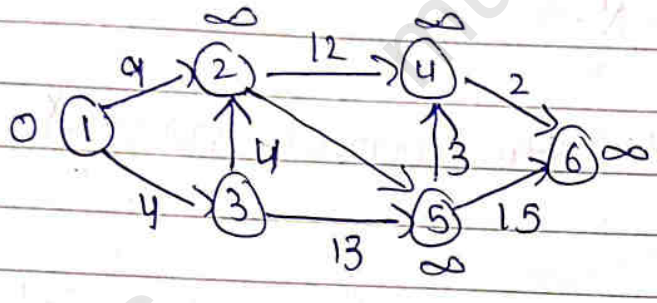
→ Social networking application.
→ Telephone network.
→ find location in the map

Q



Sol let A as a source vertex:

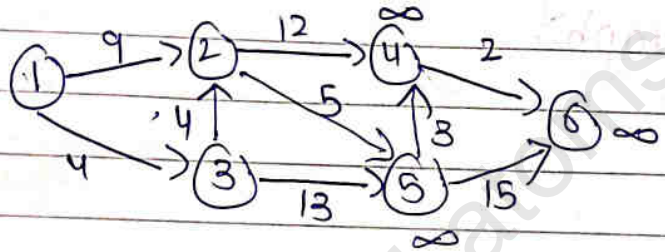
Step1: Initialization



$S = \{\}$

	1	2	3	4	5	6
Q =	0	∞	∞	∞	∞	∞

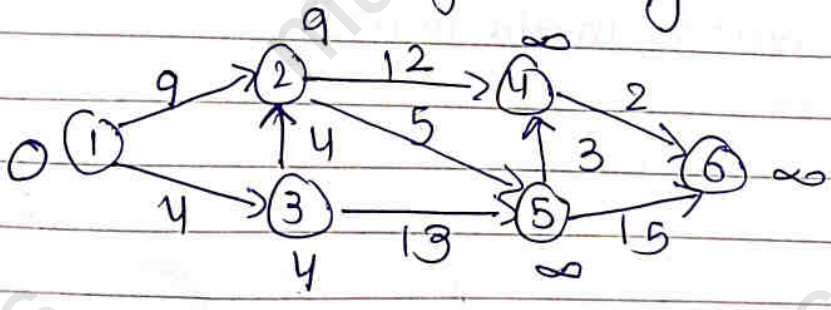
Step2: Extract-Min[1]



$S = \{1\}$

	1	2	3	4	5	6
Q =	$\boxed{0}$	∞	∞	∞	∞	∞

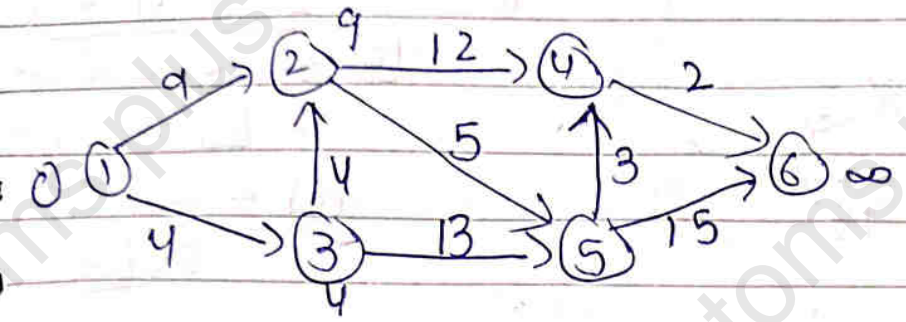
Relax all edges leaving 1



$S = \{1\}$

	1	2	3	4	5	6
Q =	$\boxed{0}$	∞	∞	∞	∞	∞
		9	4	∞	∞	∞

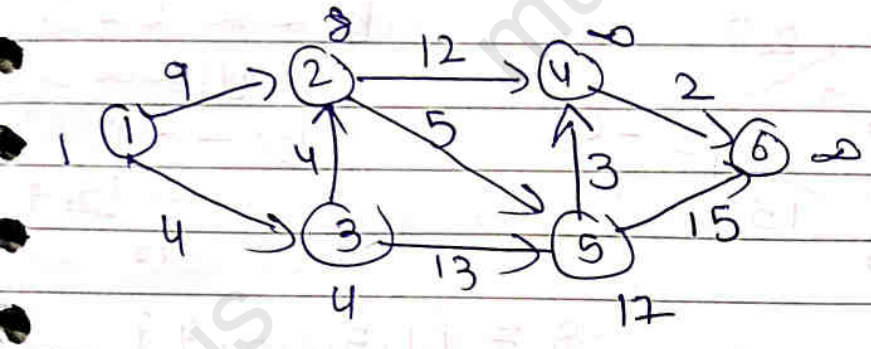
⇒ Extract - Min [3]



$S = \{1, 3\}$

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
9	$\boxed{4}$	∞	∞	∞	∞

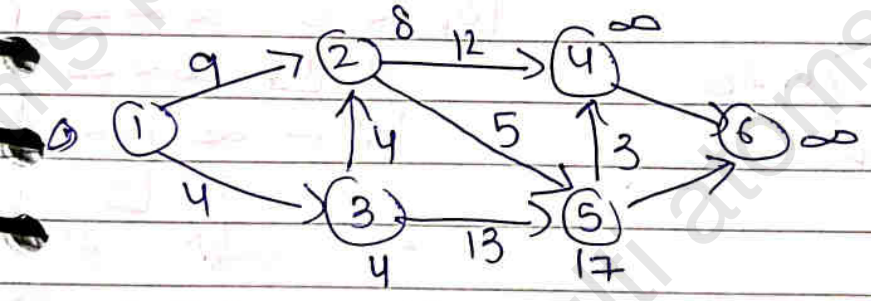
Relax all edges leaving [3]



$S = \{1, 3\}$

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
9	$\boxed{4}$	∞	∞	∞	
8	-	∞	17	∞	

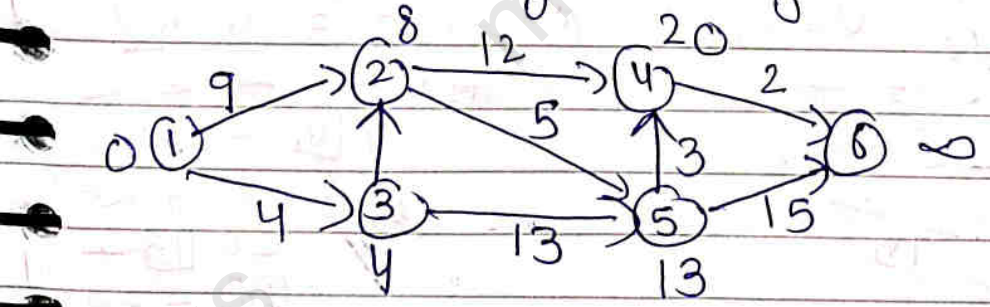
⇒ Extract - Min [2]



$S = \{1, 3, 2\}$

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
9	$\boxed{4}$	∞	∞	∞	
$\boxed{8}$	-	∞	17	∞	

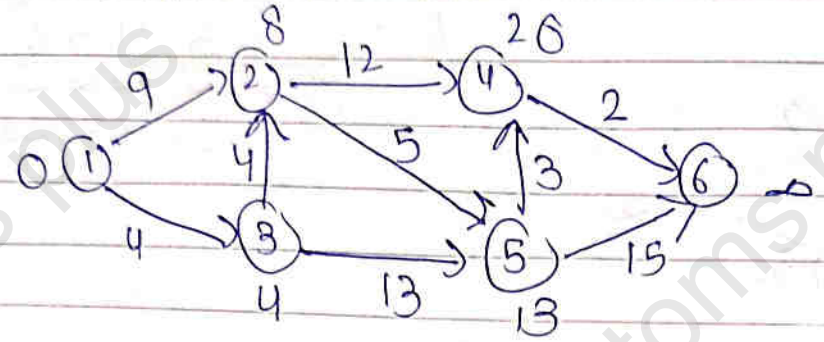
Relax - all edges leaving [2]



$S = \{1, 3, 2\}$

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
9	$\boxed{4}$	∞	∞	∞	
$\boxed{8}$	-	∞	17	∞	
20	13	∞			

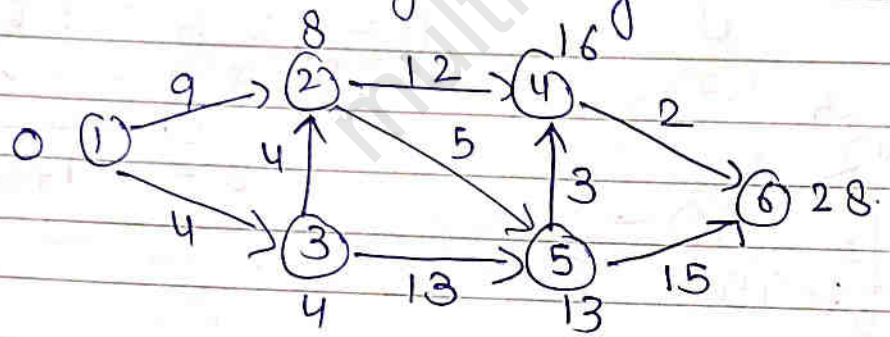
Extract - min[5]



S = {1, 3, 2, 5}

Q =	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
	9	4	∞	∞	∞	
	8	–	∞	17	∞	
		20	13	∞		

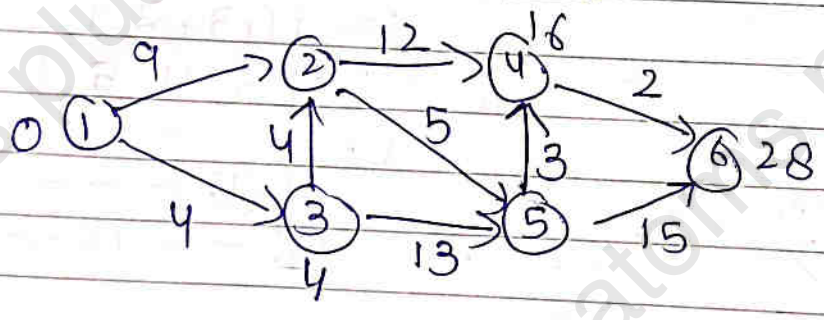
Relax - all edges leaving [5]



S = {1, 3, 2, 5}

Q =	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
	9	4	∞	∞	∞	
	8	–	∞	17	∞	
		20	13	∞		
		16		28		

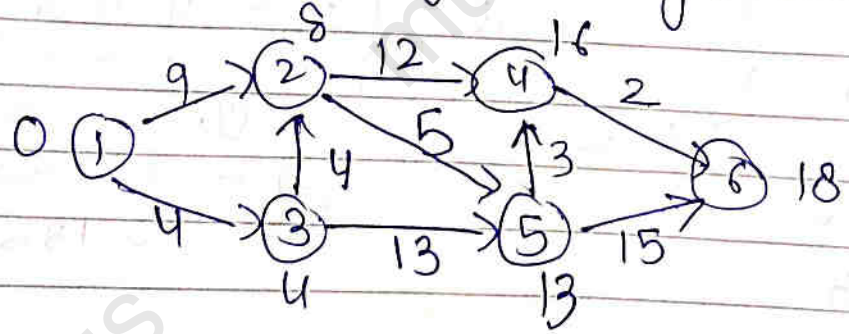
Extract - Min[4]



S = {1, 3, 2, 5, 4}

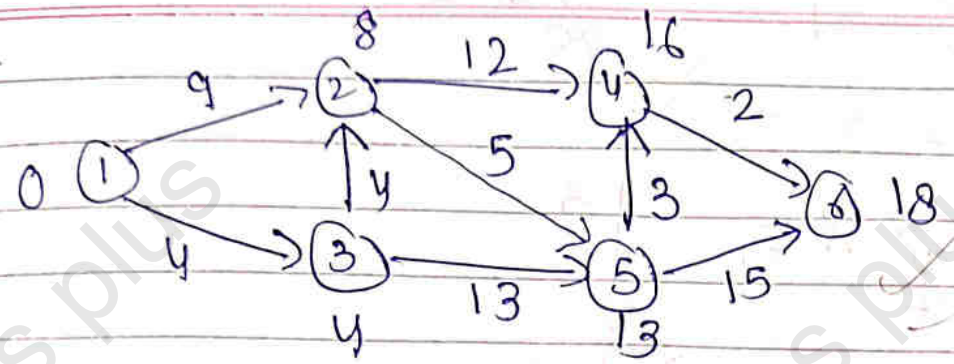
Q =	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
	9	4	∞	∞	∞	
	8	–	∞	17	∞	
		20	13	∞		
		16		28		

Relax all edges leaving [4]



S = {1, 3, 2, 5, 4}

Q =	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
	9	4	∞	∞	∞	
	8	–	∞	17	∞	
		20	13	∞		
		16	–	28		
				18		



$S = \{1, 3, 2, 5, 4, 6\}$

Q =	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	∞	∞	∞	∞
	$\boxed{8}$	–	∞	17	∞	
		20	$\boxed{13}$	∞		
			$\boxed{16}$	–	28	
					$\boxed{18}$	

⇒ Bellman Ford algorithm :- {AKTU 2023-2024}

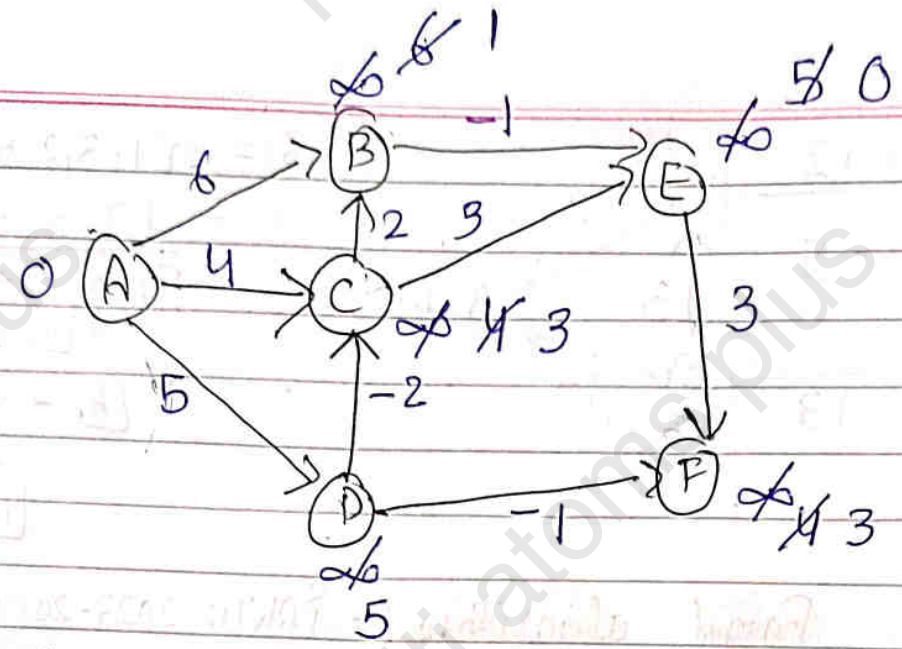
- It is used to find all shortest path in a graph from one source to all other nodes
- The algorithm requires that the graph does not contain any cycle of negative length but if it does, the algorithm is able to detect it.

No. of iteration = No. of vertices - 1

- This algorithm, use the notation of edge relaxation but does not use with greedy method
- The algorithm return boolean True if the give diagraph contains no negative cycles than are reachable from source vertex otherwise it return false

Time complexity = O(E·V)

Q



edges:-

(A,B) (A,C), (A,D), (B,E), (C,E), (D,C), (D,F)
(E,F), (C,B)

Iteration 1 : Complete
Iteration 2 : Complete
Iteration 3 : ⟶ No change, further, exit

A – 0
B – 1
C – 3
D – 5
E – 0
F – 3

A – B – C – D – E – F

Sum of path = 0 + 1 + 3 + 5 + 0 + 3
= 12 Ans

	A	B	C	D	E	F
0	∞	∞	∞	∞	∞	∞
1	0	~~6~~ 7+1	~~4~~ 3	5	5	4
2	0	1	3	5	0	3
3	0	1	3	5	0	3
4	0	1	3	5	0	3
5	0	1	3	5	0	3

Ans

UNIT-3 COMPLETE THANK YOU

LIKE SHARE SUBSCRIBE

UNIT-4
ONE SHOT
DESIGN ANALYSIS OF ALGORITHM

© MULTIATOMS © MULTIATOMS+

Topics :-

- Dynamic Programming
- All Pair Shortest Path Problem {AKTU 2022-2023}
- Resource Allocation
- Backtracking {AKTU 2023-2024}
- N-Queen Problem {AKTU 2022-2023}
- Sum of Subsets {AKTU 2022-2023}
- Hamiltonian Circuit {AKTU 2023-2024}
- Graph Coloring
- Branch and Bound Method
- Travelling Salesperson Problem {AKTU 2022-2023}

⇒ DYNAMIC PROGRAMMING

⇒ Dynamic Programming is a technique that breaks the problems into sub-problems and save the result for future purpose that we do not need to compute the result again.

⇒ The subproblems are optimized to optimize the overall solution is known as optimal substructure property

⇒ The main use of D.P is to solve optimization problem

Works

⇒ It breaks down the complex problem into simpler subproblems

⇒ It finds the optimal solution to the sub-problems

⇒ It stores the result of subproblem (memoization).

⇒ The processing of storing the result of sub-problem is known as memorization.

⇒ It reuses them so that same sub-problem is calculated more than once.

⇒ finally calculate the result of the complex problem

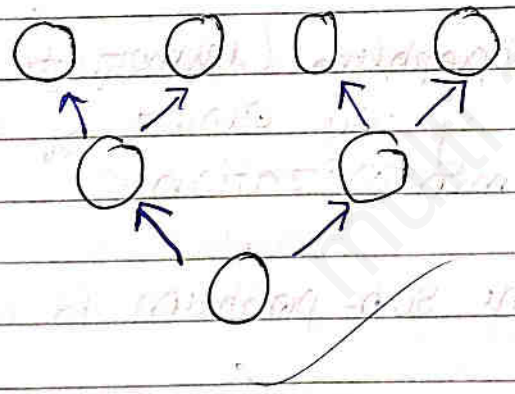
⇒ Approaches of DP :-

1. Memoization (Top-Down approach) :

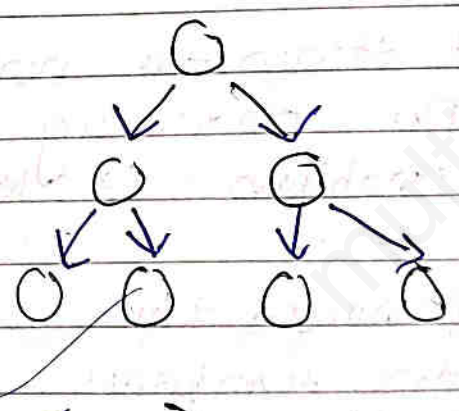
- Starting solving the problem from the top.
- Break it into smaller subproblem as needed
- Use a table or dictionary to store result of already solved sub-problems
- Check if the subproblem has been solved before fetching the answer from the table.

2. Tabulation (Bottom-Up approach) :

- Start solving from the smallest subproblem (base case) and build up solution for larger problems
- Store the result in a table, filling it step by step until the main problem is solved



Bottom-Up



Top-Down

⇒ 0/1 Knapsack Problem :- Done in Unit-3

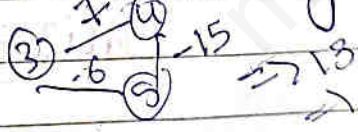
⇒ ALL PAIR SHORTEST PATH PROBLEM :-

⇒ The ALL pair shortest path problem involves finding the shortest path between all pairs of nodes in a weighted graph.

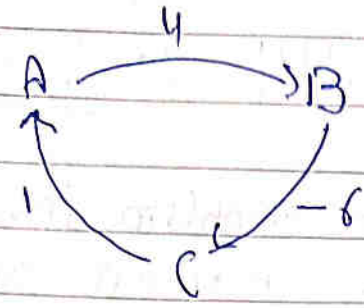
Approach :- Floyd- Warshall Algorithm

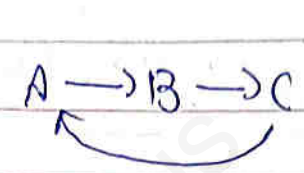
⇒ Floyd - Warshall Algorithm {AKTU 2023-2024}

- The problem is to find shortest distances between every pair of vertices in a given weighted directed graph
- It is a dynamic programming approach to find the shortest path
- It is particurly useful for dense graph and works for both directed and undirected graph.
- It can handle both positive & negative weight but it cannot handle graphs with negative weight cycles



- A negative weight cycle :- is a loop in a graph where the sum of the weight of all the edges in the cycle is negative.





$= 4 + (-6) + 1 = -1$

Time Complexity : $O(V^3)$

Steps to Execute Floyd-Warshall Algorithm

1. Initialization :- Create an initial matrix A0 where:

-> A[i][j] is the weight of the edges from vertex i to j, if edge exists
-> If there is no direct edge, set $A[i][j] = \infty$
-> set $A[i][j] = 0$ for all i.

2. Iterative Updates

-> For each vertex k (intermediate vertex), update the matrix A

$$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$$

-> This check whether the path from i to j through k is shorter than the direct path

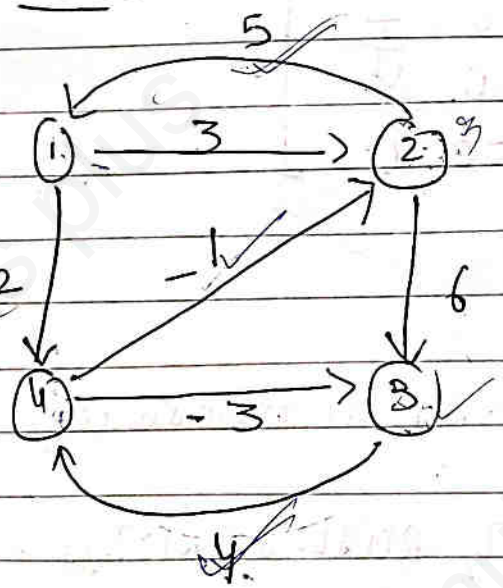
3. Repeat for all vertices:

- for $K=1$ to n, compute the shortest paths considering all vertices 1 to K as possible intermediate points.

4. Final Result:-

- After n iterations (where n is the number of vertices), the matrix A_n contains the shortest path b/w all pairs of vertices

Example:-



$4-1 = 3+3 = 8$

Step 1 =

$$A_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 2 \\ 2 & 5 & 0 & 6 & \infty \\ 3 & \infty & \infty & 0 & 4 \\ 4 & \infty & -1 & -3 & 0 \end{array}$$

Step 2 Iteration for $K=1$

Update $A[i][j]$ using vertex 1 as an intermediate

$$A[i][j] = \min(A[i][j], A[i][1] + A[1][j])$$

∞ $5+2 = 7$

$A[2][4] = 7$

eg: $A[3][2] = \infty$, $A[3][1] + A[1][2] = \infty$

$\infty + 3$

$A[3][2] = \min(A[3][2], A[3][1] + A[1][2])$

e.g $A[4][2] = -1$, $A[4][1] + A[1][2] = \infty$

$\infty + 3$

$(-1, \infty) = -1$

you have do for every $A[i][j]$

$$A' = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 2 \\ 2 & 5 & 0 & 6 & 7 \\ 3 & \infty & \infty & 0 & 4 \\ 4 & \infty & -1 & -3 & 0 \end{array}$$

Step-3 Iteration for $K = 2$

Update $A[i][j]$ using vertex 2 as an intermediate!

$A[i][j] = \min(A[i][j], A[i][2] + A[2][j])$.

e.g $A[1][3] = \infty$, $A[1][2] + A[2] + A[3] = 9$

$3 \quad 6$

$\min(\infty, 9) = 9$

e.g $A[4][1] = \infty$, $A[4][2] + A[2][1] = 4$

$-1 + 5$

$\min(\infty, 4)$

$A[4][1] = 4$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 2 \\ 5 & 0 & 6 & 7 \\ \infty & \infty & 0 & 4 \\ 4 & -1 & -3 & 0 \end{bmatrix} \end{array}$$

Step 4: Iteration for $k = 3$

Update $A[i][j]$ using vertex 3 as an intermediate:

$$A[i][j] = \min(A[i][j], A[i][3] + A[3][j])$$

e.g $A[2][1] = 5$, $A[2][3] + A[3][1] = \infty$

$\min(5, \infty)$

$A[2][1] = 5$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 2 \\ 5 & 0 & 6 & 7 \\ \infty & \infty & 0 & 4 \\ 4 & -1 & -3 & 0 \end{bmatrix} \end{array}$$

Step 5: Iteration for $k = 4$

Update $A[i][j]$ using vertex 4 as an intermediate:

$$A[i][j] = \min(A[i][j], A[i][4] + A[4][j])$$

e.g $A[1][3] = 9$, $A[1][4] + A[4][3] = -1$

$2 + (-3)$

$\min(9, -1) = -1$

$A[1][3] = -1$

e.g $A[3][1] = \infty$, $A[3][4] + A[4][1] = 8$

$4 + 4$

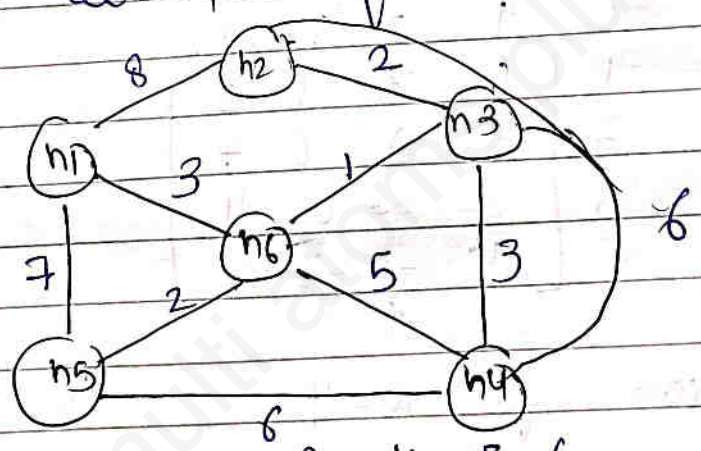
$\min(\infty, 8) = 8$

$A[3][1] = 8$

$$A_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & -1 & 2 \\ 2 & 5 & 0 & 4 & 7 \\ 3 & 8 & 3 & 0 & 4 \\ 4 & 4 & -1 & -3 & 0 \end{array}$$

The matrix A_4 gives the shortest distances between all pair of vertices

Q 4.10



$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 6 & 4 & 7 & 5 & 3 \\ 2 & 6 & 0 & 2 & 5 & 5 & 3 \\ 3 & 4 & 2 & 0 & 3 & 3 & 1 \\ 4 & 7 & 5 & 3 & 0 & 7 & 4 \\ 5 & 5 & 5 & 3 & 7 & 0 & 2 \\ 6 & 3 & 3 & 1 & 4 & 2 & 0 \end{array}$$

⇒ RESOURCE ALLOCATION PROBLEM

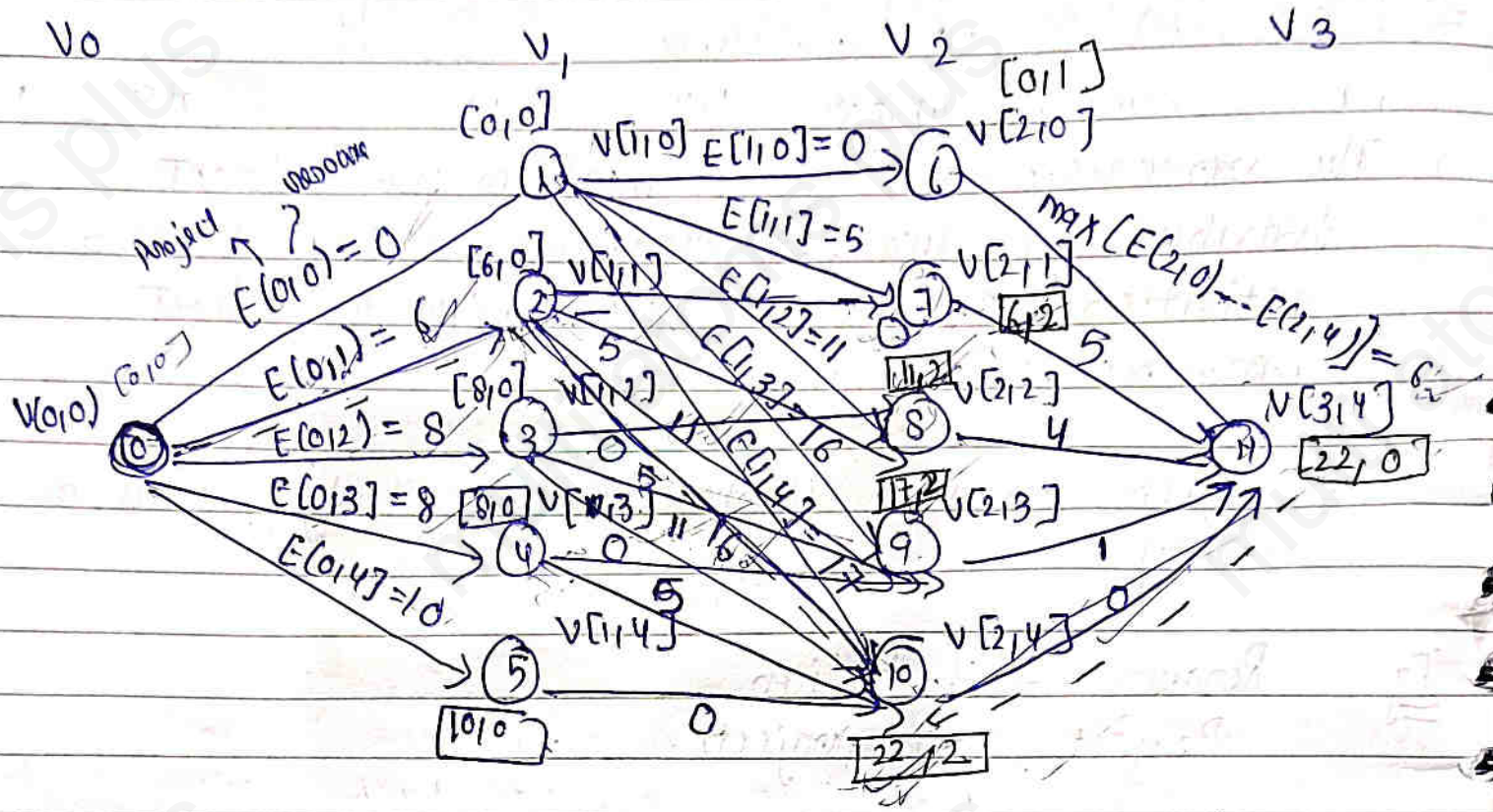
- The resource allocation problem is about distributing limited resources among various activities or projects to achieve the best possible outcome.

- It often involves constraints like budget, time or capacity.

Eg ⇒ Resource – 4 billion
Project – 3 project

	0	1	2	3	4
0	0	6	8	8	10
1	0	5	11	16	17
2	0	1	4	5	6

3 billion = 16
1 billion = 6
22



1 Resource	0 project	6 Profit
3 Resource	1 project	16 Profit
		22 profit

=> N-QUEENS PROBLEM {AKTU 2021-2022}

=> In N-Queens problem, all possible placement of n queen on an nxn chess board so that they non-attacking (not in same row, column or diagonal)

=> No diagonal should contain more than one queen

=> No row should contain more than one queen

- No column should contain more than one queen

(i) This algorithm first calls Q places (k, i) algorithm to check whether the queen k can be placed on column i or not.

Algorithm :- NQueens (k, n)

```
{
    for i = 1 to n do
    {
        if Q place (k, i) then
        {
            x[k] = i;
            if (k == n) then
                write [x [1:n]];
            else
                N Queens (k+1, n);
        }
    }
```

=> The below algorithm Q place (k, i) returns true if Queen k can be placed in k^{th} row & ith column otherwise return false.

=> Q place (k, i) {

```
    for j = 1 to k-1 do
    {
        if((x[j] = i) or (Abs(x[j] - i) = Abs[j - k])) then
            // if two queens are in same column or
            // same diagonal.
            return false;
    }
    return true;
}
```

4- Queens Problems :-

- Suppose we have 4x4 chessboard with 4- queens each to be placed in non-attacking position
- Now, we will place each queen on a different row such that no two queens attacks each other.
- We place the queen q_1 in the very first accept position (1,1)
- Now if we place queen q_2 in column 1 and 2 then the dead end is encountered
- $q_2 \rightarrow (2,3)$, but then no position is left for q_3. so, we backtract one step and $q_2 \rightarrow (2,4)$
- $q_3 \rightarrow (3,2)$, but now there is no place left for q_4.
- Then, we have to backtracks till queen q_1 and $q_1 \rightarrow (1,2)$ $q_2 \rightarrow (2,4)$ and $q_3 \rightarrow (3,1)$, $q_4 \rightarrow (4,3)$ we get solution <2,4,1,3> this one possible sol.

	1	2	3	4
1	a_1			
2				a_2
3	a_3	a_3		
4				

1	a_1			
2			a_2	
3				
4				

No, place for q_4

	1	2	3	4
1		a_1		
2				a_2
3	a_3			
4			a_4	

$\langle 2, 4, 3, 1 \rangle$

⇒ For other sol. whole method repeated

Sol2 $\langle 3, 1, 4, 2 \rangle$

	1	2	3	4
1			a_1	
2	a_2			
3				a_3
4		a_4		

8 Queens Problem

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

8x8

Solution Vector

1	2	3	4	5	6	7	8
4	6	8	2	7	1	3	5

SUM OF SUBSETS {AKTU 2022-2023}

=> It is one of most important problems in complexity theory.

=> Let understand this problem through an example:

-> We have a set of 5 integers given below:

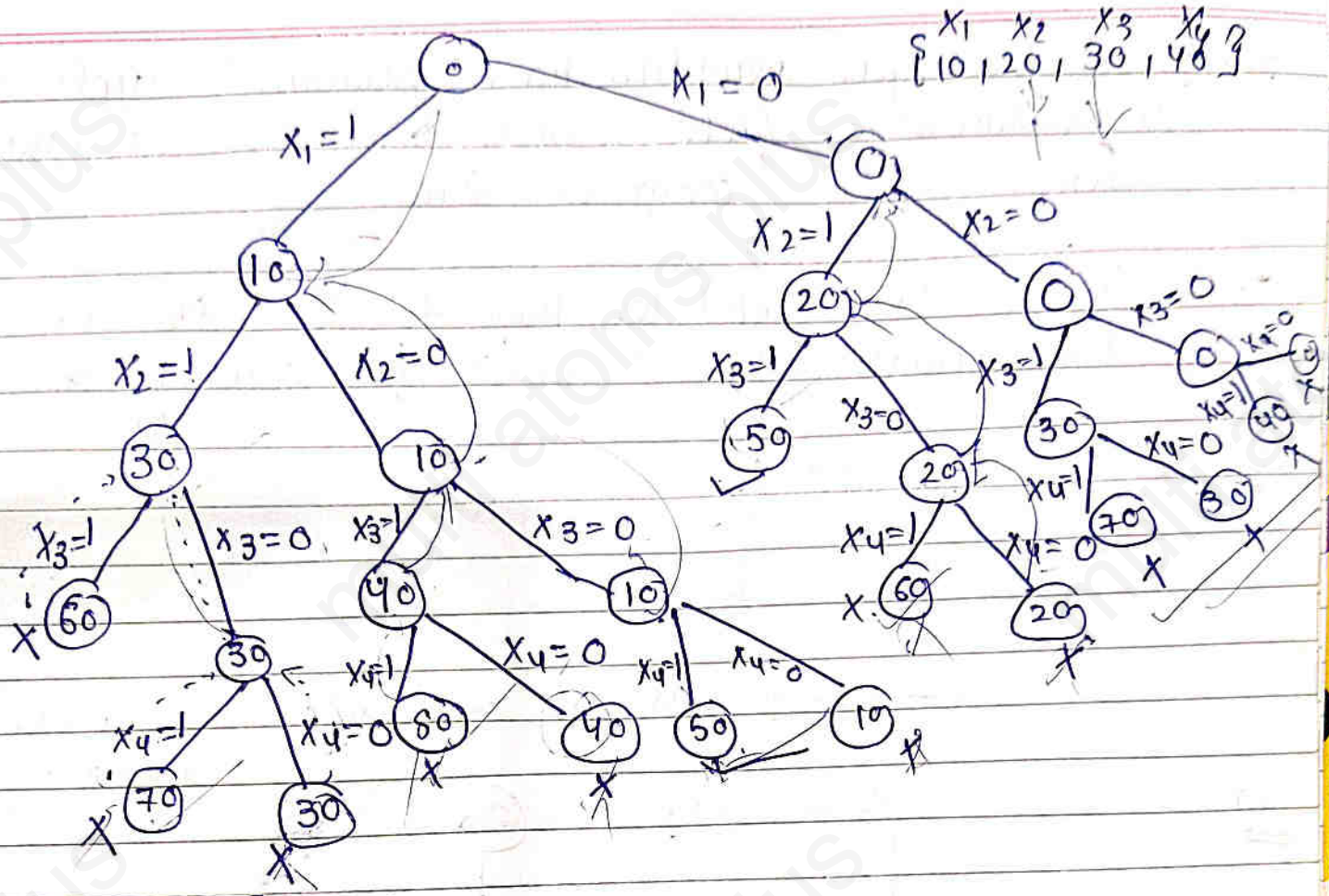
N : 4, -2, 2, 2, 3, 1
M = 5

-> We want to find out the subset whose sum is equal to 5. There are many solution to this approach

if $x_1 = 1$ -> include
$x_1 = 0$ -> Exclude.

Q A = {10, 20, 30, 40}
Sum = 50

Subset = {20, 30}
{10, 40}



{1 | 0 | 0 | 1} = Subset

| 0 | 1 | 1 | 0 | =

Ans

⇒ HAMILTONIAN CIRCUIT {AKTU 2021-2022}

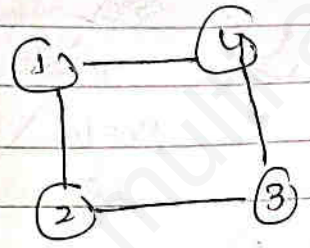
⇒ Hamiltonian cycle or circuit in a graph G is a cycle that visit every vertex of G exactly once and returns to the starting vertex.

⇒ If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian

⇒ Hamiltonian cycle problem has practical applications in various field, such as logistics, network design and computer science

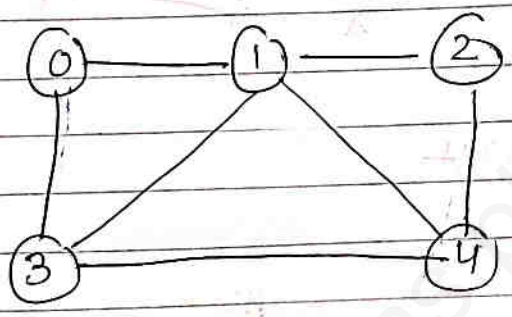
⇒ Graph $G = (V, E)$ we have to find the say, Hamiltonian circuit using backtracking approach.

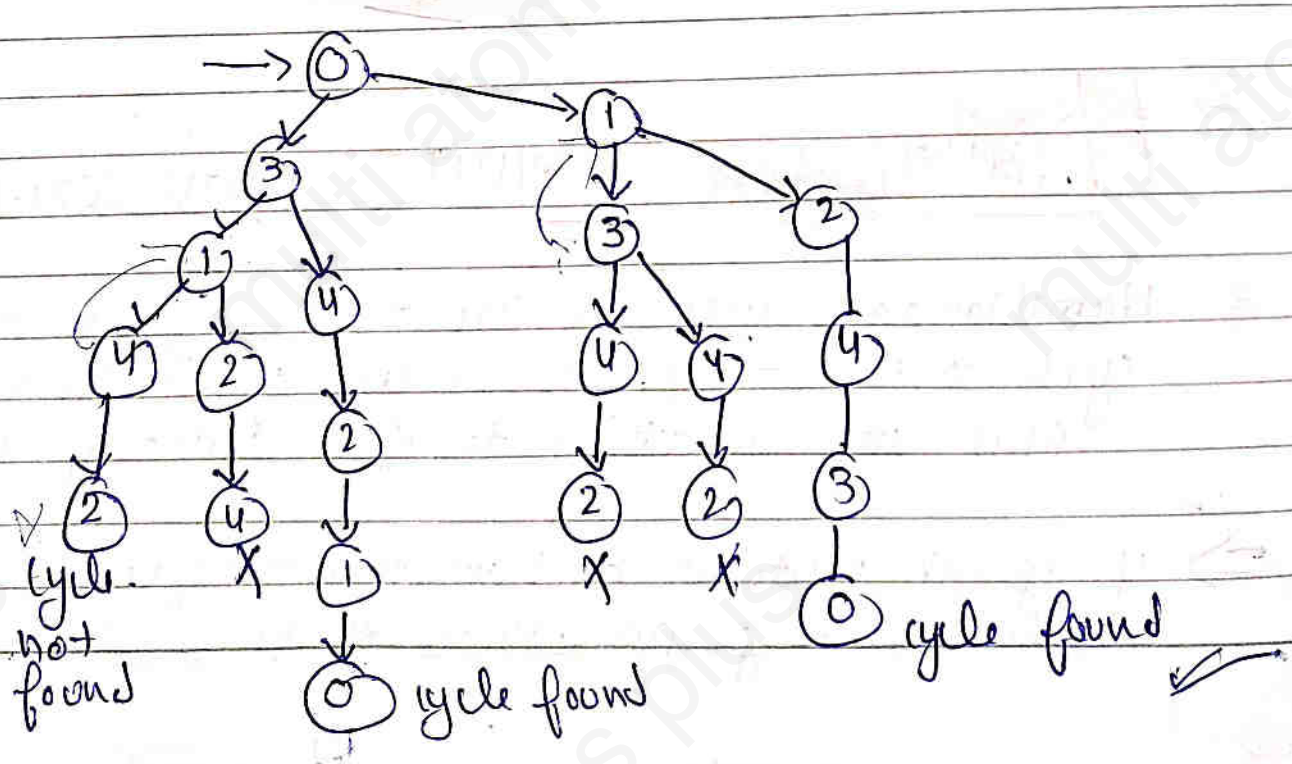
Eg



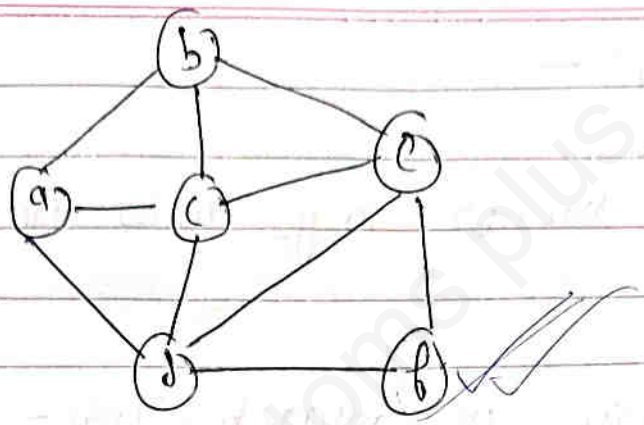
1 — 2 — 3 — 4 — 1 ⇒ Hamiltonian cycle

Eg





Q HW



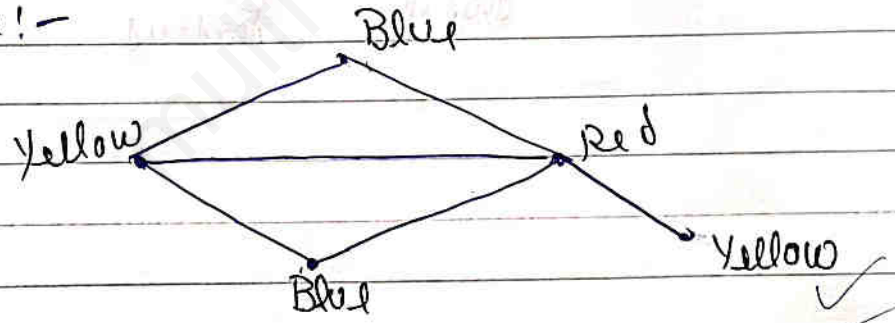
Sol Type in comment

⇒ Graph Coloring

- In graph coloring problem we color the vertices of graph in such a way that no two adjacent vertices share the same color.

- In this problem we have a given a chromatic number m which is the least number of color needed to color a graph.

- For ex :-



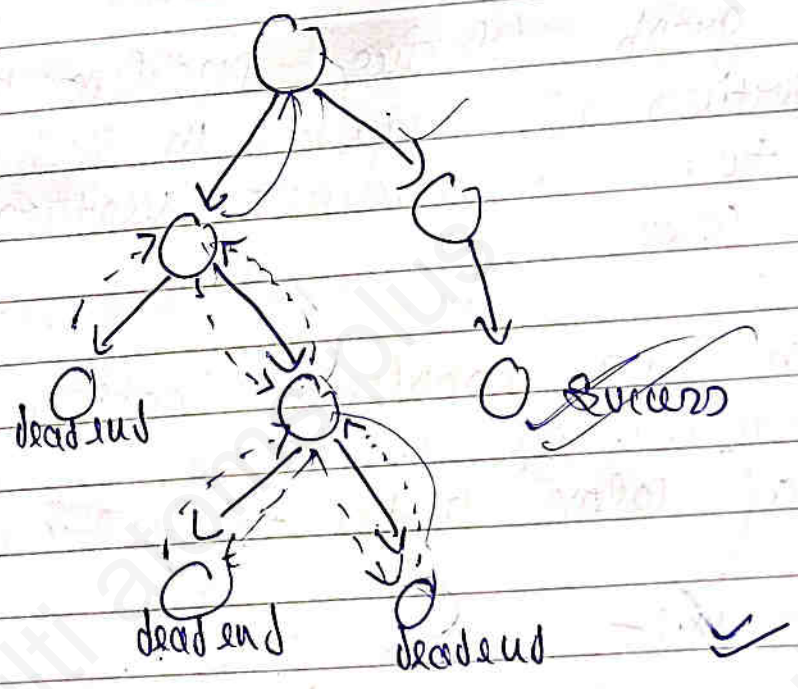
chromatic no :- no of colors used for coloring graph.

Backtracking :-

=> Backtracking means to go back the same way you come

=> In backtracking we search depth-first for solution.

=> In backtracking technique we backtrack to the last valid depth as soon as we hit a dead end



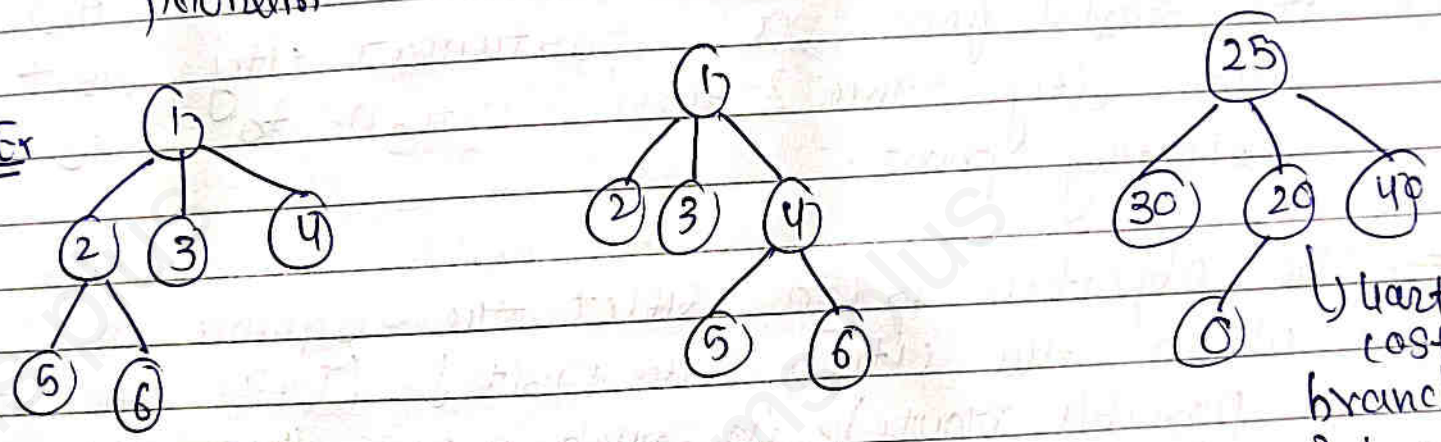
Branch and Bound Method

- The branch and bound algorithm is similar to backtracking but is used for solving optimization problems.

- Branch and Bound method applied where greedy and D.P techniques fails

- This approach is effective for optimization problem

Ex



Queue | 2 | 3 | 4 |

FIFO Branch & Bound

4
3
2

LIFO Branch bound

TRAVELLING SALESPERSON PROBLEM [AKTU 2023-24]

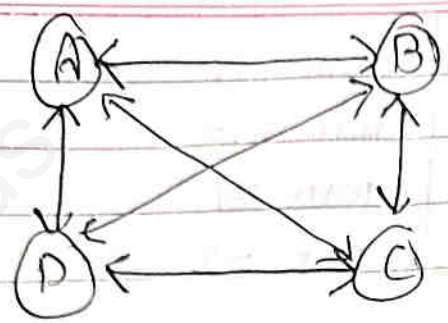
⇒ Given a set of cities & distance between every pair of cities.

⇒ The problem is to find the shortest possible route that a salesman visit every city exactly once & returns to the starting point

⇒ Assuming a salesman has to travel n cities
⇒ He start from a particular city, visit each city once & then return to his starting point.

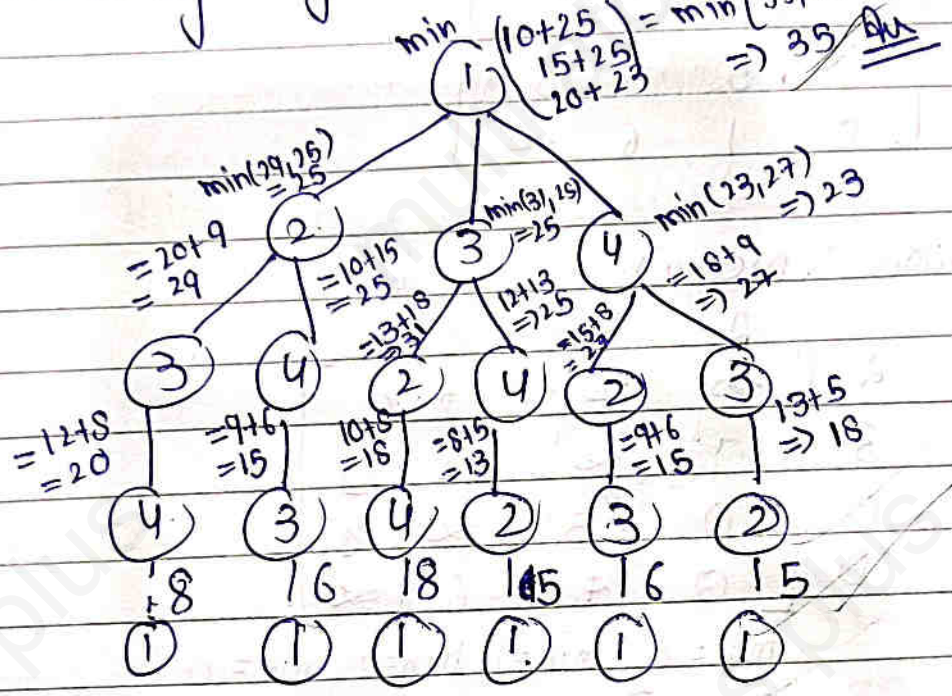
⇒ The objective is to select the sequence in which the cities are visited (with shortest possible route) in such a way that his total travelling cost is minimized.

Q



$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 9 & 10 \\ 3 & 6 & 13 & 0 & 12 \\ 4 & 8 & 8 & 9 & 0 \end{array}$$

=> using dynamic approach



=> Using Branch and Bound

Q

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & 10 & 5 & 3 \\ B & 8 & \infty & 9 & 7 \\ C & 1 & 6 & \infty & 9 \\ D & 2 & 3 & 8 & \infty \end{array}$$

① Row Reduction matrix =

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & 10 & 5 & 3 \\ B & 8 & \infty & 9 & 7 \\ C & 1 & 6 & \infty & 9 \\ D & 2 & 3 & 8 & \infty \end{array} \quad \begin{array}{l} \min = 3 \\ \min = 7 \\ \min = 1 \\ \min = 2 \end{array}$$

$$\begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{bmatrix} \infty & 7 & 2 & 0 \\ 1 & \infty & 2 & 0 \\ 0 & 5 & \infty & 8 \\ 0 & 1 & 6 & \infty \end{bmatrix}$$

(b) Col Reduction matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & 7 & 2 & 0 \\ 1 & \infty & 2 & 0 \\ 0 & 5 & \infty & 8 \\ 0 & 1 & 6 & \infty \end{bmatrix} \end{array}$$

min = 0 min = 1 min = 2 min = 0

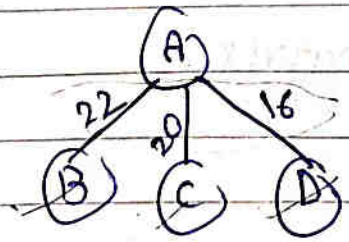
$M_1 =$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & 6 & 0 & 0 \\ 1 & \infty & 0 & 0 \\ 0 & 4 & \infty & 8 \\ 0 & 0 & 4 & \infty \end{bmatrix} \end{array}$$

Parent matrix cost = Sum of row reductn + col reductn

$= 3+7+1+2+2+1$

$= 16$



① A − B

$$M_2 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0 \\ 0 & \infty & \infty & 8 \\ 0 & \infty & 4 & \infty \end{bmatrix} \end{array}$$

There is no need of row reduction and column reduction because each row & column contain 0.

Total cost = Parent matrix cost + Reduction + Dis (A-B)

$= 16 + 0 + 6$

$= 22$

① A − C

$$M_3 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 0 \\ \infty & 4 & \infty & 8 \\ 0 & 0 & \infty & \infty \end{bmatrix} \end{array} \begin{array}{l} \\ \\ \min = 4 \\ \\ \end{array}$$

Row reduction

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 0 \\ \infty & 0 & \infty & 4 \\ 0 & 0 & \infty & \infty \end{bmatrix} \end{array}$$

Total cost $= 16 + 4 + 0$

$= 20$

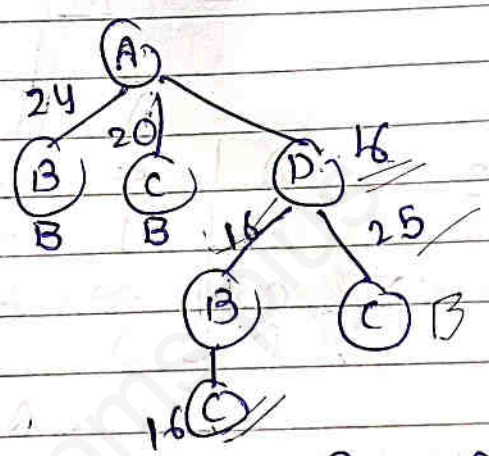
⊙ A-D

parent-Matrix = M_4 =

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 1 & \infty & 0 & \infty \\ C & 0 & 4 & \infty & \infty \\ D & \infty & 0 & 4 & \infty \end{array}$$

There is no used of reduction = 0

Total cost = 16 + 0 + 0
= 16



① A-D - B
row column

parent Matrix M_5 =

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & \infty & \infty & 0 & \infty \\ C & 0 & \infty & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array}$$

There is no reduction = 0

Total cost = parent Matrix cost + reductn + Dis (D-B)
⇒ 16 + 0 + 0
⇒ 16

$\Rightarrow$ A - D - C

row col

$$\begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{bmatrix} \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty \\ \infty & 4 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix} \begin{array}{l} \\ \text{Min} = 1 \\ \text{Min} = 4 \\ \\ \end{array}$$

Row Reduction

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 0 & \infty & \infty & \infty \\ C & \infty & 0 & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array}$$

Total cost = Parent Matrix cost reduction + Dis(D-C)

$= 16 + 5 + 4$

$= 25$

$\Rightarrow$ A - D - B - C

row col

$$\begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

No, Reduction

Total cost = Parent Matrix cost + Redu + Dis(B-C)

$\Rightarrow 16 + 0 + 0$

$\Rightarrow 16$

Shortest Path = A - D - B - C - A

$\Rightarrow 3 + 3 + 9 + 1 = 16$

UNIT-4 COMPLETE

THANK YOU FOR WATCHING

LIKE SHARE COMMENT

SUBSCRIBE

MULTI ATOMS

@ MULTI ATOMS +

DESIGN AND ANALYSIS OF ALGORITHM

ONE - SHOT BCS 503

SUBSCRIBE - MULTI ATOMS & PLUS

Topics :-

- Algebraic Computation.
- Fast Fourier Transformation {AKTU 2022-23/24}
- String Matching Algorithm {AKTU - 2021-22/23/24}
- Theory of NP - Completeness {AKTU - 2023-2024}
- Approximation Algorithm {AKTU 2021-22 24}
- Randomization Algorithm {AKTU 2023-2024}

multi atoms plus

⇒ Algebraic Computation :-

- Algebraic computation is the study of algorithm and software that manipulate expressions.

- Algebraic computation is solving problem or equation that include letters (variables) and numbers.

- It is also used for complete scientific computation when purely numerical method fails

→ Strassen Matrix Multiplication
→ Convex Hull problem
→ Public key Cryptography

⇒ Fast Fourier Transformation :- {AKTU 2023-24}

- A Fast Fourier Transform is an algorithm that computes the Discrete Fourier Transform of a sequence.

- Whe a signal is digital (a sequene of numbers), we use the Dirsicrete Fourier Transform (DFT). It convert a sequence of N Numbers into N frequency component

- The DFT formula take $O(N^2)$ steps to compute, which can be very slow for large N. The FFT speed this up by reducing the steps to $O(N \log N)$, making it much faster.

FFT work :-

1. Divide and Conquer :- It break the original signal into smaller part
2. Symmetry :- It reuses calculation by taking advantage of patterns in the sine and cosine function
3. Combine Results :- It efficiently combines the smaller part to get the full result.

$$x(t) \xrightarrow{FT} x(j)$$

$$x(j) = \int_{-\infty}^{\infty} x(t)\, e^{-i2\pi \cdot k \cdot n/N}\, dt$$

output frequency

input signal at time t

A complex exponential that represent sinusoidal waves

Application of Fast Fourier transform

1. Signal Processing
2. Image Processing
3. Fast multiplication of large integers
4. Solving Poisson equation nearly optimally

String Matching :- { AKTU 2022-23

- A string matching algorithm is a method to find a specific pattern (substring) within a larger text (string).

- The goal is to find all the positions at which the pattern appear in the text.

- The string matching problem is a classical problem in computer science where you want to find all occurances of a shorter string ("a pattern") within a longer string (a "txt").

Assume Text array $T[1 \ldots n]$
Patter array $P[1 \ldots m]$ $m \le n$

Element of P and T $\in \Sigma$ → $\{0, 1, \ldots 9\}$
→ $\{a, b, c \ldots z\}$

e.g

T	a	b	c	a	b	a	a	b	c	a	b	a	a

① ②

P	a	b	a	a

Algorithm used for String Matching :-

1. Naive String Matching Algorithm
2. Robin - Karp Algorithm
3. Finite - Automata String Matching Algo.
4. Knuth - Morris - Pratt Algorithm

⇒ Naive String Matching Algorithm {AKTU 2022-23

- The naive algorithm is the simplest method where you find all occurence of a pattern in a given text.
- It works by checking each possible starting position in the text to see if the pattern matches, character by character.

⇒ Steps of the Algorithm :-

1. Start at the first character of the text
2. Compare the pattern with the substring of the text starting at the current position.
3. If all character match, record the position as a match
4. move to the next position in the text and repeat steps 2-3 until the end of the text.
5. Return all the starting indices where the pattern matches

- NAVIE - STRING - MATCHER (T, P) {

```
n = T.length()
m = P.length()
for s=0 to n-m
    if (P[1..m] == T[s+1..s+m]
    print " pattern occur with shift " s
}
```

→ Pros:

- Very simple to understand and implement
- Works well when the pattern are short and text is not too large.

→ Cons:-

- Can be very slow, especially if the pattern occurs frequently but with mismatches

Q. T = | a | b | a | b | b | a | b | a | a |

P = | a | b | a | a |

Sol

| → match ↯ not match

S = 0

T = | a | b | a | b | b | a | b | a | a |

P = | a | b | a | a |

S = 1 T = | a | b | a | b | b | a | b | a | a |

P = | a | b | a | a |

S = 2 T = | a | b | a | b | b | a | b | a | a |

P = | a | b | a | a |

S = 3 T = | a | b | a | b | b | a | b | a | a |

| a | b | a | a |

S = 4 T = | a | b | a | b | b | a | b | a | a |

P = | a | b | a | a |

$S = 5$ $T =$

0	1	2	3	4	5			
a	b	a	b	b	a	b	a	a
					a	b	a	a

-> Pattern match at shift $S = 5$ Ans

Time Complexity $= O(n-m+1)$.
Worst case $= O(n*m)$.

=> Rabin - Karp Algorithm :- {AKTU - 2021-22/23

- The rabin-karp algorithm uses hashing to find any set of pattern occurances. Instead of checking all character of the pattern, it checks a hash value.

- It's especially useful when you need to search for multiple pattern's in a single text.

Pros :-
- Faster than the naive approach on average
- Very efficient for multiple pattern searches at one.

Cons :
- Requires a good hash function to avoid frequent supurious hits
- The worst case time complexity can be as bad as the naive algo if many hash collision occur

Q For $q=11$, how many valid and spurious hit found for given text and pattern $P=26$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T:	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3

P: | 2 | 6 |

Sol $P \bmod q = 26 \% 11 = 4$ Hash value $= 4$

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$31 \bmod 11 = 9$

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$14 \bmod 11 = 3$

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$41 \bmod 11 = 8$

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$15 \bmod 11 = 4$ — supurious hit

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$59 \bmod 11 = 4$ — Supurious hit

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$92 \bmod 11 = 4$ — Spurious hit

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

$26 \bmod 11 = 4$ — Valid hit

T: | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 |

10 9 2 3 1 9 2 5

Pattern found at shift = 6

Algorithm :-

Step1 : Initially calculate the hash value of the pattern

Step2 : Starting iterating from the starting of the string

- Calculate the hash value of the current substring having length m.
- If the hash value of the current substring and the pattern are same check if the substring is same as the pattern
- If they are same, store the starting index as a valid answer. Otherwise, continue for the next substring.

Step3 : Return the starting indices as the required answer

Time Complexity :- O(n+m)

worst case : O(n*m)

⇒ Knuth - Morris - Pratt - Algorithm (KMP)

{AKTU 2021-2022/23/24}

① The Knuth - Morris - Pratt algorithm is an efficient string matching techniques that avoids unnecessary comparisons by preprocessing the pattern.

① It uses a partial Match table or longest prefix that same as suffix to decide how much to shift the pattern upon a mismatch

P :- | a | b | c | d | a | b | c |

~~prefix :- a, ab, abc, abca ...~~
~~suffix :- a, ca, bca, abca ...~~

prefix :- a, ab, abc, abcd, abcda ...
suffix :- c, bc, abc, dabc, cdabc ...

Direct :-

	1	2	3	4	5	6	7	8	9	10
P_1	a	b	c	d	a	b	c	a	b	F
LPS/π	0	0	0	0	1	2	0	1	2	0

	1	2	3	4	5	6	7	8
P_2 :	a	b	a	b	a	b	c	a
LPS/π	0	0	1	2	3	4	0	1

Q string : a b ab c abc ab ab abd

	1	2	3	4	5
P :	a	b	a	b	d
	0	0	1	2	0

Sol start $i=1$, $j=0$

$i=1$

string : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=0$

$\Pi \rightarrow$ | a | b | a | b | d |
| 0 | 0 | 1 | 2 | 0 |

$\Rightarrow$ compare i and j+1 if match than move i and j to righ

$i = a$

$j+1 = a$ match

$i=2$

string : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=1$

$\Pi \xrightarrow{P:}$ | a | b | a | b | d |
| 0 | 0 | 1 | 2 | 0 |

$\Rightarrow$ $i = b$.

$j+1 = b$ match move right i and j

$j=3$

string : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=2$

$\Pi \xrightarrow{P:}$ | a | b | a | b | d |
| 0 | 0 | 1 | 2 | 0 |

$\Rightarrow$ $i = a$, $j+1 = a$ match move right i and j

i=4

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

j=3

P:	a	b	a	b	d
Π:	0	0	1	2	0

i = b, j+1 = b. match move right i and j

i=5

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

j=4

P:	a	b	a	b	d
Π:	0	0	1	2	0

i = c, j+1 = d mismatch, then move to index(j) of Π value at 2, i exist at same place

i=5

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

j=2

P:	a	b	a	b	d
Π:	0	0	1	2	0

i = c, j+1 = a, mismatch, then move j to index (j) of Π value at (0), i exist at same place.

i=5

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

j=0

P:	a	b	a	b	d
Π:	0	0	1	2	0

Now since j at 0, no move further backward so move i to right (i++).

$i=6$

string : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=0$

P:	a	b	a	b	d
Π:	0	0	1	2	0

$i=a$, $j+1=a$, move i and j to right

$i=7$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=1$

P:	a	b	a	b	d
Π:	0	0	1	2	0

$i=b$, $j+1=b$, move i and j to right

$i=8$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=2$

P:	a	b	a	b	d
Π:	0	0	1	2	0

$i=c$, $j+1=a$, mismatch, move j to index(j) of Π value at (0), i exist at same place

$i=8$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=0$

P:	a	b	a	b	d
Π:	0	0	1	2	0

Now since j at 0 , no move further , backward so move i to right (i++)

$i=9$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=0$

P :	a	b	a	b	d
Π :	0	0	1	2	0

$i=a$, $j+1=a$, match move right i and j

$i=10$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=1$

P :	a	b	a	b	d
Π :	0	0	1	2	0

$i=b$, $j+1=b$, match move right i and j

$i=11$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=2$

P :	a	b	a	b	d
Π :	0	0	1	2	0

$i=a$, $j+1=a$, match move right i and j

$i=12$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=3$

P :	a	b	a	b	d
Π :	0	0	1	2	0

$i=b$, $j+1=b$, match move right i and j

$i=13$

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=4$

P :	a	b	a	b	d
Π :	0	0	1	2	0

$i = a$, $j+1 = d$, mismatch move to j at π value (2).

$j=2$

P:	a	b	a	b	d
π:	0	0	1	2	0

$i = a$, $j+1 = a$, match move right i and j

$i=4$

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=3$

P:	a	b	a	b	d
π:	0	0	1	2	0

$i = b$, $j+1 = b$, match move right i and j

$i=5$

String: | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

$j=4$

P:	a	b	a	b	d
π:	0	0	1	2	0

$i = d$, $j+1 = d$, match move right i and j, and i move to out of size stop it

Longest patten = a b a b d
11 12 13 14 15

Algorithm :-

- First creat and calculate the π table
- Take two variable i and j
- Start i=1 and j=0
- check i and j+1
  if match then move i and j to right
  else
  move j to at π value (index).
- if j=0, then dont move further backward so move i right to (i++).
- Rpt step until you find the longest pattern.

Time Complexity :-

Searching :- O(n)

Total = Search + Table :- O(m+n).

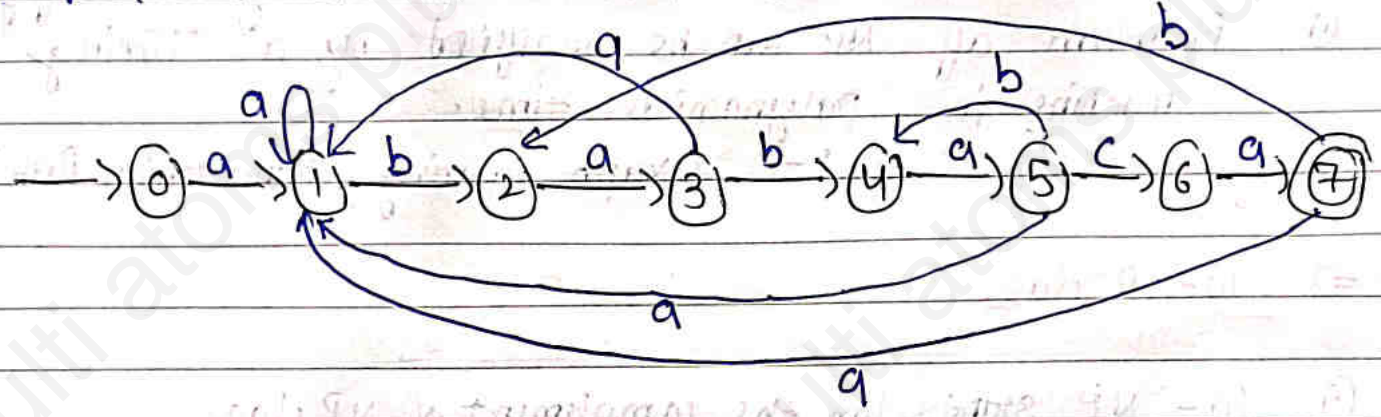
=> String Matching with Finite automata

- There string matching are very efficient to examine each text character exactly one hence time complexity O(n).

Finite automata :- Idea of this approach is to build finite automata to scan text T for finding all occurence of pattern P.

① It is defined by tuple $M = \{Q, \Sigma, q, F, \delta\}$
where Q = set of state in FA
$\Sigma \rightarrow$ set of alphabet
$\delta \rightarrow$ transition mapping
$q_0 \in Q$ initial state
$F \rightarrow$ final state

Eg P : a b a b a c a
T : a b a b a b a c a b a



⇒ Theory of NP - Completeness :- {AKTU 2023-24}

⇒ P - problem :- P is a set of problems that can be solved in polynomial time using deterministic algorithm.

- The solution to P problem is easy to find.
- P is often a class of computational problems that are solvable and tractable.

① Tractable — problem can be solved in practical as well as theory.

① Intractable — that solve in theory but cannot in practical

Eg :- Merge Sort

⇒ NP class

- NP class stands for Non-deterministic Polynomial time
- It is the collection of decision problem that can be solve by a non-deterministic machine in polynomial time.
- The solution of the NP class are hard to find since they are being solved by a non-deterministic machine but the solution are easy to verify.
- Problem of NP can be verified by a Turing machine in polynomial time.

Ex :- Graph coloring, Hamiltonian Path Problem

⇒ CO-NP class

- CO-NP stands for the complement of NP class.
- It means if the answer to a problem in CO-NP, No there is proof that can be checked in polynomial time
- If a problem X is in NP, the its complement X' also in CONP.
- For an NP and CONP problem, there is no need to verify all the answer at once in polynomial time, there is a need to verify one particular answer "yes" or 'no' in polynomial time for a problem to be in NP or CONP

Ex :- Integer-Factorization

⇒ NP hard class

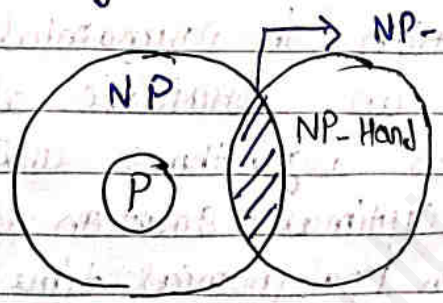
- An NP hard class problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.
- All NP-hard problems are not in NP
- It takes a long time to check them.
- A problem A is in NP-hard, for every problem L in NP, there exists a polynomial time reduction from L to A.

  Ex :- Halting problem, No Hamiltonian cycle

⇒ NP-complete class

- A problem is NP-complete if it is both NP and NP-hard.
- NP-complete problems are the hard problems in NP
- NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.
- If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem in polynomial time

  Ex :- Hamiltonian cycle, vertex cover.



⇒ Randomized Algorithm :- {AKTU 2023-24}

- Randomized algorithm in data structure and algorithm are algorithm that use randomness in their computation to achieved a desired outcome

- These algorithms introduce randomness to improve efficiency or simplify the algorithm design.

- They are particularly useful in situation where exact solution are difficult to find or when a probabilistic approach is acceptable.

Eg Randomized Quick sort

- Improved efficiency ① Simpler Design ① Probabilistic Guarantees

⇒ Approximation Algorithm :- {AKTU 2022-2023}

- These algorithm are ways to deal with NP-completeness for an optimization problem
- Does not gurantee the best explanation
- The goal is come as close as possible to the optimal solution in polynomial time
- Also called as heruistic algorithm
- Approximation algorithms are used to get an answer near the (optimal) solution of an optimization problem in polynomial time.

UNIT-5 COMPLETE

LIKE COMMENT & SHARE

SUBSCRIBE ~ MULTI ATOMS & PLUS

multi atoms plus