Unit-1

Analysis of Insertion Sort

Insertion Sort (A, n) — Frequency

for i = 2 to n — n

{

Key = A[i] — n-1

j = i-1 — n-1

while (j>0 and A[j] > Key) — $\sum_{i=2}^{n} t_i$

{

A[j+1] = A[j]

i = j-1

}

A[j+1] = Key — n-1

}

Best Case :-

If array is already sorted

$t_i = 1$ for i=2 to n

$$\sum_{i=2}^{n} t_i = \sum_{i=2}^{n} (1) = n-1$$

$$\sum_{i=2}^{n} (t_i - 1) = \sum_{i=2}^{n} (0) = 0$$

$$S_n = n + 3(n-1) + (n-1) + 0$$

$$= 5n - 4$$

$$= \Theta(n)$$ [For Highest Power of n]

Worst Case :-

If array is reverse sorted
$t_i = 1$. for $i = 2$ to n

$$\sum_{i=2}^{n} (t_i) = \sum_{i=2}^{n} i = 2+3+4+\ldots+n = \frac{n(n+1)}{2} - 1$$

$$t_{i-1} = i-1$$

$$\sum_{i=2}^{n} (t_i - 1) = \sum_{i=2}^{n} (i-1) = \frac{n(n-1)}{2}$$

$$S_n = n + 3(n-1) + \frac{n(n+1)}{2} - 1 + \frac{2n(n-1)}{2}$$

$$= n + 3n - 3 + 2n^2 + 2n - 2 + n^2 - n$$

$$= 3n^2 + 5n - 5 \qquad [An^2 + Bn + C]$$

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

Average Case :-

$$t_i = i/2$$

$$\sum_{i=2}^{n} (t_i) = \sum_{i=2}^{n} (i/2) = \frac{1}{2} \sum_{i=2}^{n} (i) = \frac{1}{2}\left[\frac{n(n+1)}{2} - 1\right]$$

$$\sum_{i=2}^{n} (t_i - 1) = \sum_{i=2}^{n} (i/2 - 1) = \frac{1}{2} \sum_{i=2}^{n} (i-2) = \frac{1}{2}\left[\frac{(n-1)(n-2)}{2}\right]$$

$$S_n = n + 3(n-1) + \frac{1}{2}\left[\frac{n(n+1)}{2} - 1\right] + \frac{\cancel{2}}{\cancel{2}}\left[\frac{(n-1)(n-2)}{2}\right]$$

$$= A'n^2 + B'n + C'$$

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

Merge Sort

Divide and Conquer-

1. **Divide :-**
   Divide the problem into sub-problem of smaller size.

2. **Conquer :-**
   Solve the subproblem recursively until the problem size is small enough that can be solved directly.

3. **Combine :-**
   Combine the solution of the sub problem to get the final solution.

Algorithm :-

```
mergesort (A, p, r)
 if (p<r)
 {
    q = [(p+r)/2]

  merge sort (A, p, q)
  merge sort (A, q+1, r)
  merge (A, p, q, r)
 }

 Merge (A, p, q, r)
   n1 = q - p + 1
   n2 = r - q
```

Define two arrays $L[1 \ldots n_1+1]$ & $R[1 \ldots n_2+1]$

$\left.\begin{array}{l} \text{for } (i=1 \text{ to } n_1) \\ \quad L[i] = A[p+i-1] \end{array}\right\} \theta(n_1)$

$\left.\begin{array}{l} \text{for } (j=1 \text{ to } n_2) \\ \quad R[j] = A[q+j] \end{array}\right\} \theta(n_2)$

i=1, j=1

for (k=p to r)

{

if $L[i] \le R[j]$

{ $A[k] = L[i]$

i=i+1

}

{ $A[k] = R[j]$

j=j+1

}

}

Complexity :-

Merge = $\theta(n_1) + \theta(n_2) + \theta(n)$

$= \theta(n_1+n_2+n)$

$= \theta(2n)$

$= \theta(n)$

Merge Sort -

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

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

$$T(n) = \begin{cases} 2T(n/2) + cn & \text{if } n > 1 \\ d & \text{if } n = 1 \quad [\text{constant}] \end{cases}$$

Let θ be constant

$T(n) = 2[2T(n/4) + cn/2] + cn$

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

$T(n) = 4[2T(n/8) + c\,n/4] + 2cn$

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

$= 8T(n/8) + 3cn$

$= 2^3 T(n/2^3) + 3cn$

$\vdots$

$= 2^k T(n/2^k) + kcn$

if $n = 2^k$ [if $n/2^k = 1$ for constant]

$\log n = k \log 2$ [taking log base 2]

$\log_2 n = k$

$= 2^{\log_2 n} T(1) + \log_2 n \, cn$

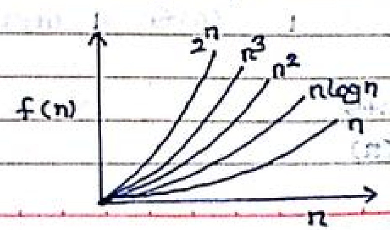
$= nd + cn \log_2 n$

$\theta(n \log_2 n)$

Time Complexity:-

- Time complexity of any algorithm is the rate of growth of time if we increase the no. of i/p n in the algorithm.
- It is represented by a ∞ in term of n.

Eg:- $\theta(n)$, $\theta(n \log n)$, $\theta(n^2)$, $\theta(2^n)$, $\theta(n^3)$



• Complexity on higher value.

Analysis of Algorithm

Eg - Sum (A,n)

```
    s=0
    for i=1 to n
        s=s+ A[i]
    return s
```

Steps/Execution (S/e)	Frequency	Steps
1	1	1
1	n+1	n+1
1	n	n
1	1	1
		2n+3

Time = Total Steps x C

time x T. steps

T(n) = T. Steps

T(n) = 2n+3

if $n \to \infty$ So T(n) = 2n

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

	S/e	Frequency	Steps
N Sum (A,n)			
for (i=1 to n)	1	n+1	n+1
Read A[i]	1	n	n
(Sum (A,n)		2n+3	2n+3
return s	1	1	1
			4n+5

T(n) = 4n+5

= $\Theta(n)$

By Using Recursion :-

```
RSum (A,n)
    if n=0
        return 0
    else
        return RSum (A,n-1) + A[n]
```

Complexity :-

S/e	Frequency n=0	Frequency n>0	Steps n=0	Steps n>0
1	1	1	1	1
1	1	0	1	0
1	0	0	0	0
$S_{n-1}+1$	0	1	0	$S_{n-1}+1$

$$S_n = \begin{cases} S_{n-1}+2 & \text{if } n>0 \\ 2 & \text{if } n=0 \end{cases}$$

$S(n) = (S_{n-2}+2)+2$

$= S_{n-2}+4$

$= (S_{n-3}+2)+4$

$= S_{n-3}+6$

$= S_{n-k}+2k$

when n=k

$= S_{n-n}+n(2)$

$= 2+2n$

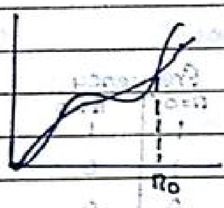
$T(n) = \Theta(n)$

Asymptotic Notations:-

1. O-Notation [Big Oh-Notation]:- or Upper Bound

$f(n) = O(g(n))$ if there exist two positive constants c and n_0 such that

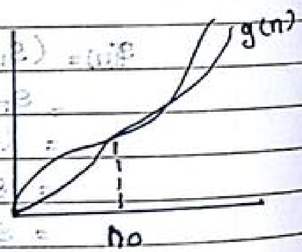
$0 \le f(n) \le c\,g(n) \quad \forall\, n \ge n_0$



2. Ω Notation [Big Omega Notation]:- [Lower Bound]

$f(n) = \Omega(g(n))$ if there exist two positive constants c and n_0 such that

$0 \le c\,g(n) < f(n) \quad \forall\, n > n_0$



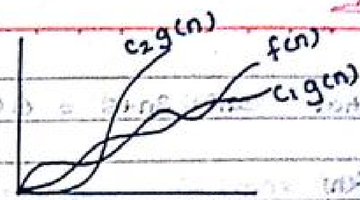
Eg- $2n^2+3n+5 \ge 2n^2 \quad \forall\, n \ge 1$

$c = 2,\ n_0 = 1$

3. Θ-Notation :- Tightly Bound:

$f(n) = \Theta(g(n))$ if there exist three positive constant c_1, c_2 and n_0 such that

$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall\, n \ge n_0$



4. o-Notation [Small oh or little oh]:-

$f(n) = o(g(n))$ if for every positive constant c there exist a positive constants n_0 such that

$0 \le f(n) \le c\,g(n) \quad \forall\, n \ge n_0$

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

$g(n) = n^3$

$2n^2+3n+5 < cn^3 \quad \forall\, n \ge n_0$

$2n^2+3n+5 < \frac{1}{105} n^3$

5. ω-Notation [Small Omega]:-

$f(n) = \omega(g(n))$ if for every positive constant c there exist a positive constant n_0 such that

$0 \le c\,g(n) < f(n) \quad \forall\, n \ge n_0$

Methods of Limits to Prove Asymptotic Bounds Only Given Function

1. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$
2. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{constant}$ then $f(n) = \Theta(g(n))$
3. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ then $f(n) = \omega(g(n))$

Ex- Prove that $2n^2+3n+5 = \Theta(n^2)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$\lim_{n\to\infty} \frac{2n^2+3n+5}{n^2}$$

$$\lim_{n\to\infty} 2+\frac{3}{n}+\frac{5}{n^2}$$

$= 2$ constant

$2n^2+3n+5 = \Theta(n^2)$

Ex- Prove that $2n^2+3n+5 = o(n^3)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$\lim_{n\to\infty} \frac{2n^2+3n+5}{n^3}$$

$$\lim_{n\to\infty} \frac{2}{n}+\frac{3}{n^2}+\frac{5}{n^3}$$

$= 0$

$\therefore$ $2n^2+3n+5 = o(n^3)$

Ex- Prove that $(n+a)^b = \Theta(n^b)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$\lim_{n\to\infty} \frac{(n+a)^b}{n^b}$$

$$\lim_{n\to\infty} \frac{n^b(1+a/n)^b}{n^b} = 1$$

$\therefore$ $(n+a)^b = \Theta(n^b)$ $\Rightarrow$ $(n+a)^b = \Theta(n^b)$

Ex. Prove that $2^{n+1} = O(2^n)$ True / False ✓

Ex. Prove that $\underbrace{f(n)+g(n)}_{F(n)} = \Theta(\underbrace{\max(f(n),g(n))}_{G(n)})$

$f(n)+g(n) \le c_1 \max(f(n),g(n))$ $\forall$ $n \ge n_0$

Let $c_1 = 2$

$f(n)+g(n) \le 2 \max(f(n),g(n))$ $\forall$ $n \ge 1$

$f(n)+g(n) \ge c_2 \max(f(n),g(n))$ $\forall$ $n \ge n_0$

Let $c_2 = 1$

$f(n)+g(n) \ge 1 \max(f(n),g(n))$ $\forall$ $n \ge 1$

QuickSort :-

Partition (A,p,r)	Frequency
i = p-1	1
key = A[r]	1
for (j = p to r-1)	n
{	
if A[j] < key	n-1
{	
i = i+1	n-1
Swap A[i] ↔ A[j]	3(n-1)
}	
}	
Swap A[i+1] ↔ A[r]	3
return (i+1)	1
	$6n+1 = \Theta(n)$

```
Quicksort (A,p,r)
  if (p<r)
  {
    q = Partition (A,p,r)
    Quicksort (A,p,q-1)
    Quicksort (A,q+1,r)
  }
```

Complexity :-

Time complexity of partition $T(n) = \Theta(n)$

Time Complexity of Quick Sort

$T(n)$ = time in partition + Time in recursive call + Time in combining results.

Time complexity of Quicksort is depend on the sequence of given input. So we should do not the case analysis here.

Best case :-

If the partitioning is perfectly balanced in each step then the algorithm will take minimum time and so it will called the best case.

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

Using Iteration method

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

Worst case :-

If every time all the elements go either side / one side of the Pivot then this type of case is called Worst case.

In Quicksort, if array is sorted or reverse sorted then it will be the worst case.

$T(n) = T(n-1) + T(0) + \Theta(n)$

Let $T(0) = d$

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

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

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

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

$\vdots$

$= T(0) + c[1+2+3+\dots\dots n]$

$= d + c\left[\frac{n(n+1)}{2}\right]$

$= \Theta(n^2)$

Time Complexity of worst case in Quick Sort.

Average Case :-

Let in the Average case the partitioning is in the ratio 10:90.

$T(n) = T(n/10) + T(9n/10) + \Theta(n)$

From the recurrence tree method

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

Randomized Quicksort :-

```
Randomized_Partition (A,p,r)
        K = Random (p,r)
    swap A[K] <-> A[r]
    Key = A[r]; i=p-1
        for(J=p to r-1)
            if A[j] < Key
            {
                i=i+1
                swap A[i] <-> A[j]
            }
            swap A[i+1] <-> A[r]
            return (i+1)

Randomized_Quick_sort (A,p,r)
  if (p<r)
  {
    q= Randomized_Partition (A,p,r)
    Randomized_Quick_sort (A,p,q-1)
    Randomized_Quick_sort (A,q+1,r)
  }
```

In QuickSort algorithm, if array is sorted or reverse sorted then it will become the worst case and the complexity in the worst case is $\Theta(n^2)$.

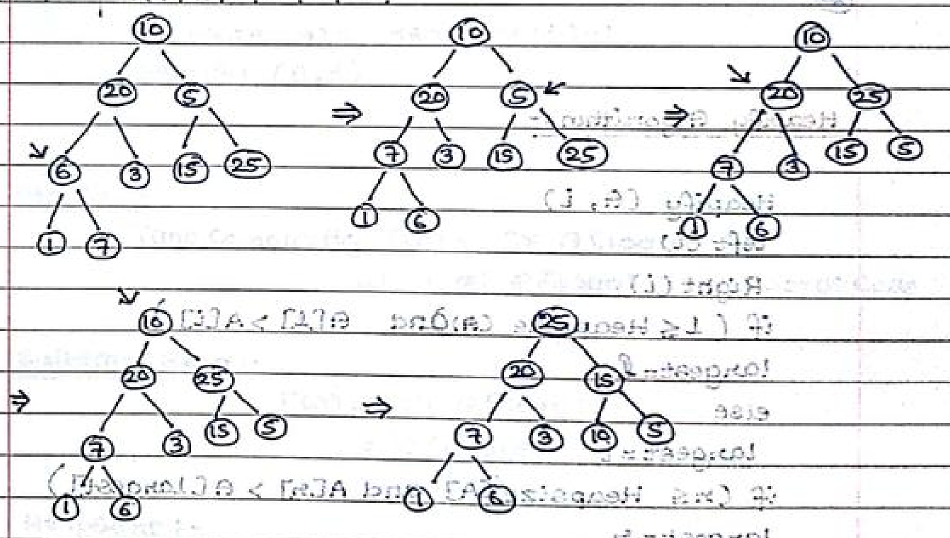
We can overcome this problem of the Quicksort if we select the pivot Randomly then the chance of worst case will be negligible bcoz the probability of the selection of largest element in every step is $\frac{1}{n} \cdot \frac{1}{n-1} \cdots \frac{1}{1} = \frac{1}{n!}$

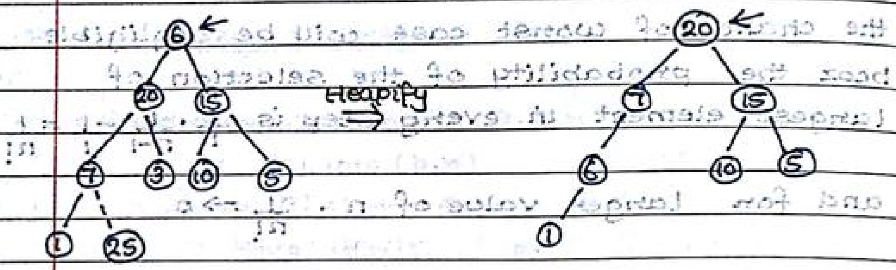
and for larger value of n. $\frac{1}{n!} \to 0$

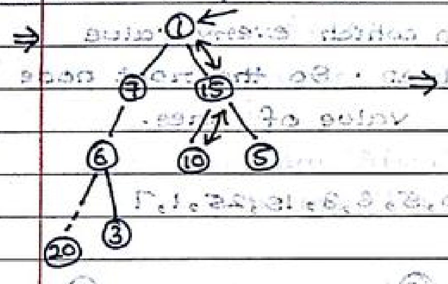
Heap Sort :-

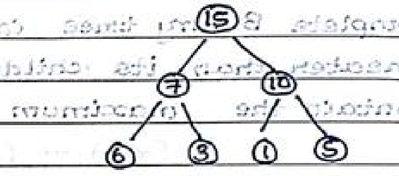
A heap or max heap is a nearly complete Binary tree in which every value greater than its children. So the root node contain the maximum value of tree.

Given array 10, 20, 5, 6, 3, 15, 25, 1, 7









15	7	10	6	3	1	5	20	25

Heapify Algorithm :-

```
Heapify (A, i)
 left (i)
 Right (i)
 if (l ≤ Heapsize (A) and A[l] > A[i])
 largest = l
 else
  largest = i
 if (r ≤ Heapsize [A] and A[r] > A[largest])
 largest = r
 if (largest ≠ i)
 swap A[i] ⟷ A[largest]
 Heapify (A, largest)
```


Build max Heap Algo :-

```
Build_max_Heap (A)
 Heapsize (A) = length (A)
 for (i = ⌊length(A)/2⌋ downto 1)
 Heapify (A, i)
```

$\Theta(n)\log n$

Heapsort Algorithm :-

```
Heap-Sort (A)
Build_max_heap (A)
for (i = length (A) down to 1)
{
    A[1] ⟷ A[i]
   Heapsize (A) = Heapsize (A) - 1
   Heapify (A, 1)
}
```

Heapify :-

Time Complexity $T(n) = c \times \Theta(\log n)$

$\Theta(\log n)$: worst Case

Build Max Heap :-

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

$= \Theta(n \log n)$

HeapSort :-

$T(n) = \Theta(n\log n) + (n-1) \times \Theta(\log n)$

$= \Theta(n \log n)$

Scanned by CamScanner

Recurrence:-

1. Iteration Method :-

$$T(n) = \begin{cases} 2T(n/2) + cn & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

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

$= 2[2T(n/4) + \frac{cn}{2}] + cn$

$= 2^2 T(n/4) + 2cn$

$= 2^2[2T(n/8) + \frac{cn}{4}] + 2cn$

$= 2^3 T(n/8) + 3cn$

$\vdots$

$= 2^K T(n/2^K) + K\,cn$

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

$2^K = n$

$K = \log_2 n$

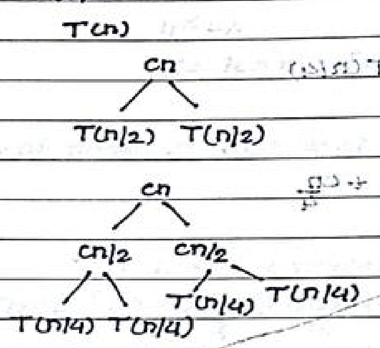
$T(n) = nT(1) + \log n \cdot cn$

$= dn + cn\log n$

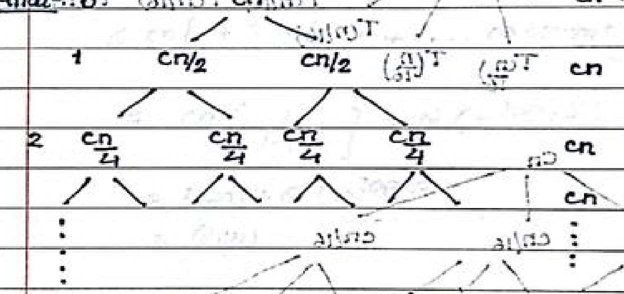
$= \Theta(n\log n)$

2. Recurrence Tree Method :-

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



Final:-



K $T(1) + T(1) \ldots\ldots\ldots T(1) = dn$

at Kth level $T(n/2^K) = T(1)$

Height of tree $= K$

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

$2^K = n$

$K = \log_2 n$

No. of nodes at Kth level $= 2^K = n$

So

$T(n) = (cn + cn + cn + \ldots$ for $\log n$ times $) + dn = cn\log n + dn$

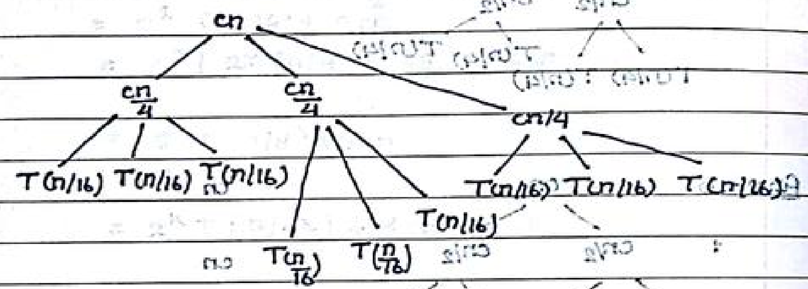
$= \Theta(n\log n)$

Ques- Solve Using Recurrence :-

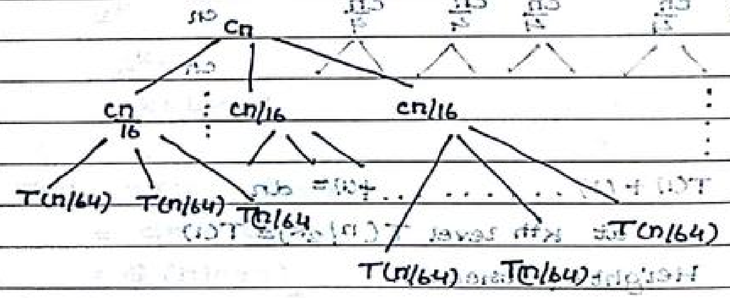
$T(n) = 3T(n/4) + cn$

Put $n = n/4$ in eq-

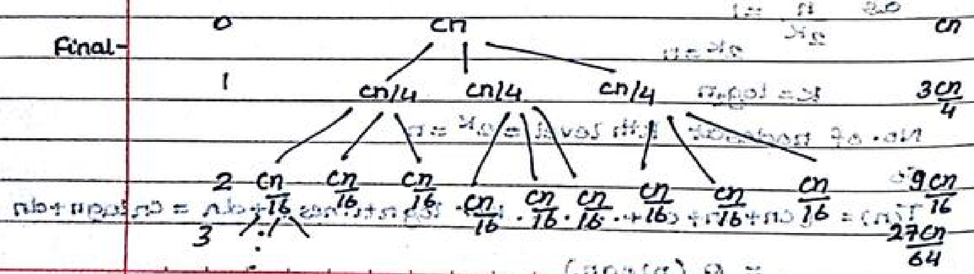
$T(n/4) = 3T(n/16) + \frac{cn}{4}$



$n = n/16$



Final-



$K\ T(1) + T(1) \ldots\ldots\ T(1)$

At Kth level

$\frac{n}{4^K} = 1$

$4^K = n$

$K = \log n$ [Height]

No. of nodes at last level $= 3^K$

$= 3^{\log_4 n}$

$= n^{\log_4 3}$

$\because$ value of 1 node $= d$

$T(n) = \left[cn + \frac{3cn}{4} + \frac{9cn}{16} + \frac{27cn}{64} + \ldots K\text{ times}\right] + n^{\log_4 3} \times d$

$T(n) = cn\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \ldots K\text{ times}\right] + \theta(n^{\log_4 3})$

$\leq cn\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \ldots \infty\text{ terms}\right] + \theta(n^{\log_4 3})$

$= cn\left[\frac{1}{1-3/4}\right] + \theta(n^{\log_4 3})$

$= 4cn + \theta(n^{\log_4 3})$

$= O(cn)$ [O = Big Oh]

For θ-Notation:-

$T(n) = cn\left[\frac{1-(3/4)^{\log_4 n}}{1-3/4}\right] + \theta(n^{\log_4 3})$

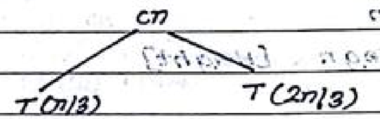
$= 4cn\left[1-\left(\frac{3}{4}\right)^{\log_4 n}\right] + \theta(n^{\log_4 3})$

if $n \to \infty$ $\left(\frac{3}{4}\right)^{\log_4 n} \to 0$

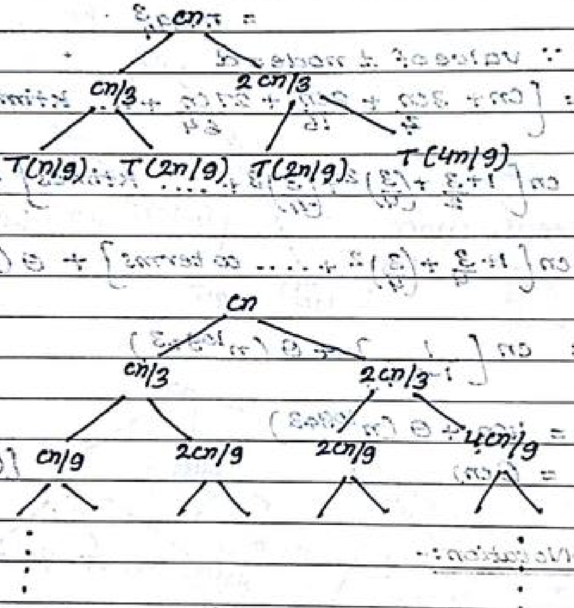
$T(n) = 4cn + \theta(n^{\log_4 3})$

$= \theta(n)$

Ques:- $T(n) = T(n/3) + T(2n/3) + cn$



$n = n/3$



cn

cn

cn

$\leq cn$

$\leq cn$

Height of Tree is

$T\left(\frac{2^k n}{3^k}\right) = T(1)$

$2^k n = 3^k$

$\left(\frac{3}{2}\right)^k = n$

$k = \log_{3/2} n$

So height $= \log_{3/2} n$

$T(n) \leq (cn + cn + \ldots + \log_{3/2} n \text{ times})$

$= cn\log n$

$= O(n\log n)$

$T(n) \geq (cn + cn + \ldots \log_3 n \text{ times})$

$= cn\log_3 n$

$= \Omega(n\log n)$

m imp

3. Master Theorem :-

It can applied only Recurrence is the form of

$T(n) = aT(n/b) + f(n)$

where $a \geq 1$, $b > 1$

- If $f(n) = O[n^{\log_b a - \epsilon}]$ for some $\epsilon > 0$
  then $T(n) = \Theta(n^{\log_b a})$

- If $f(n) = \Theta[n^{\log_b a}]$ then
  $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

$f(n) >$

- If $f(n) = \Omega[n^{\log_b a + \epsilon}]$ for some $\epsilon > 0$
  and $af(n/b) < cf(n)$ for some $c < 1$
  then $T(n) = \Theta(f(n))$

Ques- Solve by master's Theoram.

$T(n) = 5T(n/2) + cn^2$

$a=5,\ b=2,\ f(n) = cn^2$

$= \theta(n^2)$

$n^{\log_b a} = n^{\log_2 5} > f(n)$

From ~~Using~~ Case-1 of Master method may be apply

$f(n) = \theta(n^2)$

$= O(n^2)$

$= O(n^{\log_2 5 - \epsilon})$

It will be true if

$\log_2 5 - \epsilon = 2$

$\epsilon = \log_2 5 - 2 > 0$

From Case-1 of Master Theoram

$T(n) = \theta(n^{\log_2 5})$

07 Aug. 2018

Ques- $T(n) = 4T(n/2) + \theta(n^2)$ using masters method.

$a=4,\ b=2,\ f(n) = \theta(n^2)$

$n^{\log_b a} = n^{\log_2 4} = n^2 = \theta(n^2)$

Case-2 canbe applied

$f(n) = \theta(n^2)$

$= \theta(n^{\log_2 4})$

So by case-2 of masters method

$T(n) = \theta(n^2 \log n)$

Example- $T(n) = 3T(n/4) + cn^2$

$a=3$

$b=4$

$f(n) = cn^2$

$= \theta(n^2)$

$n^{\log_b a} = n^{\log_4 3} < f(n)$

Case-3 of masters method can be applied.

$f(n) = cn^2$

$\left.\begin{array}{l} = \theta(n^2) \\ = \Omega(n^2) \end{array}\right\}$ (Given)

$= \Omega(n^{\log_4 3 + \epsilon})$

It will be True if $\log_4 3 + \epsilon = 2$

i.e., $\epsilon = 2 - \log_4 3 > 0$

$af(n/b) = 3f(n/4)$

$= 3\frac{cn^2}{16}$

$= \frac{3}{16}cn^2$

$= \frac{3}{16}f(n) < cf(n)$

It will be true

if $\frac{3}{16} < c < 1$

From case-3 of Master methods

$T(n) = \theta(f(n))$

$T(n) = \theta(n^2)$

Ques:- $T(n) = 2T(n/2) + n\log n$

$a = 2$ $f(n) = n\log n$

$b = 2$

$n^{\log_b a} = n^{\log_2 2} = n^1 < n\log n$

Case-3 can be applied. >

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

$= \Omega(n\log n)$

$= \Omega(n^{\log_2 2 + \epsilon})$

$n\log n = n^{1+\epsilon}$

$\cancel{n}\log n = \cancel{n}\cdot n^{\epsilon}$

$\log n = n^{\epsilon}$

$\epsilon = \log_n \log n$

From here we can not find out constant value of ϵ.
This is the case in b/w case-2 and case-3 of the master method.
So we cannot apply master method on this question.

Ques:- $T(n) = 3T(n/4) + n\log n$

$a = 3$

$b = 4$

$f(n) = n\log n$

$n^{\log_b a} = n^{\log_4 3} < n\log n$

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

$= \Omega(n\log n)$

$= \Omega(n)$

$= \Omega(n^{\log_4 3 + \epsilon})$

It will be true if

$\log_4 3 + \epsilon = 1$

$\epsilon = 1 - \log_4 3 > 0$

$3f(n/4) = 3n/4 \log n/4$

$\frac{3}{4} n(\log n - \log 4)$

$\frac{3}{4} n(\log n - 2)$ ... and so on

$\frac{3}{4} n\log n - \frac{3}{2} n$

$\frac{3}{4}(f(n) - 6n) < cf(n)$

$c = 3/4 < 1$

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

Stable Algorithm:-

- The sorting algorithm is called stable if there are two or more elements with the same value in the input sequence and after sorting the order of these elements will be same as it is in the input sequence.

For Ex:- i.e. the element that appear first in the input sequence should be 1st in the out sequence also and so on....

- Counting Sort is a Stable Algorithm

Radix Sort:-

Radix_Sort (A,n,d)

for i=1 to d

Apply Stable Sort (counting Sort) Algorithm to the element according to digit i

$$T(n) = d \times \Theta(n+k)$$

Here K is constant
if d is constant

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

$$\begin{bmatrix} 0-0.2 \\ 0.2-0.4 \\ 0.4-0.6 \\ 0.6-0.8 \end{bmatrix}$$

Bucket Sort:-

Algo:-

Bucket_Sort (A,n)

1. for i=1 to n
   Insert A[i] in the list at LIST $[\lfloor A[i] * n \rfloor]$
2. for i=0 to n-1
   Sort the list LIST[i] using insert sort.
3. Concatenate the lists LIST[0], LIST[1] ..... LIST[n-1] in the order

Time Complexity, $T(n) = \Theta(n) + \Theta(n) + \Theta(n)$
$= \Theta(n)$

Bcoz the no. of element in each list are constant So, the time required to sort the listed using insertion sort will be $\Theta(1)$.
So, the Time Complexity for 2nd for loop will be $\Theta(n)$

Note:- In this Algorith we will always consider the no. of Buckets equal to the no. of elements & the elements should be uniformly distributed in the whole range. In above algo we are considering our range is from [0-1).

- If range is given from 0 to K, then the formula will be.

$$LIST\left[\left\lfloor \frac{A[i] * n}{k} \right\rfloor\right]$$

- For Range $\overset{K_1}{100}-\overset{K_2}{200}$
  $n=10$

$$\left\lfloor \frac{(A[i]-K_1)*n}{K_2-K_1} \right\rfloor$$

Range is Given from K_1 to K_2 then formula is above.

Median:-

- If n is odd then median is $\frac{n+1}{2}$ th smallest element
- If n is even then medians are $\frac{n}{2}$ th and $(\frac{n}{2}+1)$ th smallest element

Lower median Upper median

Order Statistics:-

ith order statistics of a given group of element is the ith smallest element of the group. So the minimum element is called 1st order and max. element called nth order statistics.
[n= size of the group]

Example. 20, 5, 10, 6, 8, 12, 30, 25, 15. (12) [i= 5]
5, 10, 6, 8, 12, 30, 20, 25, 15, (12)
5, 10, 6, 8, 2, (12) 20, 25, 15, 30
5, 10, 6, 8, (2).
(2) 10, 6, 8, 5
10, 6, 8, (5)
(5) 6, 8, 10
6, 8, (10)

13 Aug. 2018

Algorithm:-

```
Order-stats (A,p,r,i)
 if p=r
     return A[p]
 else
     q= Randomized_Partition (A,p,r)
     r= q-p+1
     if (i=r)
 return A[q]
 else
     if (i<r)
       Order-stats (A,p,q-1,i)
 else
       order-stats (A,q+1,r,i-r)
```

Median_Stats :-

```
Median (A,n)
  if n is odd
        i= (n+1)/2
  else
        i = n/2
  K= Order-stats (A,1,n,i)
  return K
```


imp # To find maximum and minimum in Array:-

Algo:-

```
     Max-min (A,n)
if n is odd
     min = A[1]
     max = A[1]
     K=2
  else
     if (A[1] < A[2])
       min= A[1]
       max= A[2]
     else
       min= A[2]
       max= A[1]
     K=3
   for (i=k to n-1 step 2)
    {
       if (A[i] < A[i+1])
       {
          if(A[i] < min)
             min = A[i]
          if A[i+1] > max)
             max= A[i+1]
       }
     else
          if (A[i+1] < min)
             min= A[i+1]
          if (A[i] > max)
             max= A[i]
```

14 Aug. 2018

Unit-2

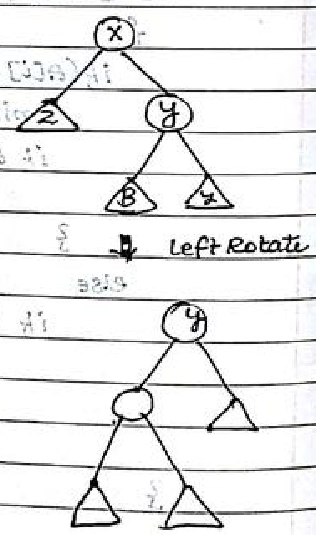
Red Black Tree [RB Tree]:-

It is a Binary Search tree with an additional field called colour. Any Binary Search tree will be an RB-Tree if it fulfill the following properties.

1. The root node is always black.
2. Every node is either Red or Black.
3. The leaf node is always black.
4. The children of red node are always black.
5. For any node (N), the no. of black nodes in each path from the node N to the leaf are always equal.

Insertion:-

Left Rotate:-




```
y = Right [x]
P[y] = P[x]
if ( P[x] = NULL)
{
    Root [T] = y
}
else
{
    if (x = left [P[x]])
        left [P[x]] = y
    else   Right [P[x]] = y
}
```



```
Right [x] = left [y]
If ( left [y] ≠ NULL)
    P [Right [x]] = x
left [y] = x
P[x] = y
```

Black Height:-

The Black height of any node in RB-Tree is equal to no. of black nodes in the path from the root node to the leaf except node.

height = highest no. of level
or
(no. of Level - 1)

- The Black height of the Root of the RB Tree is called height of RB-Tree.

- For any n-node Red Black Tree, the height h is atmost is twice $2\log(n+1)$

$2\log(n+1)$ i.e., the height

$h \leq 2\log(n+1)$

where n is the number of nodes.

16 Aug 2018

Lemma:-

For any n-nodes of RB Tree, the height is atmost $2\log(n+1)$

i.e. height $h \le 2\lg(n+1)$ where n = no. of internal nodes

imp

Proof:-1. To Prove the Lemma first we need to proof that the no. of internal node in any subtree rooted at x is atleast $2^{bh(x)}-1$ where $bh(x)$ is the black height of the node x.

Let the black height of the node x, $bh(x)=0$

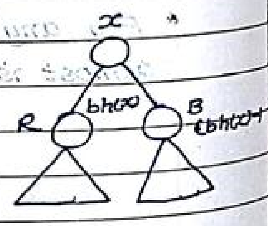
So the no. of internal nodes $= 2^{bh(x)}-1 = 2^0-1 = 0$

and it is true bcoz black heigh can be 0 if there is no internal node

that is x is leaf node

Now: let the formula is true for $bh(x)-1$

If the Black height of the nodes x is $bh(x)$ then black height of its child is either $bh(x)$ or $bh(x)-1$ if the colour of child is Red or Black respectively.



that means the black height of the children of nodes x is atleast $bh(x)-1$.

So the number of Internal node is the subtree at rooted at nodes x, is $\ge 2(2^{bh(x)-1}-1)+1$

$= 2^{bh(x)}-1$

So, the formula is true for $bh(x)$ if it is true for $bh(x)-1$

So, By the Induction method this formula is true for all the nodes.

Second: For any Red Black Tree, if height of the Red Black Tree is H, then its black height is atleast $h/2$ because the children of Red node in RB Tree is always black, So we cannot have to continuous Red nodes in any path that's why atleast half of the nodes in any path should be black.

So the no. of internal nodes in the RB tree of height h is atleast

$(2^{h/2}-1)$

i.e, $n \ge 2^{h/2}-1$

$\Rightarrow 2^{h/2} \le n+1$

taking log on both sides

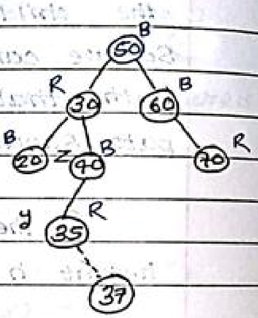
$\Rightarrow \frac{h}{2} \le \log(n+1)$

$\Rightarrow h \le 2\log(n+1)$

Inorder:-

```
Inorder (x)
if (x ≠ NIL)
{
  Inorder( left [x])
  print ( key[x])
  Inorder ( Right [x])
}
```

Insertion in Red Black Tree :-




```
RB_Insert (T,z)
x = Root [T]
y = nil [T]
while( x ≠ nul [T])
{
  y = x
  if ( key[z] < key [x])
      x = left [x]
  else
      x = Right [x]
}
if ( y = nil [T] )
    Root [T] = z
else
{
  if(key [z] < key [y])
     Left[y] = z
  else
     Right [y] = z
}
P[z] = y
Left[z] = nul [T]
Right [z] = nul [T]
Color [z] = Red
```

→ RB-Insert Fixup (T,z)

imp Numerical # Procedure for Insertion and Arbitary:-

Step-1:- Insert a new nodez, in the same as in BST

Step-2:- Give Red Colour to the Inserted nodz and called RB-Insert Fixup.

Step-3:- If colour of parent of z = Black then do nothing and stop the procedure.

Step-4:- If colour of parent of z is also Red then find out the Uncle of z [Sibbling of Parent of z] and denote it by y

Step-5:- If y is the Right child of its parent then apply one of the following 3 case on the basis of colour of y.

Case 1:- If Colour of y is Red then apply case 1 & do the following in this case.

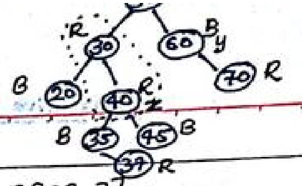
Color [y] = Black

Color [P[z]] = Black

([Color [P[P[z]]] = Red

(y = Uncle)

Case-1 →



z = P[P[z]]

go to step-(3)

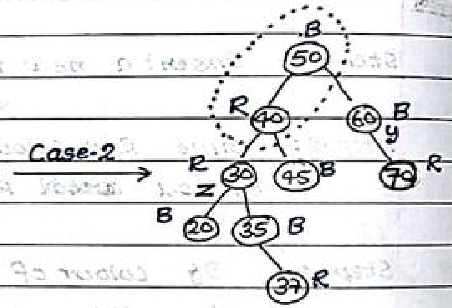
18 Aug. 2018

Case-2: If colour of y is Black and z is the right child of its parent then case-2 will be applied. and we will do the following in this case.

z = P[z]

left-Rotate [T, z]

Case-2 →



Case-2 will not solve the problem but it will Rotate the tree in such a way so that we can apply Case-3 on it.

After applying case-2, case-3 must be applied.

Case-3:-

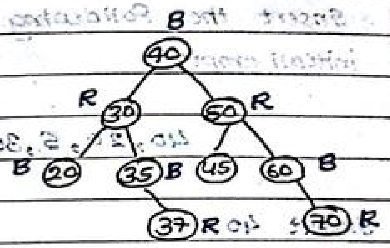
If colour of y is Black and z is the left child of its parent then Case-3 will be applied and we will do the following in this case.

Colon [P[z]] = Black

Colon [P[P[z]]] = Red

and then Right-rotate [T, P[P[z]])

Case-3 →



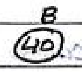
Step-6:- If due to above Rotations/Cases if Root become Red then make it Black and stop.

Step-7:- If y is the left child of its parent then Same steps will be applied but exchange left by Right [z is Right then Case-3 otherwise Case-

Example- Insert the following key in RB-Tree which is initiall empty.

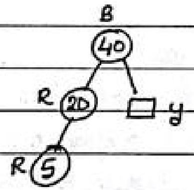
40, 20, 5, 30, 35, 10, 37, 7,

Insert 40



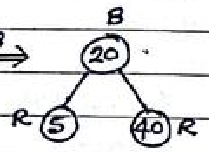
Insert 20

Insert 5

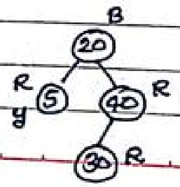


By Apply Case-3

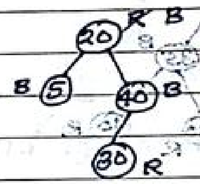
$\xRightarrow{\text{Case-3}}$



Insert 30 -

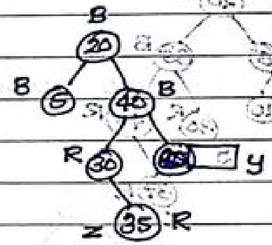


y is Red then Apply Case-1

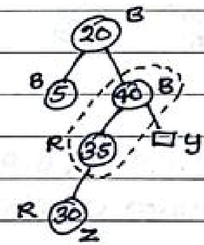


[Red Become Black]

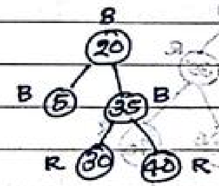
Insert 35



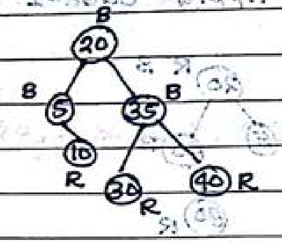
Using Case-2 we get



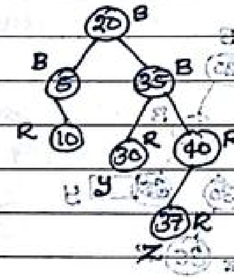
Now again apply Case-3 in above tree.



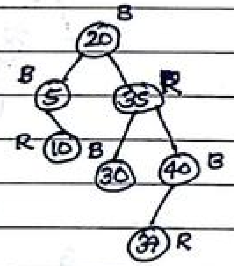
Insert 10



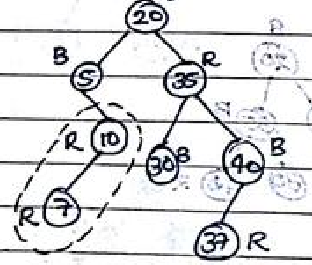
Insert 37



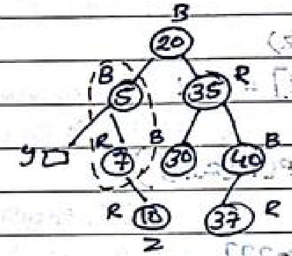
y is Red then Apply Case-1 [Reverse]



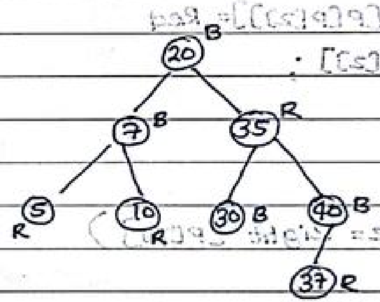
Insert 7



By appling Reverse Case-2



By appling Reverse case-3



Ques 10, 9, 8, 1, 2, 3, 4, 5, 6, 7 Insert them in RB-Tree which is initally empty.

Algorithm:-

```
RB Insent_Fixup (T,z)
while  colon [P[z]] = Red
{
 if ( P[z] = left [P[P[z]]] )
   {
     y= Right[P[P[z]]]
     if  colour [y]=Red
→     {
         colon [P[z]]= Block
         colon [y]= Black
         colon [P[P[z]]]= Red
         z= P[P[z]] ;
      }
     }
     else
      {
         if (z= Right [P[z]] )
→        {
           z= P[z]
           Left Rotate (T,z)
         }
→        {colon [P[z]] = Black
          Colon [P[R[z]]] = Red
          Right Rotate (T, P[P[z]] )
         }
      }
     else
       { same as If clause  just exchange
            left and night.
       }
  }
  Colon [Root [T]] = Black .
```


C

Deletion in RB-Tree:-

```
Tree - Successon  (T,x)
If ( Right [x] ≠ nil [T] )        Condition { when Right child exist }
 {
     y = Right [x]
    while ( Left [y] ≠ nil [T] )
        y = left [y]
 }
else
 { while (x= Right [P[x]])
        x= P[x]
        y = P[x]
 }
Return y
```

21 Aug. 2018

```
RB - Deletion (T,z)
If (left [z] = nil [T]  on  Right[z]=nil[T])
        y=z
else
     y= Tree Successon (T,z)
If  (left [y] ≠ nil [T])
    x= left [y]
else
    x = Right [y]
  If x ≠ nil [T]
    P[x] = P[y]
If (P[y]= nil [T])
    Root[T]=x
```

```
else
{
   if (y = left [P[y]])
       left [P[y]] = x
   else
       Right [P[y]] = x
}
if (z ≠ y)
   Key [z] = Key [y]

If colour [y] = Black
   RB_Delete_Fixup (T, x)
```

Procedure For Deletion in RB-Tree:-

Step-1 Delete the node z in the same way as in BST. The node that we have removed from the tree [y] is either the node z or its successor.

Whatever the case, the problem will appear due to removal of y.

Step-2 If color of y is Red then we need to do nothing.

Step-3 If Color of y is Black then will call RBTree Fixup at x as follows.

Step-4 If color of x is Red then make it Black and stop the procedure.
If x is Root then also stop the procedure.

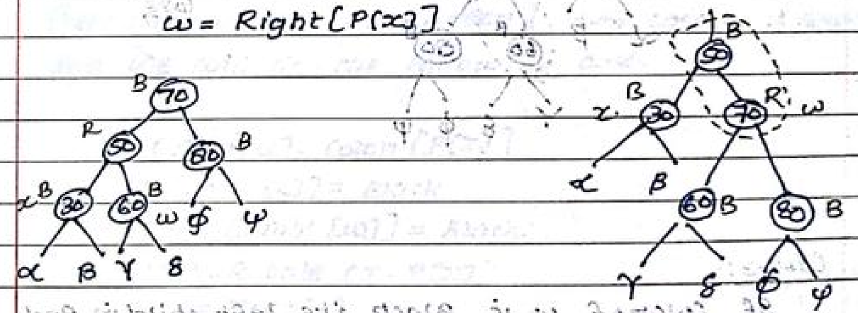
Step 5 If color of x is also Black then find out the sibbling of x and denote it by ω. On the basis of the color of ω and its children one of the following 4-Normal cases will be applied [If ω is the Right child of Parent only then Normal case is applied].

24 Aug 2018

Case:1:-

If color of ω is Red then Case1 will applied and we will do following in this case

Color [ω] = Black
Color [P[x]] = Red
left_Rotate (T, P[x])
ω = Right [P(x)]



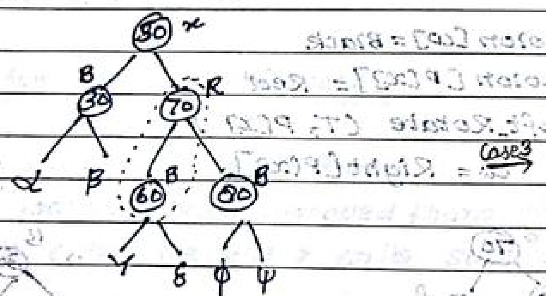
This case will not solve the problem but it rotate the tree in such a way so that the new sibbling of x, ω will be black now. So we can apply the Cases- 2,3 & 4 on the new tree.

Case-2:- If color of ω is Black and both of its children are also Black then case-2 will applied and we will do the following in this case.

Color $[\omega]$ = Red

$x = P[x]$

go to step-4



Case-3:-

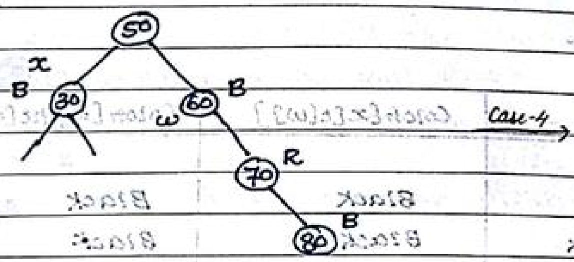
If color of ω is Black its left child in Red and Right child is Black then case-3 is applied and we will do the following in this case.

Color $[\omega]$ = Red

Color [left $[\omega]$] = Black

Right-Rotate (T, ω)

ω = Right $[P[x]]$



Case-3 will not solve the problem but it rotate the tree in such a way so that we can apply Case-4 on the tree now.

Case-4:-

If color of ω is Black and its Right child is Red [left child may be Black or Red] then case-4 is applied and we will do the following cases.

Color $[\omega]$ = Color $[P[x]]$

Color $[P[x]]$ = Black

Color [Right $[\omega]$] = Black.

left-Rotate $(T, P[x])$

x = Root$[T]$

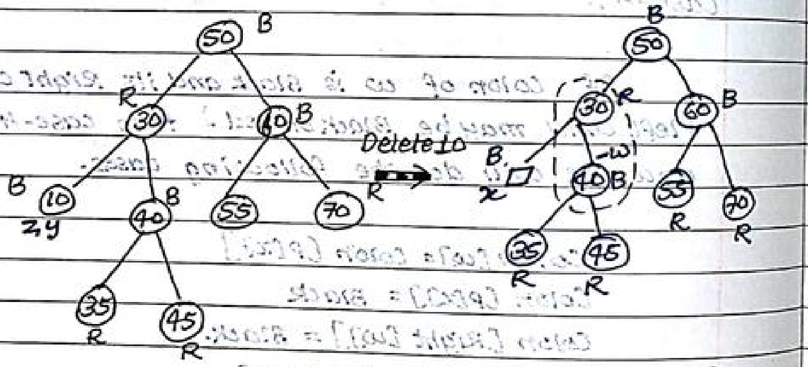
Case-4 will solve the problem.

Step-6 If ω is the left child of its Parents then for Reverse cases is applied and this cases are similar to the normal case and just exchange left to Right.

Table:-

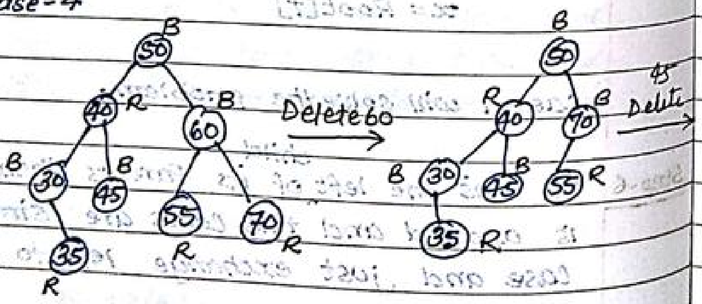
Color	Color[Left[w]]	Color[Right[w]]	Cases
Red	Black	Black	Case-1
Black	Black	Black	Case-2
Black	Red	Black	Case-3
Black	Red / Black	Red	Case-4

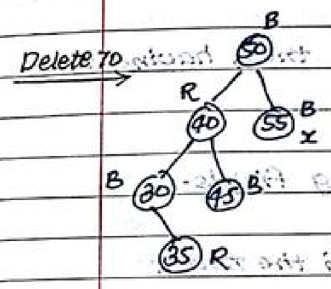
Numericals:-

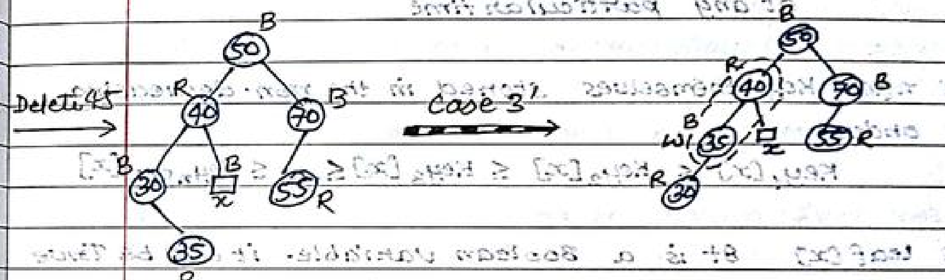


w = Sibbling of x

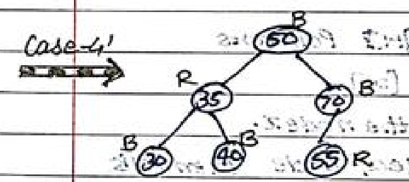
By Appling Case-4







In Reverse case if left is Red then apply Case-4' then.



27/08/2018

B-Tree:-

A B-Tree is a rooted tree, having the following properties.

1. Every node x have the following fields.

(a) $x[x]$ contain the no. of keys in the node x at any particular time

(b) $n[x]$ key themselves stored in the non-decreasing order.

$$Key_1[x] \le Key_2[x] \le Key_3[x] \le \ldots \le Key_{n[x]}[x]$$

(c) leaf $[x]$ It is a Boolean variable. it will be True if the node $[x]$ is leaf node otherwise it will be False.

2. Every node x also have $n[x]+1$ Pointers as $C_1[x], C_2[x] \ldots\ldots C_{n[x]+1}[x]$. that point to the children of the node x. If node $[x]$ is leaf node then this pointers are undefined.

3 $Key_i[x]$ seperate the keys present in the subtree rooted at node $[x]$. If K_i is any key in the subtree rooted at $C_i[x]$.

$$K_1 \le Key_1[x] \le K_2 \le Key_2[x] \le K_3 \le \ldots\ldots \; Key_{n[x]}[x] \le K_{n[x]+1}$$

4. All the leaf nodes should be at the same depth and that will be the height of the B-Tree.

5. There is an Upper and Lower Bound on the no. of keys in any node of the B-Tree. These Bounds are defined by an integer value $t \ge 2$, called the minimum degree of the B-Tree.

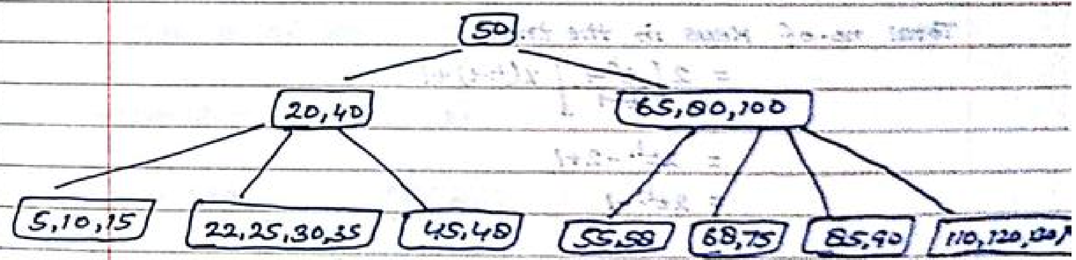
(a) Any node $[x]$ except the root should have atleast $t-1$ keys i.e., atleast t children if it is not a leaf. Root can have minimum 1 key.

(b) Every node $[x]$ can have maximum $2t-1$ keys.

Eg. $t=3$

min. keys $= t-1 = 2$

min keys $= 2t-1 = 5$



27/08/2018

Lemma :- (RB-TREE)

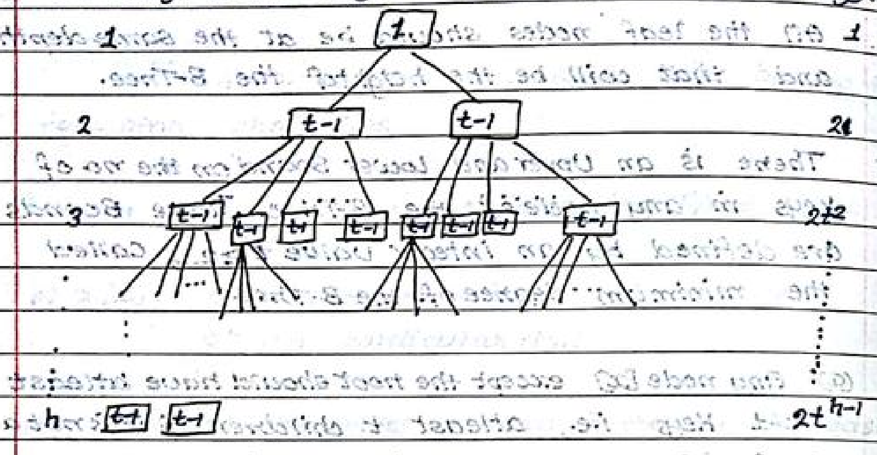
For any 'n' key B-Tree of height 'h' and minimum degree $t \ge 2$.

$$h \le \log_t \left[\frac{n+1}{2}\right]$$

m.imp

Ques:-1 Prove that the height of B-Tree is $h \le \log_t \left[\frac{n+1}{2}\right]$ for any 'n' keys.

we design a tree height 'h' of minimum no. of Keys.



Total of nodes in the tree with t-1 Keys

$= (2+2t+2t^2+\dots\dots\ 2t^{h-1})$

$= 2(1+t+t^2+\dots\dots t^{h-1})$

$= 2\left[\dfrac{t^h-1}{t-1}\right]$

Total no. of Keys in the tree:

$= 2\left[\dfrac{t^h-1}{t-1}\right] \times (t-1)+1$

$= 2t^h-2+1$

$= 2t^h-1$

So No. of Keys in the tree, $n \geq 2t^h-1$

$2t^h \leq n+1$

$t^h \leq \dfrac{n+1}{2}$

$h \leq \log_t \left[\dfrac{n+1}{2}\right]$

Hence Proof

Insertion in B-Tree :-

Step-1:- In B-Tree, the Keys are always inserted at the leaf nodes. To insert a new key we will start from root & traverse downward to the and insert the key at the last node that we have traversed.

Step-2:- In the path from the root to the leaf if we get any full node [Contain 2t-1 Keys] then first we will split the node and then go downwords.

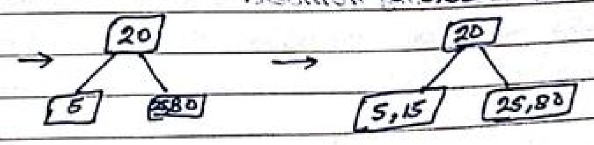
imp
Ques:- Insert the following keys in a B-Tree of min. degree t=2 initially the B-Tree is empty

20, 80, 5, 25, 15, 65, 30, 40, 50,

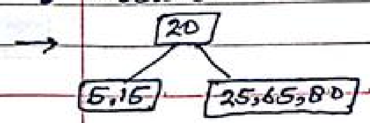
min. no. of Keys = t-1 = 1 [∵ t=2] Given

max. no. of Keys = 2t-1 = 3

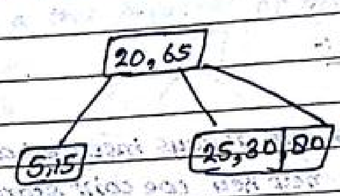
- Insert 20 - $\boxed{20}$
- Insert 80 - $\boxed{20,80}$
- Insert 5 - $\boxed{5,20,80}$
- Now Split then, Insert 25 will be as • Insert 15



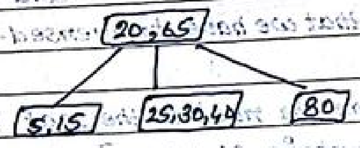
- Insert 65 -



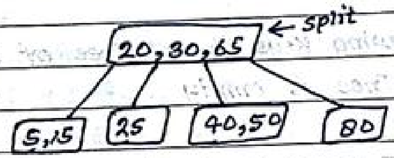
Now Split and Insert 80



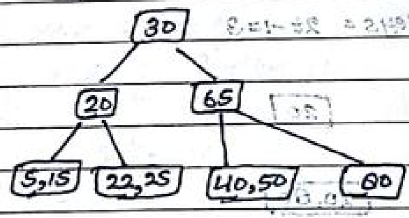
- Insert 40 -



- Split and Insert 50



- Split and Insert 22



Note:- In this first we split them and after that we can insert any number.

28/08/2018

Deletion in B-Trees:-

The Following cases can be applied to delete the tree in B-Trees.

Case-1:-

If key K is in the leaf node and the leaf node contain more than minimum no. of keys then delete the key 'K' directly

Case-2:-

If the key 'K' is in the Internal node then do the following

(a) If the child y that preceeds K has more than minimum no. of keys [t-1] then, find the predecessor K' of K, recursively delete K' and replace K by K'

(b) If the child y that succeeds K has more than min. no. of keys then find the successor K' of K, recursively delete K' and replace K by K'.

(c) If preceding and succeding nodes of K, both contains only (t-1) keys then combine both the nodes and delete K recursively from the new nodes.

leaf, internal
1 2 ?

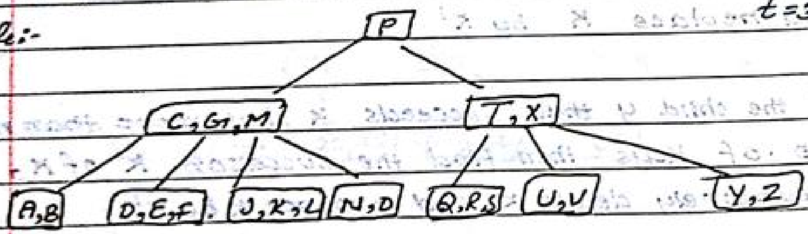
Case-3:-

If in the path from the root to the node x that contain K, any node contain minimum no. of keys then first we increase the key in that node by using the case-3(a) or 3(b) then go downward.

(a) If any immediate sibbling of x contain more than minimum no. of key then shift one key from the sibbling to the node x by using rotation through the parents.
If sibbling contain children then shift 1 child also.

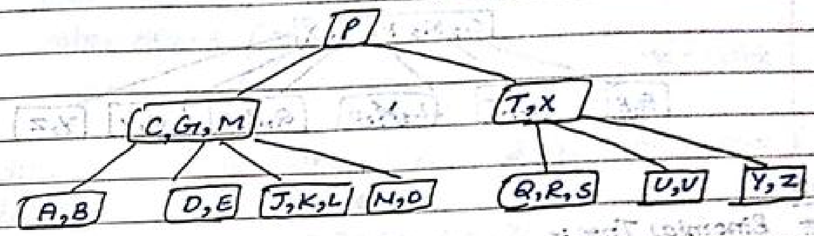
(b) If both immediate sibblings of x contain the minimum no. of keys then combine x with anyone of the immediate sibblings.

Example:- $t=3$

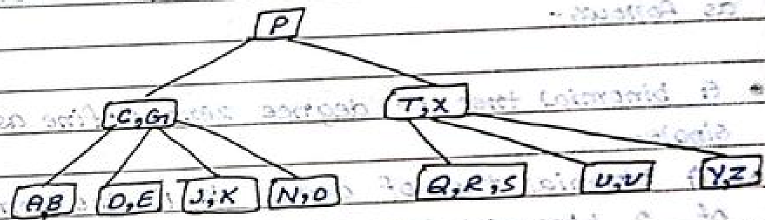


min key $= t-1$
$= 3-1 = 2$

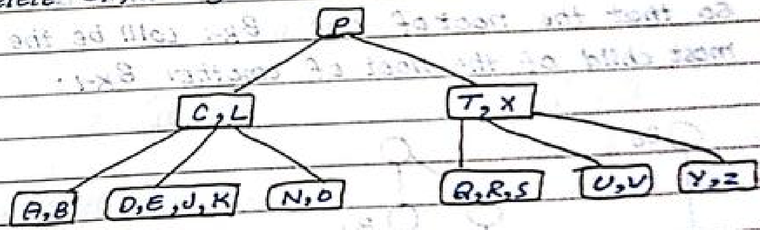
- Delete F
By Using case 1



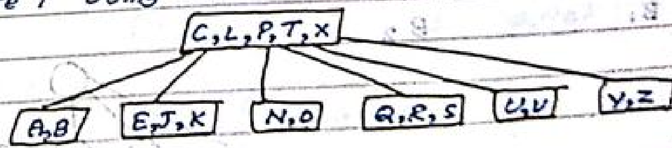
- Delete M From case 2(a)



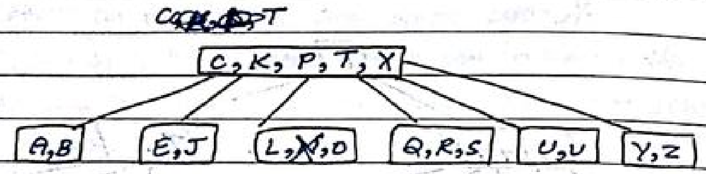
- Delete G, using case-2c



- Delete T using case- 3(b) and



- Delete N

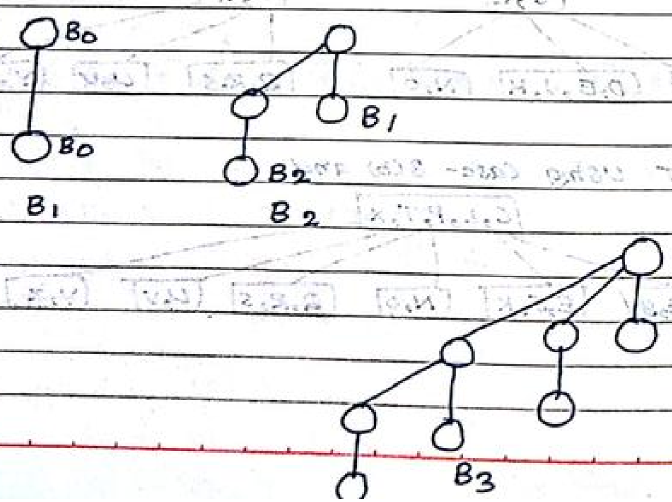


30/08/2018

Binomial Tree :-

Binomial tree is a rooted tree define recursively as follows -

- A binomial tree of degree zero define as B_0 is a single.
- A binomial tree of degree K is the combination of 2 binomial tree of degree K-1
- This 2 B_{K-1} will be connected in such as way so that the root of one B_{K-1} will be the left most child of the root of another B_{K-1}.



Insertion in B-Tree [If Order m is Given] :-

max. keys = m-1

min. keys = $\lceil \frac{m}{2} \rceil - 1$

Ques:- Insert the following keys in B-Tree of order 5. let initially tree is empty.

50, 10, 25, 70, 30, 90, 40, 75

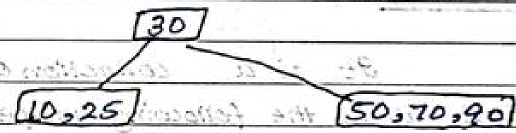
max. keys = m-1

= 5-1 = 4

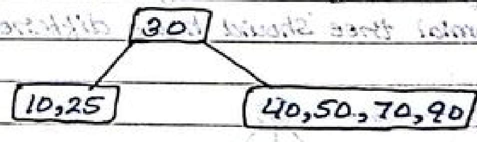
$\boxed{10, 25, 50, 70}$

$\boxed{10, 25, 30, 50, 70}$

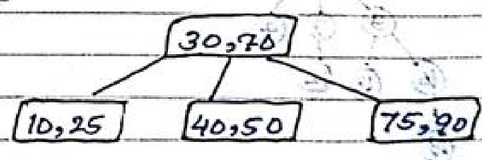
Insert 30 :-

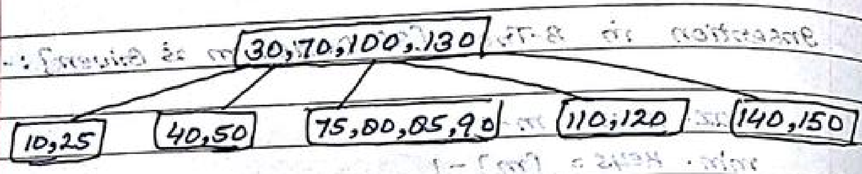


Insert 90, 40 :-

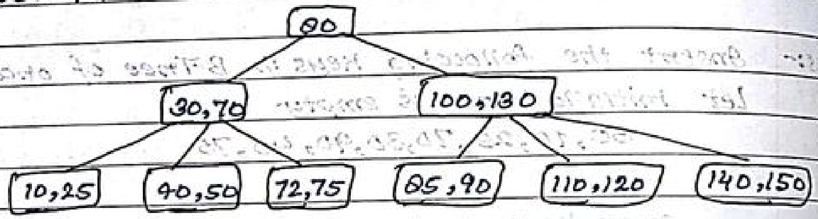


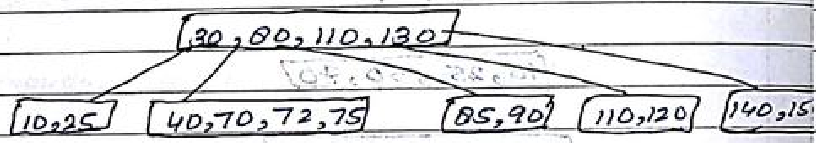
Insert 75 -





Insert 72 -

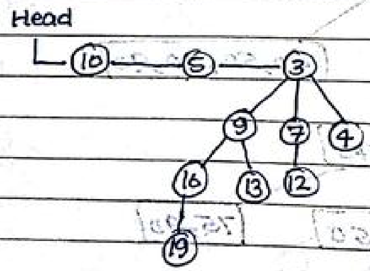




Binomial Heap :-

It is a collection of Binomial trees that fullfill the following properties.

- All the Binomial trees shouldbe min. head order.
- For any non-negative integer K, there should be atmost 1 Binomial tree should have different order.



Structure of Binomial Heap:-

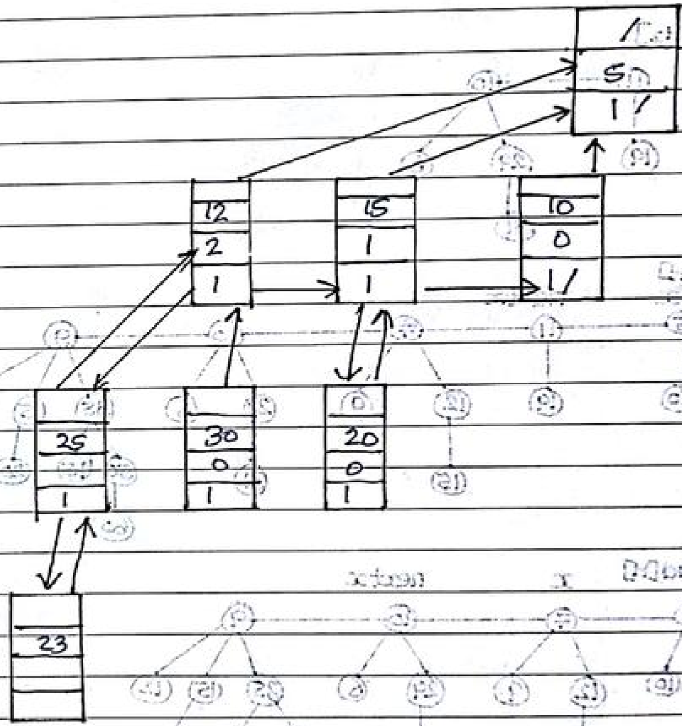
Key → Contain value

P → Address of Parent

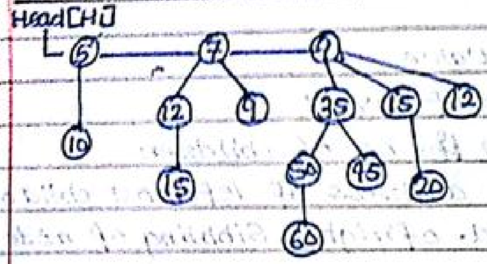
Degree → Contain the no. of children

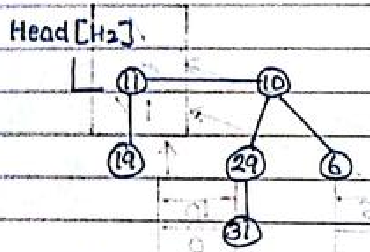
child → Contain address of leftmost child of node

Sibbling → Contain add. of right Sibbling of node.

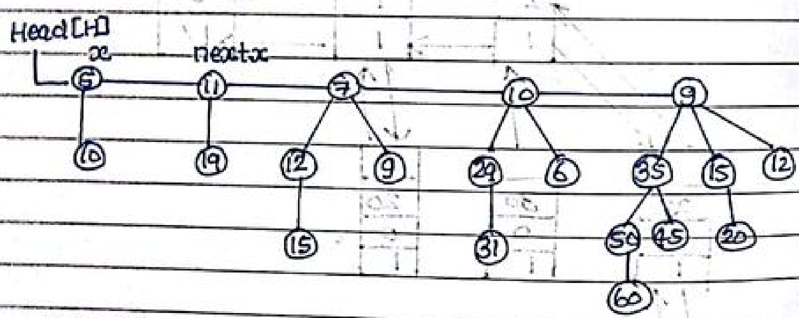


Union of Two Binomial Heap :-

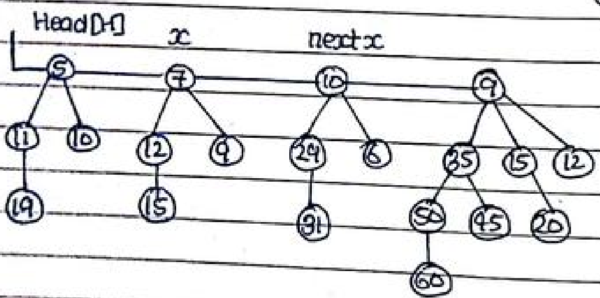




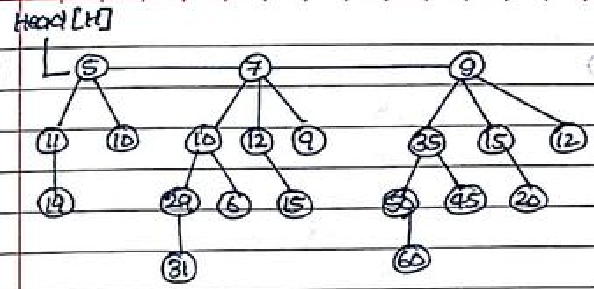
(a)

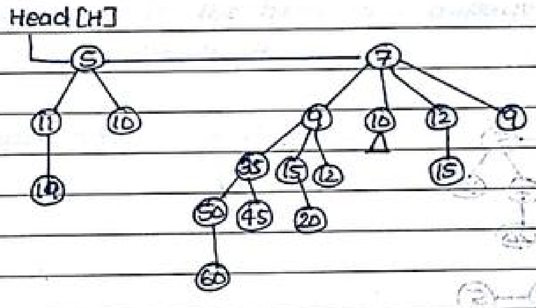


(b)



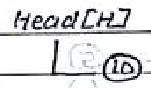
(c)

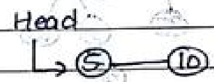


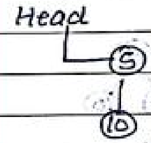


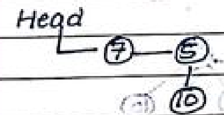
Ques. Insert the following keys in Binomial Heap.

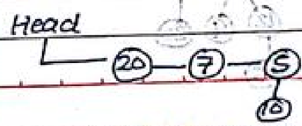
10, 5, 7, 20, 25, 12, 9, 8

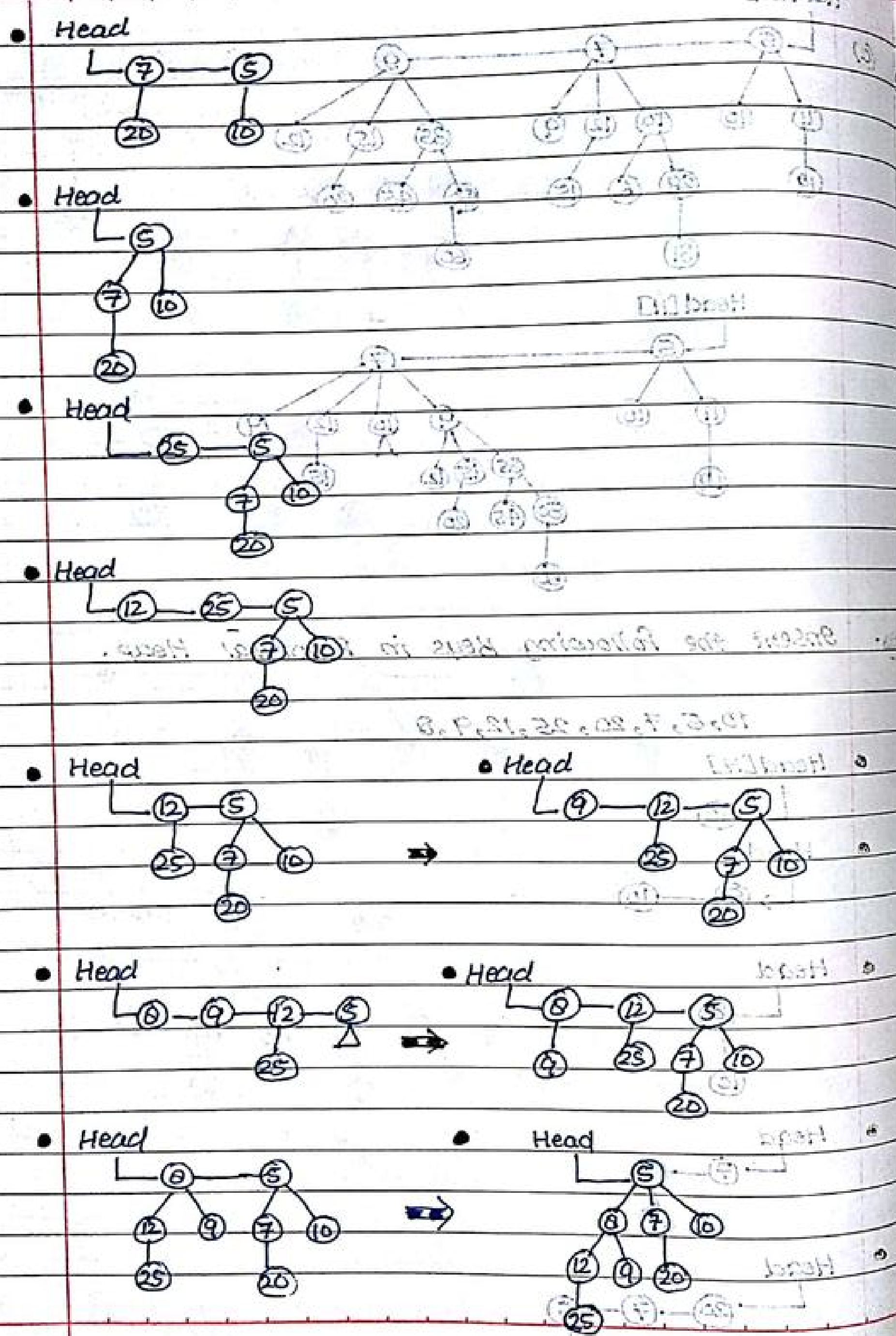
- 

- 

- 

- 

- 



ves- $23 \Rightarrow \overset{43210}{(10111)_2}$

$\Rightarrow B_0, B_1, B_2, B_4$

$7 \Rightarrow (111)$

$\Rightarrow B_0, B_1, B_2$

Note:- *Binomial tree in the heap are always arranged from Low order to High Order.*

Structure of Binomial Heap [Written in Previous Page]

31/08/2019

Algorithm

```
Binomial-Heap-Union (H1,H2)
  H = Make-Binomial-Heap ()
  Head[H] = Binomial-Heap-Merge (H1,H2)
  If Head[H]= NULL
     return
  prev_x = NULL
    x = Head[H]
  next_x = Sibbling [x]
  while next_x ≠ NUL
  {
    if((degree [x] ≠ degree [next_x]) OR (Sibbling [next
                                            ≠ NULL ])
          and degree [x] = degree(Sibbling [next_x]])
          prev_x = x
             x = next_x
          next_x = Sibbling [x]
    else
     {
       if (Key[x] ≤ Key [next_x])
       {
         Sibbling [x] = Sibbling [next_x]
       Binomial_Link (next_x, x)
       }
     else
        {if prev-x = NULL
              Head (H]= next-x
          else
              Sibbling [prev-x] = next_x
```



```
        Binomial-Link (x, next-x)
             x = next_x
             next-x = Sibbling [x]
        }
   }
 return H.
```

Binomial Link Algo:-

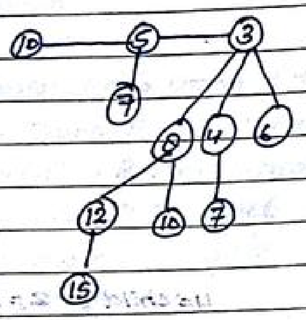
y = child , z = Parent

```
Binomial_Link (y,z)
  P.[y] = z
  Sibbling [y] = child [z]
  child [z] = y
  degree [z] = degree [z] + 1
```

Insertion in Binomial Heap:-

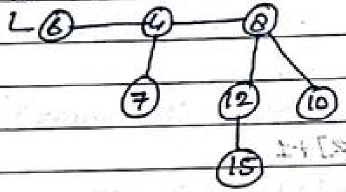
```
Binomial-Heap-Insert (H,x)
H' = Malloc-Binomial-Heap()
 Head [H'] = x
    P[x] = NULL
   child [x] = NULL
   Sibbling [x] = NULL
   degree [x] = 0
  H = Binomial-Heap-Linear (H,H')
```

Extracting Minimum Key From Binomial Heap :-

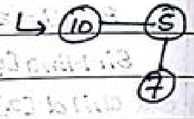


Reverse-

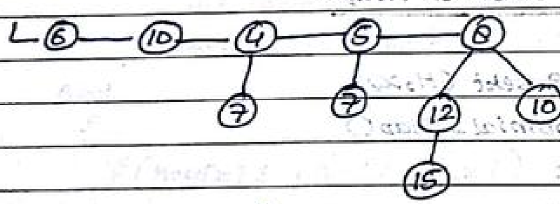
Head[H']

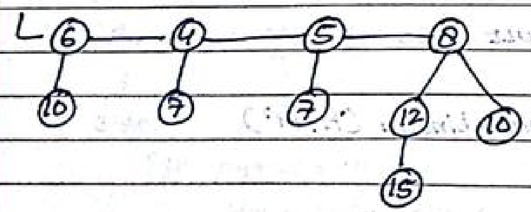


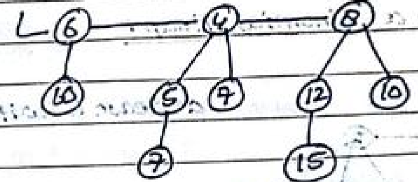
Head[H]

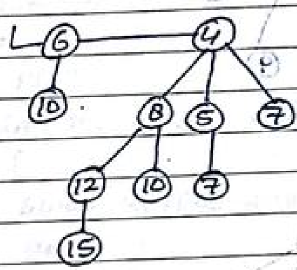


Union :-









04 Sept 2018

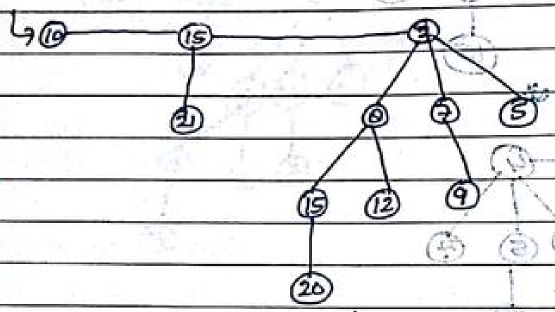
Algorithm :-

Binomial-Heap-Extract-Min (H).

1. Find the root x with the min. key in the root list of H. and remove x from the list of H.
2. H' = Make-Binomial-Heap ()
3. Reverse the order of the linked list of the children of x. and set Head[H'] to point to the head of the resulting list.
4. H = Binomial-Heap-Union (H, H')
5. Return x.

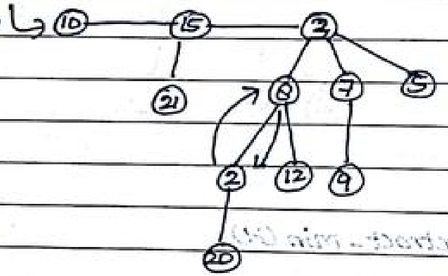
Decreasing a Key in a Binomial Heap :-

decrease 15 with 2



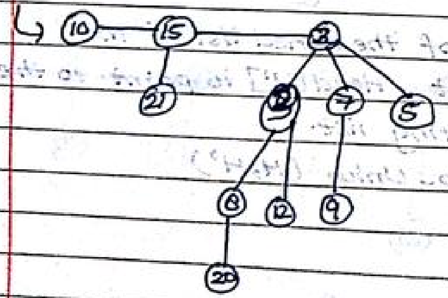
⇓

swap



swap 2 and 8 whenever the child in ~~greater~~ smaller than parent

⇓



Algorithm [z= Parent and y=child]

Binomial-Heap-Decrease-Key (H, x, k)

1. if K > Key [x]
   then error and return
2. Key [x] = K
3. y = x
4. z = P[x]
5. while (z ≠ NULL and Key[y] < Key [z])
   {
   Swap key[y] and Key[z]
   y = z
   z = P[y]
   }

m.imp

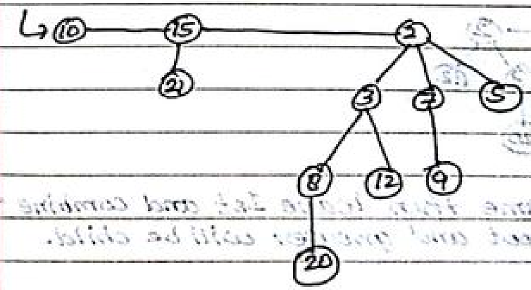
Deleting a Key from in a Binomial Heap :-

Algorithm :-

Binomial-Heap-Delete-Key (H, x)

1. Binomial-Heap-Decrease-Key (H, x, -∞)
2. Binomial-Heap-Extract-min (H)

Ques-



1. Delete & Decrease Key, and made them (−∞) and swap.

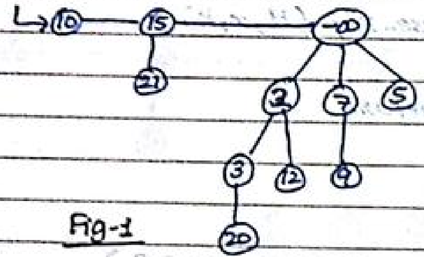


Fig-1

2. Extract min(H)

Head[H]

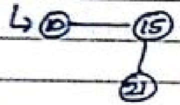


Fig-2

3.

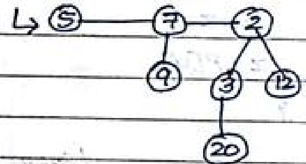


Fig-3

3. Union

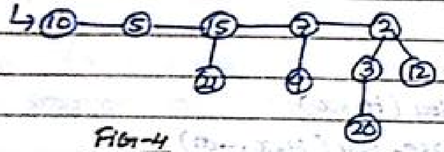


Fig-4

4. degree and same then combine them.

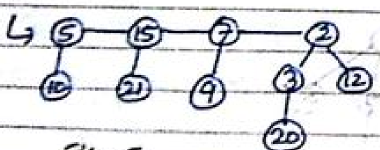


Fig-5

Again degree are same then leave 1st and combine 2&3 compare, if smaller will be root and greater will be child.

Fig-6

Again degree are same then union them

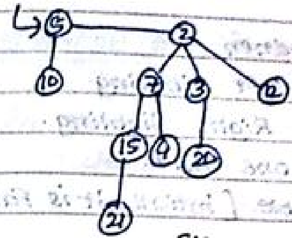
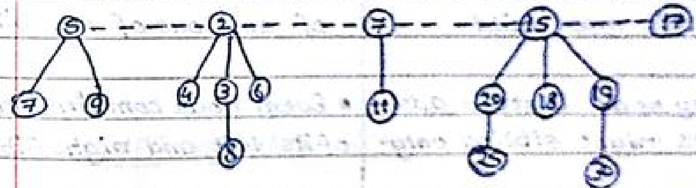


Fig-7

Note:- when degree are same then Union it. and then compare the same degree if one is small & other is greater then. Smaller one will be root and greater one will be its child.

04 Sept. 2018

Fibonacci Heap :-



- Fibonacci Heap is a Collection of rooted trees that are min-Heap order, i.e. each trees obeys the min heap property.

- Every node of the Fibonacci Heap Contain the following key.

1. Key
2. P → Address of Parent.

3. degree → No. of children
4. Left → Address of Left Sibbling
5. Right → Address of Right Sibbling.
6. Child → Add. of any one child.
7. Mark → Boolean Variable [intially it is False]

Difference b/w Binary Heap and Fabonacci Heap:-

Binary Heap (B.H)	Fabonacci Heap (F.H)
• In Binary Heap trees should be Binomial.	• In Fabonacci trees may be any rooted trees.
• The child is singly linked list.	• The child is Circular doubly linked list.
• Parent contain the address of left most child.	• Parent contain address of any one of the child.
• Every node contain add. of its right sibling only.	• Every node contain add. of its left and right Sibbling.
• Root list is singly linked list and Head[H] is a pointer that point to the left most root of the list.	• The root list is circular doubly linked list and min[H] is a pointer that point to the root node with min. key.
• It contain 5 Fields.	• It contain 7 Fields.

Insertion in Fabonacci Heap:-

Fib-Heap-Insert (H, x)

degree [x] = 0

P[x] = NULL

child[x] = NULL

Left[x] = x

Right[x] = x

Mark[x] = False

If (min[H] = NULL)

• Created root list of H containing just x, then min[H] = x.

else

{insert x in root list of H}

if Key[x] < Key[min[x]]

min[H] = x.

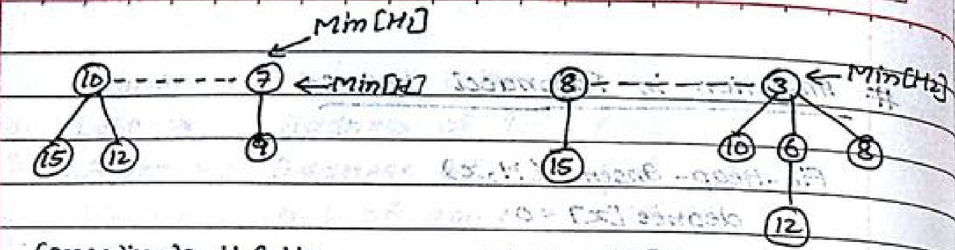
05 Sept. 2018

Union of Two Fabonacci Heaps:-

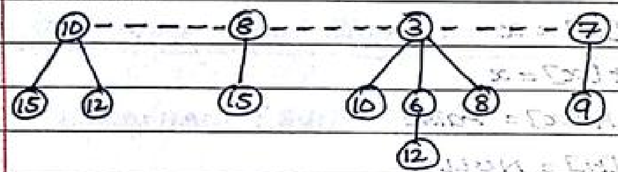
Fib-Heap-Union (H1, H2)

1. H = Make-Fibonacci-Heap()
2. min[H] = min [H1]
3. concatinate the root list of H2 with the root list of H.
4. If (min[H1] = NIL) OR (min[H2] ≠ NIL and Key [min[H2]] < Key[min [H1]] min[H] = min[H2]
5. n[H] = n [H1] + n[H2]
6. return H.

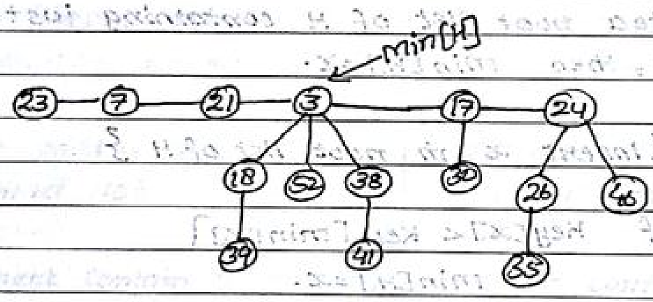
Example:



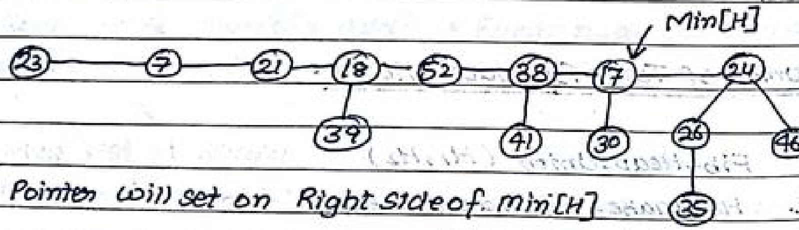
Concatinate H & H_2



Imp # Example

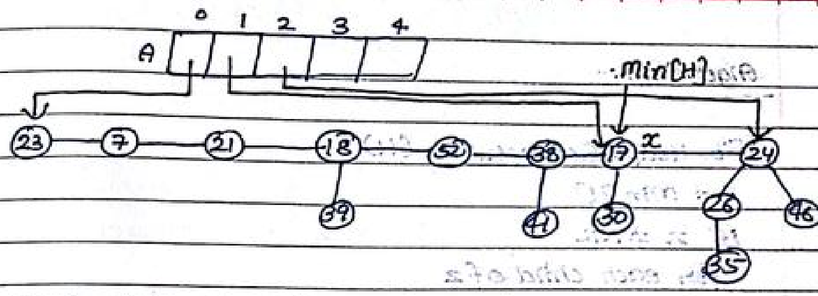


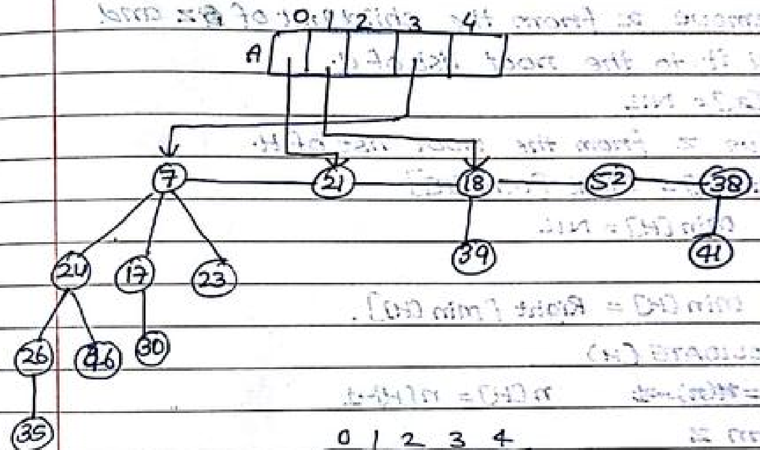
• Extract 3

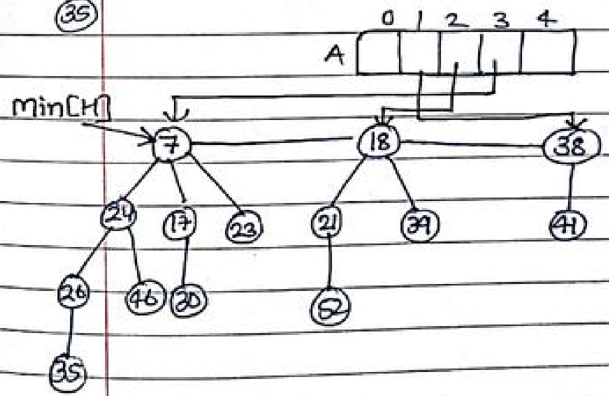


Pointer will set on Right side of min[H]

•







when degree are different then stop and then set min[H] which is smallest one.

Algorithm :-

Fib-Heap-Extract-min (H)

z = min [H]

if z ≠ NIL

for each child of z

Remove x from the child list of z and add it to the root list of H.

P[x] = NIL

Remove z from the root list of H.

If min [H] = Right [min [H]]

min [H] = NIL

else

min [H] = Right [min [H]].

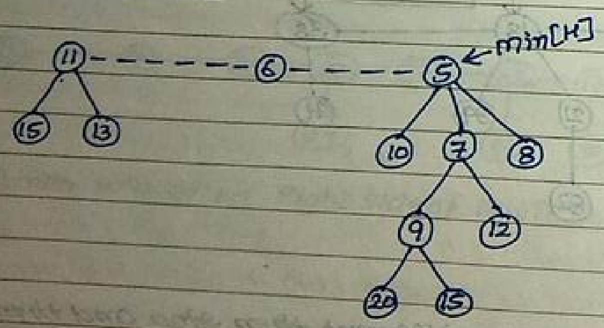
CONSOLIDATE (H)

~~H(n) = H(n) - 1~~ n [H] = n [H] - 1

return z

6 Sept. 2018

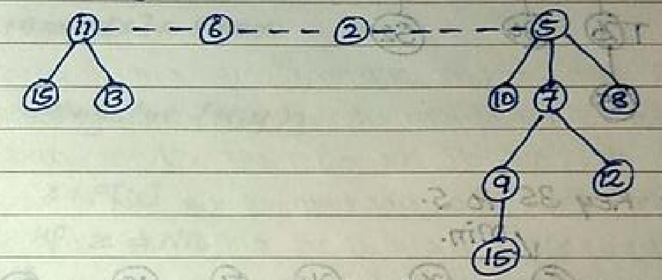
Decreasing key :-



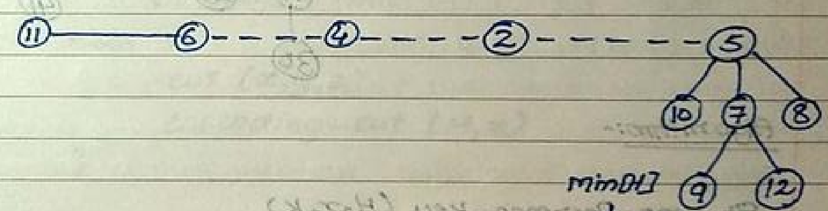
Ques- Apply decrease key two times on the given heap.

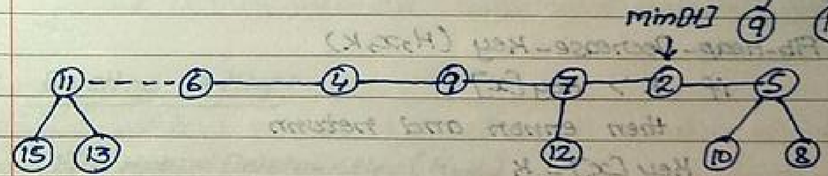
1. decrease 20 to 2 then
2. decrease 15 to 4

1. decrease key 20 to 2

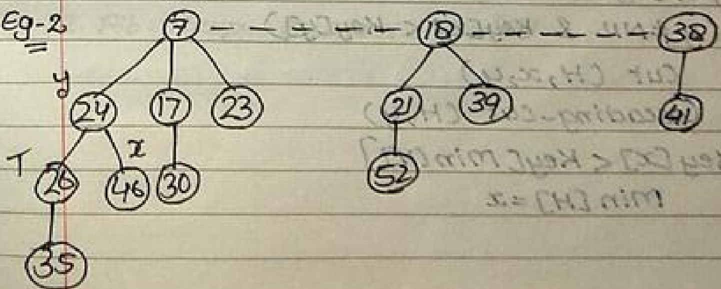


2. decrease key 15 to 4.

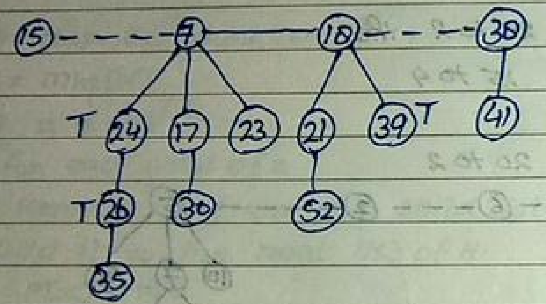




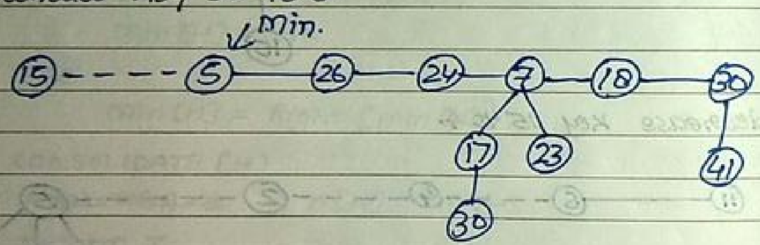
Eg-2



Decrease key 46 to 15.



Decrease key 35 to 5.



Algorithm:-

```
Fib-Heap-Decrease-key (H,x,k)
    if  k > key[x]
        then error and return
    key[x] = k
    y = P[x]
    If  (y ≠ NIL & key[x] < key[y])
        Cut (H,x,y)
        Cascading-cut (H,y)
    If  key[x] < key[min(H)]
        min[H] = x
```


Cut (H,x,y)

1. Remove x from child list of y.
2. Add x to root list of H.
3. degree [y] = degree [y] - 1
4. P[x] = Nil
5. mark[x] = False

Cascading-Cut (H,y)

```
1.  z = P[y]
2.  If z ≠ NIL
    {
      If (mark[y] = False)
         mark[y] = True
      else
         cut (x,y,z)
         cascading-cut (H,z)
    }
```

Deleting a key:-

Fib-Heap-Delete-key (H,x)

1. Fib-Heap-Decrease-key (H,x,-∞)
2. Fib-Heap-Extract-min(H)

pt·2018

Unit-3

Dynamic Programming

Dynamic Programming :-

It is a bottom-up approach that is used when subproblems overlapped i.e. when subproblem share sub-sub-problem.

In this case if we will use divide and conquer then we need to solve the shared sub-subproblem more than one times and waste our time in repeated work.

In dynamic programming we start to solve the problem from the bottom & save their answers in a table and avoid the recomputation of the answers everytime.

When developing a dynamic programming algorithm we will follow the sequence of 4 steps.

1. Characterise the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution; typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Eg:-1 # Matrix Chain Multiplication :-

$[A_1]_{2\times10}$ $[A_2]_{10\times2}$ $[A_3]_{2\times20}$

- $(A_1\times A_2)\times A_3$
- $A_1\times(A_2\times A_3)$

$2\times10\times2+2\times2\times20$
$=40+80=120$

Note:- dimension will remain only 1st and last then

$2\times10\times2=$ $[\ ]_{2\times2}$

$[\ ]_{2\times10}=[\ ]_{10\times20}$

1. $(A_1\times A_2)\times A_3$
$= 2\times10\times2+2\times2\times20$
$= 40+80$
$=120$

2. $A_1\times(A_2\times A_3)$
$= 10\times2\times20+2\times10\times20$
$= 400+400$
$= 800$

In Matrix chain multiplication problem, we are given a chain of n matrices $<A_1, A_2 \ldots A_n>$. Here we want to find out optimal parenthesisation of the chain of matrices so that the no. of multiplication required is minimum.

The matrix A_i as the dimension $P_{i-1}\times P_i$

$A_1 = P_0\times P = 5\times4$
$A_2 = P_1\times P_2 = 4\times3$
$A_3 = P_2\times P_3 = 3\times2$
$A_4 = P_3\times P_4 = 2\times10$
$A_5 = P_4\times P_5 = 10\times6$

	P_0	P_1	P_2	P_3	P_4	P_5
P	5	4	3	2	10	6

$(P_{i-1} \times P_k)\ (P_k \times P_j)$

$(A_i, A_{i+1}, \ldots\ldots A_j)$ ↓ split into two Parts
$(A_i, A_{i+1}, \ldots\ A_k)$ $(A_{k+1} \ldots\ A_j)$

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k \le j-1} \left(m[i,k] + m[k+1,j] + P_{i-1} P_k P_j\right) \end{cases}$$

Let $m[i,j]$ Represent the Optimal no. of multiplication required to multiply the chain of matrices $[A_i, A_{i+1} \ldots A_j]$

To Find out the optimal no. of multiplication we will break this chain at k $[i \le k \le j-1]$.

The recursive formula to find out $m[i,j]$ will be as shown below

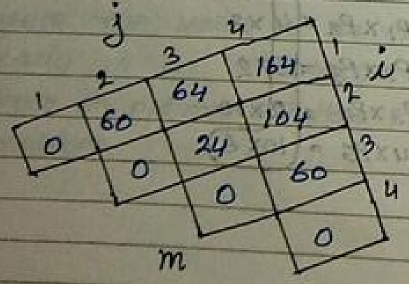
$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k \le j-1} \left(m[i,k] + m[k+1,j] + P_{i-1} P_k P_j\right) & \text{if } i<j \end{cases}$$

m.m.f

Example:

	0	1	2	3	4
P	5	4	3	2	10

[Half matrices with rotated at 45°



Note • Last Row is Always zero in every ques.
• For 2 multiply first 3 value, for 3 multiply next third from 2nd

$m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$

$= 0 + 0 + 5 \times 4 \times 3$

$= 60$

Now
• It will break on 1 and 2

$$m[1,3] = \min_{k=1,2} \begin{bmatrix} m[1,1] + m[2,3] + P_0 P_1 P_3 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{bmatrix}$$

$$= \min \begin{bmatrix} 0 + 24 + 5\times4\times2 \\ 60 + 0 + 5\times3\times2 \end{bmatrix} \quad \begin{matrix} 64 \\ 90 \end{matrix}$$

Sortcut method: -

$$m[1,3] = \min \begin{cases} 0 + 24 + 40 \\ 60 + 0 + 30 \end{cases}$$

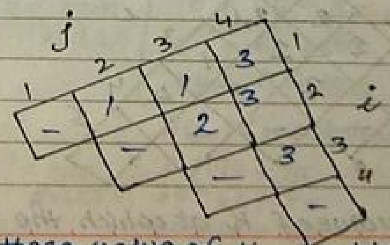
$= 64$

$$m[2,4] = \min_{k=2,3} \begin{bmatrix} 0 + 60 + 4\times10\times3 \\ 24 + 0 + 4\times10+3 \end{bmatrix}$$

$= 104$ (is minimum)

$$m[1,4] = \min_{k=1,2,3} \begin{bmatrix} 0 + 104 + 200 \\ 60 + 60 + 150 \\ 64 + 0 + 100 \end{bmatrix}$$

$= 164$ (bcoz it is minimum value)



Note: we put those value of k at which we can split and get minimum values.

$(A_1 A_2 A_3 A_4)$

$((A_1 A_2 A_3) A_4)$ at $k=3$ will break and get min value

$((A_1 (A_2 A_3)) A_4)$

14 Sept-2018

m.imp

Ques- Find out the optimal parenthesisation of the chain of matrices

$A_1 (3\times5)$, $A_2 (5\times2)$, $A_3 (2\times6)$, $A_4 (6\times2)$, $A_5 (2\times5)$

	P_0	P_1	P_2	P_3	P_4	P_5
P	3	5	2	6	2	5

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i\le k\le j-1} (m[i,k] + m[k+1,j] + P_{i-1} P_k P_j) \end{cases}$$

m

i \ j	1	2	3	4	5
1	0	30	66	66	96
2		0	6.0	44	94
3			0	24	44
4				0	60
5					0

i \ j	1	2	3	4	5
1	—	1	2	2	4
2		—	2	2	2,4
3			—	3	4
4				—	4
5					—

Put the value of K at which the value is minimum.

$$m[1,3] = \min_{1\le k\le 2} \begin{cases} m[1,1] + m[2,3] + P_0 P_1 P_3 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{cases}$$

$$= \min \begin{cases} 0+60+90 \\ 30+0+36 \end{cases}$$

$= 66$ (minimum)

OR

$$\Rightarrow \min_{k=1,2} \begin{cases} 0+60+90 \\ 30+0+36 \end{cases}$$

$= 66$ (is minimum)

$$m[2,4] = \min_{k=2,3} \begin{cases} 0+24+20 & P_1 P_2 P_4 \\ 60+0+60 & P_1 P_3 P_4 \end{cases}$$

$= 44$ (is minimum)

$$m[3,5] = \min_{k=3,4} \begin{cases} 0+60+60 & P_2 P_3 P_5 \\ 24+0+20 & P_2 P_4 P_5 \end{cases}$$

$= 44$ (is minimum

$$m[1,4] = \min_{k=1,2,3} \begin{cases} 0+44+30 & P_0 P_1 P_4 \\ 30+24+12 & P_0 P_2 P_4 \\ 66+0+36 & P_0 P_3 P_4 \end{cases}$$

$= 66$ (is minimum)

$$m[2,5] = \min_{k=2,3,4} \begin{cases} 0+44+50 & P_1 P_2 P_5 \\ 60+60+150 & P_1 P_3 P_5 \\ 44+0+150 & P_1 P_4 P_5 \end{cases}$$

$= 94$ (is minimum)

$$m[1,5] = \min_{k=1,2,3,4} \begin{cases} 0+94+75 & P_0 P_1 P_5 \\ 30+44+30 & P_0 P_2 P_5 \\ 66+60+90 & P_0 P_3 P_5 \\ 66+0+30 & P_0 P_4 P_5 \end{cases}$$

$= 96$ (is min)

(1,5) $(A_1 A_2 A_3 A_4 A_5)$

(1,4) → $((A_1 A_2 A_3 A_4) A_5)$

→ $(((A_1 A_2) (A_3 A_4)) A_5)$

→

- we will break at value is more than 2 if it equal to 2 or less than 2 then do not break

14 Sept. 2018

Eg-2

Longest Common Subsequence [LCS] :-

let we have two sequences

$X = <x_1, x_2 \dots x_n>$

$Y = <y_1 . y_2 \dots y_n>$

Let $Z = < z_1 z_2 \dots z_K >$ is a longest common subsequence of X and Y.

1. If $x_m = y_n$ then $Z_K = x_m = y_n$ and Z_{K-1} is an LCS of X_{m-1} and Y_{n-1}

2. If $x_m \neq y_n$ then $Z_K \neq x_m$ implies that Z is an LCS of x_{m-1} and Y

3. If $x_m \neq y_n$ then $Z_K \neq y_n$ Implies that Z is an LCS of X and Y_{n-1}

Let C[i,j] is the length of the LCS of X_i and Y_j then the recursive formula of C[i,j] as follows:-

$$C[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ C[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(C[i-1,j], C[i,j-1]) & \text{if } x,j>0 \text{ \& } x_i \neq y_j \end{cases}$$

Prev. Row Prev. Col.

↑ = Same

↖ = diagonal

← = left value

diagonal+1 (if match)

(upper value & prev) (not match)

imp Ques. Find out an LCS of sequences

$X = abcaba$

$Y = bcabcab$

X+1 = Row

Y+1 = Column

x_i \ y_j	i \ j	0	b 1	c 2	a 3	b 4	c 5	a 6	b 7
	0	0	0	0	0	0	0	0	0
a	1	0	0↑	0↑	1↖	1←	1←	1↖	1←
b	2	0	1↖	1←	1↑	2↖	2←	2←	2↖
c	3	0	1↑	2↖	2←	2	3↖	3←	3←
a	4	0	1↑	2↑	3↖	3←	3↑	4↖	4←
b	5	0	1↖	2↑	3↑	4↖	4←	4↑	5↖
a	6	0	1↑	2↑	3↖	4↑	4↑	5↖	$\boxed{5↑}$

- Follow arrow and move in arrow direction. and circle the diagonal element.

LCS = abcab

another LCS = bcaba

Note:-
- Row will make as value of X+1 [No of X+1]
- Column will make as value of Y+1
- When we matching them, if match then make the value as diagonal+1 and write them.
- If not match then see the upper value and left value and compare them and write the greater one.

17 Sept. 2018

imp # Greedy Algorithms :-

imp 1. Fractional Knapsack Problem :-

Ques Find out the optimum/ maximum profit of following fractional knapsak problem if the capacity of the knapsack, $K=30$:

$K=30$

	I_1	I_2	I_3	I_4	I_5
w_i	10	5	7	15	8
Val_i	2000	7000	2100	1500	400
Cost/unit	200	1400	300	100	50
f_i	1	1	1	8/15	0

Remaining Capacity $K' = 30$ $[K' = K]$

Step-1 Most profitable item is I_2

$w_2 = 5 < K'$

$f_2 = 1$

$K' = 30 - 5 = 25$

Step-2 Next Profitable item is I_3

$w_3 = 7 < K'$

$f_3 = 1$

$K' = 25 - 7 = 18$

Step-3 Next profitable item is I_1

$w_1 \; 10 < K'$

$f_1 = 1$

$K' = 18 - 10 = 8$

Step-4 Next profitable item is I_4

$w_4 = 15 \nless K'$

$f_4 = \frac{K'}{w_4} = \frac{8}{15}$

$K' = 0$

Step-5 $f_5 = 0$

Optimum/ maximum Profit $= \sum f_i V_i$

$= (2000\times1) + (7000\times1) + (2100\times1) + (1500\times 8/15) + (400\times0)$

$= 2000 + 7000 + 2100 + 800 + 400$

$= 11900$

Example # Task scheduling :-

In this problem we are giving a set of task with its deadline and profit [penalty]. Each task can be performed in unique time. If task will be completed within its deadline then we will get the profit otherwise we will lost the profit.

We need to schedule this task in such a way so that the profit is maximum [penalty is min.].

Example :-

		T_1	T_2	T_3	T_4	T_5	T_6	T_7
Deadline,	D_i	3	1	3	4	2	4	6
Profit,	P_i	70	60	50	40	30	20	10

Step-1 Arrange the task in decreasing order of their Profit.

Step-2-

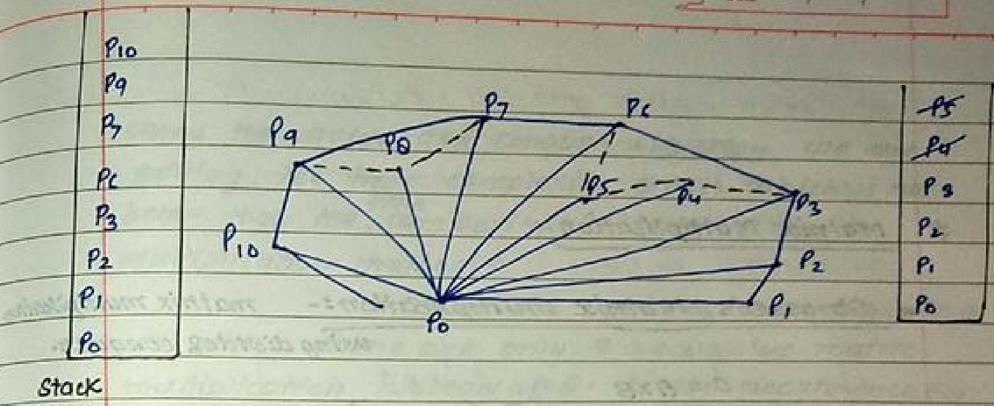
Sr.No	Task	optimal Schedule	Profit
1	T_1	$\langle T_1 \rangle$	70
2	T_1T_2	$\langle T_2T_1 \rangle$	130
3	$T_1T_2T_3$	$\langle T_2T_1T_3 \rangle$	180
4	$T_1T_2T_3T_4$	$\langle T_2T_1T_3T_4 \rangle$	220
5	$T_1T_2T_3T_4T_5$	No Optimal Schedule	
6	$T_1T_2T_3T_4T_6$	No Optimal Schedule	
7	$T_1T_2T_3T_4T_7$	$\langle T_2T_1T_3T_4T_7 \rangle$	230

So optimal Schedule is $\langle T_2T_1T_3T_4T_7T_5T_6 \rangle$ (Leaves last one)

imp # **Convex Hull :-** [Grahm's Scan] [Short Notes]

. We are given a set of points in 2 dimension space our problem is to find out a convex polygon such that all the given points are either in the interior of the polygon or at the boundary of the polygon.

Note: • Point are taken in increasing order.
• From point if we take left point then push them.
while if take right turn then ~~Pop~~ pop the top element of stack.



18 Sept-2018

Grahm's Scan Algorithm:-

Grahm_Scan(Q)

1. let P_0 be the point in Q with the minimum Y co-ordinate or the left most such point in case of a tie.

2. let $\langle P_1 P_2 \dots P_n \rangle$ be the remaining points in Q, sorted by polar angle in counter clockwise order around ~~by~~ P_0.

3. let S be an empty stack then push (P_0,S)
5. Push (P_1,S)
6. Push (P_2,S)
7. for (i=3 to n)
{
8. while the angle formed by the points Next_to_top(S), ~~top~~ (P_i), Top(S)
9. pop (S)
10. Push (P_i,S)
}
11. return S

Matrix Multiplication:-

Strassen's Matrix Multiplication:- matrix multiplication using divide & conquer.

$C = A \times B$

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$r = ae + bg$
$s = af + bh$
$t = ce + dg$
$u = cf + dh$

$T(n) = 8T(n/2) + \Theta(n^2)$

$a = 8$
$b = 2$

$f(n) = n^2$

$n^{\log_b a} = n^{\log_2 8} = n^3$

$f(n) = O(n^2)$
$= O(n^{\log_b a - \epsilon})$
$3 - \epsilon = 2$
$\epsilon = 1 > 0$
So $T(n) = \Theta(n^3)$

Using the divide and Conquer Strategy we are getting the time complexity $\Theta(n^3)$ which is not better than the time complexity of iterative matrix multiplication method.

Strassen's proposed an algorithm in which there are only 7 recursive matrix multiplication instead of 8. So the recurrence for Strassen's algorithm will be as follows.

$T(n) = 7T(n/2) + \Theta(n^2)$
$= \Theta(n^{\log_2 7})$
$= \Theta(n^{2.81})$

The time complexity of Strassen's algorithm is better than normal matrix multiplication.

19 Sept. 2018

Algorithm :-

1. Divide the input matrices A and B and o/p matrices C into $\frac{n}{2} \times \frac{n}{2}$ submatrices

2. Create 10 matrices $S_1, S_2, \ldots S_{10}$ each of which is $\frac{n}{2} \times \frac{n}{2}$ and is the some odd difference of two matrices created in Step-1. It will take $\Theta(n^2)$ time.

3. Using the submatrices created in step-1 and the 10 matrix created in step-2, recursively comput

the 7seven matrix product $P_1, P_2 \ldots\ldots P_7$

4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of P_i matrices. using $\Theta(n^2)$ time.

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

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$ — (1)

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$ — (2)

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$ — (3)

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$ — (4)

$S_1 = B_{12} - B_{22}$	$P_1 = A_{11}S_1$
$S_2 = A_{11} + A_{12}$	$P_2 = S_2B_{22}$
$S_3 = A_{21} + A_{22}$	$P_3 = S_3B_{11}$
$S_4 = B_{21} - B_{11}$	$P_4 = A_{22}S_4$
$S_5 = A_{11} + A_{22}$	$P_5 = S_5S_6$
$S_6 = B_{11} + B_{22}$	$P_6 = S_7S_8$
$S_7 = A_{12} - A_{22}$	$P_7 = S_9S_{10}$
$S_8 = B_{21} + B_{22}$	
$S_9 = A_{11} - A_{21}$	
$S_{10} = B_{11} + B_{12}$	

Now $C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

imp

Example: Find the matrix multiplication of following matrices Using Strassen's matrix multiplication.

$$= \begin{bmatrix} 2 & 1 \\ 3 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

Sol:

$S_1 = 5-4 = 1$	$P_1 = 2 \times 1 = 2$
$S_2 = 2+1 = 3$	$P_2 = 3 \times 4 = 12$
$S_3 = 3+2 = 5$	$P_3 = 5 \times 3 = 15$
$S_4 = 2-3 = -1$	$P_4 = 2 \times -1 = -2$
$S_5 = 2+2 = 4$	$P_5 = 4 \times 7 = 28$
$S_6 = 3+4 = 7$	$P_6 = -1 \times 6 = -6$
$S_7 = 1-2 = -1$	$P_7 = -1 \times 8 = -8$
$S_8 = 2+4 = 6$	
$S_9 = 2-3 = -1$	
$S_{10} = 3+5 = 8$	

$C_{11} = P_5 + P_4 - P_2 + P_6 = 28 - 2 - 12 - 6 = 8$

$C_{12} = P_1 + P_2 = 2 + 12 = 14$

$C_{21} = P_3 + P_4 = 15 - 2 = 13$

$C_{22} = P_5 + P_1 - P_3 - P_7 = 28 + 2 - 15 + 8 = 23$

Result is $\begin{bmatrix} 8 & 14 \\ 13 & 23 \end{bmatrix}$

20 Sept-2018

Solution of 0/1 Knapsack problem using Dynamic Programming:-

$(\omega_1, \omega_2, \omega_3) = (2,3,3)$ $n=3$

$(P_1, P_2, P_3) = (1,2,4)$; $m=6$

(f_1, f_2, f_3) = Profit-

0	0	0	0
0	0	1	4
0	1	0	2
0	1	1	6
1	0	0	1
1	0	1	5
1	1	0	3
1	1	1	Not Possible

Non dynamic method

let $c[i,\omega]$ be the solution for items $1,2,\ldots,i$ and maximum weight ω if n items are given and m is the capacity of Knapsack then we need to find out.

$c[n,m]$,

Recursive Formula to Calculate $c[i,\omega]$

$$c[i,\omega] = \begin{cases} 0 & \text{if } i=0 \text{ or } \omega=0 \\ c[i-1,\omega] & \text{if } i>0 \text{ and } \omega_i > (\omega) \\ \max[v_i + c[i-1, \omega-\omega_i], c[i-1,\omega]] & \text{if } i>0 \ \& \ \omega_i \le \omega. \end{cases}$$

i \ ω	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	
2	0	0	1	2	2	3	3	
3	0	0	1	4	4	5	6	

$(\omega_1, \omega_2, \omega_3, \omega_4) = (4,3,6,2)$

$(P_1, P_2, P_3, P_4) = (10,15,12,8)$

$m=10$

$n=4$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	10	10	10	10	10	10	10
2	0	0	0	15	15	15	15	25	25	25	25
3	0	0	0	15	15	15	15	25	25	27	27
4	0	0	0	15	25	23	23	25	25	33	33

$(f_1 f_2 f_3 f_4) = (1,1,0,1)$

and Optimal height = 33

21 Sept 2018

Huffman Coding:-

Example: char: frequency

a: 5

b: 35

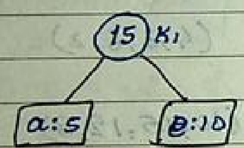
c: 30

d: 20

e: 10

• Firstly extract minimum two values

• and create new node



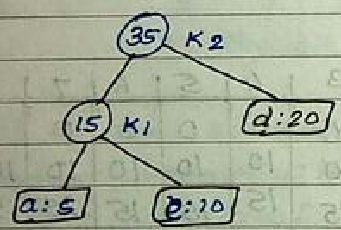
b = 35

c : 30

d : 20

K : 15

new address will insert in queue.

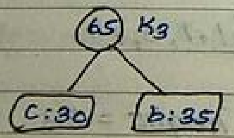


b:35

c:30

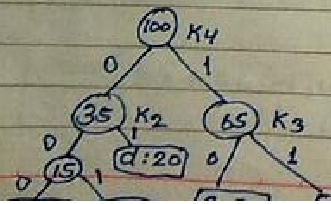
K2:35

K2 will insert in queue



K2 : 35

K3 : 65



Coding:-

a → 000	15 bit	3x5	a → 1100
b → 11	35 bit	35x2	b → 0
c → 10	60 bit	30x2	c → 10
d → 01	40 bit	20x2	d → 111
e → 001	30 bit	10x3	e → 1101
	215 bit		

Bitsaving = 300 - 215
= 85

bcabdebade

decoding 0101100 0111110101100111101

bcabdebade

24 Sept. 2018

Back Tracking:-

It is a method to solve the problems. In Back Tracking, we consider all the possibilities of the solution and goo. with one of these possibilities

At the next level we can also have more than one possibilities for the solution so again we consider one of them and go forward.

At any point if we reached to a deadend solution is not possible from here then we will back track to the previous decision point & check other possibilities. By following this procedure we will reach to the sol. of the problem.

If we required a single sol. then we can stop here otherwise we can consider all the possibilities to findout all the possible solutions.

• Back Tracking is not use for optimization problems.

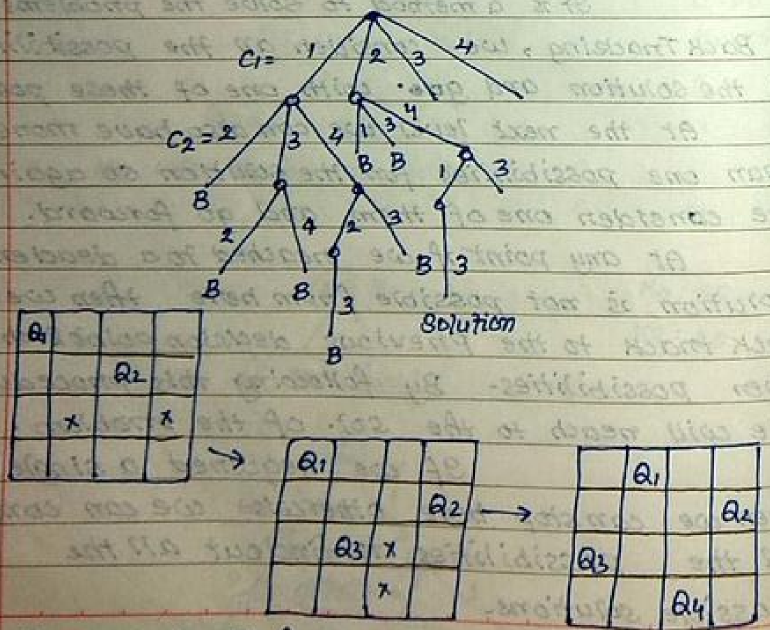
Example n queen problem:-

In this problem we are given a chaseboard of n×n dimension in n queens. We need to put n queens on the chaseboard in such a way so that no two queens can attack to eachother.

We can solve in problem using Back Tracking method.

Bcoz n queens are there so one queen should be put in each row. We need to find out which queen will be put in which column.

$n=4$



* At 3 this will form mirror image

Subset Sum Problem:-

In this problem we are given a S set of elements and the no. M.

We need to find out all the subsets of sets such that the sum of the elements of the subset = M

Example. $S = \{1,2,3,4,7,10,15\}$ $M=17$

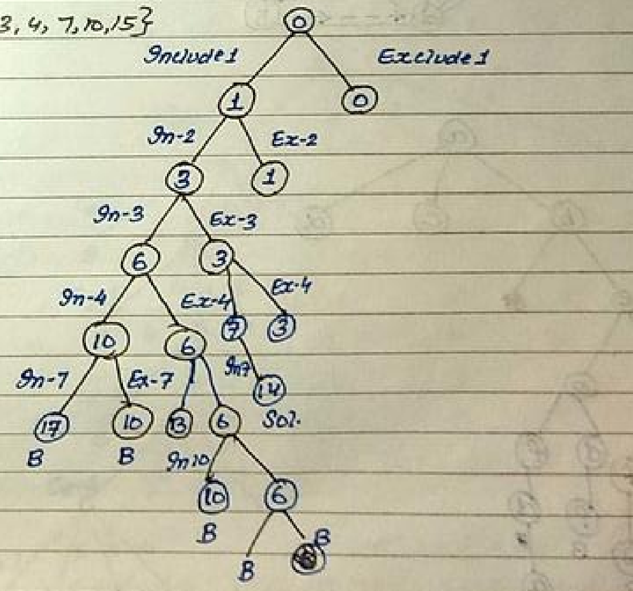
$\{1,2,3,4,7\}$
$\{3,4,10\}$
$\{7,10\}$
$\{2,15\}$
$\{1,2,4,10\}$

[value > m then stop]

$S=\{1,2,3,4,7,10,15\}$

$m=14$

B = Back Track



$\{1,2,4,7\}$

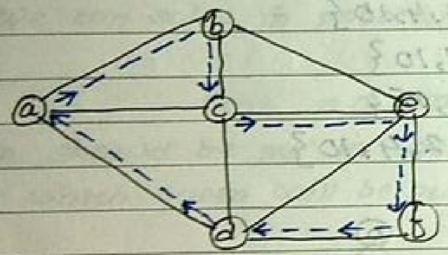
Note:- when value is greater than m then stop it

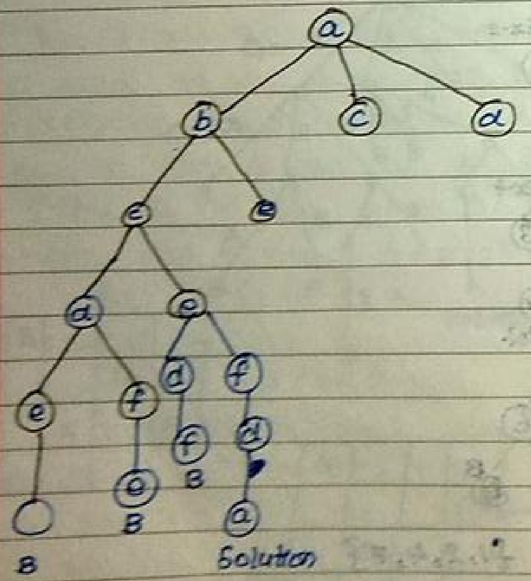
Hamiltonian Circuit:-

We are given a graph $G = (V, E)$ Hamiltonian circuit of the graph is a cycle in the graph that covers all the vertices of the graph once and only once.

We can solve this problem using Back Tracking.

Example.

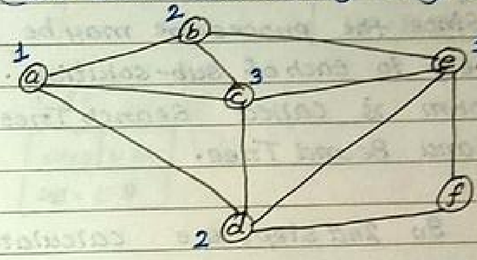




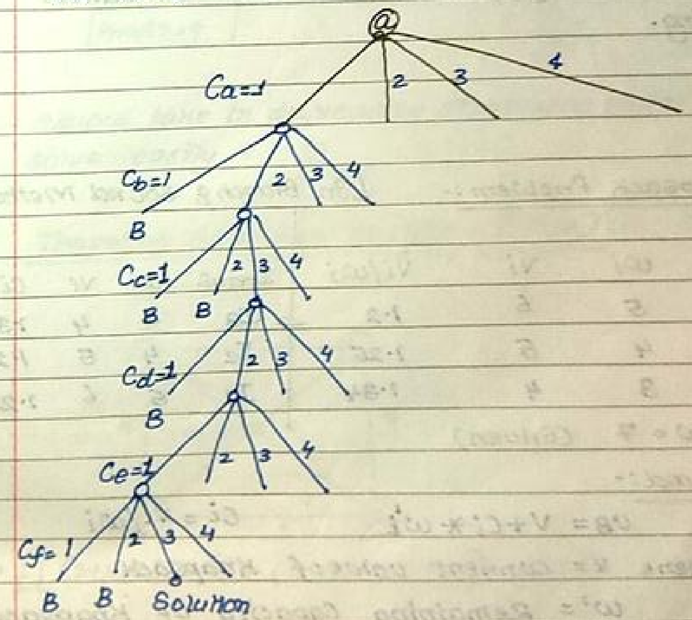
Graph Colouring Problem:-

We are given a graph and a positive no. 'n' the graph colouring problem is to findout whether the nodes of Graph, G can be coloured in such a way so that no two adjacent nodes at the same colour, by using only m colours.

Example.



let $m=4$



So, Graph can be coloured using 3-colours

Branch and Bound Method:-

This procedure or method requires / use two steps.

In the 1st step, we try to cover the feasible ~~solution~~ solution by using smaller feasible sub-solutions, this is called Branching.

Since the procedure may be repeated recursively to each of sub-solution. So the tree form is called Search Tree or Branch and Bound Tree.

In 2nd step, we calculate the upper or lower bounds for the optimal solutions at each branching points, this step is called Bounding.

25 Sept. 2018

0-1 Knapsack Problem:- [In Branch & Bound Method]

imp

Example:-

Items	w_i	v_i	v_i/w_i
I_1	5	6	1.2
I_2	4	5	1.25
I_3	3	4	1.34

Items	w_i	v_i	c_i
I_3	3	4	1.34
I_2	4	5	1.25
I_1	5	6	1.2

$W = 7$ (Given)

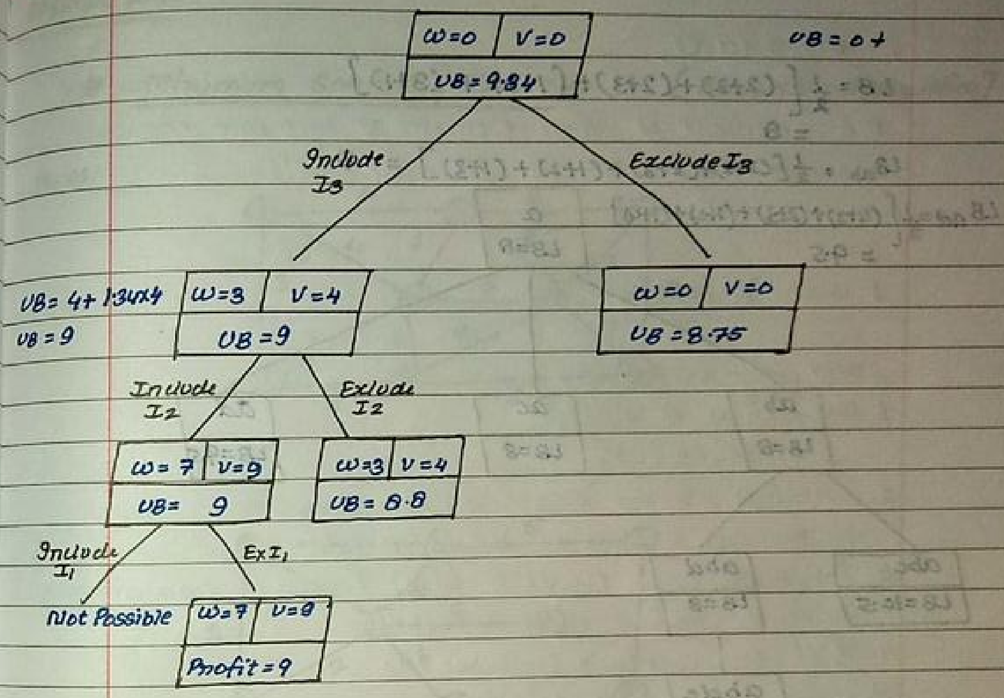
Upper Bound:-

$UB = V + C_i * W'$ $\qquad C_i = V_i/W_i$

where V = current value of Knapsack

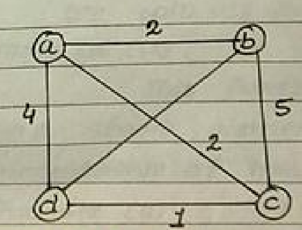
W' = Remaining Capacity of Knapsack

C_i = Cost/unit of the most costly items from remaining one



Note:- Always take in decreasing decreasing order bcoz it solve easily.

Traveling Salesman Problem [TSP]:-



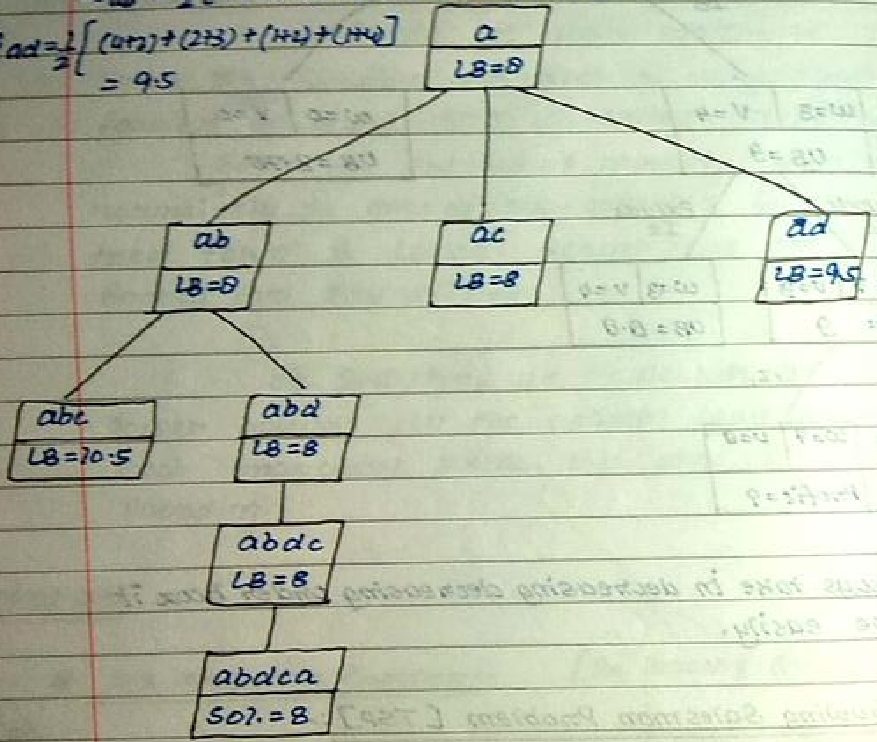
$LB_{abc} = \frac{1}{2}[(2+2)+(2+3)+(5+1)+(1+3)] = 10.5$

$LB_{abd} =$

$LB = \frac{1}{2}[(2+2)+(2+3)+(1+2)+(3+1)]$
$= 8$

$LB_{ab} = \frac{1}{2}[(2+2)+(2+3)+(1+2)+(1+3)] = 8$

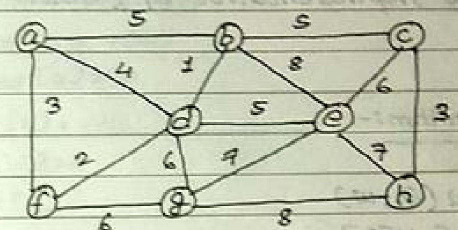
$LB_{ad} = \frac{1}{2}[(1+2)+(2+3)+(1+2)+(1+2)]$
$= 9.5$

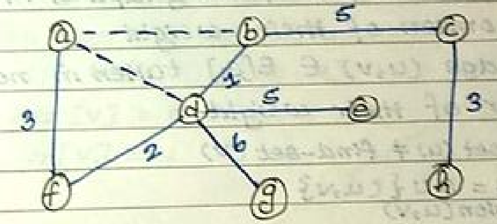


26 Sept 2018

Minimum Spanning Tree :- [Kruskal Algorithm]

Ques:-





weight of Minimum Spanning Tree = $1+2+3+\cancel{3}+5+5+6$
$= 25$

Data Structure for disjoint Set :-

we will use 3-operation in this.

1. Make_Set (u)
This function will createa new set with a single element u, and u will called representative of this sets.

2. find-set (u)
It will return the representativeof the set in which element u is present.
If find-set (x) = find-set (y) that means x&y belong to same set.

3. Union (u,v)

It will unite the two sets in which the element u and element v are present and make a new representative of new sets.

Kruskal Algorithm:-

0. MST_Kruskal (G, ω)
   S = φ
1. for each u ∈ V[G]
   Make-set (u);
2. arrange all the edges of the graph G in non decreasing order of their weight.
3. for each edge (u,v) ∈ E[G] taken in non-decreasing order of their weight.
4. { if find-set (u) ≠ find-set (v)
5. S = S ∪ {(u,v)}
   union(u,v)
   }
6. return S

Example

{a} {b} {c} {d} {e} {f} {g} {h}

{a} {b,d} {c} {e} {f} {g} {h}

{a} {b,d,f} {c} {e} {g} {h}

{a,b,d,f} {c} {e} {g} {h}

{a,b,d,f} {c,h} {e} {g}

{a,b,c,d,f,h} {e} {g}

{a,b,c,d,e,f,h} {g}

{a,b,c,d,e,f,g,h}

27 Sept. 2018

Prim's Algorithm:-

MST – Prim (G, ω, s)
for each u ∈ V[G]
    key [u] = ∞
    π [u] = NIL
  key [s] = 0
  Q = V [G]
  while (Q ≠ φ)
  {
    u = Extract_min (Q)
    for each v ∈ Adj [u] and v ∈ Q
      if ω(u,v) < key [v]
      key [v] = ω (u,v)
      π [v] = u
  }

Example:- Using Prim's

Fig-1

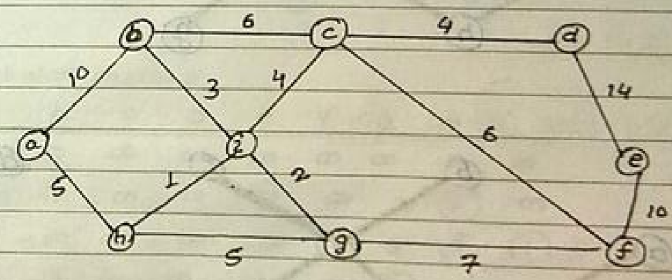


Fig-2

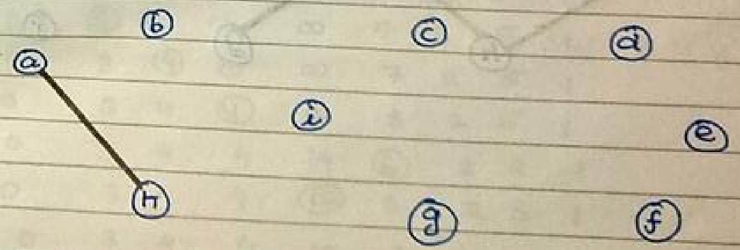


Fig-3

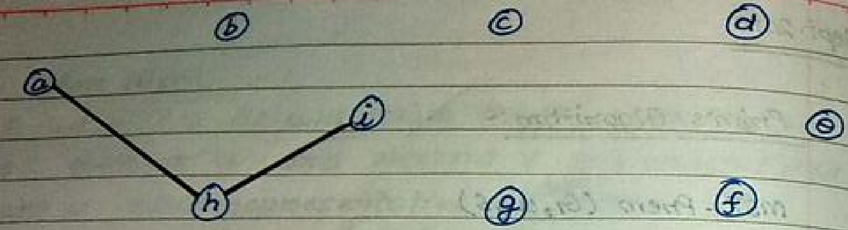


Fig-4

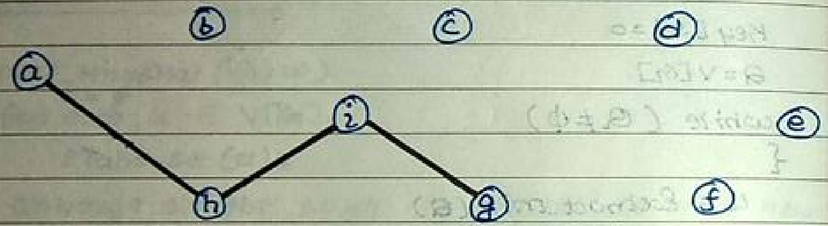


Fig-5

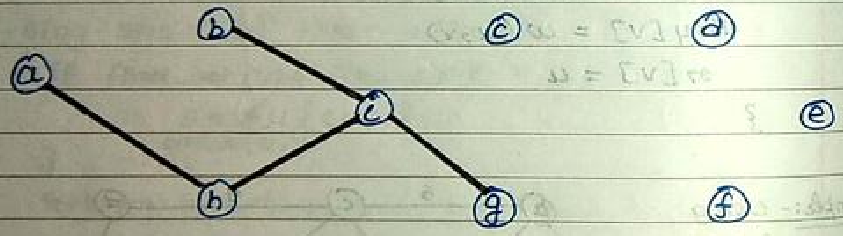


Fig-6

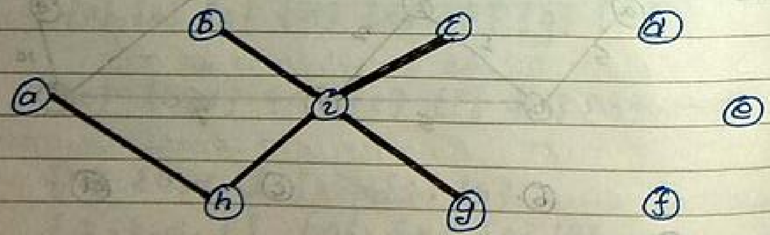


Fig-7

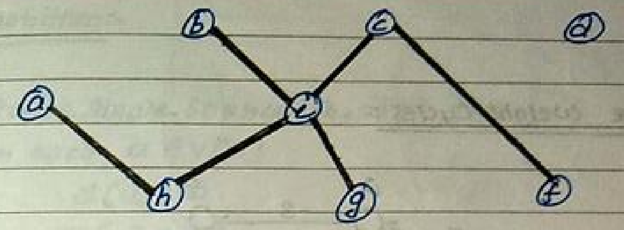


Fig-8

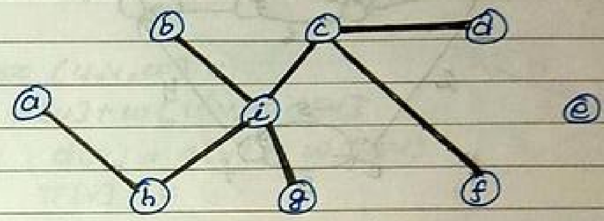
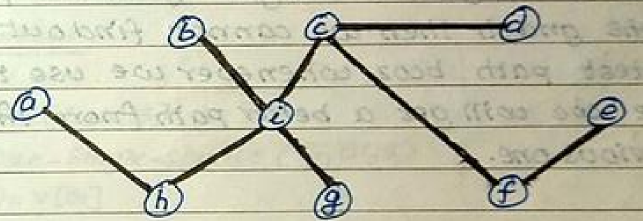


Fig-9

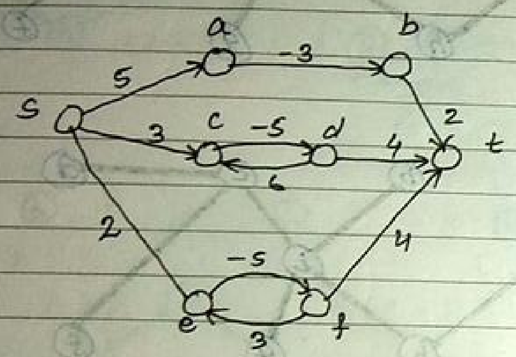


method-2 Tabular method

Q	a	b	c	d	e	f	g	h	i
	0	∞	∞	∞	∞	∞	∞	∞	∞
a	0	10	∞	∞	∞	∞	∞	5	∞
h	0	10	∞	∞	∞	∞	6	5	1
i	0	3	4	∞	∞	∞	2	5	1
g	0	3	4	∞	∞	7	2	5	1
b	0	3	4	∞	∞	7	2	5	1
c	0	3	4	4	∞	6	2	5	1
d	0	3	4	4	14	6	2	5	1
f	0	3	4	4	10	6	2	5	1
e	0	3	4	4	10	6	2	5	1

28 Sept 2018

Negative weight cycles :-



In this, graph is the cycle with negative weight if negative weight cycles are present in the graph then we cannot find out the shortest path bcoz whenever we use the cycle we will get a better path from the previous one.

Single Source Shortest Path :-

1. Dijkstra Algorithm :-

We can apply the Dijkstra Algorithm if all the weights of the graph are +ive bcoz it cannot find out the negative weight cycle graph.

Algorithm :-

```
Initialise_Single_Source( G, w, s)
for each  u ∈ V[G]
        d[u] = ∞
        π[u] = NIL
        d[s] = 0
```

Relax Algo :-
```
Relax (u, v, w)
 if d[u] + w[u,v] < d[v]
        d[v] = d[u] + w(u,v)
        π[v] = u
```

Algorithm :-

```
Dijkstra (G, w, s)
Initialize-Single-Source (G, w, s)
   Q = V[G]
 while( Q ≠ Φ)
{
   u = Extract_min(Q)
   for each v ∈ Adj[u]
      Relax ( u, v, w)
}
```

Light line → - - - - -

Example:

1method:-

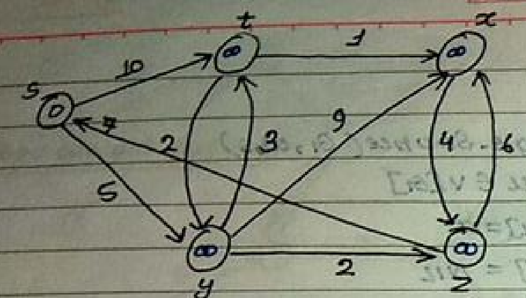


Fig-1

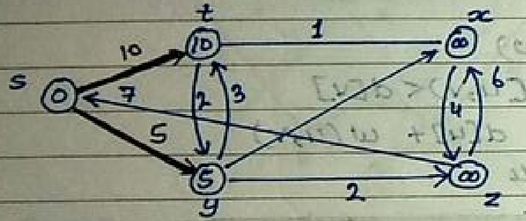


Fig-2

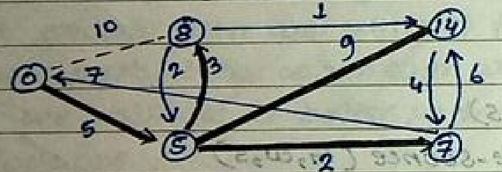


Fig-3

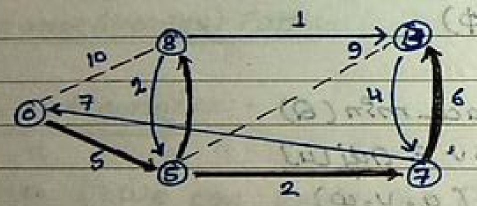


Fig-4

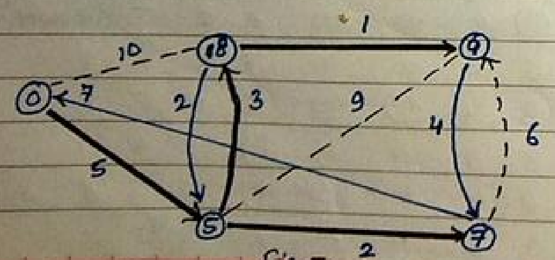


Fig-5

01 Oct. 2018

2ndmethod :-

	s	t	x	y	z
	0	∞	∞	∞	∞
s	0	10	∞	5	∞
y	0	8	14	5	7
z	0	8	13	5	7
t	0	8	9	5	7
x	0	8	9	5	7

Bellman Ford Algorithm:-

Bellman_Ford (G, ω, s)

1. Initialize-Single-Source (G, ω, s)
2. for i = 1 to |V[G]| -1
3. for each edge (u,v) ∈ E[G]
4. Relax (u, v, ω)
5. for each edge (u,v) ∈ E[G]
6. if d[u] + ω[u,v] < d[v]
   return false.
7. return True

• less than change

Example:-

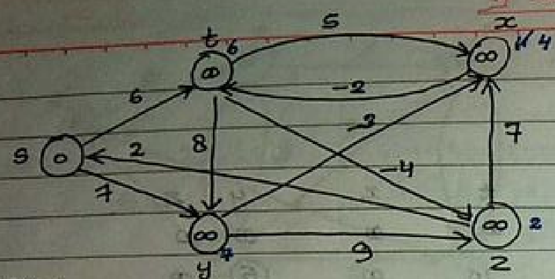


Fig-1

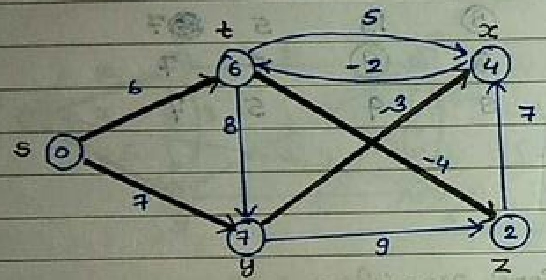


Fig-2 (1pass)

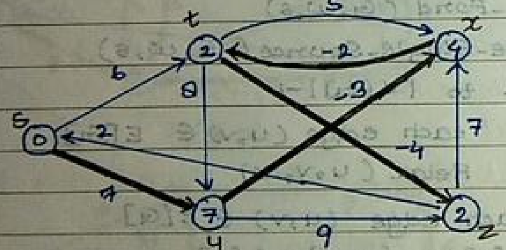


Fig-3 (2nd Pass)

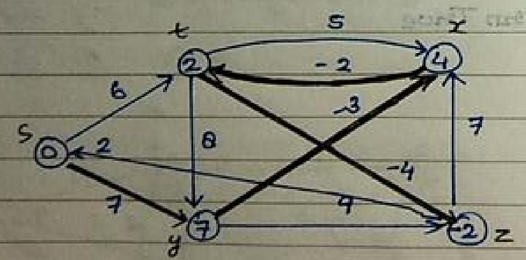


Fig-4 [3rd Pass]

Fig-5 - No change

n = no. of vertices

All Pairs Shortest Path :-

1. Floyd Warshall Algorithm :-

Floyd_warshall (D^0)

n = rows (ω)

$D^0 = \omega$

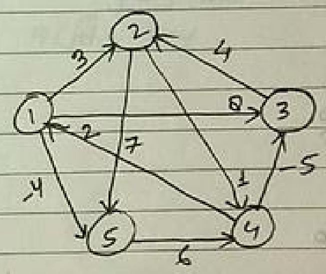
for $k = 1$ to n

for $i = 1$ to n

for $j = 1$ to n

$$d_{ij}^{k} = \min\left(d_{ik}^{k-1} + d_{kj}^{k-1},\ d_{ij}^{k-1}\right)$$

Example



Take Select Col.
Add Select Row

add to Selected Row1

$$D^0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^0 = \begin{bmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & N & 4 & N & N \\ N & N & N & 5 & N \end{bmatrix}$$

$$D^1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^1 = \begin{bmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{bmatrix}$$

← add 0 to Selected Row1
← add ∞ ,, ,, ,, 1
→ add ∞ ,, ,, ,, 1
← add ,, ,, ,, 1
← add ,, ,, ,,

$$D^2 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \Pi^2 = \begin{bmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{bmatrix}$$

m.imp
Numerical + algorithm

Shell Sort :-

[Advance Version of Insertion Sort]

10, 6, 2, 8, 5, 15, 1, 3, 9

(gap b/w = 4)

$inc = \frac{\text{no. of element}}{2}$

$inc = \frac{9}{2} = 4.5 = 4$

inc = 4

increment 5, 6, 2, 8, 9, 15, 1, 3, 10

5, 6, 2, 8, 9, 15, 1, 3, 10

5, 6, 1, 8, 9, 15, 2, 3, 10

5, 6, 1, 3, 9, 15, 2, 8, 10

increment = 2

1, 6, 2, 3, 5, 15, 9, 8, 10

1, 3, 2, 6, 5, 8, 9, 15, 10

increment = 1

1, 2, 3, 5, 16, 8, 9, 10, 15

~~algorithm~~

05 Oct. 2018

Algorithm Shell Sort :-

Shell Sort (A, n)

1. inc = ⌊n/2⌋
2. while inc ≥ 1

```
{
  for i = inc+1 to n
   key = A[i]
   j = i - inc
   while (j > 0 and A[j] > key)
        A[j+inc] = A[j]
        j = j - inc
   A[j+inc] = key
}
```

Ques- Find out optimal parenthesisation for the chain of matrices

$A_1(2\times5)$, $A_2(5\times4)$, $A_3(4\times2)$, $A_4(2\times6)$, $A_5(6\times3)$

P	2	5	4	2	6	3

- starting of all matrices dimension and last dim. of last matrix
- 1st Row is 0, Fixed for all matrices.
- multiply 3-dimensions
- For 3, start from $(i-1)$ and left gap of 3.
- For 2, " " $(i-1)$ " " " of 2

UNIT - 5

09 Oct-2018

String Matching:-

We are given a String T of length n and a pattern P of length m such that $m \le n$.

We need to find out the occurrence of pattern P in the string T as a substring.

T = abababacaba

P = ababaca

Pattern P occurs in T after 2 shifts

Algorithm:-

Time Complexity = $O(m(n-m+1))$

if $m < n$ then $T_{comp} = O(n)$

if $m \ge n$ $T(n) = O(n)$

if $m = n/2$ $T(n) = O(n^2)$

Naive String Matching

```
Naive-String-Matcher (T,P)
  n = length (T)
  m = length (P)
    for s=0 to n-m
{ if P[1,m] = T[s+1 .... s+m]
    print "pattern occurs with shift", s;
}
```

Robin Karp Algorithm:-

[[illegible] remainder]

T =	5	2	8	3	2	3	7	5	7	2	5	8	7	3

P = 2 3 7   q = 7

p = 6   7 | 237 | 6

	0	1	2	3	4	5	6	7	8	9	10		
t	3	3	6	1	6	4	1	5	4	6	6	6	3

Valid match

Spurious hit

$10\times(528-5\times10^2)+3$
$= 283$
$10(283-2\times10^2)+2$
$= 832$

1st next term

* $t_{i+1} = 10[t_i - T[i] \times 10^{m-1}] + T[m+i]$

  $t_2 = 10(633-6\times10^2)+1 = 331$

* $t_{i+1} = (10[t_i - T[i] \times h) + T[m+i]) \bmod q$ where $[h = 10^{m-1} \bmod q]$

$m = length$

10 Oct-2018

Example:-

	1	2	3	4	5	6	7	8	9
T	5	3	2	1	6	3	2	1	7
t	0	6	6	1	2	6	0	+	

$\Rightarrow -k \bmod q$

$\Rightarrow q - (k \bmod q)$

$P = 321 \quad p = 6$

$q = 7$

2nd method:-

Formula- $t_{i+1} = (10(t_i - T[i] \times h) + T(m+i)] \bmod q$ where $h = 10^{m-1} \bmod q$

$t_2 = (10(0 - 5 \times 2) + 1) \bmod 7$

$= -99 \bmod 7$

$= 7 - (99 \bmod 7)$

$= 7 - 1$

$= 6$

$t_3 = (10(6 - 3 \times 2) + 6) \bmod 7$

$= 6$

Algorithm:- [Rabin Karp]

```
Rabin_Karp_matcher (T,P,q,d)
 n=length (T)
 m=length (P)
 h= d^(m-1) mod q
 p=0
 t0 =0
 for i=1 to m
   p = (dp + P[i]) mod q
   t0 = (dt0 + T[i]) mod q
 for s=0 to n-m
   { if p=ts
      { if P[1...m] = T[s+1....s+m]
```



```
      }
      print "Pattern occurs with shift" s
   if (s<n-m)
      ts+1 = (d ( ts - T[s+1]xh) + T [s+m+1]) mod q
   }
```

- Time Complexity $= \Theta(n-m+1)$

19 Oct-2018

String Matching with Finite Automata:-

Algorithm:-

```
Compute_Transition_Function (P,Σ)
 m= length (P)
 For (q=0 to m)
    For each a ∈ Σ
       K = min(m+1, q+2)
       repeat
          K=K-1
       until Pk ] Pqa
       δ (q,a)=K
 return δ.
```

imp # Finite_Automata_matcher (T,δ,m)

```
 n= length (T)
 q=0
 For i=1 to n
    q = δ(q,T[i])
    if q=m
    print "Patten occurs with shift", i-m;
```

24 Oct-2018

Knuth Morriss Pratt Algorithm [KMP] :-

Algorithm :-

```
Compute_Prefix_Function(P)
  m = length(P)
  π[1] = 0
  K = 0
  For q = 2 to m
      while K>0 and P[K+1] ≠ P[q]
          K = π[K]
      if P[K+1] = P[q]
          K = K+1
      π[q] = K
  return π
```

Formula :-

$\pi[q] = \min\{K : K<q \text{ and } P_K \sqsupset P_q\}$

P = ababaca

q	1	2	3	4	5	6	7
P[q]	a	b	a	b	a	c	a
π[q]	0	0	1	2	3	0	1

Note :- Match from (diagonal +1) and that terms match with other one.


```
KMP_Matcher (T, P)
  n = length (T)
  m = length (P)
  π = Compute_Prefix_Function (P)
  q = 0
  For i = 1 to n
      while q>0 and P[q+1] ≠ T[i]
          q = π[q]
      if P[q+1] = T[i]
          q = q+1
      if q = m
          print "Pattern Occurs with Shift" i-m
          q = π[q]
```

25 Oct-2018

NP Completeness :-

On the basis of time complexity of the algorithm we can classify the algorithm in different classes.

1. P-Class
2. NP-Class
   (a) NP-Complete
   (b) NP-Hard

1. P-Class :-

The Algorithm which can exicute in polynomial running time are called P-type problem.

a. The Algorithms that have the time complexity $O(n^K)$ for $K \geq 0$ are called polynomial time solveable algorithms.

OR

a. The problems which can be solved in polynomial type are called P-type problems.

Most of the problems that we have studied are polynomial time soluable problem.

Eg- Sorting, Searching, Shortest Path Algo, MST, LCS, matrix chain multiplication,

2. NP- Class :-

The problem which can solved in polynomial time or can not be but, if they are varifiable in polynomial time then they are called NP-type problems.

That means if we are given a certificate of the solution then we can verify the solution in polynomial time.

The problems which can be solved in polynomial time, must be verifiable in polynomial time. So all the P-type problem are also the NP-type.

i.e., $P \subseteq NP$.

NP class can be further divided into two class.

(a) NP complete

(b) NP-Hard.

15 Oct 2018

(a) NP- Complete :-

That NP problems which cannot be solve in polynomial time and for that it is not proved that they will never solve in polynomial time are called NP-Complete Problems.

This is an open class, today we don't have polynomial time solution for this problems but in future we make at polynomial time sol. for this problems bcoz till data no one ~~can~~ could proof that the polynomial time sol. never be available for this problems.

To proof any Problem NP-Complete we show that the problem is reducable to any proved NP-Complete problem.

So all the NP-Complete Problems are reducable to each other.

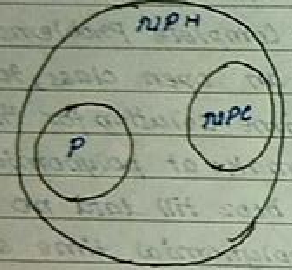
In future, if we will get polynomial time solution for any NP-Complete Problem then the polynomial time sol. will be available for every NP-Complete Problem. In that case the whole NP-Complete class will be dissolve in P-class.

Eg. TSP, Hamiltonian Circuit Problem, Circuit Satisfiability, Vertex cover problem, Set covering problem, etc.

iii) NP-Hard Class/Problem:-

The NP Problem which cannot be solve in polynomial time & for that it is proved by the mathematician that they will never solve in polynomial time called NP-Hard Problem.

Eg. Halting Problem



Figure

Eg- Halting Problem:-

19 Oct. 2018

Approximation Algorithms:-

We know that NP-Complete Problems cannot be solved in polynomial time, so we try to find out the approximate sol. for these problem in polynomial time. So approximation algorithms polynomial time sol. for the problem which cannot be solved in polynomial time.

imp 1. Vertex Cover Problem:-

- This problem is to find out a vertex cover of min. size in a given undirected graph.
- The vertex cover of an undirected graph $G = (V, E)$ is a subset V' $[V' \subseteq V]$ such that if (u, v) is an edge of G then either $u \in V'$ or $v \in V'$ [or both]

The size of the vertex cover is no. of vertices in it.

Example:-

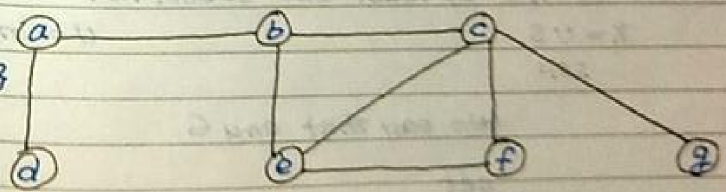
$E' = \{a,b,c,d,e,f,g\}$

$C = \{a,b\}$

$E' = \{(c,e)(c,f)(c,g)(e,f)\}$

$C = \{a,b,c,e\}$

$E' = \phi$



- This problem is NP-Complete problem so we cannot find out the polynomial time sol. for this problem but we can design an approximation algorithm for this.

Algorithm:-

Approx_vertex_Cover(G)

$C = \phi$

$E' = E[G]$

while ($E' \neq \phi$)

{ Let (u,v) be an arbitrary edge of E'

$C = C \cup \{u,v\}$

remove from E' every edge incident on either u or v.

}

return C

2. Set Covering Problem:-

An instance (X,F) of the set covering problem consist of a finite set, X and a Family, F of subsets of X, such that every element of X ∈ at least one subset in F that is

$X = \bigcup_{S \in F} S$ U = Union

~~We say that any~~ G

A set cover C subset of F [$C \subseteq F$] is cover all the elements of X that is

$X = \bigcup_{S \in C} S$ U = Union

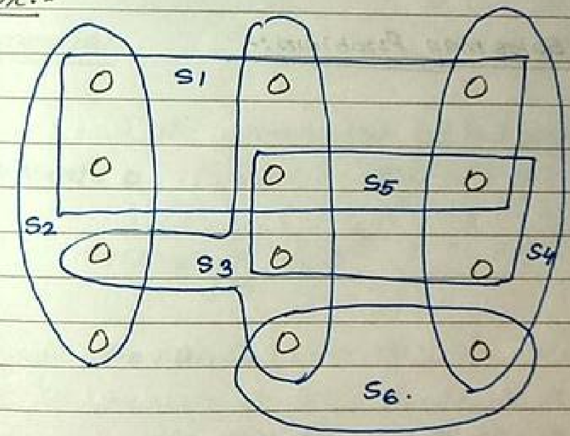
So we can say the set cover of any instance of the problem (X,F) is if the subset of F that can cover all the elements in X.

Our problem is to find out the minimal set cover.

In the given Eg. the min. set cover is 3. [S2, S3, S4] having size 3. but to find out this soln. we need to use an NP-Complete Algorithm.

So we will use an approximation algorithm called Greedy_set_covers that will give us approximate sol. in polynomial time.

Example:-



Algorithm:-

Greedy_Set_Cover (X,F)

$C = \phi$

$U = X$

while $U \neq \phi$

Select an $S \in F$ such that $|S \cap U|$ maximize

$U = U - S$

$C = C \cup \{S\}$

return C

3. Travelling Salesman Problem:-

Algorithm:-

Approx_TSP_Tour (G,W)

1. Select a vertex $r \in G$ as a root vertex
2. Compute a minimum spanning tree T for G from root r using MST_Prim (G,W,r)
3. Let H be a list of vertices ordered according to when they are visited in a preorder tree walk of T
4. Return the Hamiltonian Cycle H

30 Oct 2018

Polynomial

A polynomial in the variable x represents a function A(x) as a formal sum

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$a_0, a_1, a_2 \ldots \ldots a_{n-1}$

We call the values $a_0, a_1, a_2 \ldots a_{n-1}$ as the coefficient of the polynomial

The integer is strictly greater than the degree of a polynomial is called degree bound of that polynomial.

Representation of Polynomial:-

1. Coeff. of Representation:-

A coeff. representation of Polynomial

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

of degree bound n is a vector of coefficient

$a = (a_0, a_1, a_2 \ldots\ldots a_{n-1})$

Example:-

$A(x) = 3+5x^2+2x^4$

$a = (3,0,5,0,2)$
$(a_0, a_1, a_2, a_3, a_4)$

We can evaluate the polynomial using the Horner's Rule.

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \ldots\ldots x_0(a_{n-2} + x_0(a_{n-1}))))$$

$$A(2) = \underset{3}{a_0} + 2(\underset{0}{a_1} + 2(\underset{5}{a_2} + 2(\underset{0}{a_3} + 2(\underset{2}{a_4}))))$$

$A(2) = 55$

From here it is clear that the time complexity to evaluate the polynomial of degree bound n is $\Theta(n)$.

2. Point Value Representation:-

A point value representation of a polynomial $A(x)$ of degree bound n is a set of n point value pairs.

$$\{(x_0, y_0), (x_1, y_2), (x_2, y_2), \ldots\ldots (x_{n-1}, y_{n-1})\}$$

Eg- $A(x) = 3+5x^2+2x^4$

$\{(0,3), (1,10), (2,55), (3,\cancel{1}\,,)\}$

Complex Roots of Unity:-

$\omega^n = 1$ $[\omega^3 = 1, -1, i, -i]$

A complex nth root of unity is a complex no. omega such that $\omega^n = 1$.

There are exactly n complex nth roots of unity $e^{2\pi i k/n}$ $i = 0, 1, \ldots n-1$

where $e^{iu} = \cos(u) + i\sin(u)$

$\omega_n = e^{2\pi i/n}$ is called the Principle nth root of unity, All other complex roots are

$\omega^0, \omega^1, \omega^2, \ldots\ldots \omega^{n-1}$

Algorithm:-

Discrete Fourier Transformation (DFT)

let the polynomial

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

is given in coefficient form

$$a = (a_0, a_1, a_2 \dots a_{n-1})$$

let us define the result y_K for $K = 0, 1 \dots n-1$ while

$$y_K = A(\omega_n^K)$$
$$= \sum_{j=0}^{n-1} a_j (\omega_n)^{Kj}$$

The vector $y = (y_0, y_1 \dots\dots y_{n-1})$ is called discrete fourier transform (DFT) of the coefficient vector.

$$a = (a_0, a_1, \dots\dots a_{n-1})$$

we can also write $y = DFT_n(a)$

imp # Fast Fourier Transform [FFT]:-

By using a method known as the Fast Fourier Transform. We can compute the $DFT_n(a)$ in $\Theta(n \log n)$ time as oppose to $\Theta(n^2)$ time of the state forward method.

The FFT method uses divide and conquer strategy, using the even indexed and odd indexed coeff. of $A(x)$ seperately to define two new polynomials ~~$A^0[x]$ and $A^1[x]$~~ $A^{[0]}(x)$ and $A^{[1]}x$ of degree bound $n/2$

$$A^{[0]}(x) = a_0x^0 + a_2x^1 + a_4x^2 \dots\dots a_n x^{n/2-1}$$
$$A^{[1]}x = a_1x^0 + a_3x^1 + a_5x^2 \dots\dots a_{n-1} x^{n/2-1}$$

We can define $A(x)$ using above two polynomials as follows

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \quad — ①$$

So the problem of evaluating $A(x)$ at $\omega_n^0, \omega_n, \omega_n^2 \dots \omega_n^{n-1}$ reduces to

Evaluating degree bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points $(\omega_n^0)^2, (\omega_n^1)^2 \dots (\omega_n^{n-1})^2$ and combine the result according to eq-①

Algorithm:-

Recursive FFT (a)
n = length (a)
If $n == 1$
  return a
$\omega_n = e^{2\pi i/n}$
$\omega = 1$
$a^{[0]} = (a_0, a_2, \dots\dots a_{n-2})$
$a^{[1]} = (a_1, a_3, \dots\dots a_{n-1})$
$y^{[0]}$ = Recursive_FFT($a^{[0]}$)
$y^{[1]}$ = Recursive_FFT ($a^{[1]}$)
for $K = 0$ to $n/2-1$
  $y_K = y_K^{[0]} + \omega y_K^{[1]}$
  $y_{K+n/2} = y_K^{[0]} - \omega y_K^{[1]}$
  $\omega = \omega\omega_n$
return y.