07620315615
sunny007.it@gmail.com

Design & Analysis of Algorithm

1. Analysis of algorithm & Asymptotic notations.
2. Design strategies
   - Divide & conquer
   - Greedy methods
   - dynamic programming.
3. Biconnected components. Art points, Bridges
4. Graph technique
5. Heap & Heapsort.
6. P, NP, Hard, Np-complete concepts.

Text Books

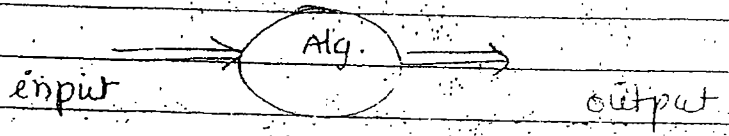
1. fundamentals of computer Algorithm — Sahani.
2. Algorithm design - Tarmassia
3. Introd. to Algo - Corman.

Algorithm:

consists of finite set of steps to solve a prog.

may contain one or more operators
↳ definite (clear)
↳ effective.

Every algorithm takes 0 or more inputs
every algorithm expected to produce atleast 1 output.



* Approaches for solving problems

1. problem ~~(design)~~ defination.
2. Condition/ Requirement specification
   L constraints
3. Design
4. expression in the form of Algorithm/ flowchart
5. validate (algo. which is used to check the algo satisfying all requirements)
6. Analysis
7. program development.
8. Testing
9. Debugging.

Algorithm is procedure / mechanism whic transforms given input to desired outpu

③ Need of Analysis :

1 to determine Resource Consuption
 ↳ Time
 ↳ space

2 Making performance Comparision betn mo than one algorithm to find best one

* Time of execution depends on architechu of processor, cpu speed

$x = x + y.$

*) It also depends on the environment
- programming language
- operating system

$x = x + y$ -
↑ Fundamental opertn 1 time on

Time taken by instruction depends on 2 factors

Architechure	environment
time of exc changes	extract time
C, C++, (Java)	Non uniform

Another is its worst case running time has lower order of Growth.

⓪ Types of Analysis :

1 Aprion Analysis :
Analysis depends on m/c and programming language
- uniform output.

2 Postprion Analysis :
Analysis depends upon OS and PL
- Non uniform Result.

⓪ Aprion Analysis :
Its principle is to determining the order of magnitude of Statement / Construct

order of magnitude :-
Means frequency count of the fundamental operation in the Statement

order of magnitude

① $x = x + y$ $\Rightarrow$ 1

② for i=1 to n
    x = x+y $\Rightarrow$ n

③ for i=1 to n
    for j=1 to n $\Rightarrow$ n^2
        x = x+y

$T(S) \propto n$

Time of Statement is proportional to order of magnitude

$T(S) = Cn$

$f(n) = 1 + n + n^2$

Time Complexity depends upon the power of 'n'
whose magnitude is Not more than n^2

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

either n^2 or
less than n^2
(highest order of magnitude)

Asympotic Notations :-

- used for knowing the behaviour of the Algorithm

* purposes

1) How does the function changes with respect to increase the value of i/p

- If f(n) is a function,
how it changes with value of
$n \geq k \ldots \infty$

Called Asympotic notation

$\overset{x}{k}$ ∝ f(n)

② n → time i/p
{worst case, best case, Avg ~~case~~}

2) To find time with fixed size of i/p
ex. if there are 50 elements.

1, 2, 3, ... n

arrangement of elements
n . . . . . 2, 1 < Decreasing
1. 2, 3. . . . . n Increasing
1, 7, 8, 10, n, 5 . . > Random order

it is called finding worst case, best case & average case complexity.

How does the algorithm behaves with the arrangement of input with fixed size

Def^n: **Big-oh (O)** :- let 'f' & 'g' be function f. ~~set of Integer/Reals to Reals~~
f(x) is O(g(x))
iff there exists constants C & K
such that

$$\boxed{f(x) \leq c \cdot g(x)}$$

x > K
c > 0

Def^n: Big-Omega (Ω) : f(x) is Ω(g(x)), iff,
~~Used~~ $\boxed{f(x) \geq c \cdot g(x)}$ c > 0, x > k

Theta (Θ) : f(x) is Θ(g(x)), iff f(x) is O(g(x)) & f(x) is

small-oh (o) f(x) is o(g(x)), iff, f(x) < c . g(x) for all c > 0,

Small-omega (ω) : f(x) is ω(g(x)) iff, f(x) > c . g(x) for all c > 0
n > k

Properties of Asymptotic Notations.

1. Reflexivity:

mapping to itself.

Big-Oh a) (Upper bound) $F(x)$ is $O(F(x))$ ✓ b'coz $a \leq b$

$a = b$

$f(n) = n \; O(n)$

Big-Omega b) (Lower bound) $F(x)$ is $\Omega(f(x))$ ✓ $a \geq b$

Theta c) (High bound) $F(x)$ is $\theta(F(x))$ ✓ $a = b$

Small-oh d) $f(x)$ is $o(f(x))$ ✗ $a < b$

This is not Reflexivity

Small-Omega(ω)

2. Symmetric property:

if $f(x)$ is $O(g(x))$ $a \leq b$

$a \leq b$ Need Not mean $b \leq a$.

∴ O is Not Symmetric

Ω is also Not Symmetric

✓ θ is Symmetric

o is also Not Symmetric

3. Transitivity :

if $f(x)$ is $O(g(x))$
& $g(x)$ is $O(h(x))$
Then
$\Longrightarrow$ $f(x)$ is $O(h(x))$

ex. n is $O(n^2)$
n^2 is $O(n^3)$
Then n is $O(n^3)$

all Notations $O, \Omega, \Theta, o, \omega$
satisfy transitivity.

4. Symmetric Transpose :

if $f(x)$ is $O(g(x))$ Then
$g(x)$ is $\Omega(f(x))$

ex n is $O(n^2)$
n^2 is $\Omega(n)$

O & Ω ✓
o & ω ✓

are symmetrically transpose of each other

5. Trichotomy property:

relationship betn - two function

There is always some relationship betn two functions as

$a < b$, $a > b$, or $a = b$

But this relation have to be same for every value of n as:

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

Trichotomy ✓

$f(n) = \cancel{\theta} n^2$; $g(n) = n^{1+\sin x}$

$\sin x$: −1 to 1 ✗

For some values the relationship changes:

Trichotomy satisfied by some functions.

6. Intersection:

i) $O(f(n)) \wedge \Omega(f(n))$

$\cancel{=} \theta$

= may be or may not be

b'coz

$a \cancel{\leq} b$: $a \geq b$

$o(f(n)) \wedge \omega(f(n)) = \phi$

Never

1\. $\sum_{k=1}^{n} O(n)$ is $O(\,?\,)$

$\Rightarrow$

$f(n) = \sum_{i}^{n} i$ is $O(?)$

$\Downarrow$

Sum of No. upto n $\Longrightarrow$ n^2

$1+2+3+ \dots n$

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

2) algorithm : What (n)

```
{
    if (n = 1) the call A();
    else
        {
            What (n-1);
            Call B(n);
        }
}
```

$A() \to O(1)$
$B() \to O(1/n)$

$\Rightarrow$ let T(n) is Time taken by What (n)

T(n) = C   When n = 1

3) Counter = 0;

~~for (i=1; i<n;~~

1 comparison
Calls to A()

O(1) represent Constant, 1 + Constant

$\therefore T(n) = C$ when $n = 1$

$= T(n-1) + 1/n + 1$ $n \geq 1$

$T(n) = T(n-1) + \frac{1}{n} + 1$ ——— ①

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

Substitute in eqⁿ ①

(1)

(1/n)

$T(n) = T(n-2) +$ ~~[illegible]~~ $+ \frac{1}{n-1} + \frac{1}{n} + 1 + 1$

$=$ $T(n-3)$ ~~[illegible]~~ $+ \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} + 1 + 1$

at some progression
$T(n-k)$ become $T(1)$ then
we can put value C from
Basic eqⁿ

8)m+

$\therefore T(n) = 1 + \frac{1}{n-3} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} + \frac{1+1+1\ldots}{n}$

$= \sum_{i=1}^{n} \frac{1}{i} = \int_{x=1}^{n} \frac{1}{x} = \log$

$= \log n + n$

$= n + \log n$

③ Counter = 0

```
for(i = 1 ; i < n ; ++i)
{
    if (A[i] == 1)    Counter ++
    else
    {
        f(counter);
        counter = 0;
    }
}
```

A[1...n] can take 0 or 1
f(m) is $\Theta(m)$

Cases

① $A[i] = 1$ $O(n)$

② $A[i] = 0$ $O(n)$

③ $A[i] = 1$ for half, & 0 for remainin

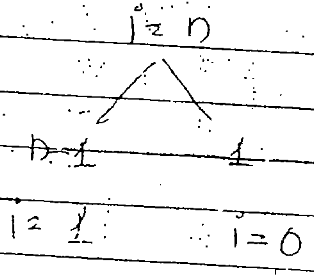
$= n/2 \cdot 1$

$\left(\frac{n}{2}\right)$ remaining → 1st · $\left(\frac{n}{2} + 1\right)$

→ $\left(\frac{n}{2} - 1\right) \cdot 1$

$= \frac{n}{2} + \frac{n}{2} + \frac{n}{2} + 1 - 1$

$= O(n)$



n

all

Big oh

Ex

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

$g(n) \neq \Theta(f(n))$

$g(n) = O(h(n))$

$h(n) = O(g(n))$

a) $f(n) + g(n) = O(h(n) + h(n))$ ✓

b) $f(n) = O(h(n))$ ✓

c) $f(n) = \Theta(g(n))$ ✗

d) $h(n) \neq O(f(n))$ ✓

e) $f(n) \cdot h(n) \neq O(g(n) \cdot h(n))$ ✗

Space Complexity

The space Needed by algorithm sum of following components.

$$S(p) = C + S_p$$

Where

① C represent Constant space which is independent of the characteristics of inputs and output. This includes the instruction space, data str, space for simple variables, space for constants. and so whose sizes are known apriory. It also includes

② Sp represents The instance characteri which is variable part. space for dynamic data st aggregate variables. Which consists of space Needed by (reference) Components variables whose size is dependent on the particular problem instance being solved, The space Needed by reference variable & the recursion stack space.

C = Instruc. space + size of variable, which are known at start.

Variable Independent of Instance

$A(n)$

└ Instance

S_p = Instance dependent variable,
(Instant characteristic function) Rec. stack

Ex

1

```
ABC( int a, int b, int c)
{
    Return (a+b*c);
}
```

$$S(ABC) = \underbrace{\overset{Syst}{C_1} + 3}_{C_2}$$

$$S_p = 0$$

$$\therefore S = C_2 = O(1)$$

2)

```
Procedure Sum( A, n)
{
    int i, sum, n;
        sum = 0
    for ( i = 1 to n)
            sum + = A[i]
    Return (sum);
}
```

Initialization → 2
increment → n
addition → n
assignment → n
comparison → n + 1

$c = c_1 + 3$

$S(sum) = \boxed{C + C_1} \quad C_2$
$+ Sp$

$Sp = n$ // to store n variable A
depends on execution i

$\therefore S(sum) = C_2 + n$

$= O(n)$

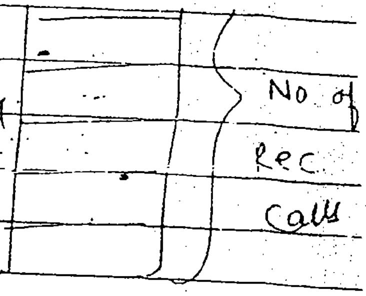
③ procedure Rsum(A, n)

```
{
    if ( n==0) Return (0);
    else
        Return ( Rsum ( A, n-1) + A[n]);
}
```

$\Rightarrow S(Rsum) = C +$

$Sp = n + \underline{\text{~~No.~~ of calls}}$ ~~mem. Per call~~

d



depth of space

$\therefore$ depth = No of calls
each call req 1 word of mem.

$\therefore$ n Recursive calls

$S(RSum) = C + n + n$

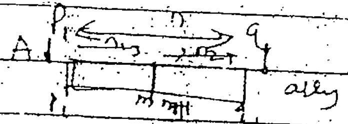
n size Array   depth of stack

$= c_1 + 2n$

$= O(n)$

④ Divide and Conquel :

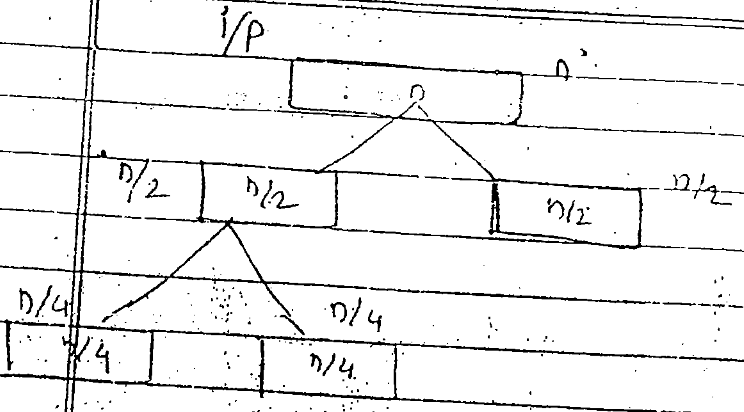
General method :



```
~~procedure DANDC (P, A, P1, q1, n,)~~
{
}

Procedure DANDC (P, A, n, p1, q1)
{
   if (SMALL(P, A, P1, q1)) then
      return (G(P1, q1))
   else
      m ← DIVIDE(P, A, n, P1, q1)
      return (COMBINE( DANDC(P, A, n1, P1, m),
                       DANDC(P, A, n2, m+1, q1)),
}
```

STACK



DANDC Recurrance Relation : (DANDC Mathema
(Model Re

⟹ let T(n) represent Time Complexit
of the procedure DANDC (n)

$T(n) = g(n)$, n is sma

$g(n)$: Time taken by solving g(p.q

n is large

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

↳ Time to Combin
to halves

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

$$\boxed{T(n) = a\,T(n/b) + f(n)}$$

General eqⁿ

```
procedure Straight min max ( A, n, min, max)
    min ← max ← A[1]
    for( i ← 2 to n )
    {
        if ( A[i] > max ) then
                max ← A(i)
    else if → end if
        if ( A[i] < min ) then
                min ← A(i)
        end if
    }
```

in else if case

Uniform case (worst, best, Avg.) same for all — Total No. of Comparison $= 2(n-1)$

When modified to else

① Worst case : elements are in Decreasing
$2(n-1)$ or $2(n-1)$

② Best case Increasing order
$(n-1)$

A 12 8 -10 34 6 -3 39 48 52

IX)

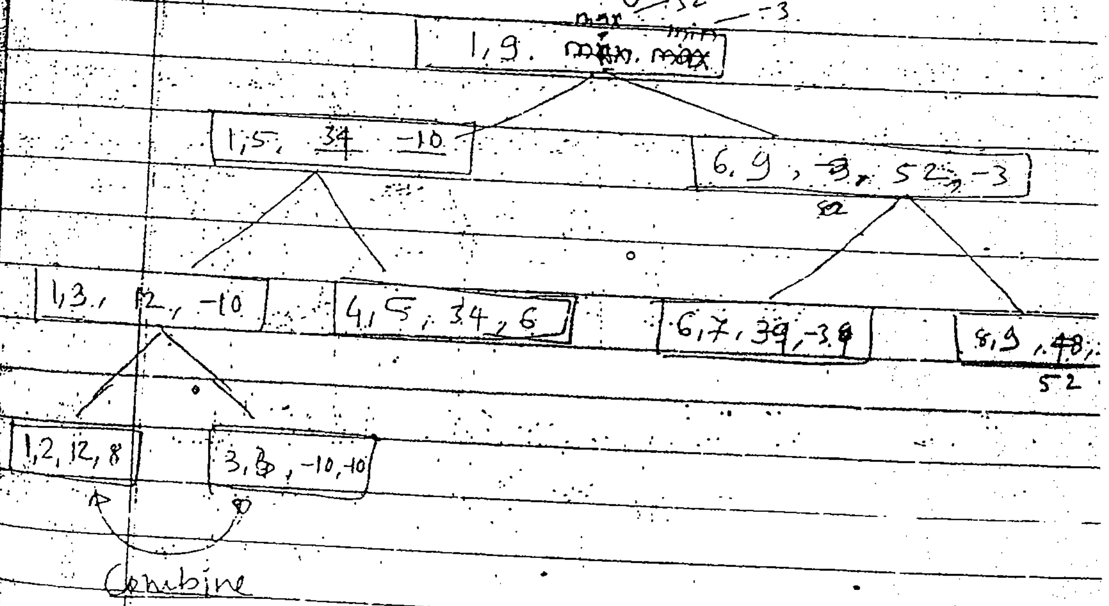
Arg. case : half are Increasing other decreasing

$(n-1) + n/2$

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

Recursive DAD

for Finding Min, Max



only 2 comparisions

sing order

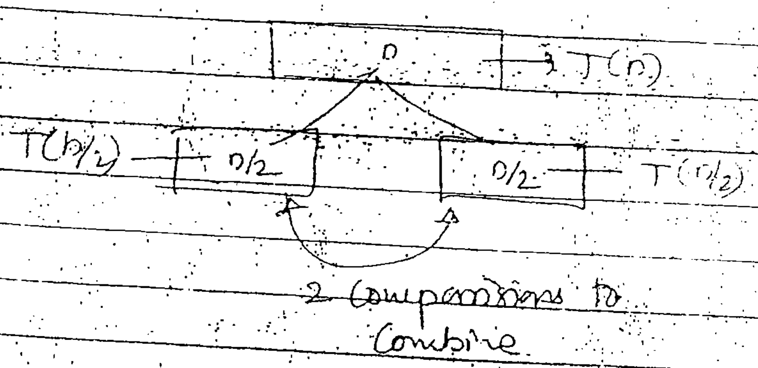
(n-1)

⇒ let $T(n)$ represent total No. of element comparisons in DivideAndC Max min (n)

$$T(n) = 1 \qquad n = 2$$

$$= 2T(n/2) + 2 \qquad n > 2$$

When $n > 2$,



Let $n = 2^k$

$$T(n/2) = 2T(n/4) + 2 \text{ ——— } ①$$

$$T(n) = 2\left[2T(n/4) + 2\right] + 2$$

$$= 4T(n/4) + 4 + 2 \text{ ——— } ②$$

$$= 8T(n/8) + 8 + 4 + 2 \text{ ——— } ③$$

$$T(n) = 2^3 T(n/2^3) + \sum_{i=1}^{3} 2^i$$

$$2^4 T(n/2^4) + \sum_{i=1}^{4} 2^i$$

$$= 2^k T(n/2^k) + \sum_{i=1}^{k} 2^i$$

$$= 2^{k-1} T(n/2^{k-1}) + \sum_{i=1}^{k-1} 2^i$$

after dividing
when $\boxed{n=2}$

$\boxed{T(n) = 1}$

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

$$= \frac{2(2^{k-1}-1)}{2-1}$$

$$= 2^k - 2$$

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

$$= 2^{k-1} \cdot T(2) + 2^k - 2$$

$$= \frac{2^k}{2} + 2^k - 2$$

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

$$\boxed{T(n) = \frac{3n}{2} - 2} \quad n > 2$$

DANDC

Recursive $\left(\frac{3n}{2}-2\right)$ Incr / decreasing

All

plain $(n-1)$ Increasing

$(2n-1) \longleftrightarrow \frac{3n}{2}-2$ ✓

$\frac{3n}{2}-1 \longleftrightarrow \frac{3n}{2}-2$ ✓

When elements are in Increasing order iterative method perform good.

⇒ But divide and conquer require extra memory for stack storage

$S(\text{plain}) = c_1 + n$

$S(\text{DANDC}) = c_3 + n + \underbrace{\log n}_{\text{extra}}$

logn if 8 elements are there depth of stack is 3.

2. DANDC- Mergsort

Merging

L_1(5 9 11 15 18) l_2 (4 8 12 20 22)
i   j

l_3(4 5 8 9 11 ....

Gate If L_1 Contains n elements, l_2 Contains element. What is O( )?

⟹ max Comparisions will be O(m+n)

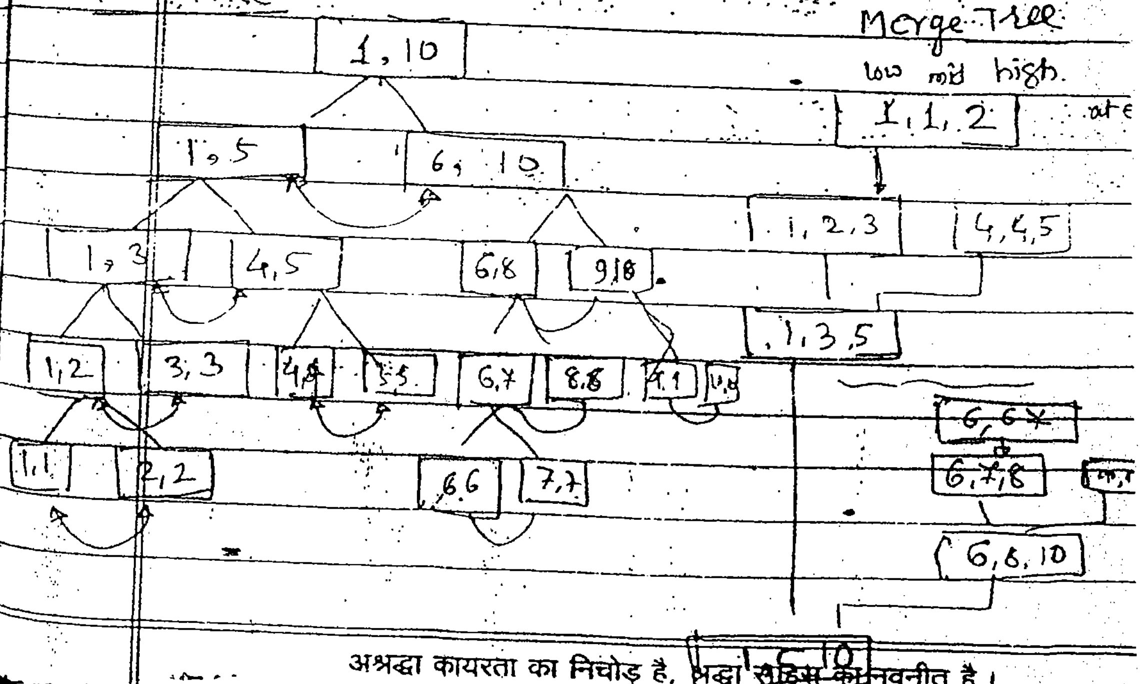
Ex (1) (2) (3) (4) (5) (6) (7) (8) (9) (1

A | 310 285 179 652 351 423 861 254 450 52

Recurrance:

Smallest element is one

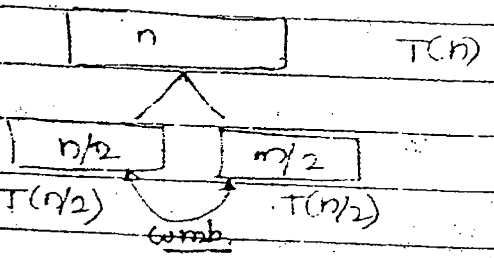
Divide Tree



* Let T(n) represents the Timing complexity of Divide AND Conquer merge sort (n).

$\Rightarrow$ $T(n) = C$ (Const.) $\quad n = 1$

$= \quad n > 1$



$T \propto n$ proportional to n

$T = bn$

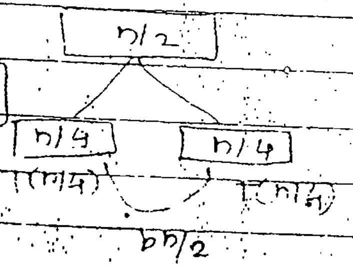
↑ Const

$\therefore T(n) = 2T(n/2) + bn \quad n > 1$

$T(n) = 2T(n/2) + bn$ —— ①

$T(n/2) =$

$[2T(n/4) + b \cdot n/2]$



putting this value in eqⁿ ①

$T(n) = 2\left[2T(n/4) + bn/2\right] + bn$

$= 4T(n/4) + \cancel{2}\,bn/\cancel{2} + bn$

$T(n) = 4T(n/4) + 2bn$ —— ②

plexity

Similarly

$T(n) = 8T(n/8) + 3bn$ ——— ③

after ample divisions n is enough small that $T(n\ \text{small}) = C$.

$\Longrightarrow T(n/4) = 2T(n/8) + bn/4$

$T(n) = 8T(n/8) + 3bn$ ——— ③

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

from eqn ① ② ③

$\boxed{n = 2^k}$ ——— ④ : $\boxed{k = \log_2 n}$

k level

$= 2^k T(n/2^k) + k\,bn$

enough small

$= 2^k T(1) + kbn$

$= n \cdot C + bn \log_2 n$

$\boxed{T(n) = nC + bn \cdot \log_2 n}$

$= O(n \log n)$

$= \Omega(n \log n)$

$\theta(n \log n)$

अश्रद्धा कायरता का निचोड है, श्रद्धा साहस का नवनीत है।

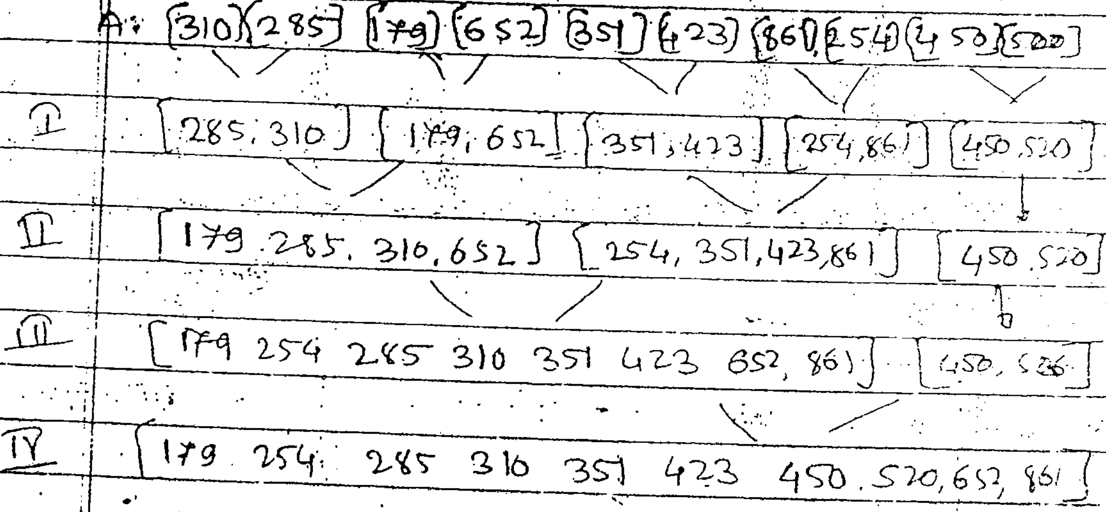
Space Complexity for DAND.C merge Sort

$n+n+\log n$

$\therefore O(n)$

Gate Q: 2-way merge sort

each element is considered as one list.



→ every pass : n Comparisions

$(1+\log n)$ → Max No. of Comparisions : $n \log n$

↑ [illegible] complexity

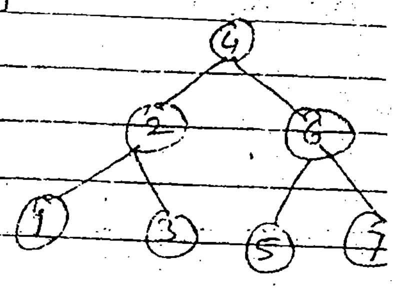
Timing complexity : $O(n \cdot \log n)$

$O(n \cdot \log n)$

3. DANDC : Binary Search.

eg $\overset{1}{[}1,2,3,\overset{a_K}{4},5,6,\overset{n}{7}]$

mid

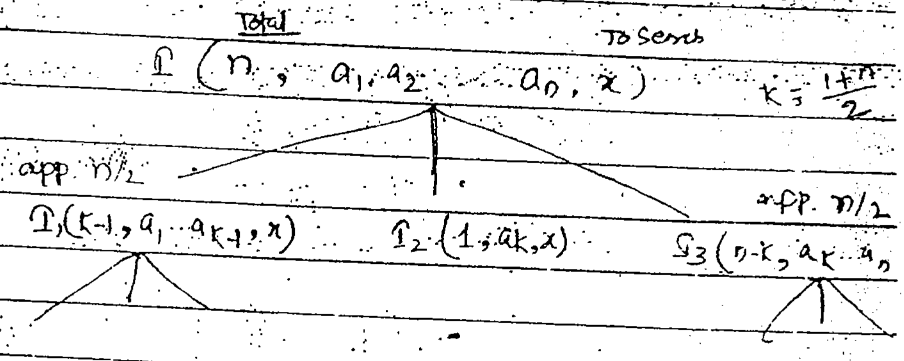


Binary search Tree

Complexity Unsuccessful :

successful { Best Case : $O(1)$
Worst Case : $O(\log n)$
Avg case $O(\log n)$

Recursive Binary Search:



Binary Search is Not True divide & Conquer strategy B'coz we only divide, Not combi

Gate Q let T(n) represent the Timing comp. of DAND BS(n)

$$T(1) = C \qquad n=1$$
$$T(n) = a + T(n/2) \qquad n>1$$

a → constant

b'coz we solve only one arm of Tree

T.

-3[500]

[50,520]

[50,520]

[50, 526]

[52, 861]

$n = 2^k$, $k = \log n$

$T(n) = T(n/2) + a$ ——— ①

$T(n/2) = T(n/4) + 2a$

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

$= T(n/2^k) + k.a$

$= T(1) + k \cdot a$

$\boxed{T(n) = c + a \cdot \log n}$

$O(\log n)$

partition ↳ (uses only [illegible]

C.J. Hoarse

4. DANDC: QuickSort

- partition - Exchange Sort.

partition / pivoting.

A 65,

N = 9 N

	1	2	3	4	5	6	7	8	9	10
A	65	70	75	80	85	60	55	50	45	∞
	i									p

i ← 1

p ← n+1,

V ← A[i] (65)

```
loop
    loop
        i = i+
    ~~until A(i]~~
    loop
        p = p-1
    until A(p
    if (i<p)
        swap(A[i
    else
        break
Infinite
```

Timing Complexity

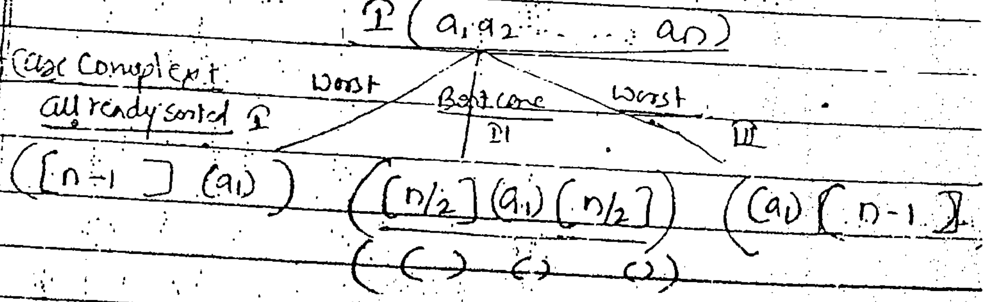
- Best case
- worst. case

Performance :-

1. Partitioning : O(n)

2. :

Worst Case Complexity



~~Best Case~~

2 element get pivoted at the end

at 2nd case 3 elements get pivoted

∴ works fast

∵ ~~more pivoted elements~~

2 elemen

gets piv

Timing Complexity in Best Case

$I(a_1\ a_2\ \ldots\ldots\ a_n)$

$[n/2]\ \boxed{a1}\ [n/2]$

1 Best case :-

$T(n) = a \qquad n=1$

$= bn + 2T(n/2) \qquad n>1$

$= O(n\log n)$ → Best case complexity

= equal to merge sort

2 Worst case :-

$T(n) = 1 \qquad n=1$

$= n + T(n-1) \qquad n>1$

This is Not DAND recurrance

b'coz standard DAND form is

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

$b>1$

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

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

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

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

$T(n)$

$= T[1]$

$= 1 + 2 + 3 + \ldots\ldots (n-1) + n$

$= n \cdot (n+1)/2 = \underline{\underline{O(n^2)}}$ → worst comple

Imp ★ No any comparision based sorting has complexity less than ___ with n elements

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

Sorting

Comparision Based
- quick
- merg

Non-comparision Based (mechanic sort)
- Radix
- heap

★ n+1 element is kept a large b'coz

If partition element is highest among 1 to n-1 then there is possibility of infinite loop at first inner loop $i = i+1$

5. Strasson's Matrix-multiplication

$A_{n\times n}$, $B_{n\times n}$ → $C_{n\times n}$

1) $A + B = C$

```
for 1 — n
    for 1 — n
        $C(i,j) = A(i,j) + B(i,j)$
```

$O(n^2)$

2) $A \times B = C$

```
for 1 — n
    for 1 — n
        $C(i,j) = 0$
        for 1 — n
            $C(i,j) = C(i,j) + A[i,j] * B[i,j]$
```

$O(n^3)$

Here $n = 4$

$n/2 \times n/2$

$$\underbrace{\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}}_{\substack{4\times4\\ n\times n}} \underbrace{\begin{bmatrix}B_{11} & B_{12}\\ B_{21} & B_{22}\end{bmatrix}}_{\substack{4\times4\\ n\times n}} = \overset{\frac{n}{2}\times\frac{n}{2}}{\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}} C$$

$T(n/2)$

$C_{11} = A_{11}\cdot B_{11} + A_{12}\cdot B_{21}$ —— 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

T(n) is DAND. matrix mul (n)

$T(n) = C$ $\quad n = 1$

$= 8T(n/2) + bn^2$

Time for mul $\quad$ Time for Addition

$A \cdot B + A \cdot B$

8 multip $\quad$ 4 Addition for each addition $O(n^2)$ comp

from eq ①

$n = 2^k$

$\Rightarrow k = \log_2 n$

$T(n) = 8T(n/2) + bn^2$ ——— ①

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

C | C_{11} | C_{22}

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

$= 64T(n/4) + 3bn^2$ ——— ②

$= 8^2 T(n/2^2) + (2^2 - 1) \cdot bn^2$

$= 8^k T(n/2^k) + (2^k - 1) \cdot bn^2$

$= 8^k \cdot T(1) + (2^k - 1) \cdot bn^2$

$= 8^{\log n} \cdot C$

$= n^{\log_2 8}$

$O(n^3)$

$n^3 C + (n-1) bn^2 = cn^3 + bn^3 - bn^2$

* Stasson reduced No. of multiplications from 8 to 7

$\therefore T(n) = 7T(n/2) + bn^2$ ——— ①

$T(n/2) = 7T(n/4) + \frac{bn^2}{4}$

$T(n) = 7\left[7T(n/4) + \frac{bn^2}{4}\right] + bn^2$

$T(n) = 49\, T(n/4) + \frac{7}{4} bn^2 + bn^2 (7/4)^0$

$= 7^2\, T(n/2^2) + bn^2 \sum_{i=0}^{1} (7/4)^i$

$= 7^k\, T(n/2^k) + bn^2 \sum_{i=0}^{k-1} (7/4)^i$

$T(n) = 7^k \cdot C + bn^2$

$\sum_{i=1}^{n} x^i \leq x^{n+1}$

if $x > 1$

$$T(n) < c \cdot 7^K + bn^2 \left(\frac{7}{4}\right)^K$$

$$< c \cdot 7^K + b\cancel{n^2} \cdot \frac{7^K}{\cancel{4^K}}$$

$$4^K = n^{\log_2}$$ =

$$T(n) < c \cdot 7^K + b \cdot 7^K$$

$$< (c+b) \cdot 7^K$$

$$< d \cdot 7^K$$

$$7^K = n^{\log_2 7} = n^{2.81}$$

$$\boxed{T(n) = O(n^{2.81})}$$

∴ Rec matrix multi vs Stassons Matrix mul

$$O(n^3) \Longleftrightarrow O(n^{2.81})$$

Q. $T(n) = k \quad , n = 1$

$= 3T(n/2) + K \cdot n \quad , n > 1$ } Solve recurrence relation

Master Method for solving Divide and Conquer Recurrances

$\Rightarrow T(n) = c$ if $n \leq d$

$= a \cdot T(n/b) + f(n)$ if $n > d$

$a > 0$, $b > 1$, $c > 0$, $f(n)$ is +ve,
$d > 0$

Case. I Let $f(n)$ and $T(n)$ is defined as above

if $f(n)$ is $O(n^{\log_b a - \epsilon})$
for some $\epsilon > 0$, then
$\boxed{\epsilon > 0}$ $T(n)$ is $\Theta(n^{\log_b a})$

Case II let $f(n)$ is
$\Theta(n^{\log_b a} \cdot \log^k n)$
Then
$T(n)$ is $\Theta(n^{\log_b a} \cdot \log^{k+1} n)$

Case III if $f(n)$ is
$\Omega(n^{\log_b a + \epsilon})$ and
$\epsilon > 0$

[illegible] $f(n/b) \leq \delta \cdot f(n)$ Then

$T(n)$ is $\Theta(f(n))$

$\boxed{\delta < 1}$

given Ex

1. $T(n) = 4T(n/2) + n$
2. $T(n) = 2T(n/2) + n\log n$
3. $T(n) = T(n/3) + n$

① $T(n) = 4T(n/2) + n$

above

$a = 4$
$b = 2$
$f(n) = n$

Check $n^{\log_b a} = n^{\log_2 4} = n^2$

i) is n is $O(n^{2-\epsilon})$ ✓ $\epsilon > 0$

$\epsilon = 1$

n is $O(n)$ ✓

Case 1 applies and

$T(n)$ is $O(n^2)$

② $T(n) = 2T(n/2) + n\log n$

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

$n^{\log_2 2} = n$

i) $n \log n$ is it $O(n^{1-\epsilon})$ ✗
ii) $n \log n$ is $\theta(n^p . \log^k n)$ ✓

$k > 1$

$\therefore T(n)$ is $\theta(n . \log^2 n)$ ✓

3.

4) $T(n) = 9T(n/3) + n^{2.5}$

$a = 9$
$b = 3$
$f(n) = n^{2.5}$

$n^{\log_a^b}$, n^2

Case 1

$n^{2.5}$ is it $O(n^{2-\epsilon})$ ✗

$\epsilon > L$

Case II > $\frac{n^{2.5}}{n^2\sqrt{n}}$ is It $\theta(n^2 \cdot \log^k n)$

$\underline{\sqrt{n} \neq \log^k n}$

Case $\underline{III}$

$n^{2.5}$ is a $\Omega(n^{2+\epsilon})$ ✓ $\underline{\epsilon =}$

a) $a \cdot f(n/b) \leq \delta f(n)$

$\frac{\cancel{9} \cdot n^{2.5}}{\cancel{9}\sqrt{3}} \leq \delta n^{2.5}$ ✓

$\delta = \frac{1}{\sqrt{3}} < 1$

$T(n)$ is $\Theta(n^{2.5})$

⑤ $T(n) = 2T(\sqrt{n}) + \log n$

$b = 1$

> 1

let $n = 2^k$

$T(2^k) = 2T(2^{k/2}) + k$

let $T(2^k) = S(k)$

$T(2^{k/2}) = S(k/2)$

∴ eqⁿ becomes

$S(K) = 2 \cdot S(K/2) + K$

[Case 2 applied &

it is $O(K \cdot \log K)$

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

अश्रद्धा कायरता का निचोड़ है, श्रद्धा साहस का नवनीत है।

Greedy Methods

Step by step

1 prob. Def
2 Constraints (Conditions)
3 Solution space (Any possible arrangment)
4 feasible sol^n (Correct solution)
5 objective function → max/ min criteria
6 optimal Solution : → only one (unique)

n-queens problem :
- No objective function
- so we find only feasible solutions

There is no meaning for optimal solution

⑫ General Approach of Greedy method :

```
Prodedure GreedyM ( A, n)
{
    Solution ← ∅
    for i ← 1 to n
    {
        x ← select (A, n)
        if( Feasible (solution, x))
            Add (x, solution);
    }
}
```

max. complexity of Greedy method is

$O(n)$

if ① select(A, n)
② feasible(sol, x)
③ Add(x, sol)
} are of order ($O(1)$

ue)

Constants

if ① $O(n)$
② $O(n)$
③ $O(1)$
} $O(n^2)$

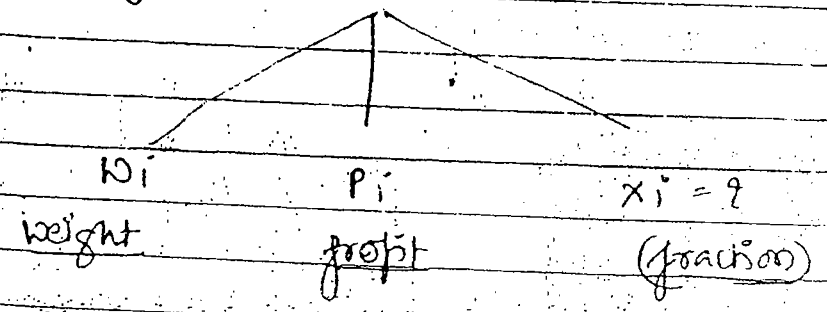
B'ons
solution

if 1) $O(1)$
2) $O(1)$
3) $O(\log n)$
} $O(n \cdot \log n)$.

② Knapsack problem

- Real knapsack
= Fractional knapsack
- Continneous knapsack

Knapsack - capacity M
n objects (Oi)



maximise the profit subject to condition That the total weights in the Knapsack do not exceed its Capacity.

ef $\sum_{i=1}^{n} w_i \le m$

problem. Need not to be solved

Total weight $\sum_{i=1}^{n} w_i x_i \le M$ ✓

∴ Constrain given $\sum_{i=1}^{n} w_i > M$

⟹ maximise $\sum_{i=1}^{n} p_i x_i$

s.t.c $\sum_{i=1}^{n} w_i x_i \leq m$

x_i can be any value
$0 \leq x_i \leq 1$

ition

∴ Solution space for Real knapsack — infinite

∴ Solution space for 0/1 knapsack — 2^n.

ded.

Ex $n=3$, $(P_1\ P_2\ P_3) = (25, 24, 15)$ $m = 0$

$(w_1\ w_2\ w_3) = (18, 15, 10)$

① Greedy method profit consideration.

$x_1 = 1$

$x_2 = 2/15$ $\sum p_i x_i = 28.2$

$x_3 = 0$

2. Greedy method about weight

$x_3 = 1$
$x_2 = 10/15 = 2/3$
$x_1 = 0$

$\Sigma p_i x_i = 31$

3. Greedy method about P/W

divide all objects into equal criteria.

$w_i \longrightarrow p_i$
$1 \longrightarrow ?$

$$\frac{p_i}{w_i}$$

$\frac{p_1}{w_1} = \frac{25}{18} = 1.4$

$\frac{p_2}{w_2} = \frac{24}{15} = 1.6$ ✓

$\frac{p_3}{w_3} = \frac{15}{10} = 1.5$

arrange into decreasing order of p_i/w_i

Then $\frac{P_2}{w_2}$, $\frac{P_3}{w_3}$, $\frac{P_1}{w_1}$

✓

$x_i = 1 \quad 5/10 = 1/2 \quad 0$

$\therefore \sum p_i x_i = 24 + \frac{1}{2}(15)$

$= 24 + 7.5$

max. profit is = 31.5

∴ it is optimal solution.

Job-sequencing with deadline (JSD)

- single cpu with N.p scheduling (Non preemptive)

n jobs (J_i)

A.T = 0, B.T(1) deadline (d_i) profit P_i

"Select a subset of n jobs, such that jobs in the subset can be complited within deadline, maximising the profit"

Size of solution space 2^n

we have to select subset of Job from n Jobs

possible subsets $= 2^n$

Ex. $n = 4$ Jobs

J_1 to J_4

$(d_1 - d_4)$: $(2, 1, 2, 1)$

$(P_1 - P_4)$: $(100, 10, 15, 27)$

Solution : J

No. of Jobs in subset

$J = \emptyset$ ✓ possible

$J = 1$ ✓

$J = 2$ ✓

$J = 3$ ✗

No. of Jobs possible = (Max. deadline in The given set)

⇒ arrange Jobs in decreasing order of profit J_1 J_4 J_3 J_2

(100, 27, 15, 10)

J4	J1

* place the job in its maximum deadline boundry.

* No. of jobs possibly Completed within The deadline = { max No. (among deadline give

Ex2 n = 7

$d_1 - d_7 = \{ \overset{1}{6}, \overset{2}{5}, \overset{3}{4}, \overset{4}{3}, \overset{5}{4}, \overset{6}{5}, \overset{7}{6} \}$

$p_1 - p_7 = \{ 18, 16, 10, 9, 19, 23, 15 \}$

J_6 = 23 – 5 ✓
J_5 – 19 – 4 ✓
J_1 – 18 – 6 ✓
J_2 – 16 – 5 ✓
J_7 – 15 – 6 ✓
J_3 – 10 – 4 ✓
J_4 – 9 – 3 ×

Total profit =

1	2	3	4	5	6
J3	J7	J2	J5	J6	J1

0 1 2 3 4 5 6

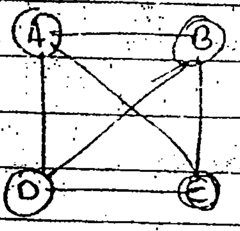
* place The Job in its highest possible deadline

*3. Minimum Cost Spanning Tree.

Graph = G = (V, E)

|V| = n n vertices

|E| = e e edges



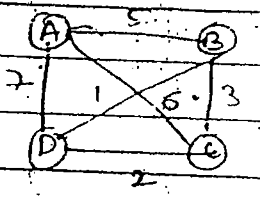
|V| = n = 4

|E| = e 6

Spanning Tree:

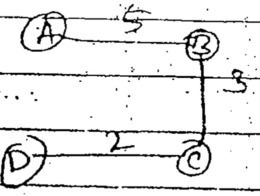
"A Subgraph T(V, E') of Graph G(V, E), where E' ⊂ E is a spanning Tree if and only if T is a Tree"

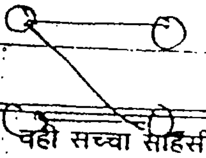
⇒ Cost of Spanning Tree:

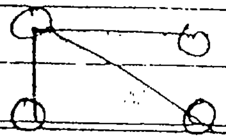


Given Graph

Spanning Trees.







Gate Q

~~minimum~~ how meny Edges must b Removed from Graph to from a Spanning Tree

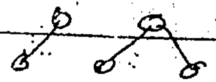
∴ n – no. of vertices
e – no. of edges

for n vertices to be connected (n−1) edges are compulsory

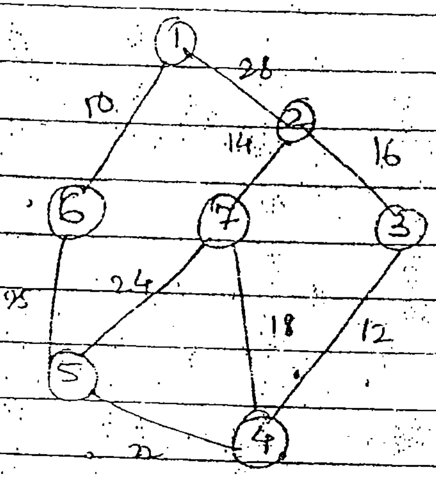
∴ $e - (n-1)$ can be Removed.

aph

Prims Algo.	Krushals algo.
Always maintain connected properties (Tree structure)	may construct a sub Tree from set of disting Tree (Forest)



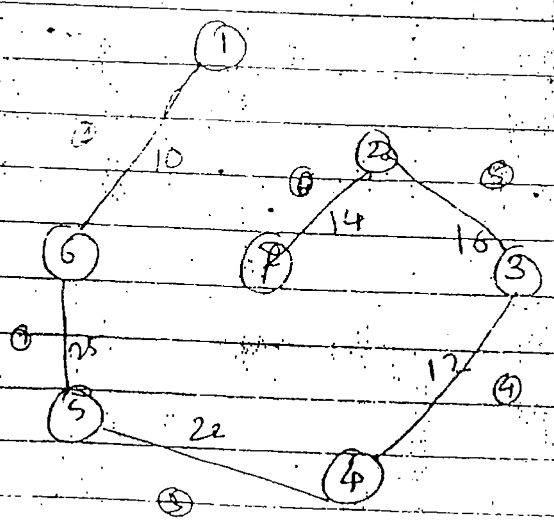
Min. Cost spanning Trees



prims method: $O(n^2)$

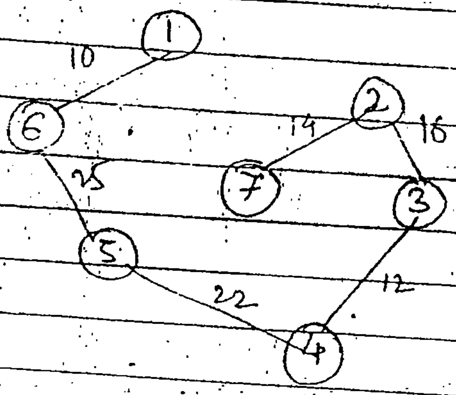
1. Start from min most value
2. Connect the nodes which are already part of connection only
3. No cycle

Always forms Tree Connected



Krushkals $O(e\log e)$

min heap : 'e'
all edges ~~10~~, ~~12~~, ~~14~~, ~~16~~, $\underset{\alpha}{18}$, ~~22~~, $\underset{X}{24}$, ~~25~~, 28



forms: disjoint Trees called forest

optimal merge pattern

Merging of files

2 way merging

Imp No. of Record movements :

f1 (4, 8, 12 ...) , f2 (5, 9, 1 ...)

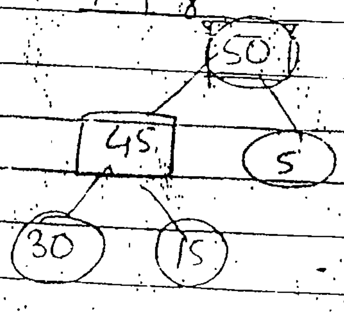
f (4, 5, 8, 9 ......)

f_1 = m elements
f_2 = n element

To get f Total Record movement = m+n

eg x = 30
y = 15
3 files z = 5

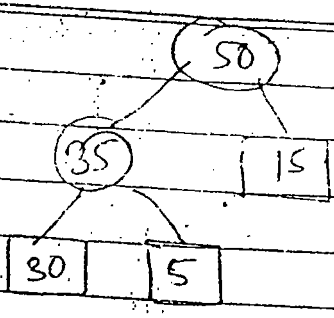
pattern: x y z



sum of internal :
45
50
95

∴ Total Rec mov = 95

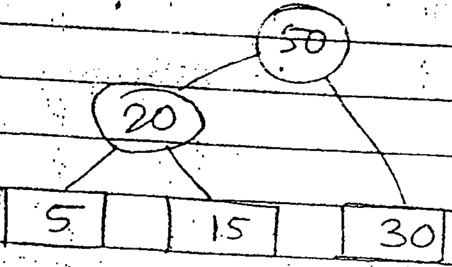
(x, z, y)



Tot. Rc. r
85

. . .)

③ (Z, Y, X)



$\boxed{70}$

= mtn

④ solution space :

If n files are there n! sol. spac

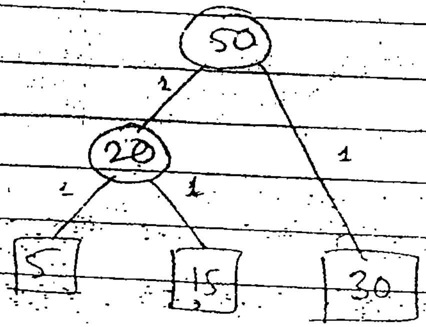
```
Algorithm Tree (n)
{
    for i <- 1 to n-1
    {
        Pt = new(terminal);
        Pt -> lChild = LEAST (list);
        Pt -> RChild - LEAST (list);
        Pt -> wt = Pt -> lChild -> wt
                   Pt -> RChild -> wt
        INSERT (list, Pt);
    }
}
```

d_i = dist from root to f_i

q_i ~~f_i~~ = size of f_i

Weighted Extended path length (B.T)

$$= \sum_{i=1}^{n} d_i q_i$$



$= 1\times30 + 2\times5 + 2\times15$

$= 30 + 10 + 30$

$= \underline{70}$

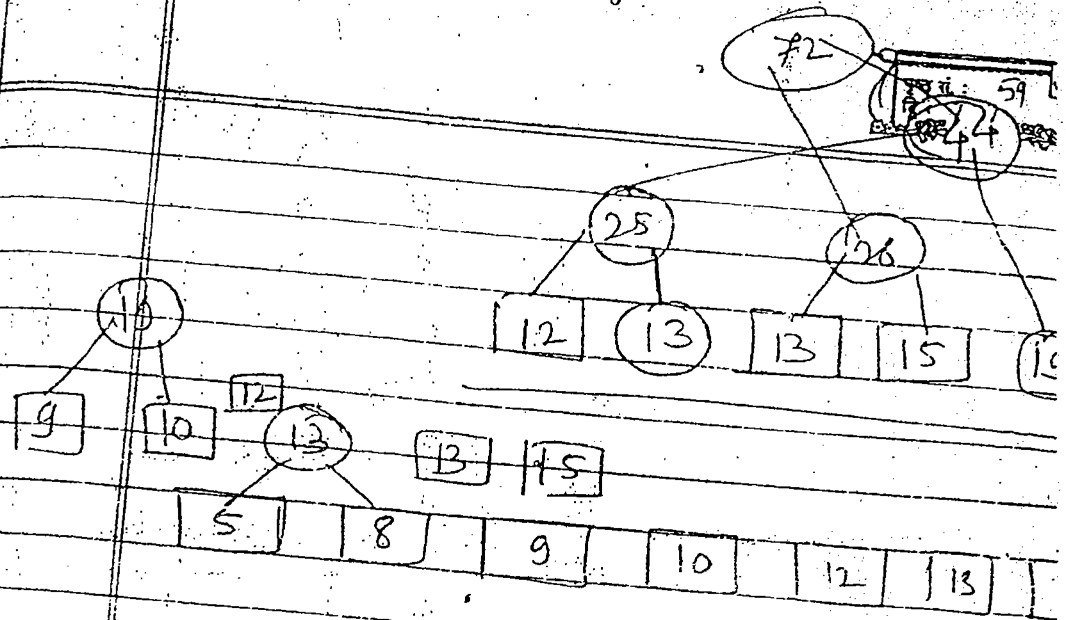
Q2 GateQ

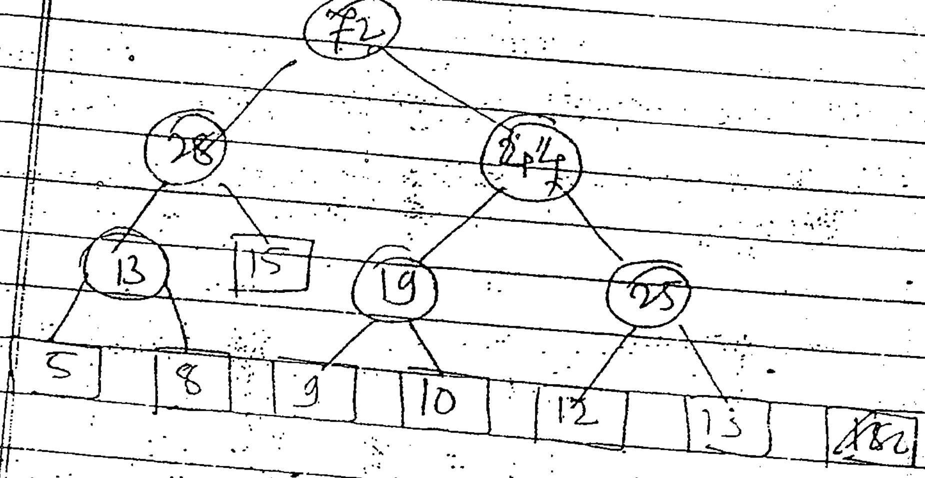
$n=7$

$(f_1 - f_7) = (5, 8, 15, 9, 13, 12, 10)$

① Arrange files in increasing order.

② After 1 merging. Again arrange.





Ans 201

or every time Take 2 smallests

Frequency dependent Coding

Huffman Coding

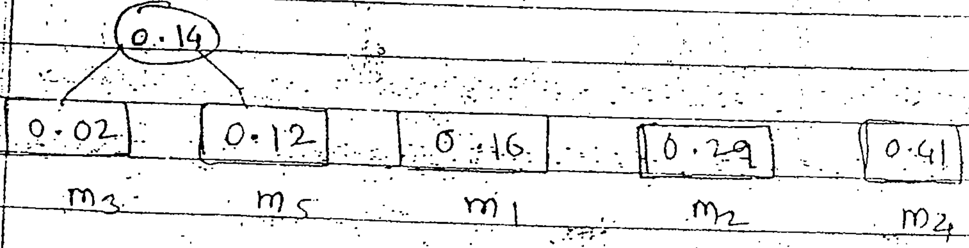
(Data Encoding + Compression)

$L = \lambda(m_1 - m_5)$ $(\underline{0.16, 0.29, 0.02, 0.41, 0.12})$

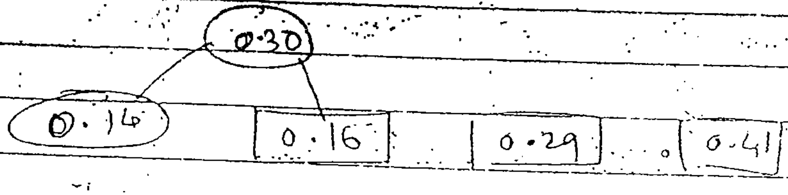
~~Freq. percent~~

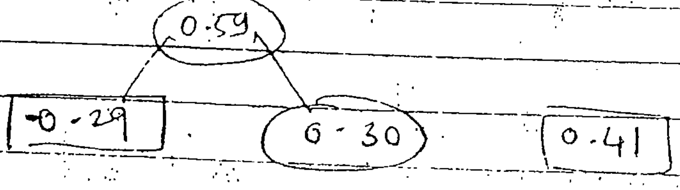
Normally for 5 msg we need 3 bits as

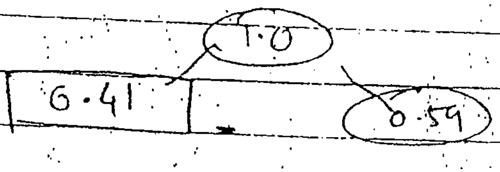
$\underline{5 < 2^3}$.



Rearrange

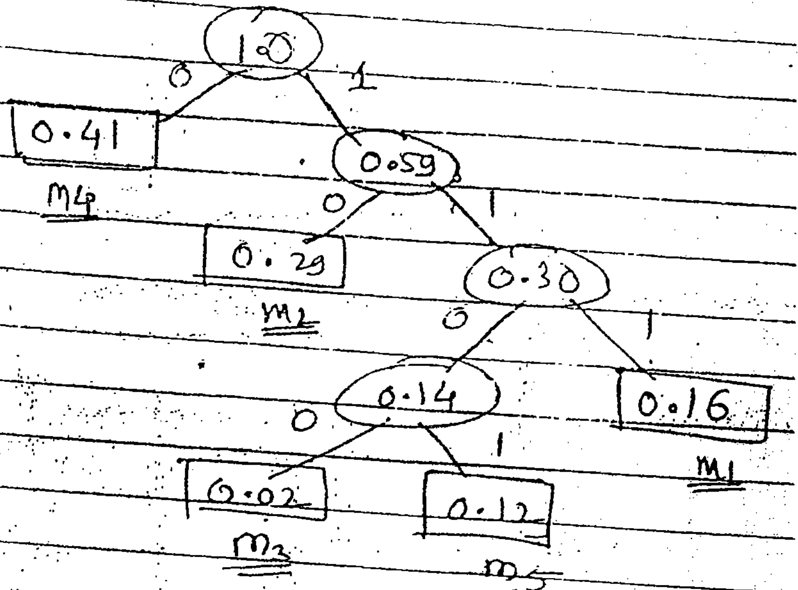






, 0.12)

Forming Tree   left 0 , Right 1



41

m4

m1 = 111 (3)
m2 = 10 (2)
m3 = 1100 (4)
m4 = 0 (1)
m5 = 1101 (4)

ex: Text $m_4 m_2 m_4 m_4 m_2 m_1 m_4 m_2 m_5 m_4$. (30 bi
0 1000 10 111 0 10 1101 0 (18 bi

Reverse

110101
search in Tree till we get
leaf Node

Avg. bits : $\sum_{i=1}^{n} d_i f_i$

d_i = dist from root
f_i = freq.

$= 3 \times 0.16 + 2 \times 0.29 + 4 \times 0.02$
$+ 1 \times 0.41 + 4 \times 0.12$

$= 0.48 + 0.58 + 0.08 + 0.41 + 0.48$

$= 2.03$

$= 2.03/5$

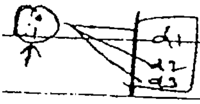
$= 0.406$

0.48
0.58
0.08
0.41
0.48
2.03

Drawback:

1 bit error during transmissio causes entire corruption of the d

Shortest paths [S.P] Problems

① single pair S.P [ⓘ ⓙ]
② single source S.P


~~② single~~
③ single destination S.P

] (Dijkstra's Alg
(+ve edge)
~~of~~ -ve edge,
Bellman

④ multi pairs shortest paths → Floyd's Warshall Alg

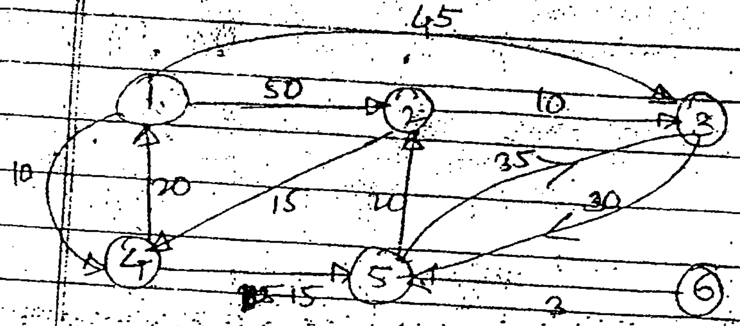
~~Shortest paths Problems~~

③ Single Source Shortest path

G(V, E)

|V| = n
|E| = e

$V_0 = 1$ (given)

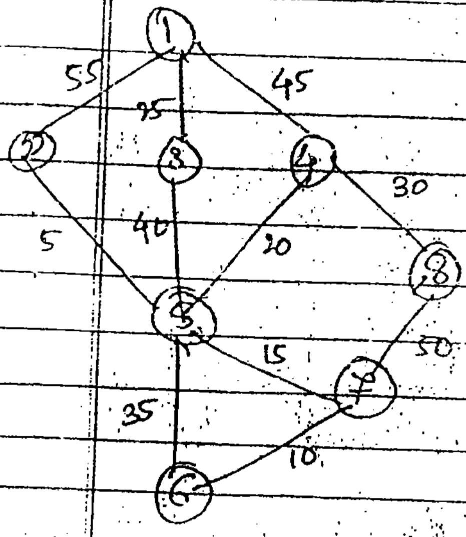


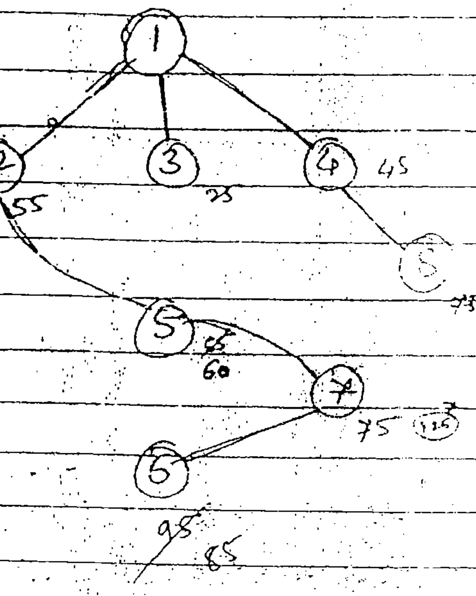
Vertex select	2	3	4	5	6
1	50	45	⑩	∞	∞
1-4-	50	45	⑩	㉕	∞
1-4-5	㊺	45	⑩	20	∞
1-4-5-2	㊺	㊺	⑩	㉕	∞
1-4-5-2-3	㊺	㊺	⑩	㉕	[illegible]

1-4-2-4

Single source shortest path spanning tree

$V_0 = 1$





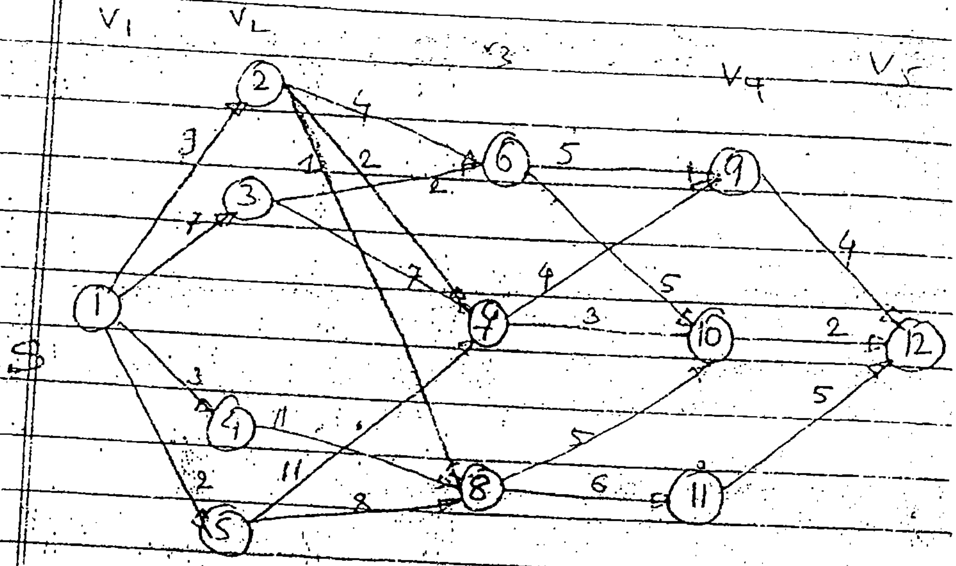
① select nodes which are directly connected from source i.e ①
& mark their weight.

② select which have min value among them.

③ Join the selected min node. According to its min value.

④ update the values of other weights According to this new node
i.e. those are <
select min weight node.

Multistage Graph



|V| = n
|E| = e

G(V,E)

n- vertic
e- edges

Greedy method sol^n. (1-5-8-10-12)
cost : 17
fails.

Greedy method : depending upon current stage and current values decide solution.

⇒ First and last stage have only 1-1 vertex
other n-2 vertexes are there
and n-2 vertexes have K-2 stages

* i/p cost matrix is given

$$\begin{array}{c|c} C & i \\ \hline j & - \; - \\ & - \; c_{ij} \end{array}$$ edge value

Greedy method : stepwise decision.
Dynamic prog : sequence of decision.
decisions on series.

" Dynamic programming is followed by a principle of optimality "
Imp

⊛ principle of optimality state that :
What ever initial state and decision are, The remaining sequence of decisions must constitute an optimal decision sequence with regards (with respect) to the state resulting problem.

Imp
(✱) ⟶ Let cost(i, j) represents the cost of reaching the destination 't' from vertex j in the stage i

(C_{ij} is edge value)

$$\therefore Cost(1,1) = \min \left\{ C(\underset{\text{First node}}{1}, \underset{\text{Any node}}{K}) + Cost(\underset{\text{Stage}}{2}, K) \right.$$

th

n

$K \in V$

$\langle 1, K \rangle \in E$

a

$C(1,K) + cost(2,K)$

$C(1,K).\ (K \ldots\ldots t)$

⊙ Recurrance :

$$Cost(i,j) = \min \left[C(j,K) + cost(i+1, K) \right] \quad ①$$

Then if there are l stages in a Graph

$$Cost(l-1, j) = Cost(j, t)$$

* let $D(i,j) = K$ that minimise eqⁿ ①.

$\Rightarrow$ From Graph directly we can write

$Cost(4,9) = C(9,12) = 4$
$Cost(4,10) = C(10,12) = 2$
$Cost(4,11) = C(11,12) = 5.$

$$Cost(1,1) = \min\left\{ C(1,2) + Cost(2,2),\ C(1,3) + Cost(2,3),\ C(1,4) + Cost(2,4) \right\}$$

$\rightarrow$ Approaching in Reverse Order

Calculate $Cost(3, \_\_)$

$$Cost(3,6) = \min\left\{ C(6,9) + Cost(4,9),\ C(6,10) + Cost(4,10) \right\}$$

$$= \min\{(6+4), (5+2)\}$$

$\therefore Cost(3,6) = 7.$

$\therefore d(3,6) = \boxed{10}$

7

Which node gave min value

$Cost(3,7) = 5$
$d(3,7) = 10$

$Cost(3,8) = 7$
$d(3,8) = 10$

ost(2,3)

cost(2,2) = 7

D(2,2) = 7.

cost(2,3) = 9

D(2,3) = 6

cost(2,4) = 18

D(2,4) = 8.

cost(2,5) = 15

D(2,5) = 8.

The value of D is used to calculate the path.

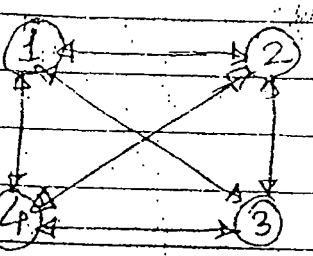
$$1 - D\left(3, D\left(2, \underbrace{D(1,1)}_{2}\right)\right) - 12$$

$$D\left(3, \underline{D(2,2)}\right)$$

$$\underbrace{D(3,7)}_{10}$$

path is 1-2-7-10-12.

2. Traveling saleperson's problem (TSP)



Greedy method

1-2-3-4-1

= 39

But

1-2-4-3-1

= 35

'j'

'i'

C	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

let $g(i, S)$ represent the cost of the tour. From vertex 'i' visiting all vertices in the set 's', exactly once and terminating at V_0.

$$g(1, V-\{1\}) = \min_{\substack{k \in V-\{1\} \\ \langle 1,k \rangle \in E}} \left(C(1,k) + g(k, V-\{1,k\}) \right)$$

$$\underline{1-K} \xrightarrow{V-\{1,k\}} \underline{1}$$

general Formula

$$g(i,S) = \min_{\substack{k \in S \\ \langle i,k \rangle \in E}} \{ c(i,k) + g(k, S-[k]$$

$$g(i, \phi) = c(i, V_0) \quad \text{———} \quad ②$$

$$\langle i, V_0 \rangle \in E$$
$$i \neq V_0$$

$J(i,S)$ = 'K' That minimizes eq^n ①

$\Rightarrow$

$$g(1,[2,3,4]) = \min \left(\begin{array}{l} c(1,2) + g(2,[3,4] \\ c(1,3) + g(3,[2,4 \\ c(1,4) + g(4,(2,3] \end{array} \right.$$

$\underline{|S|=3}$ $\qquad$ $\underline{|S|=}$

let us consider $|S| = \phi$

$g(2,\phi)$ = cost$(2 \to 1)$ = 5
$g(3,\phi)$ = $3 \to 1$ = 6
$g(4,\phi)$ = $4 \to 1$ = 8

$|S| = 1$

$g(2,[3]) = C_{23} + g(3,\phi) = 9+6 = 15$
$g(2,[4]) = C_{24} + g(4,\phi) = 10+8 = 18$

$g(3,[2]) = C_{32} + g(2,\phi) = 13+5 = 18$
$g(3,[4]) = C_{34} + g(4,\phi) = \;\; = 20$

$g(4,[2]) = C_{42} + g(2,\phi) = \;\; = 13$
$g(4,[3]) = C_{43} + g(3,\phi) = \;\; = 15$

$|S| = 2$

$g(2,[3,4]) = \min[\overset{9\;\;+20}{C_{23} + g(3,[4])}, \overset{10+15}{C_{24} + g(4,[3])}]$
$g(2,[3,4]) = 4$

$g(3,[2,4])$
$J(3,[2,4])$

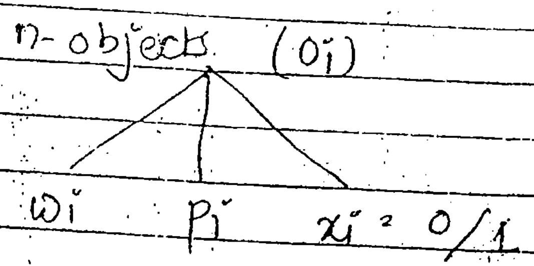
$g(4,[2,3])$
$J(4,[2,3])$

path:

1
$J(1,[2,3,4]) = 2$
$J(2,[3,4]) = 4$
$J(4,[3] = 3$
1

3. 0/1 Knapsack problem.

- Knapsack 'm'



15
18
= 18
= 20
= 13
= 15

let $F_n(m)$ represents the profit obtain with n objects on the knapsack with capacity m

$$F_n(m) = \max\left\{ \overset{x_n = 1 \text{ if included}}{P_n + F_{n-1}(m - w_n)},\ \underset{\substack{x_n = 0 \\ \text{if Not included}}}{0 + F_{n-1}(m)} \right\}$$

Start from empty knapsack.

$$\therefore \quad F_0(X) = \phi$$

Prob $n = 3$ $(p_1\ p_2\ p_3) = (1, 2, 5)$

$(w_1\ w_2\ w_3) = (2, 3, 4)$

$m = 6$

$F_n \Rightarrow S^n \Rightarrow S^{n-1} \Rightarrow S^{n-2} \Rightarrow S^0$

$S^3 \Rightarrow S^2 \Rightarrow S^1 \Rightarrow S^0$

$S^0 = [(0, 0)]$ initial value of knapsack

profit weight

after this much attempts

$S^0 [(0,0)] \xrightarrow{\text{merge}} S^1 [(0,0), (1,2)]$

$S^1_1 = [(1,2)]$

consider first (p_1, w_1)

after adding $p_1 w_1$ to S

$S^1 [(0,0), (1,2)] \longrightarrow S^2 [(0,0)\ (1,2)\ (4,3)\ (3,5)]$

$S^2_1 [(2,3)\ (3,5)]$

$S^2 [(0,0)\ (1,2)\ (2,3)\ (3,5)] \longrightarrow$?

$S^3_1 [(5,4), (6,6), X, X]$ order is important

after adding $p_3 w_3$ to S^2

weight exceeded

(*) purging Rule: Removal Rule.

a) (3,5)   (5,4)
$P_i \; w_i$ $P_j \; w_j$

if $(P_i < P_j)$ && $(w_i > w_j)$ then the touple $P_i w_i$ may be purged.

ok

∴ S^2 [(0,0) (1,2) (2,3) (3,5)] → S^3 [(0,0), (1,2), (2,3), (5,4) (6,6)]

S_1^2 = [(5,4), (6,6)] ↗

2)

∴ max w (6,6)

∴ P = 6 x_1 = ?   0 or 1
w = 6 x_2 = ?   Include or Not.
x_3 = ?

To find the value of x_1 = 1 or 0

(6,6) ∈ S^3 ∴ $x_3 = 1$
$-P_3 w_3$ ∉ S^3
(1,2) ∈ S^2
∈ S^1 ∴ $x_2 = 0$
(1,2) ∈ S^1
∉ S^0 ∴ $x_1 = 1$

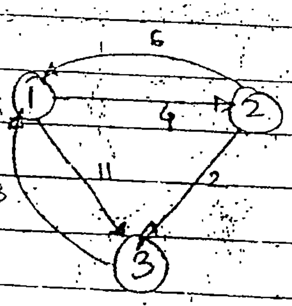
Rule :

if $(P, W) \in S^{i}$, $\notin S^{i-1}$ $\therefore x_i = 1$

if $(P, W) \in S^{i}$, $\in S^{i-1}$ $\therefore x_i = 0$

dynamic programming

All pairs Shortest path :-



C	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

This prob can be solved by GM as well as dynamic programming.

⇒ let $A^{k}(i,j)$ represent the cost of the path between vertices i & j not going through intermediate vertices of Indexes greater than 'K'

1 ... n   nodes are there

$(i \cdot k{-}1 \; \boxed{k} \cdots k{-}1 \;\; j)$

if the path passes through k then

$$A^{k}(i,j) = \min\{A^{k-1}(i,k) + A^{k-1}(k,j),$$

$$A^{k-1}(i,j)\}$$

↑ if path does not pass throu K

$$A^{0}(i,j) = c(i,j)$$

↳ This is directly an edge

eg $A^{3}(i,j) \Rightarrow A^{2}(i,j) \Rightarrow A^{1}(i,j) \Rightarrow A^{0}(i,j)$

calc. in terms of

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

path

$$\begin{array}{c|ccc} A^1 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

i = 1, j = 2, k = 1

$A^1(1,2) = \{ A^0(1,1) + A^0(1,2) \;,\; A^0(1,2) \}$

$0 + 4 \;,\; 4$

$= 4$

$A^1(2,3) \quad \min\{ 2-1-3 \;,\; 2-3 \}$

$\underline{6+11} \;,\; 2$

$\textcircled{2}$

$A^1(3-2) = \{ 3-1-2 \;,\; 3,2 \}$

$\textcircled{7} \;,\; \infty$

$$\begin{array}{c|ccc} A^2 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

$A^2(1,3) = \{ 1-2-3 \;,\; 1-3 \}$

$\textcircled{6} \;,\; 11$

2)}

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

$A^3(2-1) = \min\{2\text{-}3\text{-}1,\ 2\text{-}1\}$

$\phantom{A^3(2-1) = \min\{} \boxed{5}\ ,\ 6$

}

Gate ①

```
1  for i ← 1 to n
       for j ← 1 to n
           $A(i,j) = C(i,j)$
```

$A^0(i,j) = C(i,$

```
2  for k ← 1 to n
       for i ← 1 to n
       for j ← 1 to n
           $A(i,j) = \min\{A(i,k) + A(k,j),\ A(i,j)\}$
```

~~this~~ is not Recursion

$A(i,k)$ is direct value.

$A(i,j) = \{\_\_\_\_\}$

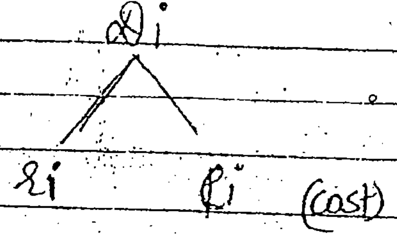
Time Complexity $O(n^3)$

Space " O(Two matrices)

$O(n^2 + n^2) = O(n^2)$

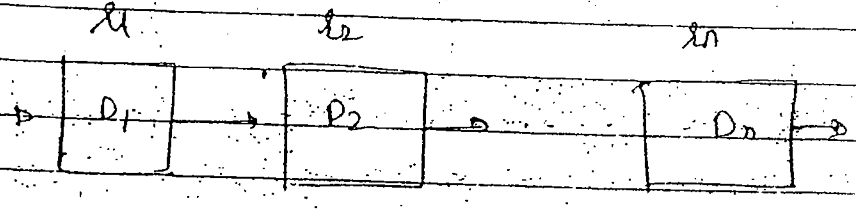
5. Reliability Design

(multiplicative optimization problem)



$0 \le r_i \le 1$

r_i: probability of reliability of device

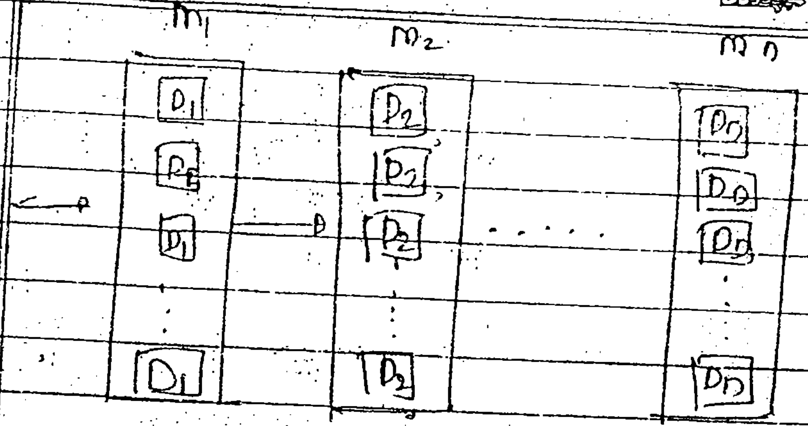


Reliability of n stage system

$$= \prod_{i=1}^{n} r_i$$

if 1 device fails then comp. system fails = multiplication of all r_i

To increase relibility we duplicate the devices.



mi = No of devices in stage i

$1 \le m_i \le u_i$

upper bound

let $\phi_i(m_i)$ denotes the reliability of stage (i) having m_i copies of devices D_i

$$\phi_i(m_i) = 1 - (1-r_i)^{m_i}$$

$(1-r_i)$: failure of 1 device

$(1-r_i)^{m_i}$: failure of m_i devices : No. of

$1-(1-r_i)^{m_i}$: reliability of m_i No. of devices

∴ Reliability of n stage system

= product of all stages

$$= \prod_{i=1}^{n} \phi_{m_i}(i)$$

Subjected to condition $\sum_{i=1}^{n} c_i m_i \le C$ given cost

Imp ⟹ n-stage system $(D_1 \ldots D_n)$

Reliability of $(D_i) \rightarrow r_i$

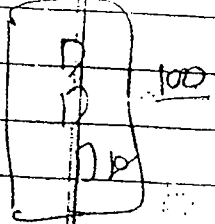
$\text{cost}(D_i) = c_i$

Total cost $= C$

Stage1 Stage2 Stage3

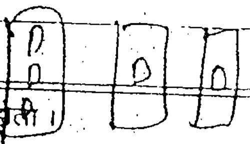
$n = 3$ (c_1, c_2, c_3)

(r_1, r_2, r_3)

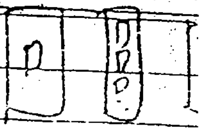


$$u_1 = \frac{C - (c_2 + c_3)}{c_1}$$

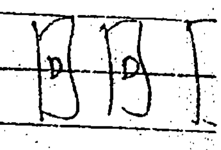
upper bound in stage 1



$$u_2 = \left[\frac{C-(C_1+C_3)}{C_2}\right]$$



$$u_3 = \left[\frac{C-(C_1+C_2)}{C_3}\right]$$



$$u_i = \left[\frac{C-\sum_{j=1}^{n} C_j + C_i}{C_i}\right]$$

$1 \le m_i \le u_i$

⟹ let $F_n(C)$ represent the Reliability of n-stage system with a total cost of 'C'

$$F_n(C) = \max\left\{\phi_n(m_n)\cdot F_{n-1}(C - C_n \cdot m_n)\right\}$$

$$1 \le m_n \le u_n$$

$F_0(x) = 1$

$F_3 = F_2 \cdot F_1 \cdot \boxed{\overset{1}{F_0}}$

Ep $\underline{n=3}$

$(r_1 - r_3)$ $\{0.9, 0.8, 0.5\}$

$(C_1 - C_3)$ $\{30, 15, 20\}$

$\underline{C = 105}$

$u_1 = \left[\frac{105-35}{30}\right] = 2$

$u_2 = \left[\frac{105-50}{15}\right] = 3$

$u_3 = \left[\frac{105-45}{20}\right] = 3$

Let $S = \{(R, C)\}$ be touple

$\underline{f \Rightarrow S^{n}}$

$S^n \Rightarrow S^{n-1} \Rightarrow S^{n-2} \ldots \Rightarrow S^0$

$S^0 = \{\underbrace{(1, 0)}\}$ initially

reliability cost

Stage 1 $S_1^1 = \{(0.9, 30)$

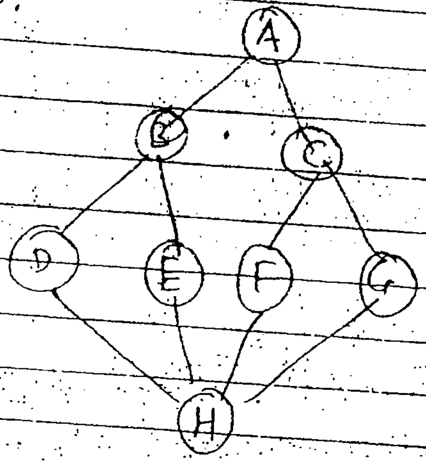
No of device

Stage $S_2^1 \{(0.99, 60)\}$

Graph Techniques

Traversal

BFS
DFS



Status of a Node

expanded

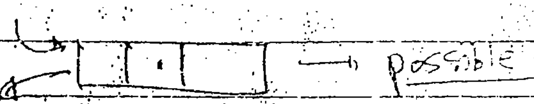
Live Node	E-node	Dead Node
>1	1	> 0
Node that yet explored	currently being explored & Expanding	Not to be explored of All childs are explored

Assume:

Tree have unique traversal
1- preorder only
But Graph have multiple traversal
DFS

BFS Queue (
DFS Stack (

Queue : By default FIFO
But it can work as FILO

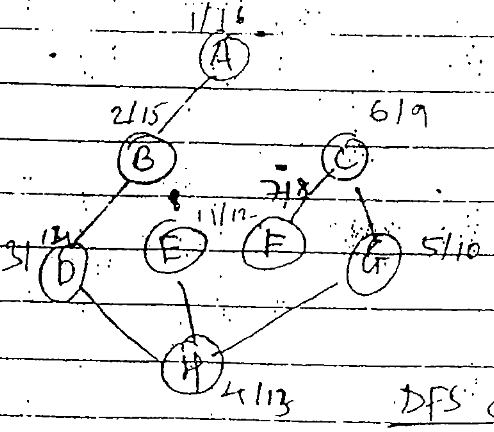


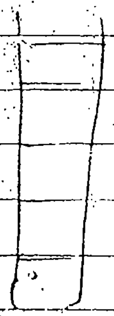
Double ended queue
'DQ'

1) D.F.S
traversal can start with Any node

Uses Stack :

A, B, D, H, G, C, F, E





DFS Spanning Tree

The above diagram is DFS Spanning Tree

(a) Back Tracking
(b) connected components of graph (undirected)
(c) strongly connected component

Gate Which of following are valid & Which are Not valid DFS Traversal

a) A, B, D, H, E, F, C, G ✓
b) A, B, E, H, D, F, C, G ✓
c) A, B, D, E, F, G, C, H ✗
d) H, D, B, A, C, G, F, E ✓
e) H, E, B, D, C, A, F, G ✗

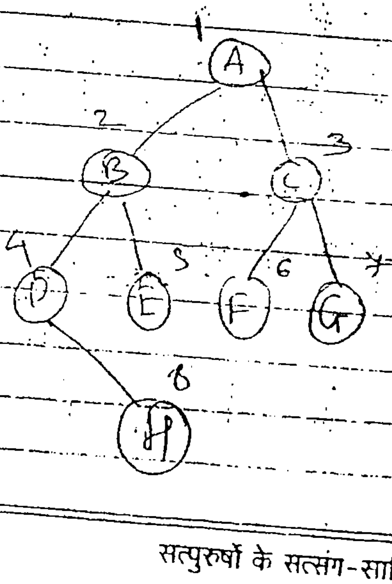
Application of DFS is Backtrackin
ARC points, cycles,

2) B.F.S :

it is call D-search

uses Queue — ef queof LIFO : used

unvisited childs : put in Q



A B C D E F G

→

B	C	D	E	F	G	H

FIFO queue

Application: Branch n Bound

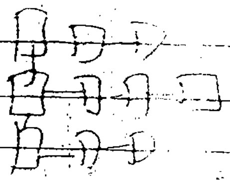
BFS Shortest path b/w 'i' & 'j'
No of hopes

Time & space Complexities of BFS | DFS

if Graph represented as

i) Adj matrix = $O(n^2)$ is stored

ii) Adj list repr = $O(n+e)$

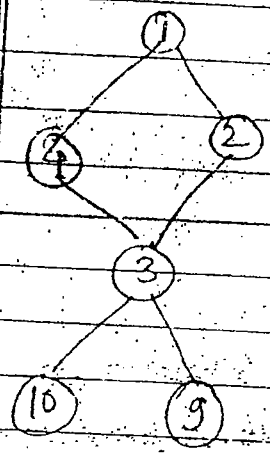


Articulation point (Cut-vertex)

Biconnected Graphs (BiC Comp)

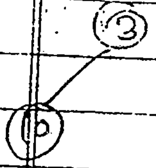
DFS

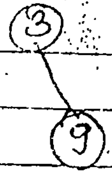
Biconnected Graph: should Not have Articulation point.

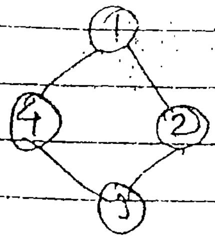


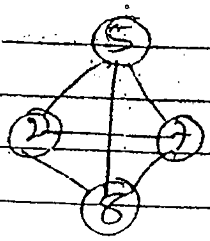
Maximul Component of Graph Which is Biconnect is said to be Biconnected component.

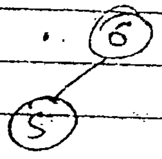
5 biconnected components











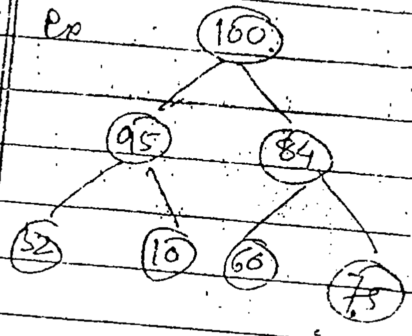
Biconnected Components: Max always
Max No of Bic graphs

✱ What

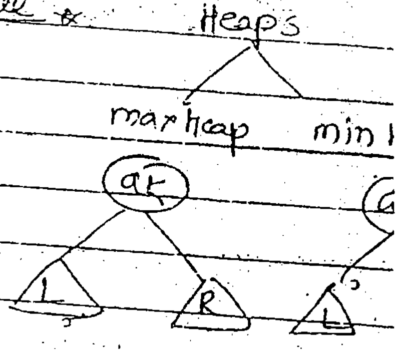
Heap & Heapsort

Heap is Complete binary Tree

ex



ex. max heap



$a_k > L$

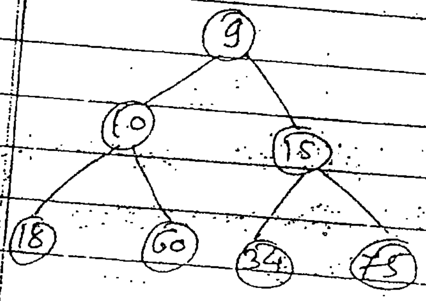
$a_k > R$

max heap

a_k

a_k

min



min heap

\# default heap is max heap.

Binary Search Tree : ($L \leq a_k < R$)

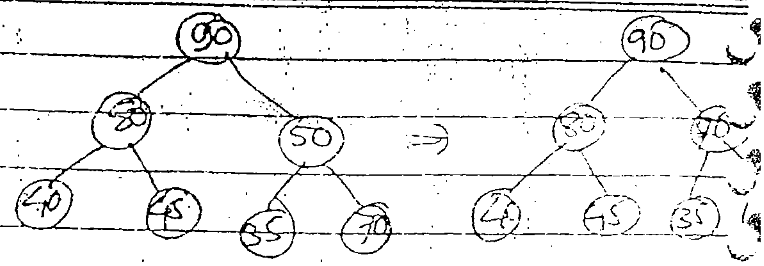
lexicalally ordered Tree

ⓐ Construction of heap:

Insertion method :

$< 40, 80, 35, 90, 45, 50, 70 >$

add to



Final

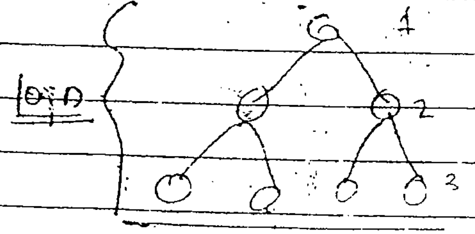
① Complexities

Best Case — Worst Case

if elements are in Decreasing order — Ascending order

Complexity

O(n) and O(nlogn)



n comparisons
logn more

for any node at level i

↳ i−1 Level Comparisons

max No of nodes at any level i = 2^{i-1}

for one level 2^{i-1} elements are present

No of Comparisions for each element is single level.

$$(i-1) \times 2^{i-1}$$

No. of Comparissions for Complete Tree.

$$F(n) = \sum_{i=1}^{K} (i-1) \cdot 2^{i-1}$$

$$n = 2^K$$
$$K = \log_2 n$$

$$F(n) < (K+1-1) \cdot 2^{K+1-1}$$

$$f(n) < K \cdot 2^K$$

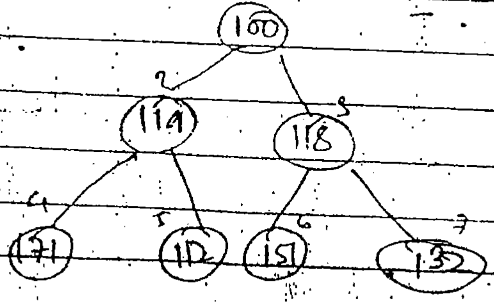
using $2^K = n$

$$f(n) < n \cdot \log_2 n$$

$$\therefore F(n) = O(n.\log n)$$

b) Heapify method

Transforming Binary Tree into Heap.



मुसीबतों को महत्त्व देने सें वे बढती हैं।

No. of Comparisions for each element in single level

$$(i-1) \times 2^{i-1}$$

No. of comparissions for Complete Tree.

$$F(n) = \sum_{i=1}^{K} (i-1) \cdot 2^{i-1}$$

$$n = 2^K$$
$$K = \log_2 n$$

$$F(n) < (K+1-1) \cdot 2^{K+1-1}$$

$$f(n) < K \cdot 2^K$$

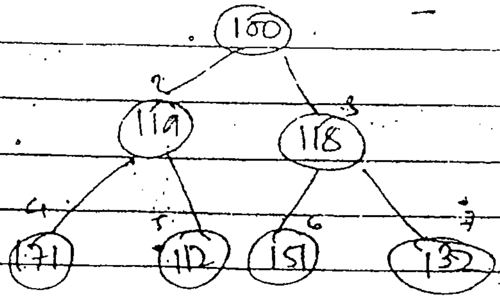
using $2^K = n$

$$f(n) < n \cdot \log_2 n$$

$$\therefore Fn\ O(n \log n)$$

b) Heapify method :

Transforming Binary tree into Heap



(JP)

vs
level

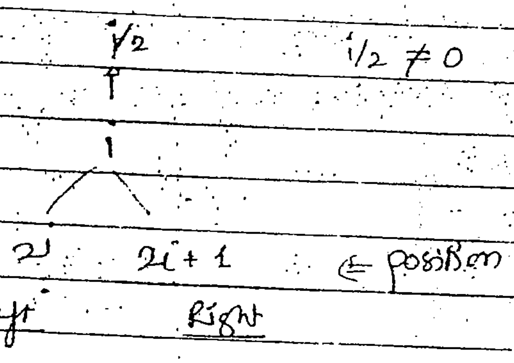
ree.

ry Tree

1	2	3	4	5	6	7
100	119	118	171	112	151	132
171	119	151	100	112	118	132

$i/2 \neq 0$

Storage



Heap

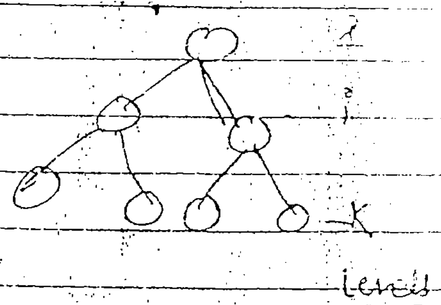
Compare Node with its both child
- Find Larger Child
- compare parent & Larger child.

if parent > Larger child
No change

if parent < Larger child
Swap.

Repeat above process till (Leaf Reached

Complexity



- For Any node being adjacent at level i requires

  K-i Comparisons downward

- max No. of Nodes at any level i are 2^{i-1}

- Total No. of comparisons for all Nodes at level i

  $$(K-1) \cdot 2^{i-1}$$

- Total No. of comp in Complete Tree of level K

  $$f(n) = \sum_{i=1}^{K} (K-i) \cdot 2^{i-1}$$

(ii) $1 \le i \le K-1$

multiply -

$$-1 \ge -i \ge 1-K$$

Add K to each term

$K-1 \geq K-i \geq 1$

$1 \leq K-i \leq K-1$ ✓

let $K-i = x$

$i = K-x$

$$\therefore f(n) = \sum_{1 \leq x \leq K-1} x \cdot 2^{K-x-1}$$

$$= \sum_{1 \leq x \leq K-1} \frac{x \cdot 2^K}{2^{x+1}}$$

$$= 2^K \sum_{1 \leq x \leq K-1} \frac{x}{2^{x+1}}$$

$$= \frac{1}{2^2} + \frac{2}{2^3} + \frac{8}{2^K}$$

decreasing fastly
can be equate to some
constant C

$= 2^K \cdot C$

but $n = 2^K$ $= nC$

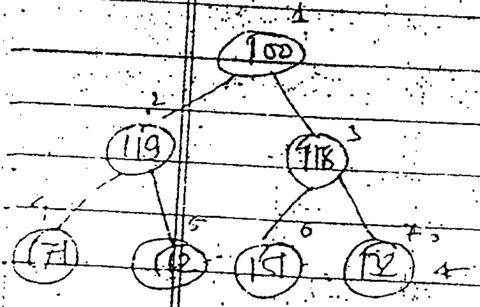
$$\boxed{\therefore F(n) = O(n)}$$

Heap soot

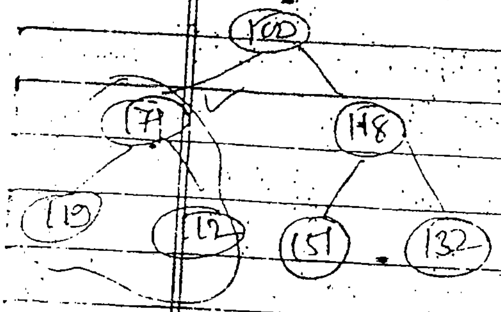
1) Heapify —— O(n)
2) Echange Root with ~~Right~~ last most child
3) Add all elements in Array ▦
4) Now the last element is pivoted (fixed)
5) Now consider 1 to last but 1 element only

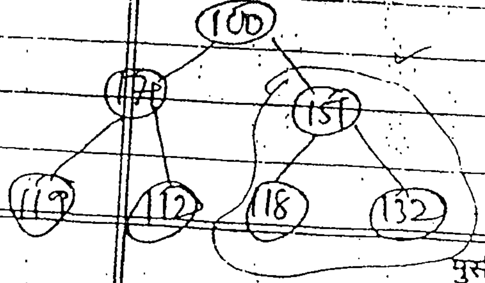
Reapeat 1—5

Heap Soot O(n.logn)

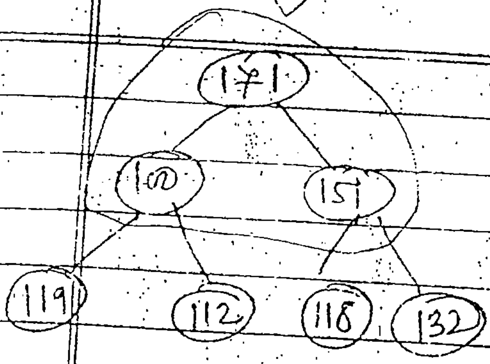


100 119 118 171 112 151 132





मुसीबतों को महत्त्व देने से वे बढ़ती है।



xed)

·max heap form.

Heap sort

171 100 151 119 112 118 13

exchange

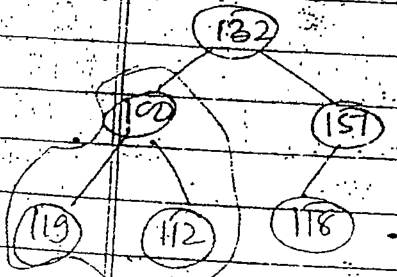
171 ⟷ 132

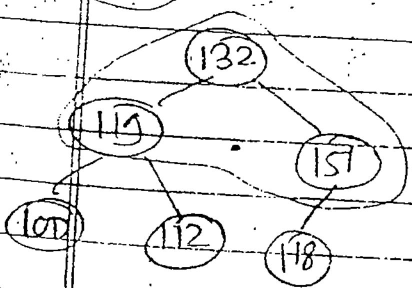
Consider 1—(n-1)
elements only

132

Fixe

132 100 151 119 112 118 $\boxed{171}$

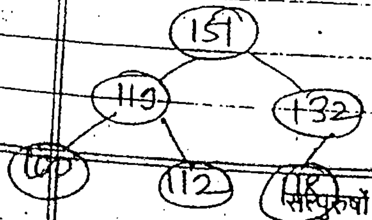


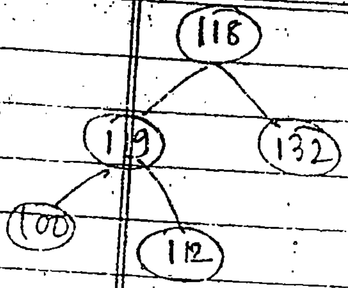


$\boxed{151 \leftrightarrow 118}$

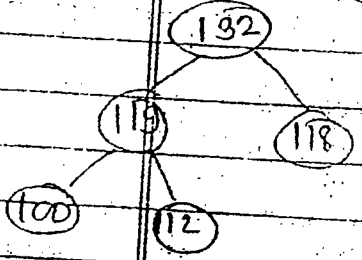
Fix

118 119 132 100 112 $\boxed{151, 171}$

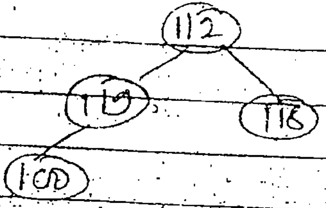




// Heapify



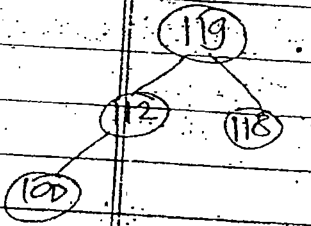
Exchange & Remove [132 ↔ 112]



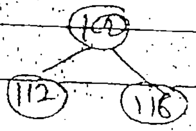
Heapify

(112, 119, 118, 100 [132, 151, 171]

Fix

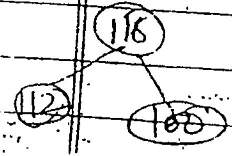


119 ↔ 100

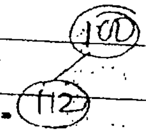


(100, 112, 118 [119, 132, 151, 171]

Fix

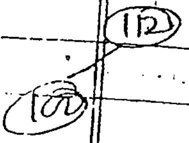


118 ↔ 100



Heapify

100, 112 [118, 119, 132, 151, 171]



112 ↔ 100

[100, 112, 118, 119, 132, 151, 171]

Fixed

P, NP, NP-Hard & NP-Complete Concepts

- Classification of problem
- Non Deterministic Computation
- problem defination
- Decision vs opt. problem
- classes of P & NP.
  - Cook's Theorem
  - Reducibility
  - polynomial Equivalance
  - NP-hard & NP complete

Fin 2,151,171 - Strategy to to NP-H and NP-C

- Case study.

151,171

problems
classification
Those have

procedure with Algo — (decidable)

procedure without Algo — (undecidable problem)

171

151,171

☆

ex. Halting prob. of TM.

procedure (decidable prob)
with Algo

Runs of two machines

deterministic m/c

Non deterministic m/c

Depending upon machine the problems have complexities in 2 types

① polynomial time based Time Complexity

(Tracable problems)

② Exponential based Time Complexity

(Intracable Time complexity)
Intracable probs

eg. Sorting, searching
JSD, greedy
dynamic

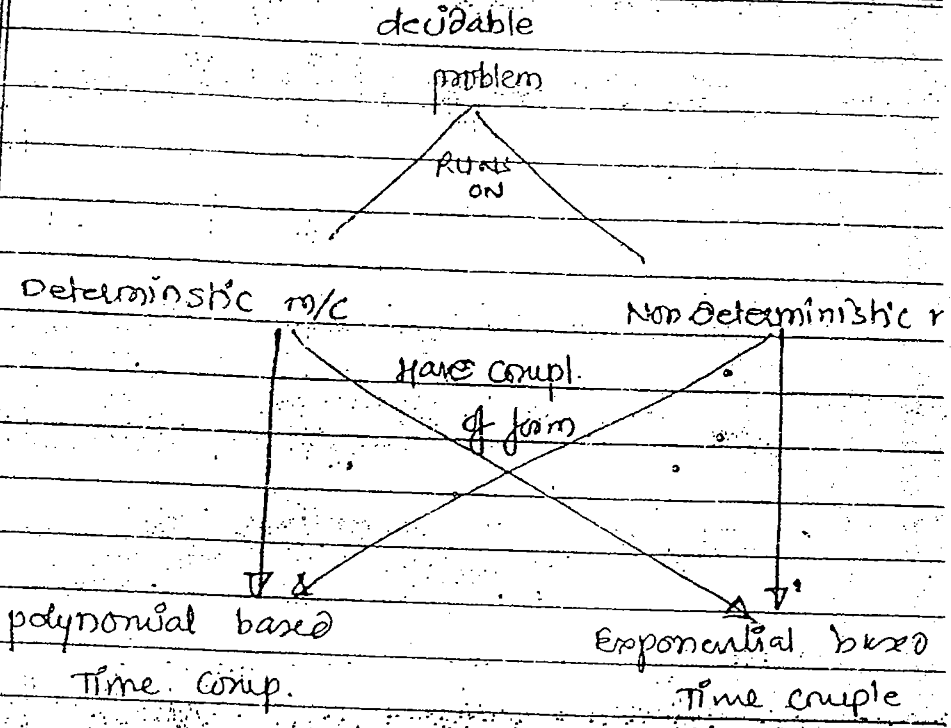
0/1 Knapsack
Graph Coloring
Hamiltonian
Trav salesman prob

Have complexity

a^{n}

I class

II class



Tractable   Intractable

a ?

objective of NPH NP-C

- it is mechanism
we try to proove prob of II class are Computationaly Related to each other

Why to proove ?

If Any one is solvable by a particular method
Then others prob can also be solved.

Non-deterministic Computation:

Non-Deterministic Algorithm:

Non-deterministic Algorithm contain operations whose outcomes are not uniquely defined but are limited to specified sets of possibilities.

The machine (Abstract) Executing such operations is allowed to choose any one of these outcomes, subject to a termination condition.

The Non-deterministic Algo introduces 3 New function

1) Choice(S) : arbitarily chooses one of element of S
2) Failure() : signals unsuccessfull complition
3) Success() : signals a successful complition.

Choice()
Failure()
Success()
} all have complexity O(1)

** A Non-deterministic algorithm terminates unsuccessfully if and only if there exists NO set of choices leading to successful signal.

Consider Algo

1 Non Deterministic Search :

A[1....n], x

(Search on Non-Deterministic m/c)

```
procedure ND.Search (A, n, x)
{
    1.  j ← Choice (1, n);                     — O(
    2   if ( A[j] = x ) then                   — O(
        {
              write (j)
              Success ();
    3   }                                        O(1
        write ('0')
           Failure();
}
```

Choice (1,n)

- chooses Arbitary No

it is Not Random No.

⇒ Complexity of this Algo O(1)

This is what power of NDM

The Best Search ever BS have

O(log n)

But

Non Determinis Machine

Search have Complexity O(1)

2 Non-deterministic ~~Search~~ Sorting:

N

```
                     Algorithm NSORT(A, n)
              {
initialise           for i = 1 to n
        O(n)                B[i] = ∅
                     for i = 1 to n
                     {
sort                        j ← choice (1, n)
        O(n)                if ( B[j] ≠ 0 ) then failure
                            B[j] = A[i]
                     }

                     for i = 1 to n-1
                     {
check   O(n)                if ( B[i] > B[i+1] then
                                          failure ();
                     }

              write( B[1 ... n] ) ;  success ()
        }
```

∴ Complexity - O(n) *

on Non-deterministic m/c

Real sorting tech.

No sorting tech on Det machine have comp less than O(n·log n)

Sorting

Comparision based

Non-comp. based
mechanical

Radix Sort

	1	2	3	4	5	6
A	5	18	10	3	6	4

temp							
	B	0	0	0	0	0	0

~~A[1]~~ A[1] is placed correctly in Array B

& so on.

```
for( i=1 to n )          every element of
{
    j = choice (1,n)     Return correct
                         position in B
    if ( B[j] ≠ 0)       Failure
}   B[j] = A[i]          put ith element
                         in j place
```

② Decision Vs optimization problem

(max / min)

Decision problem
O/p is
Yes No

optimization prob.
Identify optimal
value on
some criteria
defined in prob.

examples

- Hamiltonian cycle identify Ex Knapsack
- n-Queen prob.

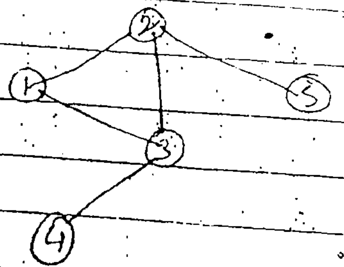
We Can Recast optimization prob as decision problem.

##

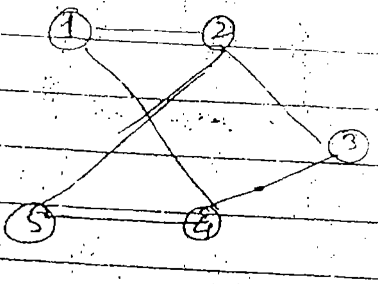
1 MAX-clique problem.

The maximal complete subgraph of a Graph G(V,E) is a clique.

G(V,E)

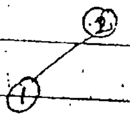


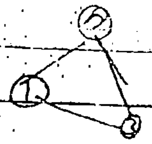
(i) G1



G2

Clique in G1





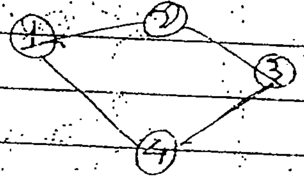
it is clique
but Not
max Clique.

max Clique
max. complete
Sub graph.

~~Sol~~ This is optimization prob.

G2

Clique : Complete Graph
- every vertex should be Reachable from every other Vertex.



Not Clique
b'coz it is Not complete
we cannot reach from ② → ④

These are optim. prob
we are solv. these by Decision method iteratively

⇒ does it have max Clique of n
n-1
n-2
⋮

2) Satisfiability problem:

propositional Formula.

$F(x_1, x_2 \cdots x_n)$ & operation $\land$ & $\lor$

literal x_i < prime $\bar{x}_i$, x_i

prapo. Formula

CNF — DNF

Conjunctive Normal Form. — Disjunctive NF

(POS) ($\land$) — ($\lor$) (SOP)

$F = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3)$

$\lor \quad \lor \quad \land \quad \lor \quad \lor$

K-CNF — K literals used

3 CNF — $\{x_1, x_2, x_3\}$

CNF General formula, K-CNF

$$F = \bigwedge_{i=1}^{n} C_i \qquad C_i = \bigvee_{i=1}^{K} x_i$$

product of C_i — $\lor$ (or) of all x_i

i 4 V

$F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge ..$

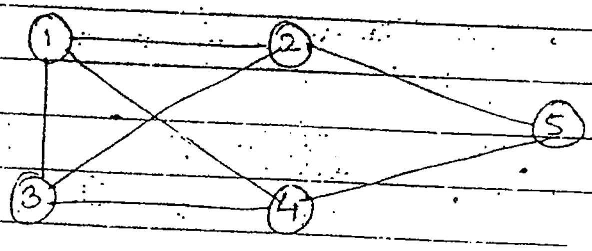
Satisfiability : decision prob:

does there exist set of values For ~~[illegible]~~ x_i such that the F value can be determined and it is True.

3) Node Cover problem:

~~[illegible]~~

$S \leq V$ of $G(V,E)$
is a node cover of graph,
iff all the edges of the graph are incident to atleast one vertex in S



- Minimum ~~Ro~~ Node Cover : {2,4}
all edges meet node ②, ④

also node cover is {1,3,5}
{1, 2, 4}

V. Imp.

P, NP, NP-Hard, NP-Complete.

P : P is the set of all decision problems solvable by deterministic algorithm ~~de~~ in polynomial time.

NP : NP is a set of all decision problems solvable by Non-deterministic algo in polynomial time.

since deterministic algorithm are special case of Non-deterministic prob

∴ P is subset of NP

P : searching, sorting

$P \subseteq NP$

NP : 0/1 knapsack, TSP, Hamiltonian cycle, graph colouring.

Halting problem < Not in P / Not in NP

⊛ Cook's Question.

1 Is there any single problem in NP, such that if it is shown to be in P then it would imply

$$P = NP$$

Theorem: NP

Satisfiability

satisfiability ⇒ N

b'coz

deterministic proble

of satisfiability has 2

complexity

$f(x_1, x_2, \ldots x_n)$

Is there exist values of x_1 to x_n such that it satisfy particular condition

Deterministic: check all possible combinations 2^n)

exponential

Non deterministic
for 1 → n
x_i ← choice(1
if ($F(x_1, x_2, \ldots x_n)$ is T
success
else
failure

O(n)

in P

in NP

Theorem:

Satisfiability is in P
if and only if P = NP.

P.

⑪ Reducibility :

let l_1 & l_2 be two problems

prob. L_1 reduces to l_2 if and only if there is a way to solve l_1 by a deterministic polynomial time algo using a deterministic algorithm that solves l_2 in polynomial time

denoted : $L_1 \propto L_2$ (reduces).

⇒ This defination implies that if we have polynomial time algo for l_2 then we can also solve l_1 also in polynomial time.

⇒ Reducibility is transitive but Not symmetric.

$L_1 \propto L_2$
$L_2 \propto L_3$
⇒ $L_1 \propto L_3$ ✓

$L_1 \propto L_2$
$L_2 \propto L_2$ ✗

⊛ But if $L_1 \propto L_2$ and
$L_2 \propto L_1$

∴ L_1 & L_2 are polynomially equivalent.

Np-Hard

A problem L is Np-hard if and only if satisfiability reduces to L.

if
by

L is Np-Hard.

Satisfiability α L.

(Halting problem NP H)

ex show l_1 is NpHard.

[illegible] L

[illegible] α L

show $L_0 \alpha L_1$

satisfi α L α L_1 ∴ satisfy α L1

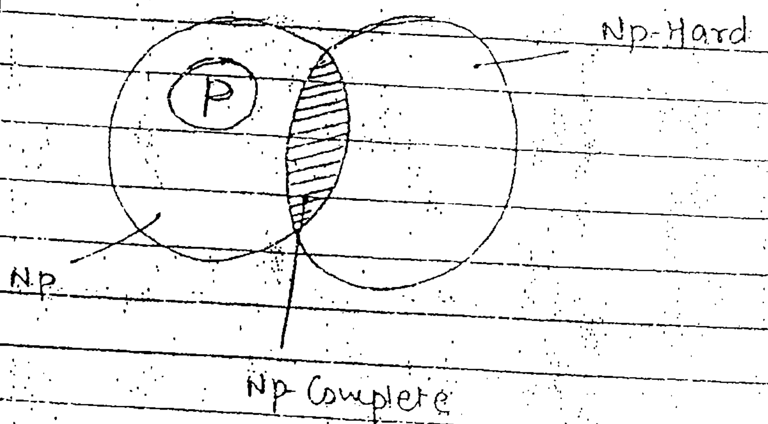
save we time

Np-Complete:

A problem is Np-Complete if and only if L is Np-Hard and L belongs to Np.

ex Halting problem is Not Np Complete.

B'coz Hp is Np Hard 1)
Hp $\notin$ NP 2)
Hp is [illegible] NPC



V. Imp

① Strategy to show prob is in Np-Hard

1 take prob L_1 already Known as Np-Hard.
[Seed satisfiability prob]
satisfy is Np Hard by Cook

2 Show how to obtain a (polynomial deterministic time) and instance I' Form L_2
From any instance i of L_1 such that from the solution of i' we can determine in polynomial time the solution to instance i of L_1

③ $L_1 \propto L_2$
④ satisfiability $\propto L_2$
⑤ L_2 is Np-Hard

1. if $L_2 \in Np$
Then L_2 is also Np-Complete.

Example.

1. Clique-Decision prob (CDP)
~~(K-SAT)~~ $L_1 \propto L_2$
CNF-satisfiability $\propto$ C.D.P

Claim: if formula f is satisfiable if and only if G has a clique of size 'K'

let $F = \bigwedge_{1 \le i \le k} C_i$ be a propositional formula in CNF.

let $1 \le x_i \le n$ be a variable in F

we show how to construct from F a graph G, consisting (V, E), such that G has a clique of size 'K' if and only if f is satisfiable.

G(V, E)

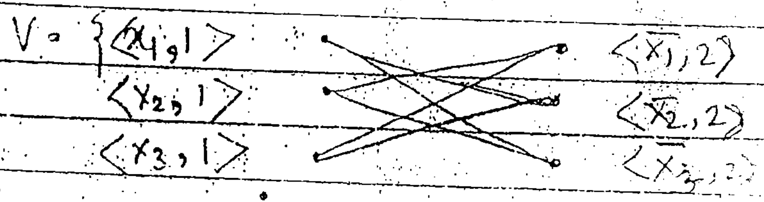
$V = \{ \langle \sigma, i \rangle / \ \sigma \text{ is a literal in } C_i \}$

$E = \{ \langle \sigma, i \rangle, \langle \delta, j \rangle \mid i \ne j \text{ and } \sigma \ne \bar{\delta} \}$

Ex. I of L_1

$F = \underbrace{(x_1 \vee x_2 \vee x_3)}_{\text{class}_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{\text{class}_2}$

I' of L_2



$G(V, E)$

∴ we can derive I' from I in polynomial time

maximum clique presense

$(\langle x_1, 1\rangle, \langle \bar{x}_2, 2\rangle) = 2 \quad K = 2$
$(\langle x_2, 1\rangle, \langle \bar{x}_1, 2\rangle)$

Satisfiability $\propto$ CDP

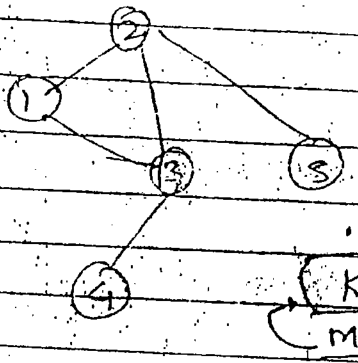
∴ CDP is NP-Hard

Node Cover Decision problem.

NCDP

$C.D.P \propto NCDP$

L_1 L_2

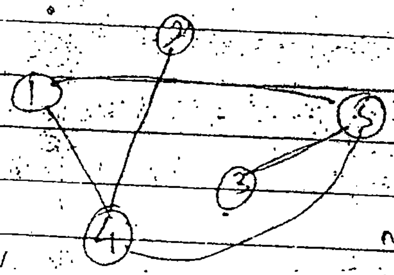


K=3

Max clique

G(V,E)

from graph G(V
derive a
graph G' as
follows.
$G' = (V,E)$
$E' = \{(u,v) \mid u,v \in E$ and $u,v \notin E\}$



G'

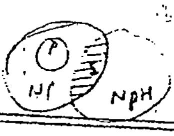
min. Node cover = n-K
= 5-3
= 2

∴ min. Node Cover = {4,5}
= 2

Satisfiability $\propto$ NCDP

& NCDP $\in$ NP

∴ NCDP is NP-complete



GateQ

Q. let S be an Np Complete prob.
Q, R be two other problems
Not known to be NP.
Q is polynomial time reducable to S and S ∝ R then which of foll. is TRUE

$Q \rightarrow S \rightarrow R$

⇒ S is NpC

Satisf ∝ S
S ∝ R

R is NpH

1 R is Np C
✓2 R is Np H
3 Q is Np C
4 Q is Np H

Q2 Let $\Pi_A \in NP$ then which of foll is TRUE

a) There is no polynomial time algo.

b) If Π_A can be solved in deterministic polynomial time then $P = NP$.

(c) If π_A is Np-Hard & it is Np.Com

(d) π_A may be undecidable.

NP-Hard, NP-Complete problems

decision problem:

A problem for which answer is either zero or one is called decision problem.

optimization problem:

A problem that involves identification an optimal value (either min/max) is known as optimization problem.

We are to discuss only Non-deterministic decision problem

→ successful complition : if o/p is 1

→ If there is NO sequence of choices leads to successful complition ⇒ 0

Many optimization problems can be recast into decision problems with property that the decision problem can be solved

- The classes NP hard and NP-Complete :

Reducability :

if $L_1 \alpha L_2$

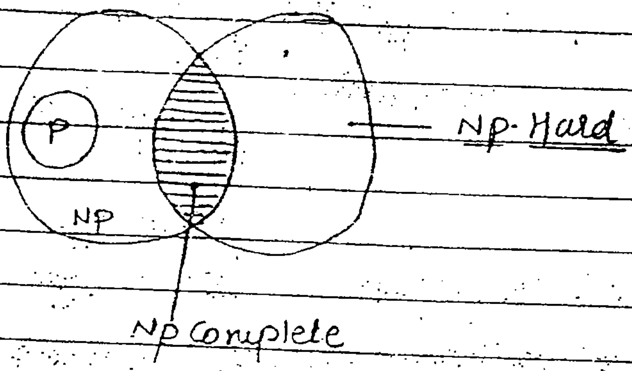
and if there is polynomial time algo for l_2,

then we can solve L1 in polynomial time.

Satisfiability:

problem L is Np-hard if and on
if satisfiability $\propto$ L
(satisfiability Reduces to L)

② A problem L is Np-complete if and only L is Np-hard and L $\in$ NP.



- *. only a decision problem can be Np complete.
- *. optimization problem may be Np-Hard.

⊛. if L1 : decision problem
L2 : optimization problem
Then it is quite possible that
$L_1 \propto L_2$

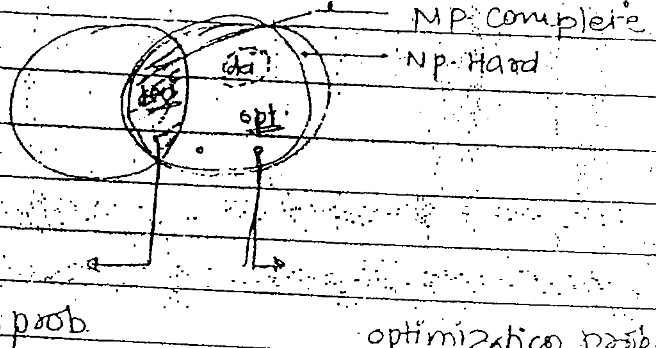
(∵) Knapsack decision problem reduces to Knapsack optimization problem

(∵) Clique decision problems also can be reduced to clique optimization prob.

∴ We can easily show that (also) optimization problems can be reduced to corresponding decision problems.

Yet optimization prob. cannot be NP-Complete.

There are NP-Hard decision prob. that are Not NP-Complete.

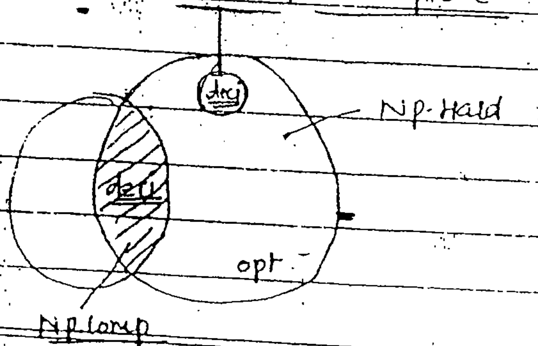


Only a decision prob. can be NP complete.

optimization problem may be NP-Hard

* optimization prob. cannot be NP complete

ⓑ There also exist NP-Hard decision problems & those are Not NP-Complete



- An Extreme Example of Np-Hard decision prob that is Not Np-complete is –
  "Halting problem of deterministic Alg

- prob. is undecidable
- There is No algorithm.
- It is Not P Nor Np

To show Satisfiability α halting prob
Algo. A halts if and only if X is satisfiable

∴ It is Np hard but Not in Np

(*) Two problems L_1 & L_2 are said to be polynomially equivalent
$L_1 \alpha L_2$ & $L_2 \alpha L_1$

vvimp

Cooks Theorem:

Cooks Theorem states that satisfiability is in P if and only if P = Np

*) we allready shown that satisfiability is in Np

B'coz Time complexity of satisfiability prob is $O(2^n)$
$F(X_1, X_2, X_3 \ldots X_n)$
2^n tries are req. for Max possibilit

⊛ Satisfiability is P iff P = NP.