Algorithms

Algorithm	Program
Design time	Implementation time
Domain knowledge	Programmer
any language	Programme language
H/w & OS	H/w & OS
Analyze	testing

Priori analysis and posteriori Testing :-

Priori Analysis	Posteriori Testing
1) Algorithm	1) program
2) Independent of language.	2) Language dependent.
3) Hardware independent	3) Hardware dependent
4) Time & space function.	4) watch time & Bytes.

Charecteristics of Algorithm :-

1) Input: These are ~~the~~ the values that are Supplied externally to the algorithm

2) output: These are the results that are ~~provided~~ produced by algorithm.

3) Definiteness: Each step must be clear and unambiguous.

4) Finiteness :- The algorithm must terminate after a finite number of steps.

5) effectiveness :- Each step must be feasible. i.e it should be practically possible to perform the step.

How to write an Algorithm :-

```
Algorithm Swap (a, b)
Begin
    temp ← a ;
    a ← b
    b ← temp ;
end.
```

```
Algo Swap (a, b)
{
temp = a ;   — (1)
a = b ;      — (1)
b = temp ;   — (1)
}
```

$f(n) = 3$

$O(1)$

Space (3 vars)

a – (1)
b – (1)
temp → (1)

$S(n)$ 3

$O(1)$

How to analyze an Algorithm :-

1. Time
2. Space

Time complexity :-

Types of Time function :-

$O(1)$ → constant

$O(\log n)$ → Logarithmetic

$O(n)$ → linear

$O(n^2)$ → quadratic

$O(n^3)$ → cubic

$O(2^n)$ → Exponential.

Asymptotic Notations :-

1) big-oh (O) upper bound

2) big-omega (Ω) Lower bound

3) Theta (Θ) Average "

Big oh :- Big oh notation is used to describe asymptotic upper bound.

Mathematically, if $f(n)$ describes running time of an algorithm; $f(n)$ is $O(g(n))$ iff there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c\, g(n) \quad \text{for all } n \geq n_0$$

Ex :- $f(n) = 2n+3$

~~$2n \leq 3$~~

$2n+3 \leq 10n \quad n \geq 1$

($f(n)$ ↑, $10n$ ↑ $c \cdot g(n)$)

$f(n) = O(n)$

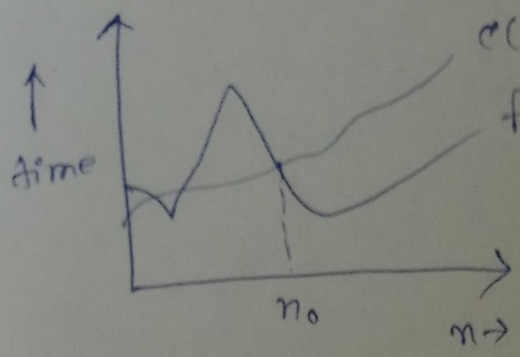
$2n+3 \leq 2n+3n$

$2n+3 \leq 5n \quad n \geq 1$

$2n+3 \leq 5n^2 \quad n \geq 1$

($f(n)$, $c\, g(n)$)

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



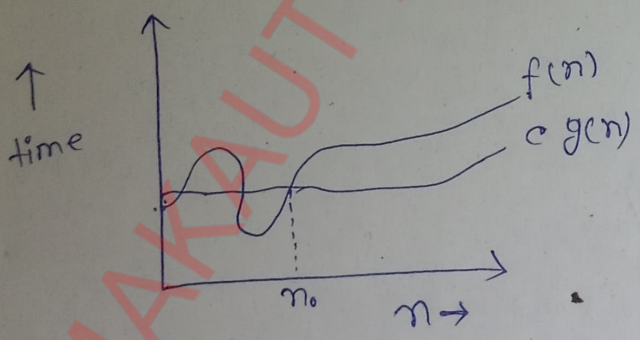
Big Omega (Ω) :- Just like O notation provides an asymptotic upper bound, Ω notation provides asymtotic lower bound. Let f(n) define running time of an algorithm;
f(n) is said to be $\Omega(g(n))$ iff there exists positive constant c and n_0 such that

$$0 \leq c\,g(n) \leq f(n)$$

Ex ~~f(x)~~ $f(n) = 2n+3$

$2n+3 \geq 1 \times n$
$f(n) \quad \uparrow c \; g(n)$

$f(n) = \Omega(n)$



Big theta notation (Θ) :-

Let f(n) define the running time of an algorithm f(n) is used to be $\Theta(g(n))$ iff f(n) is $O(g(n))$ and f(x) is $\Omega(g(n))$. both.

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

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

Mathematically Merging both the equations, we get

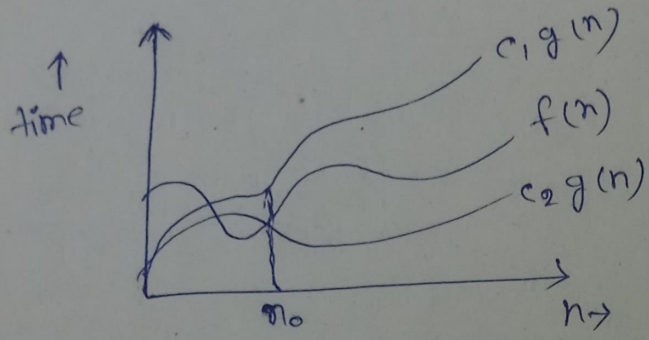
$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n) \quad \forall\, n \geq n_0$

Ex:- $f(n) = 2n+3$

$1 \times n \leq 2n+3 \leq 5 \times n$

$c_1 g(n) \quad f(n) \quad c_2 g(n)$

$f(n) = O(n)$



Big oh (O) :-

$f(n) = 2n^2 + 3n + 4$

$2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$2n^2 + 3n + 4 \leq 9n^2 \quad n \geq 1$

$c \cdot g(n)$

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

Big omega :-

$f(n) = 2n^2 + 3n + 4$

$2n^2 + 3n + 4 \geq 1 \times n^2$

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

Big theta :-

$1 \times n^2 \leq 2n^2 + 3n + 4 \leq 9n^2$

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

$f(n) = n^2 \log n + n$

$1n^2 \log n \leq n^2 \log n + n \leq 10n^2 \log n$

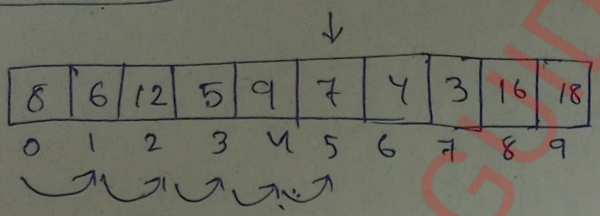
$\Rightarrow \Omega$

$\Rightarrow O$

Best, worst and Average case Analysis :-

1) Linear search
2) Binary search tree

Linear search :-



target = 7
target = 20

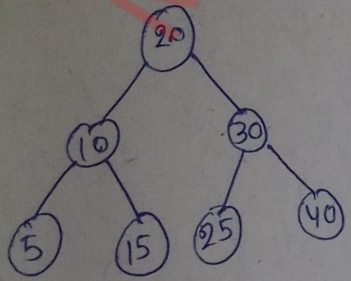
Best case: Searching key element present at first index [O(1)]

worst case: Searching a key at ~~inde~~ last index O(n)

Avg case: $\frac{\text{All possible case}}{\text{No. of case}}$

$$\frac{1+2+3+\cdots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2} \Rightarrow O(n)$$

Binary search tree (BST) :-



key = 15

$20 < 15$

$20 < 10$

~~20~~

$10 > 15$

Height $\rightarrow \log n$

Best case :- search root ele O(1)

worst " :- search for leaf ele O(log(n))

Recurrence

```
void Test (int n)
{
if(n>0)
{
Printf ("%d ",n);
  Test(n-1);
}
}
```

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

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

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

$= T(n-2) + 1$

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

$=$

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

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

$= T(n-3) + 1$

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

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

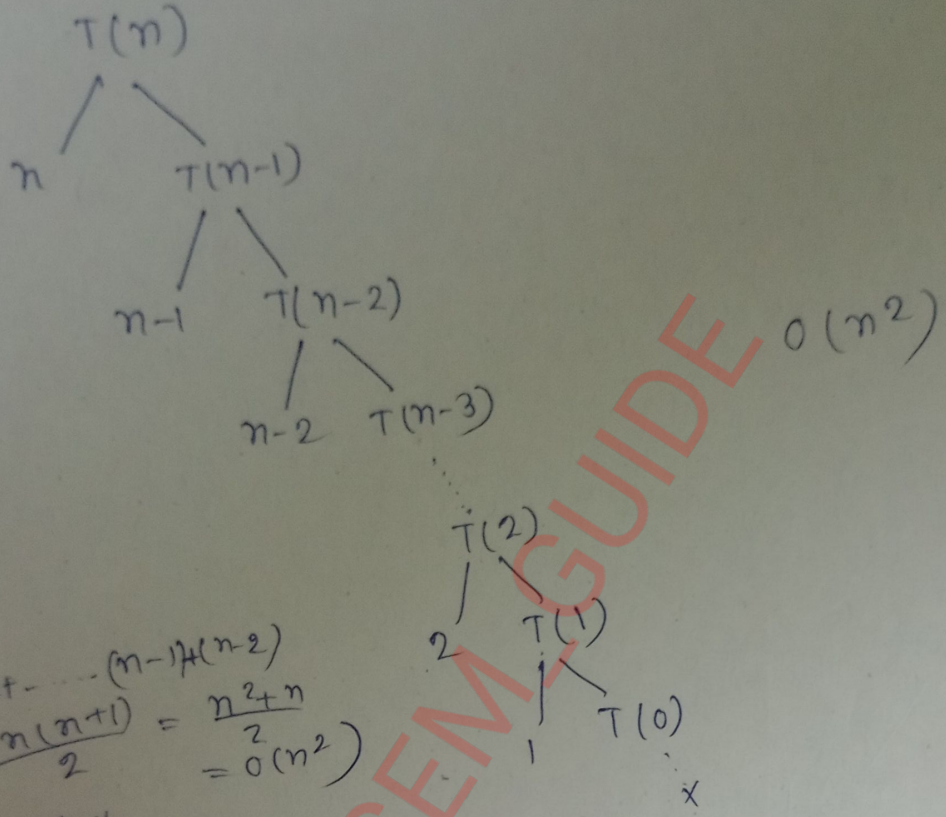
$T(n-k) = 0$

$n = k$

$T(n) = T(0) + n$

$= 1 + n$

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



$O(n^2)$

$1+2+\dots+(n-1)+(n-2)$

$= \frac{n(n+1)}{2} = \frac{n^2+n}{2} = O(n^2)$

Substitution method:-

Time complexity :-

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

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

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

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

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

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

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

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

$n-k = 0$

$n = k$

$= T(0) + 1 + 2 + 3 + \dots$

$= 1 + \frac{n(n+1)}{2} = 1 + \frac{n^2+n}{2} = O(n^2)$

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

$O(n \log n)$

Solving Recurrances :

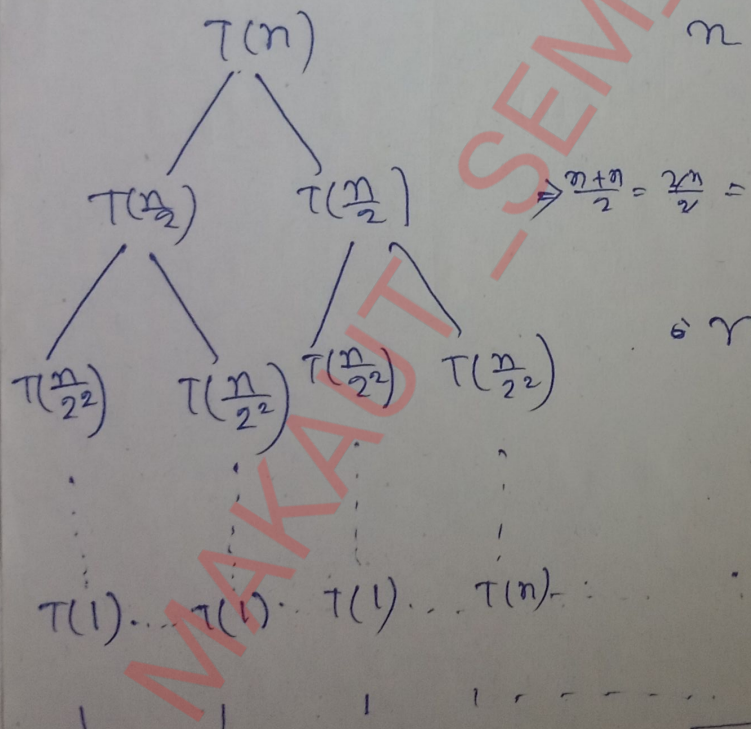
There are 3 methods to Solve recurrance.

1) Substitution method
2) Recurrance tree method
3) master method.

Iteration method.

Recurrance tree method :-

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



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

n

n

n

kn

$\Rightarrow n \log n$

$\therefore T(n) = O(n \log n)$

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

$n = 2^k$

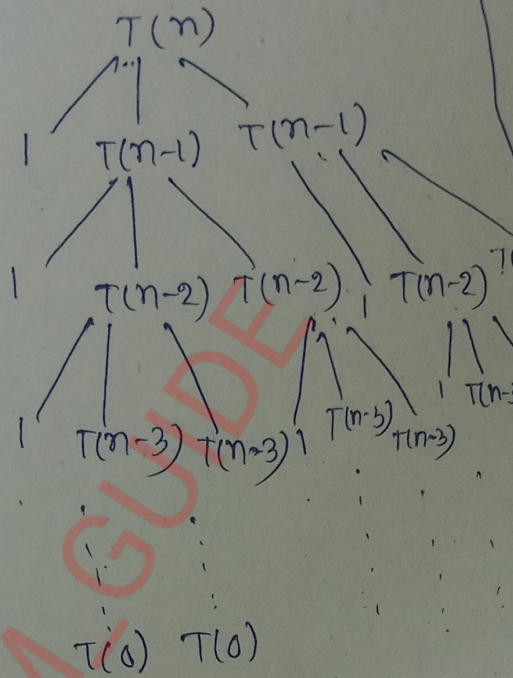
$\log n = k \log 2$

$\boxed{k = \log n}$

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

Incomplete

Recurrance tree :-




```
Algo Test(int n)
{
if (n>0)
  {
printf("%d",n);
Test(n-1);
Test(n-1);
  }
}
```

$n - k = 0$

$n = k$

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

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

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

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

$T(n) = 2\{2 \cdot (2T(n-3) + 1)\}$

Master theorem for Decreasing functions :-

$T(n) = T(n-1) + 1 \rightarrow O(n)$

$T(n) = T(n-1) + n \rightarrow O(n^2)$

$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$

$T(n) = 2T(n-1) + 1 \rightarrow O(2^n)$

$T(n) = 2T(n-1) + n \rightarrow O(n 2^n)$

1 case

if $a = 1$ $O(n^{k+1})$ or $O(n \cdot f(n))$

if $a > 1$ $O(n^k a^{n/b})$ or $O(f(n) a^{n/b})$

if $a < 1$ $O(n^k)$ or $O(f(n))$

$T(n) = aT(n-b) + f(n)$ → general form of recurrence relation

where $a > 0$, $b > 0$ and $f(n) = O(n^k)$ where $k \geq 0$

**

Master Theorem

gate smashers

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

$a \geq 1$ $\quad$ $b > 1$

Solution is:

$$T(n) = n^{\log_b a}\,[U(n)]$$

$U(n)$ depends on $h(n)$

$$h(n) = \frac{f(n)}{n^{\log_b a}}$$

Relation between $f(n)$ and $U(n)$ is

if $h(n)$	$U(n)$
n^n, $n > 0$	$O(n^n)$
n^n, $n < 0$	$O(1)$
$(\log n)^i$, $i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

$T(n) = 1 \cdot T(n/1) + 1$

master method

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

Example:

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

$a = 8$, $b = 2$

$$T(n) = n^{\log_b a}\,[U(n)]$$

$$= n^{\log_2 8} \cdot U(n)$$

$$= 3n^2$$

$$= n^{3\log_2 2} \cdot U(n)$$

$$= n^3 \cdot \underbrace{U(n)}_{1}$$

$$= n^3 \cdot 1$$

$$= O(n^3)$$

$$h(n) = \frac{f(n)}{n^{\log_b a}}$$

$$= \frac{n^2}{n^{\log_2 2}}$$

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

$$= n-1$$

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

$a = 1, \quad b = 2 \qquad f(n) = c$

Sol

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

$= n^{\log_2 1} U(n)$

$= n^0 U(n)$

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

$\log 1 = 0$

$n^0 = 1$

height

$h = \log n$

$h(n) = \frac{f(n)}{n^{\log_b a}}$

$= \frac{c}{n^0}$

$= c$

$h(n) = (\log_2 n)^0 : c$

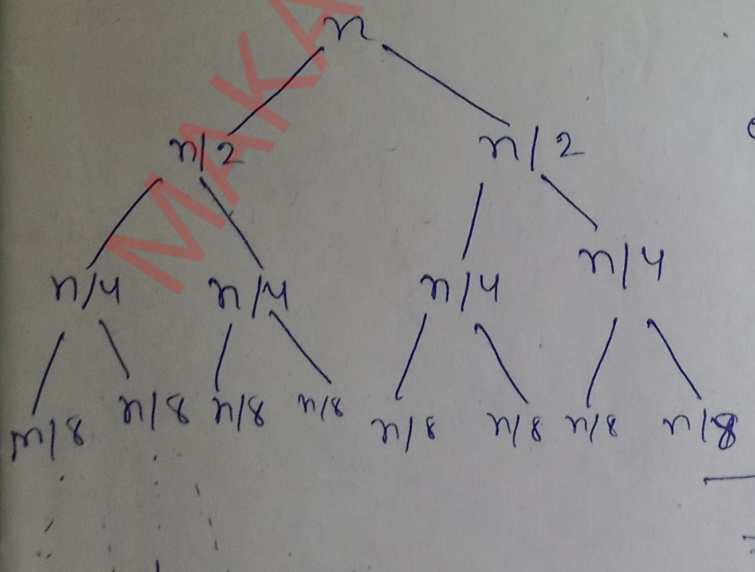
$U(n) = \frac{(\log_2 n)^{0+1} : c}{0+1}$

$= \log_2 n \, c$

$= O(\log n)$

Recurrance relation using tree method :

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



$= cn \log n$

$= O(n \log n)$

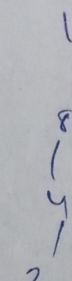
height of the tree if $n = 16$

$\log_2 n$

$= \log_2 16$

$= 4 \log_2 2$

$= 4$



height $= \log n$

$= cn \cdot \log n$

Time complexity $= O(n \log n)$

$\log n = k \log 2$

$\log n \neq k$

Shot on moto g31

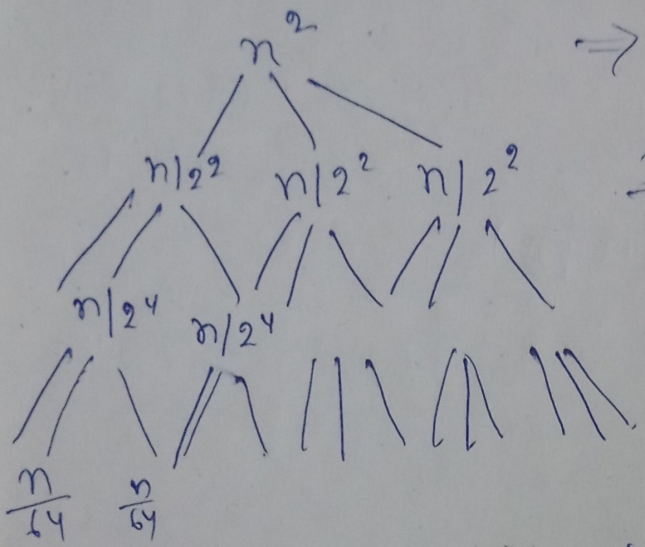
DIPANWITA❤️❤️

28 Jan 2025, 8:29 pm

Recurrance tree :

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

wrong



$\Rightarrow cn^2$

$\frac{3}{16} n^2$

$\left(\frac{3}{16}\right)^2 n^2$

$\left(\frac{3}{16} n^3\right) n^2$

kcn^2

$\frac{n}{4} = \frac{n}{4} = \frac{n}{16}$

$cn^2 = c\left(\frac{n}{4}\right)^2 = c\left(\frac{n^2}{16}\right)$

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

$n = 2^k$

$\log n = k$

$\Rightarrow \log n\, cn^2$

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

$\frac{\frac{n}{16}}{4} = \frac{n}{64}$

$c\left(\frac{n}{16}\right)^2 =$

$cn^2 + \frac{3}{16} cn^2 + \left(\frac{cn}{16}\right)^2$

$cn^2 + \frac{3}{16} n^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 n^2$

$= cn^2 \left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \cdots\right]$

$= cn^2 \left[1 + h + h^2 + h^3 + \cdots\right]$

G.P series

$\frac{1}{1-h}$ $h < 1$

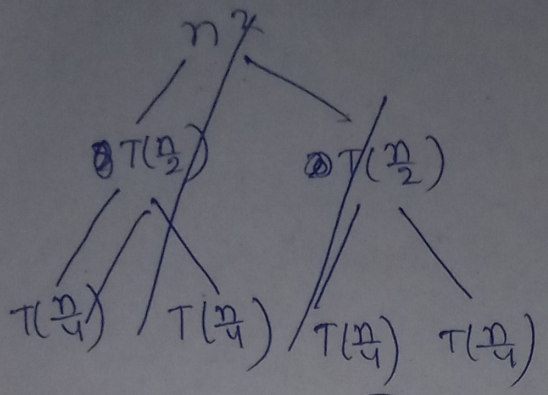
$= cn^2 \times \frac{1}{1 - \frac{3}{16}}$

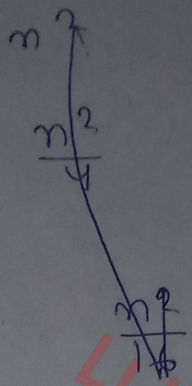
$= cn^2 \left(\frac{16}{13}\right)$

$= O(n^2)$

Recursion tree method :

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

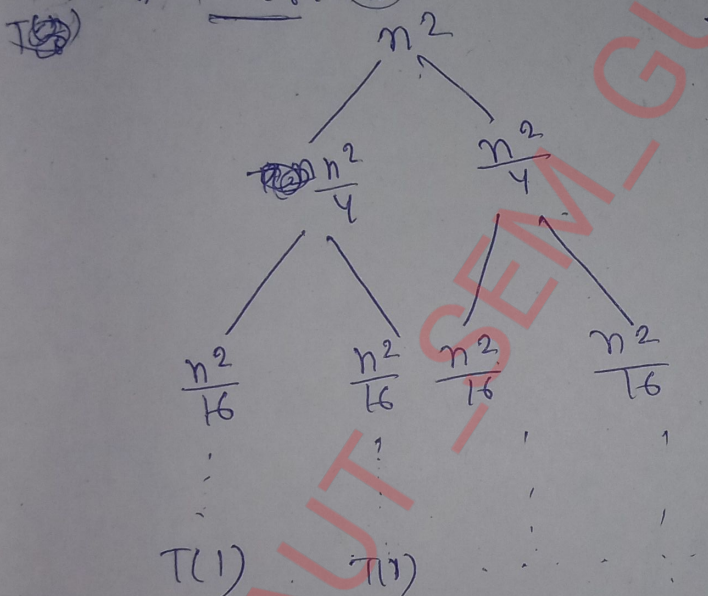




$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n^2}{4}$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n^2}{16}$

$T(\frac{n}{4})$ $\frac{n^2}{16}$



n^2

$n^2/2$

$n^2/4$

$n^2/2^{k-1}$

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

$n^2 = 2^k$

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

$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \cdots + \frac{n^2}{2^{k-1}}$

$= n^2\left[\left(\frac{1}{2}\right)^0 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \cdots + \left(\frac{1}{2}\right)^{k-1}\right]$

$= n^2 \times \frac{1}{1 - \frac{1}{2}}$

$= n^2 \times 2$

$= 2n^2$

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

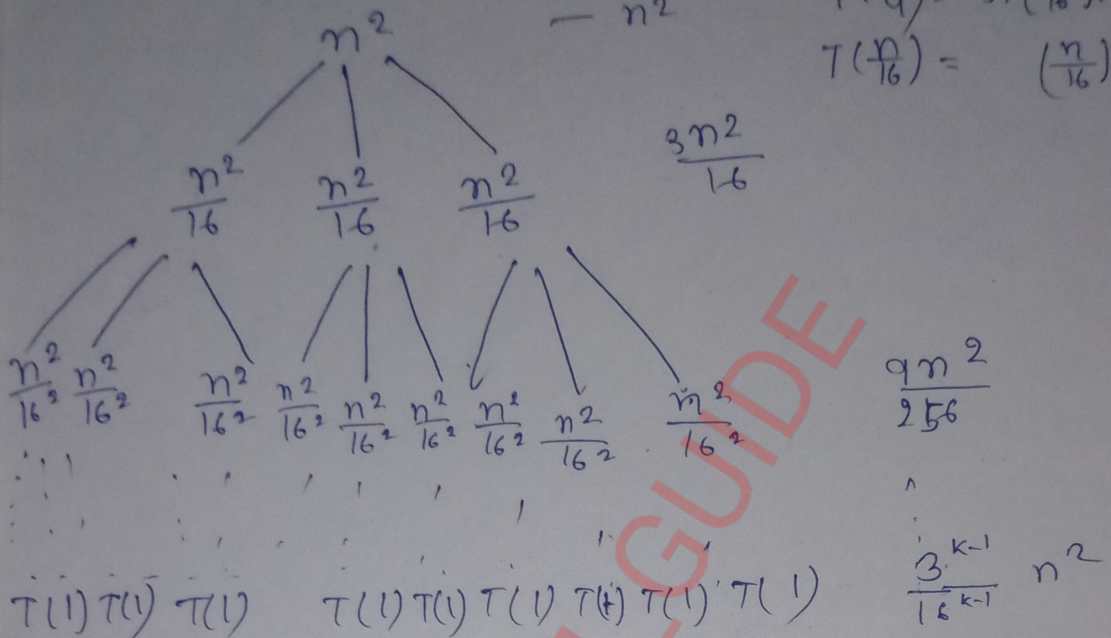
Recursion Tree method :-

$T(n) = 3T\left(\frac{n}{4}\right) + n^2$

non recursive part $= n^2$

$T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{16}\right) + \left(\frac{n}{16}\right)^2$

$T\left(\frac{n}{16}\right) = \left(\frac{n}{16}\right)^2$



$\frac{n^2}{4^k} = 1$

$n = 4^k$

$\log n^2 = \log 4^k$

$n = k \log 2^2$

$n = 2k \log 2$

$n = 2k$

$T(n) = n^2 + \frac{3n^2}{16} + \left(\frac{3}{16}\right)^2 n^2 + \left(\frac{3}{16}\right)^3 n^2$

$= n^2\left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \cdots\right]$

$= n^2\left[1 + h + h^2 + h^3 + \cdots\right]$

$= n^2 \times \frac{1}{1 - \frac{3}{16}}$

$= n^2 \times \frac{16}{13}$

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

SOLVE recurrance relation using genarating function.

Divide and conquer :

Divide and conquer is top down approach to designing algorithm consist following phase

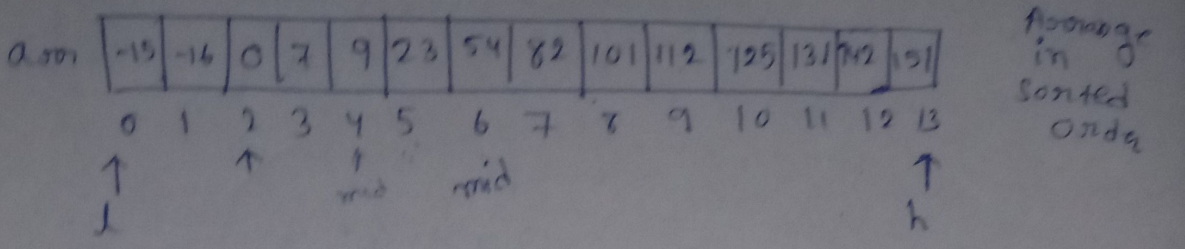
1) Divide: divide the problem into subproblem that similar to the original problem and smaller in size.
2) conquer: Solve the subproblem recursively.
3) combine: combine these solution of subproblem to create solution of original problem.

Algorithm :-

```
Algo DAC (P)
{
if ( Small(P) then return S(P)        → solution
else
  {
  divide P into smaller P1, P2, .... Pk   where k≥1
  Apply DAC to each subproblem Recursively
  Return combine (DAC(P1), DAC(P2)... DAC(Pn))
  }
}
```

Example Merge Sort Algorithm.

Binary Search (Divide & conquer) :-



Arrange in Sorted Order

Algo BinSearch (arr, l, high, target)

```
{
int low=0;
```

```
int binary_search (int *arr, int size, int value)
{
int low=0;
int high= size-1;
int mid;
while (low <= high) {
mid = (low+high)/2;
if(arr[mid]==value) {
return mid;
} else if (arr[mid] < value) {
low= mid+1;
} else {
high= mid-1;
}
}
return -1;
}
```

target = 23

mid = $\frac{0+13}{2}$

= 6

54 > 23

high = mid - 1

= 5

mid = $\frac{0+5}{2}$

= 2

2 < 23

low = mid + 1

= 3

high = 5

mid = $\frac{3+5}{2}$

= 4

9 < 23

low = mid + 1

= 5

high = 5

mid = 5

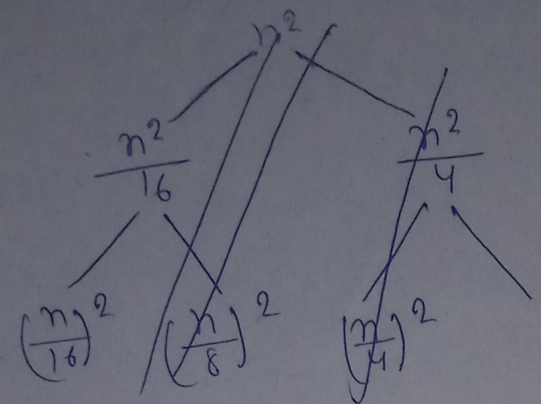
arr[5] = 23

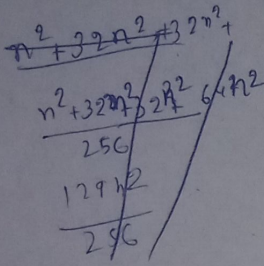
item found

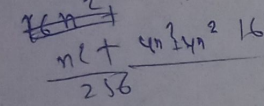
(Subir)

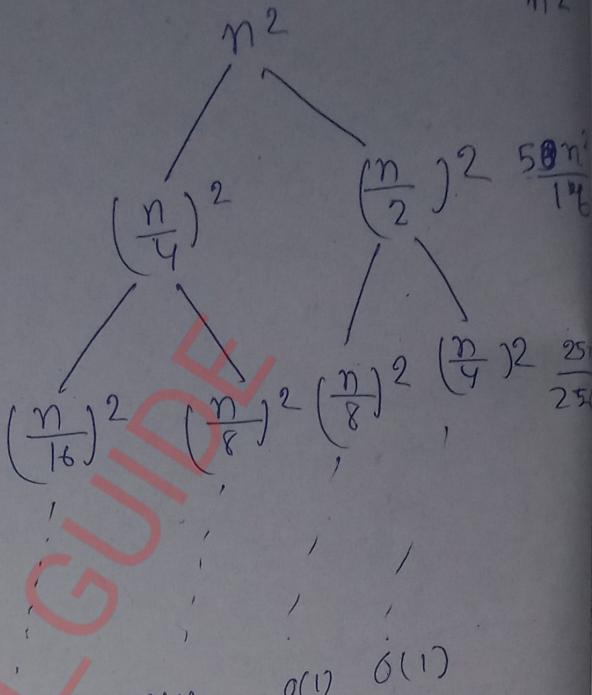
Recursion tree method:-

① $T(n) = T(n/4) + T(n/2) + n^2$









$O(1)$ $O(1)$ $O(1)$ $O(1)$

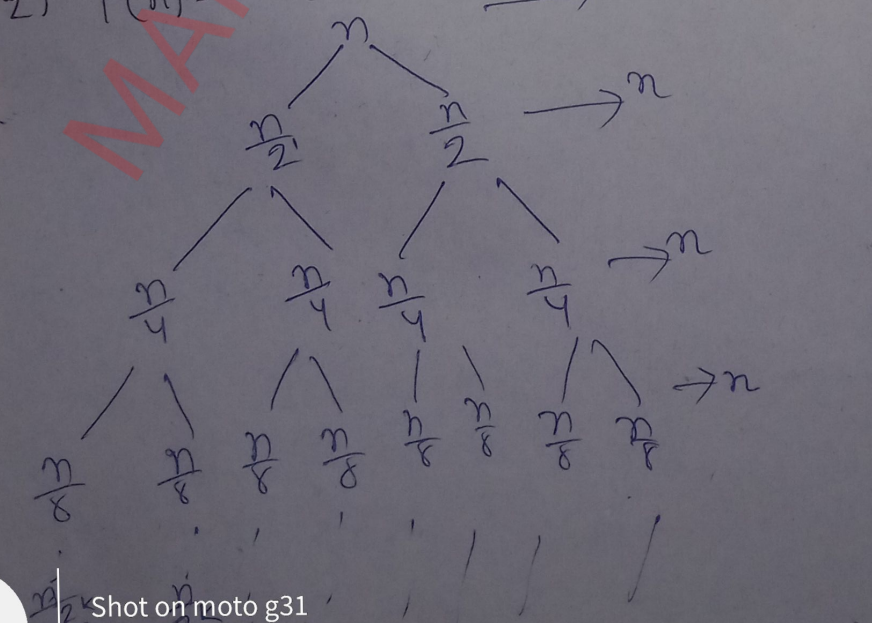
$$T(n) = n^2 + \frac{5}{16}n^2 + \frac{25}{256}n^2 + \cdots$$

$$= n^2\left[1 + \left(\frac{5}{16}\right)^1 + \left(\frac{5}{16}\right)^2 + \cdots\right]$$

$$= O(n^2)$$

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

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



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

$n = 2^k$

$\log n = k$ → height

$n \log n$

$T(n) = n + n + n + n \cdots + cn$

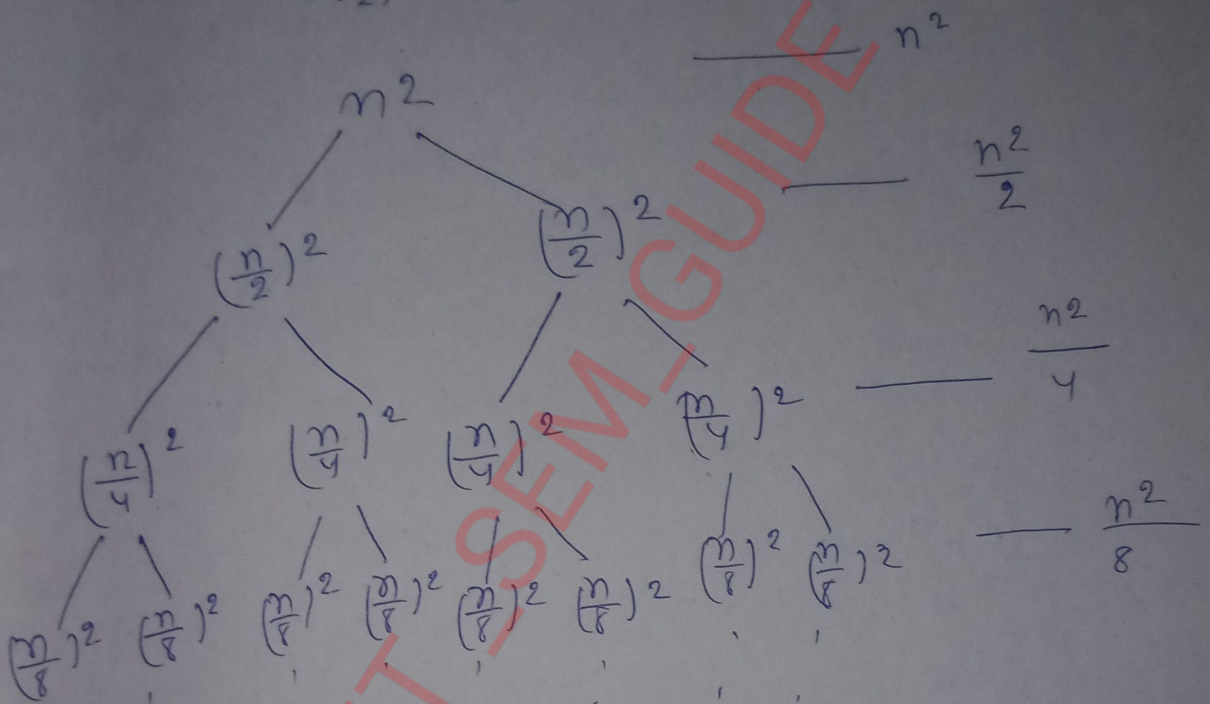
$= k \cdot n$

$= n \log n$

$= \boxed{O(n \log n)}$

3) $T(n) = 2T(\frac{n}{2}) + n^2$

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



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

$= n^2 \left[1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots\right]$

$= O(n^2)$

Substitution Method

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

~~Guess: O~~

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

Guess: ~~$T(n)=O(\log n)$~~ $T(n) = O(\underbrace{\log n}_{g(n)})$

Induction goal: $T(n) \le d \log n$

Induction hypothesis: $T(\frac{n}{2}) \le d \log(\frac{n}{2})$

Proof of induction goal:

$T(n) = T(\frac{n}{2}) + c \le d\log(\frac{n}{2}) + c$

$\le d\log(\frac{n}{2}) + c$

$= d\log n - d\log 2 + c \le d\log n$

if: $-d+c \le 0$, $d \ge c$

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

Guess: $T(n) = O(\underbrace{n^2}_{g(n)})$

Induction goal: $T(n) \le cn^2$

Induction hypothesis: $T(n-1) \le c(n-1)^2$

Proof of induction goal:

$T(n) = T(n-1) + n \le c(n-1)^2 + n$

$= c(n^2 - 2n + 1) + n$

$= cn^2 - (2cn - c - n) \le cn^2$

if $2cn - c - n \ge 0 \Rightarrow c(2n-1) - n \ge 0$

$c \ge \frac{n}{2n-1}$

$c \ge \frac{2n-1}{n}$

$c \ge (2 - \frac{1}{n})$

$\frac{n}{2n} \to \frac{1}{2}$, $\frac{n}{1} \to 1$

$c \ge 1$

For good guess

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

Case 1:

$a = b = 2$

$T(n) = O(f(n)\log n)$

Case-2

$a = 1$, $b =$ any value

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

Case 3

$a \ne 1$

$b > a$

$T(n) = O(f(n).n)$

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

Guess : $T(n) = O(n \log n)$

Induction goal: $T(n) \leq c\, n \log n$

Induction hypothesis : $T(\frac{n}{2}) \leq c\, \frac{n}{2} \log \frac{n}{2}$

Proof of induction goal:

$T(n) = 2T(\frac{n}{2}) + n \leq 2 \cdot c\, \frac{n}{2} \log \frac{n}{2} + n$

$= cn \log n - cn + n \leq cn \log n$

$n(-c+1) = 0$

$-c = -1$

i.f: $-cn + n \leq 0$

$\Rightarrow c \geq 1$

5) Prove that if $f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n^1 + a_0$ then $f(n) = O(n^m)$.

6) What is algorithm? Define the charecteristics and categories of Algorithm.

The important categories of algorithms are:

i) Search :- Algorithm to search an item in data structure.

ii) Sort: Algorithm to sort items in a certain order.

iii) Insert: Algorithm to insert item in a data structure.

iv) Update: Algorithm to update an existing item in a data structure.

v) Delete: Algorithm to delete an existing item from a data structure.

7) Write an algorithm to add two numbers and discuss the time and space complexity.

Ans

```
Add(a,b) :-
{
    a = 10;
    b = 20;
```

```
x = a+b;    1
return x;   1
}
```

$S(P) = C + S_p$

Time complexity $f(n) = 4$

~~$O(1)$~~

Space complexity

Space complexity = constant + variable part

Space complexity

a = 1
b = 1
x = 1

$S(n) = 3$ $O(1)$.

Q) Calculate the space for recursive factorial function of an element.

Ans :-

$$T(n) = \begin{cases} a & n=0 \\ T(n-1)+b & n>1 \end{cases}$$

```
Factorial(n)
{
if n==0
  return 1
else
  return n x Factorial(n-1)
}
```

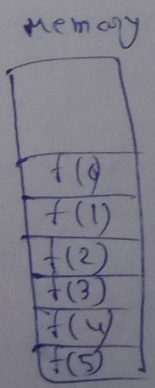
Space complexity:

n
(n+1)

f(5)
↓
f(4)
↓
f(3)
↓
f(2)
↓
f(1)

Max- depth = 5

Space on $O(n)$



Sem Preparation / Previous qn

a) Calculate the space requirement for
i) Sequential search algorithm / Linear Search
ii) Find an element from a set of array element.

Algorithm:

```
Search(A[], N, K)
{
for i = 0 to N-1
    if K == A[i] then
  return i;
return -1;
}
```

K = key ele

-1 (invalid)

Space complexity:

A → n
N → 1
K → 1
i = 1

n+3

= 2(n+1) O(n)

=

b) Find the time complexity of an algorithm that computes the multiplication of a numbers stored in an array.

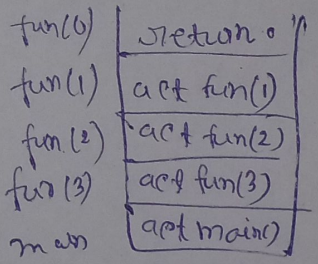
```
Multiply(a, n)                    1
{
initialize m ← 1;                 1
for i ← 1 to n do                 n-1
    m = m * a[i];                 n
return                            1+1+n-1+n+1
```

2)b) what is the tail recursion? Give an example.

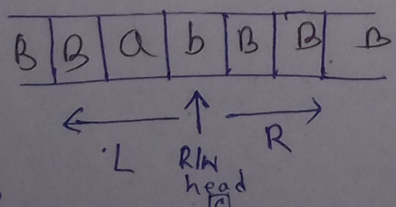
A recursive function is called tail recursive if recursion is the last thing done by function. There is no need to keep record of previous state.

```
void fun(int n)
{
if(n==0)
    return;
else
    Printf("%d", n);
return fun(n-1)
}
int main()
{
fun(3);
return 0;
}
```

3 2 1



a)b) write a short note on Turing machine



Read write control

$M = (Q, \Sigma, T, \delta, q_0, F, B)$ 7 tuples

Q = set of states

Σ = Input alphabet (a, b)

T = Tape alphabet (a, b, B)

δ = Transition function.

q_0 = Initial state

B = Blank Symbol

F = Final States ($F \subseteq Q$)

Q. write a short note on Recursion tree.

Key points :-

1) Steps to Solve Recurrance Relations using recurrence tree method.

2) ~~Set~~ Step by step will see how to draw recurrance tree.

3) Finally how to find time complexity using recurrance tree.

Steps :-

1) Draw a recurrance tree.

2) calculate time taken by every level of tree.

3) Finally we sum the work done at all levels.

Com Predavation | Dipanwita Das

Chap-2

*1) What is union-find algorithm? give an example.

⇒ A union-find Algorithm is an algorithm that performs two useful operations on such a data structure:

Find → (Determine which set a particular element is in. Also useful for determining if two elements are in the same set.)

Union → (combine or merge two sets into a single set.)

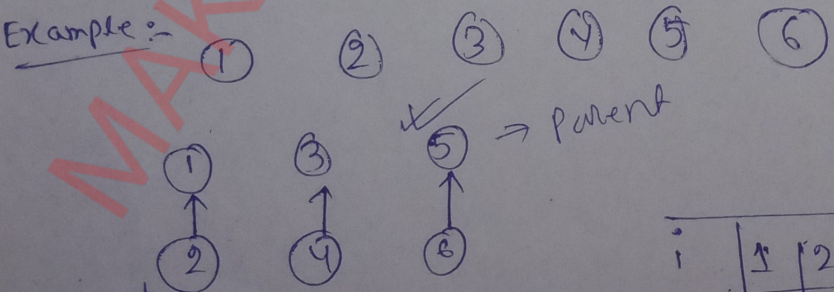
Because it supports these two operations, a disjoint-set data structure is sometimes called a merge-find set.

Algo:

```
Simple-Union (i,j)
{
P[j] = i;
}
```

Time complexity O(n)

Example:-



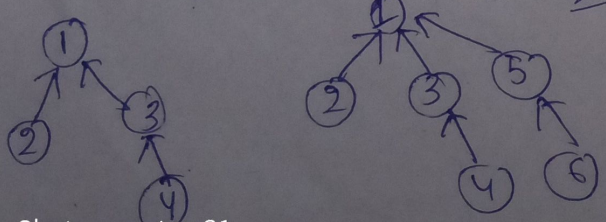
Apparent

(taking root node)

union (1,3)

union (1,5)

i	1	2	3	4	5	6
1st it P[i]	-1	1	-1	3	-1	5
2nd iter	-1	1	-1	3	-1	5
	-1	1	-1	3	1	5



Find Algorithm :- Searches for an element returns root node if element is available.

Algorithm :-

```
Find (i)
{
while (P[i] ≥ 0)
  {
  i = P[i];
  }
return i;
}
```

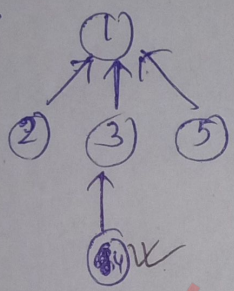
(1)

4, 3, 1

P[4] ≥ 0	P[3] ≥ 0	Time complexity
3 ≥ 0	P[3] = i	
i = 3 ; ≥ 0		O(n²)
i = 1	-1 ≥ 0 F	

Find 4

Example :-



i	1	2	3	4	5
P[i]	-1	1	1	3	1

Trying to find element 4

Parent of 4 is 3

" " 3 " 1

return 1

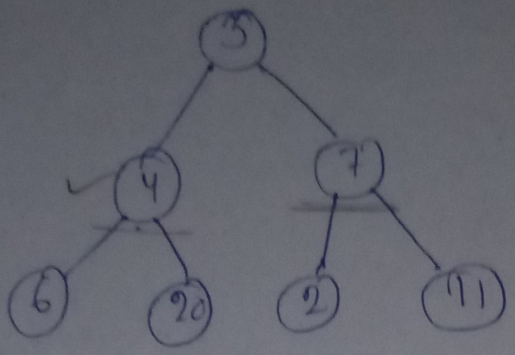
2) What is the basic characteristics of a greedy Algorithm ? or what do you mean by greedy method ?

A Greedy algo is a strategy that makes the optimal choice at each stage with hope of finding a global optimal.

Pros : simple & easy to implement, run fast

Cons : not always yield optimal solution.

Ex:



Minimization:

Greedy approach:

3 + 4 + 6 = 13

actual:

3 + 7 + 2 = 12

Maximization:

Greedy: 3 + 7 + 11 = 21

actual:

3 + 4 + 20 = 27

** Application:

1. Activity selection problem.
2. Huffman coading
3. Job Sequencing problem
4. Fractional knapsack problem
5. finding minimum spanning tree
6) Single source shortest path

3) State the 0/1 knap sack problem?

**

The 0/1 knapsack problem is a classic optimization problem where the goal is to maximize the value of items selected, given a constraint on the total weight. Each item has a weight and a value, and the knapsack has a maximum weight capacity. The problem is called '0/1' because for each item, you can either take it (1) or leave it (0), you cannot take a fraction of an item.

Here's the formal definition:

Given:

- n items, indexed from 1 to n.
- for each item i, a weight w_i and a value v_i.
- A knapsack of capacity W.

Find:

A subset S of items such that the total weight does not exceed W and the total value is maximized.

Example:

Let's say we have 4 items with their respective weights and values:

item 1: $w_1 = 2, v_1 = 10$

Item 2: $w_2 = 3, v_2 = 15$

item 3: $w_3 = 5, v_3 = 20$

item 4: $w_4 = 7, v_4 = 25$

And the knapsack capacity is $W = 10$

we need to find the subsets of items to maximize the total value without exceeding the knapsack's capacity.

Brute force

A brute force algorithm simply tries all possibilities until a satisfactory solution is found.

String Matching :-

Example :-

NOBODY NOTICED → NOT (target) (search)

N O B O D Y N O T I C E D

1. N O T
2) N O T
3) N O T
4) N O T
5) N O T
6) N O T
7) N O T
8) N O T

return element found

Algorithm Bruteforce SM (T[0...n), P[0...m])

// Bruteforce string pattern matching
// i/p → array T (text) & P (pattern)
// o/p → element position, -1 (if not present)

for i ← 0 to n-m do
    j ← 0
    while j < m P[j] = T[i+j] do
        j ← j+1
    if j = m return i

Time complexity :

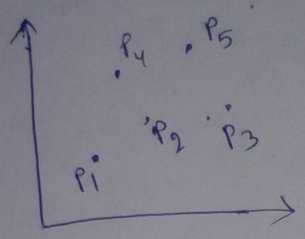
$(n * m)$

$O(n) \approx O(n \cdot m)$

$t(n) \approx O(nm)$

Closest pair problem :

Ex



P1 to P2
P1 " P3
P1 " P4
P1 " P5
P1 " P6

P2 to P3
P2 to P4
P2 " P5
P2 " P6
~~P2 "~~
~~P2 "~~

(1,2) (3,4)
x_1, y_1 x_2, y_2

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Algorithm :-

Algorithm BFClosestPair(P)
// Closest pair using Brute force.
// input Set of point with coordinates
// op index 1 & index 2

dmin ← ∞
for i ← 1 to n-1 do
  for j ← i+1 to n do
    d ← sqrt((xi + xj)^2 + (yi + yj)^2]
    if d < dmin
      dmin ← d ; index1 ← i ; index2 ← j

return index1, index2

Complexity:- $C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2$ → Basic operation

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

$= 2[(n-1) + \ldots]$

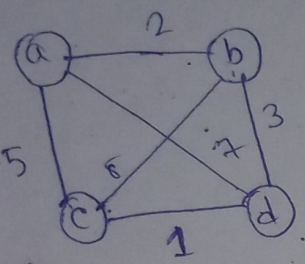
$= \frac{n(n-1)}{2}$

$= n(n-1)$

$= n^2 - 1$

$= O(n^2)$

Travelling Salesman problem - Exhaustive search



Feasible Soln		cost
$a \to b \to c \to d \to a$	$\Rightarrow$	$2+8+1+7 = 18$
$a \to c \to b \to d \to a$	$\Rightarrow$	$5+8+3+7 = 23$
$a \to d \to b \to c \to a$	$\Rightarrow$	$7+3+8+5 = 23$
$a \to b \to d \to c \to a$	$\Rightarrow$	$2+3+1+5 = 11$
$a \to c \to d \to b \to a$	$\Rightarrow$	$5+1+3+2 = 11$
$a \to d \to c \to b \to a$	$\Rightarrow$	$7+1+8+2 = 18$

optimal :

$a \to b \to d \to c \to a$ $\Rightarrow 11$

$a \to c \to d \to b \to a$

Knapsack

Example :

item	I
weight	
value	

Subset

{I 1}
{I 2}
{I 3}
{I 4}
{I1, I2}
{I2, I3}
{I3, I4}
{I1, I3}
{I1, I4}
{I2, I4}
{I1, I2, I
{I1, I2, I
{I1, I3, I
{I2, I
{I1, I2

I3,

Knapsack Problem - Exhaustive Search using Greedy

Example :-

item	I_1	I_2	I_3	I_4
weight	7	3	4	5
value	₹42	₹12	₹40	₹25

$W = 10$

Subset	total weight	Total value
$\{I1\}$	7	42
$\{I2\}$	3	12
$\{I3\}$	4	40
$\{I4\}$	5	25
$\{I_1, I_2\}$	10	54
$\{I_2, I_3\}$	7	52
$\{I_3, I_4\}$ ✓	9	65 ✓
$\{I_1, I_3\}$	11 X	
$\{I_1, I_4\}$	12 X	
$\{I_2, I_4\}$	8	37
$\{I_1, I_2, I_3\}$	14 X	
$\{I_1, I_2, I_4\}$	15 X	
$\{I_1, I_3, I_4\}$	16 X	
$\{I_2, I_3, I_4\}$	12 X	
$\{I_1, I_2, I_3, I_4\}$	19 X	

$\boxed{I_3, I_4}$ → 65 Maximum

Assignment Problem - Exhaustive Search :-

Ex:-

	Job 1	Job 2	Job 3
Person 1	9	2	7
Person 2	6	4	3
Person 3	5	8	1

All possible combination first person first job second pers second

1 2 3 → 9 + 4 + 1 = 14
1 3 2 → 9 + 3 + 8 = 20
2 1 3 → 2 + 6 + 1 = 9
2 3 1 → 2 + 3 + 5 = 10
3 2 1 → 7 + 4 + 5 = 16
3 1 2 → 7 + 6 + 8 = 21

Optimal → Minimal time take

9 → person 1 → Job 2
person 2 → Job 1
person 3 → Job 3

Greedy Algorithm :-

A greedy algorithm is a problem solving approach like subtract and conquer, divide and conquer and dynamic programming, which is used for solving optimality problem (one solution), out of all feasible solution.

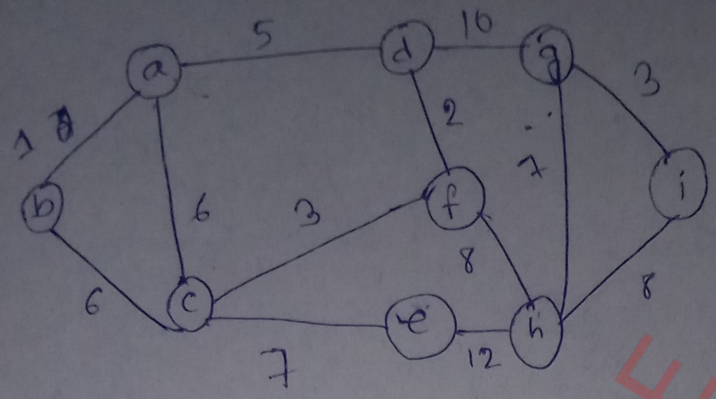
- Minimum Spanning Tree
- Single source Shortest path
- Huffman coding
- Knap Sack problem
- optimal merge pattern

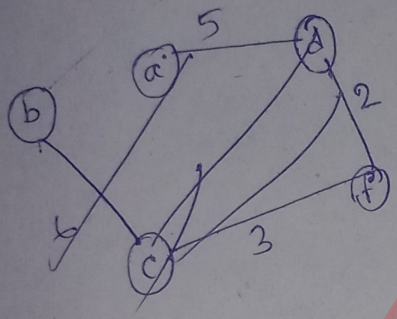
Prims algorithm :-

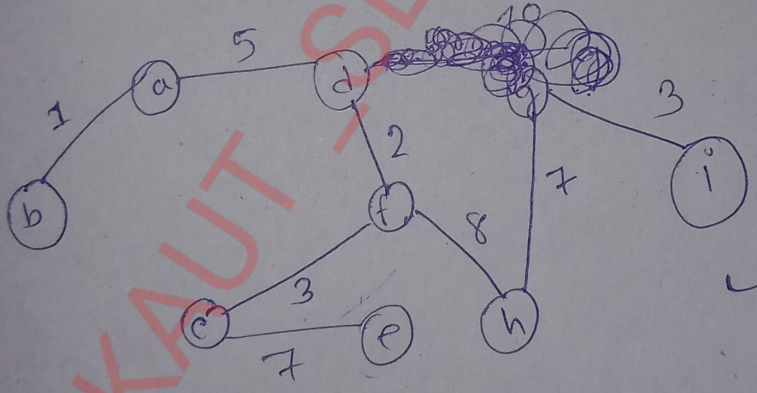
Minimum Spanning tree

vertex V.

O/P
= No. of edges
= (V-1)



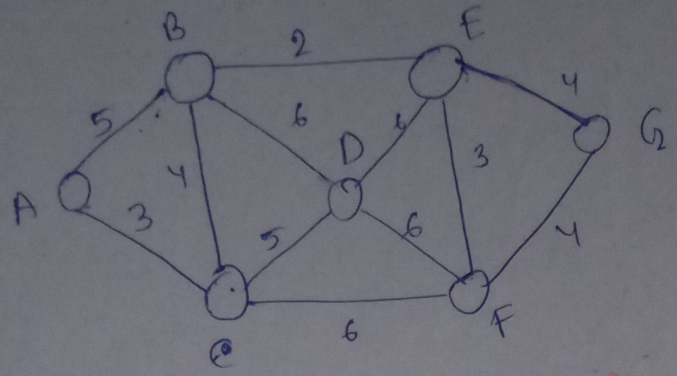




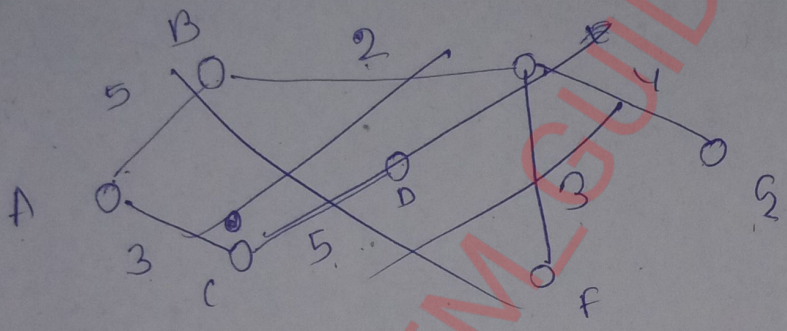
edges = 8

Total cost =

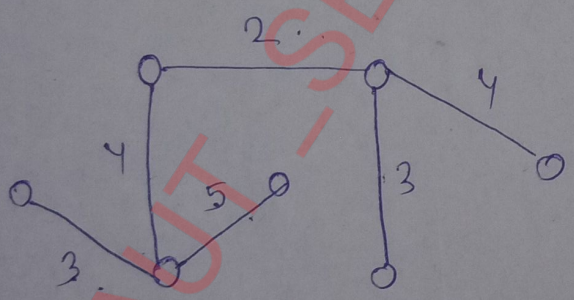
Kruskal Algo :-



$v = 7$



no of edges
$= n-1$
$= 7-1$
$= 6$

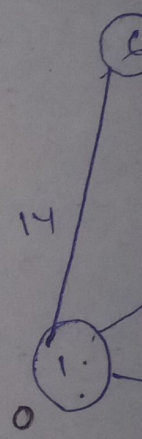


$(n-1)$
$(\log n)$
n

$(n-1) \log e$

$O(n \log e)$

cost (21)



Source
1

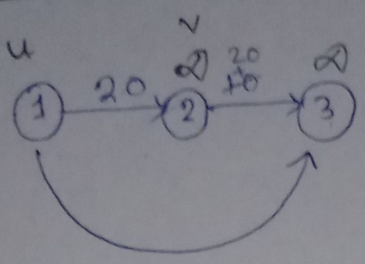
1; 2

1; 2, 3

1; 2, 3

1; 2, 3, 6

Dijkstra's Algorithm :-

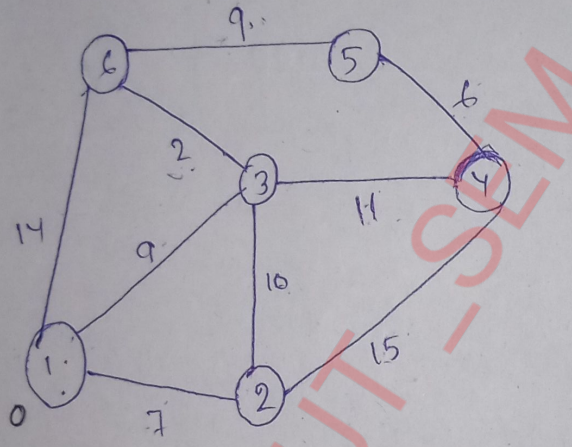


* Relaxion

if $d(u) + c(u,v) < d(v)$

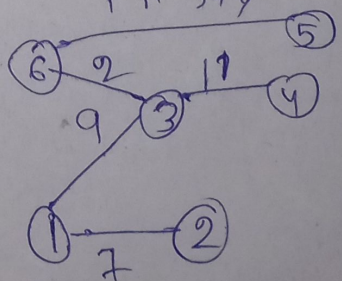
$d(v) = d(u) + c(u,v)$

$d(u) + c(u,v)$	$d(v)$
$0 + 20$	$< \infty$
$20 + 10$	$< \infty$
30	$< \infty$
$0 + 40$	$< \infty$
40	$< \infty$

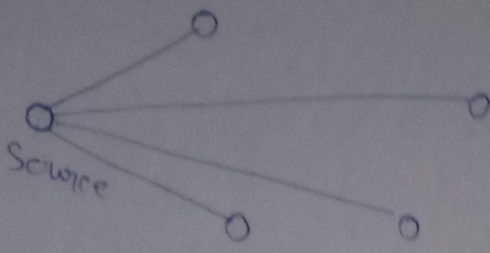


Source	Destination					
1	2	3	4	5	6	
	∞	∞	∞	∞	∞	(initially)
	(7)	9	∞	2	14	
1,2	(7)	(9)	22	2	14	
1,2,3	(7)	(9)	20	2	(11)	
1,2,3,6	(7)	(9)	(20)	2	(11)	
1,2,3,6,5	(7)	(9)	(20)	(20)	(11)	

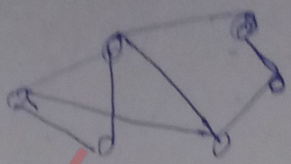
1 to 2 } initially
1 to 3 } ∞
1 to 2 → 7
1 " 3 → 9



All Pair Shortest Path (Floyd Warshall Algo)



$E \log v$



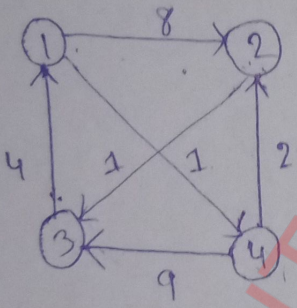
$V E \log v$

$V E \log v$

$V \cdot V^2 \log v$

$V^3 \log v$

Floyd warshall working with example:



1 to 1, 1 to 2

2 to 3

2 to 1, 1 to 3

via — all min distance calculated

$D^0 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$D^1 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

4-1
1-3

$D^2 =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

3-4
3-2 2-4

1-2
2-1 3-1-4
3-2-

4-1

$D^3 =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

2-4

2-3, 3-4

3-2-1-4

D^1, D^2, D^3 ... same

3rd row and 3rd col

Same formula for 2nd

2-1

2-3, 3-1 1-2

1+4 1-3, 3-2

=5

2-4 1-4

2-3, 3-4 1-3, 3-4

1+5 (4+2)

=6

4-1

4-3, 3-1

9+4 / 3+4

=7

1-2

1-4, 4-2

3-2

3-4 4-2

1-3

1-4, 4-3

1+3=4

2-1

2-4, 4-1

=

3-2 = 12

3-4 4-2

5+2 = 7

$D^4 =$

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

Time Complexity

$O(n^3)$

Huffman coding :-

(Greedy Technique)

a = 50
b = 10 ie 3
c = 30 , f = 2
d = 5 3

M = 100 charectas

ASCII

0 - 127 (7_{bit})

↓

1111111

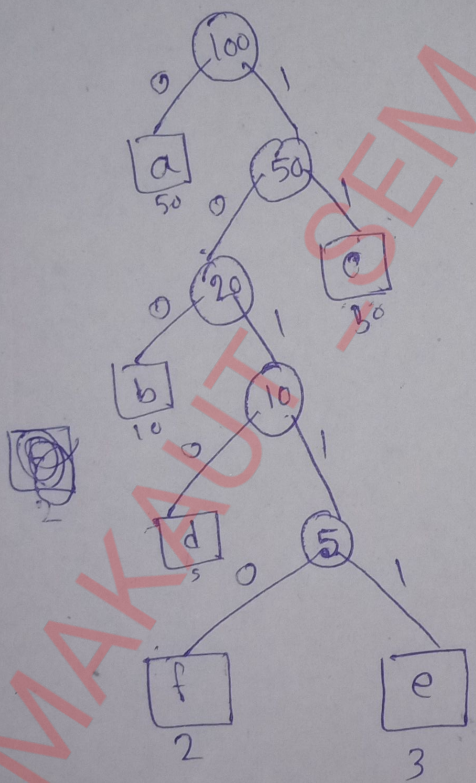
7 x 100 = 700 bits

If we know that there are only 6 characters in my message.
a, b, c, d, e, f
(6 type char)

a = 000
b = 001
c = 010
d = 011
e = 100
f = 101

3 x 100 = 300 bits

2 | 127
2 | 63 - 1
2 | 31 - 1
2 | 15 - 1
2 | 7 - 1
2 | 3 - 1
2 | 1 - 1
0 - 1



a = 0 50 x 1 = 50
b = 100 10 x 3 = 30
c = 11 30 x 2 = 60
d = 1010 5 x 4 = 40
e = 10111 3 x 5 = 15
f = 10110 2 x 5 = 10

185 bits

Avg bits required to represent each char

$\frac{185}{100}$

= 1.85 bits / char

~~Trans bits~~

MT T↑

M↓ T↓

Consider the following message

aa bbbb a bbb ccc ddd eee ccc eee dd eee

Find the no. of bits required for Huffman encoding of above message also find the avg bits required to represent a character?

a = 97 (ASCII value)

7 bit

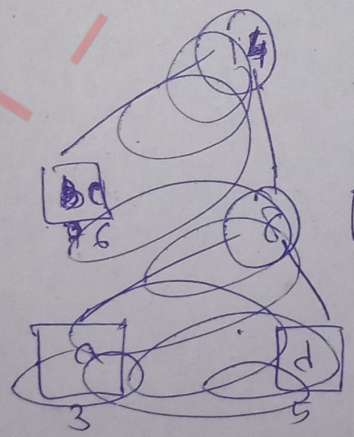
30 × 7 = 210

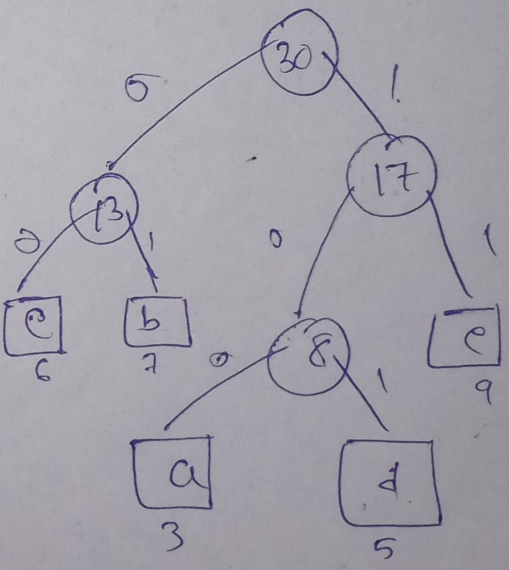
a = 3
b = 7
c = 6
d = 5
e = 9

2 | 97
2 | 48 - 1
2 | 24 - 0
2 | 12 - 0
2 | 6 - 0
2 | 3 - 0
2 | 1 - 1
0 - 1

(3) 7 6 5 9
a b c d e
min

a = 100 · 3×3 =
b = 01 7×2 =
c = 00 6×2 =
d = 101 5×3 =
e = 11 9×2 =
68



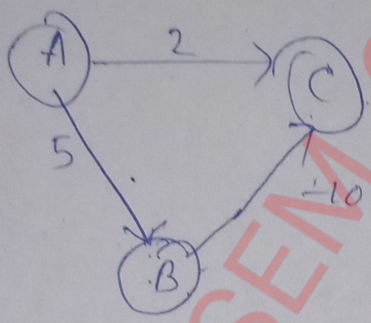


Huffman graph

Avg = 68/30 = 2.26

Why does Dijkstra fail on Negative weights?

It happens because in each iteration, the algorithm only updates the answer for the nodes in the queue. So, Dijkstra's algorithm does not reconsider a node once it marks it as visited even if a shortest path exists than the previous one. Hence, Dijkstra's algorithm fails ~~on~~ in graphs with negetive edge weights



	A	B	C
		∞	∞
		5	2
A, C		5	2
A, C, B			

Dynamic programming

It divide the problem into series of overlapping sub-problems.

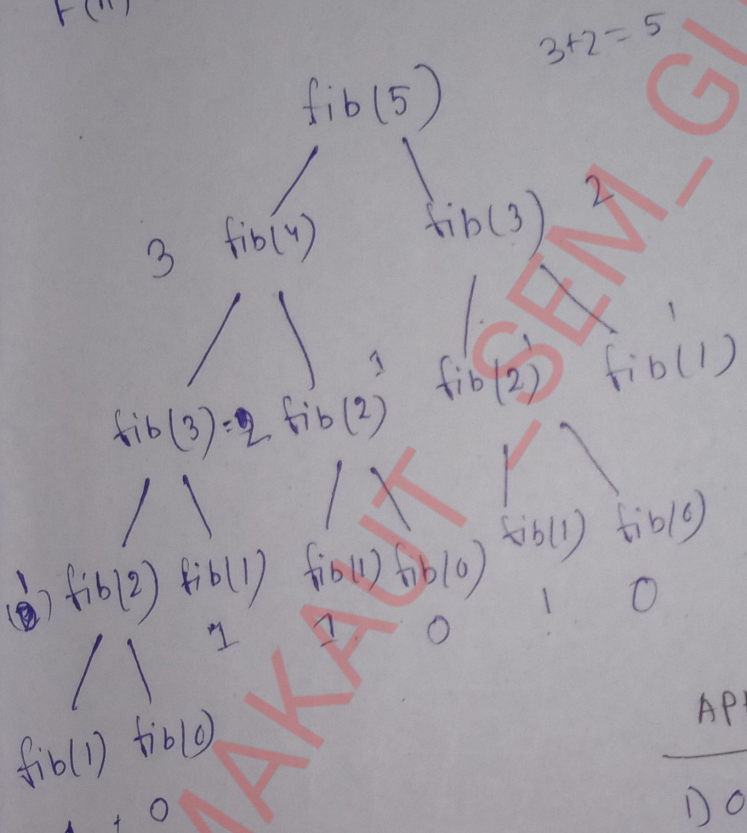
Two features :- 1) optimal substructure
2) overlapping subproblems

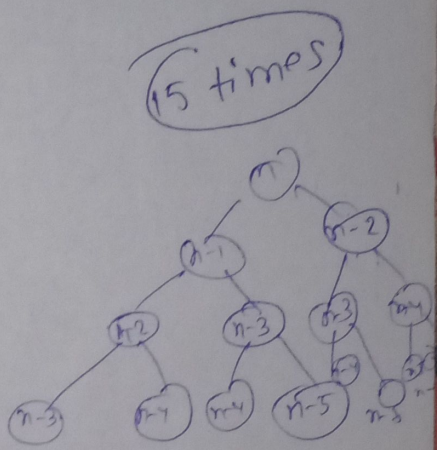
$F(0) = 0$

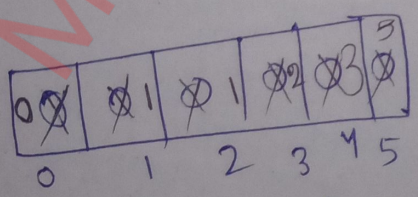
$F(1) = 1$

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

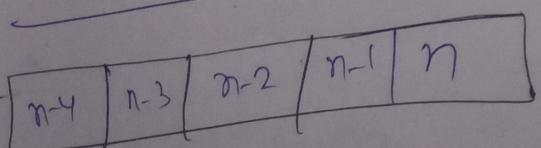
Concept of recursion is used mostly.







Bottom up dynamic prog.



Applications :

1) optimal binary search tree
2) 0/1 knapsack problem.
3) All pairs shortest prob
4) TSP
5) Reliability design

Branch and Bound :-

- used to find optimum solutions
- Similar to backtracking
  ↓
  DFS is used

** Max profit - cost less

branch and bound - BFS is used.

branching process of generating subproblems.

State space tree - all possible paths.

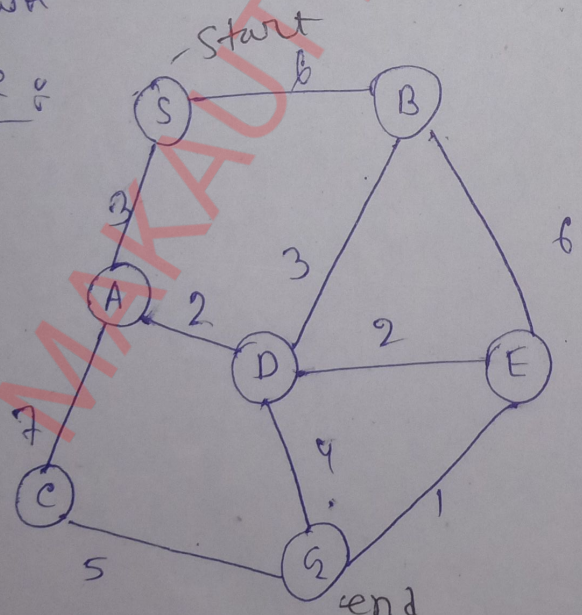
Among all these, we keep extending the cheapest path.

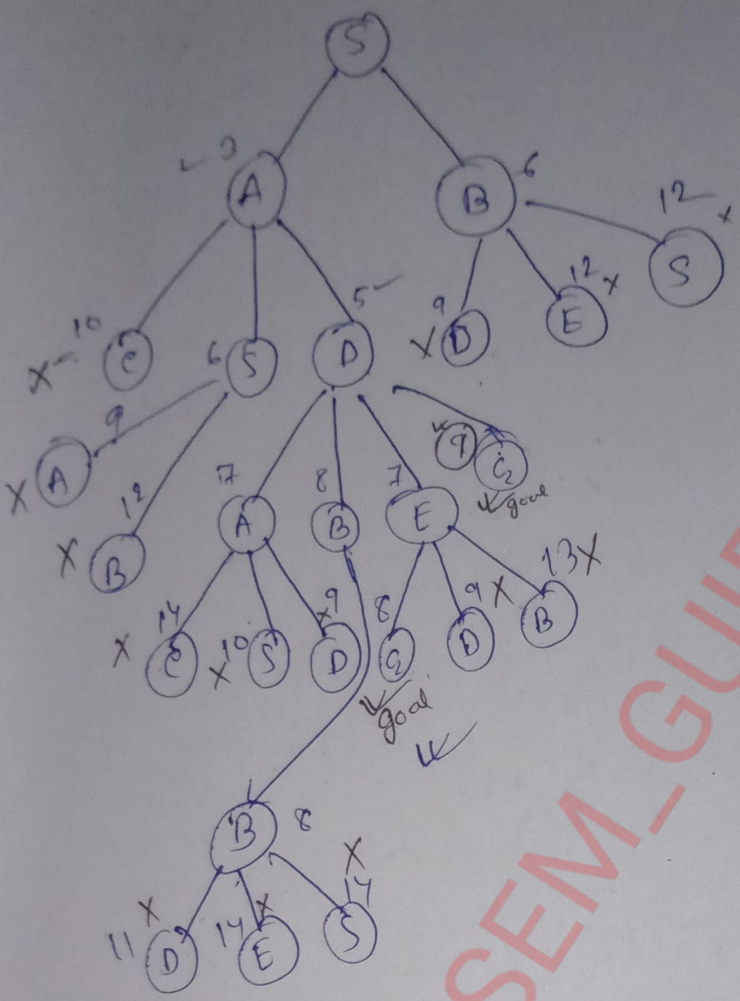
Application :

TSP

0/1 knapsack
↓ ↘
Least cost FIFO

Example :





1st it → A, B → (A)

2nd → C, S, D, B

3rd → ~~S~~, A, B, E, B

9 10 > 9

S → A → D → E → G_2

Backtracking :-

→ Problem solving technique

→ used for solving recursive problems

→ Multiple solutions, not a single solution.

going back and coming again until condition is satisfied.

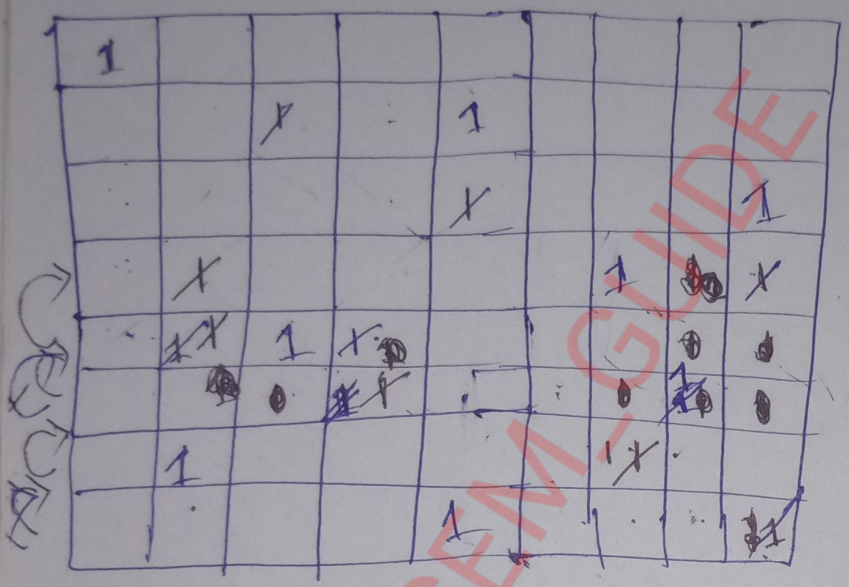
Application s:

N queens

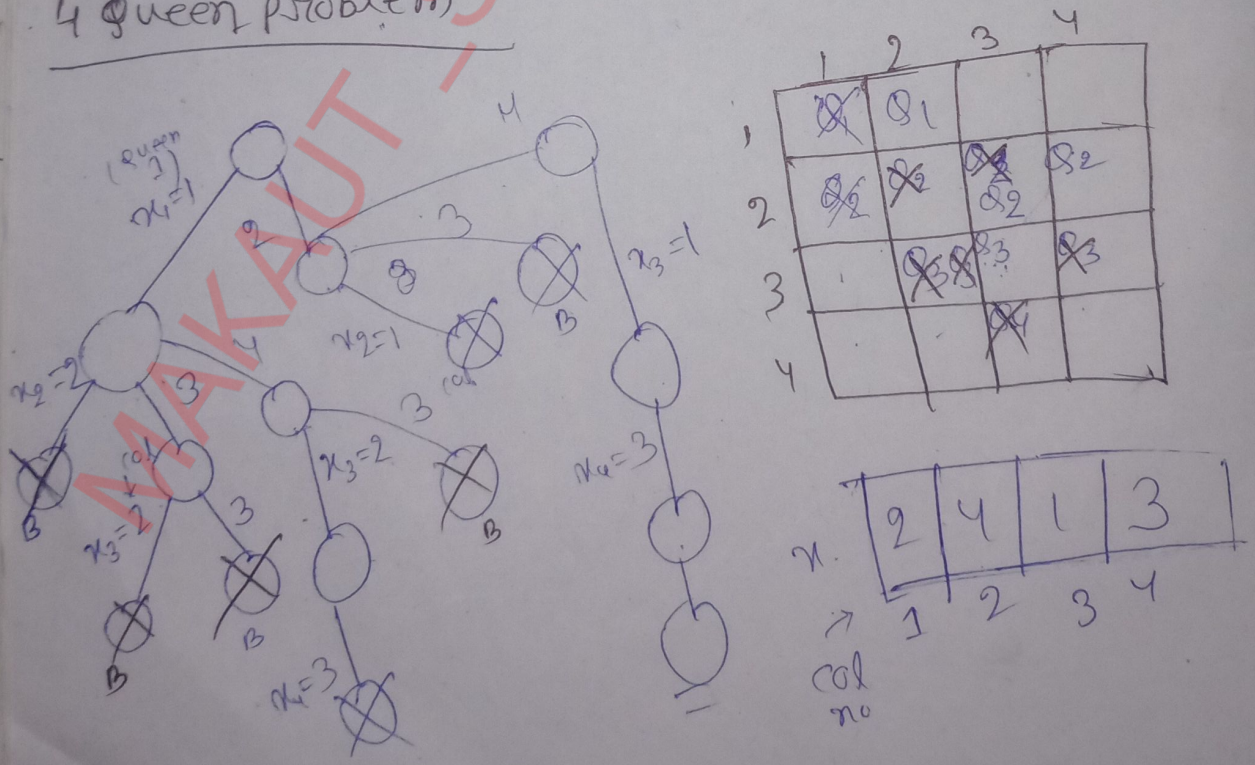
Sum of subsets

Graph coloring.

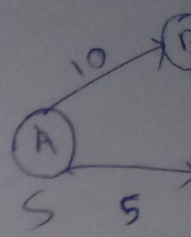
8 queen problem



4 queen problem



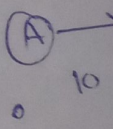
Bellman



Dijkstra will not there ar edges wo

result c have contin So we

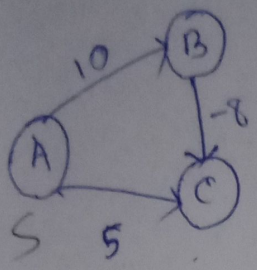
N -



Bellman Ford algorithm :

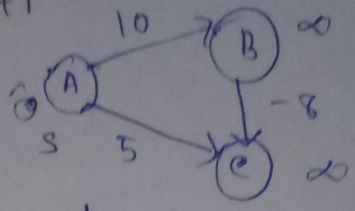
$(V-1) = 3-1 = 2$ বার করে relax করতে হবে

$(V-1)+1$

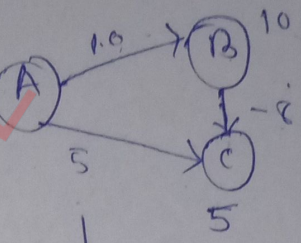


A	B	C
0	∞	∞
	10	5
A, C	10	5

A, C, B



↓ R_1



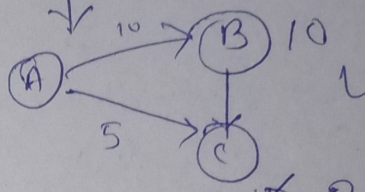
~~A~~ ~~B~~

(A, B) (B, C)

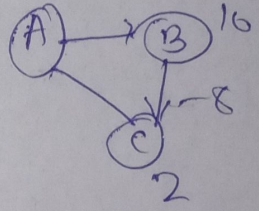
(A, C)

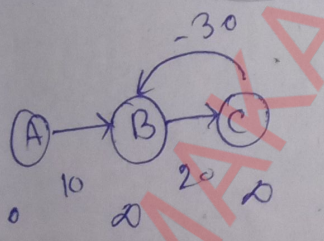
Dijkstra algorithm will not work if there are negative edges. it will not work properly. it may give correct result or it may not give correct result. So we cannot use Dijkstra algo here and confirm that it is giving correct answer. So we want a algorithm which confirms that the answers are correct. So that algorith is Beleman ford and it follow Dynamic programming strategy

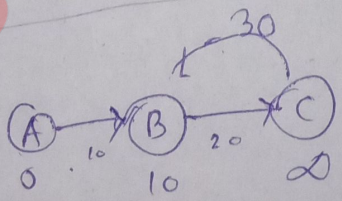
↓ R_2

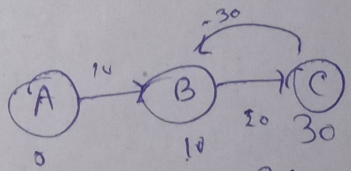


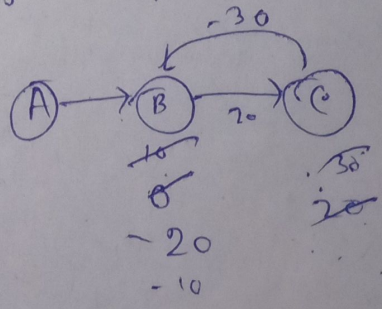
↓ R_3











Bellman Ford algorithm :

```
BellmanFord(G, V, E, S)
Step 1
for each vertex v in G do          } O(V)
    dist[v] = ∞
    dist[source] = 0
Step 2
for i = 1 to |V| - 1
    for each edge(u, v) ∈ G
        Relax(u, v, w)
Step 3
    for each edge(u, v) ∈ G
        if (dist[u] + w(u, v) < dist[v])
    return "Graph contain Negetive cycle"
return distance
```

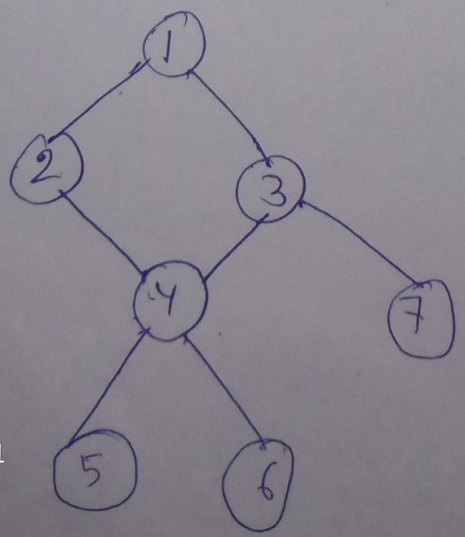
T.C : O(V·E)

Introduction to graph traversal :

The process of visiting and exploring a graph for processing is called graph traversal.

- Breadth first search (BFS) (queue)
- Depth " " (DFS) (stack)

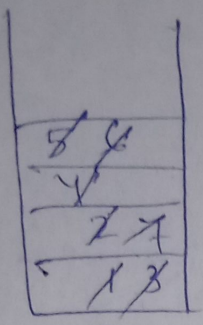
BFS :



1	2	3	4	7	5	6

1 2 3 4 7 5 6

DFS:-



1, 2, 4, 5, 6, 3, 7

$O(m^2)$

Time complexity :-

|E| No. of edges are there

(|V|-1) time it is relaxing

$O(|E||V|-1)$

$= O(|V||E|)$

$= O(n^2)$

Bellman ford algorithm :-

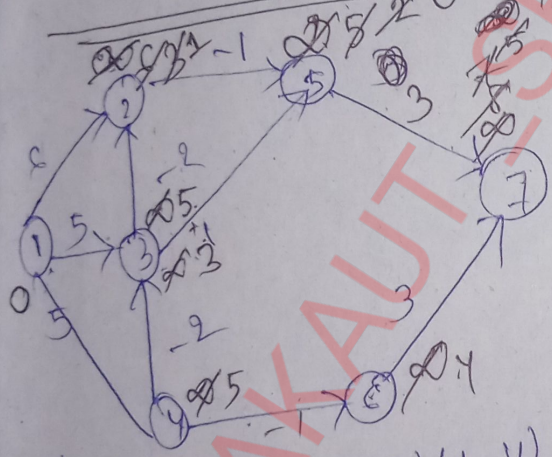
|V| = n = 7

(no. of vertices)

edges = |V| - 1 = 7 - 1 = 6

if (d[u] + c(u,v) < d[v])

d[v] = d[u] + c[u,v]



edge list (1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7)

0+6 < ∞

0+5 < ∞

0+5 < ∞

6-1 < ∞

5-2 < ∞

5+1 < ∞

5-2 < ∞

5-2 < ∞

5+3 <

2nd iteration :-

(2,5) → 3-1 = 2 < ∞

(3,2) → 3-2 = 1 < ∞

(5,7) → 3+2 = 5 < ∞

3rd iteration :-

1-1 = 0 < ∞

0+3 = 0 < ∞

4 iterations

vertex	dist a
1	0
2	1
3	3
4	5
5	0
6	4
7	3

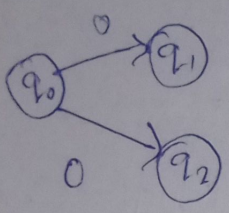
Com Preparation

NP Hard and NP complete :

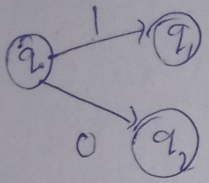
Non deterministic Algorithm : (NP)

O/P cannot be predicted properly even we know the i/p.

- Same i/p will give different o/p for diff rounds of execution.



Same i/p → 0, but diff o/p → q_1 & q_2
→ Non-deterministic algo



→ deterministic algo

Reduction :

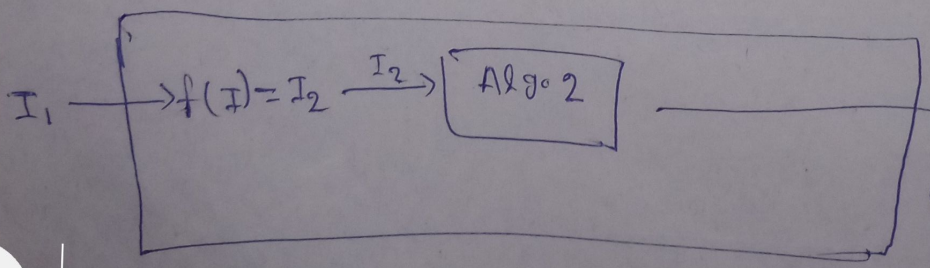
consider we have two decision problems P_1 & P_2

P_1 → Input(I_1)
→ Algorithm (A_1) - unknown

P_2 → Input(I_2)
→ Algorithm (A_2) - known

If P_1 can be solved with help of A_2, then we convert i/p I_1 into I_2 and find solution for P_2

→ P_1 is reducible to P_2

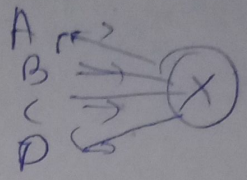


P_2 ↔ P_2 =

NP-hard :

A problem is called NP hard if every problem in NP can be polynomially reduced

A, B, C, D → NP Problem

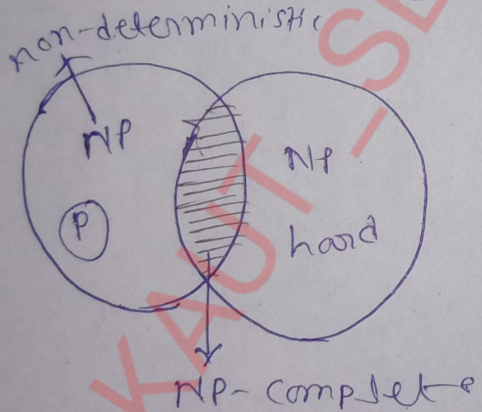


if all these can be reduced to X, it is called NP hard

NP complete :

A problem is called NP complete if the problem P is

- P is in NP
- P is in NP hard (reducible to X)



Com Preparation

Tail recursion

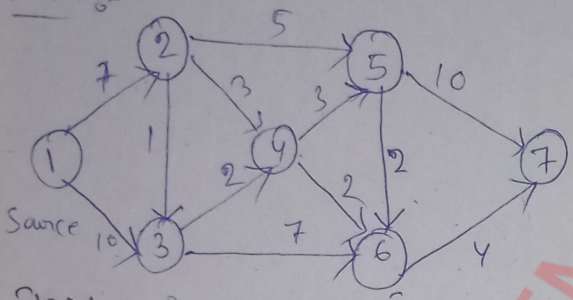
Ford Fulkerson Alg.

- It is used to find maximum flow.

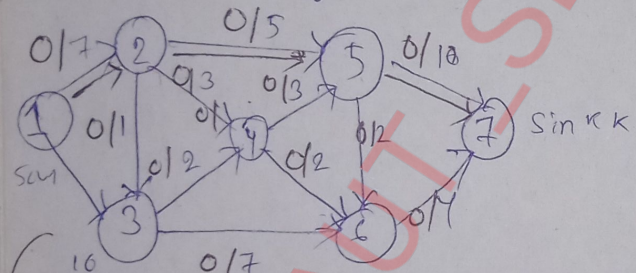
Terms used

1) Source, Sink
2) Bottleneck capacity (সবচেয়ে কম capacity যা)
3) Flow
4) Augmenting path
5) Residual capacity ((যেটা বের করে)

Exam:-



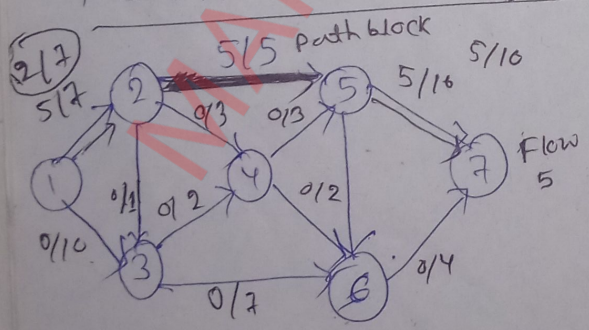
Step 1 :- initially Flow = 0



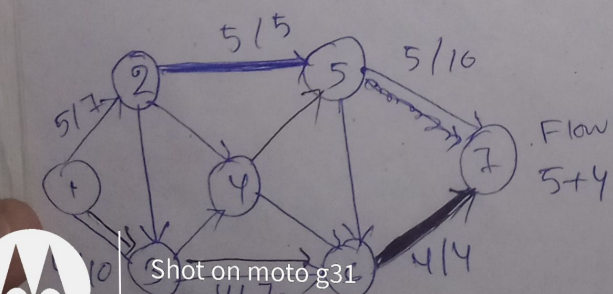
black on the movement path

Choose random path 1 → 7 (সবচেয়ে কম capacity যাবে)

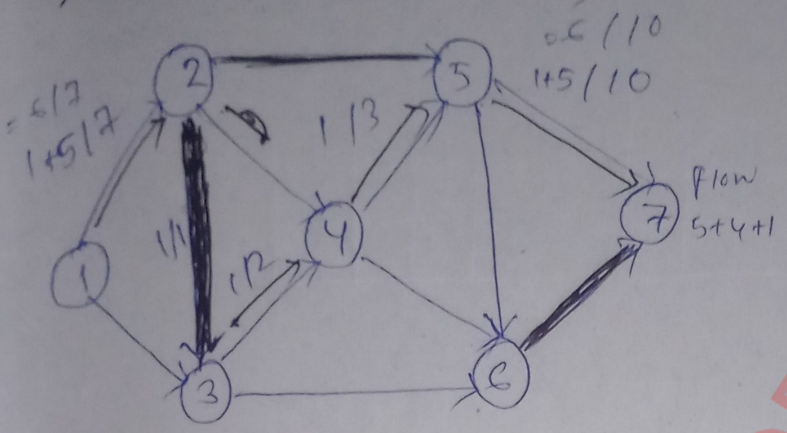
1) Bottleneck capacity fill :



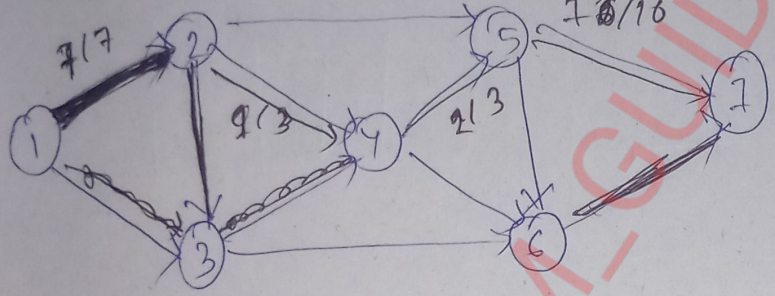
Augmenting path	Bottle neck capacity
1-2-5-7	5

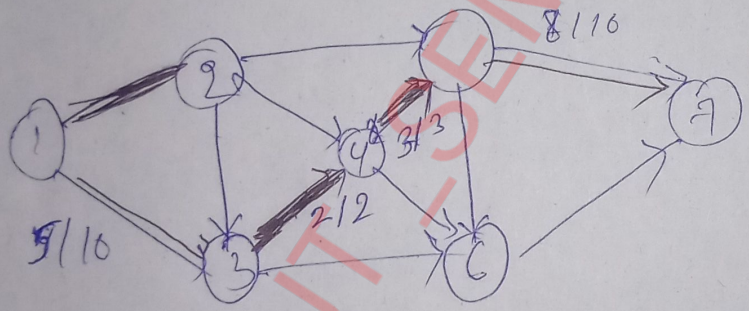


Matrix Chain Multiplication :-



Augmenting path	Bottleneck capacity
1-2-5-7	5
1-3-6-7	4
1-2-3-4-5-7	1
1-2-4-5-7	~~1~~
1-3-4-5-7	1





Flow
5+4+1+1+1
= 12

Max flow = 12

0/1 Knapsack Problem:

$M = 8$ → knapsack weight
$n = 4$ → item

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

$x_i = 0/1$
Example $x = \{1, 0, 0, \ldots\}$ → set

$\max \sum p_i x_i$

$\sum w_i x_i \le m$

0 0 0 0
1 1 1 1
0 0 0 1
1 0 0 0
1 1 0 0

2^4 2^n

$O(2^n)$

Tabulation method:

$m = 8$
$n = 4$

P	W (weight)	
		0
1	2	1
2	3	2
5	4	3
6	5	4

	0	1	2	3	4	5	6	7	8
0	0 (Profit)	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

x_1 x_2 x_3 x_4
$\{0\ 1\ 0\ 1\}$

Max profit = 8
$8 - 6 = 2$
8 এর respect w = 5

w = {2, 3, 4, 5} compare
{0, 1, 0, 1}

remain 2
2 এর respect এ weight 3
$2 - 2 = 0$

Profit
$8 - 6 = 2$

3 row তে 2 থাকলেও consider হবে না কারণ 2 2nd row তে আগে থেকে আছে।

0/1 knapsack problem using Set method

$m = 8$ $\quad p = \{1, 2, 5, 6\}$

$n = 4$ $\quad w = \{2, 3, 4, 5\}$

(p, w) → profit, weight

$S^0 = \{(0,0)\}$ ← No profit, No weight

set 0

$S_1^0 = \{(1,2)\}$

$S^1 = \{(0,0), (1,2)\}$

$S_1^1 = \{(2,3), (3,5)\}$

$S^2 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S_1^2 = \{(5,4), (6,6), (7,7), (8,9)\}$ exceeding capacity

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7)\}$

$S_1^3 = \{(6,5), (7,7), (8,8), (11,9), (12,11), (13,12)\}$ ← profit

$S^4 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (7,7), (6,5), (7,7), (8,8)\}$

① $(8,8) \in S^4$ but $(8,8) \notin S^3$ $\therefore x_4 = 1$

$(8-6, 8-5) = (2,3)$ ← prof

② $(2,3) \in S^3$ $\therefore x_3 = 0$

$(2,3) \in S^3$

⑤ $(2,2) \in S^2$ but $(2,3) \notin S^1$ $\therefore x_2 = 1$

$(2-2, 3-3) = (0,0)$

④ $(0,0) \in S^1$; $x_1 = 0$

$(0,0) \in S^0$

$\{0, 1, 0, 1\}$

ANS

Red Black Tree :

Red black tree is a Binary Search tree with extra bit of storage per node

LP	color	key	RT

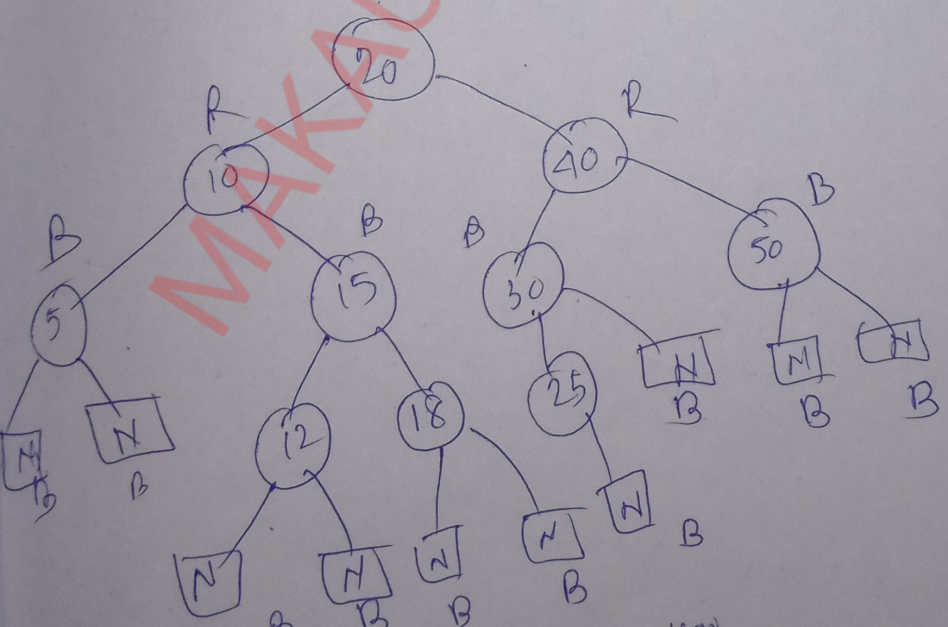
B = 0
Red = 1

Property :-

1) Every node is either Red or Black.
2) The root is black.
3) Every Leaf (NIL) is black. (Black)
4) If a node is Red, then both its children are black.

No R-R parent-child relation

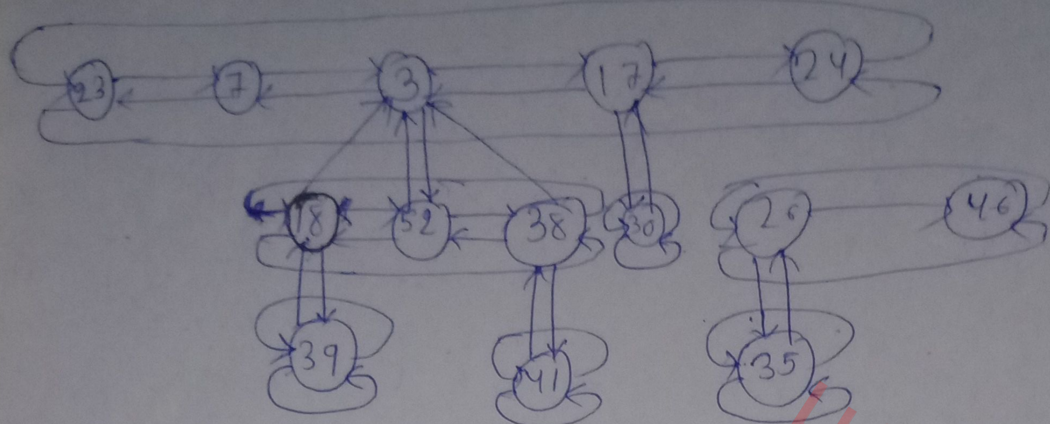
5) For each node, all paths from node to descendant leaves contain the same no. of black nodes.



Mean heap fo daw unsorted arrange possible

Fibonacci Heap :-

doubly circular linked list



Min heap property follow করে।

p[x], degree[x], mark[x]
left[x], right[x], child[x]

* Fibonacci heap এ binomial tree হতে পারে, না ও পারে,

P(X) হলো কোনো element এর parent কে বোঝায়,

P(38) = 3 (upward arrow)

P(3) = Null

(random) * unsorted arrangement possible

deg(18) = 1

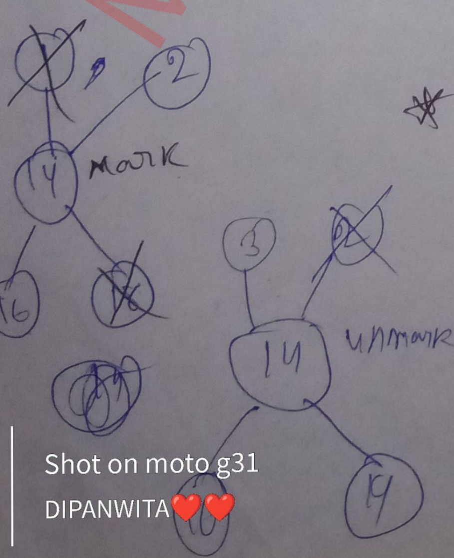
deg(3) = 3

left(52) = 18 | left(38) = 52

right(52) = 38 | right(38) = 18

Child(3) = 52 (3 only 52 কে point করছে [arrow 52 তেই]

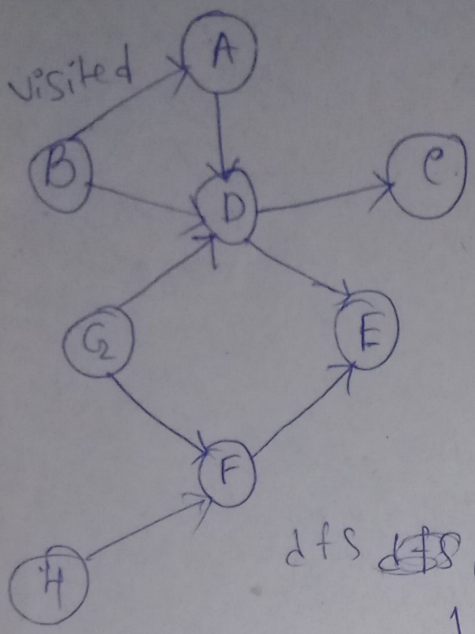
Mark(x) T/F

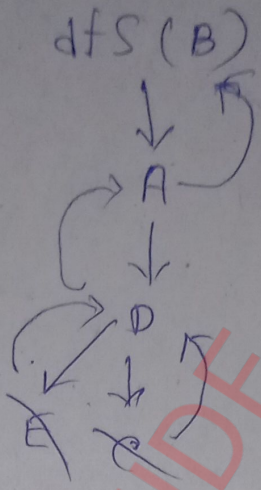


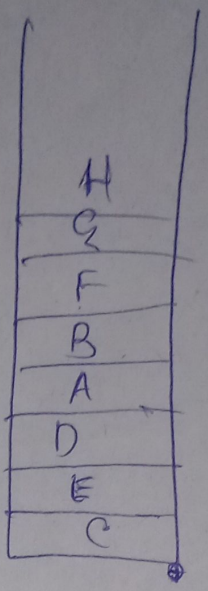
1st
* delete parent
14 এর parent 1 (deleted)
14 এর নতুনকে adopt করবে 2
এর delete করা হবে.
16, 18 এর কাছে

Topological Sort

LIFO







dfs ~~dfs~~ (G2)

dfs (H)

H G2 F B A D E C

TC :-

$O(V+E)$

Matrix Chain Multiplication :-

We are given sequence of matrices find the most efficient way to multiply these matrices together.

$\langle A_1, A_2, A_3 \rangle$

$2 \times 3 \quad 3 \times 2 \quad 2 \times 3$

2×2

$2 \times 2 \quad 2 \times 3$

2×3

$\langle A_1 \cdot A_2 \cdot A_3 \rangle = ((A_1 \cdot A_2) \cdot A_3) = (A_1 \cdot (A_2 \cdot A_3))$

$= =$

$A_1 \cdot A_2 \cdot A_3 \cdot A_4 = 5$ Process

1. $(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$
2. $((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$
3. $(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$
4. $(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$
5. $(A_1 \cdot (A_2 \cdot ((A_3 \cdot A_4))$

Time complexity

$\frac{1}{n} \times {}^{2(n-1)}C_{n-1}$

$= \frac{1}{3} \times 2(3-1) C_2$

$= \frac{1}{3} {}^4C_2$

$= \frac{1}{3} \times \frac{4!}{2!(4-2)!}$

$= \frac{1}{3} \times \frac{4 \times 3 \times 2!}{2! \cdot 2!}$

$= \frac{4}{2}$ 2

$= 2$

Solve

$\frac{1}{4} \times 2 \cdot 3 C_3$

$= \frac{1}{4} {}^6C_3$

$= \frac{1}{4} \frac{6!}{3!\,3!}$

$= \frac{1}{4} \times \frac{6 \times 5 \times 4 \times 3!}{3! \cdot 3!}$

$[\ ]_{2\times 2} \underbrace{[\ ]_{2\times 2}}_{p \quad q} = \frac{6 \times 5 \times 4}{4 \times 3 \times 2 \times 1}$

$r = 5$

$p \cdot q \cdot r = 2 \times 2 \times 2$

$= 8$

Example

$A_1 = 10 \times 100$ (p, q) $= 10 \times 5$

$A_2 = 100 \times 5$ $= 100 \times 50$

$A_3 = 5 \times 50$ $= 10 \times 50$

$\langle A_1, A_2, A_3 \rangle =$

Find out best parenthesis

$((A_1 \times A_2) \cdot A_3) = (10 \times 100 \times 5) + (10 \times 5 \times 50)$

$= 5000 + 2500 = 7500$ (multiplications)

$(A_1 \cdot (A_2 \times A_3)) = 100 \times 5 \times 50$ $((100 \times 5 \times 50))$

$= ((10 \times 100 \times 50) + 25000)$

$= 50000 + 25000$

$= 75000$

Matrix chain Multiplication:-

Consider following four matrices find optimal parenthesization of matrix chain Multiplication

Matrix	order
A_1	20×30
A_2	30×50
A_3	50×10
A_4	10×5

$\langle A_1, A_2, A_3, A_4 \rangle$

1. $(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$
2. $((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$
3. $(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$
4. $((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$
5. $(A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$

Find out Optimal Solⁿ

$\frac{1}{n} {}^{2(n-1)}C_{n-1}$

$= \frac{1}{4} {}^{6}C_3$

$= 5$

Ordered matrix chain multiplication

$P\langle 20, 30, 50, 10, 5 \rangle$

$P_0 \; P_1 \; P_2 \; P_3 \; P_4$

cost matrix M →

i \ j	1	2	3	4
1	0	30000	21000	13000
2		0	15000	10000
3			0	2500
4				0

Split Marker Matrix S

i \ j	1	2	3	4
1		1	1	1
2			2	2
3				3
4				

① 1-4
② 2-4
③ 3-4

$A_1 \times A_2 \times A_3 \times A_4$

$= 20 \times 30 \times 50 \times 10 \times 10 \times 5$

$P_0 \quad P_1 \quad P_2 \quad P_3 \quad P_4$

$= 20 \times 50 \times 50 \times 10 \times 10 \times 5$

$= 20 \times 10 \times 10 \times 5$

$= 20 \times 5$

$= 20 \times 10 \times 5$

Diagonal

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \}$$

$$s[i,j] = k$$

$P\langle 20, 30, 50, 10, 5 \rangle$

$P_0 \; P_1 \; P_2 \; P_3 \; P_4$

$i \; j$

$$m[1,2] = \min_{1 \le k < 2} \{ m[1,1] + m[2,2] + p_0 p_1 p_2 \}$$

$$= \{ 0 + 0 + 20 \times 30 \times 50 \}$$

$$= \{ 30000 \}$$

$m[2,3] = \min_{i \le k < j} \{ m[2,2] + m[3,3] + P_1 P_2 P_3 \}$

$= 0 + 0 + (30 \times 50 \times 10)$

$k=2 \quad = 15000$

$m[3,4] = \min_{i \le k < j} \{ m[3,3] + m[4,4] + P_2 P_3 P_4 \}$

$k=3 \quad = 0 + 0 + 2500$

$= 2500$

$m[1,3] = \min_{i \le k < j} \{ m[1,1] + m[2,3] + P_0 P_1 P_3$

$k=1 \quad 1 \quad 3 \quad 0 + 15000 + 600$

$k=2 \quad m[1,2] + m[3,3] + P_0 P_2 P_3$

$30000 + 0 + 1000$

$= \{ 21000 \leftarrow \text{Min}$

40000

$m[2,4] = \min_{i \le k < j} \{ m[2,2] + m[3,4] + P_1 P_2 P_4$

$2 \quad 4 \quad 0 + 2500 + 7500$

$k=2,3 \quad m[2,3] + m[4,4] + P_1 P_2 P_3$

$15000 + 0 + 1500$

$= \{ 10000$

$16,500$

$m[1,4] = \min \{ m[1,1] + m[2,4] + P_0 P_1 P_4$

$0 \quad 10000 + 3000 = 13000$

$m[1,2] + m[3,4] + P_0 P_1 P_4$

30000

$m[1,3] + m[4,4], P_0 P_3 P_4$

$k=1$

$k=2$

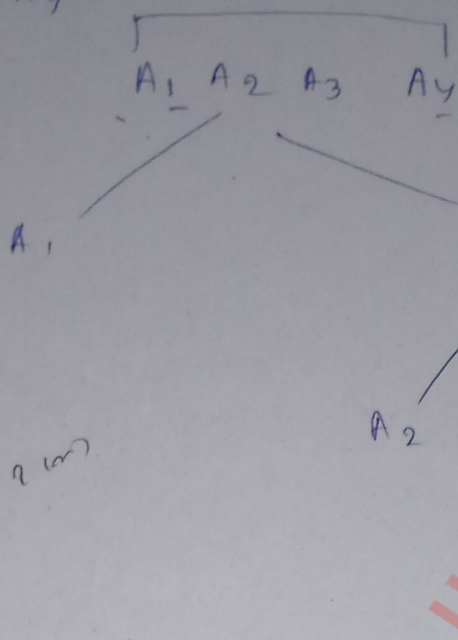
$k=3$

$= \{ 0 + 10000 + 3000 = 13000$

$30000 + 2500 + 5000 = 37500$

$21000 + 0 + 10000 = 22000$

$A_1\ A_2\ A_3\ A_4$



2 to 4
= 2 bt
split 2 ans

$\Rightarrow (A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$ ← ANS

optimal soln

~~(50×5)~~

~~(50×10×5)~~

~~7500~~ + 2500

= $(20\times30\times5 + (30\times50\times5 + (50\times10\times5)))$

30 × ~~50~~
~~50 × 5~~

30 × 50 × 5

= ~~45~~

= 7500

20 × 30

30 × 5

30 × 50
50 × 5 $A_3 A_4$

30 × 50 × 5

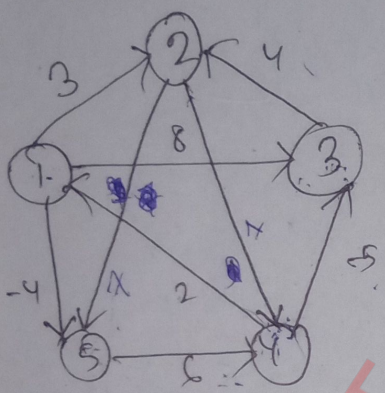
20 × 30

30 × 5

20 × 30 × 5

All pair Shortest Path (Floyd warshall Algo)

```
FLOYD_WARSHALL(w)
1. n ← rows[w]                                   n
2. D^0 ← w                                       n
3) For k ← 1 to n                                n
4) do for i ← 1 to n  (row)
5)     do for j ← 1 to n (col)
6)         do d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
                                   T(n) = O(n^3)
7) return D^n
```



$D^0 =$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^1 =$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^2 =$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^3 =$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

$D^4 =$

	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^5 =$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

12) Find the minimum number of operations required for the following matrix chain multiplication using dynamic programming.

A(10×20) * B(20×50) * C(50×1) * D(1×100)

Matrix	order
A	10×20
B	20×50
C	50×1
D	1×100

$P\langle 10, 20, 50, 1, 100\rangle$

$P_0\ P_1\ P_2\ P_3\ P_4$

$B \times C = \begin{matrix}10\times20\\20\times1\end{matrix} = \begin{matrix}10\times1\\1\times50\end{matrix}$

Cost matrix j →

i ↓	1	2	3	4
1	0	10000	1200	2200
2		0	1000	3080
3			0	5000
4				0

Split Marker matrix S

i ↓	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1, j] + P_{i-1} P_k P_j \}$$

$S[i,j] = k$

$$m[1,2] = \min_{1 \le k < 2} \{ m[1,1] + m[2,2] + P_0 P_1 P_2 \}$$

$k = 1$

$$= \{ 0 + 0 + 10 \times 20 \times 50 \}$$

$$= 10000$$

$$m[2,3] = \min_{2 \le k < 3} \{ m[2,2] + m[3,3] + P_1 P_2 P_3 \}$$

$k = 2$

$$= 0 + 0 + (20 \times 50)$$

$$= 1000$$

$$m[3,4] = \min_{3 \le k < 4} \{ m[3,3] + m[4,4] + P_2 P_3 P_4 \}$$

$k = 3$

$$= \{ 0 + 0 + (50 \times 100) \}$$

$$= 5000$$

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

$k = 1, 2$

$$= \begin{cases} 0 + 1000 + 10 \times 20 \\ 10000 + 0 + 10 \times 50 \end{cases}$$

$$= \begin{cases} 1000 + 200 \\ 10000 + 500 \end{cases}$$

$$= \begin{cases} 1200 \\ 10500 \end{cases}$$

$$m[2,4] = \min_{2 \le k < 4} \begin{cases} m[2,2] + m[3,4] + P_1 P_2 P_4 \\ m[2,3] + m[4,4] + P_1 P_3 P_4 \end{cases}$$

$k = 2, 3$

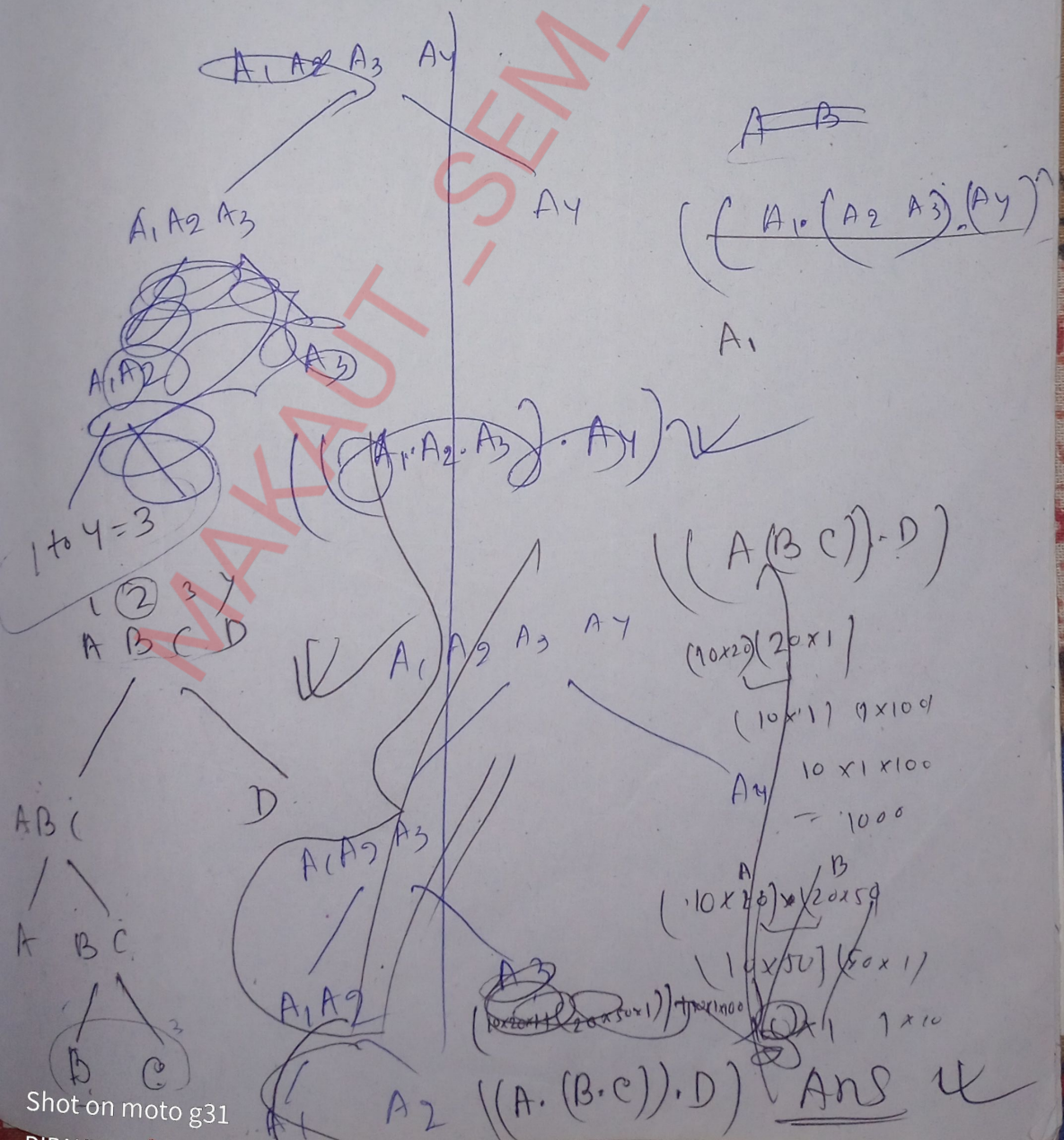
$$= \begin{cases} 0 + 5000 + 20 \times 50 \times 100 \\ 1000 + 0 + 20 \times 100 \\ 5000 + 100000 = 105000 \\ 1000 + 2000 = 3000 \end{cases}$$

$$m[1,4] = \min_{1 \le k < 4} \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \end{cases}$$

$k = 1, 2, 3$

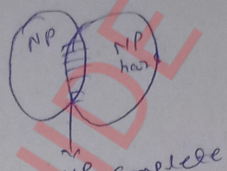
$$= \begin{cases} 0 + 3000 + (10 \times 100 \times 20) \\ 10000 + 5000 + (10 \times 50 \times 100) \\ 1200 + 0 + (10 \times 1 \times 100) \end{cases}$$

$$= \begin{cases} 3000 + 20000 = 23000 \\ 10000 + 5000 + 50000 = 65000 \\ 1200 + 1000 = 2200 \end{cases}$$



[3] write a short note on Approximation Algorithm.

An approximation algorithm is way of dealing with NP-completeness for optimization problem. The goal of approximation Algo is come close as possible to optimal solution in polynomial time



C → cost of solution

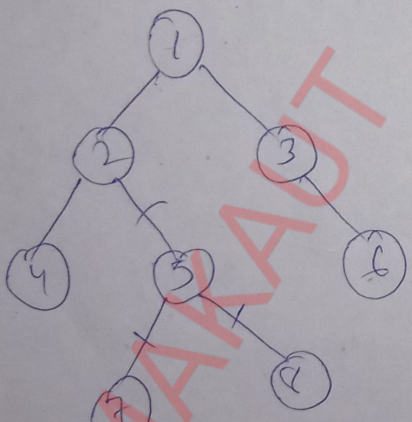
C* → cost of optimal solution

$\rho(n)$ → approximation Ratio

n → input size

1. maximization : $\frac{C^*}{C} \leq \rho(n)$

2) minimization : $\frac{C}{C^*} \leq \rho(n)$ $\rho(n) \neq 1$



$= \{1, 3, 2, 5\}$

No. of vertex $= \{5, 6, 2, 1\}$

$= \{2, 3, 5\}$

choose minimum vertex cover

Algo :- APPROX-COVER-Problem (G_2)

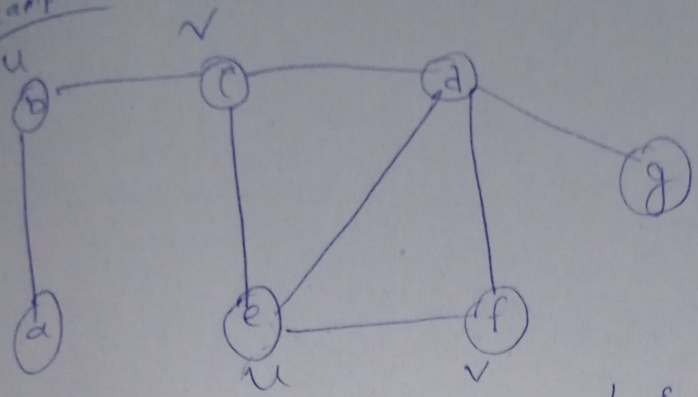
1) $C \leftarrow \phi$
2) $E' \leftarrow E[G_2]$ { # edge of graph }
3) while $E' \neq \phi$
4) do let (u, v) an arbitrary edge in E'
5) $C \leftarrow C \cup \{u, v\}$

Remove From E' every edge incident either u or v.

$\phi \cup \{b, c\}$
$\{b, c\}$

6) Return.

Example



$E' = \{(u,b), (b,c), (c,d), \ldots\}$

$C = \phi$

$= C \cup \{b,c\}$

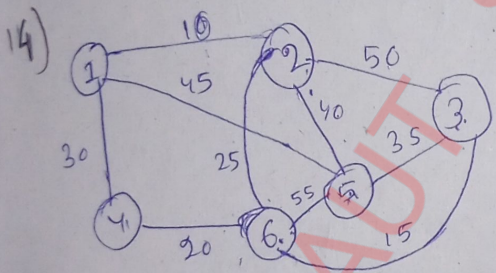
$= \{b,c\} \cup \{e,f\}$

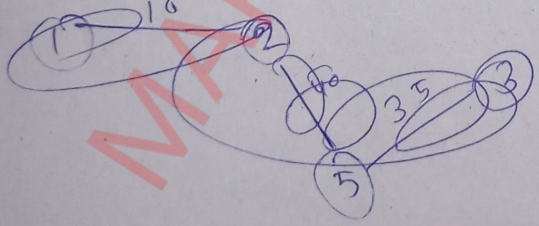
$= \{b,c,e,f\} \cup \{d,g\}$

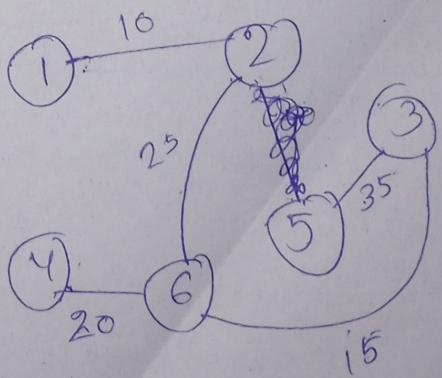
$= \{b,c,e,f,d,g\}$

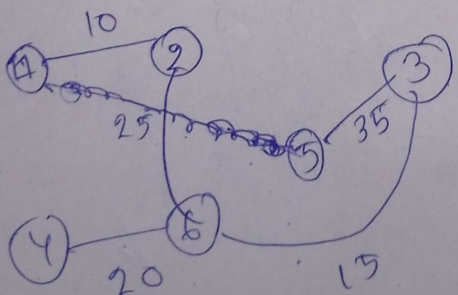
No of edges $V-1$.

14) using Prims algorithm







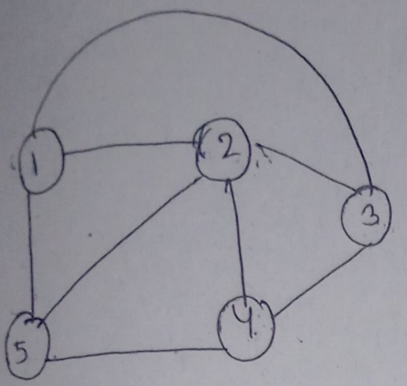


MST = 10+15+20+25+35

= 110

Hamiltonian cycles

The problem is find the hamiltonian cycle in a graph or not



$$G = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

↑ adjacency matrix

** Starting vertex টাকে fixed ধরতে হবে,

** using this array finding the cycle.

initially

	1	2	3	4	5
x	0	0	0	0	0

0, 1, 2, 3, 4, 5, 0

	1	2	3	4	5
x	1	2	3	4	5

	1	2	3	4	5
x	1	2	3	4	0

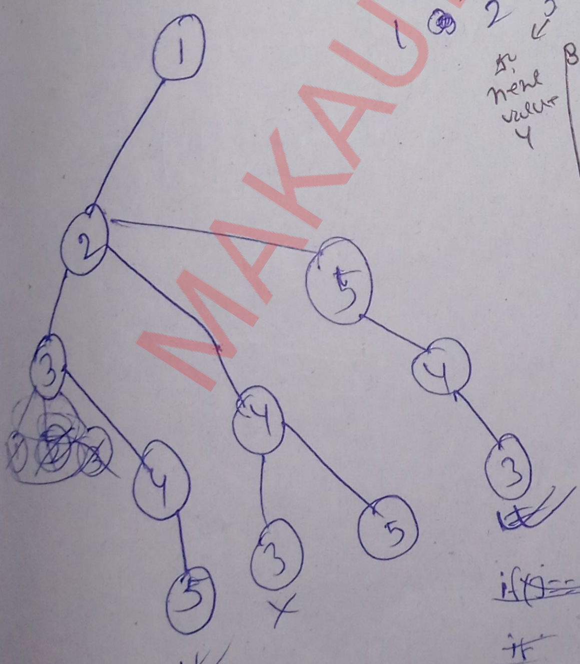
3 → next value 4

3-5 এর bond নেই 0

2-4=1

4-3=1

3-5=0



if(j==k)

1, 2, 3, 4, 5, → path

Algorithm Hamiltonian(k)

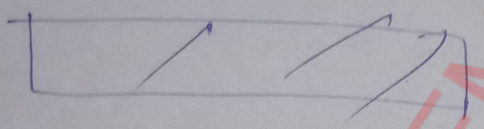
```
{
  do
  {
    nextvertex(k);
    if (x[k]==0)
      return;
    if(k==n)
      Print(x[1:n]);
    else
      Hamiltonian(k+1);
  } while(True);
}
```

Algorithm NextVertex(k)

```
{
  do {
    x[k]=(x[k]+1) mod(n+1);
    if(x[k]==0) return;
    if(G[x[k-1], x[k]] ≠ 0)
    {
      for j=1 to k-1 do if (x[j]==
        break;
```

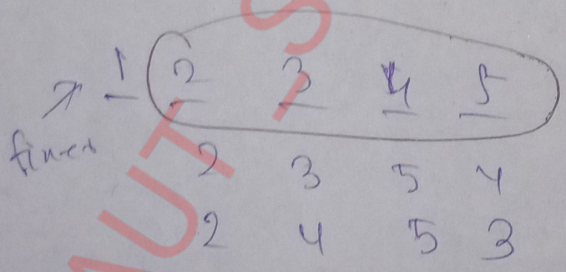
First things that are checked in bonding function first thing you should not take duplicate. Second thing whenever Second you take any vertex there should be an edge from the previous vertex. 3rd thing if you are on the last vertex so then there should be an edge it to the first vertex if so then it's an answer.

```
if(k<n or (k==n) && g[x[n]
                    x[1]]!=0
    return 0;
while(true);
}
```



0, 1, 2, 3, 4, 5, 0

(5+1) = 6



All possibility

4!

(n-1)!

T.C $O(n^n)$

Job sequencing with deadlines: Greedy method

- $n = 5$

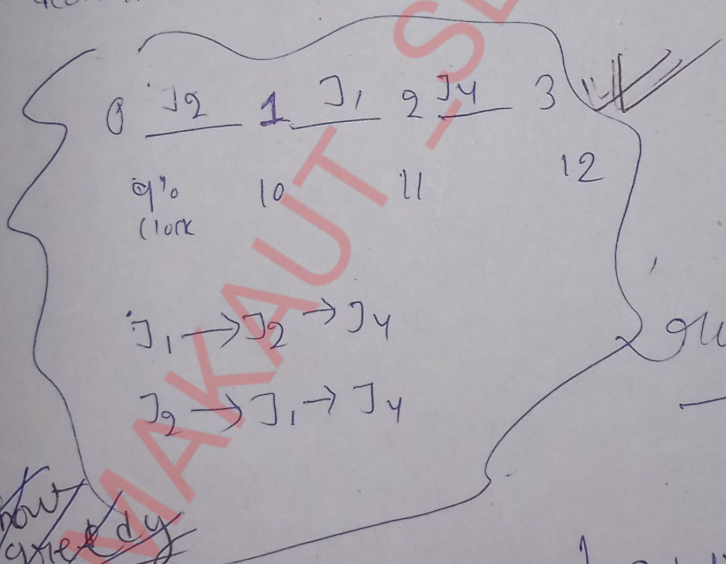
Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
deadlines	2	2	1	3	3

} maximization

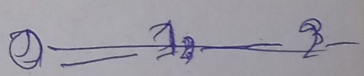
[Highest profit job should be done first]

constraint

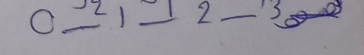
See here 5 jobs are given, 5 tasks are given and if that task is completed we will get this profit so associated with each job there is some profit but that job must be finished within this deadline. It must be complete within the deadline like first job should be completed within a deadline.



20+15+5 = 40
{J1, J2, J4}



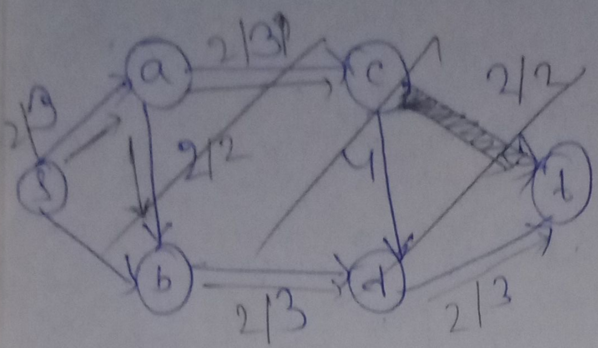
Gantt chart



without greedy

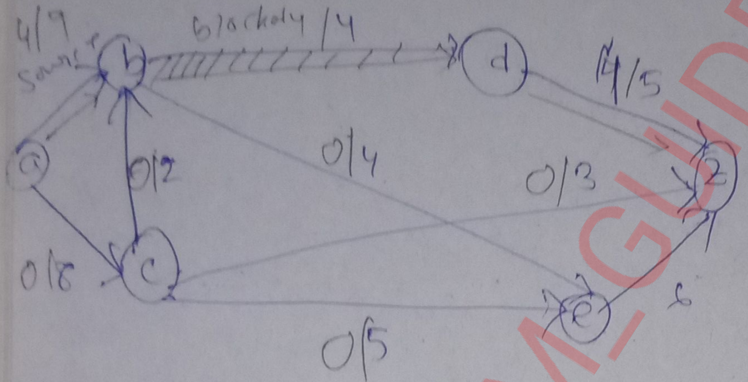
Job consider	slot assign	Solution	Profit
—	—	ϕ	0
1	[1, 2]	J_1	20
2 (second job)	[0, 1], [1, 2]	J_1, J_2	20+15
3	[0, 1], [1, 2]	J_1, J_2	20+15
4	[0, 1], [1, 2], [2, 3]	J_1, J_2, J_4	20+15+5

Max flow min cut :



***

Min cut এ source, sink এ আলাদা হবে.
Maxflow = mincut হবে,
Sum of source should be equal to maxflow



a→b→d→z

a→b→d→z→4
a→b→e→z→4
a→c→z→3
5