Page No. _____
Date _____

Design Analysis And Algorithm :-

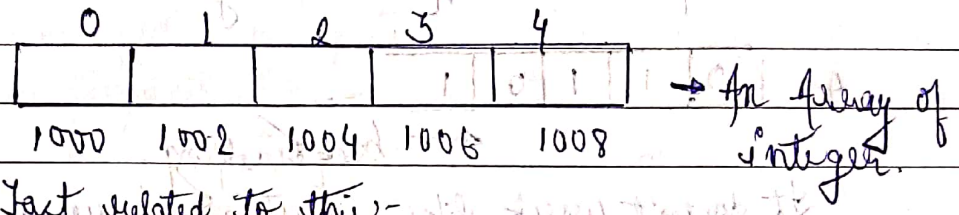
Array :-

definition :- Array is collection of similar datatype or homogenous datatype having continuous memory allocation.

properties of an Array :-

(i) Continuous Memory Allocation :-

A[5]



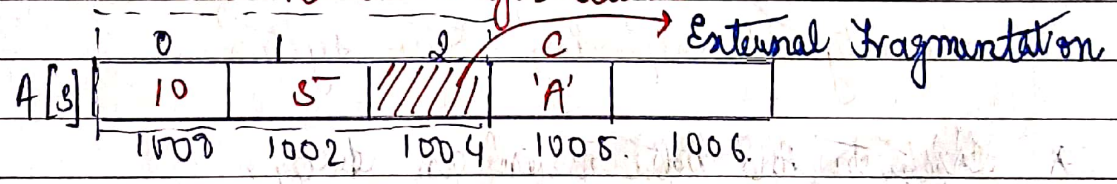
Some fact related to this :-

(i) Memory is allocated during compile time.

(ii) So, the run-time speed is induced.

problems :-

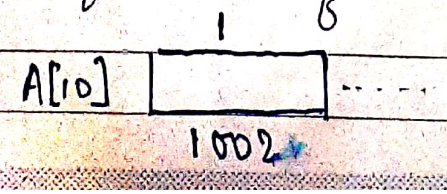
* External Fragmentation :-



Suppose we created A[3] and we store number as A[0] = 10, A[1] = 5, and then allocate a variable type character C = 'A', then 'A' will be allocated to memory address 1005, & the space A[2] will be wasted.

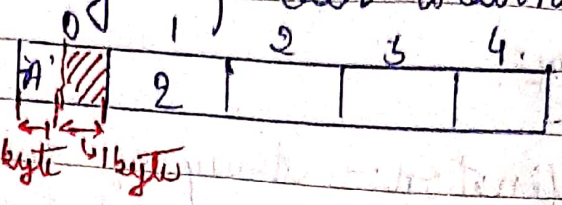
(ii) Why Array have similar datatype?

Ans :- This is because of internal Fragmentation.



Scanned with CamScanner

Suppose we have an array A[5] of type integer, so, it will take two byte for each location.
Now,



if we place
A[0] = 'A', so it will occupy 1 byte and then if A[1] = 2, then the 1 byte will be left as internal fragmentation.

(Imp.) * Semaphore :- An application of an Array :-
Semaphore works on Array and not on stack.

A	0	1	1	0	1	...

← back tracking.

It doesn't work like system have a single space & there it writes 0 & then replaces it by 1.
Actually, like an Array, it stores the globally value of semaphore because it have to maintain a log (record of operations performed) for security & restoring the system.

* Advantages & disadvantage of an Array :-

(1) Space Complexity :- The space complexity of an algorithm, hence program, is the amount of memory it needs to run to completion. Some reason for studying space complexity are :-

- If the program is to run on multi-user system, it may be required to specify

Unit 2 :- Chapter (1) :-

Page No. _____
Date _____

Design Analysis And Algorithm :-

Algorithm Complexity :-

Algorithm complexity is something that defines the efficiency of the program.

Types :-

(1) Space Complexity :-

the space complexity of an algorithm, hence program is the amount of memory it needs to run to completion.

* Reason to study space complexity :-

→ If the program is to run on multi-user system, it may be required to specify the amount of memory to be allocated to the program.

→ We may be interested to know in advance that whether sufficient memory is available to run the program.

→ Can be used to estimate the size of the largest problem that a program can solve.

(2) Time Complexity :-

the time complexity of an algorithm is the amount of time it needs to run to completion.

Reason to study time complexity are:-

We may be interested to know in advance that whether the program will provide a satisfactory real-time response.

Time-Space Tradeoff :-

the best algorithm, hence best program, to solve a given problem is one that requires less space in memory and takes less time to complete execution.

Scanned with CamScanner

But in practice it is not always possible to achieve both of these objectives.
If we try to increase the space, result in reduce in time & vice-versa. This is called as space-time tradeoff.

Algorithm Complexity :-

Best Case Complexity :- the term best-case performance is used in computer science to describe an algorithm's behaviour under optimal conditions.

Average Case Complexity :- the average case complexity of an algorithm is the amount of some computational resource used by the algorithm, averaged over all possible inputs.

Worst Case Complexity :- The worst-case measures the resources (e.g. running time, memory) an algorithm requires in worst-case. It gives an upper bound on the resources required by the algorithm.

* Asymptotic Notations :-

① Big oh 'O' Notation :- Big O is the formal method of expressing the upperbound of an algorithm running time. It is a measure of longest amount of time it could possibly take for the algorithm to complete.

→ It gives the best case complexity.

denoted as $f(n) \leq c.g(n)$

Where, $c > 0$.

Statement :-

For function, f(n) & g(n) of g(n) if there exists an Integer n_0 and a constant $c>0$ such that for all Integers $n>n_0$, $f(n) \le c.g(n)$, then f(n) is big O of g(n).

Small 'o' :- $f(n) < c.g(n)$.

② Big Omega (Ω) Notation :- It describes the least that can happen for a given data size.

* For functions, f(n) & g(n) if there exist an Integer no & a constant $c>0$ such that for all Integers $n>n_0$, $f(n) \ge c.g(n)$ then f(n) is Omega of g(n).

This is denoted by

$$f(n) \ge c.g(n)$$

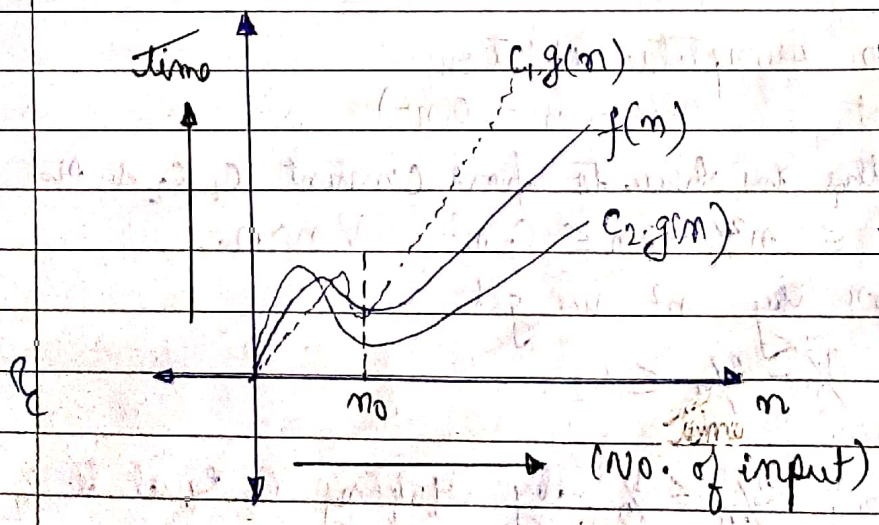
Big Omega is opposite to Big Oh(O) and specifies are lower bound instead of upper bound.

Small 'ω' :- $f(n) > c.g(n)$.

③ Theta 'θ' notation :- In Theta notation, f(n) is bounded from the top & bottom by same function g(n).

denoted by

$$c_1.g(n) > f(n) \ge c_2.g(n)$$



Note :-

* g(n) is Asymptotically tight bound for f(n).
* f(n) is given function
* n is number of inputs.

Page No.
Date

Various Complexities of Algorithm :-

Constant $O(1)$:→ Constant number of operation, not depending on the input data size.
e.g., $n = 1000 \rightarrow 1$ operation.

Logarithmic :- $O(\log n) \rightarrow$ No. of operations proportional of $\log n$ where n is the size of input data.
e.g. $n = 1000 \rightarrow 10$ operations

Linear :- $O(n) \rightarrow$ No. of operations proportional to the input data size, e.g. $n = 10000 \rightarrow 10000$ operations.

Quadratic :- $O(n^2) \rightarrow$ No. of operations proportional to square of size of the input data.
e.g. $n = 500 \rightarrow 250000$ operations

Cubic $O(n^3) \rightarrow$ No. of operation proportional to the cube of the size of the input data,
e.g. $n = 10 \rightarrow 1000$ operations.

Exponential $(e^n) \rightarrow$ Exponential no. of operations fast growing :-
e.g. $n = 20 \rightarrow 1048576$ operations.

Questions on Asymptotic Notations :-

(1) prove that $n^2/2 - 3n = \Theta(n^2)$.

Ans:- So, for this we have to find constant c_1, c_2 & n_0.

$$c_1 n^2 \le n^2/2 - 3n \le c_2 n^2 \quad \forall n > n_0$$

dividing eqn by n^2 we get

$$c_1 \le 1/2 - 3/n \le c_2$$

if $n \ge 1$

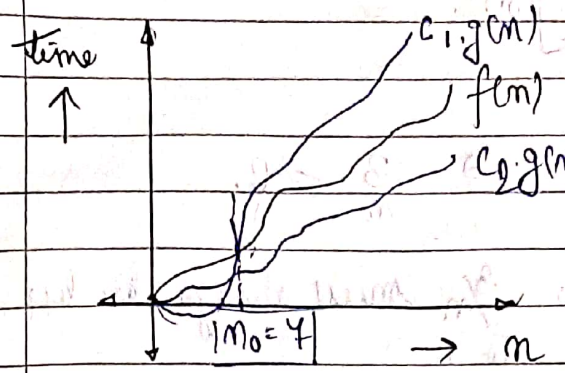
then $1/2 - 3/n \le 1/2$ by making c_2 equal to $1/2$

Scanned with CamScanner

Page No. _____
Date _____

and, $\frac{1}{2} - \frac{3}{n} \geq \frac{1}{14}$ when $n \geq 7$ ($\frac{1}{2} - \frac{3}{n} = 0$ when $n = 6$).

So, $C_1 = \frac{1}{14}$ and, $C_2 = \frac{1}{2}$ with $n_0 = 7$.



So,
$\frac{n^2}{2} - 3n = \Theta(n^2)$.

Note: $an^2 + bn + c = \Theta(n^2)$
Always true.

(2) Is $3n^3 \in \Theta(n^4)$?

Ans:- For this we have to find constant C_1, C_2 & n_0.
Now, $C_1 n^4 \leq 3n^3 \leq C_2 n^4$.
dividing by n^4
$C_1 \leq \frac{3}{n} \leq C_2$.

Now, for $n \geq 3$, $\frac{3}{n} \leq 1$, by making $C_1 = 1$.
and, for $3/n$, there is no positive value exist for which it will always be greater.
So, $3n^3$ doesn't belong to $\Theta(n^4)$.

(3) $2^{2n} \in \Theta(2^n)$

Ans: $2^{2n} = (2^2)^n = 4^n = f(n)$ and, $g(n) = 2^n$.
i.e. $C_1 \cdot 2^n \leq 4^n \leq C_2 \cdot 2^n$.
dividing eqn by 2^n
$C_1 \leq \frac{4^n}{2^n} \leq C_2$.

$\Rightarrow C_1 \leq 2^{2n-n} \leq C_2 \Rightarrow C_1 \leq 2^n \leq C_2$

Now, for $n \geq 1$; $2^n \geq 2$. So lower bound is $C_1 = 2$.
But, for no positive value of n, 2^n is always $\leq C_2$.
(C_2 be any constant (positive))
So, $2^{2n} \notin \Theta(2^n)$.

Scanned with CamScanner

Page No.
Date

Ques:- (4) Show that $3n^3 = O(n^4)$ for appropriate c and n_0.

Ans:- $f(n) = 3n^3$ $\quad g(n) = n^4$.

$0 \le f(n) \le c.g(n)$.

$f(n) \le c.g(n)$

$\Rightarrow 3n^3 \le c.n^4$.

dividing eqn by n^4 $\Rightarrow \frac{3}{n} \le c$.

Now, for $n \ge 3$; $3/n$ must always be less than or equal to 1.

So, $c = 1$.

So, check table

n	$f(n)$	$c.g(n)$
1	$3(1)^3 = 3$	$1.(1)^4 = 1$.
2	$3(2)^3 = 24$	$1.(2)^4 = 16$.
3.	$3(3)^3 = 81$	$1.3^4 = 81$. $f(n) = g(n)$
4.	$3(4)^3 = 192$	$1.4^4 = 64 \times 4 = 256$ $f(n) \le g(n)$

i.e., $f(n) = O(g(n))$ or $\Rightarrow 3n^3 = O(n^4)$.

$\boxed{\text{Note:- Any linear function } an+b \text{ is in } O(n^2)}$

(5) $\sqrt{n} = \Omega(\lg n)$. Choose c and n_0.

Ans:- $f(n) = \sqrt{n} = n^{1/2}$

and, $g(n) = \lg n$.

$f(n) \ge c.g(n)$

i.e. $n^{1/2} \ge (\lg n).c$ $\Rightarrow \sqrt{n} \ge c.\lg n$.

Applying log to both side or can use double derivative test.

let $q(n) = \sqrt{n} - \log n$.

Now, $q'(n) = \frac{1}{2} n^{-1/2} - (1/n)$

$q'(n) = 0 \Rightarrow \frac{1}{2} n^{-1/2} = 1/n \Rightarrow \frac{1}{2} = \frac{n^{1/2}}{n}$

$\Rightarrow n = 2n^{1/2} \Rightarrow n^2 = 4n$

$\Rightarrow \boxed{n = 4}$

Scanned with CamScanner

Page No. _____
Date _____

double differentiating

$$g''(n) = \frac{d^2}{dn}\left(\sqrt{n} - \log n\right) = -\frac{1}{4} n^{-3/2} - (-2) n^{-2}$$

$$= \frac{2}{n^2} - \frac{1}{4}\left(\frac{1}{n^{3/2}}\right)$$

$$= \frac{2}{(16)} - \frac{1}{4}\left(\frac{1}{2^{\frac{3}{2}} \times x}\right)$$

$$= \frac{1}{8} - \frac{1}{4}\left(\frac{1}{8}\right) = \frac{1}{8} - \frac{1}{32} = -\frac{3}{32} \geq 0$$

ie. concave up.

So, $\sqrt{n}$ is always greater than or equal to $\log n$.

So, $c = 1$, & $n_0 \geq 1$.

Properties of Asymptotic Notations :-

Applied to all Notations ie. Θ, Ω, O.

① Transitivity :- (All three)

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

$\Rightarrow f(n) = \Theta(h(n))$

② Reflexive Property :- (All three)

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

③ Symmetric Property :- (only Θ)

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

④ Complementarity (only O, Ω)

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

Note :-

Natural log : $\ln a = \log_e a$
Binary log : $\lg a = \log_2 a$

Scanned with CamScanner

Page No.
Date

* Recursion Vs Iteration:-

Key differences:-

1. Recursion is when a method in a program repeatedly calls itself whereas, iteration is when a set of instructions in a program is repeatedly executed.

2. A recursive statement contains set of instructions, statement calling itself, & a termination condition whereas iteration statement contains initialization, increment condition, set of instruction within a loop & a control variable.

3. A conditional statement decides the termination of recursion and control variable value decide the termination of the iteration statement.

4. Infinite recursion can lead to system crash whereas infinite iteration consumes CPU cycles.

5. Variable created during recursion are stored in stacks, whereas iteration doesn't have a stack requirement.

6. Due to function calling overhead execution of recursion is slower whereas, execution in iteration is faster.

7. Recursion reduces the size of the code whereas, iteration is for the faster execution of code.

Scanned with CamScanner

Page No. _____
Date _____

Recursion (detail study):-

* Defination:- the process in which a function call itself directly or indirectly is called recursion & corresponding function is called as recursive function.

* Base Condition in Recursion:-
In recursive program, the solution to base case is provided & solution of bigger problem is expressed in terms of smaller problems.

Ex:-
```
int fact(int n)
{
    if(n<=1)        // base case
        return 1;
    else
        return n* fact(n-1);
}
```

(Imp.) * How memory is allocated to different function calls in recursion?

- When any function is called from main(), the memory is allocated to it as stack.
- A recursive function calls itself, the memory for called function is allocated to top of memory for calling function & a different copy of local variable is created for each function call.
- When the base case is reached, the function return its value to the function by whom it is called & the memory is deallocated & the process continues.

* Why Stack Overflow error occurs in recursion?
- If base case is not reached, then stack overflow problem may arise.

Scanned with CamScanner

Page No.
Date

* What is the difference between direct & indirect recursion?

Ans:- A function is called direct recursive if it calls the same function fun.
A function fun is called indirect recursive if it calls another function say fun_new & fun_new calls fun directly or indirectly.

Ex:-
```
void fun()
{
    // some code
    fun();
}
```
(direct recursion)

```
void fun()
{
    // some code
    new_fun();
}
void newfun()
{
    // some code
    fun();
}
```
(indirect Recursion)

* What is the difference between tailed & non-tailed recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function.
If some more code are written after that, then it is non-tailed recursive relation or recursion.

* Disadvantage:- Recursive program has greater space requirements than iterative function as all functions will remain in stack until base case is reached.
It also has greater time requirement because of function call & overhead condition.

Scanned with CamScanner

Page No. _____
Date _____

Recurrence Relation
- Iterative Relation
- Recurrence Relation.

To solve Recurrence Relation; there are three types of methods:-

1. Recursion Tree
2. Substitution Method
3. Master Method.
4. Iteration Method / back Substitution Method.

Note:- General Form of any Recurrence Relation:-

$$\Rightarrow \boxed{T(n) = aT(n/b) + f(n)}$$

Where, n is size of problem
n/b is the size of sub-problem
$f(n)$ is cost of work done outside the recursive calls

* Recursion Tree:-

Ex:- $T(n) = 2T(n/2) + n$ Find the Time complexity of this equation.

Note:- The idea of a Recursion Tree is to expand $T(n)$ to a tree with the same total cost. Two things are imp:-

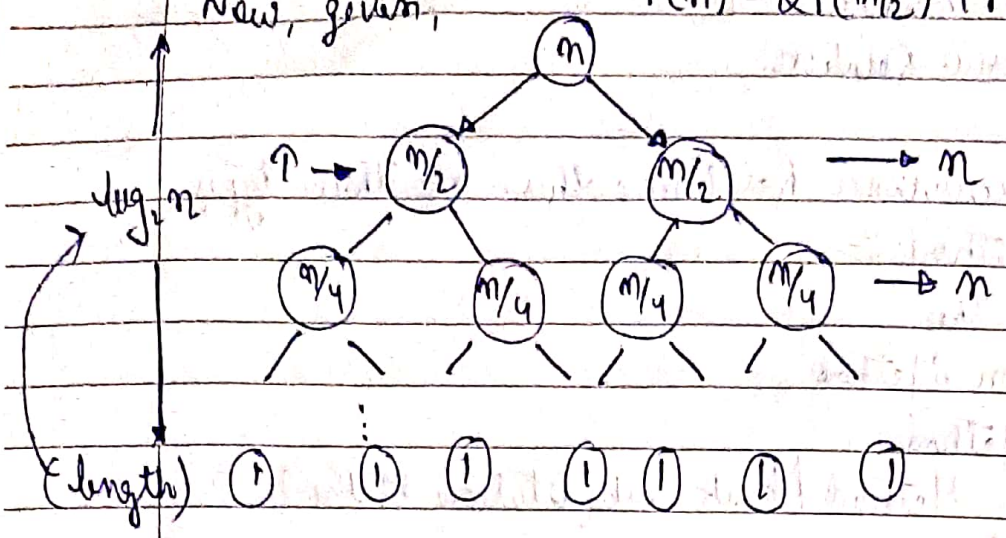
(i) height of the tree
(ii) The cost of the nodes at each level.

Steps:-

Step 1:- Find out the last level
Step 2:- Complexity at last level
Step 3:- Total complexity = complexity of all levels + complexity of last levels.

Scanned with CamScanner

Now, given, $T(n) = 2T(n/2) + n$



Find last level $n/2^i = 1$. Note:- this 2 is nothing but value of b.

$\Rightarrow n = 2^i$

$\Rightarrow \log_2 n = \log_2 2^i$

$\Rightarrow \boxed{i = \log_2 n}$

total number of levels of tree $= \log_2 n + 1$

Complexity at last level = no. of nodes at last level

$= 2^i = 2^{\log_2 n} = n$.

Note:- this 2 is nothing but value of a.

Total Complexity = Complexity of all levels + Complexity of last level.

$= \sum_{i=0}^{\log_2(n-1)} n + O(n)$.

OR

$T(n) = n + 2[n/2] + 4(n/4) + \ldots$ upto $\log_2 n$ times

$= n(1+1+1+\ldots \log_2 n \text{ time})$

$= n[\log_2 n]$

So, complexity $= n[\log_2 n]$.

Practice Questions :-

(1) $T(n/2) + 1 = T(n) \Rightarrow T(n) = T(n/2) + 1$ → Constant

Recurrion	Node	Tree formation	Row Sum
$T(n)$	1	c	c
$T(n/2)$	1	c (One Recursive Call)	c
$T(n/2^2)$	1	c	c
⋮	1	↓	⋮
$T(n/2^i)$	i upto sh time.	c	c
$T1$		c	

$n/2^i = 1 \Rightarrow 2^i = n \Rightarrow i = \log_2 n$. $\quad \sum_{i=0}^{\log_2 n} (c)$.

$$= c \sum_{i=0}^{\log_2 n} (1) = c \log_2 n$$

Since c is a constant i.e. let take 1 here.

So, complexity = $\log_2 n$.

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

Two Recursive Call

$T(n) = 2T(n/2) + 1$ → Constant

Recurrion	No. of Node	Tree formation	Row Sum
$T(n)$	1	c	c
$T(n/2)$	2	c c	2c
$T(n/2^2)$	$4 = 2^2$	c c c c	4c
⋮			⋮
$T(n/2^i)$	2^i	c c c c c c	$2^i c$

$n/2^i = 1 \Rightarrow n = 2^i \Rightarrow \log_2 n = \log_2 2^i \Rightarrow i = \log_2 n$.

$= c + 2c + 2^2c + \dots + 2^i c = c(1 + 2 + 2^2 + 2^3 + \dots + nc)$

$= c(1 + 2 + 4 + 8 + 16c \dots + 2^i)$

$= c\left(1 + \frac{(2^{i+1} - 1)}{(2-1)}\right) = c(2^{i+1} - 1) = 1(2(2)^i - 1)$

Scanned with CamScanner

Now, $2^i = n$. So, Complexity $= 2n - 1$

So, complexity $= O(n)$.

(3) $T(n) = 16T(n/4) + n$.

Recurrence	Tree	Root sum
$T(n)$	n	n
$T(n/4)$	$n/4 \quad n/4 \quad n/4 \quad n/4 \cdots n/4$ (16 times)	$16(n/4) = 4n$
$T(n/4^2)$	$T(n/4^2) \quad T(n/16)$ similarly to all	$16(16(n/16)) = 4^2 n$
$T(n/4^i)$	$T(1) \cdots T(1)$	$4^i n$.

total term

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

$\Rightarrow n = 4^i$

$\Rightarrow \log_4 n = i \log_4 4$

$\Rightarrow i = \log_4 n$.

Total Root sum $= n + 4n + 4^2 n + \cdots + 4^i n$,

$= n(1 + 4 + 4^2 + \cdots + 4^i)$

$= n\left(\frac{1(1-(4)^i)}{(1-4)}\right)$

$= n\left(\frac{4^i - 1}{3}\right)$.

$= \frac{n}{3}\left(4^{\log_4 n} - 1\right)$ $(\because i = \log_4 n)$

$= n/3(n-1) = n^2/3 - n/3$.

So, complexity $= O(n^2)$.

Scanned with CamScanner

Page No. _____
Date _____

(4) $T(n) = 16T(n/4) + n^3$.

Recurrence	tree	Row sum
$T(n)$	n^3	n^3
$T(n/4)$	$(n/4)^3 \; (n/4)^3 \; \cdots \; (n/4)^3$ (16 times)	$16(n/4)^3 = 16\left(\frac{n^3}{4^3}\right) = \frac{n^3}{4}$
$T(n/4^2)$	$(n/4^2)^3$ — 16 times	$= 16\left(16\left(\frac{n^3}{4^{2*}}\right)^3\right)$
		$= 4^4\left(\frac{n^3}{4^6}\right) = n^3/4^2$
$T(n/4^i)$	$T(1), \ldots, T(1)$	$= n^3/4^i$

No. of terms (calculation) $\Rightarrow n/4^i = 1 \Rightarrow n = 4^i$

$\Rightarrow \boxed{\log_4 n = i}$

Total Row Sum $= n^3 + \frac{n^3}{4} + n^3/4^2 + \cdots + \frac{n^3}{4^i}$

$$= n^3\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^i}\right)$$

$$= n^3\left(\frac{1\left(1-(1/4)^i\right)}{(1-1/4)}\right)$$

$$= n^3\left(\frac{(1/4)^i - 1}{1/4 - 1}\right) = n^3\left(\frac{4^{\frac{1}{\log_4 n}} - 1}{-3/4}\right)$$

$$= n^3\left(\frac{1/n - 1}{-3/4}\right) = n^3\left(\frac{(1-1/n)4}{3}\right)$$

$$= n^3\left(\frac{n-1}{3n}\right) \times 4$$

$$= 4n^2(n-1)$$

$$= 4n^3 - 4n^2.$$

So, complexity $= O(n^3)$

Scanned with CamScanner

Page No.
Date

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

Recurrion tree

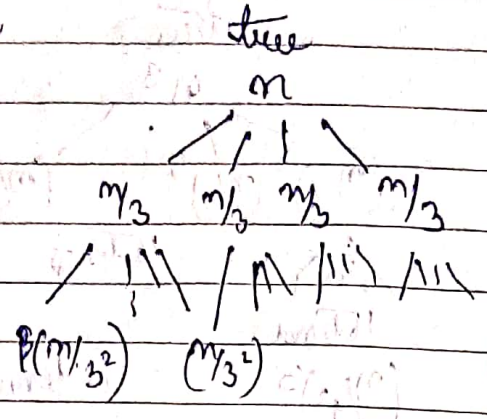
$T(n)$

$T(n/3)$

$T(n/3^2)$

$\vdots$

$T(n/3^i)$ $= T(1)$



Row Sum

n

$4(n/3)$

$16(n/3^2)$

$= 4^2(n/3^2)$

$= 4^i(n/3^i)$

No. of terms

$n/3^i = 1 \Rightarrow n = 3^i$

$\Rightarrow \boxed{\log_3 n = i}$

So, Now, Row Sum

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

$$= n\left[1 + 4/3 + (4/3)^2 + (4/3)^3 + \dots + (4/3)^i\right]$$

$$= n\left(1\left[\frac{1-(4/3)^i}{(1-4/3)}\right]\right)$$

$$= n\left[\frac{1-(4/3)^{\log_3 n}}{\frac{(3-4)}{4}}\right]$$

$$= n\left[\frac{(4/3)^{\log_3 n} - 1}{1}\right] \quad 4\left[n^{1.26} - n\right]$$

Property of log

$$\boxed{a^{\log_b c} = c^{\log_b a}}$$

$$= 4n\left[\frac{4^{\log_3 n}}{3^{\log_3 n}} - 1\right]$$

$$= \cancel{4n}\left[\frac{4^{\log_3 n} - n}{\cancel{n}}\right] = 4\left[4^{\log_3 n} - n\right]$$

$$= 4\left[n^{\log_3 4} - n\right]$$

So, Complexity $= O(n^{1.26})$

Scanned with CamScanner

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

Page No. _____
Date _____

Master's Theorem:-

Let $a \geq 1$ and $b > 1$ be constant, let $f(n)$ be a function, & let $T(n)$ be defined on the non negative integers by the recurrence

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

Step 1:- Calculate $X = n^{\log_b a}$

Step 2:- Compare X with $f(n)$

a) if $f(n) < x$, then $T(n) = \Theta(X)$

b) if $f(n) = X$, then $T(n) = \Theta(X \log n)$

c) if $f(n) > X$, then $T(n) = \Theta(f(n))$.

Alternative:-

general form

$T(n) = aT(n/b) + \Theta(n^k \log^p n)$. where

$a \geq 1$, $b > 1$ $k \geq 0$. & p is any real number.

then.

(i) if $a > b^k$, then $\boxed{T(n) = \Theta(n^{\log_b a})}$

(ii) if $a = b^k$

→ if $p > -1$, then $\boxed{T(n) = \Theta(n^{\log_b a} \log^{p+1} n)}$

→ if $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log(\log n))$

→ if $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

(iii) if $a < b^k$

→ if $p \geq 0$, then $\boxed{T(n) = \Theta(n^k \log^p n)}$

→ if $p < 0$, then $T(n) = \Theta(n^k)$.

Master Method:- It categorises the function in one of the three categories & then calculate time complexity on basis of defined formulas.

Scanned with CamScanner

Page No.
Date

* (1) $T(n) = 9T(n/3) + n$.

$X = n^{\log_b a} = n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2}$
$= n^2$!

and,

$f(n) = n$.

$f(n) = X$ Case I:- $f(n) = \Theta(X \log n) = (n^2 \log n)$
for $n = 1$.

$f(n) < X$ Case II:- $f(n) = \Theta(T(n)) = \Theta(n^2)$
for $n > 1$.

* (2) $T(n) = T(2n/3) + 1$

$X = n^{\log_b a} = n^{\log_{(3/2)} 1} = n^0 = 1$.

And, $f(n) = 1$.

Now, $f(n) = X$

So, Complexity $= \Theta(\log n)$

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

$a = 3$; $b = 4$. $X = n^{\log_3 4}$.

Now, $X < f(n)$.

So, Complexity $= \Theta(f(n))$
$= \Theta(n \log n)$.

* (4) $T(n) = T(n/2) + 1$

$X = n^{\log_a b} = n^{\log_2 1}$ $f(n) = 1$.
$= n^0 = 1$.

$f(n) = X$.

So, Complexity $= \Theta(X \log_2 n)$
$= \Theta(1 \cdot \log_2 n)$

Scanned with CamScanner

Page No. _____
Date _____

* (5) $T(n) = 2T(n/2) + 1$

$X = n^{\log_a b} = n^{\log_2 2} = n$.

$f(n) = 1$. Now, $X > f(n)$.

So, complexity $= \Theta(X) = \Theta(n)$.

* (6) $T(n) = 16T(n/4) + n$

$X = n^{\log_b a} = a^{\log_4 b} = n^{2 \log_4 4} = n^2$.

$f(n) = n$. Now, $X > f(n)$.

So, complexity $= \Theta(X) = \Theta(n^2)$.

* (7) $T(n) = 16T(n/4) + n^3$

$X = n^{\log_b a} = n^{\log_4 16} = n^2$.

$f(n) = n^3$ Now $f(n) > X$

So, complexity $= \Theta(f(n)) = \Theta(n^3)$.

* PRACTICE QUESTIONS:-

* (1) $T(n) = 3T(n/2) + n^2$

$X = n^{\log_b a} = n^{\log_2 3}$ $f(n) = n^2$

Now, $f(n) > X$ So, Complexity $= \Theta(n^2)$

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

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

$f(n) = n^2$.

Now, $f(n) = X$ So, complexity $= \Theta(X \cdot \log n)$

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

* (3) $T(n) = T(n/2) + 2^n$.

$X = n^{\log_b a} = n^{\log_2 1} = n^0 = 1$. $f(n) = 2^n$.

$f(n) = \Theta(f(n)) = \Theta(2^n)$ $(\because f(n) > X)$

Scanned with CamScanner

Page No.
Date

* (4) $T(n) = 2^n T(n/2) + n^n$.

Since $a = 2^n$ i.e. not constant.

So, Master theorem is not applicable.

* (5) $T(n) = 16T(n/4) + n$.

$X = n^{\log_b a} = n^{\log_4 16} = n^2$.

$f(n) = n$.

So, Complexity $= \Theta(X) = \Theta(n^2)$ $(\because X > f(n))$

doubt * (6) $T(n) = 2T(n/2) + n \log n$.

Actual answer $= n \log^2 n$.

$X = n^{\log_b a} = n^{\log_2 2} = n$

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

Now, $f(n) > X$ $\quad \frac{n^{\log_2 2}}{n \log n} = \frac{n}{n \log n} = \frac{1}{\log n}$

So, Complexity $= \Theta(f(n)) = \Theta(n \log n)$

Not applicable because of non-polynomial difference $(a=b)$

* (7) $T(n) = 2T(n/2) + \frac{n}{\log n}$ $\quad \frac{n^{\log_2 2}}{n}(\log n) = \log n$ non-polynomial difference.

(Not applicable because of non-polynomial difference between $f(n)$ and $n^{\log_b a}$ $n^{\log_a b}$)

* (8) $T(n) = 2T(n/4) + n^{0.51}$

$X = n^{\log_b a} = n^{\log_4 2} = n^{1/2} = n^{0.5}$.

$f(x) = n^{0.51}$.

Now, $f(x) > X$.

So, Complexity $= \Theta(f(n)) = \Theta(n^{0.51})$.

* (9) $T(n) = 0.5\, T(n/2) + 1/n$.

$a < 1$; Master theorem is not applicable.

* (10) $T(n) = 16T(n/4) + n!$

$X = n^{\log_b a} = n^{\log_4 16} = n^2$ $\quad f(x) = n!$

Now, $f(x) > X$

So, Complexity $= \Theta(n!)$

Scanned with CamScanner

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

$a > 0$

$b > 1$

$a \neq b$

Page No.
Date

(11) $T(n) = \sqrt{2}\, T(n/2) + \log n$.

$X = n^{\log_b a} = n^{\log_2 2^{1/2}} = n^{\frac{1}{2}\log_2 2} = n^{1/2}$.

and, $f(n) = \log n$.

Now, $X > f(n)$

So, complexity $= \Theta(\sqrt{n})$.

(12) $T(n) = 3T(n/2) + n$.

$X = n^{\log_2 3}$ $\quad f(n) = n$.

and, $f(n) < X$.

So, complexity $= \Theta(n^{\log_2 3})$.

(13) $T(n) = 3T(n/3) + \sqrt{n}$.

$X = n^{\log_3 3} = n$ $\quad f(n) = \sqrt{n}$.

and, $f(n) < X$.

So, complexity $= \Theta(n)$

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

$X = n^{\log_2 4} = n^2$.

and, $f(n) = cn$ so, $f(n) < X$

So, complexity $= \Theta(n^2)$.

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

$X = n^{\log_4 3}$ and $f(n) = n \log n$.

and, $X \nless f(n)$.

So, complexity $= \Theta(n \log n)$.

(16) $T(n) = 3T(n/3) + n/2$.

$X = n^{\log_3 3} = n$. and, $f(n) = n/2$.

Comp. $X = f(n)$.

So, Complexity $= \Theta(X \log n) = \Theta(n \log n)$.

Scanned with CamScanner

Page No.
Date

* (17) $T(n) = 6T(n/3) + n^2 \log n$.

$X = n^{\log_3 6}$ & $f(x) = n^2 \log n$.

$f(n) > X$.

So, complexity $= n^2 \log n$.

Doubt

* (18) $T(n) = 4T(n/2) + \frac{n}{\log n}$

$X = n^{\log_2 4} = n^{\log_2 2^2} = n^2$.

$f(n) = n/\log n$.

$f(n) < X$.

So, complexity $= \Theta(n^2)$.

* (19) $T(n) = 64T(n/8) - n^2 \log n$

$X = n^{\log_8 64} = n^2$.

But Master theorem is not applicable since $f(n)$ is not positive.

* (20) $T(n) = 7T(n/3) + n^2$

$X = n^{\log_3 7}$ $f(n) = n^2$.

$f(n) > X$

So, complexity $= \Theta(f(n)) = \Theta(n^2)$

* (21) $T(n) = 4T(n/2) + \log n$.

$X = n^{\log_2 4} = n^2$

$f(n) = \log n$

So, complexity $= (f(x) < X) = \Theta(X) = \Theta(n^2)$

* (22). $T(n) = T(n/2) + n(2 - \cos n) \Rightarrow$ Does not apply.

But the regularity condition is violated.

(Consider $n = 2\pi k$, where k is odd & arbitrary large. For any such choice of n, you can show $c \geq 3/2$ thereby violating the regularity conditions.)

Scanned with CamScanner

Page No
Date

Substitution Method:-

Step 1:- Guess the solution.
Step 2:- Verify using the Principle of Mathematical Induction

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

Suppose the complexity be $O(n^2)$.
i.e.

$$f(n) \le c \cdot g(n).$$
$$\Rightarrow T(n) \le c \cdot n^2.$$

Given

$$T(n) = T(n-1) + n.$$
$$\le c \cdot (n-1)^2 + n. \quad (\because T(n) \le c \cdot n^2$$

So, $\quad$ So, $T(n-1) \le c \cdot n^2$).

for Now, $n=1$, $c=1$

LHS $T(1) = T(1-1)^2+1 = \therefore T(1) =$ RHS $= 1 \cdot (1-1)^2+1 = 1$.

for $n=2$. $\quad$ i.e. true for $n=1$.

LHS $T(2) = T(2-1) = T(1) = 1$.

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

i.e. $T(n) \le c \cdot g(n)$.

Hence, proved that complexity $= O(n^2)$

Iteration Method / Back Substitution Method:-

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

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

replace, n by $n-1$

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

So, $T(n) = [T(n-2) + n-1] + n.$

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

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

Scanned with CamScanner

Page No.
Date

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

similarly, $T(n) = T(n-k) + kn - (1+2+3+\dots+k)$

$n-k = 1$.

$n = (1+k) \Rightarrow k = (n-1)$.

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

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

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

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

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

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

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

$$= 1 + n^2/2 - n.$$

So, complexity $= O(n^2)$.

Scanned with CamScanner

Unit - 2 "Chapter (3)"

Page No. _____
Date _____

Sorting Technique :- Sorting in linear time

(i) Counting Sort
(ii) Radix Sort

* Counting Sort :-

Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values. (kind of hashing).

Time Complexity :-

$O(n+k)$ where n is the number of elements in input array and k is the range of input.

Auxiliary Space :-

$\boxed{O(n+k)}$

Points :-

(1) Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted.

(2) It is not a comparison based sorting. It running time complexity is $O(n)$ with space proportional to the range of data.

(3) It is often used as a sub-routine to another sorting algorithm like radix sort.

(4) Counting sort uses a partial hashing to count the occurrence of the data object in $O(1)$

(5) Counting sort can be extended to work for negative inputs also.

Scanned with CamScanner

Page No.
Date

* Algorithm :-

```
Counting Sort (A, B, k)
for i = 0 to k
    C[i] = 0
for j = 1 to A length or n
    C[A[j]] = C[A[j]] + 1
for i = 1 to k
    C[i] = C[i] + C[i-1]
for j = n for A length to 0.
    B[C[A[j]]] = A[j],
    C[A[j]] = C[A[j]] - 1,
```

Ex:- A

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

Step 1:-

	0	1	2	3	4	5
C	0	0	0	0	0	0

size of Array C = Maximum element in A
here = 5. (0-5)

Step 2:-

	0	1	2	3	4	5
C	0	2	2	3	1	1

Number of time occurence of each ~~cha~~ number.

A

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

C

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

step ②

step ①

C

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

B[7] = 3

B

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

Scanned with CamScanner

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

Page No. _____
Date _____

$j = 9$	$j = 8$	$j = 7$	$j = 6$	$j = 5$
$A[9] = 3$	$A[8] = 1$	$A[7] = 3$	$A[6] = 2$	$A[5] = 4$
$C[3] = 7$	$C[1] = 2$	$C[3] = 7$	$C[2] = 4$	$C[4] = 8$
$B[7] = 3$	$B[2] = 1$	Already done	$B[4] = 2$	$B[8] = 4$
		$7 - 1 = 6$		
		$B[6] = 3$		

$j = 4$	$j = 3$	$j = 2$	$j = 1$
$A[4] = 1$	$A[3] = 3$	$A[2] = 5$	$A[1] = 2$
$C[1] = 2$	$C[3] = 7$	$C[5] = 9$	$C[2] = 4$
Already done	Already done	$B[9] = 5$	$B[4] = 2$
$2 - 1$	$7 - 1 = 6$ Already done		
$B[1] = 1$	$6 - 1 = 5$		
	$B[5] = 3$		

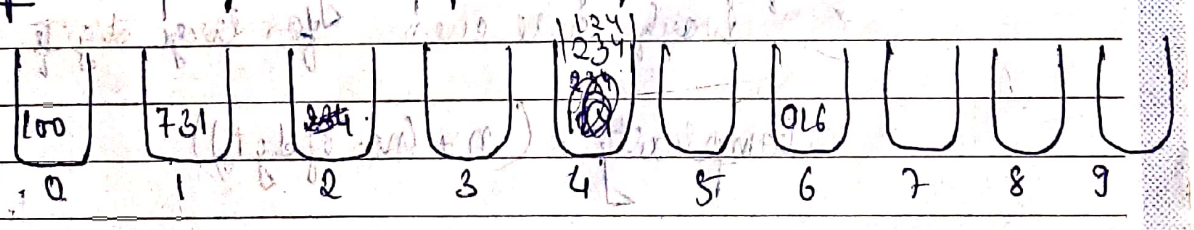
drawback:- space complexity is very large if the size increases.

* RADIX SORT → (using - Bucket Sort
- Counting Sort.)

234	124	016	731	100

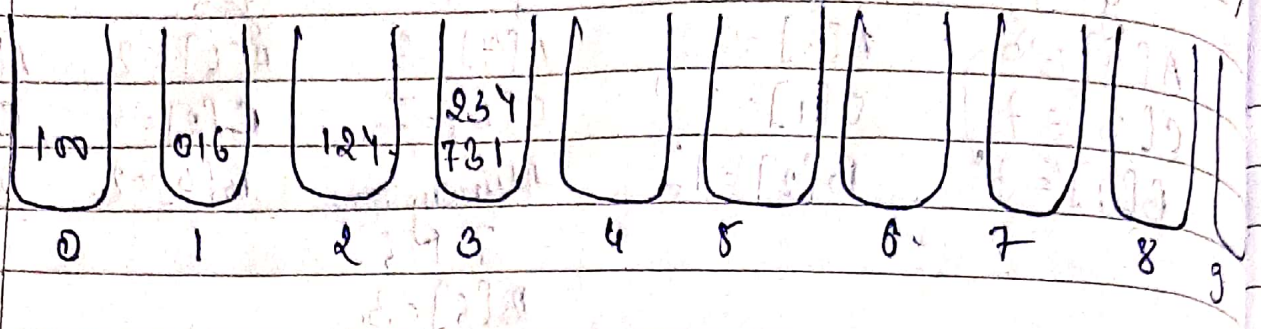
if 3-digit number → it will run three time sorting run.
⇒ n-digit numbers → n-time sorting run.

Step 1:- Unit place will put in pocket



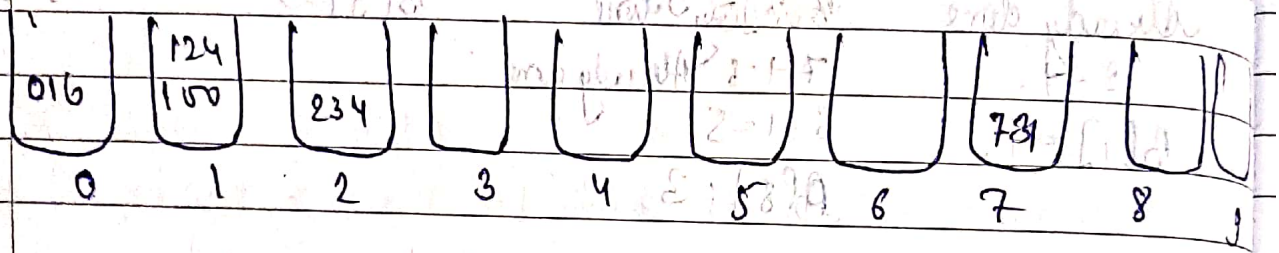
Step 2:- put tenth digit to sort:-
Current Status 100, 731, 234, 124, 016.

100, 431, 124, 234, 016



Current Status :- 100, 016, 124, 734, 234

Step First



016, 100, 124, 234, 731

Algorithm (using Bucket Sort) :-

```
Radix Sort (int a, int n, int digits)
{
    for (i=1; i< digits; i++)
        Count Sort (0, n, i);
}.
```

* Benifits :- * Space Complexity Reduces.
* Work well even for large digits.

Complexity = (n * (no. of digits)).

Abhishek

Page No. _____
Date _____

Sorting Algorithm (Divide And Conquer):-

* Insertion Sort :-
  - → Design Approach :- incremental
  - → Sort in place :- Yes
  - → Best case :- $\Theta(n)$
  - → Worst Case :- $O(n^2)$

* Bubble Sort :-
  - → Design Approach :- incremental.
  - → Sort in place :- Yes
  - → Running time :- $O(n^2)$
  - → Best Case :- $\Theta(n^2)$

* Selection Sort :-
  - → Design Approach :- incremental
  - → Sorts in place :- Yes
  - → Running time :- $\Theta(n^2)$.

* Merge Sort :-
  - → Design Approach :- divide & Conquer
  - → Sorts in place :- No.
  - → Running time :- (Compute)

"Divide And Conquer"

* Divide the problem into the number of sub-problems :-
  - → similar sub-problems of smaller size.

It basically solve problem using three steps :-

1. Divide :- Break the given problem into sub-problems of same type.
2. Conquer :- Recursively solve these problem
3. Combine :- Appropriately combine the result.

Scanned with CamScanner

Page No
Date

Algorithm Working On Divide And Conquer:-

① Binary Search:- is a searching algorithm. At each step, the algorithm compares the input element X with the value of the middle element in array.

* Sequential searching solve this using complexity O(n), while Binary Search solve this using O(log n) complexity.

Reccursive Code:-

Algorithm:-
```
Binary Search (A[], value, low, high)
{ if(low <= high)
  {
    int mid = low + (high - low)/2;
      if(A[mid] == value)
        return mid;
    if (A[mid] > value)
        return Binary Search(A, value, low, mid - 1);
    else
        return Binary Search(A, value, mid+1, high)
  } return -1;
```

Complexity:- Data Structure Used:- Array
Worst Case Performance:- O(log n)
Best Case Performance:- O(1)
Average Case Performance:- Θ(log n)
Worst Space Complexity:- O(1)
(∵ uses single Array)

Reccursive Eqn:- $\boxed{T(n) = T(n/2) + C}$

$X = n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

∵ X Complexity = $f(n) = C$ (Complexity)

So, ∴ Applying Master's theorem Case 2 and apply, Complexity = $O(X \cdot \log n) = O(\log n)$

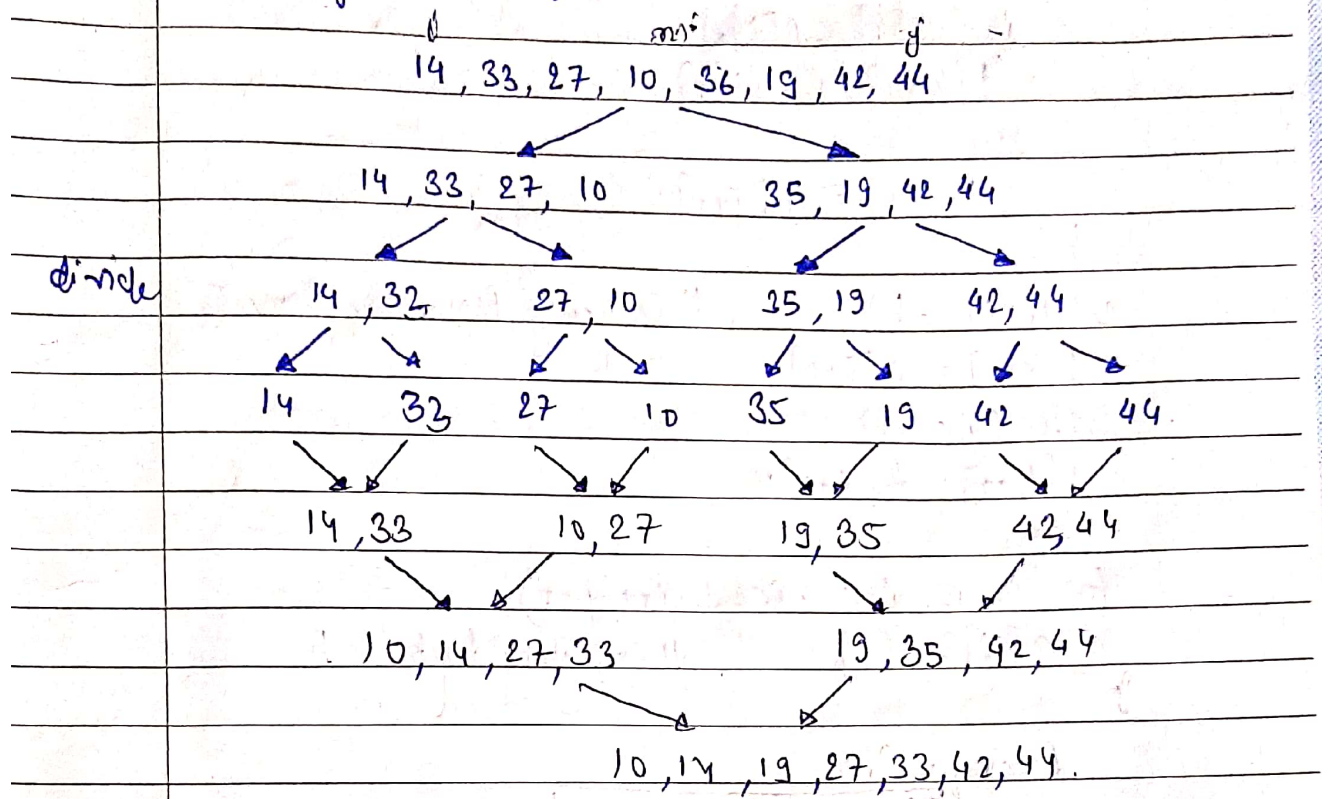
Scanned with CamScanner

Page No. _____
Date _____

* Merge Sort :-

It is a sorting algorithm. The algorithm divides array in two halves, recursively sorts them and finally merges the two sorted halves.

How Merge Sort Work :-



* Algorithm :-

```
void MergeSort (int i, int j)
{
    int m;
    if (i<j)
    {
        m = (i+j)/2;
        MergeSort (i, m);
        MergeSort (m+1, j);
        // Merging The Array
        ArrayMergeSort (i, m, m+1, j);
    }
}
```

Advantage :-

→ Guaranteed to run in $O(n \log n)$ in any case.

Disadvantage :-

- Require Extra Space $= N$.

Array 1 initial Array 1 final Array 2 initial Array 2 final

```
Array Merge Sort (int a, int b, int c, int d)
{
  int t[50], k=0;
  int i=a, j=c, k=0;
  while (i<=b && j<=d)
  {
      if (A[i] < A[j])
        t[k++] = A[i++];
      else
        t[k++] = A[j++];
  }
  while (i<=b)            // Collecting Remaining Elements
    t[k++] = A[i++];
  while (j<=d)
    t[k++] = A[j++];

  for (i=a, j=0; i<=d; i++, j++)
    a[i] = t[j];          // inserting back to A
}
```

Recurrence Eqn:-

$$T(n) = 2T(n/2) + c.n$$

Worst Case Complexity (Time) :- $O(n * \log n)$

Best Case Complexity (Time) :- $\Omega(n * \log n)$

Average time Complexity :- $\Theta(n * \log n)$

Space Complexity :- $O(n)$

Proof :- Using Master's theorem :

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

and, $f(x) = c.n$ Complexity $f(n) = n$

Now, since both are same,

Case Two Applied,

Complexity of Algorithm $= \Theta(X . \log n)$

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

Scanned with CamScanner

Page No. _____
Date _____

Quick Sort :-

It is a sorting algorithm. The algorithm picks a point element, rearranges the array element in such a way that all elements smaller than the pivot element move to left side of the pivot & all greater element move right side of the pivot.

Finally, the algorithm recursively starts sorting the sub-arrays on left & right of pivot element.

Illustration :-

pivot

(5) 3 2 6 4 1 3 7
i j

A[i] < pivot, i++.

(5) 3 2 6 4 1 3 7
i j

A[i] < pivot, i++

(5) 3 2 6 4 1 3 7
i j

A[i] > pivot, stop i, A[j] > pivot, j--

(5) 3 2 6 4 1 3 7
i j

A[j] < pivot swap A[i] & A[j] j--

(5) 3 2 3 4 1 6 7
i j

Now, A[i] < pivot, i++

(5) 3 2 3 4 1 6 7
i j

Again, A[i] < pivot, i++, Now, i = j, stop. And Replace A[i] or A[j] with pivot element,

$\underbrace{1 \quad 3 \quad 2 \quad 3 \quad 4}_{\text{Array 1}}$ (5) $\underbrace{6 \quad 7}_{\text{Array 2}}$

pivot

And so, on.

Page No.
Date

Algorithm:-

* Algorithm for Partioning:-

```
int partition (left, right, pivot) {
l = left+1;                    /* ∵ first element is pivot */
r = right;
while (l < r)
{
    while ( A[l] < pivot)
        l++;

    while ( r > 0 && r > pivot)
        r--;

    if (l >= r)
            break;
    else
        swap (A[l], A[R])
}
swap (A[left], A[l])          // swapping Pivot Element
}
```

* Algorithm for Quick Sort:-

```
QUICK SORT ( A[], p, r){      Initially p = 1, r = n
if p < r
{
    q = partition (A[], p, r)
    QUICK SORT (A, p, q)
    QUICK SORT (A, q+1, r)
}
}
```

* Recurrence Relation:-

$$\boxed{T(n) = T(q) + T(n-q) + n}$$

Scanned with CamScanner

Page No _____
Date _____

Analysis:-

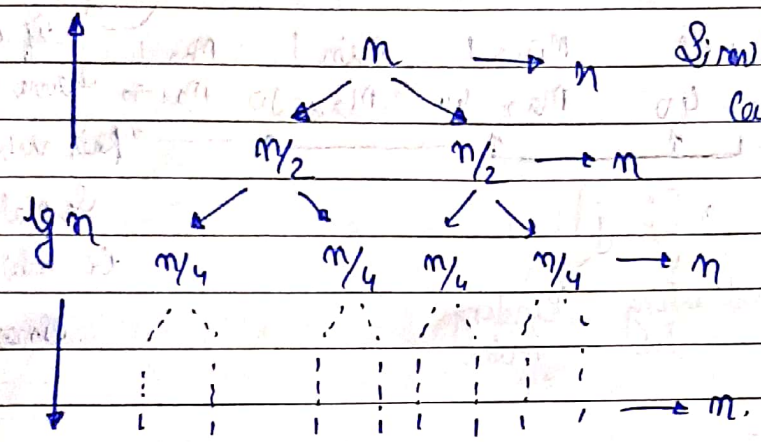
① **Best-Case Performance:** $\Theta(n \log n)$

Partitioning produces two region of size $n/2$.

Recurrence: $q = n/2$

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

$$T(n) = \Theta(n \log n) \quad \text{(by Master's theorem)}$$



Similarly, Average Case = $\Theta(n \log n)$

② **Worst Case Partitioning:-**

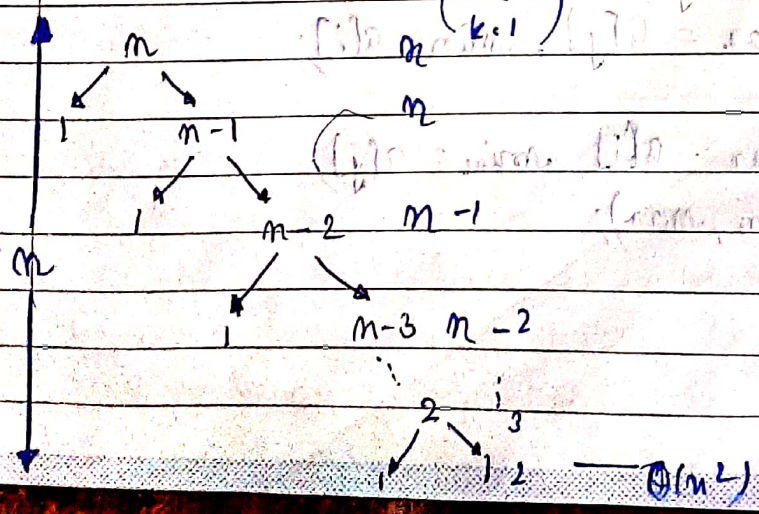
- One region has one element & the other has $(n-1)$ element, Maximumally Unbalanced.
- The worst case occur in Already Sorted Array.

Recurrence,

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

Where, $T(1) = \Theta(1)$

$$= n + \left(\sum_{k=1}^{n} k\right) - 1 = \Theta(n) + \Theta(n^2) + \dots = \Theta(n^2)$$

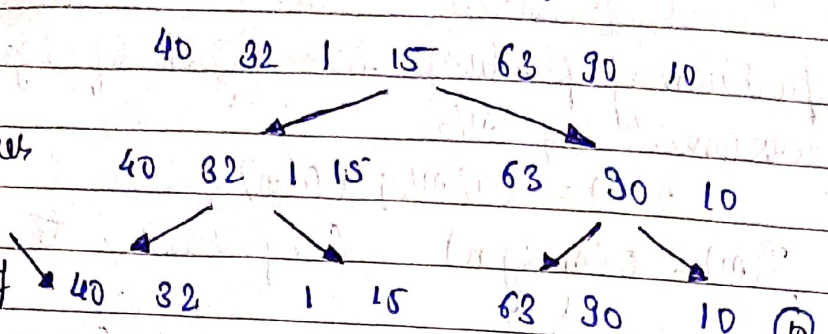


Scanned with CamScanner

Page No.
Date

Min-Max :-

(Using divide & Conquer Algorithm):-



Binary Conquer i.e. till two number array are formed

If we don't have any 2nd pair value, we repeat the digit in order to form Binary Conquer

a[i j]

starting index

Ending index

Algorithm :-

```
DAC-MinMax (a, i, j)
{
    if (i == j)
    {
        max = min = a[i];
        return (min, max);
    }
    if (i == (j-1))
    {
        if (a[i] < a[j])
            max = a[j], min = a[i];
        else
            max = a[i], min = a[j];
        return (min, max);
    }
    else
    {
```

Array Contain One Element

Array Contain two Elements

```
mid = (i+j)/2 ; m = mid;
max1, min1 = DCA_MinMax (a, i, m);
max2, min2 = DCA_MinMax (a, m+1, j);

if (max1 < max2)
        max = max2;
else
        max = max1;
if (min1 < min2)
        min = min1;
else
        min = min2;

return (min, max);
}
}      // End of Functions.
```

* Greedy Algorithm :- greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.
* Working :- Greedy algorithms work by recursively constructing a set of objects from the smallest possible constituent parts.

* Huffman Coding :-

* Huffman Coding :- assigns codes to characters such that the length of the code depends on the relative frequency or weight of the corresponding character.
* Huffman Codes are of variable-length, and prefix-free (no code is prefix of any other). Any prefix-free binary code can be visualized as a binary tree with the encoded

characters stored at the leaves.

* **Huffman Coding tree** :- an Huffman tree is a full binary tree in which each leaf of tree corresponds to a letter in the given alphabet.

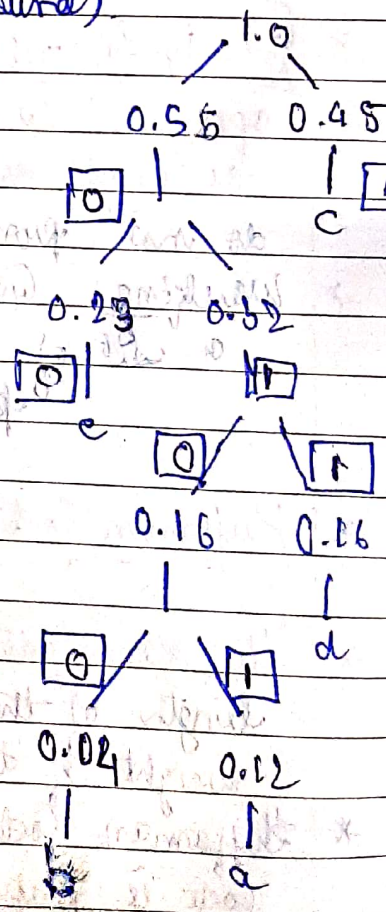
same as optimal merge pattern

Symbols	Frequency OR Probability	bits / symbols Actually	Now →	
* a	0.12	3 bits	0101	4 bits
* b	0.04	3 bits	0100	4 bits
* c	0.45	3 bits	1	1 bits
* d	0.16	3 bits	011	3 bits
* e	0.23	3 bits	00	2 bits

To represent 5 bits, we require minimum of three bits.
→ general, $\lceil \log_2 n \rceil$ (upper bound)
upper bound.

Constructing Prefix Codes:-

Steps :- ① Prepare the frequency table
② Construct the binary tree
③ Extract the Huffman Code



Steps for Construction of tree :-
(i) Left tree node must be the smaller.
(ii) Start from leaf node.
(iii) leaf node is of least frequency
(iv) left node is assigned 0 & right as 1 or vice-versa & hence Two patterns are possible.

Page No.
Date

Checking size:-

Actual:-

$3 \times 0.12 + 0.04 * 3 + 3 * 0.45 + 3 * 0.16 + 3 * 0.23$

$= 0.36 + 0.12 + 1.35 + 0.48 + 0.69 = 3$

Compressed

$4 * 0.12 + 4 * 0.04 + 1 * 0.45 + 3 * 0.16 + 2 * 0.23$

$= 0.48 + 0.16 + 0.45 + 0.48 + 0.46$

$= 2.03$

Average $= \frac{2.03}{3} \times 100 = 67.66\%$

Means A hundred MB file is converted into $67.66 \simeq 68$ MB file.

* Algorithm:-

```
Huffman(C)          // Any character Recieved
{
  n = |C|           // Number of Occurance (frequency)
  for y = 1 to n-1
  {  Allocate a new node z;
     z.left = x = Extract Min(Q);      // x is true
     z.right = y = Extract Min(Q);     // Q is table
                                          (prefix table)
     f(z) = f(x) + f(y);
     Insert(Q, z);
  }
  return (Extract Min(Q));
}
```

Complexity Analysis:-

Time complexity of Huffman algorithm is $O(n \log n)$ where each iteration requires $O(\log n)$ time to determine the cheapest weight & there would be $O(n)$ iteration.

Scanned with CamScanner

Page No.
Date

* Knapsack Problem:-
→ Fill the bag to maximum extent & with maximum cost

Type of Knapsack problem:-

(i) 0,1 Knapsack problem:-
Either bag will completely full or will be not filled with that particular item.

(ii) Fractional Knapsack:-
According to this, if we have some space left in the bag, then we can partially fill the bag with the equipment (some fraction of equipment) if its actual size/weight is greater.

Ques:-

1	2	3	4	5	(Item List)
12	1	2	1	4	weight (in kg)
4	2	2	1	10	cost (in ₹)

Solve above using (i) Fractional & (ii) 0,1 knapsack problem.

Maximum Capacity of sac (Bag) is 15 kg

Solⁿ:-

Step I:- Build table & calculate Value = $\frac{\text{cost}}{\text{weight}}$

N	Weight (kg)	Cost (₹)	Value
1	12	4	4/12 = 0.33
2	1	2	2/1 = 2.0
3	2	2	2/2 = 1.0
4	1	1	1/1 = 1.0
5	4	10	10/4 = 2.5

Step IIⁿᵈ:- Sort in decending order of value

(5)(2.5) > (2)(2.0) > (3)(1.0) > (4)(1.0) > (1)(0.33)

Step IIIʳᵈ:- Start filling in the sac in descending order as to obtained.

Scanned with CamScanner

Page No. _____
Date _____

0,1 knapsac

Cost	Knapsac	Weight
1	n4	1
2	n3	2
2	n2	1
10	n5	4

left, 15-8=7

7 kg left, while n1 is of 12 kg. So, since it is 0,1 knapsac, we can't add fraction of item, so we can't take n1 or discard it.

So, total cost

= 10+2+2+1 = 15 ₹

Fractional knapsac.

12 → 4
1 → 4/12
7 → 4/12 * 7 = 2.3

Cost	Knapsac	Weight
	n1	7
1	n4	1
2	n3	2
2	n2	1
10	n5	4

} 7 kg left (15-8).

7 kg left, while n1 is of 12 kg but since it is a fractional knapsac problem, then so we can use fraction (7th part) of the item n1.

So, cost of 7th part = 4/12 * 7 = 2.3 ₹

So, total cost

= 1+2+2+10+2.3 = 17.3 ₹

Ques (2): * "Note:- Instead of cost, either value, profit or anything can be there, but we have to follow same process & calculate the value"

Item	profit	cost	value
A	280	40	280/40 = 7
B	100	10	100/10 = 10
C	120	20	120/20 = 60/10 = 6
D	120	24	120/24 = 5

Soln:- Fraction knapsac:- capacity = 64

Cost	Knapsac	Weight
6 × 14 = 84	C	14
280	A	40
100	B	10

left space = 64-50 = 14.

total cost = 100+280+84 = 464 ₹.

0,1-knapsac:-

Cost	Knapsac	Weight
	left 14 kg.	D X (discard) C X (discard)
280	A	40
100	B	10

Total cost = 280+100 = 380 ₹

Scanned with CamScanner

Page No.
Date

* Algorithm for knap-sac problem :-
* Approach Used :- Greedy Algorithm
* Defination :- The "Knapsac problem" or "rucksack problem" is a problem in combinational optimization. Given a set of items, each with a weight & a value, determine the number of each item to include in a collection, so that total weight is less than or equal to a given limit & total value is as large as possible.

```
void Greedy_knapsac ( )
{
    for int i = 1 to n
        Compute pi/wt
```

m = Maximum Capacity
p = cost.
n = No. of item.
wi = weight of Material

```
Sort objects in non-increasing order of p/w
                    // O(n log n) → If Merge Sort.
for i = 1 to n {
    if (m > 0 && wi ≤ m)
        m = m - wi
        p = p + pi          // O(n).
    else
        break
    }
if (m > 0)                  // If some fraction left.
    p = p + (pi * m/wi)
}
```

// Algorithm for fractional knapsac problem.

Complexity of Algorithm :- O(n log n)

One More Concept of Divide & Conquer: & Greedy Algorithm
"Matrix Multiplication"

→ General Approach

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

* $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
* $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
* $C_{21} = A_{21}B_{11} + A_{22}B_{22}$
* $C_{22} = A_{21}B_{12} + A_{22}B_{22}$

total 8 Multiplication
general eqn of recursion
$\Rightarrow 8T(n/2) + T(n^2)$

$\Rightarrow$ Complexity $O(n^3)$.

* Greedy Algorithm:-
  ↳ (i) divide:- Find the best among
  (ii) divided: Among them find best

* Strassen's Method:-
  (of Matrix Multiplication):-

$p = (A_{11} + A_{22})(B_{11} + B_{22})$
$q = (A_{21} + A_{22})B_{11}$
$r = A_{11}(B_{12} - B_{22})$
$s = A_{22}(B_{21} - B_{11})$
$t = (A_{11} + A_{12})B_{22}$
$u = (A_{21} - A_{11})(B_{11} + B_{12})$
$v = (A_{12} - A_{22})(B_{21} + B_{22})$

Total number of Multiplication = 7.
Recurrence Relation
$7T(n/2) + T(n^2)$

Complexity = $O(n^{2.8})$

$C_{11} = p + s - t + v$
$C_{12} = r + t$
$C_{21} = q + s$
$C_{22} = p + r - q + u$.

$O(n^{2.8}) < O(n^3)$

Hence, at higher level, Strassen's Multiplication is useful

Scanned with CamScanner

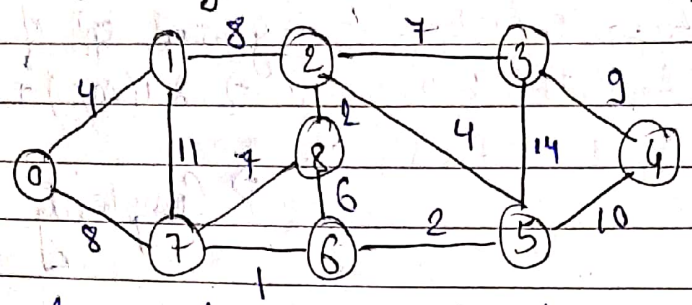
Page No.
Date

Concept of Minimum Spanning Tree:-

Minimum Spanning Tree (MST)

Kruskal's Algorithm — Prime's Algorithm

Kruskal's Algorithm:- (Static Approach)

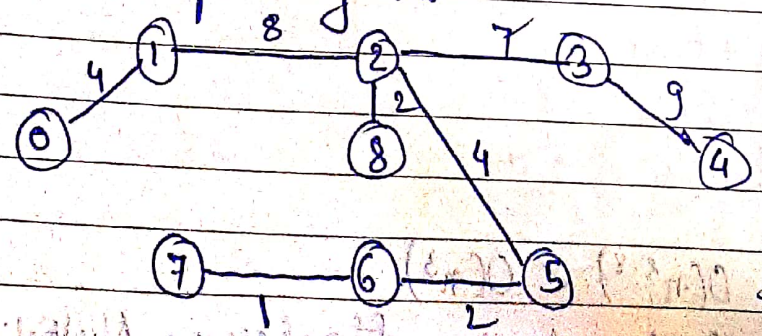


Condn:- (of MST)
- Vertices Require = 9
- Edges Require = 8
- Cycle Avoided.

Step (1):- Weight According to Ascending Order

Source	Destination	Weight	Status
7	6	1	included
6	5	2	included
2	8	2	included
0	1	4	included
2	5	4	included
8	6	6	X (∵ Circle Form)
7	8	7	X (∵ Circle Form)
2	3	7	included
1	2	8	included
3	4	9	Excluded X (∵ circle form)
5	4	10	included X (∵ circle form)
1	7	11	X (Excl ∵ circle)
3	5	14	X (Excluded ∵ circle form)

Minimum Spanning Tree:-



Cost of MST
= Sum of all weights
= 4 + 8 + 2 + 4 + 7 + 9 + 1 + 2
= 37

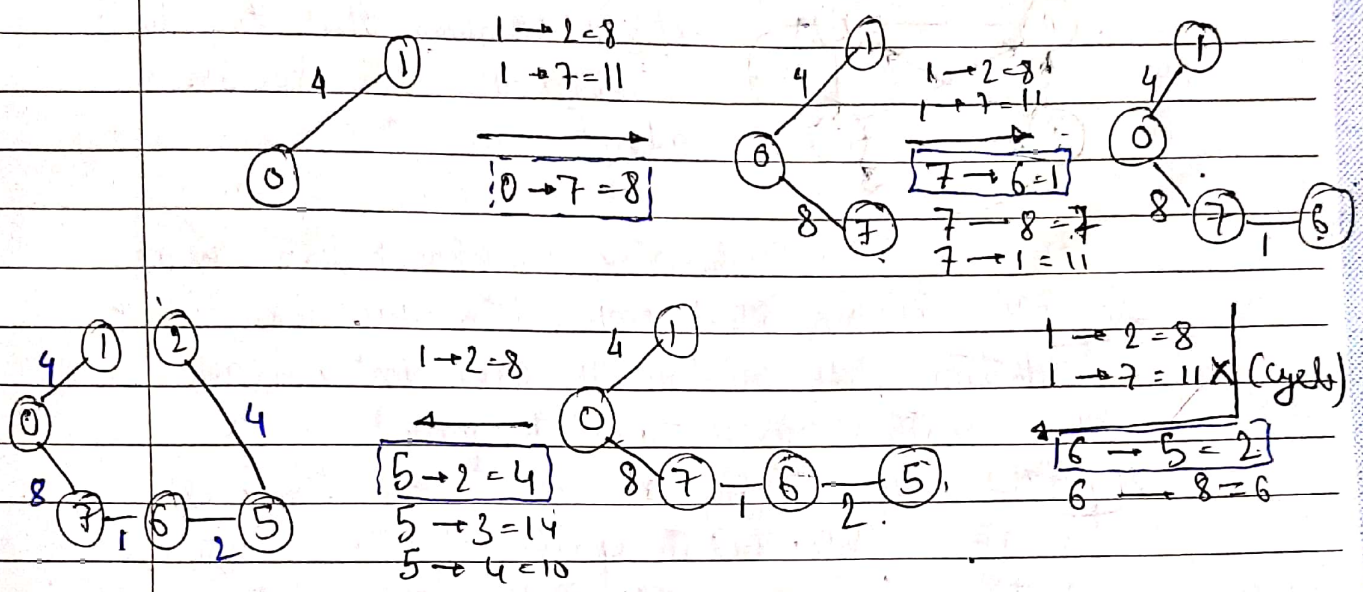
Scanned with CamScanner

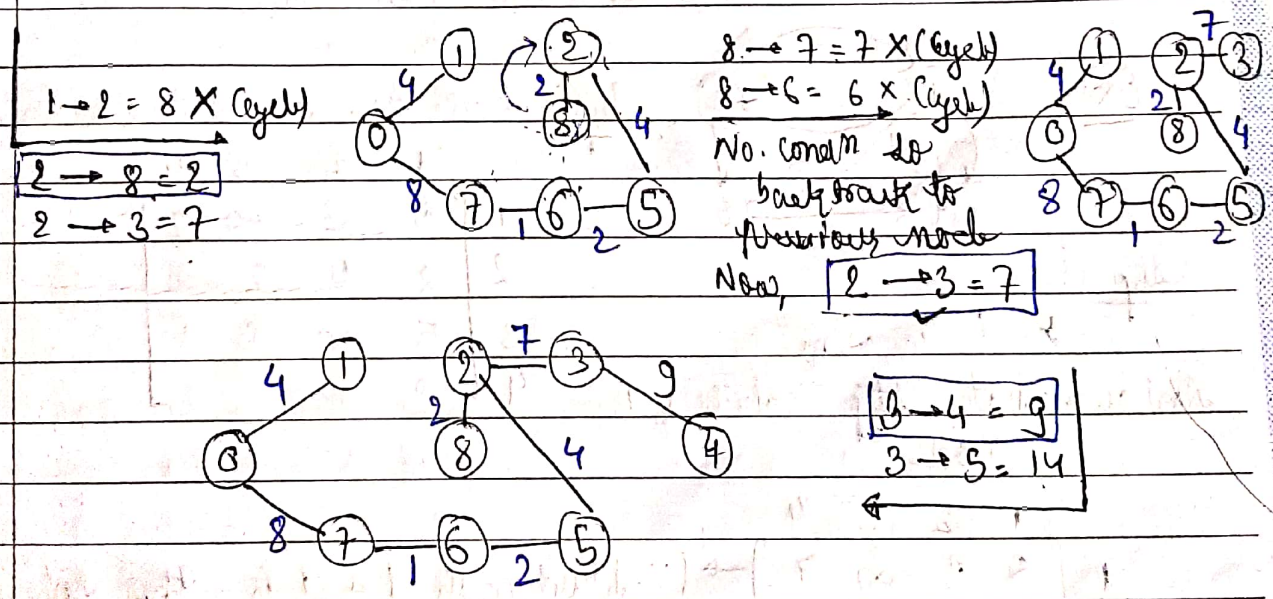
Page No. _____
Date _____

* Prims Algorithm:-

Note:- * Kruskal is pure greedy algorithm & follow static approach.
* Prims Algorithm follow dynamic approach

* In this initial time complexity is O.
* Always check for open edges & go from left to right or from one end.





total vertices = 9; total edges = 8, stop (Completed MST)

Now, cost of MST

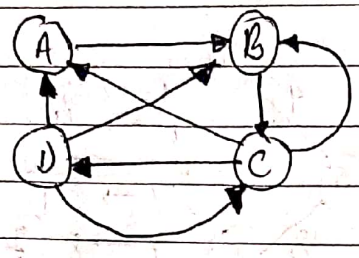
$= 4+8+1+2+4+2+7+9 = 37$

All Pair Shortest path problem:-

problem:- the problem is to find the shortest distances between every pair of vertices in a given edge weighted directed graph.

Floyd Warshall Algorithm:-

the floyd Warshall Algorithm is for solving all pair shortest path problem.



Step I:- Assume that the direct edges are the minimum edges.

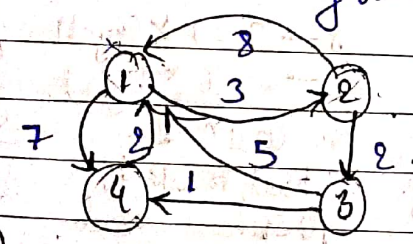
$k = 0$

where, k = intermediate edge

Step IInd: If no direct path, then we have ∞ distance, so we have to take intermediate path (one intermediate path)

Step IIInd:- After taking all vertices one time as an intermediate path, the final obtained matrix give shortest path between two vertices.

Ex:-



Step ①:- $k = 0$ (i.e. no intermediate path):

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

Step ②:-

$k = 1$ i.e. one intermediate step which is ① node

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & (8+7) \\ 3 & 5 & (5+3) & 0 & 1 \\ 4 & 2 & (2+3) & \infty & 0 \end{array}$$

→ (i.e. start with 1 & 1 is the intermediate)

→ $\{ 2 \to 3 = 2$, $2 \to 1 \to 3 = 8 + \infty$ $\}$ ⇒ $(2, \infty)$ Min = 2.

↓

Intermediate 1 Step

Page No _____
Date _____

Step 3:- k=2 as an intermediate:-

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

→ take k=1 as (because optimization intermediate take place at every stage)

↳ take 2 as an intermediate

Step 4:- k=3 as an intermediate

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

Step 5:- k=4 as intermediate

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

Minimum Required Matrix

So, Nearest to 1 → is 3 (i.e. =3)
& Nearest to 2 → 3 (i.e. = 2)
& Nearest to 3 → 4 (i.e. = 1)
& Nearest to 4 → 1 (i.e. = 2)

* **Note:-** General Eqn:- $d^k(i,j) = \begin{cases} w_{ij} & k=0 \\ \min(d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)) & \text{if } k \geq 1 \end{cases}$

* **Algorithm:-**

```
void FloydMarshal (int graph[ ][V])
for (k=0; k<V; k++)
{                                   /* Complexity = O(n^3)
  for (i=0; i<V; i++)
  {
    for (j=0; j<V; j++)
    {
      d^k(i,j) = { w_ij                          k=0
                 { min(d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j))   if k≥1
    }
  }
}
```

Scanned with CamScanner

Page No.
Date

Bellman Ford Algorithm
(Dynamic Approach):-

Defination:- Bellman Ford Algorithm help us find the shortest path from a vertex to all other vertices of a weighted graph.

Note* It is similar to Dijkstra's Algorithm but it can work with graphs in which edges can have negative weights.

* Negative Weight in Real life:-
Negative edges explain a lot of phenomena like cashflow, heat released/absorbed in a chemical reaction etc.
For instance, if there are different ways to reach from one chemical A to another chemical B each method will have sub-reaction involving both, heat dissipation & absorption.
If we want to find the set of reactions where minimum energy is required, then we will need to be able to factor in the heat absorption as negative weights & heat dissipation as positive weights

* Why we need to be careful with negative weights:-
Negative weight edges can create negative weight cycles i.e., a cycle which will reduce the total path distance by coming back to the same point.
Shortest path algorithms like Dijkstra's Algorithm that are not able to detect such a cycle can go through a negative weight cycle & reduce the path length.

Scanned with CamScanner

Page No. _____
Date _____

* How Bellman Ford Algorithm Works:-

Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices.
Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated path.
By doing this repeatedly for all vertices, we are able to guarantee that the end result is optimized.

* Note:- (i) if not given, take "leftmost" node as the source node (first source node)

(ii) Initially, source node is assigned as distance 0 & all other node distance from source node is considered as infinity.

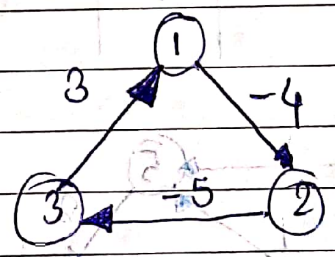
(iii) Cost Calculated as

$$\boxed{d(u) + c(u,v)}$$

d → distination c → cost u → source (initial) v → Final

(imp) (iv) It doesn't applied for the -ve cycle i.e for those cycle whose end value is -ve.

Ex:-

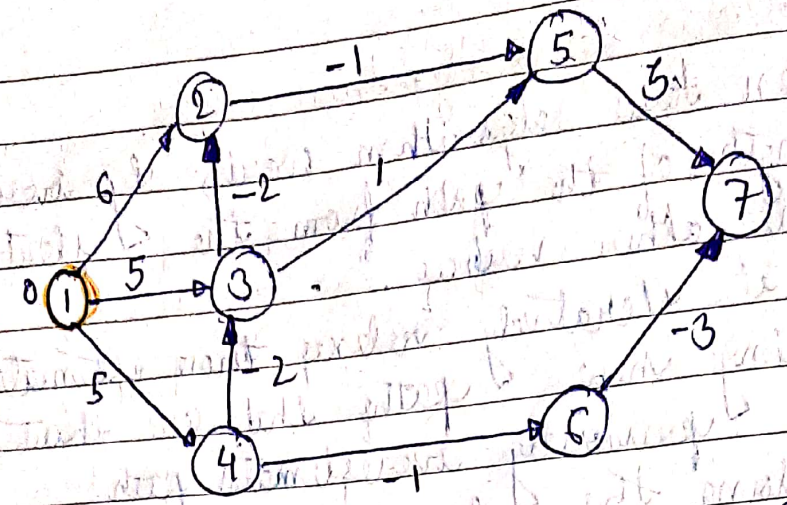


total cost/weight
= 3 - 4 - 5
= -6 i.e.

cycle is -ve so, Bellman Ford is not applicable to this.

Scanned with CamScanner

Example:-



Apply Bellman Algorithm to node ①.

Soln:- table:-

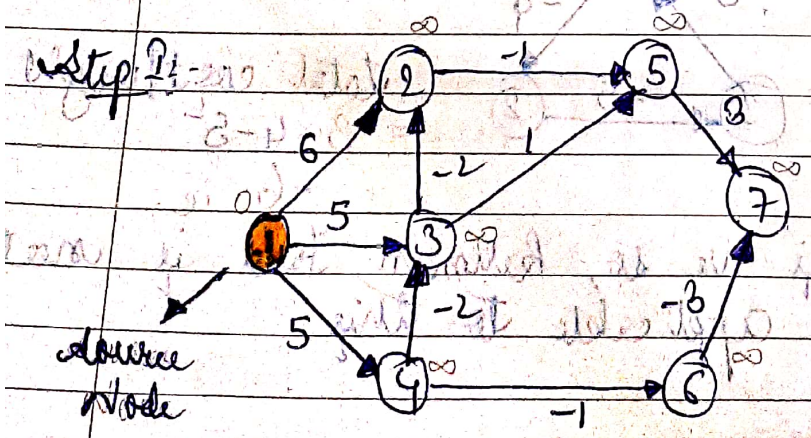
Source	Destination	(1)	(2)	(3)	(4)	(5)	(6)
					← max^m 6 step require		
1	2	6	3	1	1		
1	3	5	5	3	3		
1	4	5	5	5	5		
3	2	3	1	1	1		
4	3	3	3	3	3		
2	5	0	0	0	0		
3	5	0	0	0	0		
4	6	4	4	4	4		
6	7	1	1	1	1		
5	7	1	1	1	1		

← edges to

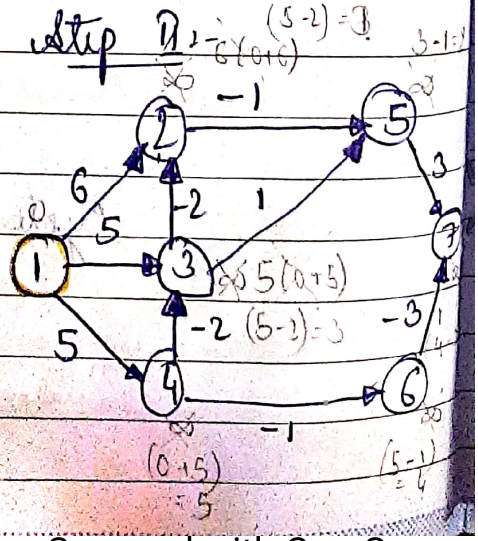
→ done as the sequence is repeated.

→ Required shortest distance from vertex ①.

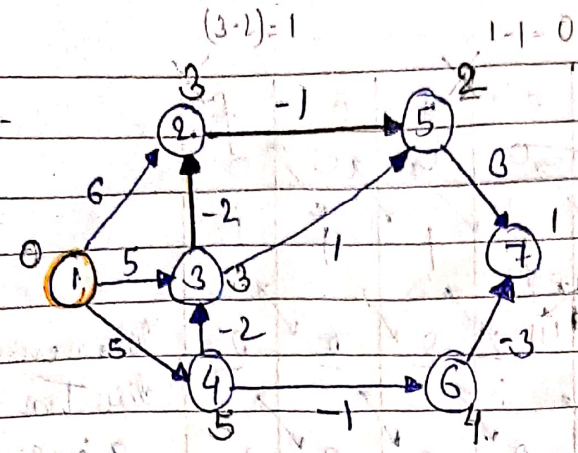
Step I:-



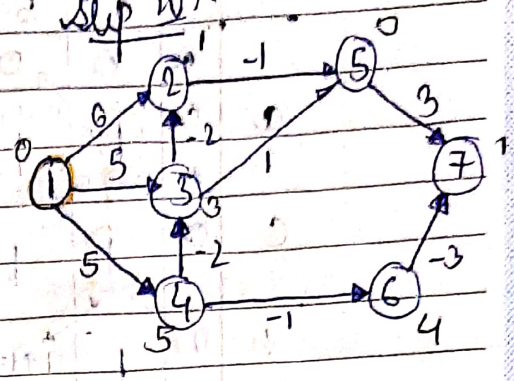
Step II:-



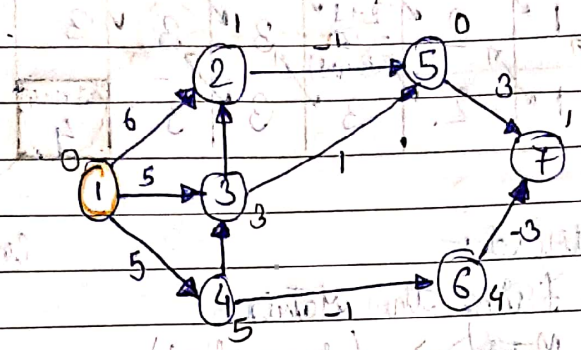
Step III:-



Step IV:-



Step V:-



* Longest Common Subsequence:- (dynamic Approach).

LCS Problem Statement:-

→ Given two sequences, find the length of longest subsequence present in both of them.

→ A subsequence is a sequence that present in same relative order, but not necessarily contiguous.

→ For example, "abc", "abg", "bdf", "ag", "acefg" ... are the subsequence of "abcdefg".

So, a string of length n have 2^n different possible subsequences.

* Ex:- let sub string ① af a b c d e f
& substring ② cf. a c b c f
common

Longest possible substring:- a b c d e f

a c b e f

So, lcs = "abcf"

(not allowed ∵ crossed)

Solution:-

		a	b	c	d	e	f
	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	1+1=2	2	2	2
b	0	1	1+1=2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	3	3	4

Whenever direction change a, b, c, f.

Back-track

length of longest common subsequence

Step followed:-

(1) Filling the Matrix:-

if Match → (diagonal +1)

if different → (other Diagonal) Max.

While doing backtracking,

(i) go diagonally when the change in no. take place & go for its origin

(ii) Finally, origin of the each no. is the required generated subsequence

Origin of 1 → aa
Origin of 2 → bb
Origin of 3 → cc
Origin of 4 → ff.

} Required longest subsequence = "abcf".

* Time Complexity of Algorithm:-

Time complexity of Algorithm (given at next page) is O(mn) which is better than usual case time complexity of Naive recursive implementation

Page No. _____
Date _____

Algorithm:-

```
int lcs (char * X, char * Y, int m, int n)
{
int l[m+1][n+1];
int i, j;
for (i=0; i<=m; i++)
{
    for (j=0; j<=n; j++)
    {
        if (i==0 || j==0)
            l[i][j] = 0;
        else
            if (X[i] == Y[j])
            {
                l[i][j] = l[i-1][j-1] + 1;
            }
        else
            {
                l[i][j] = max (l[i-1][j], l[i][j-1]);
            }
    }
return l[m][n]; // size of Max^m subsequence
}
```

constraint :-
X → substring 1
Y → substring 2
m → size of substring 1
n → size of string 2

* Travelling Salesman Problem :- (TSP):-
(dynamic Approach) :-

Statement :- given a set of cities & distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once & returns to the starting point.

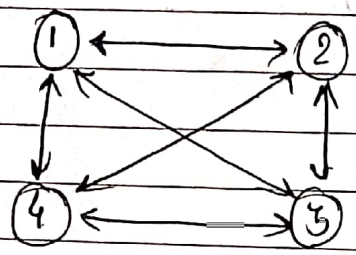
Page No.
Date

difference b/w Hamiltonian Cycle & TSP:-

The Hamiltonian cycle problem is to find if there exist a tour that visits every city exactly once.

But in TSP, we know that Hamiltonian tour exists (because graph is complete), & in fact many such tour exist but the problem is to find minimum weight of Hamiltonian cycle.

Ex:- Given graph as:- & direct route matrix as:-



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

Find the path with minimum weight using TSP starting point be 1.

Ans:-

General Form:-

$$g(i, S) = \min_{k \in S} \{ i_k + g(k, S - \{k\}) \}$$

Where, i = starting index
S = Remaining set
$g(\ )$ = general function.

Step 1:- $g(1, \{2,3,4\}) = \min\{ c_{1k} + g(k, \{2,3,4\} - \{k\}) \}$

$$= \min \begin{cases} c_{12}^{10} + g(2, \{3,4\})^{25} = 35 \\ c_{13}^{15} + g(3, \{2,4\})^{25} = 40 \\ c_{14}^{20} + g(4, \{2,3\})^{23} = 43 \end{cases}$$

Min

$\Rightarrow 10 + 25 = 35$

Scanned with CamScanner

Page No. _____
Date _____

Step 2:- $g(2, \{3,4\}) = \min \begin{cases} C_{23} + g(3, \{4\}) = 29 \\ C_{24} + g(4, \{3\}) = 25 \end{cases}$

Min = 25

Step 3:- $g(3, \{2,4\}) = \min \begin{cases} C_{32} + g(2, \{4\}) = 31 \\ C_{34} + g(4, \{2\}) = 25 \end{cases}$

Min = 25

Step 4:- $g(4, \{2,3\}) = \min \begin{cases} C_{42} + g(2, \{3\}) = 23 \\ C_{43} + g(3, \{2\}) = 23 \end{cases}$

Min = 23

$g(3, \{4, \phi\}) = C_{34} + g(4, \phi) = 12 + 8 = 20$

$g(4, \{3, \phi\}) = C_{43} + g(3, \phi) = 9 + 6 = 15$

$g(2, \{4, \phi\}) = C_{24} + g(4, \phi) = 10 + 8 = 18$

$g(4, \{2, \phi\}) = C_{42} + g(2, \phi) = 8 + 5 = 13$ (min)

$g(2, \{3, \phi\}) = C_{23} + g(3, \phi) = 9 + 6 = 15$

$g(3, \{2, \phi\}) = C_{32} + g(2, \phi) = 13 + 5 = 18$

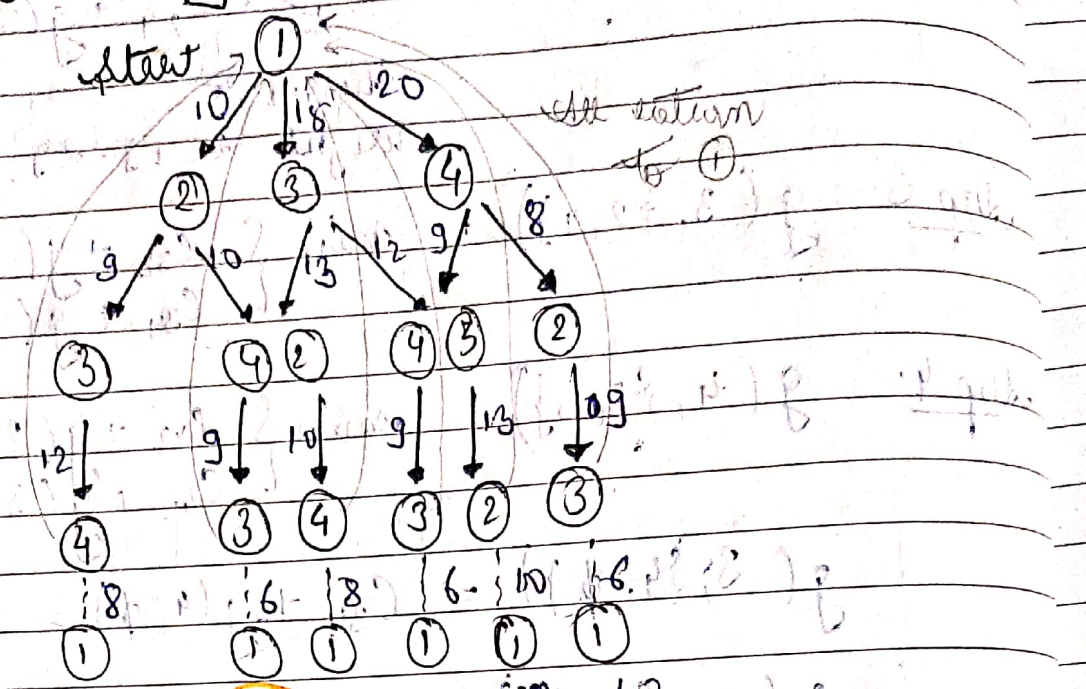
So, minimum weight is = 35.
& path to be followed is

$$1 \xrightarrow{10} 2 \xrightarrow{10} 4 \xrightarrow{9} 3 \xrightarrow{6} 1$$

Weight = 10+10+9+6 = 35

is required Minimum or shortest path for the salesman.

generalized using tree :- (Branch & Bound) :-



P:- possible route (2, 3, 4)

⋮ similarly all

Total = 39 = 35 = 46 = 42 = 52 = 43

→ Minimum among all

So required path is 1 → 2 → 4 → 3 → 1 (10, 10, 9, 6)

KNAPSAC PROBLEM

- Greedy → Fractional Knapsac
- Dynamic Programming → 0-1 Knapsac problem
- Back And Bound → 0-1 Knapsac problem

→ 0-1 Knapsac Problem (Using Dynamic Programming) :-

Benifit (DP over general Approach) :- If the bag size reduces, still we can find the max profit one can make.

Page No _____
Date _____

Example:

item	weight	Value
1	1	10
2	2	10
3	5	12
4	4	18

Maximum Capacity of Bag = 6

Capacity of Bag →

Item Available	1	2	3	4	5	6
1	10	10	10	10	10	10
2	10	10	10+10 = 20	20	20	20
3	10	10	20	20	20	10+12 = 22
4	10	10	20	20	28	$\boxed{28}$

Only item 1 is available

Meaning: when capacity of Bag is 1 kg only item 1 can be placed with cost/value 10 (i.e. at this stage only item 1 is available)

Since bag capacity = 3 (i.e. item ① + item ②)
item ① + item ②
(wt 1) (wt 2)
= 3

Max m weight = 6
Max m value = item ① + item ③
(wt 1) (wt 5)
= 10 + 12 = 22

Maximum value obtained

Now, suppose if bag size reduced to 5 then Max m capacity = 28 (last index if Matrix is of size 5 in coloumn)

* Rule to fill the table :-

if wt[i] > wt then
  V[i, w] = V[i-1, w]
else if wt[i] <= wt then
  V[i, w] = max(V[i-1, w], val[i] + V[i-1, w-wt[i]])

where, wtat = total Capacity
i, w => variable for two-d array as item & weight.
V[i][w] required from Matrix & val → value Array wt = weight array

Page No.
Date

Max^m capacity of bag — total item in bag — Value of each item — Weight of each item

* Algorithm :-

```
void knapsack (int W, int n, int val[], int wt[])
int i, j;
int V[n+1][W+1]  // Value table.
for (j=0; j<=W; j++)
{
        V[0][j] = 0;    // Fill row i=0 with value 0
}
for (i=0; i<=n; i++)
{
        V[i][0] = 0;    // Fill column j=0 with value 0
}
// fill the value table
for (i=1; i<=n; i++)
{
    for (j=1; j<=W; j++)
    {
        if (wt[i] <= j)
        {
            V[i][j] = getmax(V[i-1][j], val[i] +
                                + V[i-1][j - wt[i]]);
        }
        else
        {
            V[i][j] = V[i-1][j];
        }
    }
}
cout << "Max^m value" << V[n][W];
}
```

	5	6
2		20
3		12+10

Means if 3 is also taken Max^m weight

+ weight, if 3 is chosen & weight included in new range

* Time Complexity :- $O(n^2)$

Time complexity of the above problem is $O(nW)$, where n is no. of item & W is the maximum capacity of the sac (or bag)

Page No. _____
Date _____

Optimal Merge Pattern:-

→ Merge a set of sorted files of different length into a single sorted file.

→ If number of sorted files are given, there are many ways to merge them into a single sorted file.

→ This merge can be perform pair wise. Hence this type of merging is called as 2-way merge pattern.

Ex:- Let us consider the given file as f_1, f_2, f_3, f_4 and f_5 with 20, 30, 10, 5 and 30 number of elements respectively.

if Merge operation performed in given sequence $f_1 \to f_2 \to f_3 \to f_4 \to f_5$

$m_1 =$ merge $(f_1 \& f_2) \Rightarrow 20 + 30 = 50$.

$m_2 =$ merge $(m_1 \& f_3) \Rightarrow 50 + 10 = 60$

$m_3 =$ merge $(m_2 \& f_4) \Rightarrow 60 + 5 = 65$

$m_4 =$ merge $(m_3 + f_5) \Rightarrow 65 + 30 = 95$

total no. of operation $= 270$

Using optimal Merge Pattern:-

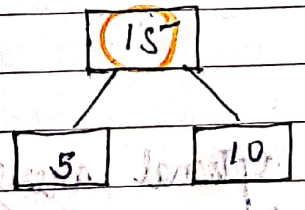
initial set :- $\boxed{5}_{f_4}$, $\boxed{10}_{f_3}$, $\boxed{20}_{f_1}$, $\boxed{30}_{f_2}$, $\boxed{30}_{f_5}$

→ left 20, 30, 30

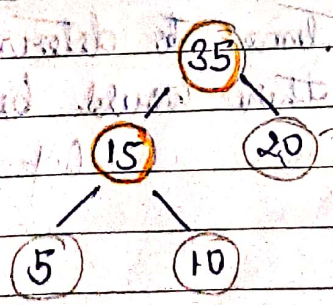
→ taken two at a time

→ place smaller one as left child and larger as right child.

Step 1:-



Step 2:-



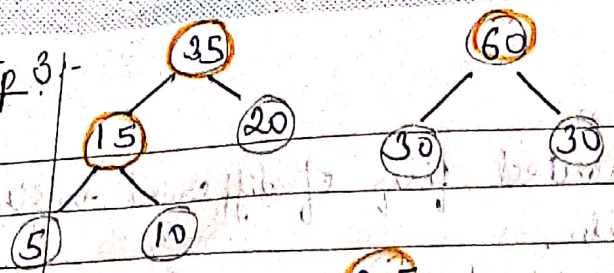
→ left 30, 30

→ ∵ greater than 15, so placed at Right sub-child)

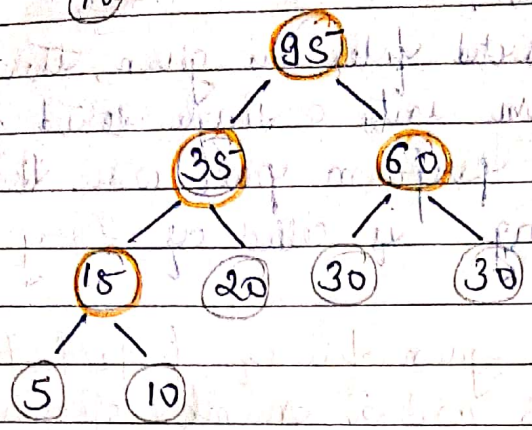
15, 20 → 20, 30

Smaller

Step 3:-



Step 4:-



Hence, the solution takes as
$15 + 35 + 95 + 60$
$= 205$

So, no of operation $= 205$.

$$\% \text{ value optimized} = \frac{205}{270} \times 100 = 75.92\%$$

→ Algorithm :-

```
Tree (n)
  for i = 1 to n-1 do
      declare new node
          node.leftchild := least (list)
          node.rightchild := least (list)
          node.weight := ((node.(leftchild).weight)
                          +((node.rightchild).weight))
      insert (list, node)
  return least (list);
```

Complexity :-

time complexity of optimal merge pattern is $O(n \log n)$ where each iteration require $O(\log n)$ time to determine the cheapest weight & there would be $O(n)$ iteration.

(if not sorted)

Page No. _____
Date _____

Sum of Subsets Problem :-

Problem :- Given a set of non-negative integer, & a value sum, determine if there is a subset of given set with sum equal to given sum.

Dynamic Approach :-

Ex :- total sum = 11 set = {2, 3, 7, 8, 10}

sum →

set ↓	0	1	2	3	4	5	6	7	8	9	10	11
2	T	F	T	F	F	F	F	F	F	F	F	F
3	T	F	T	T	F	T	F	F	F	F	F	F
7	T	F	T	T	F	T	F	T	F	T	T	F
8	T	F	T	T	F	T	F	T	T	T	F	T
10	T	F	T	T	F	T	F	T	T	T	T	T

subset = {2, 8}

Means it is possible.

Note :- ① Repetition is not allowed in sets

② Go to (n, n) [After first row completed] & mark it as true & copy to the left side the value of (n-1) row.

③ For other nodes (on RHS of (n, n) node), move n step backward from (n-1) row & that will be the value

if false above

④ If above is true, then copy as it is true to down.

* Finding that subset :-

{2, 3} → For these two, we have to use backtrack.

Drawback :-

don't tell about the state (all possible sets)

Advantage :-

title 11, all sum can check if possible or not.

* Complexity Analysis:-
the algorithm runs in $O(2^n)$ time, because in the worst case, we need to create every possible subset of array to check if any of them will be equal to the goal sum, & there are 2^n possible sets.

Using Branch & Bound:-

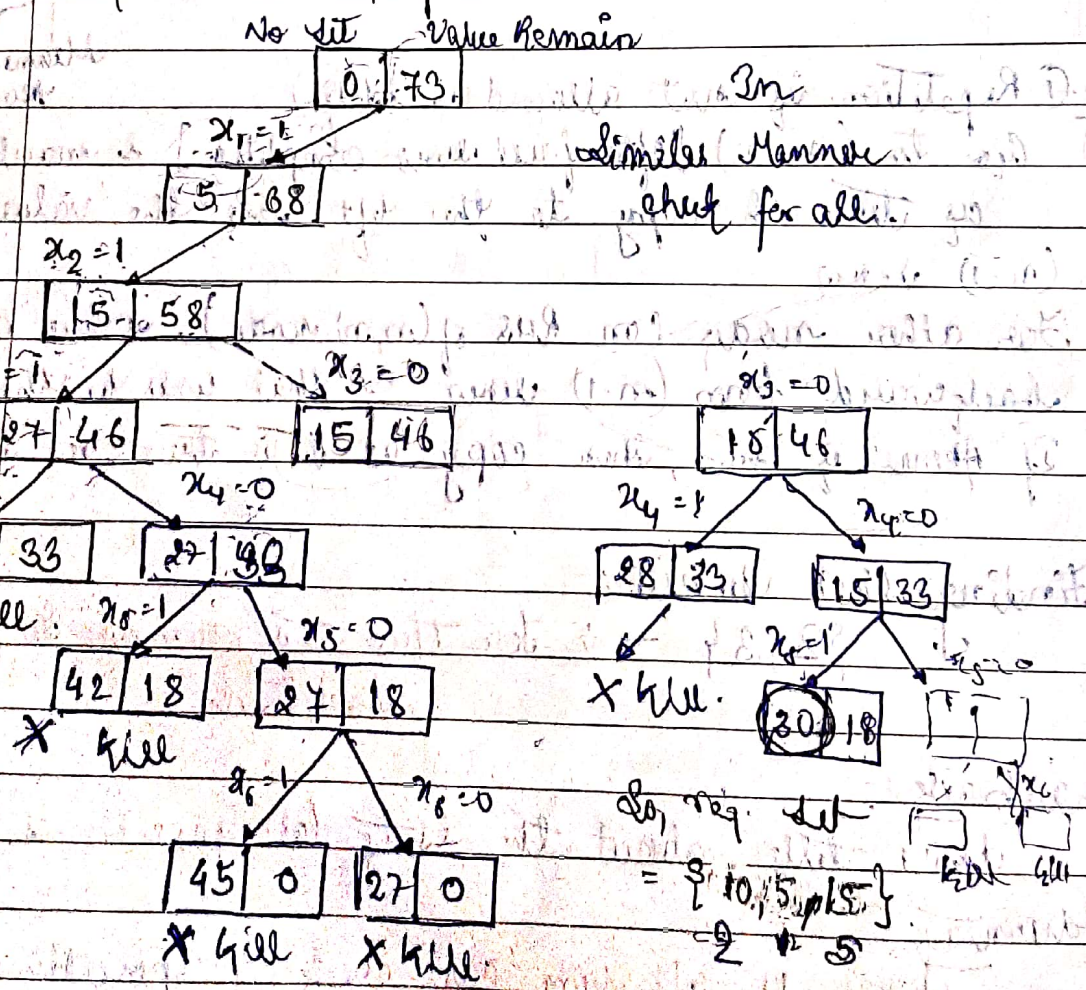
$W[1:6] = \{5, 10, 12, 13, 15, 18\}$

Max^m size = 6.

Max^m sum = 30 = m

1	2	3	4	5	6
5	10	12	13	15	18

0 → Discard 1 → Accept



In similar manner check for all.

So, req. set = {10, 5, 15}
2 1 3

Scanned with CamScanner

Page No. _____
Date _____

Application of sum of subset Problem :-
"Coin Change Algorithm"

Problem :- Is it possible to get exactly that amount of coin that are given by the user.

Condn :- Supply of the given coin is infinite.

Ex :- Coin Available = 1, 5, 6, 8
total Money need = 11.

Total Money :-

Coins Available		0	1	2	3	4	5	6	7	8	9	10	11
	1	0	1	2	3	4	5	6	7	8	9	10	11
	5	0	1	2	3	4	1	2	3	4	5	2	3
How many	6	0	1	2	3	4	1	1	2	3	4	2	3
	8	0	1	2	3	4	1	1	2	1	2	2	2

coins are required.

Minimum no. of two coins needed to make required money.

Check 5 back + we get 4+1=5 but if we get a smaller 2 or above 2, why not the greater

One of possibility is :- {5, 6}
since both of them come from backtracking.

Steps :-

(i) First row filled accordingly that how many coins are required to do above.

(ii) (m, n) is filled with 1 & then,

RHS of (m, n) :-

x = Value (backtrack or backward in same row) + 1

Compare this with Above Value (say y) (row above it)

if (x > y { x : y).

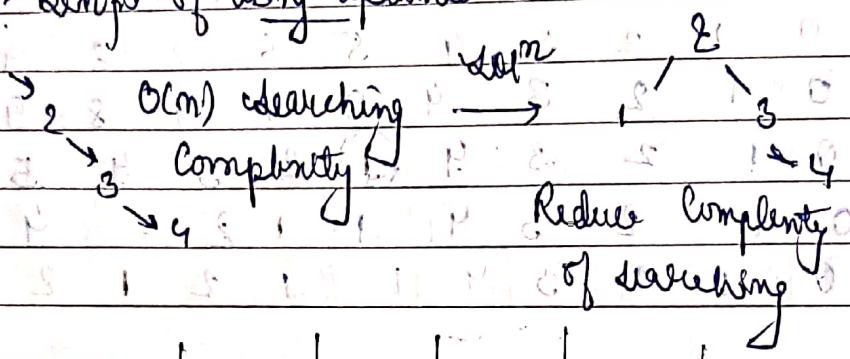
LHS of (m, n)
Copy the above value as it is from above.

Page No.
Date

* Optimal Binary Search tree :-

Problem: Given a sorted array key [0...n-1] of search keys and an array freq [0...n-1] of frequency count, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all searches is as small as possible.

Problem: Benefit of using optimal BST :-



Ex:-

Node No.	0	1	2	3
Node	10	12	16	21
frequency	4	2	6	3

Step 1:- Arrange in Ascending order.

	0	1	2	3
0	4	$8^{[0]}$	$20^{[2]}$	$26^{[2]}$ → final value
1		2	$10^{[2]}$	$16^{[2]}$
2			6	$12^{[2]}$
3				3

superscript = 2.
So, 2 is root node.

Step 2:- Case 1:- when l (length = 1)
index (n, n) → value of (n) or frequency (n)
So, (0,0) → 4 (1,1) → 2 (2,2) → 6 & (3,3) → 3

Step 3:- Case 2:- when length (l) = 2.

Scanned with CamScanner

Page No. _____
Date _____

possible set of two in sequence; = (0,1), (1,2) & (2,3).

(i) for (0,1)

0	1
10	12
4	2

Formula = sum of frequency of both + min { Removed frequency.

$= 4 + 2 + \min\{2 \leftarrow 0,\ 4 \leftarrow 1\}$

$= 4 + 2 + 2 = 8^{[0]}$ because it comes from [0].

(ii) for (1,2)

1	2
12	16
2	6

$= 2 + 6 + \min\{2 \leftarrow 2,\ 6 \leftarrow 1\} = 2 + 6 + 2 = 10^{[2]}$

(iii) for (2,3)

2	3
16	21
6	3

$= 6 + 3 + \min\{6 \leftarrow 3,\ 3 \leftarrow 2\} = 6 + 3 + 3 = 12^{[2]}$

Step IInd: Case IIIrd: when l = 3.

two possible set in sequence

0, 1, 2

0	1	2
10	12	16
4	2	6

Value from Matrix

$= 4 + 2 + 6 + \min\{10 \leftarrow 0\ \{1,2\}$ for i/p$,\ 4 + 6 \leftarrow 1\ \{(0,0)_L + (2,2)_R\}$ left right$,\ 8 \leftarrow 2\ \{(0,1)$ in left$\}\}$

$= 4 + 2 + 6 + 8 = 20^{[2]}$.

from (0,2).

1, 2, 3.

1	2	3
12	16	21
2	6	3

$= 2 + 6 + 3 + \min\{12 \leftarrow 1\ \{2,3\}$ for R.$,\ 9 + 2 \leftarrow 2\ \{(0,0)_L + (2,2)_R\},\ 10 \leftarrow 3\ \{(0,2)_R\}\}$

$= 2 + 6 + 3 + 5$

$= 16^{[2]}$

Scanned with CamScanner

Case IV for $l = 4$ (length):-

0	1	2	3
10	12	16	21
4	2	6	3

$$= 4 + 2 + 6 + 3 + \min \begin{cases} 16 \leftarrow 0 & 1,2,3 \text{ go right } (1,3) \text{ on R} \\ 4+12 \leftarrow 1 & (0,0) \text{ L } \& \ (2,3) \text{ on right} \\ 8+3 \leftarrow \boxed{2} & (0,1) \text{ on L } \& \ (3,3) \text{ on R} \\ 20 \leftarrow 3 & (0,2) \text{ L on left } (0,1,2) \end{cases}$$

$= 4 + 2 + 6 + 3 + 11$

$= 26^{[2]}$.

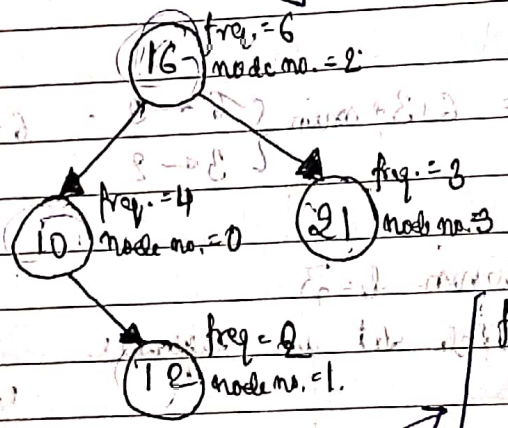
Root Node = subscript of last node

* Creating Optimal Binary Search Tree :-

0	1	2	3 ← Node no.
10	12	16	21 ← value
4	2	6	3 ← freq.
left	Root		Right

0 → 1

subscript is 0 so (0) is root node



	0	1	2	3
0	4	$8^{[0]}$	$20^{[2]}$	$26^{[2]}$
1		2	$10^{[2]}$	$16^{[2]}$
2			6	$12^{[2]}$
3				3

```
for l in 1...n loop
  for i in 1... n-l+1 loop
    j = i+l-1
    for r in i...j loop
```

Complexity :- $O(n^3)$

Algorithm :- Algorithm to find optimal tree for sorted, distinct keys $k_0 \ldots k_j$

Formula :-
$$e(i,j) = \begin{cases} 0, & \text{if } i = j+1 \\ \min_{i \le r \le j} \{ e(i, r-1) + e(r+1, j) + w(i,j) \}, & \text{if } i \le j \end{cases}$$

where $w(i,j) = \sum_{k=1}^{j} p_i$ is increase in cost if $k_i \ldots k_j$ is subtree of node.

Since three loop of i, r & j are used, hence Complexity $= O(n^3)$

Page No. _____
Date _____

* 0-1 Knapsack Using Branch & Bound:-

Ex:-

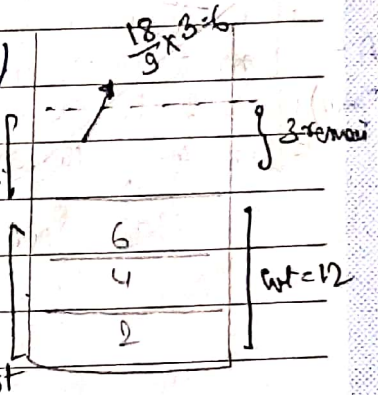
	1	2	3	4
profit	10	10	12	18
weight	2	4	6	9

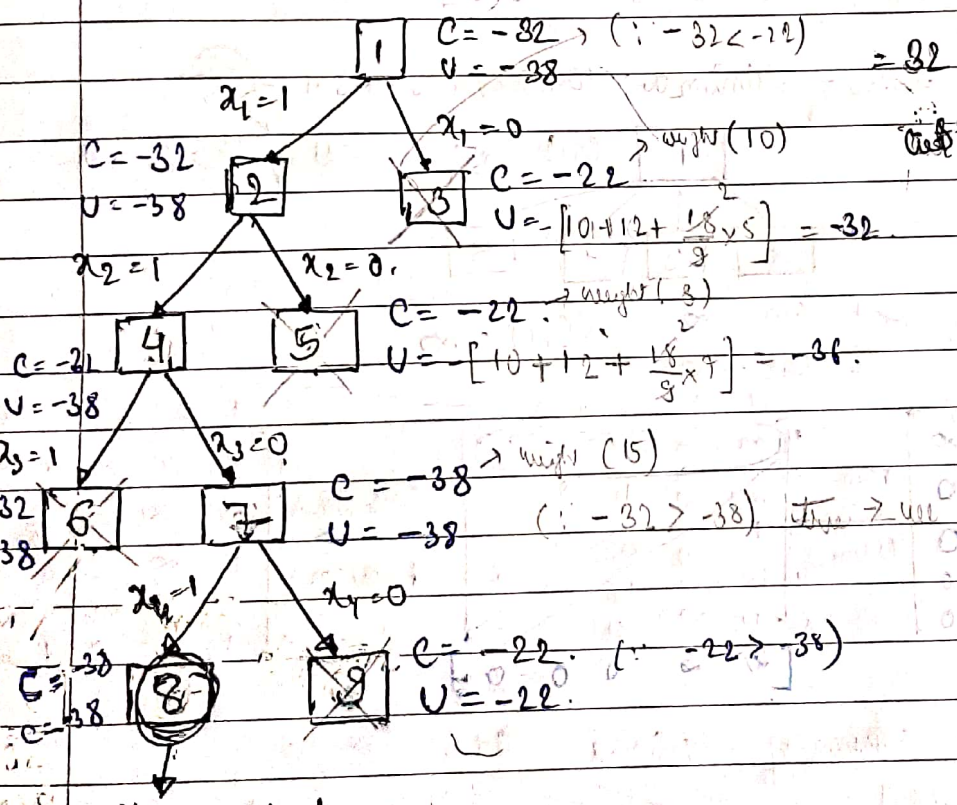
m (bag weight) = 15
$n = 4$ (no. of item.

Suppose above is arrang as increasing order of the value.

Note:- Branch & Bound is a Maximization approach but is a minimization approach. But, knapsack is Maximization approach, so to convert the above, we bring -ve sign with the profit.

C (cost Actual) = -32 (using simple knap-sac)
U (upper bound) = -38 (Max^m Earning)
(using fractional knapsack)





Final Node:-

So, the thief will select item $x_i = 1, 2, 4$.
So, total cost earn = -38
total item earn = 3.

Scanned with CamScanner

Travelling Salesman Problem:-

Problem description:- Travelling salesman problem (TSP) given set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Using Branch & Bound:-

$$A = \text{Matrix} \begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \text{Min} \\ \hline 1 & \infty & 20 & 30 & 10 & 11 & 10 \\ 2 & 15 & \infty & 16 & 4 & 2 & 2 \\ 3 & \infty & 5 & \infty & 2 & 4 & 2 \\ 4 & 3 & 6 & 18 & \infty & 3 & 3 \\ 5 & 16 & 4 & 7 & 16 & \infty & 4 \end{array} \xrightarrow{\text{Row reduce}} \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 10 & 20 & 0 & 1 \\ 2 & 13 & \infty & 14 & 2 & 0 \\ 3 & 1 & 3 & \infty & 0 & 2 \\ 4 & 16 & 3 & 18 & \infty & 0 \\ 5 & 12 & 0 & 3 & 12 & \infty \end{array} \xrightarrow{\text{Column Reduce}} \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 10 & 17 & 0 & 1 \\ 2 & 13 & \infty & 11 & 2 & 0 \\ 3 & 1 & 3 & \infty & 0 & 2 \\ 4 & 15 & 3 & 15 & \infty & 0 \\ 5 & 11 & 0 & 0 & 12 & \infty \end{array}$$

Matrix {A[i] - min} $[1 \quad 0 \quad 3 \quad 0 \quad 0]$ → Min

Sum of Minima = 10 + 2 + 2 + 3 + 4 + 3 + 1 = 25

Cost = 25 , Cost of Matrix = 25

Now, 1 - 2 → 1 - row = ∞ , 2 - col = ∞ , (2,1) = ∞

$$\begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & \infty & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & 11 & 2 & 0 & 0 \\ 3 & 1 & \infty & \infty & 0 & 2 & 0 \\ 4 & 15 & \infty & 12 & \infty & 0 & 0 \\ 5 & 11 & \infty & 0 & 12 & \infty & 0 \end{array} \xrightarrow{\text{Row reduce}} \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & 11 & 2 & 0 \\ 3 & 1 & \infty & \infty & 0 & 2 \\ 4 & 15 & \infty & 12 & \infty & 0 \\ 5 & 11 & \infty & 0 & 12 & \infty \end{array} \xrightarrow{\text{Column Reduce}} \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & 11 & 2 & 0 \\ 3 & 1 & \infty & \infty & 0 & 2 \\ 4 & 15 & \infty & 12 & \infty & 0 \\ 5 & 11 & \infty & 0 & 12 & \infty \end{array}$$

$[\infty \quad \infty \quad 0 \quad 0 \quad 0]$

Sum of Minima = 0 + 0 + ... + 0 = 0

total cost

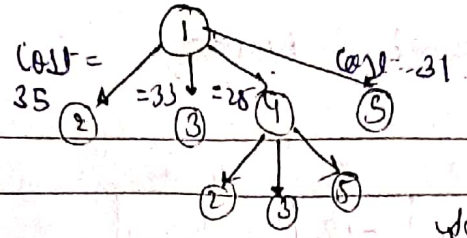
$T(2) = P(1) + A(1,2) + r$ → total reduction cost

P(1) → tentative Matrix Cost

$= 25 + 10 + 0$

$= 35$

Similarly, 1 → 3 , 1 → 4 and 1 → 5.



Page No. _____
Date _____

A = Actual Matrix

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 10 & 17 & 0 & 11 \\ 2 & 12 & \infty & 11 & 2 & 0 \\ 3 & 0 & 3 & \infty & 0 & 2 \\ 4 & 15 & 3 & 12 & \infty & 0 \\ 5 & 11 & 0 & 0 & 12 & \infty \end{array}$$

Matrix obtained is same Matrix

So,

$T(2) = T(4) + A(4,2) + r$

$= 25 + 3 + 0 = \underline{\underline{28}}$

deleted from

$T(2) = T(4) + A(4,2) + r$

4th Reduce Matrix

4th row
2nd - Col } All fill up ∞.

and, (2, 4)

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 12 & \infty & 11 & \infty & 0 \\ 3 & 0 & 3 & \infty & \infty & 2 \\ 4 & 15 & 3 & 12 & \infty & 0 \\ 5 & 11 & 0 & 0 & \infty & \infty \end{array} \quad \begin{array}{c} \infty \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$$

occurs when we want $1 \to 4$,

$1 - row = \infty$

$4 - col = \infty$

$(4,1) = \infty$.

After applying Row reduction & column reduction

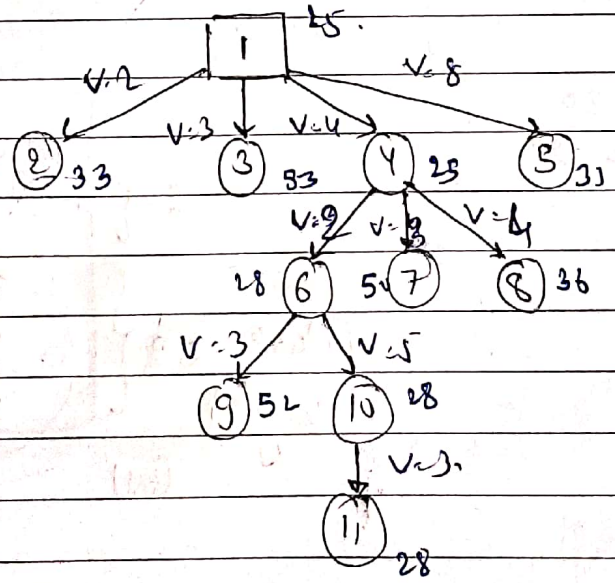
Reduce this

$T(4) = T(1) + A(1,4) + r$

$= 25 + 0 + 0$

$= 25$.

Similarly for all. Finally, the obtained solution will be :-



(Branch & Bound)

Page No.
Date

Knapsac 0-1:-

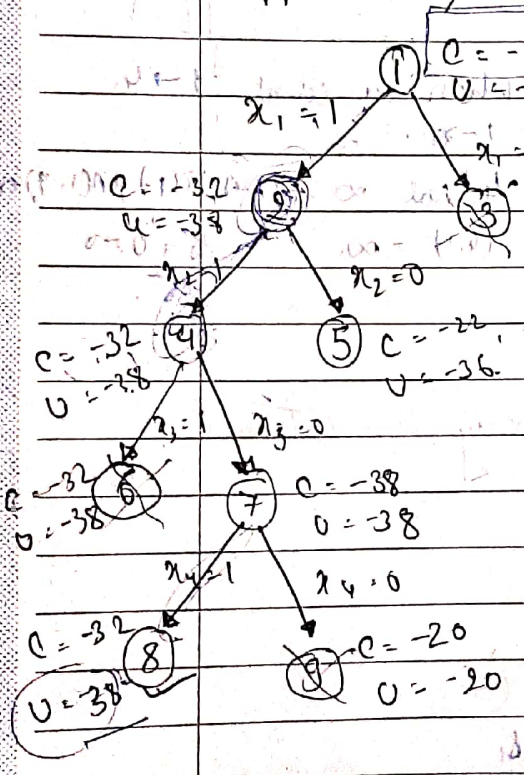
	1	2	3	4
profit	-10	-10	-12	-18
weight	2	4	6	9

m = weight of bag = 18
$n = 4$.

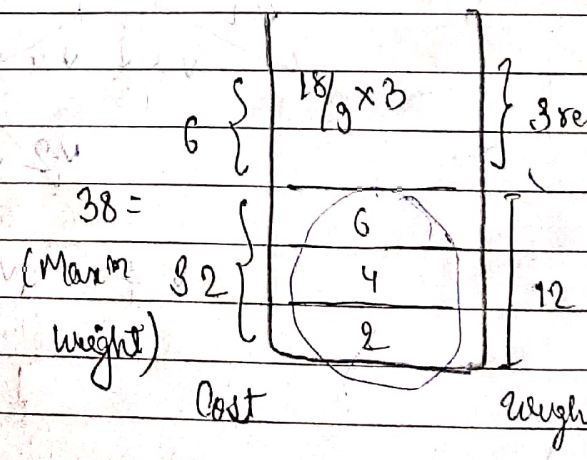
Since branch & bound is a Minimization Approach & here we have to use Maximization Approach.

C (cost) = -32 (using 0,1-knapsac.)

U (upper bound) = -38 (if use fractional knapsac)



So, items to be selected are $\boxed{1 \rightarrow 2 \rightarrow 4}$.



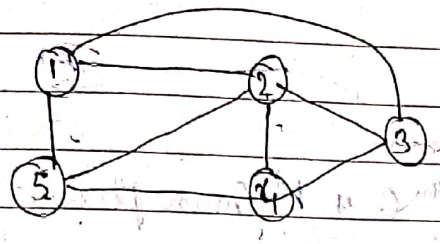
(Backtracking)

Page No. _____
Date _____

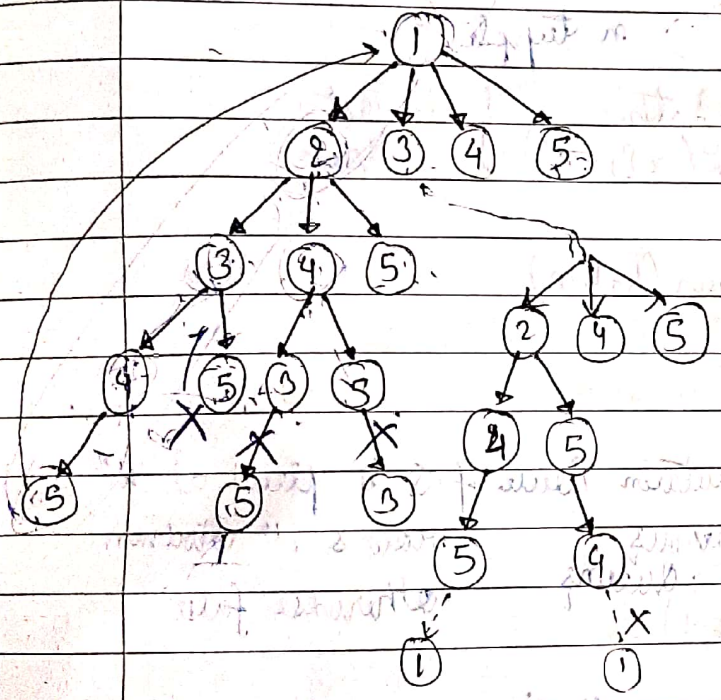
Hamilton Cycle :-

Hamilton path in an undirected graph is a path that visits each vertex exactly once.

A hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path such that there is an edge (in graph) from the last vertex to the first vertex of the hamilton path.



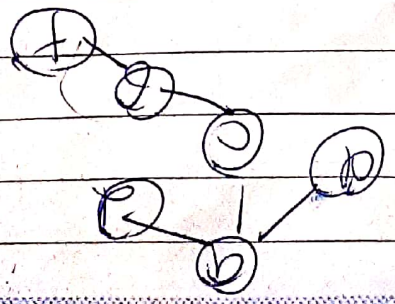
$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$



Similarly evaluate all the cases.

Final solutions are :-

1 → 2 → 3 → 4 → 5 → 1
1 → 2 → 5 → 4 → 3 → 1
1 → 3 → 2 → 4 → 5 → 1
1 → 3 → 4 → 2 → 5 → 1
1 → 3 → 4 → 5 → 2 → 1



Scanned with CamScanner

Page No.
Date

N-Queens Problem:-

The N-Queen is the problem of placing N chess queens on an N*N chessboard so that no two queens attack each other

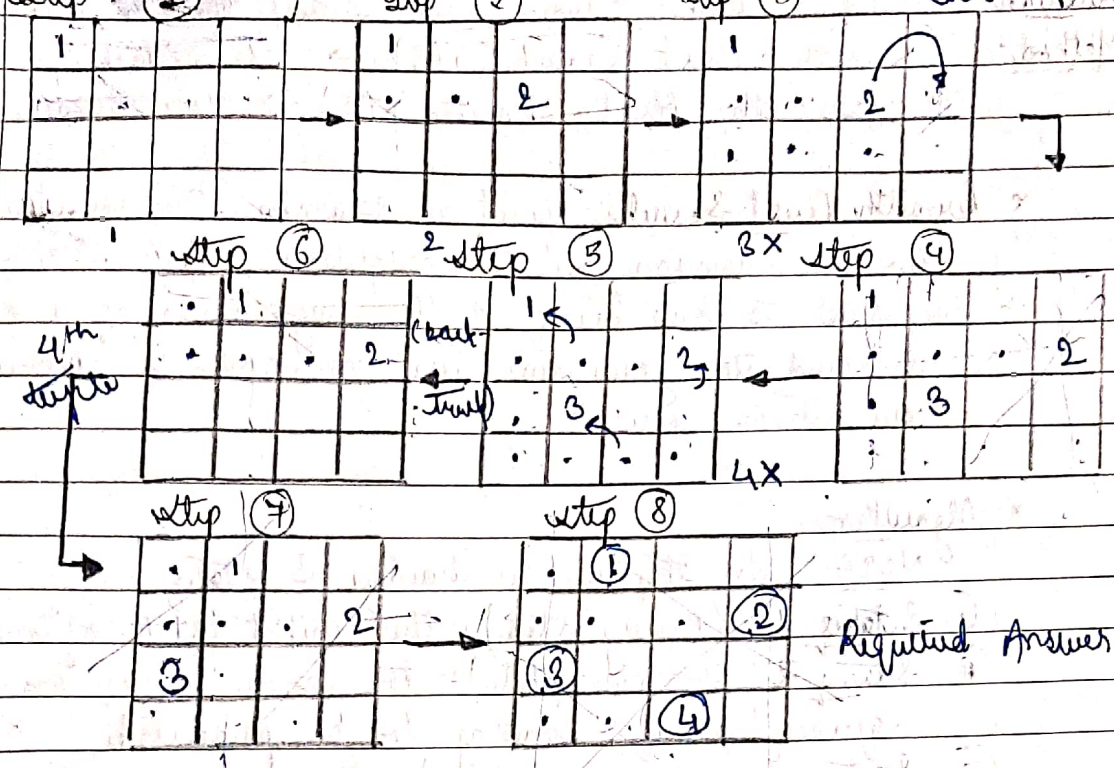
→ **Algorithm:-**

```
                k^th Queen
                  |   |  → Coloumn no
Algo NQueen ((k)(n))
{
    for (i = 1 to n) do
    {                    True if i^th C is k^th Queen place
        if (place (k,i)) then
        {
            x[k] = i        → tuple
            if (k == n) then
                print (x[1:n])   } if last Queen
            else
                NQueen (k+1, n)
        }
    }
}

place (k, i)        // return True if Q is placed at k^th row & i^th coloumn
{                   Previous Queens        otherwise false
    for (j = 1 to k-1) do
    {
        if (x[j] == i)        // in same coloumn
            or (Abs(x[j] - i) = Abs(j-k)) then
            return false      ↳ // in same diagonal
    }
    return true
}
```

Page No _____
Date _____

4-Queen Problem Solved:-

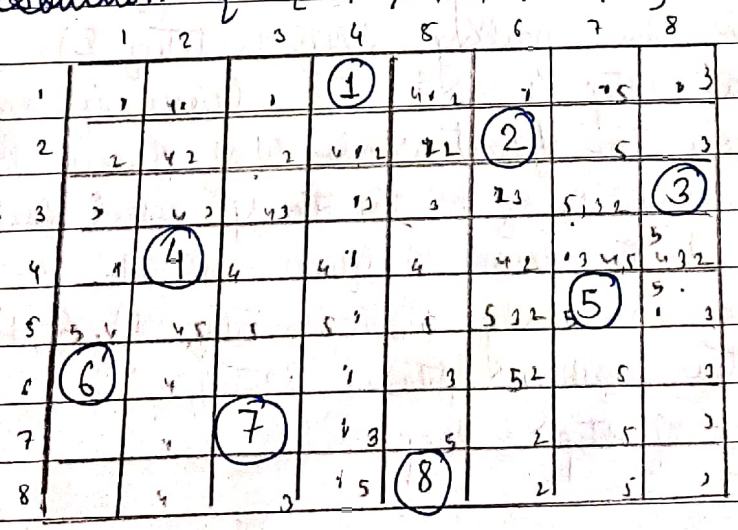


Required Answer

So, Queen must be placed at position (2, 4, 1, 3) at their respective rows.

8-Queen's problem:-

Solution is {4, 6, 8, 2, 7, 1, 3, 5}



Unit III rd :-

Page No.
Date

TRAVERSAL OF GRAPH :-

* Problem :- find a path between two nodes of the graph.
* Method :- (i) Depth - First - Search (DFS) → use Stack
  (ii) Breadth - First - Search (BFS) → use Queue.

* Breadth First Search :- first we examine the starting node. Then we examine all the neighbours of A & so on.
→ We need to keep track of the neighbours of a node & we need to guarantee that no node is processed more than once.

* Algorithm :-
  NOTE :- In this we are having 3 states :-
  (i) States 1 :- (Ready state) :- the initial state of node N.
  (ii) States 2 :- (Waiting state) :- the node 'n' is on the queue (stack) waiting to be processed.
  (iii) States 3 :- (Processed state) :- the node 'n' has been processed.

Algorithm :- (BSF) :- or Level Order Traversal :-
(i) Initialize all nodes to the ready state (Status = 1)
(ii) put the starting node A in Queue & change its status to the waiting state (Status 2).
(iii) Repeat step 4 & 5 until Queue is empty.
(iv) Remove the front node N of Queue. Process N & change the status of N to the processed state (Status 3).
(v) Add to the rear of Queue all the neighbours of N that are in ready state (Status = 1) & change their status to the waiting state (Status = 2).
[End of step 3 loop]
(vi) Exit.

Algorithm (DFS) :- or (Depth Search First) :-
(i) Initialize all nodes to ready state (Status = 1)

Scanned with CamScanner

(ii) Put the starting node A in stack & change its status to waiting state (status = 2)

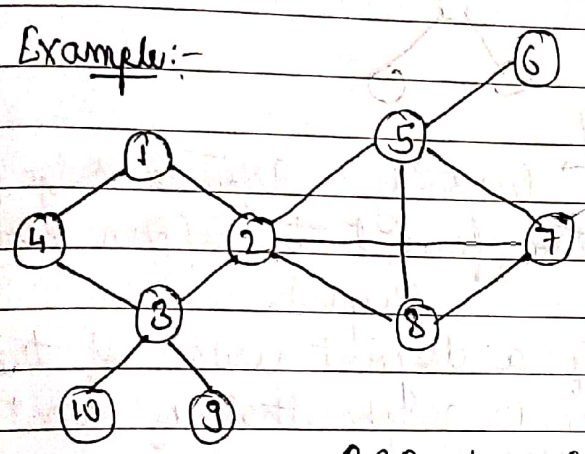
(iii) Repeat steps 4 and 5 until stack is empty.

(iv) Pop the top node N of stack, process N and change the status of N to the processed status (status = 3).

(v) Push onto stack all the neighbours of N that are still in the ready state (status = 1) & change their status to waiting state (status = 2)
[End of step 3 loop]

(vi) Exit.

Example:-



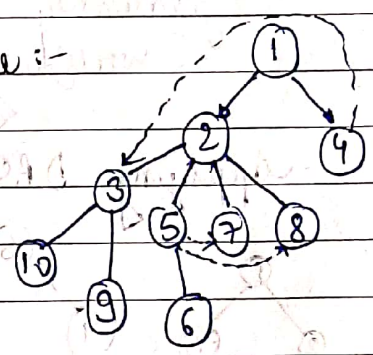
BFS:-

Queue:-

1	2	4	3	5	7	8	10	9	6

Output:-
1 2 4 3 5 7 8 10 9 6

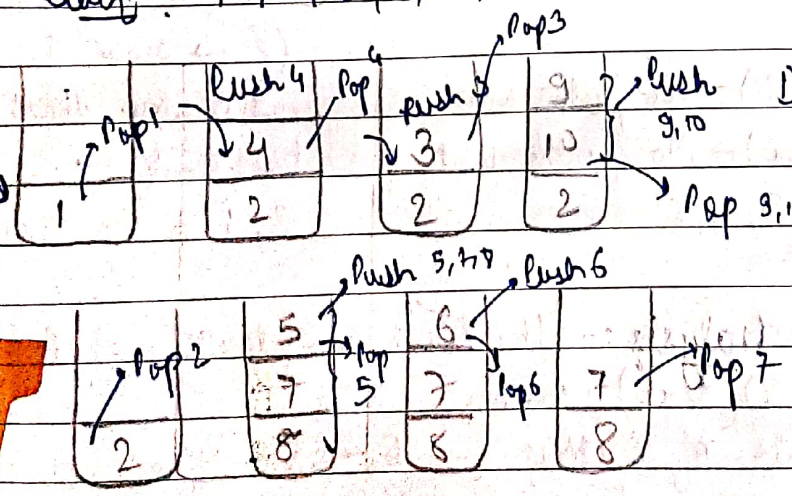
BSF spanning tree:-



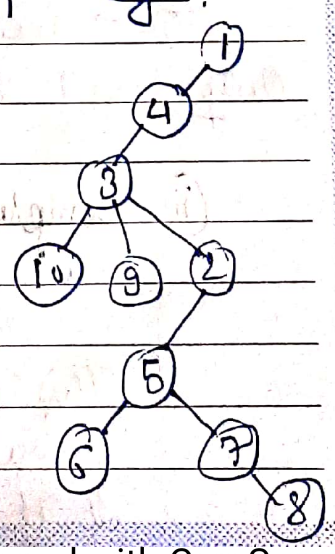
DFS:-

Output:- 1, 4, 3, 9, 10, 2, 5, 6, 7, 8

Stack:- 1, 4, 2, 3, 9, 10, 5, 7, 6

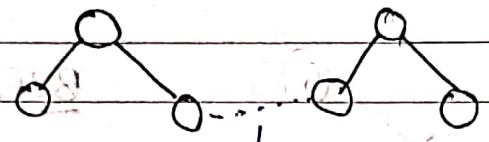


DFS Spanning tree:-



Special Points :- (About DFS & BFS) :-

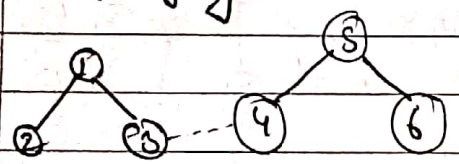
(i) DFS works on both Connected & Non-connected graph as it choose a new node every time a level is completed, but,
BFS works only on connected graph as it evaluates all its child & then do backtracking to switching to new node is not possible.



↳ Cross Edge :- Virtual Edge joining two forest tree (Non-Connected Graph).

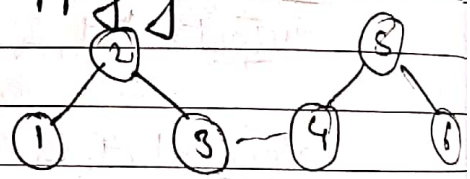
Forest :- A forest is a disjoint union of trees or equivalently an acyclic graph that is not necessarily connected.

Applying DFS



(1) → root, (2, 3) → No further root
(5) → root (4, 6) → No further root
Output :- 1, 2, 3, 5, 4, 6.

Applying BFS :



(2) → root 1, 3
1, 3 no root,
Backtrack → (2) end.
2, 1, 3 = Output.

(ii) Complexity Analysis :- The time complexity of BFS & DFS will be $O(V+E)$, where,
V → No. of vertices E → No. of edges.
This again depends on the data structure that we represent the graph.
If it will be adjacency matrix, it will be $O(V^2)$

Scanned with CamScanner

* If we use adjacency list, it will be O(V+E). Now, to explain how, let us consider the difference between sparsely connected graph & densely connected graph.

* Dense Graph :- is a graph in which the number of edges is close to maximal number of edges.

* Sparse Graph :- Sparse graph is a graph in which number of edges is close to minimal no. of edges.

→ Sparse graph can be disconnected.

* Now, what this means is that, if the graph has so many edges & is still a small graph with only few vertices, then the number of iterations & recursive call is dominated by number of edges.
This type of graph is dense graph & Time complexity will be O(E).

* Consider an example, where the number of vertices are very much larger than the number of edges. This kind of graph is sparsely connected graph.
And it may or may not be disconnected graph.
So in BFS in this case will be purely dominated by the number of vertices, rather than by number of edges. So, time complexity is O(V).

* So, what does it mean by O(V+E)? It means, whichever term is bigger will dominate the time complexity.
That is why time complexity of BFS / DFS is O(V+E).

Scanned with CamScanner

Page No.
Date

Dijakstra's Algorithm :-

Problem:- This algorithm is to find a shortest-path spanning tree

NOTE:-

(i) difference b/w Minimum Cost & shortest path tree :-

→ A minimum cost spanning tree :- It is one which has the smallest possible total weight (where weight represent cost or distance). There might be more than one such tree but Prim's & Kruskal guarantee one of them.

→ Shortest path tree :- For a specified vertex (say X), a shortest path tree is a spanning tree such that the path from X to any other vertex is as short as possible (ie has the minimum possible value).

(ii) difference b/w Kruskal's, Prim's and Dijakstra's :-

Global Approach

* Kruskal's :- At each step, choose the cheapest available edge anywhere which doesn't violate the goal of create a spanning tree.

Local Approach

* Prim's :- Choose a starting vertex. At each successive step, choose the cheapest available edge attached to any previously chosen vertex which doesn't violate the goal of creating a spanning tree.

Mix Approach

* Dijakstra's :- At each step, choose the edge attached to any previously choosen vertex (the local aspect) as small as possible (which make the total distance from the starting vertex (the global aspect)), and doesn't violate the creation of a spanning tree.

* Dijakstra is also not applicable for -ve weights.

Complexity :-

if adjacency Matrix is given, then $O(V^2)$

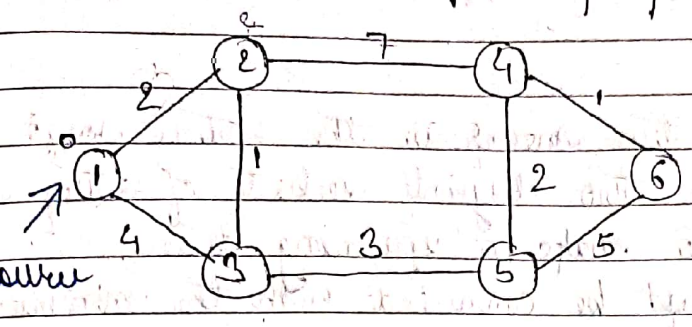
if not Adjacency Matrix ie a binary heap is given then $= O(E \log V)$.

Scanned with CamScanner

Page No
Date

Example of Dijaktra:-

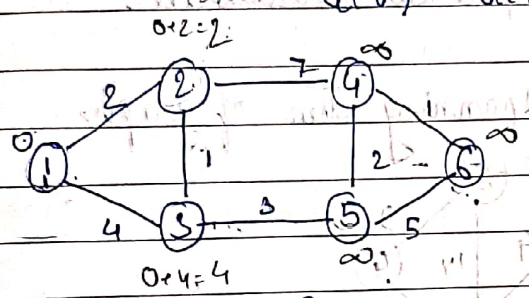
* Find the shortest path for following graph:-



Ans:- Relation:-

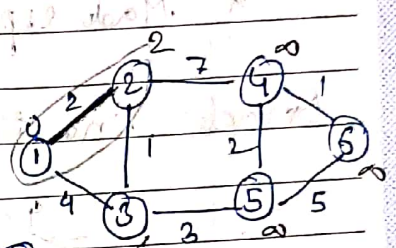
$d(u) + c(u,v) < d(v)$, then

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



Step ①

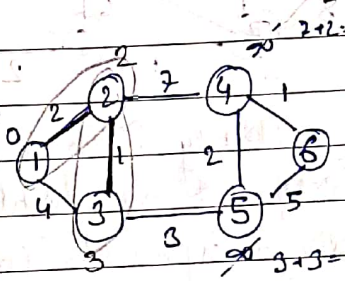
$1 \to 2 = 2$ ✓
$1 \to 3 = 4$ ✗



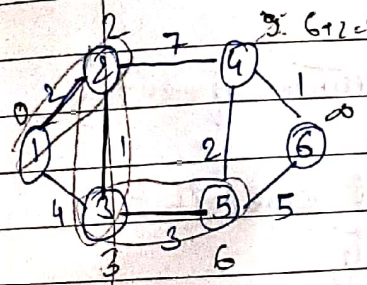
Step ②

2+1=3.

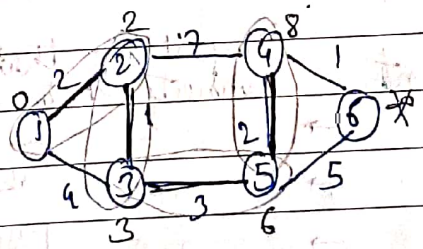
$1 \to 3 = 4$
$2 \to 3 = 2+1 = 3$.
(✓)



$2 \to 4 = 2+7 = 9$ ✗
$3 \to 5 = 3+3 = 6$ ✓
$1 \to 3$ (cycle) ✗



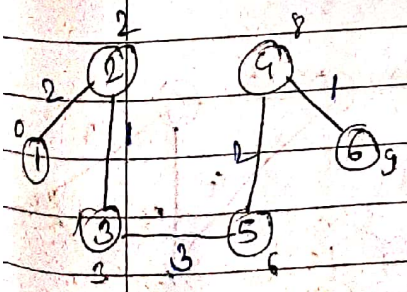
$5 \to 4 = 2+6 = 8$ ✓
$2 \to 4 = 7+2 = 9$ ✗

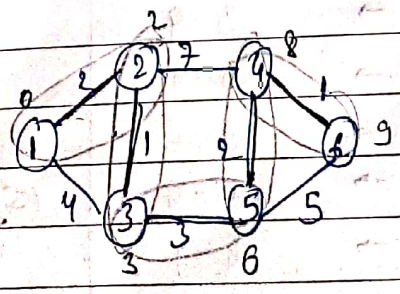


8+1=9

$5 \to 6 = 6+5 = 11$ ✗
$4 \to 6 = 8+1 = 9$ ✓

final





Total cost distance (b/w $1 \to 6$) = $2+1+3+2+1 = 9$.

Page No.
Date

Sollin's Algorithm or Borůvka's Algorithm (Greedy Approach):-

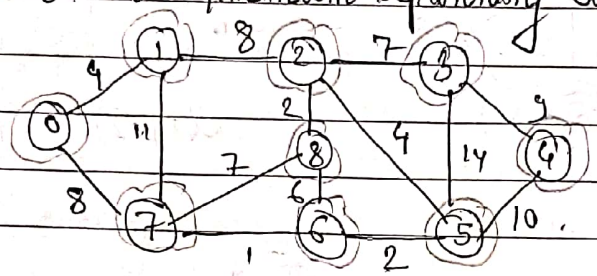
Spanning tree:-

A spanning tree means all the vertices must be connected. So, the two disjoint subsets of vertices must be connected to make a spanning tree.

And they must be connected with the minimum weight edge to make it a Minimum Spanning tree.

* Made especially to use Minimum resources.

Example:- Find the Minimum spanning Tree of graph :-

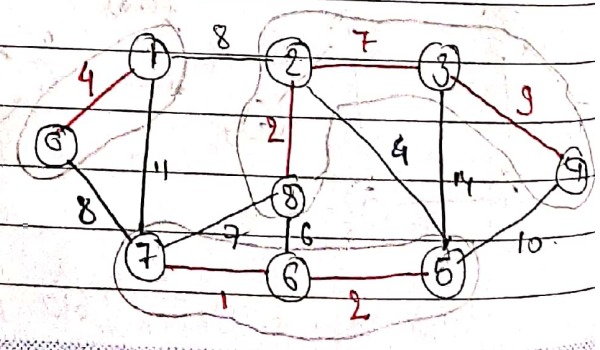


Soln:- Step ① - Initially MST is empty. Each vertex is a single component as highlighted as circle.

* For each component, find the cheapest edge that connects it to some other component.

Component		Cheapest Edge that connects it to some other component.
0	→	0-1
1	→	0-1
2	→	2-8
3	→	3-2
4	→	3-4
5	→	5-6
6	→	6-7
7	→	6-7
8	→	2-8.

Step ②:- The cheapest edges are highlighted. Now, MST become {0-1, 2-8, 2-3, 3-4, 5-6, 6-7}.

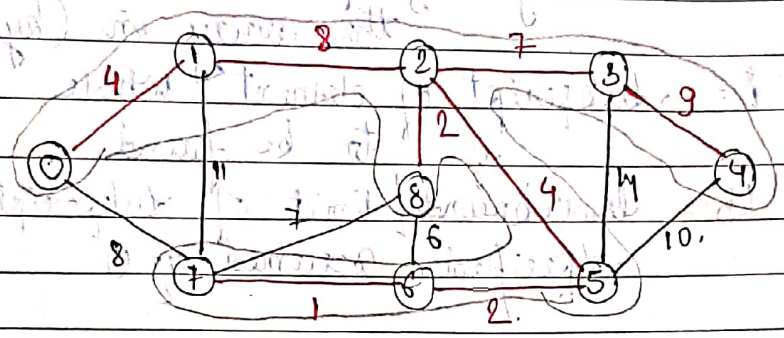


Scanned with CamScanner

Step II:- After above step, components are {{0,1}, {2,3,4,8}, {5,6,7}}.
The component are enriched.

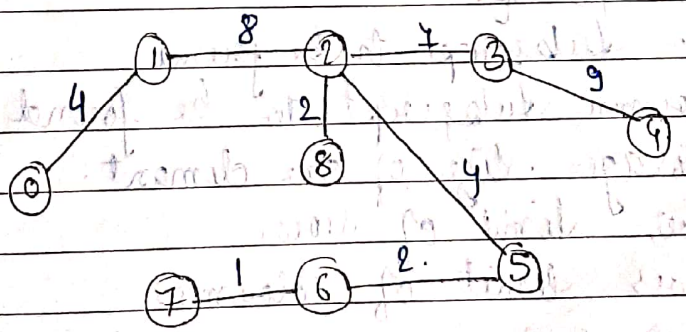
* Again find the cheapest edge for every component that connect it to some other component.

Component	Cheapest Edge
{0,1}	1-2 or 0-7
{2,3,4,8}	2-5
{5,6,7}	2-5



At this state, there is only one component {0,1,2,3,4,5,6,7,8} which has all edges. Since there is only one component left, we stop & return MST.

So, required Minimum Spanning Tree is:-



total weight of MST is
$= 4+8+7+2+4+2+1+9$
$= 37.$

Unit II nd :-

Page No.
Date

Data Structure Reviews

* Arrays :- (Already done)

Calculation of Memory Address in an Arrays :-

Single Array :-

$$A[i] = B + W * (i - lb)$$

Where, B = Base Address
W = Storage size of one element stored in the array in (bytes)
i = subscript of element whose address is to be found.
Lb = lower bound / limit of subscript, if not specified, assumed as 0.

2-D Array :-

Row Major System

$$A[i][j] = B + W * [N * (i - lr) + (j - Lc)]$$

Coloumn Major System

$$A[i][j] = B + W * [(i - lr) + m * (j - Lc)]$$

B = Base Address.
i = Row subscript to be found.
j = Coloumn subscript to be found
W = Storage size of one element.
lr = lower limit of row.
lc = lower limit of coloumn.
m = No. of rows in given matrix
N = No. of coloumn in given matrix.

if given row / coloumn as $A[Lr \ldots ur, Lc \ldots uc]$ in that case,

No. of rows calculated as $= (ur - Lr) + 1$
No. of coloumn calculated as $= (Uc - lc) + 1$.

Scanned with CamScanner

Page No. _____
Date _____

Note:- Some points about Algorithms:-

→ Characteristics of Fundamental instructions:-

(i) Definiteness:- Every fundamental instruction must be defined without ambiguity

e.g. a = a ↑ b

↳ invalid symbol → Invalid inst.

(ii) Finiteness:- Every fun^n inst. must terminate within finite amount of time.

e.g. i = 1

```
while (i)
{ i = i+1; // infinite loop.
}
```

(iii) Every fundamental instruction must accept atleast 0 input & provide atmost 1 output.

→ About analysis of Algorithm:- process of comparing two algorithms w.r.t time, space, no. of registers, network bandwidth etc is called Analysis.

→ two type of Analysis:-

Priority Analysis	Posteriori Analysis
→ Analysis done before executing	→ Analysis done after executing
e.g. x = x+1	e.g. x = x+1 → sys I. 0.3 nS, sys II. 0.3 nS, sys III. 0.6 nS
Principle:- frequent count since x+1 being carried out only 1 time, so, its complexity is O(1) [order of 1]	
→ It provides estimated value	→ It provides exact values
→ It provides uniform value	→ It provides non-uniform value
→ It is independent of CPU, O/S & system architecture	→ It is dependent

Scanned with CamScanner

Page No.
Date

Algorithm Complexity Analysis :-

Unit II :-

part (1) :- Divide And Conquer :

→ (1) Binary Search :-

general Eqn $T(n) = 2T(n/2) + 1$

Best Case = $\Omega(1)$

Worst Case = $O(\log n)$

part (2) Min-Max Problem

general Eqn $T(n) = 2T(n/2) + 2$

Compared to Naive Method, in divide and Conquer approach, the number of comparison is less, However, using Asymptotic notation, both of the approaches are represented by $O(n)$

Complexity = $O(n)$

More specified eqn :- $T(n) = \begin{cases} 2T(n/2) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$

part (3) :- 2-Way Merge Sort

general Eqn $T(n) = 2T(n/2) + n$

Worst, Best, Average Case Complexity = $O(n \log n)$

part (4) :- Selection Sort

Best Case Complexity = $O(n)$

Worst Case Complexity = $O(n^2)$

Scanned with CamScanner

Page No.
Date

part (5) :- Quick Sort

For Best case & Average case } general eqn $= T(n) = 2T(n/2) + n$

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

For Worst Case } → general eqn :- $T(n) = T(n-1) + n$.

Complexity $= O(n^2)$

part (6) :- Strassen's Matrix

general eqn $T(n) = 7T(n/2) + n^2$.

Complexity $= O(n^{2.8})$.

→ "Greedy Approach" ←

(1) Fractional knapsack :- due to sorting $O(n \log n)$
(2) Prim's Algorithm :- general $O(V \log V + E \log V)$
(3) Kruskal's Algorithm :- general $O(V \log V + E \log V)$.

→ general eqn $= O(E + \log V)$ [if use Fibonacci Heap].

(4) Huffman Coding :- $O(n \log n)$ — $O(\log n)$ ~~deterministic~~ cheapest weight
— $O(n)$ iterations.

(5) Optimal Merge pattern $= O(m+n)$
size of file m, size of file n.

Dynamic Programming :-

(1) general Method (0-1 Knapsack using dynamic Approach) $= O(n^2)$

(2) Subset sum problem } $O(n^2)$
(3) Chain Making problem }
(4) Floyd Warshall Algorithm $O(n^3)$. (All pair shortest path).

Scanned with CamScanner

(5) Longest Common Subsequence $O(m \times n)$ (string, substring length)

(6) Bellman's Ford Algorithm $O(VE)$ vertex × edge (single source shortest path)

(7) Travelling Salesman Problem (TSP) $(n^2 * 2^n)$ Evaluating all cases

(8) Optimal binary search tree $O(n^3)$ in dynamic approach

(8) * Backtracking :-

(i) N-Queen's Problem $O(n^2)$ [best] [worst] $O(n!)$

(ii) Sum of sub-sets $O(n\, 2^n)$

(iii) Hamiltonian Cycle $O(n^2 2^n)$

→ Branch and Bound

(i) 0/1 knapsack problem

(ii) Travelling Salesman problem

(1) time next term

(2) term satisfying

(3) bounding fun^n

(4) S_k satisfying B_k

Unit I :-

(i) Count Sort :- $O(n+k)$ → Range

(ii) Bucket Sort :- $O(n + (\text{no. of digits}))$

Unit II :-

(i) Topological $O(V+E)$ } → BFS & DFS
Bipartite $O(V+E)$

(ii) String (i) KMP :- $O(n)$

(ii) Rabin Karp :- $O(mn)$, $O(n-m+1)$

(iii) Boyer Moore :- $O(mn)$, $\Omega(m/n)$

(iv)

(10) ~~Kruskal~~ Dijkstra & Bellman

$O(E \log V + V \log V)$

Knapsack Eqn

$$V[i,w] = \max(V[i-1,w],\ V[i-1][w - w[i]] + P[i])$$

"Abhishek Jha"

Boyer Moore Algorithm:-

* An algorithm for pattern searching especially when we have a very large text given.
* doesn't work good for small text.

* Approach :- (i) Bad Character Rule
  (ii) Good Suffix Rule.

(i) Bad Character Rule:-

Ex:- Given String :- TRUST HARD TOOTH BRUSH
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Pattern :- TOOTH.

Step (1):- Construct "Bad Character table"
→ Using Minimum shift Algorithm

Letter	T	O	H	*
Value	~~4~~ 1 4	~~3~~ 2 2	5 5	5 5

Note :- * last character = length (if not repeated)
* Formulae Used

$$\boxed{\text{Value} = \text{length} - \text{index} - 1}$$

Here:-

	0	1	2	3	4
Pattern :	T	O	O	T	H
Index :	0	1	2	3	4

T = 5 - 0 - 1 = 4
O = 5 - 1 - 1 = 3
O = 5 - 2 - 1 = 2.
T = 5 - 3 - 1 = 1
H = (last character) = 5

* → Represent all character present in text but not in pattern & $\boxed{* = \text{length}}$

Scanned with CamScanner

Bad character table:- Long

letter	T	O	H	*
Value	1	2	5	5

Index →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
String	T	R	U	S	T	H	A	R	D	T	O	O	T	H	B	R	U	S	H
Pattern	T	O	O	T	H														
	*	T	O	O	T	H													
							T	O	O	T	H								
										T	O	O	T	H					

→ Shift by 1
→ Shift by 5
→ Shift by 2

→ All Match ✓

Algorithm:- $last(c) = \begin{cases} \text{index of the last occurrence of } c \text{ in pattern} & \text{if } c \text{ is in 'P'} \\ -1 & \text{otherwise.} \end{cases}$

$i \rightarrow m-1$ $\quad$ $n \Rightarrow$ No. of character in String

$j \rightarrow m-1$ $\quad$ $m \Rightarrow$ No. of character in pattern.

```
Repeat until (i > n-1)
    if ( P[j] = T[i]) then
        if j = 0 then
            return i        // we have a match.
        else
            i = i-1;
            j = j-1;
    else
        i = i + m - Min (j, 1+ last[T[i]])   → from value table
        j = m-1
Return "no Match".
```

Scanned with CamScanner

Complexity Analysis:-

(i) Worst Case / Average Case:-

Everytime we get a match but shift only by one character.

So, complexity = $O(m * n)$

(ii) Best Case:-

Everytime we skip the whole character of pattern (jump by)

So, complexity = $O(n/m)$.

* Problem With Bad Character:- (if no "Bad character table" is created)

→ In some cases, bad character produces some negative shifts.

Example:-

Mismatch character

	0	1	2	3	4	5	6	7
String:-	N	O	N	O	N	Y	O	O
pattern:-	Y	O	N	O				

So, we have to shift by "(-2)" towards left

* This means we need some extra information to produce a shift on encountering a bad character. The information about last position of every character in the pattern & also the set of character used in the pattern.

Scanned with CamScanner

Complexity Analysis:-

(i) Worst Case / Average Case:-

Everytime we get a match but shift only by one character.

So, complexity = $O(m * n)$

(ii) Best Case:-

Everytime we skip the whole character of pattern (jump by)

So, complexity = $O(m/m)$.

* Problem With Bad Character:- (if no "Bad character table" is created)

→ In some cases, bad character produces some negative shifts.

Example:-

Mismatch character

	0	1	2	3	4	5	6	7
String:-	N	O	N	O	N	Y	O	O
pattern:-	Y	O	N	O				

So, we have to shift by "(-2)" towards left

* this means we need some extra information to produce a shift on encountering a bad character. The information about last position of every character in the pattern & also the set of character used in the pattern.

Scanned with CamScanner

(Abhishek Kumar Jha)

(i)

(ii) Good Suffix Rule :-

* generally used when some successfull attempt of matching is done already.

* Use concept of Reusing the already matched string.

Cases Occur :- (i) → weak good suffing

Bad character

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	C	A	B	A	B	A	C	B	A
P	C	B	A	A	B						
						C	B	A	A	B	

→ search for AB in pattern, ∵ no AB move whole by length of Pattern

B character shift by 5.

(ii) suffix is also present in the other subset of the substring :-

Bad character

i	0	1	2	3	4	5	6	7	8	9	10
T	A	B	A	A	B	A	B	A	C	B	A
P	C	A	B	A	B						
			C	A	B	A	B				

→ Another "AB" in the pattern displace to overlap.

(iii) A prefix of P which matches with suffix of T

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	B	A	B	A	B	A	C	B	A
P	A	B	B	A	B						
				A	B	B	A	B			

→ Not full 'BAB', but AB (sub-part) is found, move to that much value

Strong Good Suffix:-

Bad character

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	A	A	B	A	B	A	B	A	C	B	A	C	A	B	B	C	A
P_s	A	A	C	C	A	C	C	A	C								
							A	A	C	C	A	C	C	A	C		
P_w				A	A	C	C	A	C	C	A	C					

we place "AC" that is not preceded by C

if done using weak Good suffix :-

* Combining both:- Algorithm:-

if (P[i] = T[j])

if more than one character Matches

Apply "Good Suffix Algorithm"

if only one character Matches

Apply "Bad character Algorithm"

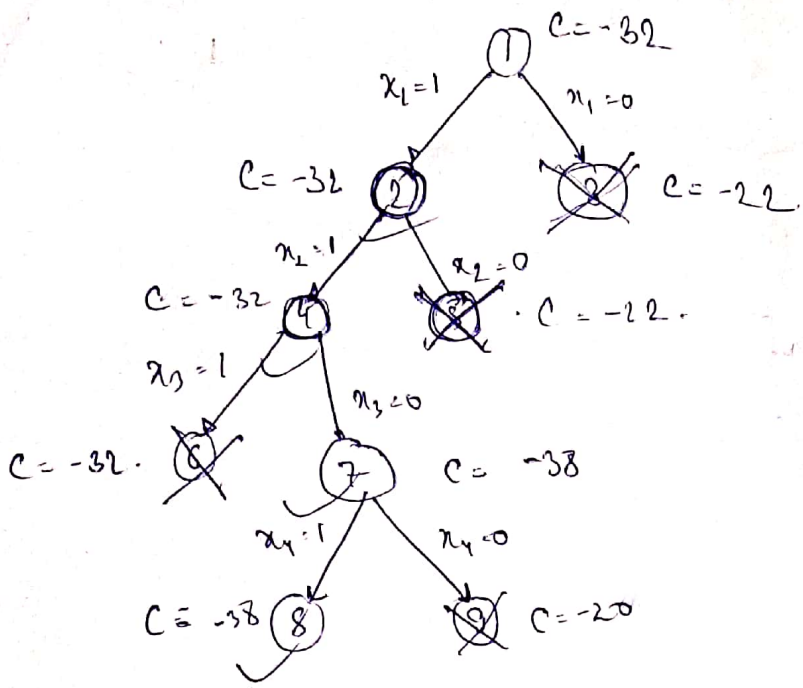
else

shift by whole length

Benifit of Combining:- Worst case complexity reduces from O(m * n) to O(m).

Abhishek (Abhishek kumar Jha)

Knapsack 0-1



done

$p = A_{11} + B$

$q =$

$r =$

$s =$

$t =$

$u =$

$v =$

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

$C_{11} = p + s - t + v$

$C_{12} = R + T.$

$C_{21} = q + s$

$C_{22} = p + r - q + u$

Scanned with CamScanner