Analysis and Design of Algorithms

module - 1

Introduction

Algorithm:

An algorithm is any "well-defined computational procedure" that takes some value or set of values "as input" and produces some value, or set of values, "as outputs".

(OR)

An algorithm is thus a sequence of "computational steps" that transform the "input into the output and its describes a specific computational procedure" for achieving that "input and output relationship"

Properties of algorithms

Input:- An algorithm must Recieve 0 or more data supplied externally.

output:- An algorithm must produce at least one output as a result.

Finiteness :-
it must contain finite numbers of steps"

Definitness :-
it must be clear, (well defined) and unambiguous

effectiveness :-
* Can be solved problem without applying Intelligence. and
* it is (dependon) Complexity of algorithms

Complexity measurements :- 1) Time
2) Space

Time :-
How much time taken to exectim of program.

Space → programs
*) How much memory required to written program in memory.

* How much memory required during program Execution.

Recurrence and NON Recurrence Algorithms

Recurrence Algorithms

⇒ when an algorithm contains a recursive call to itself, its running time can often be described by a recurrence.

→ A recurrence is an Equation or inequality that describes a function in terms of its value on smaller inputs.

~~NONRecurrence Algorithms~~

→ A Recursive algorithms is one in which objects are defined in terms of other objects of the same type

Advantages

- Simplicity of code
- Easy to understand

Finding time complexity

1.
```
A()
<
int i;
for(i=0 i<n i++) {n+1 times
Printf("Hello"); n. times
```

Soln n

2
```
AC)
{
 Out i,j
for(i=1  i<n i++) n+1 times
for(j=1  j<n j++) n times
Printf("Hello"); n*n times
}
```

Soln n^2

3.
```
A()
{
Out i,j,k,n
for(i=1; i<=n; i++)   n+1 times
{
for(j=1; j<=i; j++)  n times
 {
printf("Hello");   n^2 times
}}
```

D D M M Y Y Y Y

Asymptotic Notations

→ Asymptotic notations are "mathematical tools to represent time complexity" of algorithms.

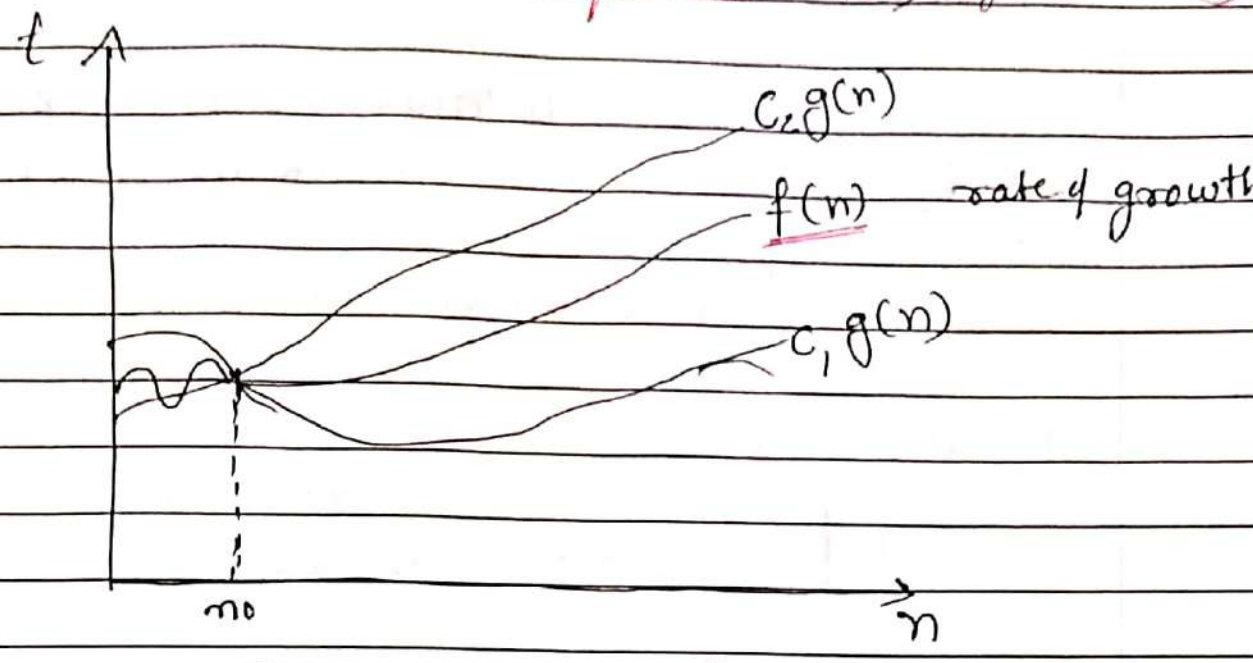
→ Asymptotic analysis is to have a "measure of efficiency" of algorithms.
→ it can be used to analyze the performance of an algorithm for some large data set.
Three asymptotic notations are mostly used to represent time complexity of algorithms

① Θ Notation (Theta) (Tight bound)
② O Notation (Big O) (upper bound)
③ Ω Notation (Omega) (lower bound)

Big Theta (Θ) Notation.

→ The Theta notation bounds a functions from above and below.
→ Θ notation gives bound for a function to within a constant factor we write $f(n) = \Theta(g(n))$ if there are positive constant n_0, c_1 and c_2 such that to the right of n_0 the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive

$\Omega(g(n)) = \{$ f(n): There exist positive constant c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$ $\}$



Proove :- $3n^2 + 3n + 1 = \Omega(n^2)$

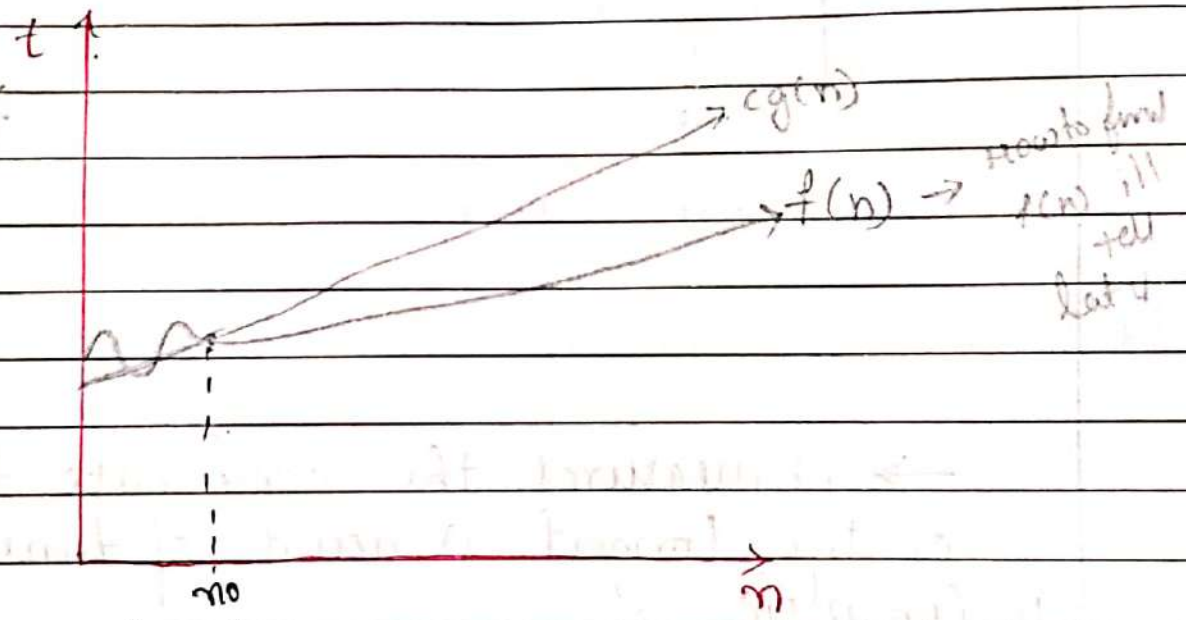
② Big O Notations.

→ it measures the "worst case time complexity" or the "longest amount" of time an algorithm to complete.

→ O Notation gives an upper bound for a function to within a constant factor we write $f(n) = O(g(n))$

if there are positive constants c and ~~n~~, n_0 such that to the right of n_0 the value of f(n) always lies on or below $cg(n)$

$O(g(n)) = \{ f(n)$: There exist positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$



$f(n)$ → function of n
n → inputs

What is upperbound or worst case for given function
i.e $g(n)$
n_0 → limit inputs

Condition : $C > 0$ $n_0 \geq 1$
$n \geq n_0$
$f(n) = O(g(n))$

D D M M Y Y Y Y

Example : $f(n) = 3n+2$ $g(n) = n$

$f(n) = O(g(n))$

① $0 \le f(n) \le cg(n)$

$3n+2 \le cn$ $c > 0$

$n \ge n_0$ $n_0 \ge 1$

$3n+2 \le 4n$

$3n+2 \le 4n$ ✓ $c \ge 4$

$n = 1$ ✗

$n \ge 2$. $n = 2$ ✓

$f(n) = O(g(n))$ ✓ $g(n) : n^2$
$n3$
⋮
n^n
Θn

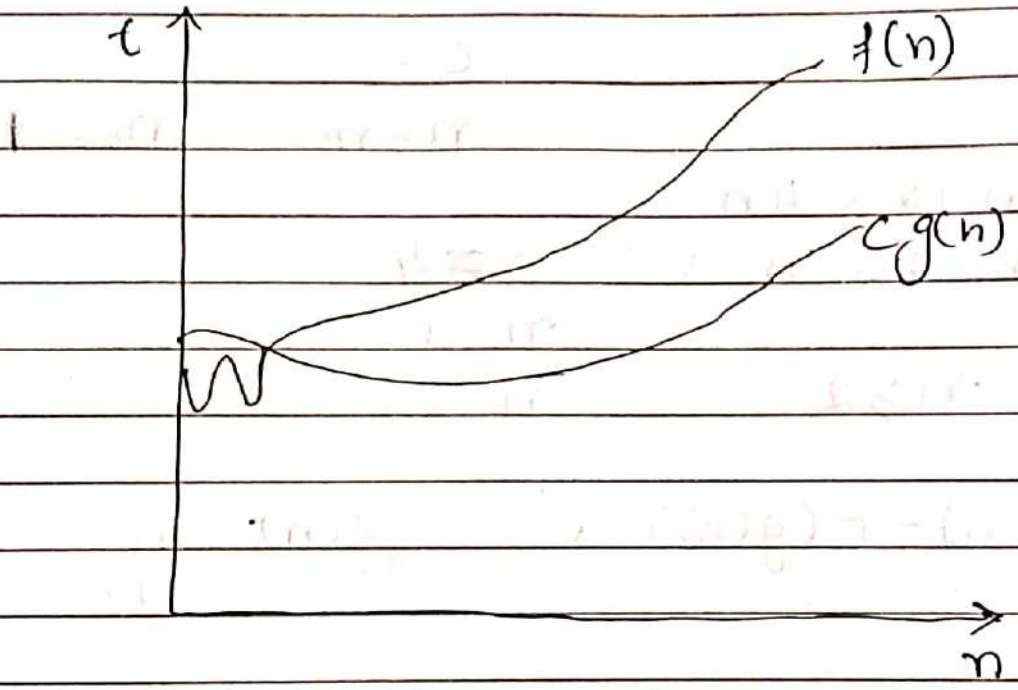
③ Big Omega (Ω) :-

→ Big Omega notation is Used to define the lower bound of any algorithm or we can say the best case of any algorithm.

→ Ω - notation gives an lower bound for a function to within a constant factor we write $f(n) = \Omega(g(n))$. if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$

D D M M

$\Omega(g(n)) =$ { $f(n)$: There exist positive constants c & n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$ }



O → worst case

Ω → best

Θ → average

D D M M Y Y Y Y

Recurrence Relation

→ A Recurrence relation is an equation or inequality that describes a function in terms of its values on smaller inputs.

→ To Solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence

There are methods for Solving Recurrence

1. Substitution Method
2. Recursion Tree method
3. master method.

Substitution Method

→ The substitution method consists of two main steps

1. Guess the Solution
2. Use the mathematical induction to find the boundary condition and shows that the guess is correct.

Example

① Solve the Equation by Substitution Method

$T(n) = T(n/2) + n$ we have to show that it is asymptotically bound by $O(\log n)$

Solution :- $T(n) = O(\log n)$

$T(n) \leq c \log n$

$\leq c \log(n/2) + n$

$\leq c \log n - c \log 2 + 1$

$\leq c \log n$ for $c \geq 1$

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

② Consider the Recurrence

$T(n) = 2T(n/2) + n \quad n > 1$ Find an asymptotic bound on T.

(i) $T(n) = 2t(n/2) + n \rightarrow (1)$

Sub $n/2$ in Eqn (1)

$T(n/2) = 2t(n/2^2) + n/2 \rightarrow (2)$

$T(n/4) = 2t(n/2^3) + n/4 \rightarrow (3)$

Sub 2 in (1)

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

$4t(n/2^2) + 2n \rightarrow (4)$

$4[2t(n/2^3) + n/2^2] + 2n$

$8t\left(n/2^3 + 2^2 \frac{n}{2^2}\right] + 2n$

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

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

$n/2^k = 1$

$k \log 2 = \log n$

$k = \log_2 n$

$\because a^{\log_a b} = b^{\log_a a}$

$\log_a^a = 1$

$2^k + 2^{n/k} + k(n)$

$2^{\log n_2} + T(1) + n \log_2 n$

$n \log_2 n + T(1) + n \log_2 n$

$n \log_2 2 \times 1 + n \log_2 n \rightarrow \log_a^a = 1$

$n + n \log_2 n = O(n \log 2^n)$

Recursion Tree Method

① Recurion Tree method is a pictorial representation

② Consider the second term in recurrence as root

* it is useful when the divide & conquer algorithm is used.

* it is sometimes difficult to come up with a good guess.

* Each root and child ~~representation~~ represents the cost of a single subproblem.

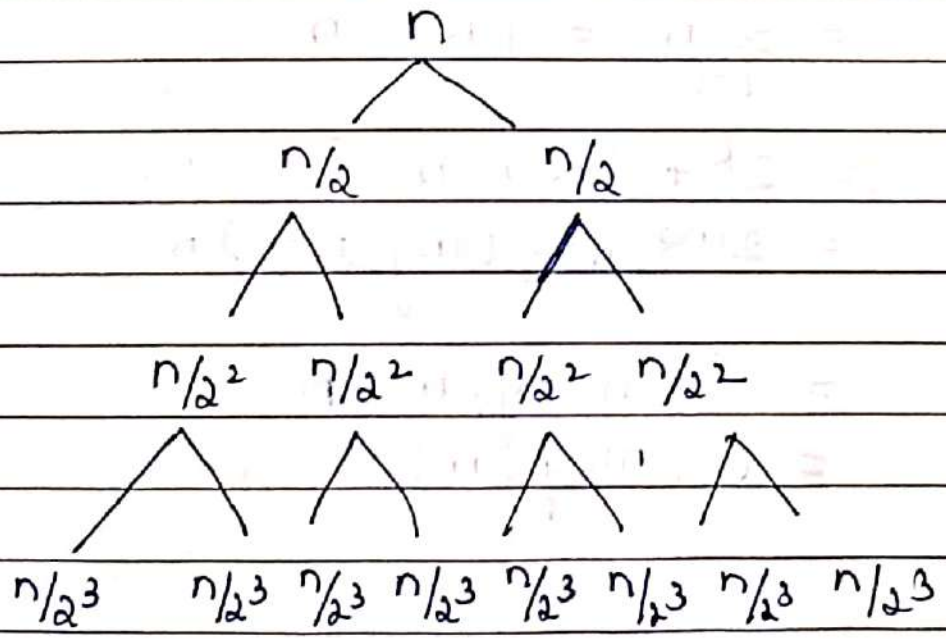
① $T(n) = 2T(n/2) + n^2$ we have to obtain the asymptotic bound using recursion tree method.

② $T(n) = 4T(n/2) + n$

③ $T(n) = T(n/3) + T(2n/3) + n$

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

level	node	Size of sub prob	Cost of each level
0	1	n	$1 \times n = n$
1	2	$n/2$	$2 \times n/2 = n$
2	2^2	$n/2^2$	$2^2 \times n/2^2 = n$
3	2^3	$n/2^3$	$2^3 \times n/2^3 = n$
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
$T(n/2^k)$	2^k	$n/2^k$	$2^k \times T(1) = 2^k$
$= T(1)$			



$n/2^k = 1 \ (n/2^k)$

$n = 2^k$

Apply log on b.s

$$k = \log_2 n$$

Cost of Each level = Total no of node in each level x size of node

Total cost of = cost of leaf level * cost of remaining level

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

Cost of leaf level $= 2^k + T(1)$

$= 2^k \cdot (1) \Rightarrow 2^k$

Cost of remaining level $= \underbrace{n+n+n+\cdots n}_{(k-1)}$

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

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

$= 2\log_2 n + (\log_2 n - 1)n$

$= n + n\log_2 n - n$

$= O(n\log_2 n)$

D D M M Y Y Y Y

Master Method

The master method is used for Solving the following types of recurrence.

$T(n) = aT(n/b) + f(n)$ with $a \geq 1$ $b \geq 1$ be constant & $f(n)$ be a function and n/b can be interpreted as

$T(n)$ is defined on non-negative integers

→ n is the Size of the problem.
→ a is the number of Subproblems in the recursion
→ n/b is the Size of each Subproblem.
→ $f(n)$ is the Sum of the work done outside the recursive calls, which includes the Sum of dividing the problem and the Sum of combining the Solutions to the Subproblems.

Three Case :-

$\log_b^a$

D D M M Y Y Y Y

$T(n) =$

$\Theta(n^{\log_b^a})$ $\quad f(n) = O(n^{\log_b^{a-\varepsilon}})$

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

$\Theta(n^{\log_b^a} \log n)$ $\quad f(n) = \Theta(n^{\log_b^a})$

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

$\Theta(f(n))$ $\quad f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$ AND

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

$af(n/b) \leq cf(n)$ for large

$\varepsilon > 0$

$c < 1$

case 1: $f(n) = O(n^{\log_b^{a-\varepsilon}})$ for some constant $\varepsilon > 0$

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

Example:-

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

compare $T(n) = 8T(n/2) + 1000n^2$

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

$a = 8 \quad b = 2 \quad f(n) = 1000n^2 \quad \log_b a = \log_2 8$

$= \log_2 2^3 = 3$

put all the values in

$$f(n) = O(n^{\log_b a - \varepsilon})$$

$$1000n^2 = O(n^{3-\varepsilon})$$

if we choose $\varepsilon = 1$ we get $1000n^2 = O(n^2)$

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

$$T(n) = \Theta(n^3)$$

Case 2:- if it is true, for some constant $k \geq 0$

$F(n) = \Theta(n^{\log_b^a} \log^k n)$ then it follows that:

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

Example

$T(n) = 2T\left(\frac{n}{2}\right) + 10n$ with $a \geq b$ & $b > 1$

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

$a = 2 \quad b = 2 \quad f(n) = 10n \quad \log_b^a = 1$

putall the values in

$f(n) = \Theta(n^{\log_b^a} \log^k n)$,

$10n = \Theta(n^1) = \Theta(n)$

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

$= \Theta(n \log n)$

Case 3:- if it is true $f(n) = \Omega(n^{\log_b a+c})$ for some constant $\varepsilon > 0$ & it also true that $af(n/b) \leq cf(n)$ for some constant $c < 1$ for large value of n.

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

Example:-

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

compare the given problem with

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \text{ with } a \geq 1 \text{ & } b > 1$$

$a = 2$, $b = 2$, $f(n) = n^2$, $\log_b^a = \log_2^2 = 1$

put all the values in

$$f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$$

$\varepsilon = 1$

$$n^2 = \Omega(n^{1+\varepsilon})$$

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

check 2nd Condition

$$af(n/b) < cf(n)$$

$$2\left(\frac{n}{2}\right)^2 \leq cn^2$$

$$\frac{1}{2}n^2 \leq cn^2$$

$$c \geq \frac{1}{2} \qquad n \geq 1$$

$$\frac{1}{2}n^2 \leq \frac{c}{2}n^2$$

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

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

Linear Search

→ Linear search is a process that checks Every Element in the list Sequentially until the desired Element is found.

→ Time Complexity for linear search is $O(n)$

→ It is also known as Sequential Searching.

Algorithm

Step 1 : Set i to 1

Step 2 : if $i > n$ then go to Step 7.

Step 3 : if $A[i] = x$, then go to step 6

Step 4 : Set i to $i+1$

Step 5 : Go to Step 2.

Step 6 : print Element x found at index i and go to Step 8

Step 7 : print Element

Step 8 : Stop.

pseudocode

```
procedure linear_search (list, value)
    for each time in the list
        if match item == value
    return the item's location
    End if
    End for
End procedure.
```

Analysis

1. Input : n elements are given, x Element new
2. Output : Elements found
3. Logic : If you n Elements

Then n elements in that we need to search

x

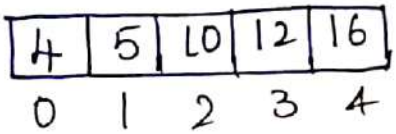
→ Element is found at 1st position

O(1) best case

O(n) worst case

O(n/2) average case.

Example



i = 0 , x = 12

A[i] = A[0] = 4

A[i] == x

4 == 12 → fails

i = 0+1 = 1

A[1] = 5

A[i] == x

5 == 12 → fails

i = 1+1 = 2

A[i] == x

10 == 12 → fails

i = 2+1 = 3

A[i] == x

3 == 12 ✓ True

Element found at index 3.

Binary Search

→ Binary Search also known as a half Interval Search, is a searching Technique used to locate a specified value (key) within the given array.

→ It follows Divide and Conquer approach.

Algorithm

Binary_Search (A, lower_bound, upper_bound, VAL)

Step 1: [INITIALIZE] Set Beg = lower_bound
End = upper_bound, pos = -1.

Step 2: Repeat Step 3 and Step 4 while BEG <= END

Step 3: SET MID = (BEG + END)/2

Step 4: If A[MID] = VAL
SET POS = MID

PRINT POS

Goto Step 6

Else if A[MID] > VAL
SET END = MID - 1 return Binary Search (a, i,

Else
SET BEG = MID + 1

[END OF IF]

[END OF LOOP]

Step 5: If POS = -1
print "value is not present in the Array")
[END OF IF]

Step 6: Stop.

pseudocode

procedure binary-Search

A → Array

n → Size of Array

x → Value to be Searched

Set lowerbound = 1

Set upperbound = n

while x not found

if upper bound < lower bound

Exit : x does not exists

Set mid point = lowerbound + (upper bound - lower bound)/2

if A [mid point] < x

Set lower bound = mid + 1

if A [mid point] > x

Set upper bound = mid - 1

if A [mid point] = x

Exit : x found at mid point

end while

End procedure.

Analysis

A[5] = [10, 20, 30, 40, 50] (0, 1, 2, 3, 4)

n = 5

low = 0

x = 40

high = 4

→ Case 1 : mid = (low + high)/2

= (0+4)/2

= 4/2 = 2

a[mid] == x

30 == 40

a[mid] > x

30 > x

30 > 40

a[mid] < x

30 < 40

→ x = 40 low = mid+1, high = 4

mid = (3+4)/2 = 3

low <= high

3 <= 4

Time Complexity

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

$T(n) = 2(n/2) + 10 : 2T(n/2) + 10$

$T(n) = O(n/2)\ 2T(n/2) + C$ → (1) Solve using Recursive tree method

Example

	0	1	2	3	4	5	6	7	8
list :	10	12	20	32	50	55	65	80	99

Search Element = 12

Step 1 : mid = 50

x == 50

x < 50 } x is smaller than 50 So take the left sub array

	0	1	2	3
list	0	12	20	32

Search Element 12 compared with mid = 12

A[i] == x

12 == 12

Element found at index 1.

Factorial of number.

Algorithm

Step 1. Start

Step 2. Read n, call fact(n)

Step 3. initialize i=1, fact=1

Step 4. Repeat Step 4 through 6 until i=n, if n==1 return 1

Step 5. ~~fact=fact*~~ f = n* fact(n-1)

Step 6. Return f

pseudocode

```
fact(n)
Begin
if n==0 or 1 then
 Return 1
else
return n* call Fact(n-1);
 end if
End.
```

Time Complexity Analysis

When we analyse time complexity of a program we make an assumption that each simple operation costs us one unit

```
Fact(n)
{
 if (n==0) — 1
    return1 — 1
 else
  return fact *n (n-1)
                 |   |
              (n-1)  1
               unit
```

Now lets say time taken to calculate factorial of n is Equal to T(n) when we try to analyse

$$T(n) = T(n-1) + 3 \quad \text{if } n>0$$

$$T(0) = 1$$

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

$$= T(n-2) + 6$$

$$= T(n-3) + 9$$

$$= T(n-4) + 12$$

$$\vdots \qquad \vdots$$

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

$$n - k = 0 \Rightarrow k = n$$

$$T(n) = T(0) + 3n$$

$$= 3n + 1$$

$$T(n) < n$$

$$O(n) //$$

Insertion Sort

Defination: Insertion Sort is a Sorting algorithm in which it sorts Elements one by one and places it to the right position to get the Sorted list of Elements

~~Algorithm~~ pseudocode

1. For $k \leftarrow 2$ to length [A]
2. Do key $\leftarrow$ A[k]
3. $i = k-1$
4. while $i > 0$ and A[i] > key
5. do A[i+1] $\leftarrow$ A[i]
6. $i = i - 1$;
7. A[i+1] $\leftarrow$ key;

~~pseudocode~~ Algorithm

Insertion_Sort

Step 1: If it is the first Element it is already Sorted return 1;

Step 2: pick next Element

Step 3: Compare with all Elements in sorted sub list

Step 4: Shift all the Elements in the sorted sublist that is greater than the value to be sorted

Step 5: Insert the value

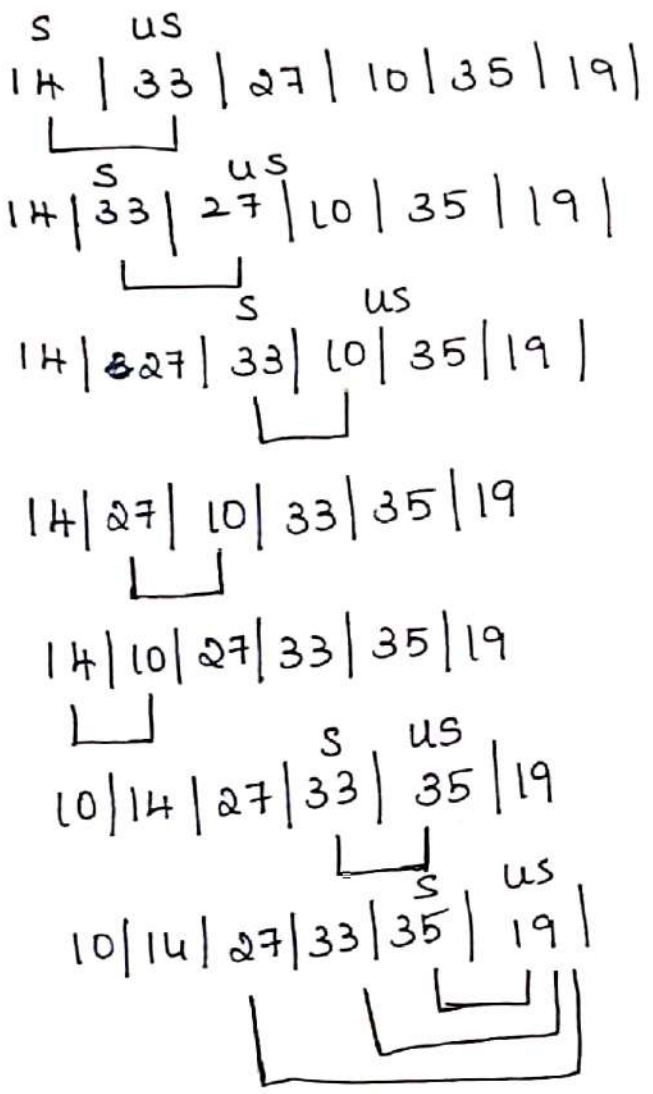
Step 6: Repeat until list is sorted.

Analysis

14 33 27 10 35 19

j = 2

n = 6



Sorted Elements

10 | 14 | 19 | 27 | 33 | 35.

① $j = 2 \rightarrow 33$

$j = 2$ to A.length

2 to 6

Key = A[j]

$33 = 6[2]$

$i = j - 1$

$= 2 - 1 = 1$

$i = 1$

$i > 0$ & $A[i] > key$

$i > 1$ & $14 > 33$

② $j = 3 \rightarrow 6$

Key = A[i] = 27

$i = j - 1 = 3 - 1 = 2$

while $(i > 0)$

$i > 0$ & $A[i] > key$

$14 > 27$

$A[i+1] = A[i]$

$A[3] = A[2]$

$i = i - 1 = 2 - 1 = 1$

$A[2] = 27$

③ $j = 4$ to 6

Key = A[j]

Key = 10

$i = j - 1 = 4 - 1 = 3$

while $(3 > 0$ & $a[i] > key$

$33 > 10$

$A[i+1] = A[i]$

$A[4] = A[3]$

$i = 2.$

Time Complexity

	compare	+	move	
$j=2$	1	+	1	= 2
$j=3$	2	+	2	= 4
$j=4$	3	+	3	= 6
⋮				
$j=n$	$(n-1)$	+	$(n-1)$	$= 2(n-1)$

$= 2+4+6+8+\ldots 2(n-1)$

$= 2(1+2+3+4+\ldots(n-1)$

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

Radix Sort

Definition: Radix Sort is a distribution sort where it distributes the given array Element in respective place value packets. If there are n digit number to be sorted then radix sort require n passes to sort the given array

Example

Input a [312, 411, 623, 517, 422, 716]

411, 312, 422, 623, 716, 517

411, 312, 716, 517, 422, 623

312, 411, 422, 517, 623, 716

Radix Sort pseudocode

```
algo radixsort (a[], n)
{
max data = get max (a[], n)
for (Exp = 1; max data/ Exp > 0, Exp = Exp * 10)
{
 countsort (a[], n, Exp)
}
}
algo countsort (a[], n, Exp)
{
 initialize count [0....9] with 0's
 for (i=0; i<n; i++)
  count [(a[i]/Exp) % 10] ++ ;
 for (i = 1; i<10 ; i++)
  {
 count [i] = count [i] + count [i-1]
  }
 // building output array
  for (i = n-1; i>=0 ; i--)
{
output [count [(a[i]/Exp) % 10]-1] = a[i]
count [(a[i]/Exp) % 10]--;
  for (i=0; i<n ; i++)
   {
    a[i] = output [i];
   }
```

Ex:

arr[] =

247	321	515	227	642	413	109	248	758	930

count[10] =

0	1	2	3	4	5	6	7	8	9
1	1	1	1	0	1	0	1+1=2	1+1=2	1

Output =

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

Step 1: Consider the last digits

Exp = 1 maxdata = 930/1 > 0 Exp = Exp * 10

pass 1

→ i = 0 ; i<n ; i++

0<10 ; i++ true

Count [a[i]/Exp] % 10]++ a[0] = 247

Count [247/1] % 10]++

Count [7]++ { increase 1 value in count[10] at index 7

→ i = 1

1<10 ; 1++ true

Count [321/1] % 10]++ a[1] = 321

Count [321 % 10] ++

Count [1]++ { increase 1 value in count[10] at index 1

→ i = 2

2<10 ; 2++

Count [515/1] % 10]++ a[2] = 517.

Count [515 % 10]++

Count [5]++ { increase 1 value count[10] at index 5

i = 3

3 < 10 ; 3++ // Condition true

Count [a[i]/1] % 10] ++ Your a[i] value is 227
∴ a[3] = 227.

Count [227/1] % 10] ++

Count [227 % 10] ++

Count [7]++ // increase value of 1 in count[10] at index 7.

→ i = 4

4 < 10 ; 4++ // condition is true

Count [642/1] % 10]++ ∴ Your a[i] = 642

Count [642 % 10]++

Count [2]++ // increase value of 1 in count [10] at index 2

→ i = 5

5 < 10; 5++ // condition True

Count [413/1] % 10]++ a[5] = 413

Count [3]++

→ i = 6

6 < 10 ; 6++ // condition true

Count [109/1] % 10]++ a[6] = 109

Count [109 % 10] ++

Count [9] ++

→ i = 7

7 < 10; 7++ // condition true

Count [248 | 1] % 10] ++

Count [248 % 10] ++

Count [8] ++

→ i = 8

8 < 10 ; i++ // Condition True

Count [758 | 1] % 10] ++

Count [758 % 10] ++

Count [8] ++

→ i = 9

9 < 10 ; 9++ // condition true

Count [930 % 10] ++

Count [0] ++.

You will perform for loop statement with condition below

ie for (i = 1; i < 10 ; i++)

1 < 10 // condition true

perform count [i] = Count [i] + Count [i − 1]

= 1 + 0

= 1

→ now i = 2

2 < 10 // True

= 2 + [2 − 1]

= 2 + 1 = 3

Count[8] = 3<10 // True

3+2 = 5

Count[4] = 4<10 // True

4+3 = 7

Count[5] = 5<10 // True

5+4 = 9

Count[6] = 6<10 // True

6+5 = 11

Now perform output conditions

→ i = n-1

= i = 10-1

i = 9

→ out counta[i] / exp % 10] - 1] = a[i]

[930 / 1 % 10 - 1]

10 - 1 = 9

output [9] = 930

Count (930/1) % 10] - -

Count [9] - -

a [0] = 930.

→ i = 9-1 = 8

[758 / 1 % 9-1]

9-1 = 8

Output [8] = 758

Count (758/1) % 10] - -

Count [8] - -

a[8] = 758

} perform for all upto condition fails.

Final sorted array after pass 1

930	321	642	413	515	247	227	248	758	109

∴ perform pass 2 the same way.

Time Complexity = $O(10 \times n)$

→ problem

Algorithm

- polynomial Time
- Non-polynomial time

→ P-class problem

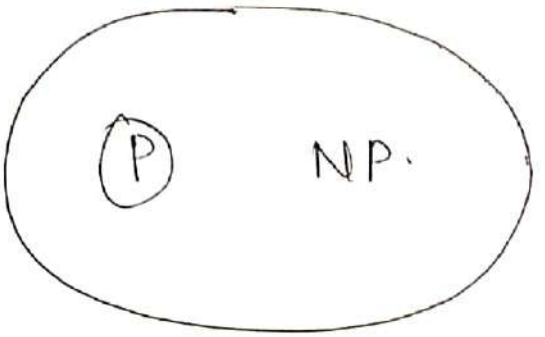
A problem which can be solved on polynomial time is known as P-class problem.

Ex: All sorting & searching Algorithms.

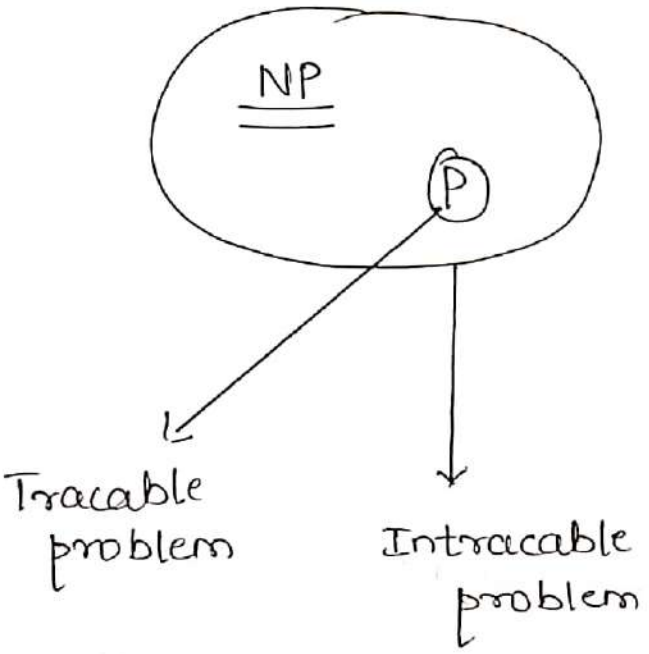
→ NP class problem

A problem which cannot be solved on polynomial time but is verified in polynomial time is known as Non Deterministic polynomial time or NP-class problem.

Ex: prime factor, Travelling, Su-do-ku.



NP class problem it is verified in polynomial time



NP class

Hard to solve & Easy to verify

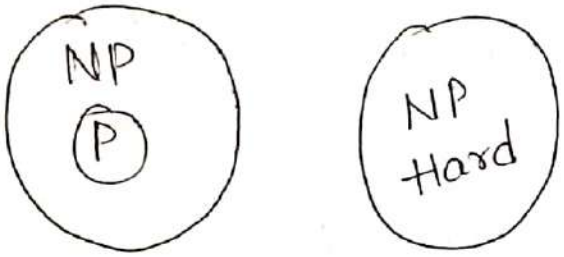
P-class

Easy to solve & Easy to verify polynomial time

$P \subseteq NP$

NP Hard problem

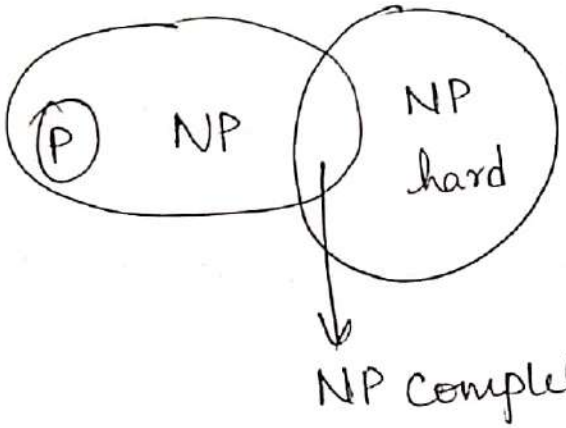
A problem is NP-hard if Every problem in NP can be polynomial reduced to it.

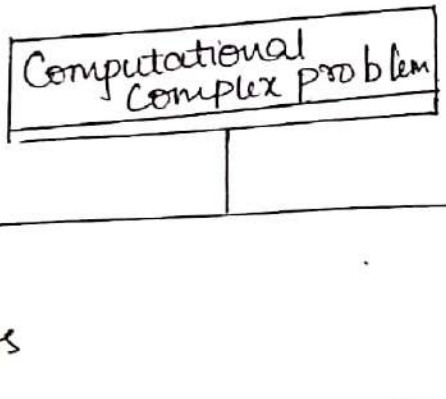


Ex: optimization problem

NP complete problem

→ A problem is a part of NP and NP-hard problem can be called as Np complete problem





Ex: Decision tree

Algorithm Design Techniques

① Divide & Conquer Approach

—> It is in top down Approach

3 steps

* divide the original problem into a set of sub problems
* Solve Every Subproblem individually
* Combine the Solution of the Subproblems into a solution of the whole Original problem

② Greedy Technique

* Greedy method is to solve the optimization problem
* An optimization problem is one in which we are given a set of input values
* which are required Either to be maximized Or minimized

(3) Dynamic programming

* Dynamic programming is a bottom-up approach

* we can solve small problems

(4) Branch and Bound

* In Branch and Bound algorithm a given sub problem which cannot be bounded has to be divided into at least two new restricted sub problems

* Its very slow algorithm

(5) Randomized Algorithm

A randomized Algorithm is defined as an Algorithm that is allowed to access a source

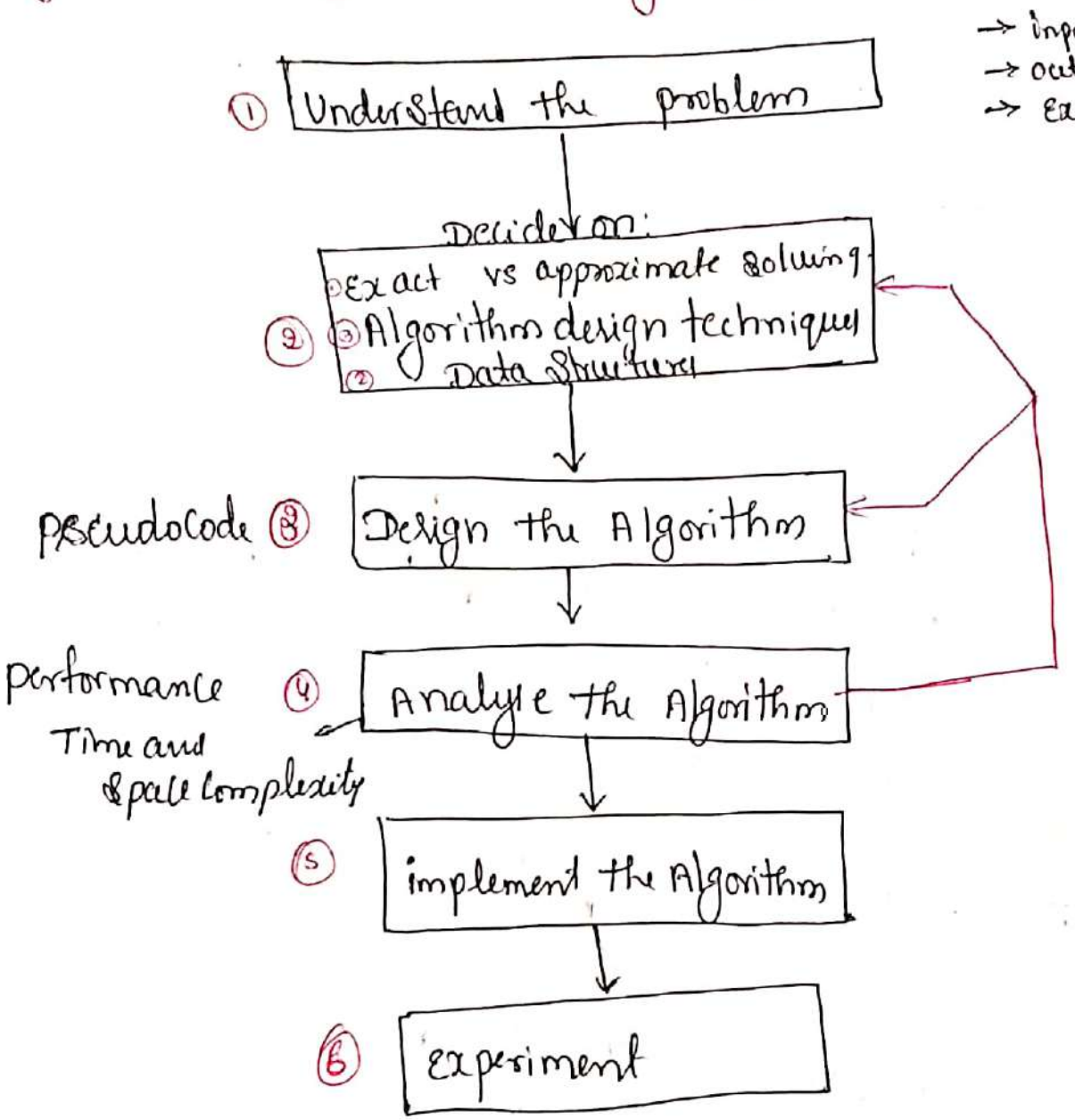
(6) Back tracking Algorithm

* Back tracking algorithm tries each possibility until they find the right one

* It is a depth-first Searched of set of possible solution.

— X —

Algorithm Design and Analysis process



Pseudocode Algorithm:- (fractional knapsack)

step1:- calculate the ratio (profit/weight) for each object.

step2:- Arrange the decreasing order of (profit / weight) ratio of all items.

step3:- Take the item with highest ratio & add them to the knapsack.

step4:- Repeat the step3 until we cannot add next item as a whole

step5:- At the end add next time as much as we can. (∵ fraction)

step6:- find the profit.

Pseudocode:-

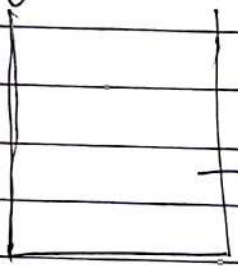
```
Knapsack( n, weight, profit, capacity)
{
    .
    .
    .
    U = capacity;
```

Sort the ratio and swap the profit and also weight

Apply any sorting technique and solve this!

Swaping down

ratio =	6	5	4.5	3	3	1.3	1
profit =	6	10	18	15	3	5	7
weight =	1	2	4	5	1	3	7



→ knapsack or bag or Container capacity is 15

→ 1st object ratio is (6) ✓

(1st) Step m = 15 i = 1

w = 1 } of 1st object
p = 6 }

m = m - w[i]

= 15 - w[i]

15 - 1 = 14

Total profit = Total profit + P[i]

0 + 6 = 6

(2) m = 14 ratio = (5) p = 10 w = 2

$\frac{p[i]}{w[i]}$ *

Refer class Notes

TW = w[i] - a;

= TW + (w[i] - a)

= 13 + (3 - 2) = (15)

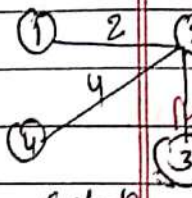
Bafna Gold

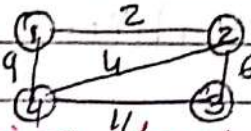
~~Program~~ :- ~~Refer~~ ~~class~~ Notes

Minimum Spanning Trees

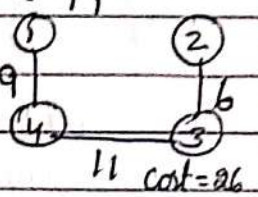
Spanning Trees (undirected graph)

it is a Trees which connects all the vertices of graph with out any cycles or loops





Minimum Cost Spanning Tree:-



A Spanning Tree weight is minimum of all possiblites is called minimum cost Spanning Tree.

Graph:- A graph data structure is a collection of nodes that have data and are connected to other nodes.

minimum cost Spanning Tree is implemented by 2 greedy algorithms

1. prim's algorithms
2. Kruskal's algorithms

```
{ totalprofit = 0, Totalweight = 0;
for(i=1; i<=n; i++)
{
    while( u > w[i] )
    {
        u = u - w[i]; // subtracting weight from capacity
        totalprofit = Totalprofit + p[i];
        totalweight = Totalweight + w[i];
    }
    if(i<n)
        totalprofit = totalprofit + u/w[i] * p[i];
        totalweight = totalweight + ( u);
    }
    break;
}
printf(" Total profit %f", totalprofit);
printf(" total weight %d", totalweight);
}
```

solve:- time complexity O(n)

Object(O) :-	1	2	3	4	5	
profit(P) :-	15	10	6	9	18	n=5
weight(W) :-	3	4	8	1	4	m=15

Time complexity $O(V^2) / O(n^2)$ // adjacent matrix

$O(E \log V)$ → binary heap

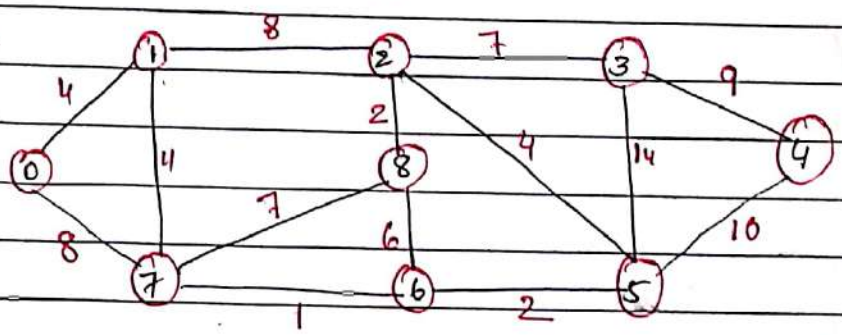
Bafna Gold

To update the key values, iterate through all adjacent vertices, For every adjacent vertex V.

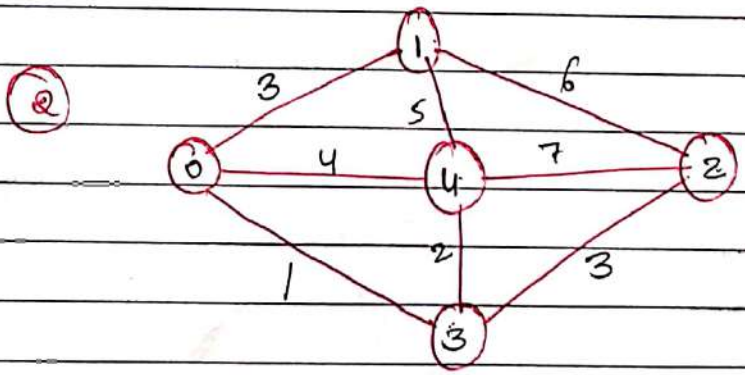
if weight of edge u-v is less than the prilious key value of v, update the key value as weight of u-v.

problem:-

(1)



V



Solution refer class notes

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

∞ = max value = 999

prim's Algorithms

→ It is an algorithm used to find minimum cost spanning tree for connected, weighted & directed graph

Steps

① Create a Set mstSet that keeps track of vertices already included in MST.

② Assign a key value to all vertices in the input graph.

Initialize all key values as INFINITE.

Assign key value as 0 for the first vertex so that it is picked first.

③ while mstSet doesn't include all vertices

ⓐ pick a vertex u which is not there in mstSet and has minimum key value.

ⓑ Include u to mstSet.

ⓒ update key value of all adjacent vertices of u.

```
int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v];
            min_index = v;
    return min_index;
}
```

```
int printMST(int parent[], int V, int graph[V
{
    printf("Edge  Weight");
    for(int i = 1; i < V; i++)
        printf("%d - %d  %d \n", parent[i], i,
        graph[i][parent[i]]);
}
```

Program

```
void primMST( graph)
{
    int parent[V];
    int key[V];
    bool mstSet[V];

    for( int i=0; i<V; i++)
    {
        key[i] = INT_MAX,
        mstSet[i] = false;
    }

    key[0] = 0;
    parent[0] = -1;

    for(int count =0; count <V-1; count++)
    {
        int u = minkey( key, mstSet);
        mstSet[u] = true;

        for( int v=0; v< V; v++)
        {
            if( graph[u][v] && mstSet[v] == false &&
                graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];
        }
    }
    printMST( parent, V, graph);
}
```

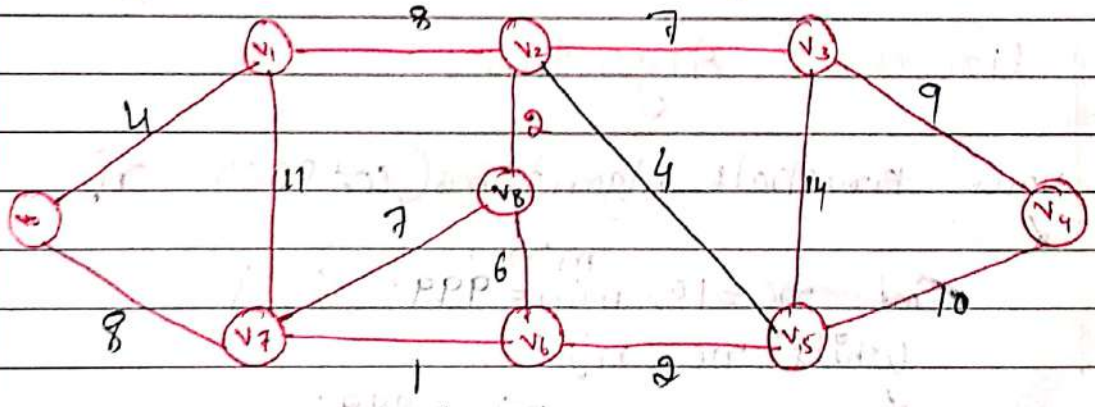
Bafna Gold
Date: Page:

Kruskal's Algorithm

Step 1:- Sort all the edges in increasing order of their weight.

Step 2:- pick the smallest edge, check if it forms a cycle with the spanning tree formed so far. if cycle is not formed, included this edge. Else, discard it.

Step 3:- Repeat Step 2 until there are (V-1) edges in the spanning tree.

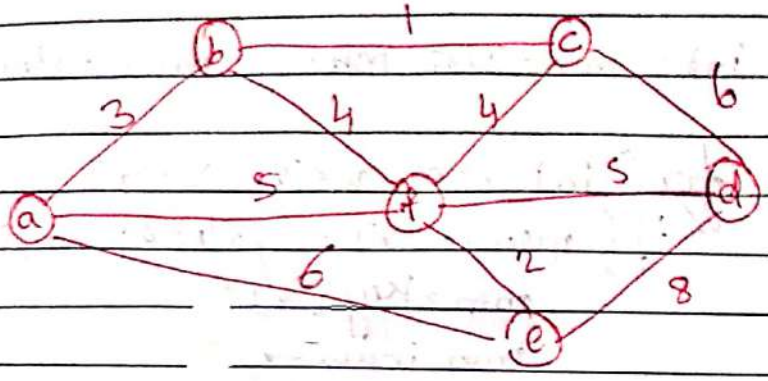


Edge	v_3v_6	v_6v_5	v_8v_2	v_2v_5	v_0v_1	v_6v_8	v_8v_7	v_2v_8	v_1v_2	v_3v_4	v_4v_5	v_1v_7	v_8
wt	1	2	2	4	4	6	7	7	8	9	10	11	14

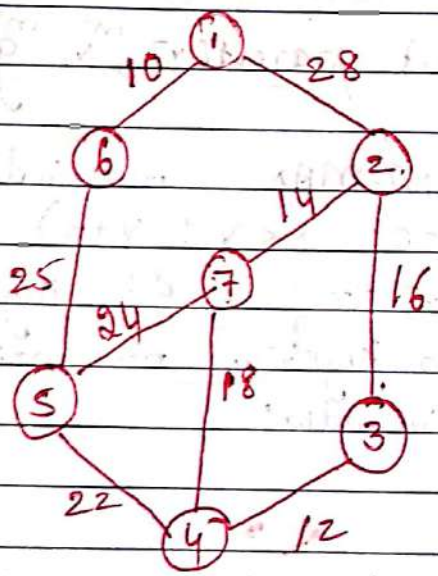
MST - KRUSKAL (G, w)

1. A = 0 → O(1)
2. for each vertex v ∈ G.V
3. MAKE-SET (v)
4. Sort the edges of G.E into increasing o
   by weight w → O(E log E)

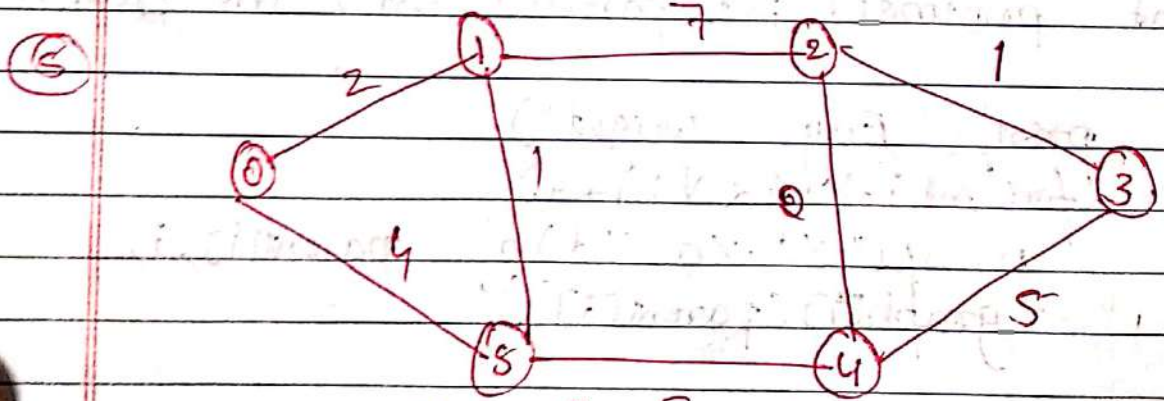
③ problem :-



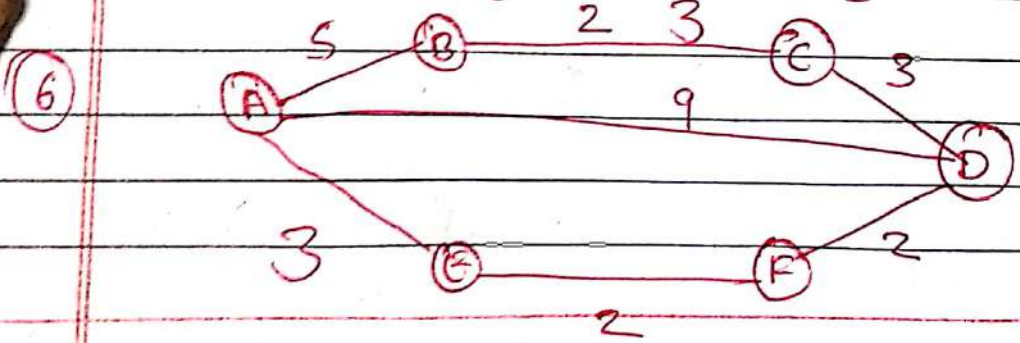
④



⑤



⑥



```
u = find(u);
v = find(v);
```

O(ElogV)

```
if( uni(u, v)
{
    printf("%d edge (%d, %d) = %d\n"
            ne, a, b, min);
    mincost += min;
}
cost[a][b] = cost[b][a] = 999;

printf("\n\t minimum cost = %d\n",
            mincost);
}

int find(int i)
{
    while (parent[i])
        i = parent[i];
    return i;
}

int uni(int i, int j)
{
    if( i != j)
    {
        parent[j] = i;
        return 1;
    }
    return 0;
}
```

5. for each edge (u, v) ∈ G.E taken in increasing order by weight

6. if FIND-SET(u) ≠ FIND-SET(v)
7. A = A ∪ {(u, v)};
8. UNION(u, v)
9. return A

→ overall Complexity = O(E logE + E log V)

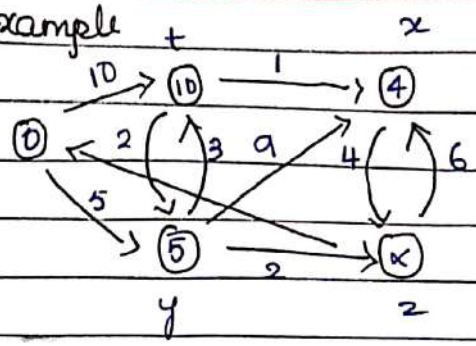
→ But value of E can be atmost $O(V^2)$, so O(LogV) and O(LogE) are same.

∴ overall time complexity is O(ElogE) or O(ElogV).

Kruskal's Algorithms

```
void kruskals_Algorithms(cost[][], n)
{
    int ne = 1, ..... , i, j;
    while(ne < n)
    {
        min = 999;
        for(i=0; i<n; i++)
        {
            for(j=0; j<n; j++)
            {
                if(cost[i][j] < min)
                {
                    min = cost[i][j];
                    a = u = i;
                    b = v = j;
                }
            }
        }
    }
```

Example



iteration 1

s.d = ∞ t.d = ∞ y.d = ∞ x.d = ∞

z.d = ∞

S {}

Q = {s, t, y, x, z}

u = s min value

S = {s}

Adj [s] = {t, y}

relax (u, v, w)

(s, t, 10) = t.d = 10 ∞ > 10

(s, y, 5) = y.d = 5 ∞ > 5

Q = {t, y, x, z}

10 5 ∞ ∞

iteration 2

u = y → min vertex

S = S ∪ {u}

S = {s, y}

Adj [y] = {t, x, z}

10 ∞ ∞

relax (y, t, w) = t.d = y.d + w (y, t) = 5+3 = 8

x.d = y.d + w(y, x) = 5+9 = 14 < ∞ . x.d = 14

z.d = y.d + w(y, z) = 5+2 = 7 .

Shortest path Algorithm

→ classic problem in graph to find shortest path in weighted graph

Types of path problems

* Single source shortest path
  → Cost of shortest path from a source vertex u to a destination vertex 'v' < Dijktra's Algorithm>

* All pair Shortest path
  → Cost of shortest path from each vertex to every other vertex < Floyd warshall's > Algorithm

DIJKstra's Algorithm

→ It is used to find shortest path between nodes in a graph

Pseudocode

Steps: 1. INITIALIZE_SINGLE_SOURCE (G,S)
2. S=0
3. Q = G.V
4. while Q ≠ 0
5. u = Extract_min(Q)
6. S = S U {u}
7. Foreach vertex (veg. Adj[u])
8. Relax (u, v, w)

if $v.d > u.d + w(u,v)$

then $d[v] = d[u] + w[u,v]$.

$S = \{s, t\}$

$Adj[t] = \{x, y\}$
$\quad\quad\quad\quad 13, 5$

relax $\{t, x, w\}$

$x.d = t.d + w(t.x)$

$= 8 + 1 = 9$

$y.d = t.d + w(t.y)$

$= 8 + 2$

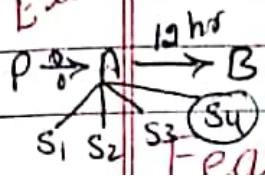
$= 10.$

→ It is used to solve an optimization problems

Optimization problem

* It is the problem which results minimum results or maximum results

Exa

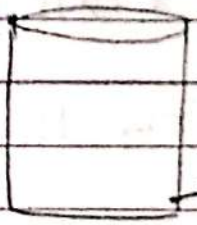


Feasible Solution :-

Knapsack problem

fractional knapsack problem

objects (o)	1	2	3	4	5	6	7
profits (p)	10	5	15	7	6	18	3
weights (w)	2	3	5	7	1	4	1



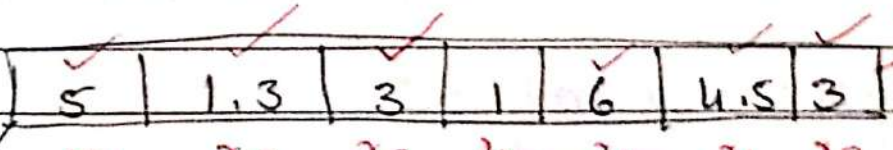
bag capacity is 15 kg (M = 15)

n = 7 → object called knapsack is container of bag

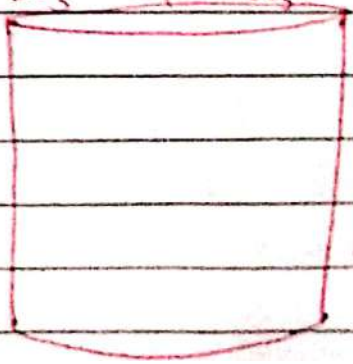
$0 \le x \le 1$

my idea take large profit object

Some will think weight ratio



2/3



15 - 1 = 14

14 - 2 = 12

12 - 4 = 8

8 - 5 = 3

Algorithm Solved by greedy technique

→ knapsack problem
→ minimum cost spanning Tree
  (a) prim's Algorithms
  (b) Kruskals Algorithms

→ Single source shortest path
  ↳ Dijkstra algorithms

General algorithm for greedy approach

```
Algorithm greedy(a, n)
{
    a[1...n] Contains n inputs

    Solution = 0 // Initialize Solution
    for( i=1 to n) do
    {
        x = Select(a);
        if feasible(Solution,x)
        then
            Solution = Union(Solutin, x);
    }

    return Solutin;
}
```

Divide and Conquer Approach

A problem is divided into smaller problems. then the smaller problems are solved independently. and finally the solution of smaller problems are combined into a solution for the large problem.

⇒ The most of the algorithms are designed using Recursion. hence memory management is very high.

Divide-Conquer algorithms have 3 parts

1. Divide the problem.
2. Conquer the sub problems.
3. Combine the solution.

* Divide the problem into a number of sub-problems that are smaller instance of the same problem.

* Conquer the sub-problems by solving them Recursively.

* Combine the solution to the sub-problems into the solution for the original problem.

→ Divide and Conquer approach support parallelism as sub-problems are independent

Application of Divide and Conquer Approach

Following are some problems which are solved using divide and Conquer Approach

① Finding the maximum and minimum of a sequence of numbers

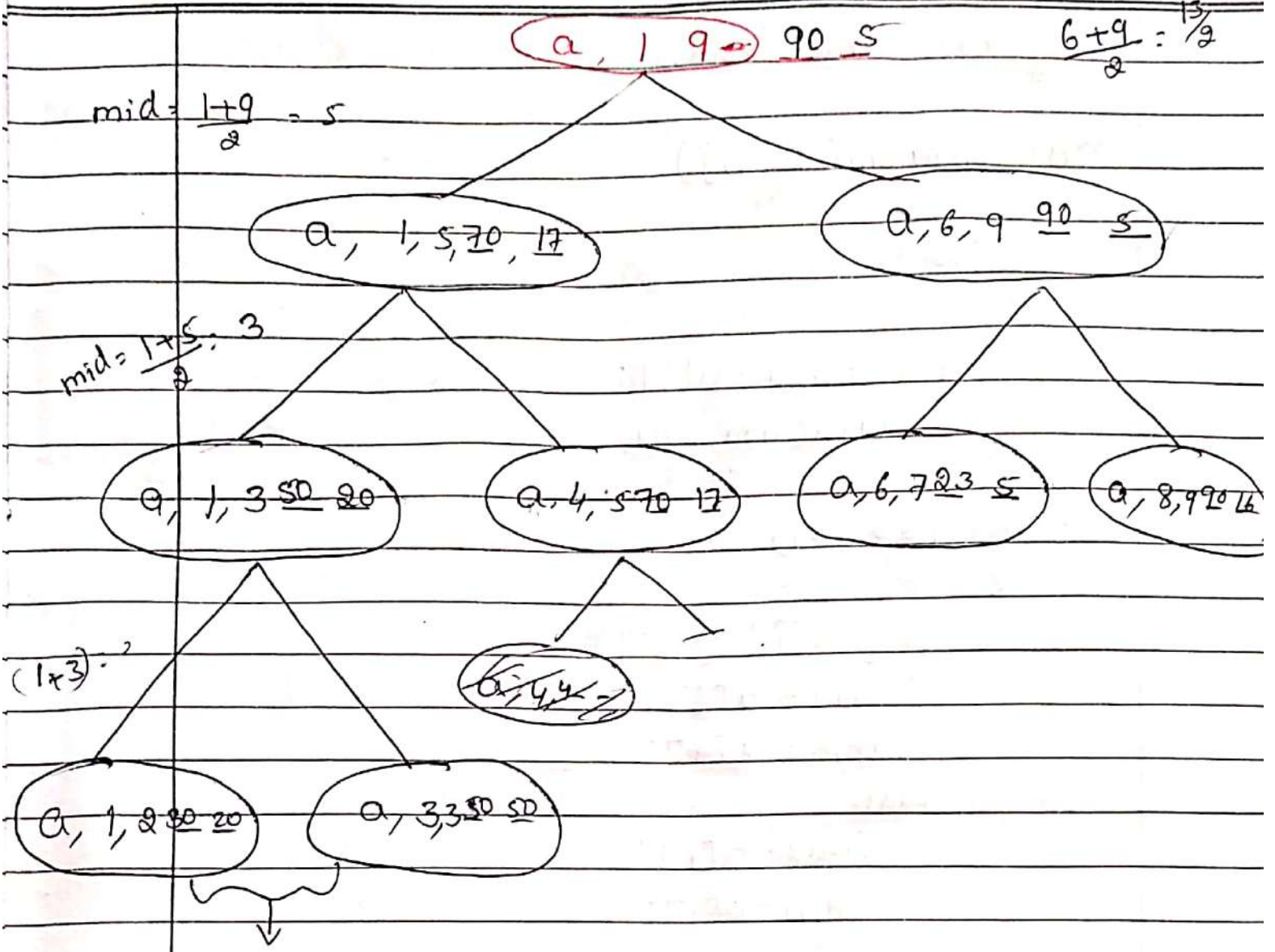
② Merge Sort
③ Quick Sort
④ Binary Search
⑤ Strassen's Matrix multiplication.

problem Statement

The max-min problem in algorithm Analysis is finding the maximum and minimum value in an array.

Solution Input : An Array of n elements
output : max & min.

Ex:- A[25 30 50 70 17 23 5 90 16]
0 1 2 3 4 5 6 7 8 9



Time Complexity

Let $T(n)$ be the no of comparisim.

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

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

Solve by using Substitutional method

D D M M Y Y Y Y

Program Pseudocode :-

```
DAC_maxmin( a, i, j)
{
    if( i == j )
    {
        max = min = a[i];
        return( min, max)
    }
else 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 {
        mid = i+j/2
T(n/2)  max1, min2 = DCA_maxmin( a, i, mid)
T(n/2)  max2, min2 = DCA_maxmin( a, mid+1, j )
        if ( max1 < max2)  → (1)
        max = max2
        else
        max = max1
```

D D M M Y Y Y Y

```
if (min1 < min2)  -> (1)
    min = min1;
else
    min = min2;
return (max, min);
}
}
```

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

$= 2[2T(n/2^2) + 2] + 2$

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

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

$= 2^k T(n/2^k) + 2^1 + 2^2 + 2^3 + \cdots 2^k$

$T(n/2^k) = T(1)$

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

$n = 2^k$

$\boxed{k = \log_2 n}$

$= 2^{\log_2 n} +$

$= \log n * 2$

$= n^{\log_2 2} +$

$= n$

$3 \; \frac{c(1 - r^{k-1})}{1 -}$

$\because a^{\log_b c}$

$c^{\log_b a} \Rightarrow \log_a a = 1$

$= n + 2\log_2 n - 2$

$= 3n - 2$

UB $= O(n)$ Ans.

$$3 = a\left(\frac{r^k - 1}{r - 1}\right)$$

$$= 2\left(\frac{2^{\log_2 n} - 1}{2 - 1}\right)$$

$a = 2$

$r = 2$

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

$= 2n - 2$

⇒ Quick Sort using Divide and Conquer algorithms.

→ It picks an element as pivot and partitions the given array around the picked pivot.

different ~~versions~~ of way of picking pivot element in Array

① Always picks first element as pivot.
② Always pick last element as pivot.
③ pick a random element as pivot.
④ pick median as pivot.

D D M M Y Y Y Y

Solution Input: An Array of n elements
Output: Sorted elements of Array in Ascending order

→ * The key process in quick Sort is partition().

* Target of partitions is given an array and an element x of array as pivot, put x at its correct position in Sorted array and put all "Smaller elements before x" and put all greater elements after x.

Pseudo Code

D D M M Y Y Y Y

```
quickSort( arr[], low, high)
{
    if( low < high)
    {
n       pi = partition( arr, low, high); →n
T(n/2)  quickSort( arr, low, pi-1); // Before pi;
T(n/2)  quickSort( arr, pi+1, high);
    }
}

partition( arr[], low, high)
{
    pivot = arr[high];

    i = low ;
for( & j = low; j<=high-1; j++)
    {
    if (arr[j] < pivot)
    {
        i++;
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
    swap( a[i+1], pivot);
    return i ;
}
```

D D M M Y Y Y Y

0	1	2	3	4	5	6	7
11	16	2	8	1	9	4	7

i j

pass 1: partition

pivot = arr[7] = 7

i = 0

① j = 0, j <= 6 ; j++

if(arr[0] < pivot)

11 < 7 ✗

② j = 1 1 <= 6

arr[1] < pivot

16 < 7 ✗

③ j = 2 2 <= 6

arr[2] < pivot

2 < 7 ✓

Swap (a[i], a[j])

(11, 2)

0	1	2	3	4	5	6	7
2	16	11	8	1	9	4	7

j = 3

④ j = 3 arr[3] < pivot

8 < 7 ✗ increment j value

⑤ j = 4 arr[4] < pivot

1 < 7 ✓

i++; i = 1

Swap (a[i], a[j])

(16, 1)

DSU

0	1	2	3	4	5	6	7
2	1	11	8	16	9	4	7
	↑ i=	↑			↑ j		

⑥ j = 5 arr[j] < pivot

9 < 7 ✗

⑦ j = 6 arr[6] < pivot

4 < 7 ✓ true

i++ i = 2

swap(a[i], a[j]):

2	1	4	8	16	9	11	7
		↑ i				↑ j=	

⑧ j = 7 ✗ j <= high-1

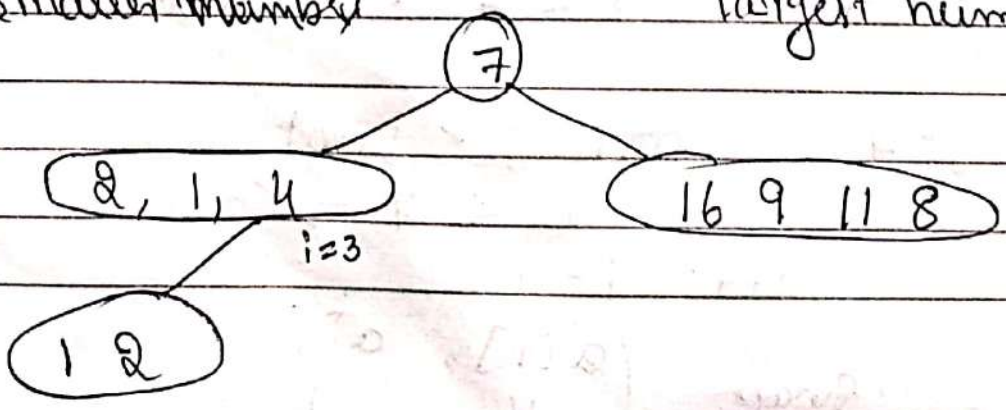
7 <= 7-1

7 <= 6 ✗ coming out of for loop

swap(a[i+1], pivot)

2	1	4	7	16	9	11	8
			↑ pivot				

Smaller numbers — largest numbers



D D M M Y Y Y Y

Time complexity

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

Merge Sort Algorithm

→ Merge Sort is divide and conquer technique of sorting elements.

→ Merge Sort is one of the most efficient sorting algorithms.

→ Merge Sort works as below.
* Divide the list of numbers into sublists.
* merge the sublists to obtain the elements in sorted order.

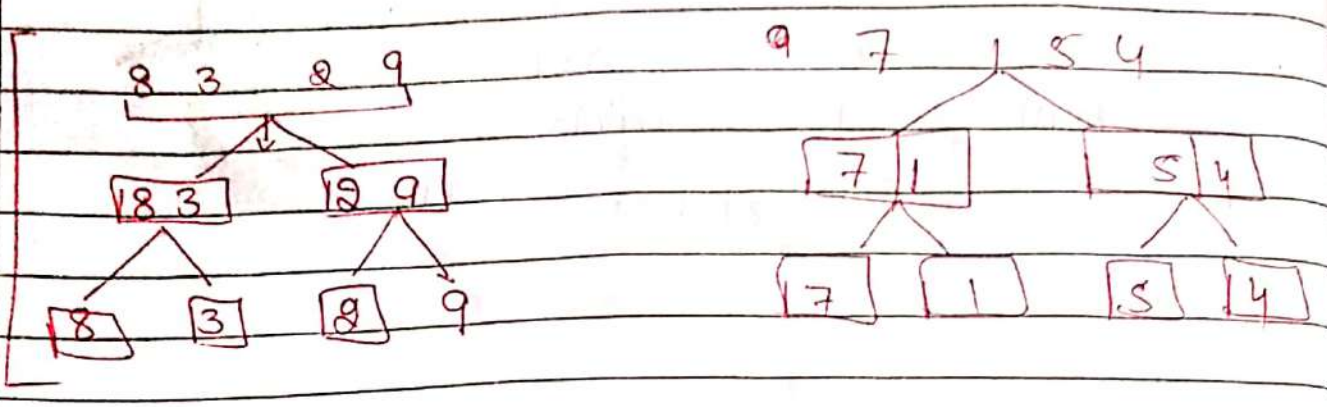
→ Merge Sort is an efficient sorting algorithm with time complexity $\Theta(n \log n)$

Solution:- Input: An array of n elements
Output: Sorted elements of Array in Ascending order.

D D M M Y Y Y Y

8 3 2 9 7 1 5 4

Divide & Conquer

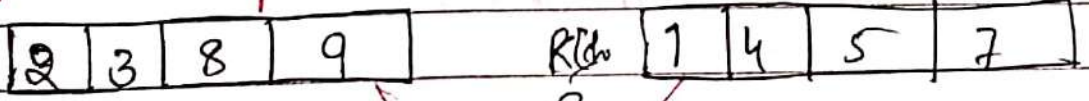


Apply Sort Sorting Techq

Sort the elements of left sublist & Sort into new L array or Right sublist into new R array.

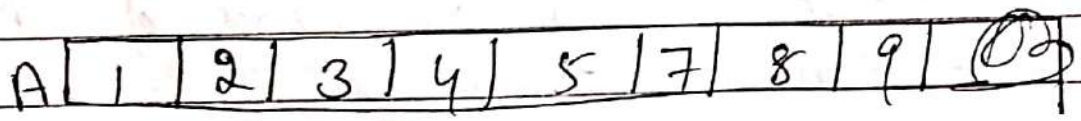
→ Then call the merge function to combine two array.

After Sorting



R

merge & ~~function~~ Sorting funct



DSU

D D M M Y Y Y Y

```
mergeSort(A, l, h)
{
    if(l < h)
    {
        mid = (l+h)/2
        mergeSort(A, l, mid);      → n/2
        mergeSort(A, mid+1, h);    → n/2

Copying element to L & R Array Refer
        Class Notes

        Merge(L, R, A);            → n
    }
}
Merge(L, R, A)
{
    nL = length(L)
    nR = length(R)
    i=0, j=0, k=0;

while(i < nL && j < nR)
{
    if(L[i] <= R[i])
    {
        A[K] = L[i])
        k++;
        i++;
    }
}
else
{
```

```
        A[K] = R[J];
            K++;
            j++;
        }
    }
while(i<NL)
    {
        A[K] = L[i];
        i++;
    }

while (j<NR)
    {
        A[K] = R[J];
        j++;
    }
}
```

OR

Another way of merging.

```
mergeSort(l, h)
{
    if (l<h)
    {
        mid = (l+h)/2;
        mergeSort(l, mid);      → n/2
        mergeSort(mid+1, h);    → n/2
        merge(a, l, mid, h);    → n
    }
}
```

D D M M Y Y Y Y

```
Merge( int a[], int start, int mid, int end)
{
    int p = start, i, q = mid+1, k = 0,
        Arr[end];

for( i = start; i <= end; i++)
{
    if (p > mid)
    {
        Arr[k++] = a[q++];
    }
    else if ( q > end)
    {
        Arr[k++] = a[p++];
    }
    else if ( A[p] < a[q])
    {
        Arr[k++] = a[p++];
    }
    else
    {
        Arr[ k++] = A[q++];
    }
}
```

Time complexity

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

DSU

D D M M Y Y Y Y

Matrix multiplication using divid & conquer multiplication

① There are two method for matrix multiplication

1. Naive method
2. Strassen's

Naive method

→ Simple multiplication method.

a Simple way to multiply two matrices.

```
void multiply( int A[][N], int B[][N],
               int C[][N]);
{
  int i, j, k;
  for( i=0; i<N; i++)
  {
    for(j=0; j<N; j++)
    {
      C[i][j]=0;
      for( k=0; k<N; k++)
      {
        C[i][j] = C[i][j] + A[i][k]*B[k][j];
      }
    }
  }
```

Time complexity of above method is $O(N^3)$.

D D M M Y Y Y Y

Divide and Conquer

→ following is Simple Divide and Conquer method to multiple two square matrices,

① Divide matrices A and B in 4 sub matrices of size $N/2 * N/2$ as shown in the below diagram.

② Calculate following values recursively. $ae+bg$, $af+bh$, $ce+dg$ and $cf+dh$.

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A}_{2\times 2/2} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{1*1} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

→ A, B and C are Square metrices of Size $N*N$

→ a, b, c and d are Submatrices of A, of Size $N/2 * N/2$

→ e, f, g and h are Submatrices of B, of Size $N/2 * N/2$

* The above method, we do 8 multiplication for matrices of Size $N/2 * N/2$ & 4 addition.

* Addition of two matrices takes $O(N^2)$ time

The time complexity can be written as

$$T(N) = 8T(N/2) + O(N^2)$$

→ if your sole DSU R9/

$O(N^3)$

D D M M Y Y Y Y

⇒ Simple Divide and Conquer also leads to $O(N^3)$. Can there be a better way?

→ The above divide & Conquer method the main Component for high time Complexity is 8 recursive calls.

⇒ The idea of Strassen's method is to reduce the number of recursive call to 7.

→ Strassen's method is similar to above simple divide & Conquer method, but in Strassen's method, the four sub-matrices of result are using following formulae

$p_1 = a(f-b)$
$p_2 = (a+b)h$
$p_3 = (c+d)e$
$p_4 = d(g-e)$
$p_5 = (a+d)(e+h)$
$p_6 = (b-d)(g+h)$
$p_7 = (a-c)(e+f)$

OR

$P = (A_{11}+A_{22})(B_{11}+B_{22})$
$Q = B_{11}(A_{21}+A_{22})$
$R = A_{11}(B_{12}-B_{22})$
$S = A_{22}(B_{21}-B_{11})$
$T = B_{22}(A_{11}+A_{12})$
$U = (A_{21}-A_{11})(B_{11}+B_{12})$
$V = (A_{12}-A_{22})(B_{21}+B_{22})$

$C_{11} = P+S-T+V$
$C_{12} = R+T$
$C_{21} = Q+S$
$C_{22} = P+R-Q+U$

D D M M Y Y Y Y

→ The AXB can be calculated using above seven multiplications

Following are values of four sub-matrices of result C.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_A \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}_B = \begin{bmatrix} P5+P4+P2+P6 & P1+P2 \\ P3+P4 & P1+P5-P3-P7 \end{bmatrix}_C$$

Time Complexity of Strassen's method

$$T(N) = 7T(N/2) + O(N^2)$$

Time Complexity of above method is $O(N^{\log 7})$ which is approximately $O(N^{2.8074})$

Generally Strassen's method is not preferred for practical Applications for following reasons

① The constants used in Strassen's method are high & for a typical application Naive method works better.

② The Submatrices in recursion take extra space.

D D M M Y Y Y Y

Divide and Conquer Strategy

```
mm(A, B, N)
{
    if (n <= 2)
    {
        C = 4 formula
    }
    else
    {
        mm(A11, B11, n/2) + mm(A12, B21, n/2);
        mm(A11, B12, n/2) + mm(A12, B22, n/2);
        mm(A21, B11, n/2) + mm(A22, B21, n/2);
        mm(A22, B12, n/2) + mm(A22, B22, n/2);
    }
}
```

ANALYSIS AND DESIGN OF ALGORITHMS.

16CA205

Thomas H Cormen.

Instructions to Algorithms.

Algorithm :

Algorithm is an well defined computational procedure that states some value or set of values as an input & produce some value or set of values as an output.

properties :-

→ Finiteness
→ Input
→ output
→ Definiteness
→ Effectiveness

Program to add 2 no.s :-

```
#include <stdio.h>
int sum(int , int) ;
int a;
main()
{
int b,c ;
printf ("Enter the 2 nos. ");
Scanf ("%od %od \n", &a , &b);
c = Sum(a , ab);
Printf ("%od", c) ; }
```

int

1. AC)
{
int i ;
for (i = 1 to n) → n+1 — time complexity =
printf ("Hello") ; ⇒ n times.
}

2. AC)
{
int j, i ;
for (i = 1 to n) ; n+1 .
for (j = 1 to n) ; n+1 = (n+1)² = n² + 1 + 2n
= n²
printf ("Hello") ; → n².
}

3. AC)
{
int i, j, k, n ;
for (i = 1 ; i <= n ; i++) . n+1 100(n(n+1)/2)
{ n+1 50n² + 50n
for (j = 1 ; j <= i ; j++) . = n².
{
for (k = 1 ; k <= 100 ; k++) .
printf ("Hello") ;
} }

4. AC)
{
i = 1
for (i = 1 ; i² <= n ; i++) — i² <= n. i <= √n. ↓ time complexity
printf ("Hello") ;
}

5. AC)
{
int i, j, k, n ;
for (i = 1 ; i <= n ; i++) — i = 1 j = 1. k = n/2 .
{ i = 2 j = 4 k = 4 × n/2 .
for (j = 1 ; j <= i² ; j++) — i = k j = k² k = k² × n/2 .
{
for (k = 1 ; k <= n/2 ; k++) . n/2 + 4n/2 .... k² × n/2
{ = n/2 (1 + 2² + 3² + ... + n²)
printf ("Hello") ; n/2 (n(n+1)(2n+1))/6
} = n⁴
}
}

6. AC)
{
for (i = 1 ; i < n ; i = i * 2) . 2^k = n. k = log₂ n
printf ( ) ;
}

- Order / Rate of growth.

Algorithms will execute faster for some values of n. But it will be slow for some other values of n, so that the behaviour of some algorithm changes with increase in the value of n. This change of behaviour is called order of growth.

The order of growth is normally determined for larger value of n.

$\log N < N < N \log N < N^2 < N^3 < 2^N < N!$

- Algorithm design techniques:

2 { 1. Divide and Conquer approach.
2. Greedy Technique.
3. Dynamic programming. → 3
4. Branch and Bound. } 4.
5. Randomized algorithm }
6. Backtracking algorithm → 5

- A C)

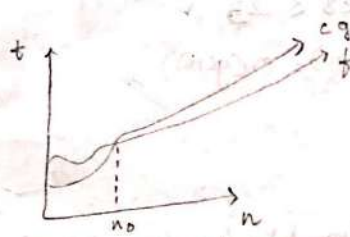
```
{
int i, j, k;
for (i = n/2 ; i<=n ; i++)
for (j=1 ; j<=n ; j = 2*j)
for (k=1 ; k<=n ; k = k*2)
printf ("Hello");
}
```

- Asymptotic Notation

1. Big O notation (O) :

When we have only an asymptotic upper bound we use 'O' notation. The 'O' notation is relax lower bound.

$O(g(n)) = T(n)$ = { f(n) there exists a +ve constants c, n_0 such that $0 \le f(n) \le c\,g(n)$ for all $n \ge n_0$, $c > 0$, $n_0 \ge 1$



1. $f(n) = 3n + 2$. $g(n) = n$.

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

$3n + 2 \le c(n)$

First decide the value of 'c'

if C =

$3n + 2 \le 4n$. $n = 1$.

$3 + 2 \le 4$.

$\times\ 5 \le 4$ (False)

$n = 2$.

$3(2) + 2 \le 4(2)$.

$6 + 2 \le 8$ ✓

2. $f(n) = 11n^3 + 5n^2 + 7$. $g(n) = n^3$

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

$11n^3 + 5n^2 + 7 \le C n^3$.

$C = 23$.

$11n^3 + 5n^2 + 7 \le 23 n^3$.

$n = 1$.

$11 + 5 + 7 \le 23$.

$23 \le 23$ ✓

$f(n) = O(g(n))$ ✓

2. Ω notation :-

When we have only an asymptotic lower bound we use Omega (Ω) notation. And it relax in upper bound.

$\Omega(g(n)) = \{ f(n)$ there exists +ve constants c, n_0 such that $0 \le c\,g(n) \le f(n)$ where $c > 0, n \ge n_0, n_0 \ge 1 \}$

1. $f(n) = 3n+2$. $g(n) = n$.

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

$c\,n \le 3n+2$.

$C = 5$

$5n \le 3n+2$.

$n \ge 1$.

$5 \le 3+2$ ✓

$5 \le 5$

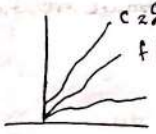
not satisfying for $n = 2$.

∴ we cannot use Ω notation.

3. Θ notation :-

When we have asymptotic lower bound and upper bound. we will use Θ notation.

$\Theta(g(n)) = \{ f(n)$ there exists +ve constants c_1, c_2 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n) \}$.



$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$.

$f(n) = 11n^3 + 5n^2 + 7$.

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

$11n^3 \le 11n^3 + 5n^2 + 7 \le (11+5+7)n^3$

$n = 1$.

$11 \le 23 \le 23$.

$n = 2$.

$88 \le 115 \le 184$.

$n = 3$.

$297 \le 349 \le 621$.

- Recursive equation :-

It is an equation which describes its function in terms of smaller inputs.

Eg :- The $T(n) = \begin{cases} \Theta(1) & n=1 \\ 2T(n/2) + \Theta(n) & n>1 \end{cases}$

There are 3 methods to solve the recurrence relation :-

1. Substitution method.
2. Recursion tree method.
3. Master method.

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

$T(n) = T(n-1) + 1$. —(1).

$T(n-1) = T(n-2) + 1$ —(2)

$T(n-2) = T(n-3) + 1$ —(3)

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

sub (2) in (1)

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

$T(n) = T(n-2) + 2$ —(4).

sub. (3) in 2.

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

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

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

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

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

$= x + n - 1$.

$= n$.

$= O(n)$.

2. $T(n) = T(n-1) + n$ —(1)

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

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

Sub. (2) in (1).

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

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

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

$T(n) = T(n-k) + (n-(k-1)) + \ldots (n-2) + (n-1) + n$

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

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

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

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

$= O(n^2)$.

3. $T(n) = 2T(n/2) + n$ —(1)

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

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

$n = n/2$ in (1)

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

$n = n/4$ in (1)

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

$n = n/8$ in (1).

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

Sub (2) in (1)

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

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

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

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

$\frac{n}{4}$, $\frac{2}{1}$, $\frac{n}{4} \times \frac{1}{2}$.

$\frac{n}{4} \times \frac{1}{4}$. $\frac{n}{16}$

$\frac{n}{4}$, $\frac{4}{1}$

$\frac{n}{4} \times \frac{1}{4}$. $\frac{n}{16}$.

$n \times \frac{2n}{2}$, $\frac{3n}{2}$

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

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

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

$= 2^{\log_2 n}$ $= b^{\log_a a}$

$= \log n + \log_2 n \, n$.

$= O(\log n)$.

$= n^1 + n\log_2 n$.

$= O(n\log_2 n)$

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

$n = 2^k$.

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

$n = 2^k$.

$2^k = n$.

$k\log 2 = \log_2 n$.

$k = \log_2 n$.

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

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

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

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

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

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

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

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

$= 2^6 T\left(\frac{n}{2^3}\right) + 2^4 \frac{n}{2^2} + 2n + n.$

$= 2^6 T\left(\frac{n}{2^3}\right) + 2^2 n + 2n + n.$

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

$= 2^{2(\log_2 n)} T\left(\frac{n}{2^{\log_2 n}}\right) + 2^{\log_2 n - 1} \cdots 2^2 n + 2^1 n + 2^0 n.$

$= 2^n T\left(\frac{n}{2^{\log_2 n}}\right) + 2$

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

$= n^2 + n.$

$= n^2$

$= O(n^2).$

$2^{\log_2 n - 1}.$

$2^{\log n - b}.$

$2^{\log_2 n} \cdot 2^{-1}.$

n

5. $T(n) = 2T\left(\frac{n}{2}\right) + 4n$ —(1)

Build solution

$T(n) = 2T\left(\frac{n}{2}\right) + 4n$ —(1)

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

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

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

Expand Equation.

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

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

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

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

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

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

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

$\frac{n}{2^k} = 1 \quad T(n) = 2^{\log_2 n} T(1) + \log_2 n\, 4n.$

$a^{\log_a b} = b^{\log_a a} = b$

$\log_a a = 1.$

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

$= n^1$

$\frac{n}{2^k} = 1 \quad T(n) = \frac{\log n}{2}$

$n' + (1) + 4n\log_2 n$

$= O(n\log_2 n)$

6. $T(n) = 2T\left(\frac{n}{2}\right) + 2n$ —(1)

$T(n) = 2T\left(\frac{n}{2}\right) + 2n$ —(1)

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

$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + 2\left(\frac{n}{2}\right)\right] + 2n.$

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

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

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

$= O(n\log_2 n - n) = O(n\log_2 n)$

7. $T(n) = T\left(\frac{n}{2}\right) + n$ —(1)

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

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

(2) in (1)

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

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

$= 1 + \frac{n}{\log_2 n} + n$

$= O\left(\frac{n}{\log_2 n}\right)$

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

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

$= 1 + \frac{n}{2^{\log_2 n - 1}}$

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

$=$

8. $T(n) = T\left(\frac{n}{2}\right) + 2n$ —(1)

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

(2) in (1)

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

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

$= 1 + 3n$

$= O(n)$

3n + 2n

9. $T(n) = 3T\left(\frac{n}{2}\right) + n$ —(1)

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

(2) in (1)

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

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

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

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

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

$= 3^k T\left(\frac{n}{2^k}\right) + 3^{k-1}\frac{n}{2^{k-1}} + \cdots + 3^2\frac{n}{2^2} + 3^1\frac{n}{2} + 3^0\frac{n}{2^0}$

$= 3^{\log_2 n} + 3^{\log_2 n - 1}\left(\frac{n}{2^{\log_2 n - 1}}\right) + \cdots + n$

$= 3^{\log_2 n} + 3^{\log_2 n - 1}\left(\frac{n}{2^{\log_2 n - 1}}\right) + \cdots + n$

$= 3^{\log_2 n} + \frac{3^{\log_2 n}}{3}\left(\frac{1}{2}\right) + \cdots + n$

$= 3^{\log_2 n} + \frac{3^{\log_2 n}}{6} + \cdots + n$

$T(n) = n^{\log_2 3}$

$\therefore T(n) = O(n^{\log_2 3})$

2. Recursive tree method :- In this method we will construct a recursion tree where each node represent the cost of single sub problem. We sum off all the cost within each level of the tree, then we sum off the cost at each level into one.

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

level	node	size of sub problem	cost of each level
0	1	n	$1 \times n$
1	2	$n/2$	$2 \times \frac{n}{2} = n$
2	2^2	$\frac{n}{2^2}$	$2^2 \times \frac{n}{2^2} = n$
.	.	.	.
.	.	.	.
k	2^k	$\frac{n}{2^k}$	$2^k \times T(1) = 2^k$


```
void Test (int n) -> T(n)
 {
 if (n > 1)
 {
 for (i=0 ; i<n ; i++) -> n.
  {
 stmt ;
  }
  Test (n/2)  -> T(n/2)
  Test (n/2)  -> T(n/2)
 }
}
```

Add, we get.

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

Cost of each level = Total no. of node in each level × cost of size of node.

Total cost of = Cost of leaf level + cost of remaining level.

cost of leaf level = $2^k \times T(1)$

$= 2^k$.

cost of remaining level = $n + n + n \ldots n^{k-1}$.

$= \sum_{i=0}^{k-1} n$.

$= (k-1)\, n$.

Total cost = $2^k + (k-1)n$.

$= 2^k + (\log_2 n - 1)\, n$.

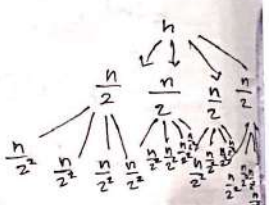
$= 2^{\log_2 n} + (\log_2 n - 1)\, n$

$= n + n\log_2 n - n.$

$= O(n \log_2 n).$

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

level	node	size of SP	cost of each level
0	1	n	n
1	4	$\frac{n}{2}$	$2^2 \times \frac{n}{2} = 2n$
2	16	$\frac{n}{2^2}$	$2^4 \times \frac{n}{2^2} = 2^2 n$
.	.	.	.
.	.	.	.
.	.	.	.
k	2^{2k}	$\frac{n}{2^k}$	$2^{2k} * T(1) = 2^{2k} = 4^k$



cost of leaf level $= 2^{2k} = 4^k$

cost of remaining level $= n + 2n + 2^2 n + \cdots 2^{k-1} n$

$= \sum_{i=0}^{k-1} 2^i n.$

Total cost $= 2^{2k} + 2^k n.$

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

$= n(n-1)$

Total cost

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

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

$= 2n^2 - n.$

$= O(n^2)$

$4^k.$

$4^{\log_2 n}.$

$= n^2.$

$S = \frac{a(r^k - 1)}{r - 1}$

$a = 1$

$r = 2$

$= 1(2^k - 1)$

$= 2^k - 1.$

$= 2^{\log_2 n} - 1$

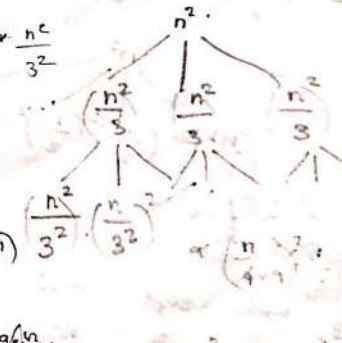
$= n - 1.$

$2^0 = 1$

$a = 2$

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

level	node	size	cost
0	1	n^2	n^2
1	3	$\left(\frac{n^2}{3}\right)$	$\frac{n^2}{3}$
2	$9 = 3^2$	$\left(\frac{n}{3^2}\right)^2$	$\frac{n^2}{3^2}$
.	.	.	.
k	3^k	$\left(\frac{n}{3^k}\right)^k$	$\frac{n^k}{3^k}$ $3^k * T(1)$



cost of leaf level $= \frac{n^k}{3^k} = \frac{n^{\log_3 n}}{3^{\log_3 n}} =$

$3^k * T(1) = 3^k$

$= 3^{\log_3 n}.$

$= n.$

Total cost = $3^k + \left(n^2 + \frac{n^2}{3} + \frac{n^2}{3^2} + \cdots + \frac{n^2}{3^{k-1}}\right)$

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

$= n + \frac{3}{2}n^2 - \frac{3}{2}n$

$= O(n^2)$

$S = \frac{a(1-r^k)}{1-r}$

$= \frac{1\left(1-\left(\frac{1}{3}\right)^k\right)}{1-\frac{1}{3}}$

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

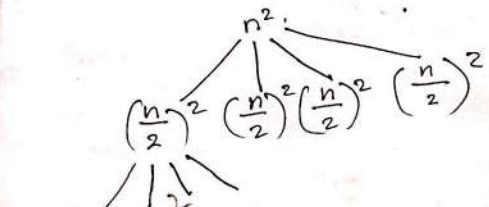
$= \frac{3}{2}\left(1-\frac{1}{n}\right)$

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

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

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

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



level	node	size	cost
0	1	n^2	n^2
1	$4=2^2$	$\frac{n}{2}$	n^2
2	2^4	$\frac{n}{2^2}$	n^2
⋮			
k	$2^{2k} = 4^k$	$\frac{n}{2^k}$	$4^k * T(1)$

$2^2 * \frac{n^2}{2^2}$

$2^4 \cdot \frac{n}{2^2 * 2^2}$

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

$n = 2^k$

$k = \log_2 n$

cost of leaf level = 4^k.

$= 4^{\log_2 n}$.

$= n^2$.

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

$\frac{n}{3^k} = 1$.

Total cost = $n^2 + \left(n^2 + n^2 + n^2 \cdots + n^2\right)$ ($k-1$ terms)

$= n^2 + \sum_{i=0}^{k-1} n^2$.

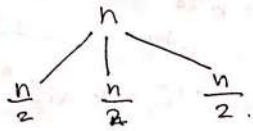
$(k-1)\,n^2$.

$= n^2 + (\log_2 n - 1)n^2$.

$= n^2 + n^2 \log_2 n - n^2$.

$= O(n^2 \log_2 n)$

- $T(n) = 3T\left(\frac{n}{2}\right) + n$.



level	node	size	cost
0	1	n	n
1	3	$\frac{n}{2}$	

$\frac{4-1}{4} \cdot \frac{-3}{4} \cdot \frac{2-3}{2}$

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

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

$n(n-3^k)$

$n-3k$

$n^{1.584} + n - n^{1.584}$

$n^1 - n^{1.584}$

$n^{0.584}$

3. MASTER METHOD.

The master method is based on master theorem, master theorem says, if a recursion relation is in the form of

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1 \quad b > 1$$

where, a = no. of sub problems.

$f(n)$ = cost of divide & combine.

then b = size of sub problem related to n.

$\frac{n}{b}$ = size of input to each recurrance relation.

The cost of given $f(n)$, in this method Compare with $n^{\log_b a}$.

case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ $\epsilon > 0$.

case 2: If $f(n) < n^{\log_b a}$

Then the solution is $T(n) = \theta(n^{\log_b a})$

case 2: If $f(n) = \theta(n^{\log_b a})$

i.e., $f(n) = n^{\log_b a}$

Then the solution is $T(n) = \theta(n^{\log_b a} \log n)$

Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\epsilon > 0$.

i.e., $f(n) > n^{\log_b a}$.

Then the sol^n is $T(n) = \theta(f(n))$

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

1. $T(n) = 9T\left(\frac{n}{3}\right) + n$

step 1. $a = 9$.

$b = 3$.

$f(n) = n$.

step 2 : $n^{\log_b a}$

$n^{\log_b a} = n^{\log_3 9}$ $\qquad \log_3 9 = \log_3 3^2 = 1^2$

$= n^2$.

$f(n) \quad n^{\log_b a}$.

$n < n^2$.

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

$= \theta(n^2)$.

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

$n = \frac{3}{2}$.

2. $T(n) = T\left(\frac{2n}{3}\right) + 1$.

$a = 1$.

$f(n) = 1$.

$b = \frac{3}{2}$

step $n^{\log_{\frac{3}{2}} 1}$

$= n^0 = 1$.

$f(n) \quad n^{\log_b a}$.

$1 \quad = 1$

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

3. $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$.

$a = 3$.

$b = 4$.

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

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

$= n^{0.8}$.

$f(n)$. $\quad n^{\log_b a}$

$n \log n \; > \; n^{0.8}$.

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

$= \theta(n \log n)$

Linear Search (x, A).

1. Set i to 1.
2. If $i > n$ then go to step 7.
3. If $A[i] = x$, then go to step 6.
4. Set i to i+1.
5. Go to step 2.
6. Print Element x found at index i & go to step 8.
7. Print 'Element not found'.
8. Exit.

Pseudo code:

Procedure linear_search (list, value). $= n+1$

for each item in the list. $= n$.

if match item = value. $= n$.

return the item's location. -1

end if. -1

End for. -1

End procedure. -1

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

$= 2n + 5$.

$> 2n$.

$= O(n)$.

Analysis.

$A[] = 20, 30, 5, 61$

$n = 5 \quad x = 5$.

(1) $i = 0$.

$A[0] == x$

$20 \neq 5$.

(2) $i = 0+1 = 1$

$A[1] == x$.

$30 \neq 5$.

(3). $i = 2$. $\quad A[2] == x$.

$5 == 5$

- Binary search (x, A, n, low, high) → T(n)

1- Set low = 0. & high = n-1. → 2
2. mid = (low + high)/2 → 3. if (low <= high) → 1
3- if (A[mid] == x) go to step 9. → 1.
5. if (A[mid] > x) → 1
6. binary (x, A, n, low, mid-1) → $\frac{n}{2}$
7. if (A[mid] < x) → 1
8. binary (x, A, n, mid+1, high) → $\frac{n}{2}$
9. Print element found.
10. End if. → 1
11. Endif. → 1
12. Element not found.

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

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

$= O(\frac{n}{2})$

- Pseudocode

Procedure binary-search (x, A, n, low, high)
mid = (low + high)/2.
if. low <= high.
If. mid element == x.
print element found.
if mid element > x. mid-1.
binary (x, A, n, low, mid-1, high).
if mid element < x.
binary (x, A, n, mid+1, high)
Print element found.
End if
End if.
End procedure.
Print element not found.

- Analysis.

n = 5. A[5] = {10, 20, 30, 40, 50}

x = 40 low = 0. high = n-1.

① mid = (low + high)/2.

$= \frac{0+4}{2} = 2.$

A[mid] == x.

30 ≠ 40.

a[mid] > x.

30 > 40 ✗

a[mid] < x.

30 < 40 ✓

x = 40, low = mid + 1, high = 4.

low <= high.

$mid = \frac{3+4}{2} = 3.$ 3 <= 4.

40 = 40 ✓

- Factorial. fact(n);

1. Set i=1. → 1
2. If [i<=n]. → 1
3. return n * fact(n-1) → n-1
4. return 1; → 1

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

- Insertion Sort :

Algorithm: 1. If it is the first element, it is already sorted. return 1;

Insertion sort (A)

```
{
for j = 2 to A length.
    key = A[j];
    i = j-1;
while (i>0 and A[i] > key)
{
A[i+1] = A[i]
    i = i-1.
}
A[i+1] = key;
}
```

14 33 27 10 35 19

14 33 27 10 35 19

14 27 33 10 35 19

14 27 10 33 35 19

14 27 10 33 35 19

Eg: 14 33 27 10 35 19.

n = 6.

① j = 2 to 6.

key = A[j] = A[2] = 33.

i = j-1 = 2-1 = 1.

while (i > 0 && A[i] > key)

{}

A[i+1] = key.

}

temp = 33

14 = 14

temp = 14

while (j = 2 && j-1 ;

14 > 33 X

14 33

A[1] a[2]

j >= 0

(2) $g=3$ to 6. 14 33 27 10 35 19

key = A[j] = 27.

i = j-1 = 3-1 = 2.

while (i>0 & A[i] > key)

1>0 ✓ 33 > 27 ✓

A[i] > key

14 > 27 ✗

{

A[i+1] = A[i]

A[3] = A[2]

i = i-1 = 2-1 = 1.

}. repeat the while loop & compare.

A[2] = 27.

14 27 33 10 35 19

- Worst case time complexity

	compare	+	move	
g = 2	1	+	1	= 2
j = 3	2	+	2	= 4
j = 4	3	+	3	= 6.
j = n	(n-1)	+	(n-1)	= 2(n-1)

$= 2 + 4 + 6 + 8 + \cdots 2(n-1)$

$= 2(1 + 2 + 3 + 4 + \cdots (n-1))$

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

$= O(n^2)$

~~1. T(n) = TT(n)~~

- Radix sort :-

pseudo code :-

```
Radix sort (a[], n)
{
max data = get max (a[], n)
for (exp = 1; max data / exp > 0; exp = exp * 10)
{
count sort (a[], n, exp);
}}
algo count sort (a[], n, exp)
{
initialize count [0 .... 9] with 0's.
for (i=0; i<n; i++).
{
count [(a[i] / exp) % 10]++;
}
for (i=1; i<10; i++).
{
count[i] = count[i] + count[i-1]
}
```

```
for (i= n-1 ; i>=0 ; i--)
{
output [count [(a[i]/exp) % 10] -1] = a[i];
 count [(a[i]/exp) % 10] --;
 }
for (i=0 ; i<n ; i++)
{
a[i] = output[i];
 }
}
```

1st for loop

count = [(247/1) % 10]

count = [247 % 10]

count [7] ++ .....

count[10]

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

1st count

3rd for loop

```
for [i=1 ; i<10 ; i++)
{
count [i] = count[i] + count[i-1]
}
```

7<10

count [7] = count [7] + count [6]

= 2 + 5

= 7.

count[] 2nd count

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

0

4th for loop

after 4th for loop

i = n-1 = 10-1 = 9

i >= 0

9 >= 0

output [count [(a[i]/exp) % 10] -1] = a[i]

output [count [(930/1) % 10] -1]

count[0] - 1

1-1

output [0] = 930

count [(930/1) % 10] count[0]--

~~count [(930/1) % 10~~

i = 8.

output [count [(758/1) % 10] -1]

count [8] -1.

8-1.

output [7] = 758

count [(758/1) % 10] --

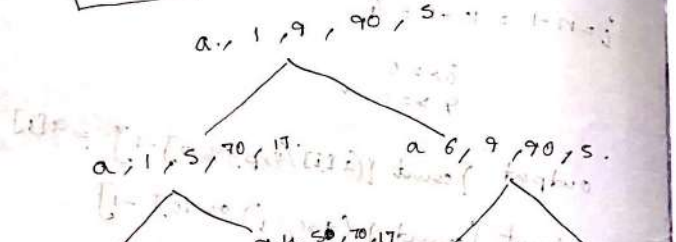
count [8] --

= 7

Divide and Conquer

- min & max

1	2	3	4	5	6	7	8	9
25	30	50	70	17	23	5	90	16



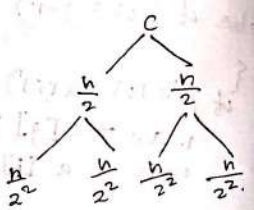
- Pseudo code:

```
DAC — maximum min (a, i, j)
{
if (i = j)
{
max = min = a[i];
return (min, max)
}
```

```
else 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
{
mid = (i+j)/2
max1, min1 = DAC_maxmin(a, i, mid).
max2, min2 = DAC_maxmin(a, mid+1, j)
  if (max1 < max2)
    max = max2.
  else
    max = max1
  if (min1 < min2)
    min = min1
  else
    min = min2.
  return (max, min)
}
}
```

Time Complexity.

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



$T(n) = 2\,T\left(\frac{n}{2}\right) + c \quad -(1)$

level	node	size of subproblem.	cost of each level.
0	1	n.	n.
1	$2 = 2^1$	$\frac{n}{2}$.	n.
2.	$4 = 2^2$	$\frac{n}{2^2}$	n.
3.	$8 = 2^3$	$\frac{n}{2^3}$	
:			
k.	2^k.	$\frac{n}{2^k}$.	2^k. $\times T(1)$

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

$k = \log_2 n.$

Total cost = cost of leaf level + cost of remaining level.

$= 2^k + (n + n + n \cdots \overset{k-1}{n})$

$= 2^k + \sum_{i=0}^{k-1} n.$

$= 2^k + (k-1)\,n.$

$= 2^{\log_2 n} + (\log_2 n - 1)\,n.$

$= n + n\log_2 n - n.$

$= O(n \log_2 n)$ (In Mo's book :) Quicksort.

- Merge Sort (A, l, h).

```
if (l<h)
{
mid = l+h/2.
merge sort (A, mid+1, h).
merge sort (L, R, A).
}
mergesort (L, R, A)
{
NL ← length(L)
NR ← length(R).
i=0.
j=0.
R=0.
while (i<nL && j<nR)
{
if (L[i] <= R[j])
{
(A[R] ← L[i])
k++;
i++;
}
else.
{
A[R] = R[i]
k++;
j++;
}
```

```
for (i=0; i<= mid ; i++)
 L [i] = A [i] .
 }
 for (mid+1 ; i<= high ; i++)
   R [i] = A[i] .
   }
      (In Mols book)
```

```
#include <stdio.h>.
void main()
{
int r1 , c1 , r2 , c2 ) i j ;
printf ("Enter no. of rows & columns of matrix A\n");
scanf ("%d %d", &r1, &c1);
printf ("Enter the no. of rows &, columns of matrix B\n");
scanf ("%d %d", &r2, &c2);
int a[r1][c1] , b[r2][c2];
printf ("Enter the elements of matrix A");
scanf ("%d", &a[i][j]);
for (i=0 ; i< r1 ; i++) .
{
for (j=0 ; j< c1 ; j++).
{ scanf ("%d", &a [i] [j]);
printf
} }
```

```
printf ("Enter the elements of matrix B");
scanf ("%d", &b [r2][c2]);
for (i=0 ; i<r2 ; i++)
{
for (j=0 ; j<c2 ; j++)
{ scanf ("%d", &b [i] [j]);
printf
} } if (c1 == r2)
int D[r3] [c3] .
D[r3] [c3] = A [r1] [c1] * B[r2] [c2] ;
printf ("The multication of %d and %d is
                %d, A[r1] [c1] ,
                B [r2] [c2], D[r3] [c3]
}
else.
{ printf ("Not possible");
} . sum=0;
{ for (i=0; i<r1 ; i++).
{
for (j=0 ; j<c2; j++) .
{ c [i] [j] = 0 ;
for (k=0 ; k <r2 ; k++) .
{ c [i] [j] = c [i] [j] + a [i] [j] * B [i] [j];
} }
```

- Matrix Multiplication :-

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = B_{11}(A_{21} + A_{22})$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{12} - B_{11})$

$T = B_{22}(A_{11} + A_{12})$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$C_{11} = R + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

- Divide and conquer.

```
mm(A, B, N)
{
if(n <= 2)
{
C = 4 formula.
}
else
{
mm(A11, B11, n/2) + mm(A12, B21, n/2);
mm(A11, B12, n/2) + mm(A12, B22, n/2);
mm(A21, B11, n/2) + mm(A22, B21, n/2);
mm(A22, B12, n/2) + mm(A22, B22, n/2);
}
```

$$\begin{bmatrix} (a_{11} * b_{11}) + (a_{12} * b_{21}) & (a_{11} * b_{12}) + (a_{12} * b_{22}) \\ (a_{21} * b_{11}) + (a_{22} * b_{21}) & (a_{21} * b_{12}) + (a_{22} * b_{22}) \end{bmatrix}$$

Addition = 4 times
$= N^2$.

Time complexity =

$$T(n) = 8T\left(\frac{n}{2}\right) + N^2.$$

MODULE - 2.

Graph :- It is a pictorial representation (photo copy) In data str....

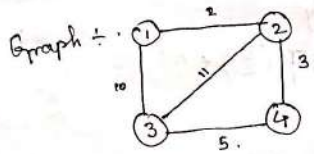
- Minimum spanning tree :-

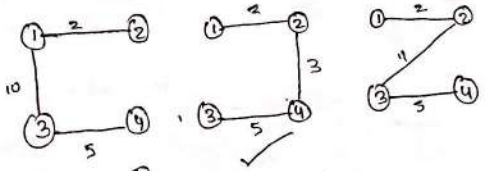
It is a tree which connects all the vertices of the graph without any cycles or loops.

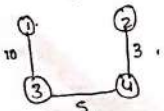
- Minimum cost spanning tree :-

A spanning tree's weight is minimum for all possibilities is called minimum cost spanning tree.

Graph :-



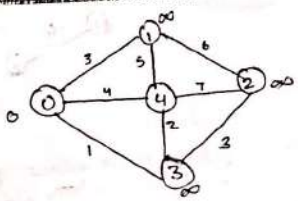


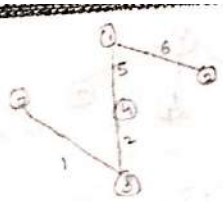


Minimum cost spanning tree is implemented by 2 greedy algorithms.

- Prim's.
- Kruskal's.

→



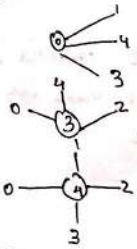


∞ = 999

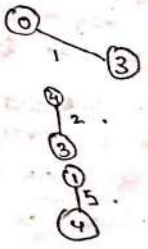
	0	1	2	3	4
0	0	3	999	1	4
1	3	999	6	999	5
2	999	6	999	3	7
3	1	999	3	999	2
4	4	5	7	2	999

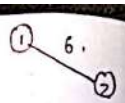
Finding Spanning tree using Normalisation method.

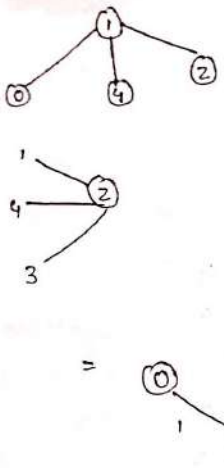
1. MST set = {0, 3, 4, 1, 2}.
2. Source vertex cost = 0.
3. Source.



4. Spanning tree.







= 14

- Prim's Algorithm :

Step 1. Create a set mstset that keeps track of vertices already visited included in mst set

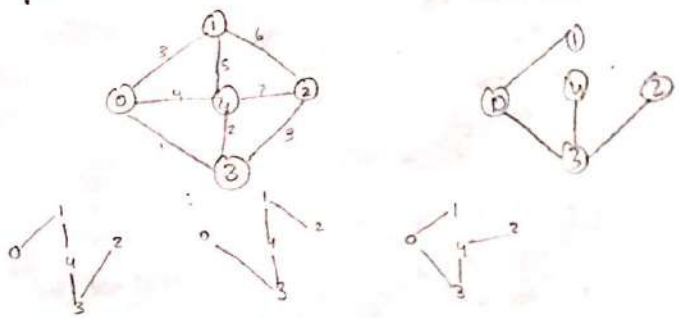
2. Assign a key value to all vertices in the input graph.

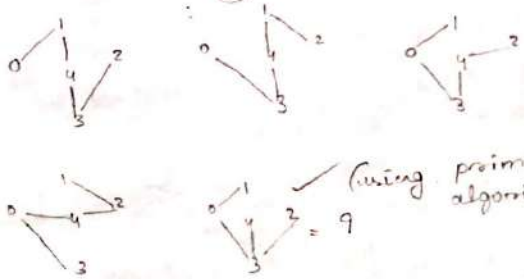
Initialize all key values as INFINITE.
Initialize all mst set as FALSE
Assign key value as zero for the first vertex so that it is picked first.
Zero

3. While mstset doesn't include all vertices.
→ pick a vertex u which is not there in mst set & has minimum key value.
→ Include u to mstset.

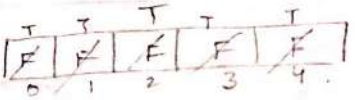
→ update key value of all the adjacent vertices of u.

To update the key values, iterate through all adjacent vertices for every adjacent vertex v If weight of edge. u+v is less than the previous key values of v update key value of v update key value of v as weight of u+v.

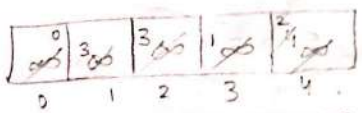




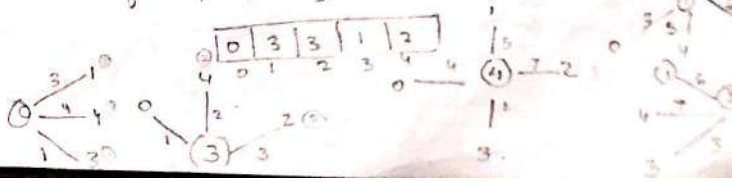
(using prim's algorithm)

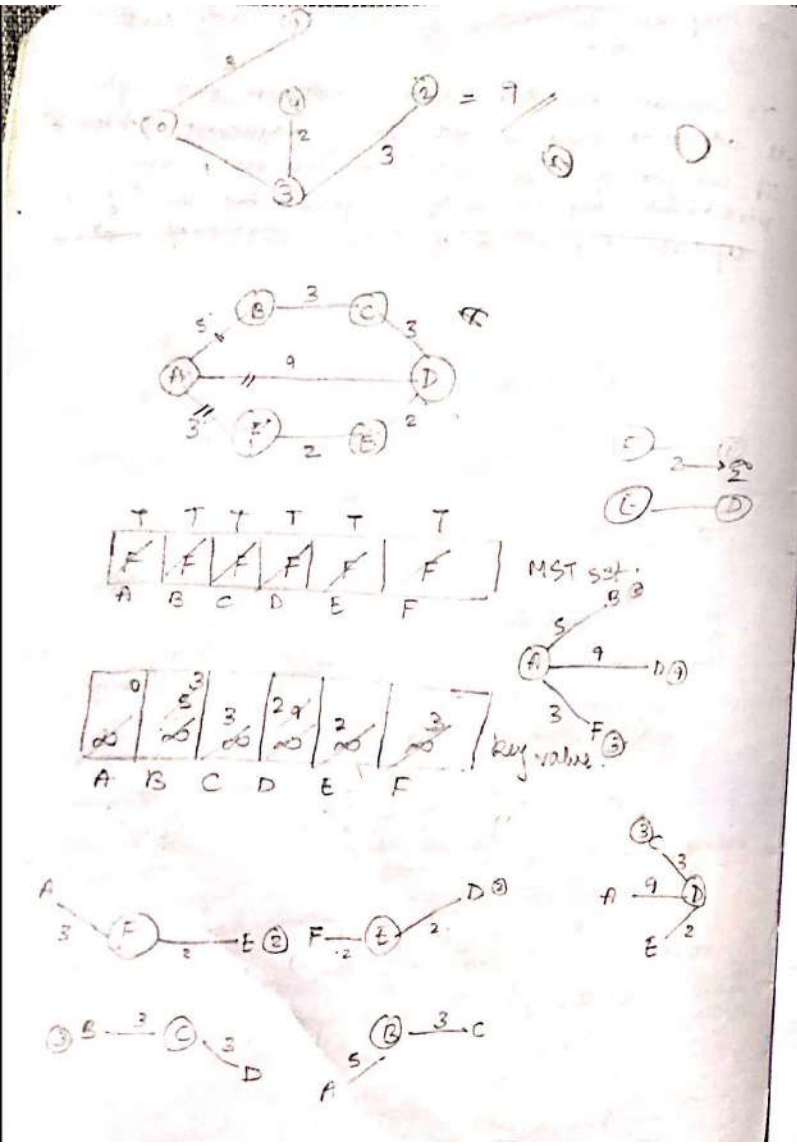


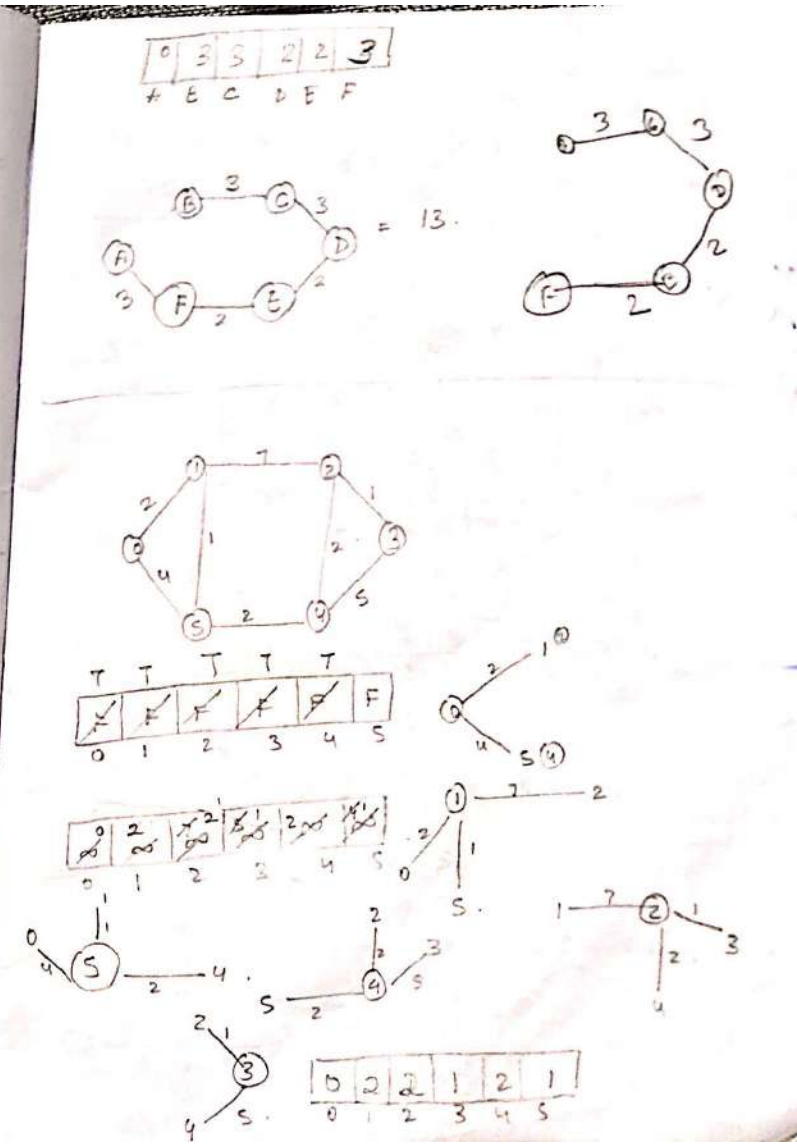
MST SET [V]

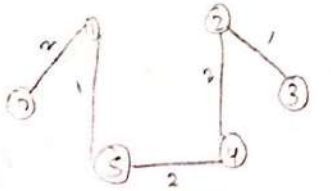


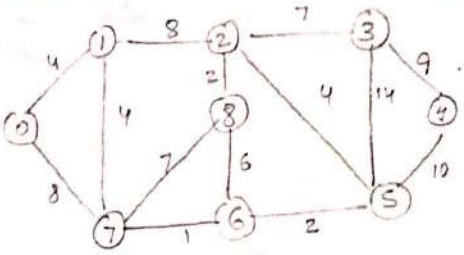
key value [v]

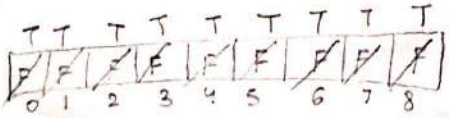


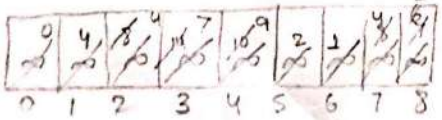


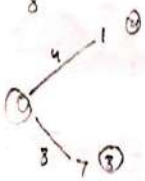


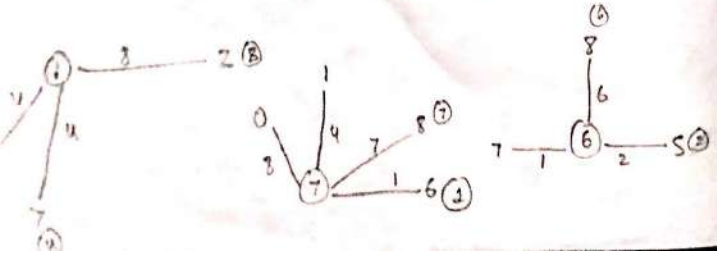


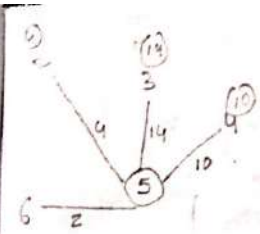


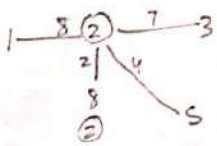


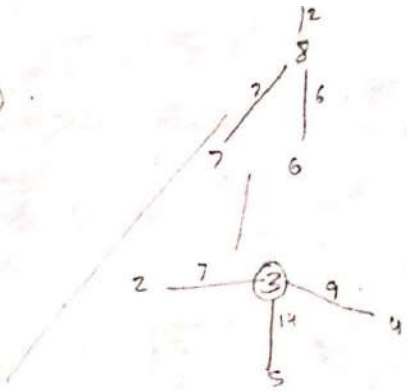


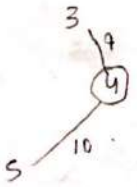


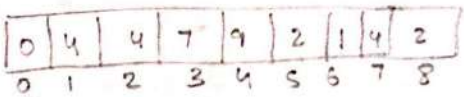


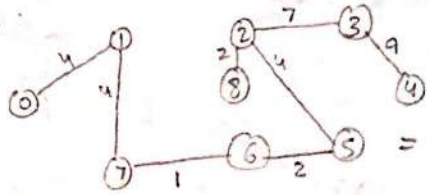


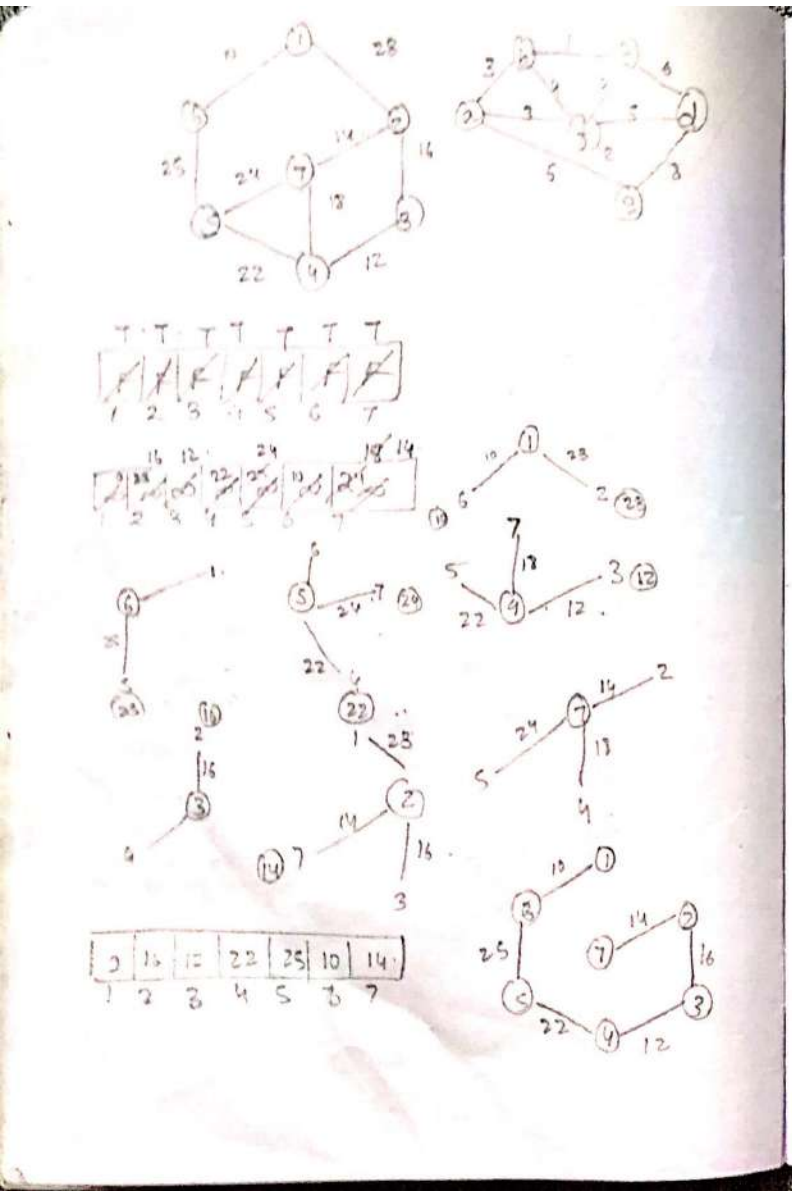


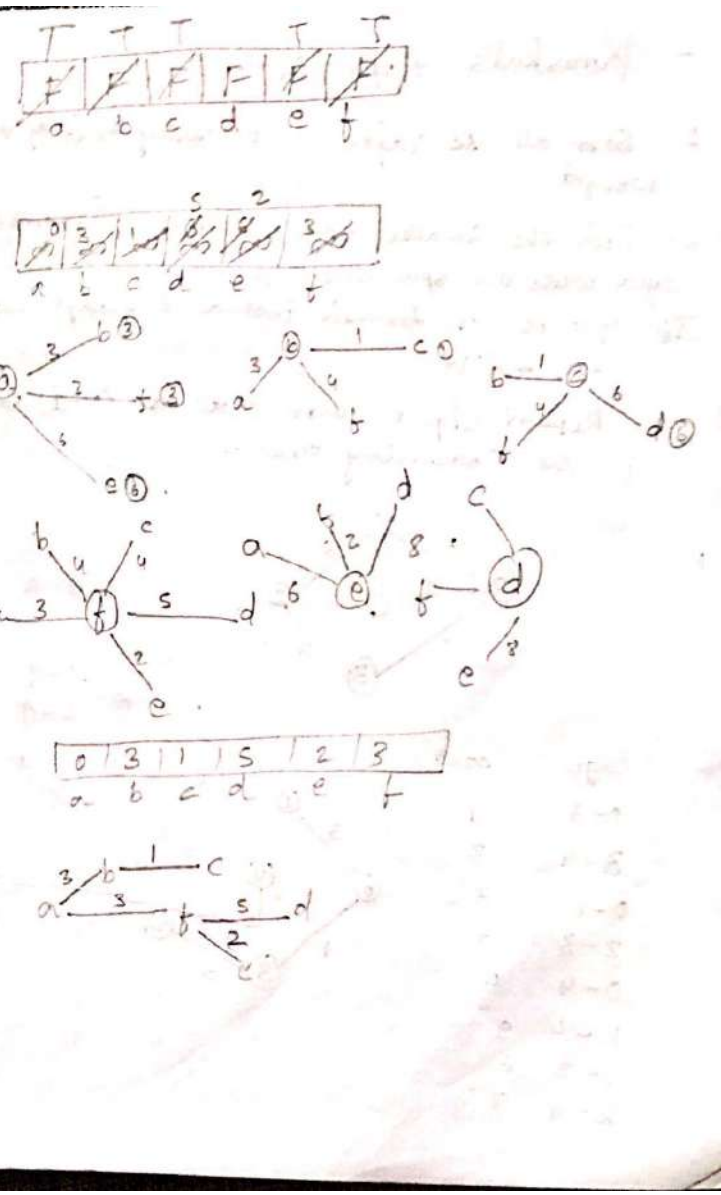






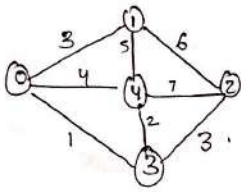




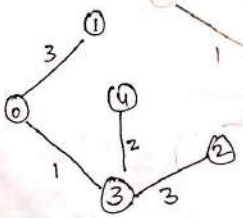


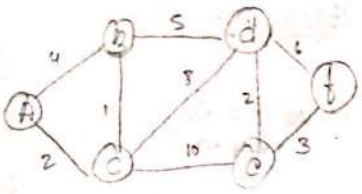
- Kruskal's Algorithm

1. Sort all the edges in increasing order of their weight.
2. Pick the smallest edge, check if it forms a cycle with the spanning tree formed so far. If cycle is not formed include the edge else discard it.
3. Repeat step 2 until there are (v-1) edges in the spanning tree.

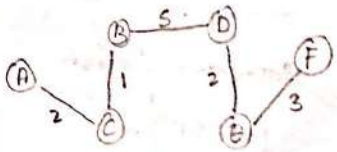


Edge.	cost.
0-3.	1
3-4	2
0-1	3.
2-3.	3
0-4	4.
1-4	5
1-2	6
2-4	7.





No.	Edge.	Cost.
1	B-C.	1.
2.	A-C.	2.
3	D-E	2.
4.	E-F	3.
5.	A-B.	4. X
6.	B-D.	5.
7.	D-F	6 X
8.	D-C.	8 X
9.	C-E	10 X.



– KRUSKALIS PSUEDOCODE :-

1. A = 0
2. for each vertex. $v \in G.V$
3. Make – SET(v).
4) Sort the edges of G.E in ↑ng order by weight w.
5. for each edge (u, v) E G.E when in ↑ng order by weight.
6. If FND-SET (u) ≠ FIND-SET(v)
7. A = A ∪ {(u, v)}.
8. UNION (u, v)
9. return A.

Shortest path Algorithm :-

Types of path problem.

Single Source shortest path. (SSSP)
- Bellman ford
- Dijkstra's

All-pair shortest path. (APSP).
- Floyd-warshalls.
- Johnson's.

Dijkstra's Algorithm :-

1. Assign every node a tentative distance.
2. Set initial node as current and mark all other nodes as unvisited.
3. For current node consider all unvisited nodes & calculate tentative distance. Compare current distance with calculated distance & Assign the smaller value.
4. When all the neighbors are considered of the current node, mark it visited {visited node is never checked again}
5. If the destination node is marked visited stop.
6. End.

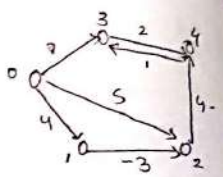
—Dijkstra's (G, w, s).

1. Initialize - single - source (G, s)
2. S = ϕ
3. Q = G.V.
4. while Q ≠ ϕ.
5. u = EXTRACT_MIN (Q)
6. S = S ∪ {u}.
7. For each vertex V ∈ G. adj [u]
8. RELAX (u, v, w).

- BELLMAN FORD (G, w, s).

1. Initialize - Single Source (G, s).
2. for (i=1 to |G.v|-1)
3. for each edge (u,v) ∈ G.E.
4. Relax (u, v, w)
5. for each edge (u,v) ∈ G.E
6. If v.d > u.d + w(u,v).

Then d[v] = d[u] + w(u,v].



Iteration 1 :

- Step 1 : 0.d = 0.
  1.d = ∞ 4
  2.d = ∞ 5 1.
  3.d = ∞ 8. 6.
  4.d = ∞ 5.

Step 2 : i=1 to 4.

Step 3 : {(0,1),(0,2),(0,3),(1,2),(2,4),(3,4),(4,3)}

Step 5 : (0,1).
u, v.

Step 6 : 1.d > 0.d + w(0,1).
∞ > 0 + 4.
∞ > 4 ✓

& 1d = 0+4. 1d=4 ∴

Iteration 2 :

Step 5 : (0,2).

Step 6 : 2.d > 0.d + w(0,2).
∞ > 0 + 5.
∞ > 5

2d = 5.

Iteration 3 :

Step 5 : (0,3).

Step 6 : 3d > 0.d + w(0,3).
∞ > 0 + 8.
∞ > 8.
3d = 8.

Iteration 4 :

Step 5 : (1,2).

Step 6 : 2d > 1.d + w(1,2).
5 > 4 + (-3).
5 > 1.
2d = 1.

Iteration 5 :

Step 5 : (2,4).

Step 6 : 4d > 2d + w(2,4).
∞ > 1 + 4.
∞ > 5.
4.d = 5.

~~Iteration 6 :-~~

Step 5 :- (3,4)

Step 6 :- $4d > 3d + w(3,4)$.

$5 > 8 + 2$.

$5 > 10$ X .

~~Iteration 7 :-~~

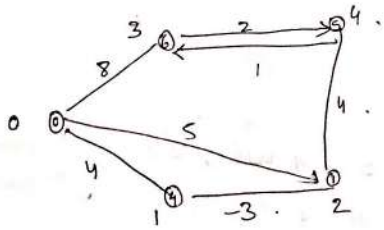
Step 5 :- (4,3).

Step 6 :- $3d > 4d + w(4,3)$.

$8 > 5 + 1$.

$8 > 6$.

$3.d = 6$.



Iteration 2 :-

Step 1 :- $0d = 0$

$1d = 4$.

$2d = 1$.

$3d = 6$

$4d = 5$.

Step 3 :- {(0,1), (0,2), (0,3), (1,2), (2,4), (3,4), (4,3)}

Step 5 :- (0,1).

Step 6 :- $1d > 0.d + w(0,1)$.

$4 > 0 + 4$.

$4 > 4$ X .

Step 5 :- (0,2).

Step 6 :- $2d > 0.d + w(0,2)$.

$1 > 0 + 5$.

$1 > 5$ X .

Step 5 :- (0,3).

Step 6 :- $3d > 0.d + w(0,3)$.

$6 > 0 + 8$.

$6 > 8$ X .

Step 5 :- (1,2).

Step 6 :- $1 > 4 + (-3)$

$1 > 1$ X .

Step 5 :- (2,4).

Step 6 :- $5 > 1 + 4$.

$5 > 5$

Step 6 :- (3,4).

Step 6 :- $5 > 6 + 2$.

$5 > 8$ X .

Step 5 ÷ (4, 3).

Step 6 ÷ is 6 > s + 1.

6 > 6 ·X.

No change in 2nd interation.

∴ No change in 3rd & 4th iteration also.

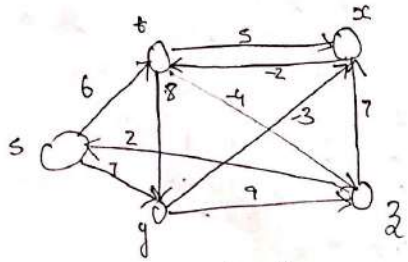
→ 0 − 0 → 0.

0 − 1 → 4. (Because it has min. value)

0 − 1 − 2 → 4 − 3 = 1.

0 − 1 − 2 − 4 → 4 − 3 + 4 = 5.

0 − 1 − 2 − 4 − 3 → 4 − 3 + 4 + 1 = 6.



Step 1 ÷

s.d = 0.

t.d = ∞

y.d = ∞

x.d = ∞

z.d = ∞

Step 2 ÷ iteration 1.

{(s, y), (s, t), (4, 2), (2, s), (4, x), (2, x), (x, t), (s, t, x), (t, z), (t, 4)}.

- DYNAMIC PROGRAMMING :-
used to solve
- Optimisation problem
similar to
- Divide & Conquer.
- To avoid overlapping of subproblem.

There are 2 methods of solving Dynamic programming

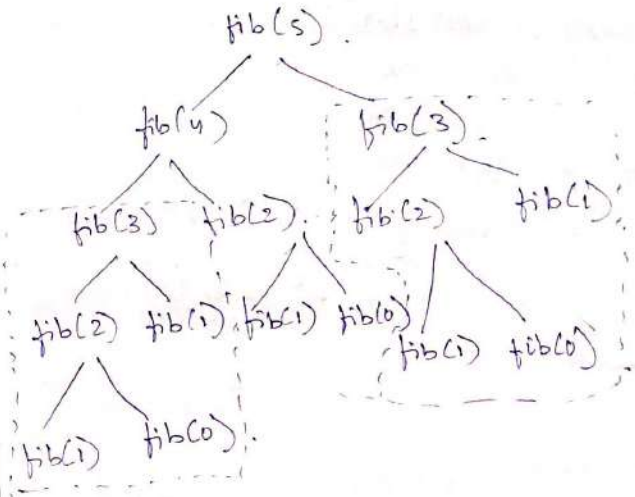
- General method :-

eg :- Fibonocci.

0	1	1	2	3
0	1	2	3	4

```
int fib(int n)
{
if(n<=1)
{
return n;
}
else
return fib(n-1)+fib(n-2).
```

n = 5

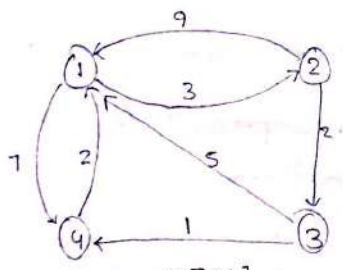


- ~~Dynamic Programming~~ :-

→ All-pair shortest paths.
1. Floyd's Algorithm.
2. Washall's Algorithm.

- Algorithm floyd's (cost, A, n)
1. Cost [i:n] [1:n].
2. {
3. for i=1 to n do.
4. for j=1 to n do.

5. A[i,j] = cost[i,j];
6. for k=1 to n do.
7. for i=1 to n do.
8. for j=1 to n do.
9. $A^k[i,j] = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$.
10. return A^k.
}.



cost[i][j]

A^0	1	2	3	4
1	0	3	∞	7
2	9	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

k=1 to n=4
i=1 to n=4.
j=1 to n=4.

(1)

→ $A^1(1,1) = \min\{A^0(1,1), A^0(1,1) + A^0(1,1)\}$
$= \min(0, 0+0)$
$A^1(1,1) = 0$.

k=1 i=1 j=2.

→ $A^1(1,2) = \min\{A^0(1,2), A^0(1,1) + A^0(1,2)\}$
$= \min\{3, 0+3\}$
$A^1(1,2) = 3$.

→ k=1 i=1 j=3.
$A^1(1,3) = \min\{A^0(1,3), A^0(1,1) + A^0(1,3)\}$
$= \min\{\infty, 0+\infty\}$
$A^1(1,3) = \infty$.

→ k=1 i=1 j=4.
$A^1(1,4) = \min\{A^0(1,4), A^0(1,1) + A^0(1,4)\}$
$= 7, 0+7$.
$A^1(1,4) = 7$.

A1	1	2	3	4
1	0	3	∞	7.

(2) k=1 i=2 j=1.

→ $A^1[2,1] = \min\{(A^0(2,1), A^0(2,1) + A^0(1,1)\}$

$= \min(9, 9 + 0)$

$= 9.$

k=1 i=2 j=2.

→ $A^1[2,2] = \min\{A^0(2,2), A^0(2,1) + A^0(1,2)\}$

$= \min\{0, 9 + 3\}$

$= 0.$

→ k=1 i=2 j=3.

$A^1[2,3] = \min\{A^0(2,3) + A^0(2,1) + A^0(1,3)\}$

$= \min\{2, 9 + \infty\}$

$= 2.$

A1	1	2	3	4
1	0	3	∞	7
2	9			
3	5			
4	2.			

→ k=1 i=2 j=4.

$A^1[2,4] = \min\{A^0(2,4), A^0(2,1) + A^0(1,4)\}$

$= \min\{\infty, 9 + 7\}$

$= 16.$

A1	1	2	3	4
1	0	3	∞	7.
2	9	0	2	16.
3				
4				

k=1 i=3 j=1.

→ $A^1[3,1] = \min\{(A^0(3,1), A^0(3,1) + A^0(1,1)\}$

$= \min\{5, 5 + 0\}$

$= 5.$

→ k=1 i=3 j=2.

$A^1[3,2] = \min\{A^0(3,2), A^0(3,1) + A^0(1,2)\}$

$= \min\{\infty, 5+3\}$

$= 8.$

→ k=1 i=3 j=3.

$A^1[3,3] = \min\{A^0(3,3), A^0(3,1) + A^0(1,3)\}$

$= \min\{0, 5+\infty\}$

$= 0.$

→ k=1 i=3 j=4.

$A^1[3,4] = \min\{A^0(3,4), A^0(3,1) + A^0(1,4)\}$

$= \min\{1, 5+7\}$

$= 1.$

A^1	1	2	3	4
1			∞	7
2	0	3	2	16.
3	5	8	0	1.

→ k=1 i=4 j=1.

$A^1[4,1] = \min\{A^0(4,1), A^0(4,1) + A^0(1,1)\}$

$= \min\{2, 2+0\}$

$= 2.$

→ k=1 i=4 j=2.

$A^1[4,2] = \min\{A^0(4,2), A^0(4,1) + A^0(1,2)\}$

$= \min\{\infty, 2+3\}$

$= 5.$

→ k=1 i=4 j=3.

$A = \min\{A^0(4,3), A^0(4,1) + A^0(1,3)\}$

$= \min(\infty, 2+\infty)$

→ k=1 i=4 j=4.

$= \min(0, 2+7)$

$= 0$

n	1	2	3	4
1	0	3	∞	7
2	9	0	2	16
3	5	8	0	1
4	2	5	∞	0

(3)

→ $k = 2 \quad i = 1 \quad j = 1$.

$A^2[1,1] = \min\{A^1(1,1), A^1(1,2) + A^1(2,1)\}$

$= \min\{0, 3 + 9\}$

$= 0$.

→ $k = 2 \quad i = 2 \quad j = 2$.

$A^2[2,1] = \min\{A^1(2,1), A^1(2,2) + A^1(2,1)\}$

$= \min\{9, 0 + 9\}$

$= 9$.

→ $k = 2 \quad i = 2 \quad j = 3$

$k = 2$

$j = 1 \rightarrow \quad i = 2 \quad i = 3 \quad i = 4$

$j = 1 \quad j = 1$ to 4 $\quad j = 1$ to 4 $\quad j = 1$ to 4

5

$k = 3$

$i = 1$

$j = 1$ to 4

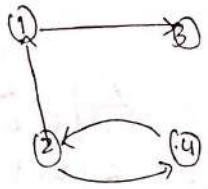
— All pair shortest path :-

Warshall's Algorithm :-

$D^k[i,j] = D^{k-1}(i,j)$ OR $D^{k-1}(i,k)$ AND $D^{k-1}(k,j)$

Algorithm :-

```
1. Cost [1:n] [1:n].
2. {
3. for i = 1 to n do.
4. for j = 1 to n do
5. A[i,j] = cost[i,j];
6. for (k=1 to n do).   -> Total no. of tables
7. for (i=1 to n do).   -> total no. of rows.
8. for (j=1 to n do).   -> Total no. of columns.
9. A^k[i,j] = A^{k-1}(i,j) OR A^{k-1}(i,k) AND
                          A^{k-1}(k,j).
10 return A^k.
}.
```



A^0	1	2	3	4
1	0	0	1	0
2	1	0	0	1
3	0	0	0	0
4	0	1	0	0

Iteration 1 :-

n = 4.

k = 1 i = 1 j = 1

$A^1[1,1] = A^0[1,1]$ OR $A^0(1,1)$ AND $A^0(1,1)$

= 0 OR 0 AND 0.

= 0.

A^1	1	2	3	4
1	0	0	1	0

k = 1 i = 1 j = 2.

$A^1[1,2] = A^0(1,2)$ OR $A^0(1,1)$ AND $A^0(1,2)$.

= 0 OR 0 AND 0 = 0.

k = 1 i = 1 j = 3.

$A^1[1,3] = A^0(1,3)$ OR $A^0(1,1)$ AND $A^0(1,3)$.

= 1 OR 0 AND 1

= 1 AND 1 OR 0

= 1.

$A^1[1,4] = A^0(1,4)$ OR $A^0(1,1)$ AND $A^0(1,4)$

= 0 OR 0 AND 0

= 0.

k=1 i=2 j=1.

A¹	1	2	3	4
1	0	0	1	0
2	0	1	0	1
3	0	0	0	0
4				

A¹[2,1] = A⁰(2,1) OR A⁰(2,1) AND A⁰(1,1)

= 1 OR 1 AND 0

= ~~1 AND 0~~ 1 OR 0 = 0 OR 1

k=1 i=2 j=2.

A¹[2,2] = A⁰(2,2) OR A⁰(2,1) AND A⁰(1,2).

= 0 OR 1 AND 0.

= 0 AND 0 = 0.

k=1 i=2 j=3.

A¹[2,3] = A⁰(2,3) OR A⁰(2,1) AND (1,3)

= 0 OR 1 AND 1

= 0 OR 1 AND 1 = 1.

k=1 i=2 j=4.

A¹[2,4] = A⁰(2,4) OR A⁰(2,1) AND A⁰(1,4)

= 1 OR 1 AND 0.

= ~~1 AND 0~~ 1 OR 0

= 0 OR 1

k=1 i=3 j=1.

A¹[3,1] = A⁰[3,1) OR A⁰(3,1) AND A⁰(1,1)

= 0 OR 0 AND 0

= 0.

K=1 i=3 j=2.

A¹[3,2] = A⁰(3,2) OR A⁰(3,1) AND A⁰(1,2)

= 0 OR 0 AND 0.

= 0.

k=1 i=3 j=3.

A¹[3,3] = A⁰(3,3) OR A⁰(3,1) AND A⁰(1,3)

= 0 OR 0 AND 1

= 0.

k=1 i=3 j=4.

A¹[3,4] = A⁰(3,4) OR A⁰(3,1) AND A⁰(1,4)

= 0 OR 0 AND 0.

= 0

$k=1 \quad i=4 \quad j=1$

$A^1[4,1] = A^0(4,1) \text{ OR } A^0(4,1) \text{ AND } A^0(1,1).$

$= 0 \text{ OR } 0 \text{ AND } 0$

$= 0.$

A^1	1	2	3	4
1	0	0	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	0	0

$k=1 \quad i=4 \quad j=2.$

$A^1[4,2] = A^0[4,2] \text{ OR } A^0(4,1) \text{ AND } A^0(1,2)$

$= 1 \text{ OR } 0 \text{ AND } 0.$

$= 1 \text{ OR } 0 = 1.$

$k=1 \quad i=4 \quad j=3.$

$A^1[4,3] = A^0(4,3) \text{ OR } A^0(4,1) \text{ AND } A^0(1,3)$

$= 0 \text{ OR } 0 \text{ AND } 1.$

$= 0.$

$k=1 \quad i=4 \quad j=4.$

$A[4,4] = A^0(4,4) \text{ OR } A^0(4,1) \text{ AND } A^0(1,4)$

$= 0 \text{ OR } 0 \text{ AND } 0$

$= 0.$

iteration (2).

A^2	1	2	3	4
1	0	0	1	0
2	1	0	1	1
3	0	0	0	0
4	1	1	1	1

$k=2 \quad i=1 \quad j=1.$

$A^2[1,1] = A^1(1,1) \text{ OR } A^1(1,2) \text{ AND } A^1(2,1)$

$= 0 \text{ OR } 0 \text{ AND } 1$

$= 0.$

$k=2 \quad i=1 \quad j=2.$

$A^2[1,2] = A^1[1,2) \text{ OR } A^1(1,2) \text{ AND } A^1(2,2)$

$= 0 \text{ OR } 0 \text{ AND } 0.$

$= 0.$

$k=2 \quad i=1 \quad j=3.$

$A^2[1,3] = A^1(1,3) \text{ OR } (1,2) \text{ AND } (2,3).$

$= 1 \text{ OR } 0 \text{ AND } 1.$

$= 1$

$k=2 \quad i=1 \quad j=4.$

$A^2[1,4] = A^1(1,4) \text{ OR } A^1(1,2) \text{ AND } A^1(2,4)$

$= 0 \text{ OR } 0 \text{ AND } 1$

$= 0.$

$k=2 \quad i=2 \quad j=1.$

$A^2[2,1] = A^1(2,1) \text{ OR } (2,2) \text{ AND } (2,1)$

$= 1 \text{ OR } 0 \text{ AND } 1$

$= 0 \cdot 1$

$k=2 \quad i=2 \quad j=2.$

$A^2(2,2) = A^1(2,2) \text{ OR } (2,2) \text{ AND } (2,2)$

$= 0 \text{ OR } 0 \text{ AND } 0 = 0.$

$k=2 \quad i=2 \quad j=3.$

$A^2(2,3) = A^1(2,3) \text{ OR } A^1(2,2) \text{ AND } (2,3)$

$= 1 \text{ OR } 0 \text{ AND } 1.$

$= 1.$

$k=2 \quad i=2 \quad j=4.$

$A^2(2,4) = A^1(2,4) \text{ OR } A^1(2,2) \text{ AND } (2,4)$

$= 0 \text{ OR } 0 \text{ AND } 1.$

$= 0$

$k=2 \quad i=3 \quad j=1.$

$A^2(3,1) = A^1(3,1) \text{ OR } A^1(3,2) \text{ AND } (2,1).$

$= 0 \text{ OR } 0 \text{ AND } 0$

$= 0.$

$k=2 \quad i=3 \quad j=2.$

$A^2(3,2) = A^1(3,2) \text{ OR } A^1(3,2) \text{ AND } (2,2).$

$= 0 \text{ OR } 0 \text{ AND } 0$

$k=2 \quad i=3 \quad j=3$

$A^2(3,3) = A^1(3,3) \text{ OR } A^1(3,2) \text{ AND } (2,3)$

$= 0 \text{ OR } 0 \text{ AND } 1$

$= 0$

$k=2 \quad i=3 \quad j=4.$

$A^2(3,4) = A^1(3,4) \text{ OR } A^1(3,2) \text{ AND } (2,4)$

$= 0 \text{ OR } 0 \text{ AND } 1.$

$= 0.$

$k=2 \quad i=4 \quad j=1.$

$A^2(4,1) = A^1(4,1) \text{ OR } A^1(4,2) \text{ AND } A^1(2,1)$

$= 0 \text{ OR } 1 \text{ AND } 1$

$= 1.$

$k=2 \quad i=4 \quad j=2.$

$A^2(4,2) = A^1(4,2) \text{ OR } A^1(4,2) \text{ AND } A^1(2,2)$

$= 1 \text{ OR } 1 \text{ AND } 0.$

$= 1.$

$k=2 \quad i=4 \quad j=3.$

$A^2(4,3) = A^1(4,3) \text{ OR } A^1(4,2) \text{ AND } A^1(2,3)$

$= 0 \text{ OR } 1 \text{ AND } 1$

$= 1.$

$k=2 \quad i=4 \quad j=4.$

$A^2(4,4) = A^1(4,4) \text{ OR } A^1(4,2) \text{ AND } A^1(2,4)$

$= 0 \text{ OR } 1 \text{ AND } 1$

$= 1.$

A^2	1	2	3	4
1	0	0	1	0
2	1	0	1	1
3	0	0	0	0
4	1	1	1	1

A^3	1	2	3	4
1	0	0	1	0
2	1	0	1	1
3	0	0	0	0
4	1	1	1	1

$k=3 \quad i=1 \quad j=1.$

$A^3[1,1] = A^2(1,1)$ OR $A^2(1,3)$ AND $(3,1)$

$= 0$ OR 1 AND 0.

$= 0.$

$k=3 \quad i=1 \quad j=2.$

$A^3(1,2) = A^2(1,2)$ OR $A^2(1,3)$ AND $(3,2)$

$= 0$ OR 1 AND 0

$= 0.$

$k=3 \quad i=1 \quad j=3.$

$A^3(1,3) = A^2(1,3)$ OR $A^2(1,3)$ AND $(3,3)$.

$= 1$ OR 1 AND 0

$= 1.$

$k=3 \quad i=1 \quad j=4.$

$A^3(1,4) = A^2(1,4)$ OR $A^2(1,3)$ AND $(3,4)$

$= 0$ OR 1 AND 0

$= 0.$

$k=3 \quad i=2 \quad j=1.$

$A^3(2,1) = A^2(2,1)$ OR $A^2(2,3)$ AND $(3,1)$

$= 1$ OR 1 AND 0

$= 1$

$k=3 \quad i=2 \quad j=2.$

$A^3(2,2) = A^2(2,2)$ OR $A^2(2,3)$ AND $A^2(3,2)$

$= 0$ OR 1 AND 0

$= 0.$

$k=3 \quad i=2 \quad j=3.$

$A^3(2,3) = A^2(2,3)$ OR $A^2(2,3)$ AND $A^2(3,3)$.

$= 1$ OR 1 AND 0

$= 1.$

$k=3 \quad i=2 \quad j=4.$

$A^3(2,4) = A^2(2,4)$ OR $A^2(2,3)$ AND $A^2(3,4)$.

$= 1$ OR 1 AND 0

$= 1.$

$k=3 \quad i=3 \quad j=1.$

$A^3(3,1) = A^2(3,1)$ OR $A^2(3,3)$ AND $A^2(3,1)$.

$= 0.$

$k=3 \quad i=3 \quad j=2.$

$A^3(3,2) = A^2(3,2)$ OR $A^2(3,3)$ AND $A^2(3,2)$

$= 0.$

$k=3 \quad i=3 \quad j=3.$

$A^3(3,3) = A^2(3,3)$ OR $A^2(3,3)$ AND $A^2(3,3)$

$= 0$

$k=3 \quad i=3 \quad j=4.$

$A^3(3,4) = A^2(3,4)$ OR $A^2(3,3)$ AND $A^2(3,4)$

$= 0$

k=3 i=4 j=1.

$A^3(4,1) = A^2(4,1)$ OR $A^2(4,3)$ AND $A^2(3,1)$

$= 1$

k=3 i=4 j=2.

$A^3(4,2) = A^2(4,2)$ OR $A^2(4,3)$ AND $A^2(3,2)$.

$= 1$

k=3 i=4 j=3.

$A^3(4,3) = A^2(4,3)$ OR $A^2(4,3)$ AND $A^2(3,3)$

$= 1$

k=3 i=4 j=4

$A3(4,4) = A^2(4,4)$ OR $A^2(4,3)$ AND $A^2(3,4)$

$= 1.$

A^3	1	2	3	4
1	0	0	1	0
2	1	0	1	1
3	0	0	0	0
4	1	1	1	1

A^4	1	2	3	4
1				
2				
3				
4				

k=4 i=1 j=1.

$A^4(1,1) = A^3(1,1)$ OR $(1,4)$ AND $(4,1)$.

k=4 i=1 j=2.

$A^4(1,2) = A^3(1,2)$ OR $A^3(1,4)$ AND $(4,2)$

k=4 i=1 j=3.

$A^4(1,3) = A^3(1,3)$.

- Optimal Binary search tree.

(Absent)

→ keys. 0 | 10 | 1 | 20 | 2 | 30 | 3 | 40 | 4

frequency. 4 | 2 | 6 | 3.

step 1 :-

i \ j	0	1	2	3	4
0	4				
1		2			
2			6		
3				3	
4					

	0	1	2	3	4
0					
1					
2					
3					
4					

0- BST [n+1][n+1]

step 2 :- $j = j - i = 0$.

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

root table [n+1][n+1]

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

Step 3 :- $l = j - i = 1$.

cost[0,1] = 4.

cost [1,2] = 1.

cost [2,3] = 1

cost [3,4] = 1.

	0	1	2	3	4
0	0	4	8		
1		0	2		
2			0	6	
3				0	3
4					0

0-BST

⇒ cost [0,1].

⇒ 10 → 1*4 = 4.

⇒ cost [1,2].

⇒ 20 → 1*2 = 2.

⇒ cost [2,3].

⇒ 30 → 1*6 = 6.

⇒ cost [3,4].

⇒ 40 → 1*3 = 3.

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

Root table.

$\frac{2\times2\ C_2}{3} = \frac{4C_2}{3}$

$= \frac{12}{12} = \frac{6}{3} = 2$

Step 4 :- $l = j - i = 2$

cost [0,2] = 2.

cost [2,4] = 2

cost [1,3] = 2

	0	1	2	3	4
0	0	4	8		
1		0	2	10	
2			0	6	12
3				0	3
4					0

cost[0,2] :- 10, 20

1*4 + 1*2 + 1*2 = 10

= 8 ₹ (min)

2-BST

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

Root table

cost [0,2] = 8.

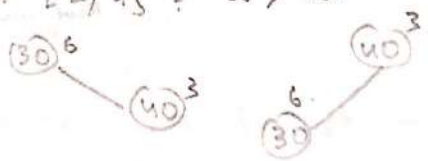
cost [1,3] :- 20, 30

= 1*2 + 1*6 + 1*6 = 1*6 + 1*2 + 1*2

= 14 = 10. (min)

cost [1,3] = 10.

cost [2,4] :- 30, 40.



= 1*6 + 1*3 + 1*3 = 1*3 + 1*6 + 1*6

= 12. (Min) = 15

cost [2,4] = 12

Step 5 :- $d = j - i = 3$.

cost [1, 4] = 3.

cost [0, 3] = 3.

cost [0, 3] = 10, 20, 30, ~~40~~.

(In Ronald's book).

	0	1	2	3	4
0	0	4	8		
1		0	2	10	
2			0	6	12
3				0	3
4					0

0-BST.

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

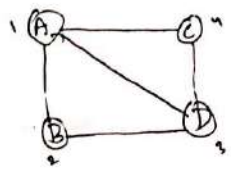
Root table.

Dynamic Programming algorithm :-

1. Optimal BST(P[1 ... n]).
2. for i ← 1 to n do.
3. C[i, j-i] ← 0.
4. C[i, i] ← P[i].
5. R[i, i] ← i.
6. C[n+1, n] ← 0.
7. for d ← 1 to n-1 do // diagonal count.
8. for i ← 1 to n-d do.
9. j ← i+d.
10. minval ← ∞
11. for k ← i to j do.
12. if C[i, k-1] + C[k+1, j] < minval.
13. minval ← C[i, k-1] + C[k+1, j];
    kmin ← k.
14. R[i, j] ← kmin.
15. Sum ← P[i]; for s ← i+1 to j do.
    Sum ← Sum + P[i].
16. C[i, j] ← minval + Sum
17. return C[1, n], R.

(In Ronald's book)

Travelling Sales person problem.



	A	B	C	D	
A	∞	4	12	7	→ 4
B	5	∞	∞	12	→ 5
C	11	∞	∞	6	→ 6
D	10	2	3	∞	→ 2

17

I

row reduction →

∞	0	8	3
0	∞	∞	13
5	∞	∞	0
8	0	1	∞

→ column reduction → 1

17+1 = 18

II

∞	0	7	3
0	∞	∞	13
5	∞	∞	0
8	0	0	∞

row reduction →

III

∞	∞	∞	∞	∞
0	∞	∞	13	→ 13
5	∞	∞	0	
8	∞	0	∞	

(Because we are considering 1-2)

IV

after row reduction →

∞	∞	∞	∞
∞	∞	∞	0
5	∞	∞	0
8	∞	0	∞

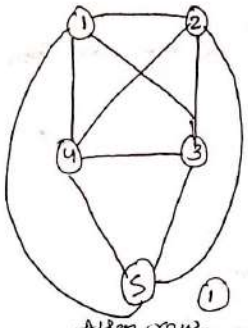
→ 5 column reduction →

V

∞	∞	∞	∞
∞	∞	∞	0
0	∞	∞	0
3	∞	0	∞

= 18 + 18 = 36.

Initial.



	1	2	3	4	5	
1	∞	20	30	10	11	→ 10
2	15	∞	16	4	2	→ 2
3	3	5	∞	2	4	→ 2
4	19	6	18	∞	3	→ 3
5	16	4	7	16	∞	→ 4

(2) 21

(1)

After row reduction

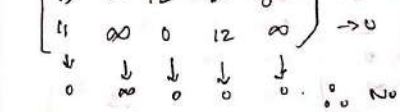
∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞
↓	↓	↓	↓	↓
1	0	3	0	0

4.

After column reduction

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

21 + 4 = 25.



because we don't want to go back to 1 from 2

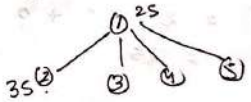
For 1 to 2.

∴ No row reduction.

∞	∞	∞	∞	∞	→ ∞
∞	∞	11	2	0	→ 0
0	∞	∞	0	2	→ 0
15	∞	12	∞	0	→ 0
11	∞	0	12	∞	→ 0
↓	↓	↓	↓	↓	
0	∞	0	0	0	

∴ No column reduction.

cost (2) = 25 + 0 + 10 = 35.



For 1 to 3.

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	→ ∞
2	12	∞	∞	2	0	→ 0
3	∞	3	∞	0	2	→ 0
4	15	3	∞	∞	0	→ 0
5	11	0	∞	12	∞	→ 0
	↓	↓	↓	↓	↓	
	11	0	∞	0	0	

No row reduction.

After column reduction.

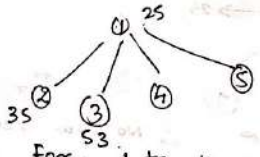
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	0
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

No column & row reduction.

25
11
36

cost(3) = 25 + 11 + 17.

= 53.

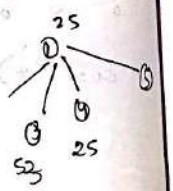


For 1 to 4.

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	→ ∞
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	15	3	12	∞	0	0
5	11	0	0	∞	∞	0
	0	0	0	∞	0	

cost(4) = 25 + 0 + 0

= 25.



For 1 to 5.

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	→ ∞
2	12	∞	11	2	∞	→ 2
3	0	3	∞	0	∞	→ 0
4	15	3	12	∞	∞	→ 3
5	11	0	0	12	∞	→ 0

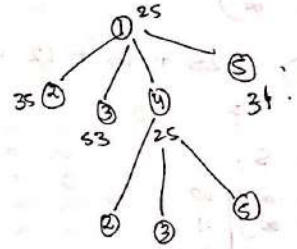
5

After row reduction.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	11	0	0	12	∞
	0	0	0	0	∞

cost(5) = 25 + 05 + 1

= 31



For 4 to 2.

∞	∞	∞	∞	∞	0
12	∞	11	∞	0	0
0	3	∞	∞	2	∞
15	3	12	∞	0	0
11	0	0	∞	∞	

∞	∞	∞	∞	∞
∞	∞	11	∞	0
0	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞
0	∞	0	∞	0

cost(2) = cost(4) + 0 + w(4,2)

= 25 + 0 + 3 = 28

For 4 to 3.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ 0 \\ 2 \\ \infty \\ 0 \end{matrix} \quad 2$$

11

After row reduct^n

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

11 0 0 ∞ 0 / 11

After column reduct^n.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 11 & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty \end{bmatrix} = 13.$$

cost(3) = 25 + 13 + 12.

= 50

For 4 to 5.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ 11 \\ 0 \\ \infty \\ 0 \end{matrix} \quad 11$$

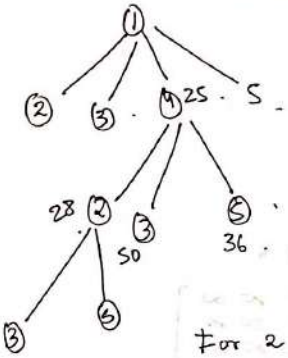
After row reduct^n.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

0 0 0 ∞ ∞.

No column reduction.

Cost(5) = 25 + 11 + 0 = 36



For 2 to 3.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ \infty \\ 2 \\ \infty \\ 11 \end{matrix} \quad 11/13$$

After row reduct^n.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

0 ∞ ∞ ∞ 0 / 2.

No column reduct^n

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

= 13.

cost(3) = 28 + 13 + 11

= 52.

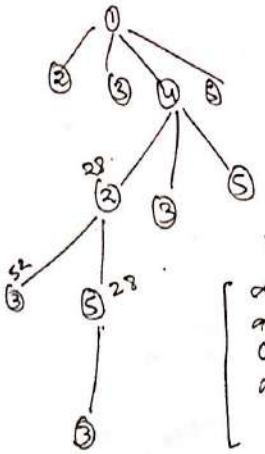
For 2 to 5.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ \infty \\ 0 \\ \infty \\ 0 \end{matrix}$$

No row reduct^n.

cost(5) = 28 + 0 + 0

= 28



For 2 to 5.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

For 5 to 3.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{matrix}$$

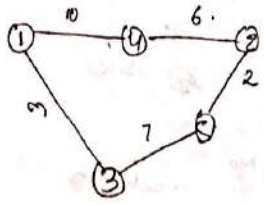
No row reduction.

$$\begin{matrix} 0 & \infty & \infty & \infty & \infty \end{matrix}$$

No column reduction.

cost (5) = 28 + 0 + 0

= 28



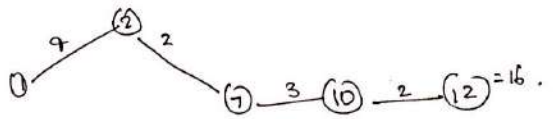
1, 4, 2, 5, 3.

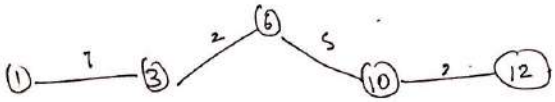
= 28

= 25

(In Kowalski Logic)

v	1	2	3	4	5	6	7	8	9	10	11	12.
cost	16	7		9	18	15	7	5	7	4	2	5	0
d.	2/3.	7		6	8	8	10	10	10	12	12	12	12.

(1) 

(2) 

Date 15/9/19

Module-2

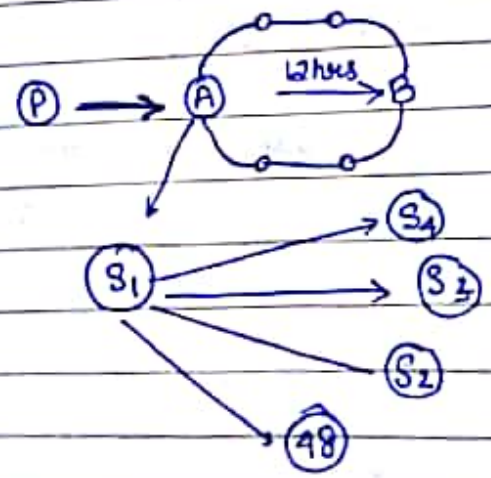
Greedy Method

- optimization problem
- feasible solution
- optimization solution

→ Soln that satisfies the condition

* General Algorithm for Greedy Method:

```
Algorithm greedy (a, n)
{
a[1 ..... n] contains n inputs
solution = 0        // Initialize solution //
for(i=1 to n) do
{
  x = select(a);
  if feasible (solution, x)
  then
  solution = union(solution, x)
}
return solution;
}
```



1) Knap Sack problem.
2) Minimum spanning Tree.
   (a) Kruskals Algorithms.
   (b) Prim's Algorithms.
3) Single - Source Shortest path.

38 - 3
3 | 115
9
25
24
10

Knap-Sack problem

container or bag

total weight
= ~~If w~~ Tw + w[i]

Total profit
= Total profit + p[i]

→ object ()
{ profit ()
{ weight ()

when the capacity is less = Total profit + rem. weight / * prof weight

4) profit() / weight() ⇒ "Depending on the heighest weight of the object put the objects into the container"

→

2	0.5	0.2	0.2	0.105	0.1

→ Selection Sort
(average in decreasing Order)

if (average[i] < average[j])

Profit of the array = PF + 10

(Fraction of weight) — Final Answer.

→ Algorithm fractional knapsack

Step 1 : Calculate the ratio (profit/weight) for each objects

Step 2 : Arrange the decreasing order of (profit/weight) ratio of all items.

Step 3 : Take the items with highest ratio & add them to the knapsack.

Step 4 : Repeat the step 3 until we cannot add next item as a whole.

Step 5 : At the end add next time as much as we can (∵ fration)

Step 6 : Find the profit.

p-I

1)

5	1.6	3	1	6	4.5	3

p-II

5	1.6	3	1	6	4.5	3

final pass :

6	5	4.5	3	3	1.6	1

Ratio	6	5	4.5	3	3	1.6	1
profit	6	10	18	15	3	5	7
weight	1	2	4	5	1	3	7

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

(knapsack)

12+1 = 13. 15 − 13 = 2

Date 24/9/19

* Minimum Spanning Tree → No cycle
  - ↳ Prim's algorithm
  - ↳ Kruskal's algorithm

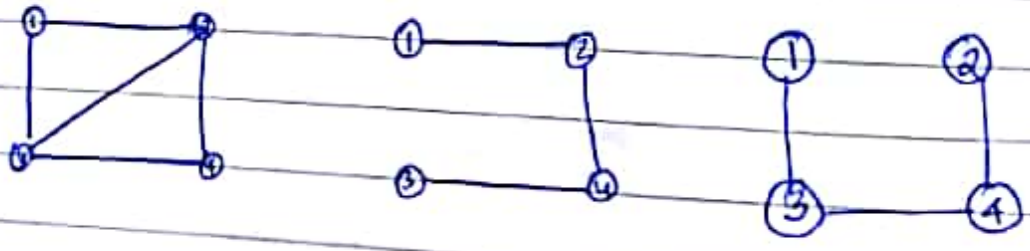
→ Spanning Tree: It must not contain any cycles.

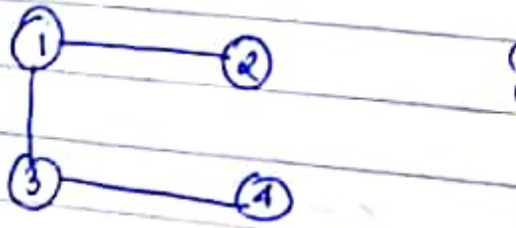
→ Graphs: Graphs, that contains no. of edges & vertices & all the vertices must be connected.

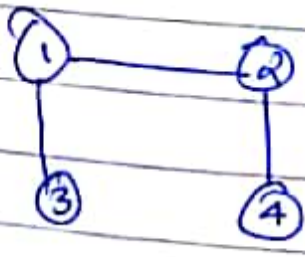
→ Spanning tree:
It is a tree that connects all the vertices of a graph without any cycles or loops.

→ ~~Minimum~~ cost SpanningTree:

A spanning Tree weight is minimum of all possibilities is called Minimum cost spanning Tree.



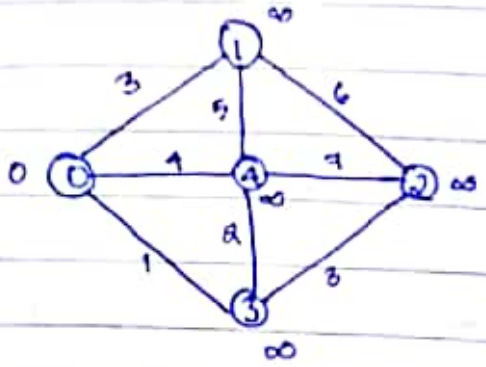




2 greedy algorithms :

→ Prims

→ Kruskals



∞ = 999

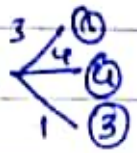
	0	1	2	3	4
0	0	3	∞	1	4
1	3	∞	6	∞	5
2	∞	6	∞	3	7
3	1	∞	3	∞	2
4	4	5	7	2	∞

① MST Set = {0, 3, 4, 1, 2}

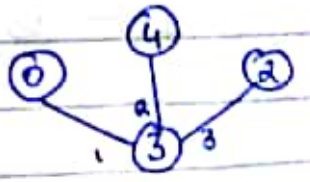
② → Source Vertex Cost → (0)

→ ∞

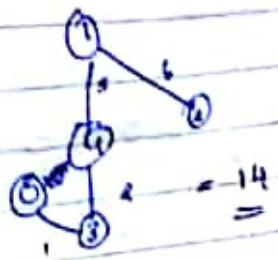
③ → i source



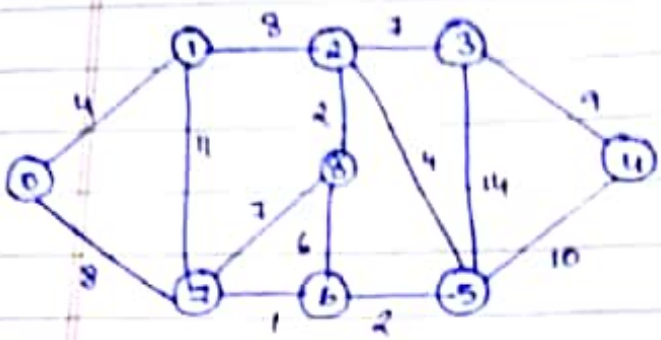
(ii)



→



= 14

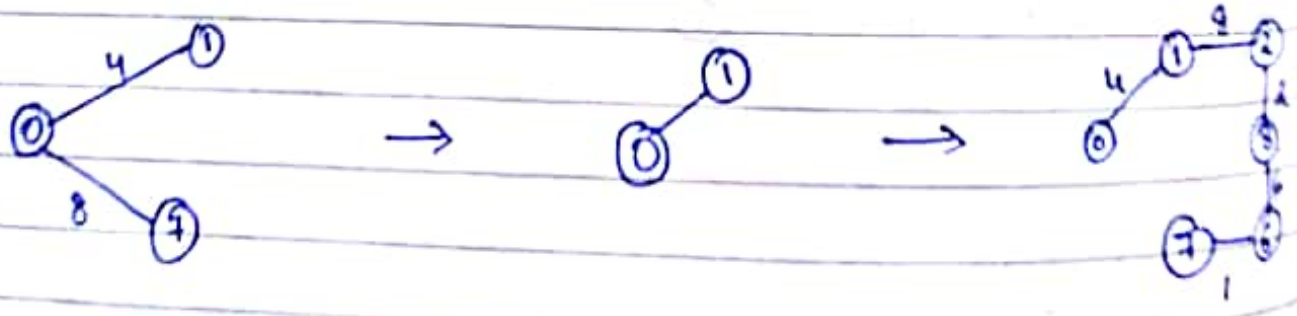


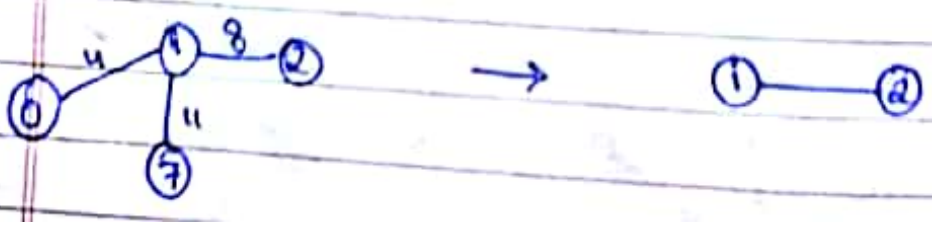
∞ = 999

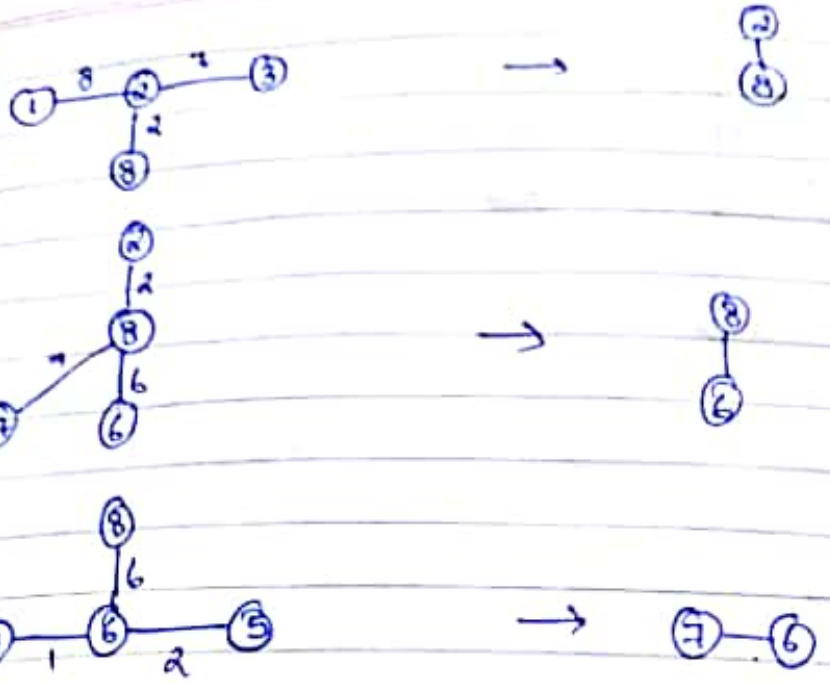
	0	1	2	3	4	5	6	7	8
0	0	4	∞	∞	∞	∞	∞	8	∞
1	4	∞	8	∞	∞	∞	∞	11	∞
2	∞	8	∞	7	∞	4	∞	∞	2
3	∞	∞	7	∞	9	14	∞	∞	∞
4	∞	∞	∞	9	∞	10	∞	∞	∞
5	∞	∞	4	14	10	∞	2	∞	∞
6	∞	∞	∞	∞	∞	2	∞	1	6
7	8	11	∞	∞	∞	∞	1	∞	7
8	∞	∞	2	∞	∞	∞	6	7	∞

Source = 0

MST set = {0, 1, 2,

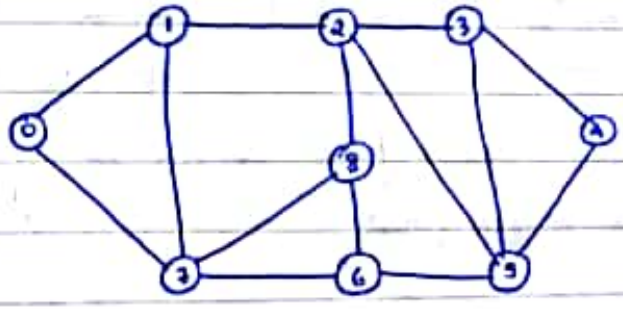


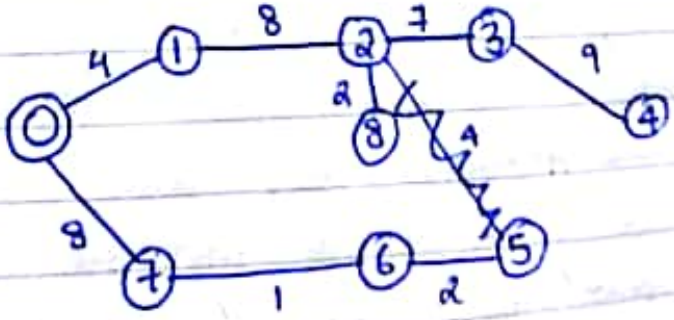




→ arrange it in increasing order.

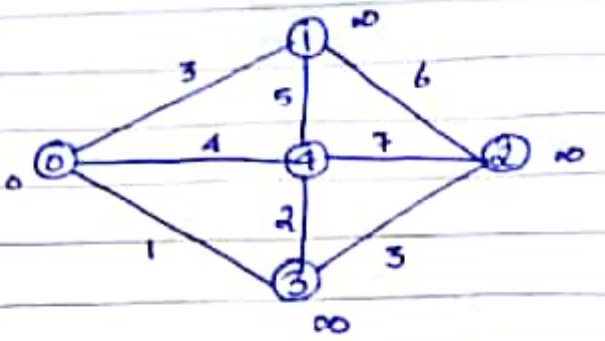
(7) - (6) = 1
(6) - (5) = 2
(8) - (2) = 2
(0) - (1) = 4
(2) - (5) = 4
~~(3) - (6) = 6~~
~~(8) - (1) = 7~~
(2) - (3) = 7
(0) - (4) = 8
(1) - (2) = 8
(3) - (4) = 9
~~(5) - (4) = 10~~
~~(1) - (4) = 11~~
~~(3) - (5) = 14~~





Total cost = 40
=

25/9/2019



Prims Algorithm

Steps :

Step1 : Create a set mstSet that keeps track of vertices already visited included in MSTset

Step2 : Assign a key value to all vertices in the input graph.

* Initialize all key values as INFINITE ——→ all mstSet as False

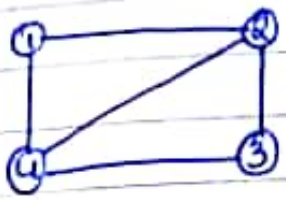
Assign key value as 0 for the first vertex 0, so that it is picked first.

Step3 : While mstSet doesn't include all vertices.

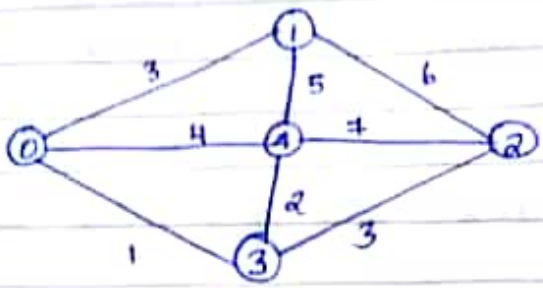
→ pick a vertex u which is not there in mstSet and has minimum key value.
→ Include u to MST set
→ Update key value of all the adjacent vertices of u.

→ To update the key values, iterate through all the adjacent vertices for every adjacent vertex v

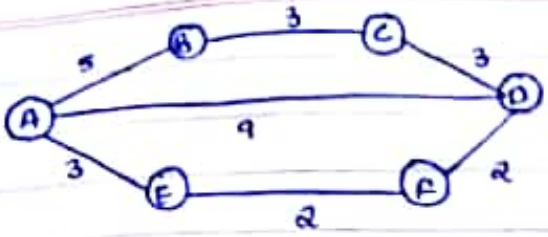
→ If weight of edge u→v is less than the previous key values of v update key value of v as weight of u-v.



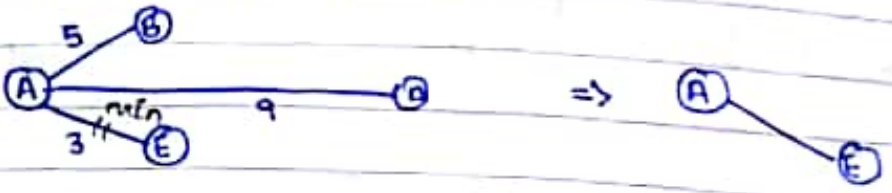
(11)



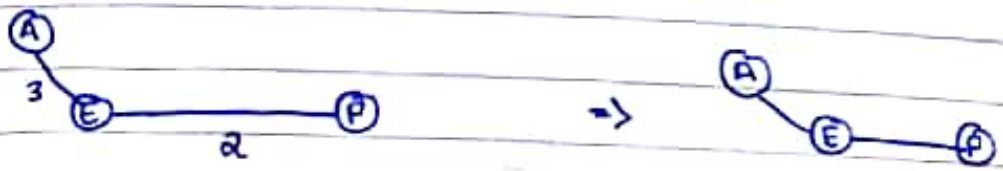
0 1 2 3 4



Step (1) : Start source A

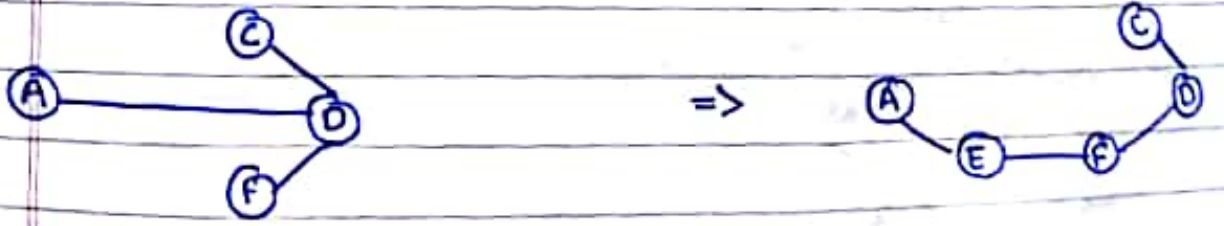


Step 2 :

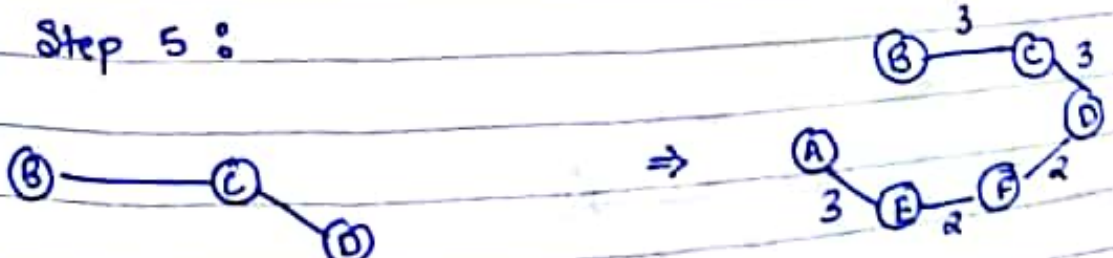


Step 3 :

Step 4 :



Step 5 :



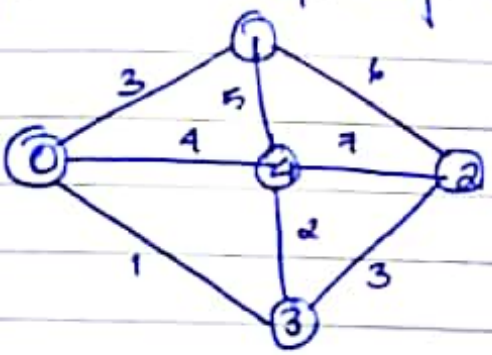
KRUSKAL'S ALGORITHM

Step 1 :- Sort all the edges in increasing order of their weight.

Step 2 :- pick the smallest edge, check if forms a cycle with the spanning tree formed so far. If cycle is not formed includes the edge else discard it.

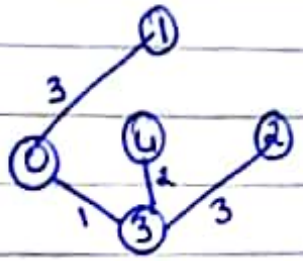
Step 3 :- Repeat Step 2 until there are (V-1) Edges in the spanning tree.

1)

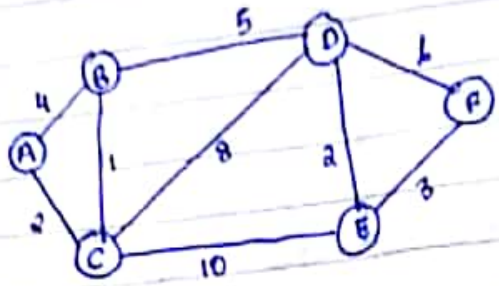


Sno.	Edge	Cost
1)	0-3	1
2)	3-4	2
3)	0-1	3
4)	3-2	3
5)	0-4	4
6)	1-4	5
7)	1-2	6
8)	2-4	7

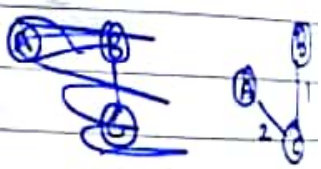
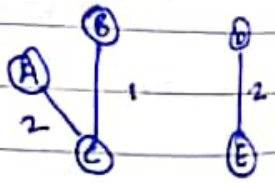
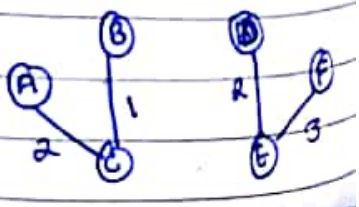
⇒



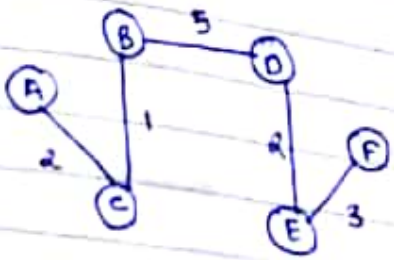
2)



S.No.	Edge	Cost
1)	B-C	1
2)	A-C	2
3)	D-E	2
4)	E-F	3
5)	A-B	4
6)	B-D	5
7)	D-F	6
8)	C-D	8
9)	C-E	10

	EDGE	COST	SPANNING TREE
1)	B→C	1	
2)	A→C	2	
3)	D→E	2	
4)	E→F	3	

(5)	A → B	4	It will form a cycle.
(6)	B → D	5	
(7)	D → F	6	will form a cycle.
(8)	C → D	8	
(9)	C → E	10	

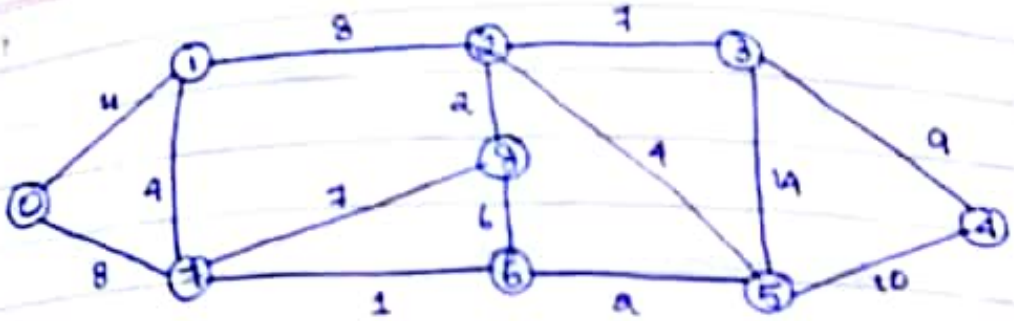


Prims pseudo code :

```
void primmst (graph)
{
  int parent[V];
  int key[V];
   bool mstset[V];
   for(int i=0; i<V; i++)
   {
    key[i] = INT_MAX;
    mstset[i] = false;
   }
    key[0] = 0;
    parent[0] = -1;
    for(int count = 0; count < V-1 ; count++)
     {
       int u = min key(key, mstset);
       mstset[u] = true;
```

```
for (int v = 0; v<V; v++)
{
if(graph[u][v] && mstset[v] == false && graph[u][v]<key[v])
parent[v] = u, key[v] = graph[u][v];
}}
printf printmst( parent, V, graph);
}
int minkey [int key[], bool mstset[])
{
int min = INT_MAX, min_index;
for (int v=0, v<V; v++)
{
if (mstSet[v] == false && key[v]< min)
{
min = key[v];
min_index = v;
}
}
return min_index;
}
int print mst (int parent[], int V, int graph[V][V])
{
printf(" Edge Weight ");
for (int i=1; i<V; i++)
printf("%d %d %d\n", Parent[i], i, graph[i],
(parent[i]));
}
```

(18)

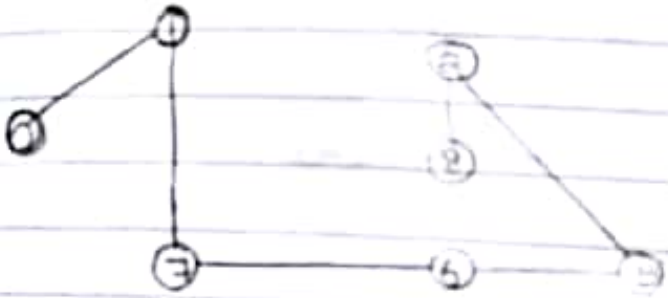


MST set [v] = { | | | | | | | | | }

0 1 2 3 4 5 6 7 8

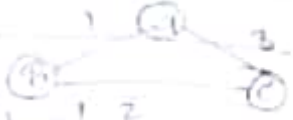
Key values [v] =

0 1 2 3 4 5 6 7 8



27/9/19

KRUSKALS PSEUDO CODE



1) A = ∅
2) for each vertex, ~~v ∈ G.V~~ v ∈ G.V (smallest element in G.V graph)
3) Make-SET(v) (make a set)
4) Sort the edges of GE into increasing order by weight. (How many edges we have)
5) for each edge (u,v) ∈ GE taken in increasing order by weight
6) if FIND-SET(u) ≠ FIND-SET(v)
7) A = A ∪ {(u,v)}
8) UNION(u,v)
9) return A (returning resulting graph)

SHORTEST PATH ALGORITHM

Types of Path problem

- Single source shortest Path [SSSP]
  - Bellmanford
  - Dijkstra's
- All-pair shortest path [APSP]
  - FLOYD-WARSHALLS
  - JOHNSON'S

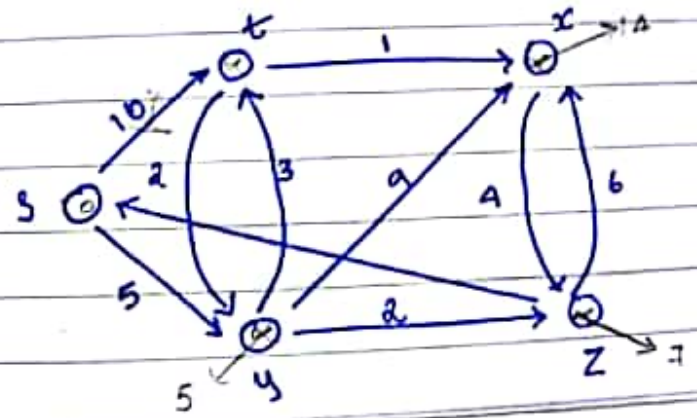
DIJKSTRA'S ALGORITHM

1) Assign every node ~~at~~ a tentative distance.
2) Set initial node as current & mark all other nodes as unvisited.
3) For current node consider all unvisited nodes and calculate tentative distance. Compare current distance with calculated distance and assign the smaller value.

5) when all the neighbours are considered of the current node, mark it visited { visited node is never marked again }
6) If the destination node is marked visited, STOP
7) END.

DIJKSTRA (G, w, s)

1) INITIALIZE - SINGLE - SOURCE (G, s)
2) S = ϕ
3) Q = G.V (Set of vertex of the graph)
4) while ϕ = Q
5) u = EXTRACT - MIN(Q)
6) S = S ∪ {u}
7) for each vertex v ∈ G. Adj[u]
8) RELAX [u, v, w)



Iteration - 1

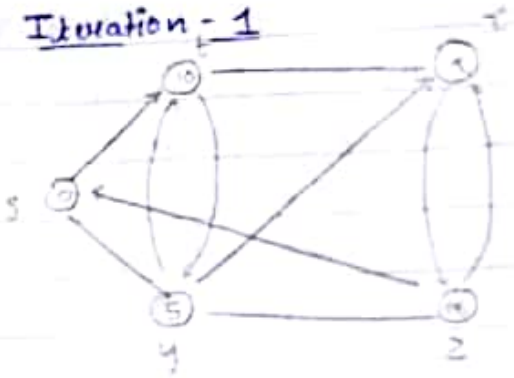
① s.d = 0, t.d = ∞, x.d = ∞
y.d = ∞ z.d = ∞
② s.d = 0
③ Q = {s, t, y, x, z} 0, ∞, ∞, ∞, ∞
④ {
⑤ Extract value
01/10/2019
⑥ S = {s}
⑦ Adj[s] = {t, y}
⑧ relax (u, v, w)
(s, t, 10) = t.d = 10 < ∞
(s, y, 5) = y.d = 5 < ∞

Q = {t, y, x, z} 10, 5, ∞, ∞

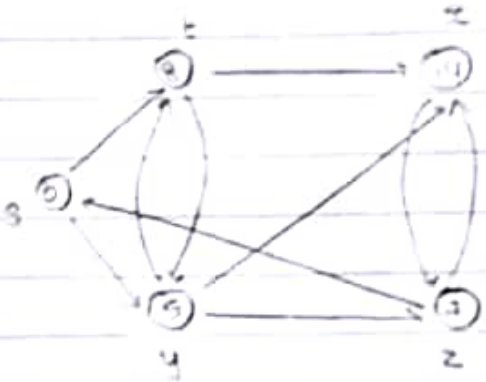
Iteration - 2

u = y → minimum vertex
S = S ∪ {u}
S = {s, y}
Adj[y] = {t, x, z}
relax (y, t, w)
t.d = y.d + w(y, t) = 5 + 3 = 8 < ∞
x.d = y.d + w(y, x) = 5 + 9 = 14 < ∞
z.d = y.d + w(y, z) = 5 + 2 = 7 < ∞
Q = {t, x, z} 8, 14, 7

Iteration - 1



Iteration - 2



Iteration - 3

$Q = \{t, x, z\}$

$u = z \rightarrow$ min

$S = S \cup \{u\}$

$S = \{s, y, z\}$

$adj[z] = \{s, x\}$

$relax(z, s, w) = s.d = z.d + w(z, s)$

$= 7 + 7 \Rightarrow 14 > 0$ ✗

$relax(z, x, w) = x.d = z.d + w(z, x)$

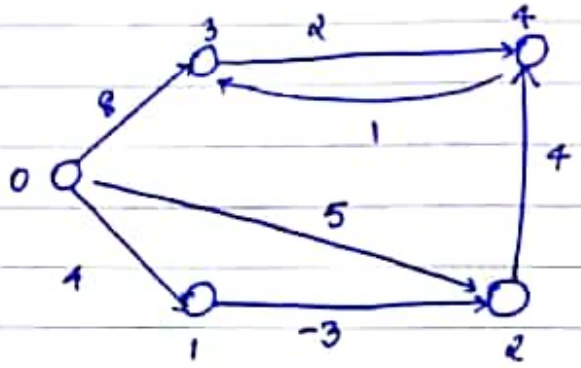
$= 7 + 6 = 13 < 14$ ✓

$Q = \{t, x\}$
$8, 13$

4/10/19

* BELLMAN – FORD (G, W, S)

1. Initialize — Single Source (G, S)
2. for (i = 1 to |G.V| − 1
3. for each edge (u, v) ∈ G.E
4. Relax (u, v, w)
5. for each edge (u, v) ∈ GE
6. if v.d > u.d + w(u, v)
   Then d[v] = d[u] + w[u, v]



Iteration 1

Step 1 : 0.d = 0
1.d = ∞ 4
2.d = ∞ ~~∞~~ 1
3.d = ∞ ~~∞~~ 6
4.d = ∞ 5

Step 2 : i = 1 to 4

Step 3 : {(0,1), (0,2), (0,3), (1,2), (2,4), (3,4), (4,3)}

Step 5 : (0,1)
u v

step 1 : $1.d > 0.d + w(0,1)$

$d > 0+4$

$d > 4$

$1.d = 0+4 \quad 1.d = 4$

(0,2)

$2.d > 0.d + w(0,2)$

$d > 0+5$

$d > 5$

$2.d = 0+5 \quad 2d = 5$

(0,3)

$3.d > 0.d + w(0,3)$

$d > 0+8$

$d > 8$

$3.d = 0+8 \quad 3.d = 8.$

(1,2)

$2.d > 1.d + w(1,2).$

$5 > 4 + (-3).$

$5 > 1$

$2.d = 1$

(2,4)

$4.d > 2.d + w(2,4)$

$d > 1+4$

$d > 5$

$4.d = 5$

(3,4)

$\Rightarrow$ $4.d > 3.d + w(3,4)$

$5 > 8 + 2$

$5 > 10$

(no need to update)

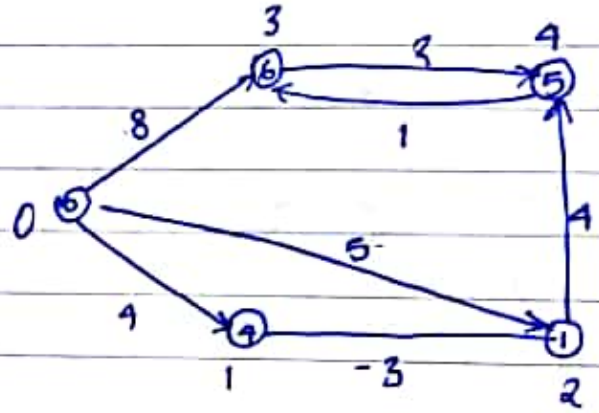
(4,3)

$\Rightarrow$ $3.d > 4.d + w(4,3)$

$8 > 5 + 1$

$8 > 6$

$3.d = 0 + 6 = 6.$



(Output for Iteration-1)

Iteration-2

Step 1 : $0.d = 0$

$1.d = 4$

$2.d = 1$

$3.d = 6$

$4.d = 5$

Step 2: {(0,1),(0,2),(0,3),(1,2),(2,4),(3,4),(4,3)}

(0,1)

=) 1d > 0.d + w(0,1)
4 > 0 + 4
4 > 4 X

(0,2)

=) 2d > 0d + w(0,2)
1 > 0 + 5
1 > 5 X

(0,3)

=> 3d > 0d + w(0,3)
6 > 0 + 8
6 > 8 X

(1,2)

=) 2d > 1d + w(1,2)
1 > 4 + -3
1 > 1 X

(2,4)

=> 4d > 2d + w(2,4)
5 > 1 + 4
5 > 5 X

(3,4)

=) 4d > 3d + w(3,4)
=) 5 > 6 + 2
= 5 > 8 X

(4,3)

=> 3d > 4d + w(4,3)
=> 6 > 5 + 1
6 > 6 X

Second iteration - no change. 4th Iteration - —"—

Third iteration - no —"—

(Final answer)

Ans:

0-0 → 0

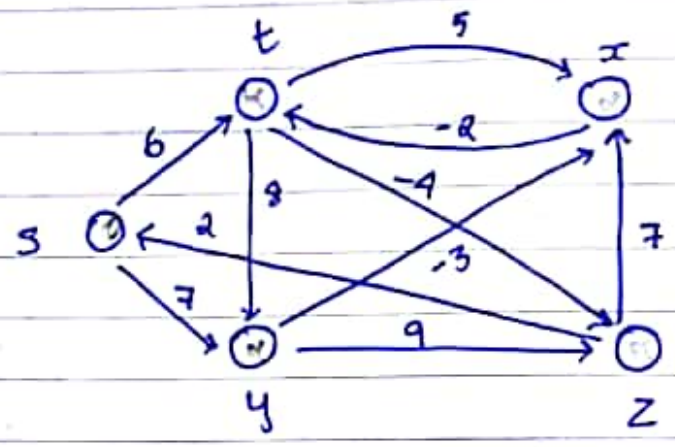
0-1 → 4

0-1-2 → 1

0-1-2-4-3 → 6

0-1-2-4 → 5

(2)



Iteration - 1

Step. 1 = ~~0-d~~ s.d = ~~∞~~ 0

~~1-d~~ t.d = ~~∞~~ 6

~~2-d~~ x.d = ∞

~~4-d~~ y.d = ~~∞~~ 7

z.d = ∞

Step. 2 = for(i=1 to 4

Step 3 = {(s,t) (s,y) (t,y) (t,x) (x,t) (y,z) (z,x), (y,x), (t,z), (z,s)}

steps :

(s, t) =

=> t.d > s.d + w(s,t)

∞ > 0 + 6

∞ > 6

~~s.d~~ t.d = 6

(s, y) =

=> y.d > s.d + w(s,y)

∞ > 0 + 7

∞ > 7

y.d = 7

(t, y) =

=> y.d > t.d + w(t,y)

7 > 6 + 8

7 > 14 x

~~y.d~~

(t, x)

=> x.d > t.d + w(t,x)

∞ > 6 + 5

∞ > 11

x.d = 11

(x, t)

MODULE - 3

"DYNAMIC PROGRAMMING TECHNIQUE"

→ Optimisation problem.
→ Divide & Conquer method
→ Overlapping Subproblem

=> Try to avoid overlapping of Subproblem.

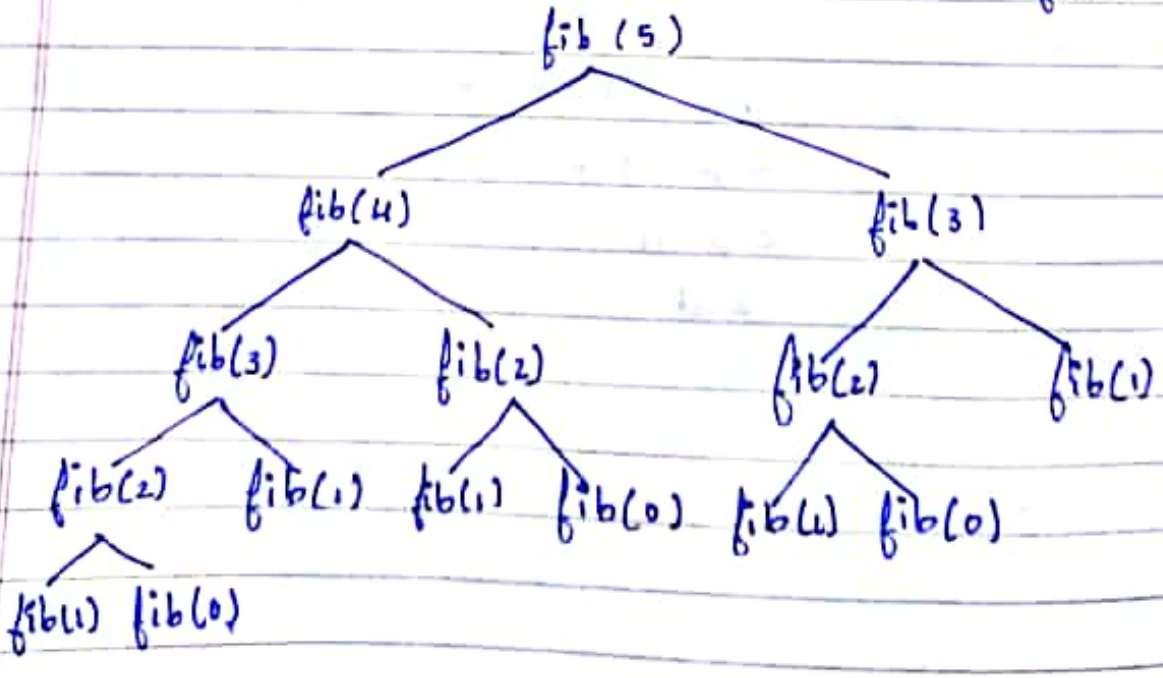
→ Memorization (Top-down) } Two
→ Tabulation (Bottom-up) } methods

Example:

```
int fib(int n) {
 if(n<=1) {
 return n;
        }
else
return fib(n-1) + fib(n-2)
        }
```

n=5

fib(0) = 0
fib(1) = 1



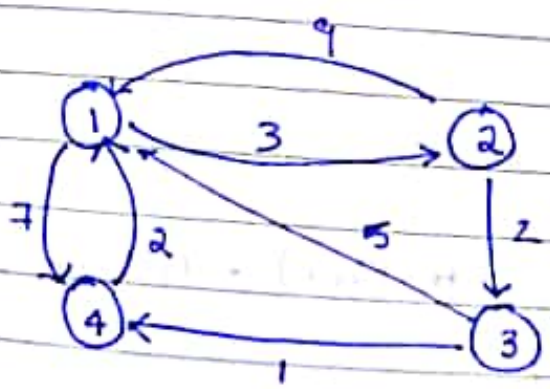
All - pair shortest path

1) Floyd's Algorithm
2) Warshall's Algorithm

Algorithm Floyd's (cost, A, n)

```
1. cost [i : n] [1 : n]
2. {
3. for i = 1 to n do
4. for j = 1 to n do
5. A[i,j] = cost[i,j];
6. for k = 1 to n do
7. for i = 1 to n do
8. for j = 1 to n do
9. A^k [i,j] = min (A^(k-1) (i,j), A^(k-1) [i,k) + A^(k-1) (k,j)]
10. Return A^k.
11. }
```

k : 1 to n



A^0	1	2	3	4
1	0	3	∞	2
2	9	0	2	∞
⇒ 3	5	∞	0	1
4	2	∞	∞	0

Date 11/10/19

A^1	1	2	3	4
1	0	3	∞	7
2	9	0	2	16
3	5	8	0	1
4	2			

(1) $k=1$ to $n=4$
$i=1$ to $n=4$
$j=1$ to $n=4$

(2)

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

$A^1(1,1) = 0$

$k=1 \quad i=1 \quad j=2$

$A^1(1,2) = \min\{A^0(1,2), A^0(1,1)+A^0(1,2)\}$
$= \min\{3, 0+3\}$
$A^1(1,2) = 3$

$k=1 \quad i=1 \quad j=3$

$A^1(1,3) = \min\{A^0(1,3), A^0(1,1)+A^0(1,3)\}$
$= \min\{\infty, 0+\infty\}$

$A^1(1,3) = \infty$

$k=1 \quad i=1 \quad j=4$

$A^1(1,4) = \min\{A^0(1,4), A^0(1,1)+A^0(1,4)\}$
$= \min\{7, 0+7\}$

$A^1(1,4) = 7$

3) K = 1 i = 3 j = 1

$A^1(3,1) = \min\{A^0(3,1), A^0(3,1) + A^0(1,1)\}$

$= \min\{5, 5+0\}$

$A^1(3,1) = 5.$

- K = 1, i = 3 j = 2

$A^1(3,2) = \min\{A^0(3,2), A^0(3,2) + A^0(1,2)\}$

$= \min\{\infty, 5+3\}.$

$\min\{8\}.$

$A^1(3,2) = 8.$

- K = 1, i = 3, j = 3.

$A^1(3,3) = \min\{A^0(3,3), A^0(3,1) + A^0(1,3)\}$

$= \min\{0, 5+\infty\}.$

$A^1(3,3) = 0.$

- K = 1, i = 3, j = 4

$A^1(3,4) = \min\{A^0(3,4), A^0(3,1) + A^0(1,4)\}$

$= \min(1, 5+7).$

$A^1(3,4) = 1$

$k=1, \quad i=4, \quad j=1.$

$A^1(4,1) = \min\{A^0(4,1), A^0(4,1) + A^0(1,1)\}$

$= \min\{2, 2+0\}.$

$A^1(4,1) = 2$

$k=1, \quad i=4, \quad j=2$

$A^1(4,2) = \min\{A^0(4,2], A^0(4,1) + A^0(1,2)\}$

$= \min\{\infty, \bullet * 2+3\}.$

$A^1(4,2) = 5$

$k=1, \quad j=4, \quad j=3$

$A^1(4,3) = \min\{A^0(4,3) + A^0(4,1) + A^0(1,3)\}$

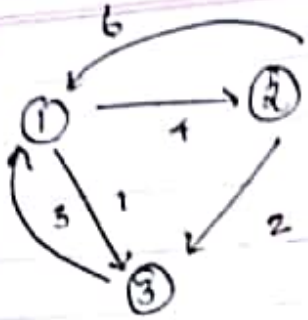
$A^1(4,3) = \infty$

$k=1, \quad i=4, \quad j=4$

$A^1(4,4) = \min\{A^0(4,4), A^0(4,1) + A^0(1,4)\}$

$= \min\{0, 2+7\}.$

$A^1(4,4] = 0$



" If no weight consider it as Transitive Closure "

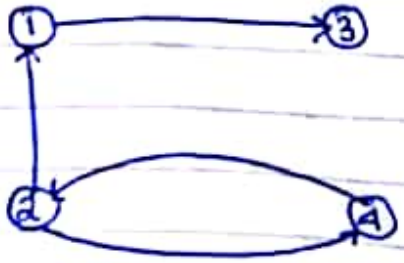
15/10/19

Warshall Algorithm

$D^k[i,j] = D^{k-1}[i,j]$ OR $D^{k-1}(i,k)$ AND $D^{k-1}(k,j)$

Algorithm :-

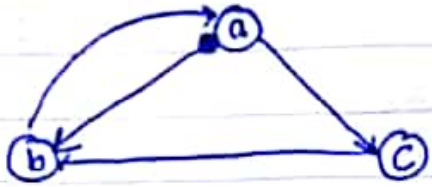
```
1> cost[1:n][1:n]
2> {
3> for (i=1 to n do
4> for j=1 to n do
5> A[i,j] = cost[i,j]
6> for(k=1 to n do] → (4) → table
7> for(i=1 to n do) → rows
8> for(j=1 to n do) → columns
9> A^k[i,j] = (A^{k-1}(i,j) OR A^{k-1}(i,k) AND A^{k-1}(k,j)
10> return A^k
}
```



A^0	1	2	3	4
1	0	0	1	0
2	1	0	0	1
3	0	0	0	0
4	0	1	0	0

16/10/19

Q1) $A^k[i,j] = [A^{k-1}(i,j)$ OR $A^{k-1}(i,k)$ AND $A^{k-1}(k,j)$



A^0	a	b	c
a	0	1	1
b	1	0	0
c	0	1	0

A^1	a	b	c
a	0	1	1
b	1	1	1
c	0	1	0

k i j

-> $A^1(1,1) = A^0(1,1)$ OR $A^0(1,1)$ AND $A^0(1,1)$

$= 0$

$A^1(1,2) = A^0(1,2)$ OR $A^0(1,1)$ AND $A^0(1,2)$

$=$ 1 OR 0 AND 1 -> 1.

$A^1(1,3)$

->

-> $A^1(2,1)$

$A^1(2,2)$

$A^1(2,3)$

-> $A^1(3,1)$

$A^1(3,2)$

$A^1(3,3)$

A^4	a	b	c
a	1	1	1
b	1	1	1
c	1	1	1

-> $A^2(1,1) =$

$A^2(1,2) =$

$A^2(1,3) =$

(14) 1.

~~A³ =~~

A^3	a	b	c
a	1	1	1
b	1	1	1
c	1	1	1

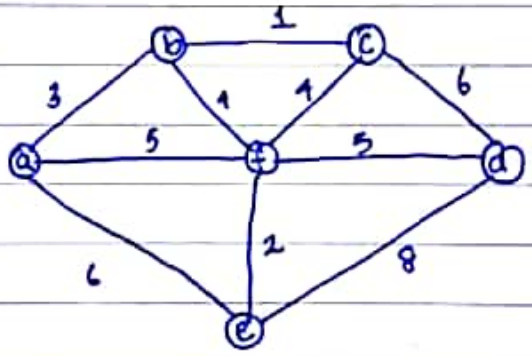
$A^3(i,j) = A^2(1,1)$ OR $A^2(1,3)$ AND $A^2(3,1)$

$= 1$ OR 1 AND 1

$= 1$

(Revision)

→ Prims Algorithm

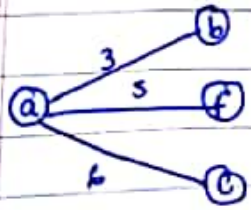


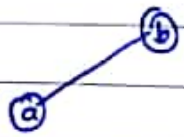
MST | F | F | F | F | F | F |

Key | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |

a b c d e f

Step : 1





MST | T | F | F | F | F | F |

Key | 0 | 3 | ∞ | ∞ | 4 | 5 |

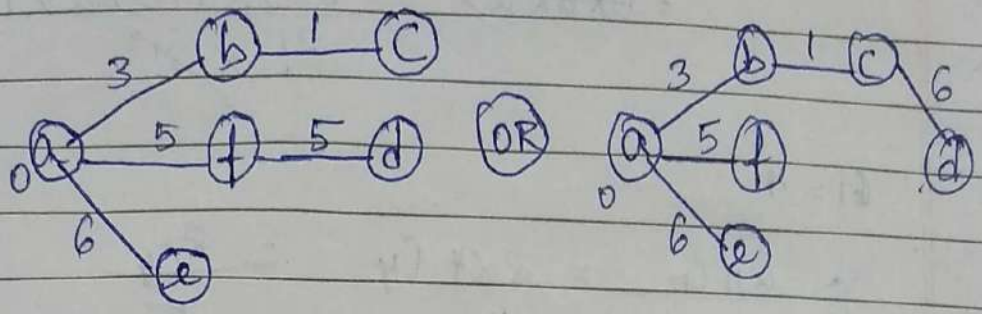
a b c d e f

$d \cdot d > l \cdot d + w(d,e)$

$10 > 6 + 8$

$10 > 14$ $\quad d \cdot d = 10$

$Q = \{d\}$ (10)



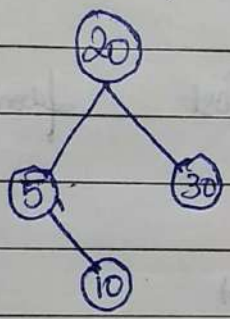
VI Sem

→ Optimal Binary search tree:

↓ Minimum → Divide → which is having each level either 0, 1 or 2 node.

↓ Maximum → increasing order

→ Binary tree 5, 10, 20, 30



no. of possibilities

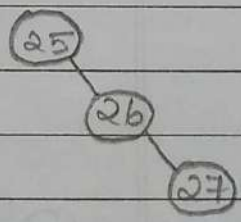
$$\frac{2nCn}{n+1} = \frac{2 \times 3 C_3}{3+1} = \frac{6C_3}{4}$$

$$= \frac{20}{4} = 5.$$

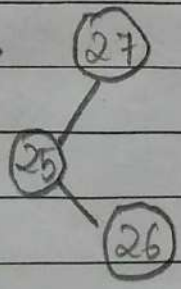
→

elements	25	26	27
frequency	4	3	2

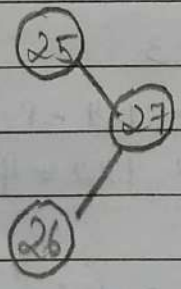
1.



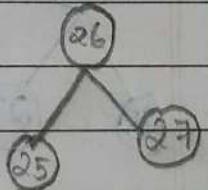
4.



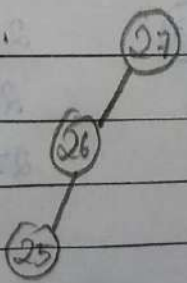
5.



2.



3.



$${}^{6}C_3 = \frac{n!}{r!(n-r)!} = \frac{6!}{3!\,3!}$$

$$= \frac{6\times5\times4\times3\times2\times1}{3!\times3!} = \frac{20}{4} = 5$$

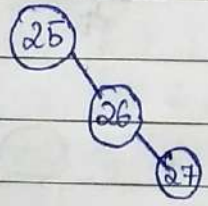
$n = 4$

$$\frac{{}^{2n}C_n}{n+1} = \frac{{}^{2\times4}C_4}{4+1} = \frac{{}^{8}C_4}{5}$$

$$\frac{8!}{4!(8-4)!} = \frac{8!}{4!\times4!} = \frac{8\times7\times6\times5\times4\times3\times2\times1}{4!\times4!}$$

$$= \frac{70}{5} = 14$$

→ Finding the minimum cost from above binary tree.

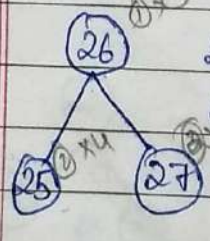
1. 

→ $1 * 4 = 4$

→ $1*3 + 1*3 = 6$ $25=26$ $26=26$

→ $1\times2 + 1\times2 + 1\times2 = 6$

Total Cost = 16

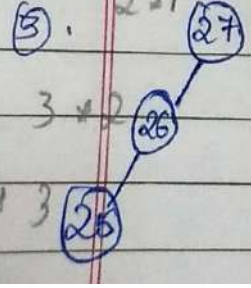
2. 

$26 \rightarrow 1\times3 = 3$

$25 \rightarrow 1\times4 + 1\times4 = 8$

$27 \rightarrow 1\times2 + 1\times2 = 4$ } 15

$3+4+8 = 15$

3. 

$25 \rightarrow$

$26 \rightarrow$ $2+6+12 = 20$

$27 \rightarrow$

	0	1	2	3	4
→ Keys	ϕ	10	20	30	40
frequency	0	4	2	6	3

Step-1: Creating the tables, 0-BST(n+1)(n+1), root table [n+1, n+1].

i/j	0	1	2	3	4
0	0	4	8	20	26
1		0	2	10	16
2			0	6	12
3				0	3
4					0

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

0-BST[n+1][n+1] root table [n+1, n+1]

Step-2: $l = j - i = 0$

Step-3: $l = j - i = 1$

Cost [0,1] = 1

Cost [1,2] = 1

Cost [2,3] = 1

Cost [3,4] = 1

Cost [0,1]

10 → 1 * 4 = 4

Cost [1,2]

20 → ~~2 *~~ 1 * 2 = 2

Cost [2,3]

30 → 1 * 6 = 6

Cost (3,4) 40 → 1 * 3 = 3

Step 4: $l = j - i = 2$

Cost [0,2] = 2

Cost [1,3] = 2

Cost [2,4] = 2

Cost [0,2]

20 → 2 * 2 = 4

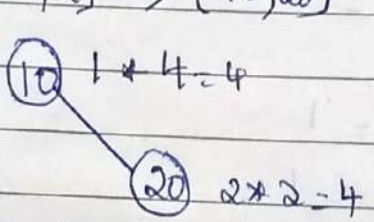
Cost [1,3]

30 → 2 * 6 = 12

Cost [2,4]

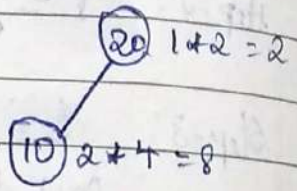
40 → 2 * 3 = 6

→ Cost [0,2] → [10,20]



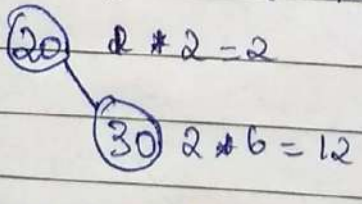
4 + 4 = 8 → min

Cost [0,2] = 8



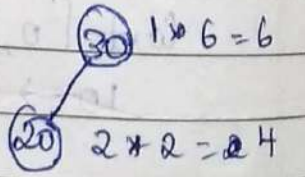
8 + 2 = 10

→ Cost [1,3] → {20,30}



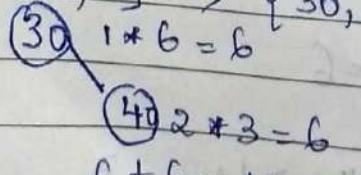
12 + 2 = 14 ~~14 + 10~~

Cost [1,3] = 10



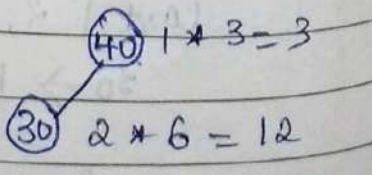
6 + 4 = 10 ✓ min

→ Cost [2,4] → {30,40}



6 + 6 = 12 ✓

Cost [2,4] = 12



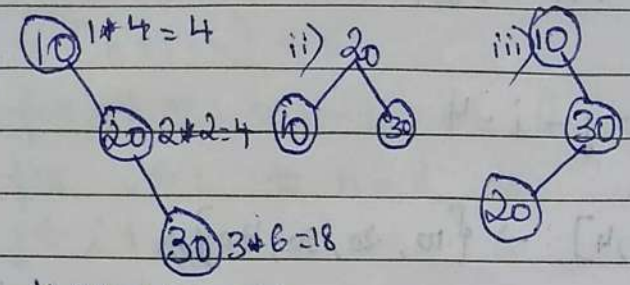
12 + 3 = 15

Step-5: $l = j - i = 3$

Cost [0,3] = 3

Cost [1,4] = 3

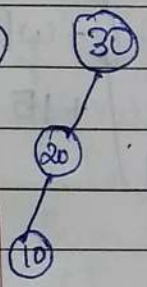
Cost [0,3) = {10, 20, 30}

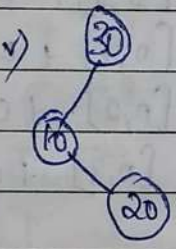
i) 

4+4+18 = 26

ii)

iii)

iv) 

v) 

ii) 1 * 2 = 2
2 * 4 = 8 = ~~28~~ 22
2 * 6 = ~~18~~ 12

iii) 1 * ~~4~~ = 4
2 * 6 = 12 = ~~22~~
3 * 2 = 6

iv) 1 * 6 = 6
2 * 2 = 4
3 * 4 = 12 = 22

v) min
1 * 6 = 6
2 * 4 = 8 = 20
3 * 2 = 6

Cost [0,3] = 20

→ Cost [1,4] → {20, 30, 40}

$$Cost[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] \} + w(i,j)$$

$w = 2+6+3$
$w = 11$

$K = 2, 3, 4$

$$Cost[1,4] = \min \begin{cases} c[1,1] + c[2,4] \\ c[1,2] + c[3,4] \\ c[1,3] + c[4,4] \end{cases} + w(i,j)$$

$$+ 11$$

$12 + 11 = 23$

$5 + 11 = 16$ ✓ min.

$10 + 11 = 21$

Step-6: $l = j - i = 4$

$Cost[0,4] \rightarrow \{10, 20, 30, 40\}$

$$Cost[0,4] = \min \begin{cases} c[0,0] + c[1,4] \\ c[0,1] + c[2,4] \\ c[0,2] + c[3,4] \\ c[0,3] + c[4,4] \end{cases} + w(i,j)$$

$w = 4+2+6+3$
$w = 15$

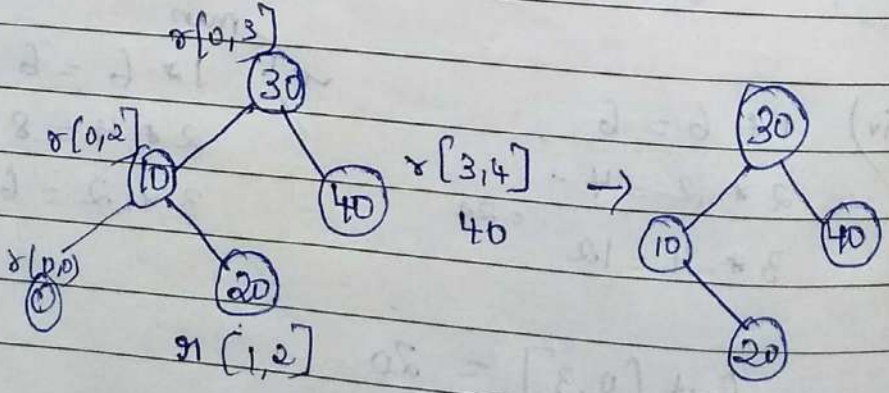
$+ 15$

$16 + 15 = 31$

$16 + 15 = 31$

$11 + 15 = 26$ ✓ min

$20 + 15 = 35$



→ Dynamic Programming algorithm :-

```
1.  Optimal BST (P[1...n])
2.  for i ← 1 to n do
3.      C[i, i-1] ← 0
4.      C[i, i] ← P[i]
5.      R[i, i] ← i
6.      C[n+1, n] ← 0
7.  for d ← 1 to n-1 do // diagonal count
8.      for i ← 1 to n-d do
9.          j ← i+d
10.         min val ← ∞
11.         for K ← to j do
12.  if C[i, K-1] + C[K+1, j] < min val
13.  min val ← C[i, K-1] + C[K+1, j]; K min ← K
14.  R[i, j] ← K min
15.  Sum ← P[i]; for S ← i+1 to j do sum ← sum + P[i
16.  C[i, j] ← min val + sum
17.  return C[1, n], R.
```

2/11/19

Module-4

Backtracking

General method of Backtracking:

Step-1: Tests to see whether a solution has been found & if so returns it, Otherwise,

Step-2: for each choice that can be made at this point.
→ make the point.
→ Recursion
→ If the recursion returns the solution return it.

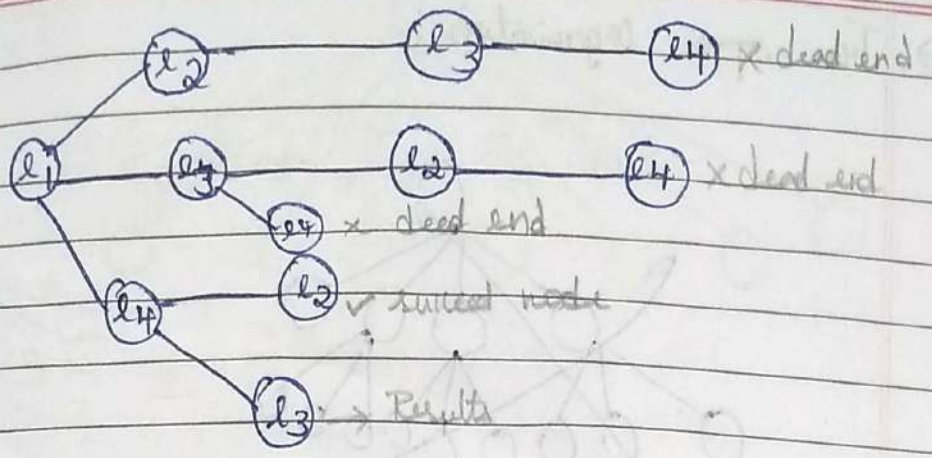
Step-3: If no choices remain, return failure.

Application:

→ Eight - Queen's problem
→ Sum of subsets
→ Graph Colouring
→ Knapsack problem.

$E = \{e_1, e_2, \cdots e_n\}$ Result-R_i

$E = \{3, 1, 2, 3\}$ $R = 7$



→ Eight - Queen's problem :

N Queen's problem of placing N Chess queen and NxN Chess board so that no two queen attack each other, We have to check that no two Queens are placed in same row, same column, and same diagonals.

4 Queen's Problem:

$Q = \{Q_1, Q_2, Q_3, Q_4\}$

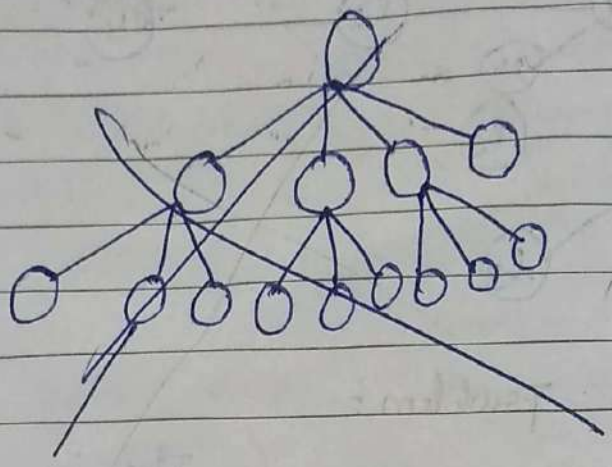
	Q_1		
			Q_2
Q_3			
			Q_4

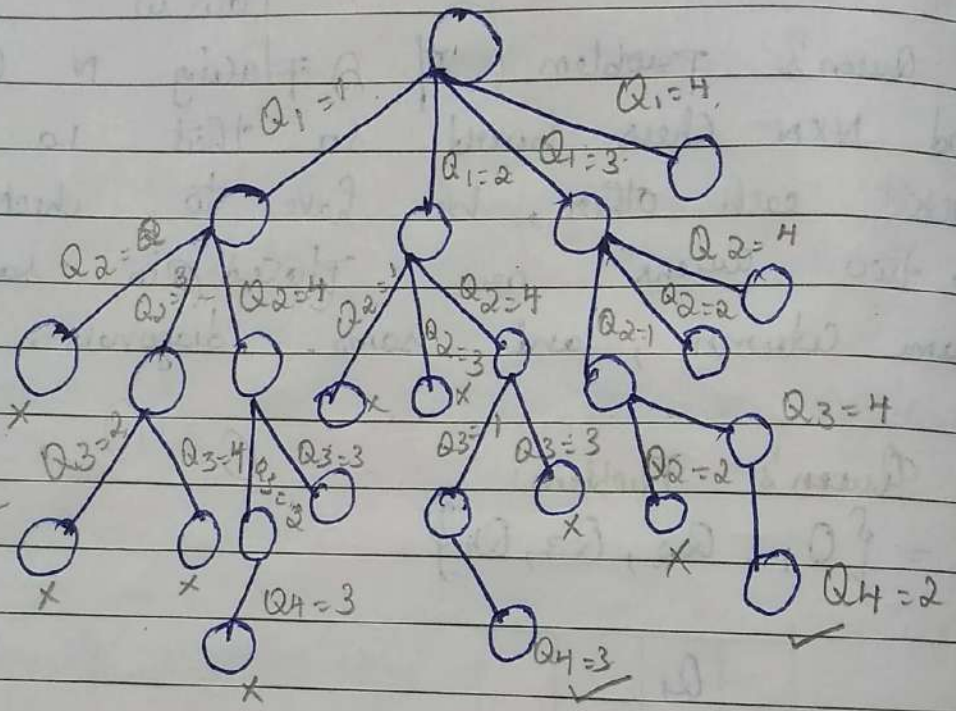
Solution 1 = $\{Q_1, Q_2, Q_3, Q_4\}$
$(1,2)\ (2,4)\ (3,1)\ (4,3)$

		Q_1	
Q_2			
			Q_3
	Q_4		

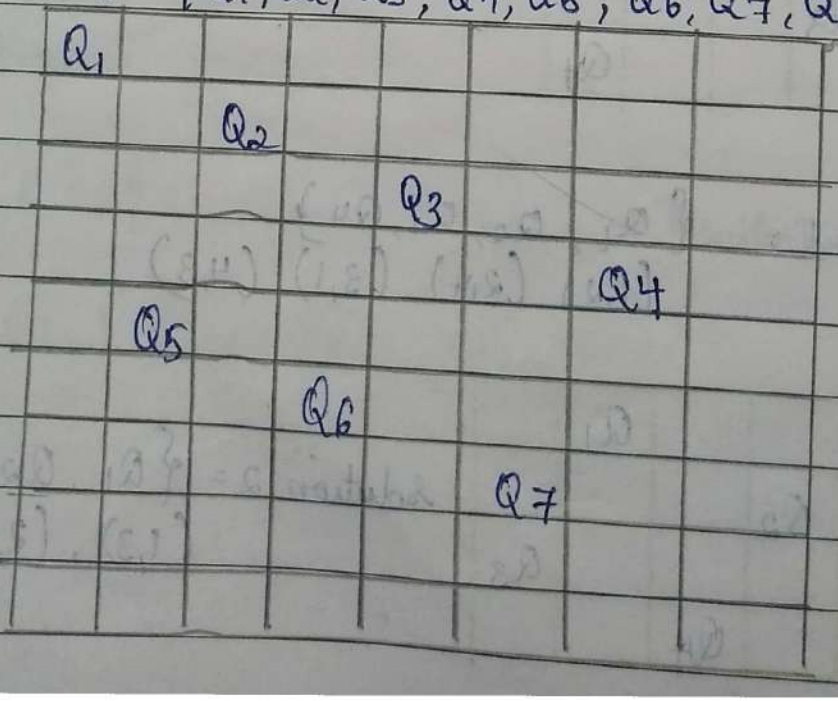
Solution 2 = $\{Q_1, Q_2, Q_3, Q_4\}$
$(1,3), (2,1)\ (3,4)\ (4,2)$

→ Tree - star Organisation :





$Q = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8\}$



→ Backtracking algorithm:

Step-1: Start in the left most Column
Step-2: If all queens are placed return true.
Step-3: Try all rows in the current Column, do following for every tried row.

a) if the queen can be placed safely in this row then mark this [row, Column] as part of the solution, & recursively check if placing queen here leads to a solution.

b) if placing the queen in [row, Column] leads to a solution then return true.

c) if placing queen doesn't lead to a solution then unmark this [row, column] backtrack & go to step (a) to try other rows.

Step-4: if all rows have been tried & nothing worked return false to trigger backtracking.

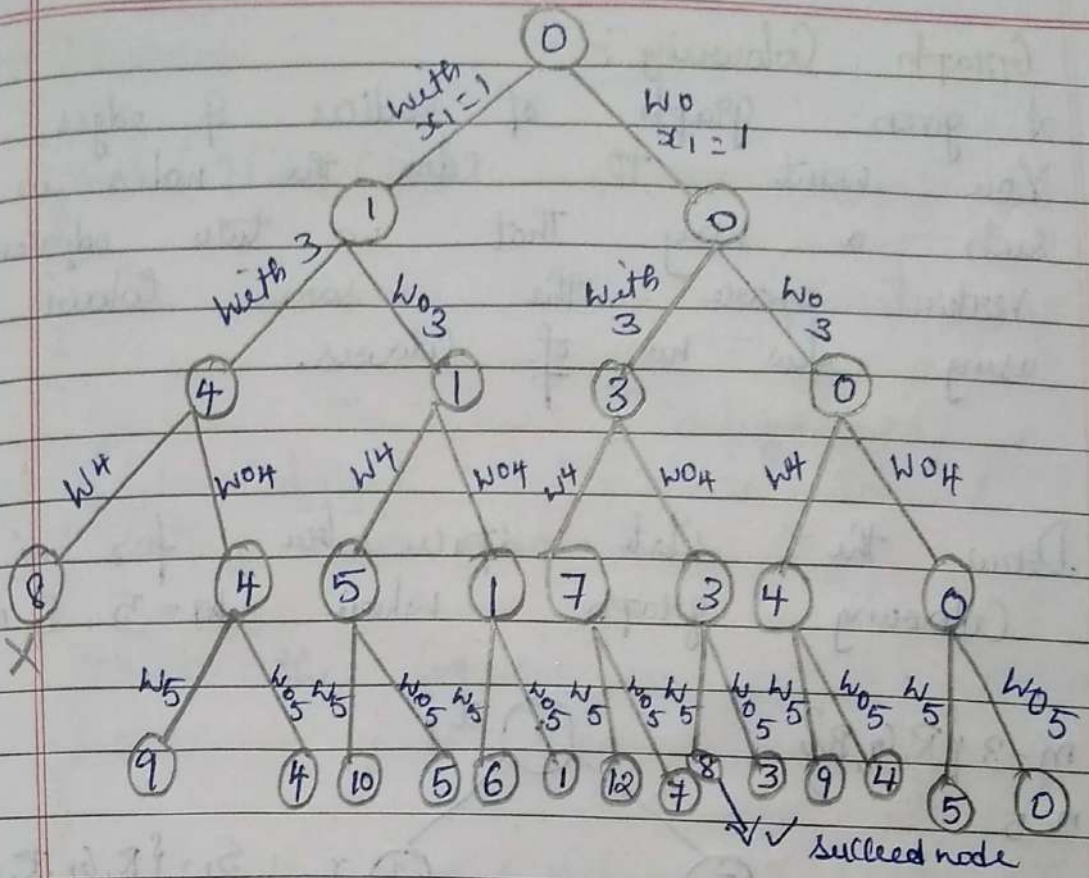
→ Sum of subsets Problem:

Problem statement:

The problem is to find a subset of a given set $S = \{S_1, S_2, S_3 \cdots S_n\}$ of n the integer whose sum is equal to a given +ve integer "d"

Example $S = \{1, 3, 4, 5\}$ $d = 8$.

$S_1 = \{1, 3, 4\}$
$S_2 = \{3, 5\}$



Q> Write the Pseudocode for backtracking algorithm let W = {3, 5, 6, 7} and m=15. find all possible subset of W that sum to be "m" and draw the state space tree that is generated.

→ Subset (W[], m)
{
if (subset is satisfying the Constraint) Print the subset Exclude the Current element and Consider next element
else
generate the nodes of present level along breadth of tree & recursive for next levels
}.

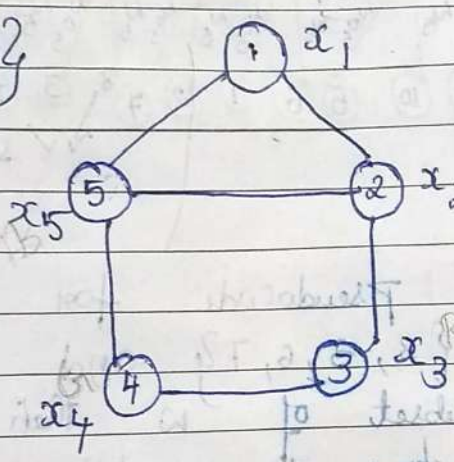
→ Graph Colouring :-

A given graph of vertices & edges, You want to color the nodes in such a way that no two adjacent vertex share the same colour by using few no. of colours.

Draw the state space tree for 'm' Colouring graph where m = 5 & n = 3.

m = 3 {R, G, B}

n = 5



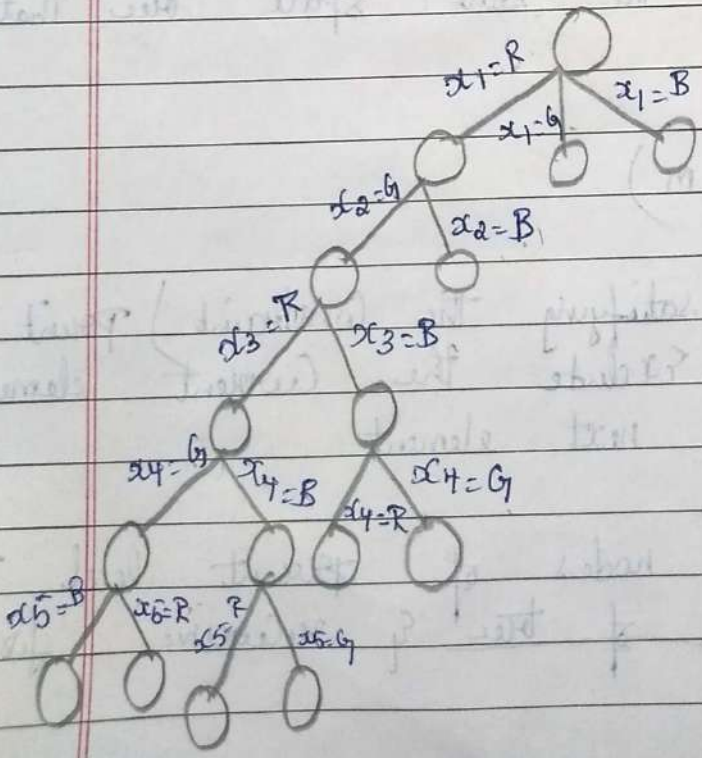
$S_1 = \{R, G, R, G, B\}$

$S_2 = \{R, B, R, B, G\}$

$S_3 = \{G, R, G, R, B\}$

$S_4 = \{B, R, B, R, G\}$

$S_5 = \{B, G, B, G, R\}$



→ Pseudocode:

```
graph ( g[], c[])
{
while there are untried configurations
{
 generate the next configuration
 if ( no adjacent vertices are coloured
   with same colour)
 {
  Print the configuration.
 }
}
}
```

→ Knapsack problem:

	o_1	o_2	o_3	o_4	o_5
Object (o):	1	2	3	4	5
Profit (P):	15	10	6	9	18
Weight (w):	3	4	8	1	14

2/11/19

Module - 5

Branch and Bound

Ll'y to BT → Tree & space, OP → (minimization), LC - least cost.

→ LC Branch & bound 0/1 Knapsack.

1. Draw a state space tree and set upper = ∞
2. Compute $\hat{c}(x)$, $u(x)$ for each node.
3. $u(x) = -\Sigma p_i$

$$\hat{c}(x) = u(x) * \left[\frac{[m \text{ is current total weight}]}{\text{actual weight of remaining object}}\right] * [\text{actual profit of remaining object}]$$

4. if $u(x)$ is minimum then upper will be set to $u(x)$.
5. if $\hat{c}(x) >$ upper, Kill node x.
6. Otherwise the min cost $\hat{c}(x)$ becomes E-node & Generate Children for E-node.
7. Repeat step 2 to 6 until all nodes get covered.
8. The min cost $\hat{c}(x)$ becomes the answer node. Trace the path in backward direction from x to root for solution subset.

→ LC ~~Branch & bound~~ 0/1 Knapsack: FIFO.

Same steps:

Step-6: Next live node becomes E-node & Generate children for E-node.

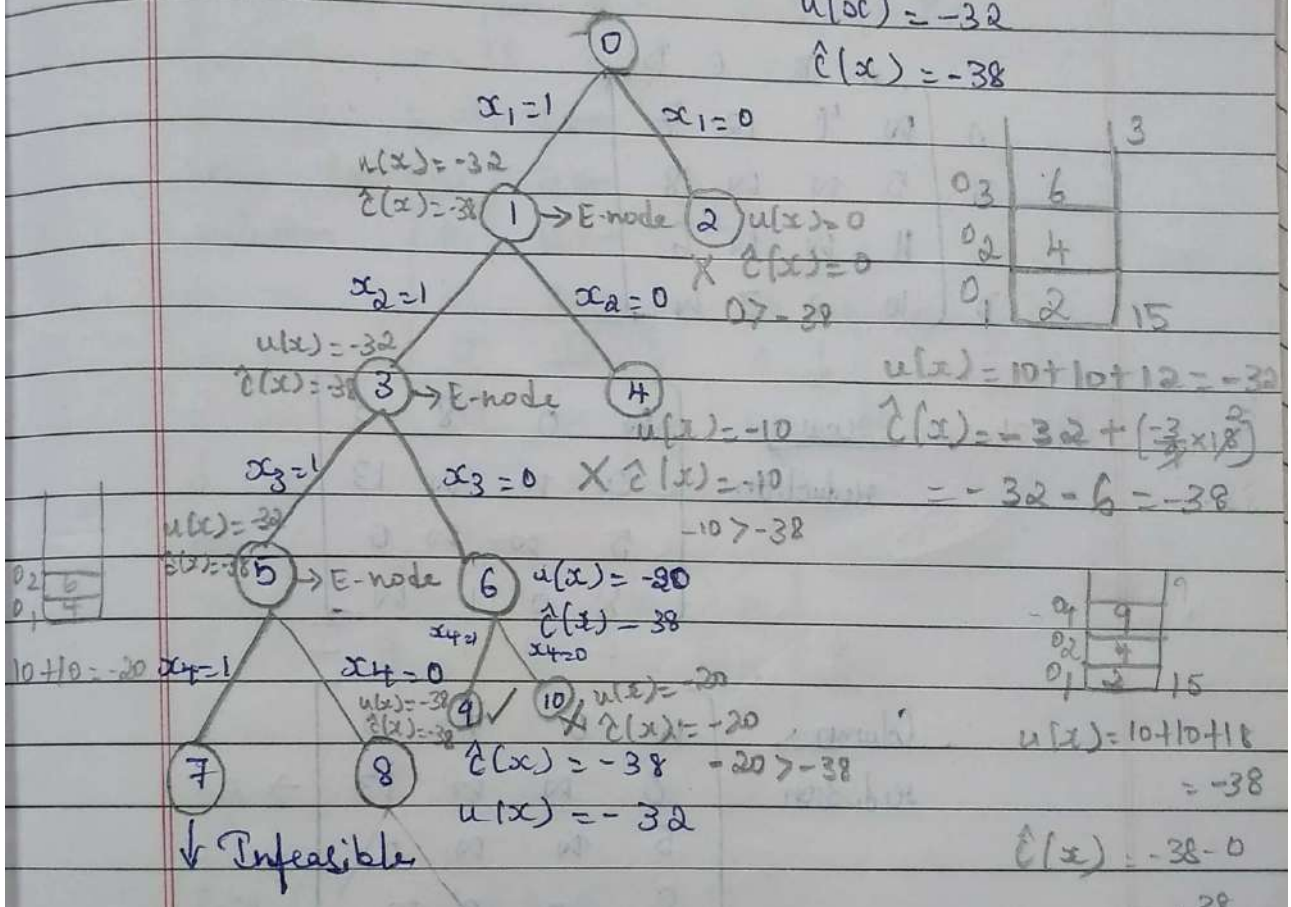
O_i	1	2	3	4
P_i	10	10	12	18
W_i	2	4	6	9

$M = 15$

$2+4+6+\frac{3}{9}*9 = 15$

$10+10+12+\frac{3}{9}\times 18 = -38$

upper = ~~-30~~ = -32 = -38



$u(x) = -32$

$\hat{c}(x) = -38$

	3
O_3	6
O_2	4
O_1	2

15

$u(x) = 10+10+12 = -32$

$\hat{c}(x) = -32 + (\frac{3}{9}\times 18)$

$= -32 - 6 = -38$

$-10 > -38$

O_2	6
O_1	4

$10+10 = -20$

O_4	9
O_2	4
O_1	2

15

$u(x) = 10+10+18$

$= -38$

$\hat{c}(x) = -38 - 0$

$= -38$

$x_4 = 1$

$x_3 = 0$

$x_2 = 1$

$x_1 = 1$

Max profit $= 10+10+18 = -38$

$O_1, O_3 = -13$

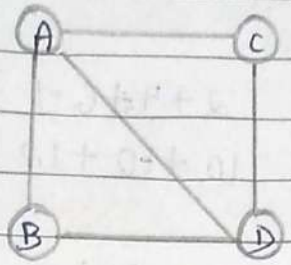
$O_1, O_4 = -13$

5
2

$u(x) = 3+10 = -13$

$\hat{c}(x) = -13.$

→ Travelling Sale's Person Problem:



	A	B	C	D	
A	∞	4	12	7	→ 4
B	5	∞	∞	18	→ 5
C	11	∞	∞	6	→ 6
D	10	2	3	∞	→ 2
					17

min cost

↓ Row reduction

∞	0	8	3	
0	∞	∞	13	
5	∞	∞	0	
8	0	1	∞	
0	0	1	0	1

↓ Column reduction

∞	0	7	3	
0	∞	∞	13	→ A
5	∞	∞	0	
8	0	0	∞	

Cost(A) = 17 + 1 = 18

↓ Row reduction

∞	∞	∞	∞	
∞	∞	∞	13	13
5	∞	∞	0	0
8	∞	0	∞	0

↓ after Row reduction

∞	∞	∞	∞
∞	∞	∞	0
5	∞	∞	0
8	∞	0	∞
5	–	0	0

Horizon
Date
Page

→ Column reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & 4 & \infty & 0 \\ 0 & \infty & \infty & 0 \\ 3 & \infty & 0 & \infty \end{bmatrix} \rightarrow B$$

Cost (B) = 13 + 5 = 18

Cost (B) = $\gamma_{pat} + \gamma$ + Cost (A, B)

= 18 + 18 + 0

= 36.

→ row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & 0 \end{bmatrix}$$

→ Column reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty \end{bmatrix} \rightarrow C$$

Cost (C) = 18 + 0 + 7

= 25

→ row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix} 5$$

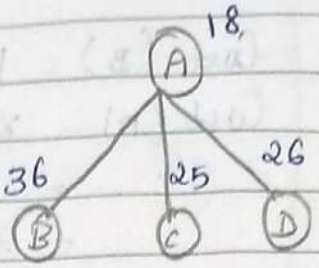
after row → reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix} 0$$

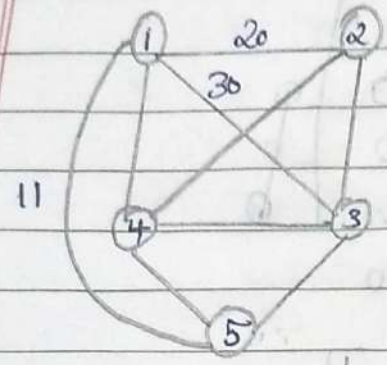
Column reduction →

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix} \begin{matrix} \\ \\ \\ 0 \end{matrix} \rightarrow D$$

Cost (D): 18 + 5 + 3
= 26.



→



	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4
						21

↓

row reduction →

∞	10	20	0	1	
13	∞	14	2	0	
1	3	∞	0	2	
16	3	15	∞	0	
12	0	3	12	∞	
1	0	3	0	0	4

∞	9	19	0	0	
13	∞	14	2	0	
-2	0	∞	0	-1	
16	3	15	∞	0	
12	0	3	12	∞	

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

→ 1

Cost (A) = 21 + 4
= 25.

Horizon
Date
Page

(1,2)

∞	∞	∞	∞	∞	–
∞	∞	11	2	0	0
0	∞	∞	0	2	0
15	∞	12	∞	0	0
11	∞	0	12	∞	0
					0

row reduction

after row reduction

∞	∞	∞	∞	∞	
∞	∞	11	2	0	
0	∞	∞	0	2	
15	∞	12	∞	0	
11	∞	0	12	∞	
0	–	0	0	0	0

Column reduction

∞	∞	∞	∞	∞	
∞	∞	11	2	0	
0	∞	∞	0	2	→ 2
15	∞	12	∞	0	
11	∞	0	12	∞	0

Cost (2) = r_{PW} + r + Cost (1,2)
= 25 + 0 + 10
= 35

(1,3)

∞	∞	∞	∞	∞	–
~~∞~~ 12	∞	∞	2	0	0
∞	~~∞~~ 3	∞	0	2	0
15	~~∞~~ 3	∞	∞	0	0
11	~~∞~~ 0	∞	12	∞	0
					0

row reduction

after row reduct

∞	∞	∞	∞	∞	
12	∞	∞	2	0	
∞	3	∞	0	2	
15	3	∞	∞	0	
11	0	∞	12	∞	
11	0	–	0	0	

Column reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix} \rightarrow 3$$

$$\text{Cost}(3) = 25 + 11 + 17$$
$$= 53.$$

(1,4)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty & 0 \\ \infty & 0 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} - \\ -0 \\ -2 \\ -0 \\ -0 \end{matrix}$$

row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} - \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

0

after row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \\ 0 & 0 & 0 & - & 0 \end{bmatrix} \quad 0$$

Column reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \rightarrow 4$$

Cost (4) = 25 + 0 + 0
= 25

(1,5)

∞	∞	∞	∞	∞	–
12	∞	11	2	∞	– 2
0	3	∞	0	∞	– 0
15	3	12	∞	∞	– 3
∞	0	0	12	∞	– 0
					5

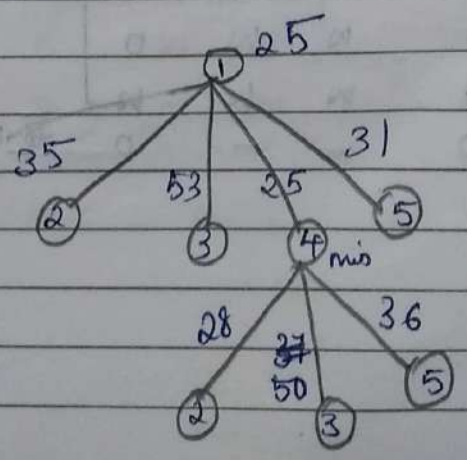
Row reduction

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞
0	0	0	0	– 0

Column reduction

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞

Cost (5) = 25 + 5 + 1
= 31



(4,2)

∞	∞	∞	∞	∞
~~∞~~ 12	∞	11	∞	0
0	~~∞~~ 3	∞	∞	2
∞	~~∞~~ 3	12	∞	0
11	~~∞~~ 0	0	∞	∞

∞	∞	∞	∞	∞	–
∞	∞	11	∞	0	0
0	∞	∞	∞	2	0
∞	∞	∞	∞	∞	–
11	∞	0	∞	∞	0
0	–	0	–	0	

Cost (2) = Cost (4) + r + cost (4,2)
= 25 + 0 + 3 = 28

(4,3)

∞	∞	∞	∞	∞	–
12	∞	∞	∞	0	0
∞	3	∞	∞	2	2
∞	3	∞	∞	0	0
11	0	∞	∞	∞	0
					2

∞	∞	∞	∞	∞
12	∞	∞	∞	0
~~∞~~ 0	3	∞	∞	2
∞	3	∞	∞	0
11	0	∞	∞	∞
~~∞~~ 11	0	–	–	0

∞	∞	∞	∞	∞
12	∞	∞	∞	0
∞	3	∞	∞	2
∞	3	∞	∞	0
0	0	∞	∞	∞

Cost (3) = ~~cost~~ 25 + 13 + 12
= ~~37~~ 50

(4,5)

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	3	∞	∞	2
∞	3	12	∞	0
11	0	0	∞	∞

row

∞	∞	∞	∞	∞	—
12	∞	11	∞	∞	11
0	3	∞	∞	∞	0
∞	∞	∞	∞	∞	—
∞	0	0	∞	∞	0

11

after row / column

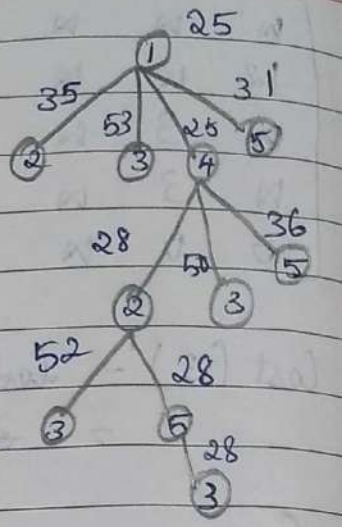
∞	∞	∞	∞	∞
1	∞	0	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞
0	0	0	—	—

0

Horizon
Date
Page

Column

∞	∞	∞	∞	∞
1	∞	0	∞	∞
0	0	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞



Cost (5) = 25 + 11 + 0
= 36.

(2,3)

∞	∞	∞	∞	∞
∞	∞	11	∞	0
0	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞

(2)

∞	∞	∞	∞	∞	-
∞	∞	∞	∞	∞	-
∞	∞	∞	∞	2	2
∞	∞	∞	∞	∞	-
11	∞	∞	∞	∞	11

13

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
0	∞	∞	∞	∞

0

Cost (2,3) = 28 + 13 + 11
= 52

(2,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \quad \underline{0}$$

Cost (2,5) = 28 + 0 + 0

= 28

Total Cost = 25 (1, 4, 2, 5, 3).

10 + 6 + 2 + 7 = 25

Module 3

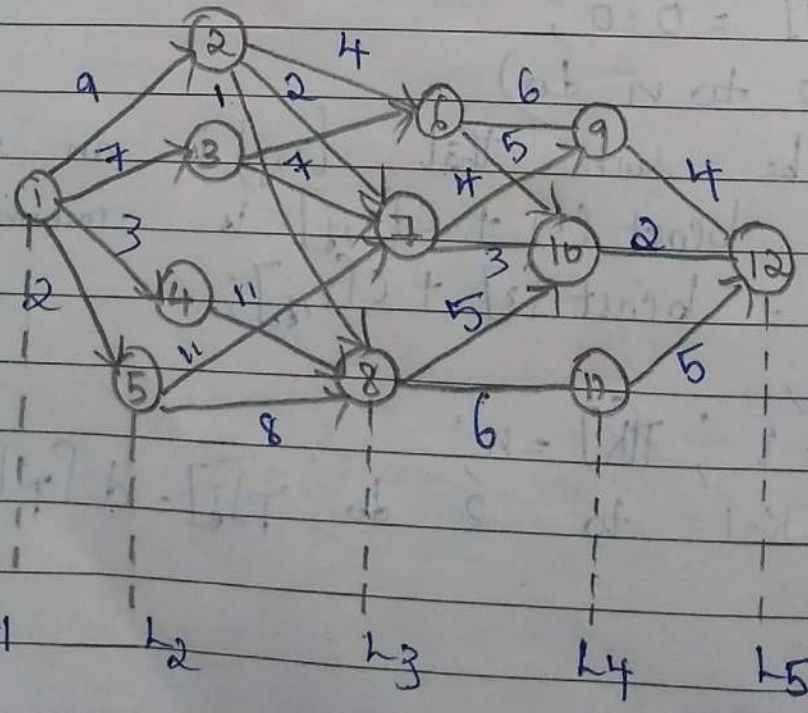
→ **Multistage graph:**

Algorithm Backward approach:

```
Algorithm BGraph (G, k, n, p)
{
 bcost[1] = 0.0;
for (j=2 to n do)
Let r be such that (r,j) is an edge of
G and bcost(r) + c[r,j] is minimum;
bcost[j] = bcost[r] + c[r,j];
d[j] = r;
p[1] = 1 ; p[k] = n;
for j = k-1 to 2 do p[j] = d[p[j+1]];
}
```

→ Algorithm for forward approach:

```
FGraph ( Graph G, int K, int n, int P[])
{
float Cost [max size], int d [max size], r;
 Cost [n] = 0.0;
for (int j = n-1; j >= 1; j--)
{
Let r be a vertex such that is an edg
  G and  C[j] [r] + Cost [r] is minimum
   Cost [j] = Cost [j] [r] + Cost [r]
   D [j] = r;
}
 P[1] = 1, P[K] = n;
for ( j = 2; j <= K-1; j++)
P[j] = d [P(j-1)];
}
```



Horizon

Date
Page

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	~~20~~ 18	15	7	5	7	4	2	5	0
d	2/3	7	6	8	8	10	10	10	12	12	12	12

Cost (5 12) = 0 } level 5
level vertex

Cost (4,9) = 4
Cost (4,10) = 2 } level 4
Cost (4,11) = 5

Cost (3,6) =
Cost (3,7) = } level 3
Cost (3,8) =

Cost (3,6) = min { Cost (6,9) + Cost (4,9)
Cost (6,10) + Cost (4,10) }

= min { 6 + 4 , 5 + 2 }
= min { 10, 7 }

Cost (3,7) = min { Cost (7,9) + Cost (4,9)
Cost (7,10) + Cost (4,10) }

= min { 4+4 , 3 +2 }
= min { 8 , 5 }

Cost (3,8) = min { Cost (8,10) + Cost (4,10)
Cost (8,11) + Cost (4,11) }

= min { 5+2 , 5+5 }
= min { 7, 10 }

Cost (2, 2) =
Cost (2,3) =
Cost (2,4) =
Cost (2,5) =

$$\text{Cost}(2,2) = \min \begin{cases} \text{Cost}(2,6) + \text{Cost}(3,6) \\ \text{Cost}(2,7) + \text{Cost}(3,7) \\ \text{Cost}(2,8) + \text{Cost}(3,8) \end{cases}$$

$$= \min \begin{cases} 4+7 \\ 2+5 \\ 1+7 \end{cases} \begin{matrix} \to 11 \\ \to 7 \\ \to 8 \end{matrix}$$

$$\text{Cost}(2,3) = \min \begin{cases} \text{Cost}(3,6) + \text{Cost}(3,6) \\ \text{Cost}(3,7) + \text{Cost}(3,7) \end{cases}$$

$$= \min \begin{cases} 2+7 \\ 7+5 \end{cases} \begin{matrix} \to 9 \\ \to 12 \end{matrix}$$

$$\text{Cost}(2,4) = \min \{ \text{Cost}(4,8) + \text{Cost}(3,8) \}$$
$$= \min \{ 11 + 7 \} \to 20\ 18$$

$$\text{Cost}(2,5) = \min \begin{cases} \text{Cost}(5,7) + (3,7) \\ \text{Cost}(5,8) + (3,8) \end{cases}$$

$$= \min \begin{cases} 11+5 \\ 8+7 \end{cases} \begin{matrix} \to 16 \\ \to 15 \end{matrix}$$