What is Algorithm ?

Algorithm is a sequence of unambigous instruction for Solving the computation Problem i.e for obtainting required output to any given legimate Input in infinite amount of time.

Algorithm	Programe
→ Design	Implement
→ Domain know	Programmer
→ any Knowledge	programing lang
→ H/w & OS	H/w & OS related
→ Analysis	Testing

Characterstics or Properties.

1. Finiteness :- Algorith should terminate (Stop) in finite or no. of Steps.

2. Definiteness : Each Step must be rigously and unambiguously Specified.

3. Input : Valid Inputs or the range of Inputs must be Clearly specified 0 or more

4. Effectiveness : Steps must be simple and basic the instruction should transform the the given input to the desired output.

5. Output : The Algorithm should produce correct result and one output has to be produced.

esign of Algorithm methods

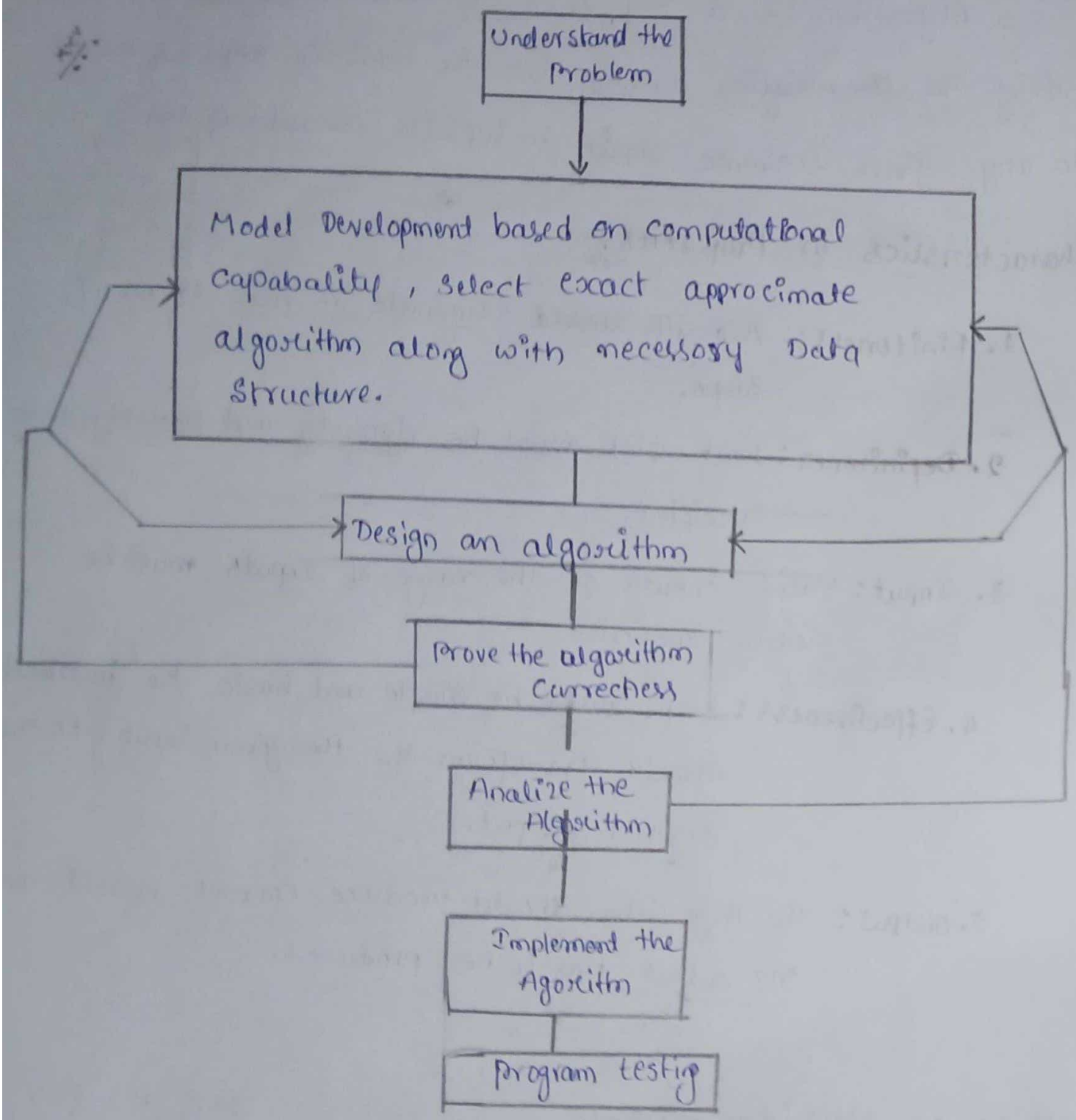
1) Natural Language → is that it can be interpreted in different ways by

2) psedocode → Mix of natural language and programming Constraints.

3) Flowchart →

↳ Advanced problem it becames difficult to represent algorithm.

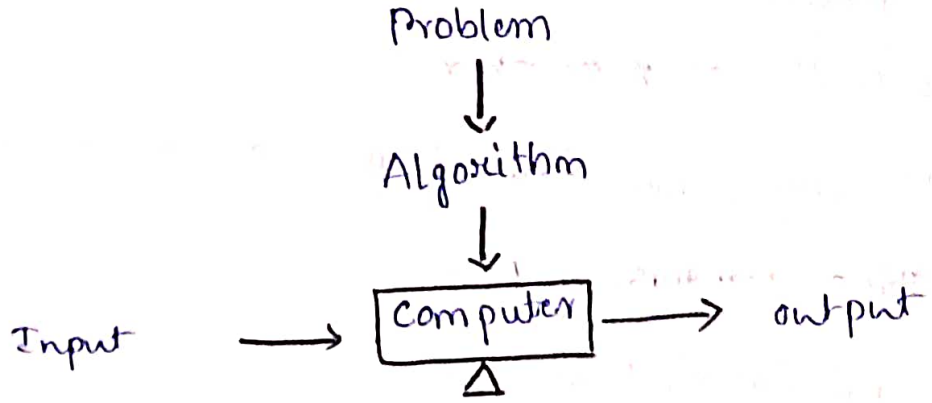
Program Development Steps.



Documentation is one of the step in program Development.

Notion of Algorithm.

ಕಲ್ಪನೆ



→ How to Analyse and Algorithm.

→ Space Efficiency (parameter) → The space complexity can be understood as the amount of space required by an algorithm to run to complition.

memory

→ Program Space

→ Data Space

→ Stack Space

→ Time efficiency → Time complexity is a function of input size n that refers to the amount of time needed by an algorithm to run complition.

→ Speed of the computer ✓

→ Choice of Programing Language ✓

→ Compiler used ✓

→ Choice of Algorithms ✓

→ Number (size) of Input output ✓

Example

Algorithm Swap (a, b)

{

temp = a; → 1

a = b; → 1

b = temp; → 1

}

$f(n) = 3$

Note : Each Statements take 1 unit of time.

$\underbrace{x = 5 \times a + 6 \times b}$ → 1

↳ this statement take one unit of time

* Pseudo Code

```
Algorithm Largest(n)
//purpose: To find of n number
//Description: Algorithm finds largest of n numbers.
// Input: Three Integers a, b, c
if (a > b) and (a > c) then
    print a is largest
else if (b > c) then
    Print b is largest
else
    Print c is largest.
```

Asymptotic Notation

1) Big - oh (O) - Upper bound

2) Big - Omega (Ω) - Lower bound

3) Big - Theata (Θ) - Average bound.
(this one will use)

Asymptotic Notation is used to describe the running time of an algorithm, How much time an algorith takes with a Given Input, n

Time efficiency is determined by the no-of repeatation of the basic operation as the function of Input Size.

Basic operation:

$T(n) \approx C_{op} C(n)$

$\approx$ → Approximately Equal to

n = Running time of an Algorithm.

= Execution time for the basic operation

= Size of Input

= No.of time for the basic operation if exit.

Time Complexity // Add 2 Matrixes

```
Algorithm Add(A, B, n)
{                                                  → n+1
    for (i=0; i<n; i++)
    {                                              → (n+1)*n = n^2+n
        for (i=0; i<n; j++)
        {
            c[i,j] = A[i,j] + b[i,j];              → (n^2 * n) * 1
        }
    }
}
```

n = 3

0

n<0 ✗

n<1

n<2

n<3 check

false n+1 times

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

Order Of Growth

Order Of growth with a constant multiple as

$n \to \infty$

Eg

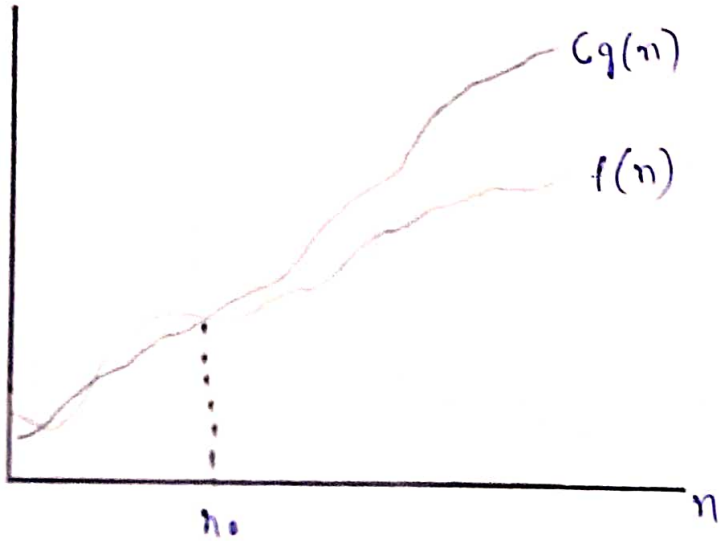
* How much faster will algorithm run on Computer that is twise as fast ?

Formal Defination Of Asymptotic Notation

1) Big-oh Notation

let $f(n)$ be the time efficiency of an algorithm the function $f(n)$ is set to be in Big-oh $O(g(n))$ denoted by $f(n)$ belongs to $f(n) \in O(g(n))$ such that thier exits a positive Constance "c" and the positive integer n_0 Satisfieing the Constraints.

$$\boxed{f(n) \leq cg(n) \ \forall \ n \geq n_0}$$

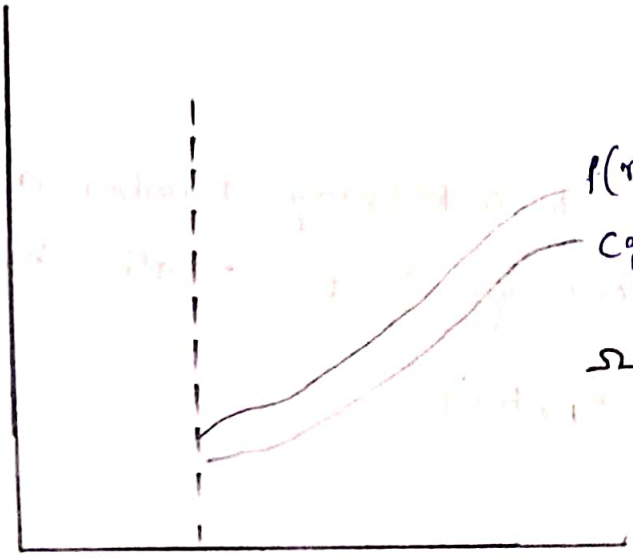


Big-oh notation: $f(n) \in O(g(n))$

Big Omega Notation - Ω

let $f(n)$ be the time efficiency of an algorithm the function $f(n)$ is set to be Big-Omega

$$\boxed{f(n) \geq c\,g(n) \;\forall\; n > n_0}$$



$\Omega g(n)$:
Class of functions $f(n)$ that grow at least as fast $g(n)$

Big-Omega notation

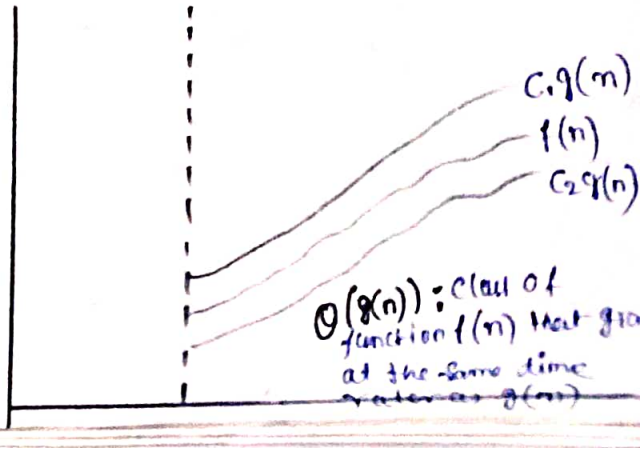
Big Theota (θ) Notation

let $f(n)$ be the time efficiency of an algorithm

$f(n) \in \theta(g(n))$

Such that, there exits a positive constant c & c_2 and the positive integer n_0 satisfying the constraints

$$\boxed{c_1 \times g_1(n) \leq f(n) \leq c_2 \times g_2(n) \;\forall\; n > n_0}$$



$\theta(g(n))$: Class of function $f(n)$ that grow at the same time rate as $g(n)$

The Property of Asymption Notation

If an Algorithm as a exicutable path the Analisis of such an analisis can be obtaing the following theorum.

$$\boxed{\begin{array}{l}\text{If } f_1(n) \in O(g_1(n)) \text{ and } f_2(n) \in O(g_2(n)) \text{ then} \\ f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})\end{array}}$$

xxx 6-10

let us consider the 4 orbitrary number a_1 a_2 and b_1, b_2 then the following simple relation holds good.

$$a_1 + a_2 \leq 2\max(b_1, b_2)$$
$$1 + 2 \leq 2\max(2, 3)$$

Proofe

By Definition of Asymptotic Notation we know that

$$\boxed{\begin{array}{l} f(n) \in O(g(n)) \text{ if} \\ f(n) \leq cg(n) \ \forall \ n > n_0 \end{array}}$$

It is given that

$$\boxed{f_1(n) \in O(g(n)) \quad f_1(n) < C_1 + g_1(n) \ \forall \ n \geq n_0}$$

$$f_2(n) \leq C_2 g_2(n) \ \forall \ n \geq n_2 \quad \text{—} (2)$$

Continue

letus assume that

$C_3 = \max\{C_1, C_2\}$ $\quad n = \max\{n_1, n_2\}$

By adding 1 and 2 we get $f_1(n) + f_2(n) \leq C_1 * g_1(n) + C_2 * g_2(n)$

$f_1(n) + f_2(n) \leq C_1 * g_1(n) + C_2 * g_2(n) \leq C_3 * g_1(n) + C_3 * g_2(n)$

using sign 3

$f_1(n) + f_2(n) \leq C_3 * 2 \max\{g_1(n) + g_2(n)\}$

$f_1(n) + f_2(n) \leq \max(g_1(n) + g_2(n))$

where $C = 2C_3 = \max(C_1, C_2)$

$n_0 = \max(n_1, n_2)$

Example

→ Find the time complexisity of an algorithm with has 2 parts.

* The sorting porting requiring $n * n - n$ comparison.

General Plan for Analysis of Non-Recursive Algorithm.

* Deciding on parameter n idicate inputsize
* Identify the basic operation on algorithm.
* Determine worst, avarage, and best cases for the Input of size n if required.
* Setup a sum for the no. of times the basic operation is executed.
* Simplify the sum using standard formulas and rules.

* Algorithm

MaxElement ([A[0.... n-1])

//Determine the values of largest element in the given array.

// Input: An Array A[0.... n-1] of real numbers.

// output: the value of largest element in a max

```
max ← A[0]
for i ← 1 to n-1 do
    if A[i] > maxValue
        maxval ← A[i]
return maxvalue;
```

→ Basic Operation

//Assuming the basic operation takes 1 unit of time.

Consider an example of 5 Elements

20 40 25 55 30

Input Size = 5

Basic operation: the no. of comparisons for n-1 times and it dependous on the input size.

$$T(n) = \sum_{i=1}^{n-1} 1 = u - l + 1$$

upper

lower

$$= n - 1 \cancel{-1+1}$$

$$= n - 1$$

$$T(n) \approx n$$

$\frac{0}{0}$ = undinid

$\frac{n}{0}$ = undi

hence $T(n) \in \Theta(n)$ ⊜

Big Theota.

*Algorithm : Unique Element (A[0.... n-1])

// Determine whether all the elements in a given array are distinct (True or False

// Input: An array A[0... n-1] → pos

//Output: Return "true" if all elements in Array are distinct.

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[j] == A[i]
            return false
return true;
```

i j

10, 10, 20, 30, 40, 50 (n-1)

10, 20, 30, 40, 50, 60

10, 20, 30, 40, 50, 50,

Ans

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if (A[i] == A[j])
```

n-2 n-1

n-1

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

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

$$\cancel{= n-1-0+1}$$

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

$$= n \cancel{-1} - i \cancel{-1} + 1$$

$$= n - i + 1$$

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

$n-2-0$

$(n-2)-0 * n-1-1$

$(n-2)-$

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

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

$T(n) \approx n^2$

Hence $T(n) \in \Theta(n^2)$

- Theora Θ
- oh Ω

$2 \quad 3$
$\rightarrow$
$5 \quad 6$

$7 \quad 8$
$\downarrow$
$5 \quad 8$

$(2\times7)+(3\times5) \quad (2\times8)+(3\times8)$

$(5\times7)+(5\times5) \quad (5\times8)+(6\times8)$

$$\begin{bmatrix} 2 & 3 & 5 \\ 4 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 2 & 5 \\ 7 & 9 \\ 8 & 4 \end{bmatrix}$$

Multiplication

Algorithm || ~~Sum~~ of 2D Matrix

```
for i ← 0 to n-1 do                          → n+1
    for j ← 0 to n-1 do                      → n(n+1)
        c[i,j] ← 0.0                         → (n×n)×1
        for k ← 0 to n-1 do                  → (n×n)×(n+1)
            c[i,j] ← c[j,j] + A[i,k] * B[k,j]  → (n×n×n)×1
```

$f=1$, $f \leftarrow 1$, $f:1$ } these are all ~~same~~ used to assigment operator

n^2

$n^3 + n^2$

n^3

-> Base Algorithm

$f(n) = 2n^3 + 3n^2 + 2n + 1$

$\Theta(n^3)$

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

$= n-1 + 0 + 1$

$= n \times n \times n$

$T(n) = n^3$

$= \Omega(n^3)$

```
for (i=1; i<n; i=i*2)
{
    sramant;
}
```

1

$1 \times 2 = 2$

$2 \times 2 = 4$

$2^2 \times 2 = 8$

$2^3 \times 2 = 16$

$\vdots$

2^k

assume $i >= n$

$2^k = n$

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

```
for (i=1; i<=n; i++)
{
    stmt;
}
```

$i = 1$

$i++ = 2$

$- = 3$

$\vdots$

n

$k = n$

```
for (i=0; i>=1; i=i/2)
{
    Statment;
}
```

$i = n$

$n/2$

$n/2^2$

$n/2^3$

$n/2^4$

$\vdots$

$\boxed{n/2^k < 1}$

Assume

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

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

$n = 2^k$

$\boxed{k = \log_2^n}$

$\frac{A}{B} = 1$

$k = 3$

Plan for Analysis of Recursive Algorithm.

* Deciding on a parameter indicating on input Size
* Identify the basic Algorithm
* Check whether the no. of times the basic operation

* Setup a recurrence relation with an appropriate initial condision expressing the no. of. times the basic operation is executed
* Solve the recurrence (or at the very least establish its solution order of growth) by backward substitutions or another method.

Factorial Algorithm

fact(4)

```
if (4 == 1)
    return (1);
else ~~fact(4-1) * 4~~
    return fact(4-1)*4;
```

```
fact(a)
    if a == 1
        return 1
    else
        fact(a-1) * a
```

→ fact(4) {
```
if(4==1)
    return(1);
else
    return (4-1)*4 --> will return 24
```
fact

→ fact(3);
```
if 3==1;
    return 1
else return fact(3-1) * 3
```
2

fact(2)
```
if(2==1)
    return 1;
else return fact(2-1)*2
```
1

fact(1)
```
if(1==1)
    return 1
else
    return fact(1-1)*1
```

Definition - $1 * 2 \ldots * (n-1) * n$ for $n \geq 1$ and $0! = 1$

$n = 10$ $(n-0)(n-1)(n-2)(n-3)(n-4)(n-5)(n-6)(n-7) \ldots (n-9)$

Recursive Defination. $n!$: $F(n) = F(n-1) * n$ for $n \geq 1$ and Algorithm

// Compute $n!$ recursively

// Input : A non-negative integer n

// Output : The value of $n!$

if $n = 0$ return

else

return $F(n-1) \times n$

Analysis of Factorial (n)

Input parameter is given number (n)

Basic operation is multiplication $M(n)$

Framing Recursive Relation.

$F(n) = F(n-1) \times n \quad n > 0$

$M(n) = M(n-1) + 1 \quad n \geq 0$ → Time for basic operation

$M(n) = M(n-1) + 1 \quad n > 0$

computing multiplying

$F(n-1) \times F(n)$ within

By applying Backword Substitution method $M(n) = M(n-1) + 1$

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

$= [M(n-2) + 1] + 1$

$= M(n-2) + 1$

$= [M(n-3) + 1] + 2$

$= M(n-3) + 3$

$\vdots$

Generaly $M(n) = M(n-i) + i$ — (1)

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

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

$= M(n-2) + 1$

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

$= M(n-3) + 1$

Taking the benifit of initial condision if we sub $i=n$ in equati ①
~~thus we get~~

$M(0) = 0$ if $n=0$

No. of mutiplication is required

Substitute $i=n$ in equation - ① we get

$$m(n) = m(n-n) + n$$
$$= 0 + n \approx n$$

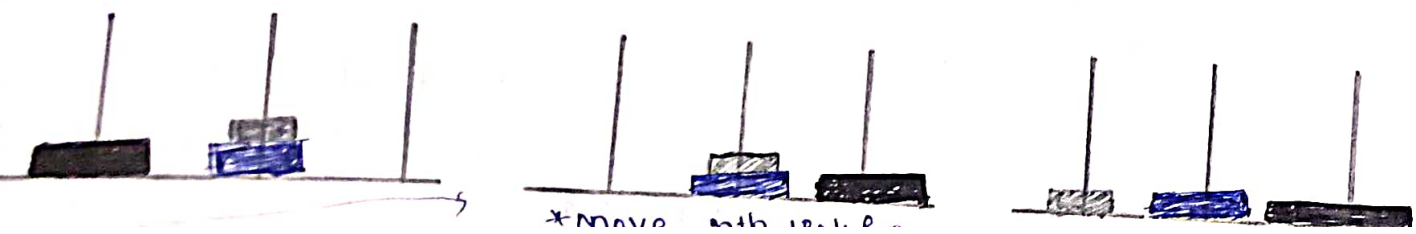
Hence the time efficiency of factorial of a number is "n"

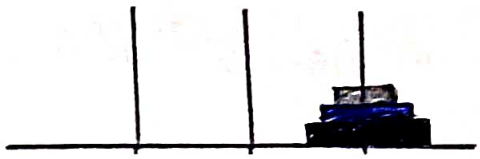
Recurrence tree for Factorial

Fact(n)
↙
Fact(n-1)
↓
Fact(n-2)
↓
Fact(n-3)
⋮
Fact(n-n) → fact(0)

ower of Hanoi

* move n-1 disks from A to B using C





n = No. of disc

* Move $(\overset{3-1}{n-1})$ disk from A to B using C
  2
* Move nth disc from A to C
* Move $(n-1)$ disc from B to C using A
  3-1

TOH(3, A, B, C)

A — C

A, C, B

A C B

Analysis of Hanoi

Input size = n disc

Basic operation: is movement of disc from one pole to another pole.

$$M(n) = \underbrace{M(n-1)}_{\text{Moving } (n-1) \text{ disc from A to B using C}} + \underbrace{1}_{\text{Moving nth disc from A to C.}} + \underbrace{M(n-1)}_{\text{Moving } (n+1) \text{ disc from b to c using A.}}$$

for $n > n_1$

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

Replacing n by $n-1$

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

$$= [2M(n-2) +$$

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

$$= 2[2M(n-2) + 1] + 1$$

$$= 2^2[M(n-3) + 1] + 2 + 1$$

$$= 2^2[(2M(n-4) + 1) + 1] + 3$$

$$= 2^3[M(n-4) + 1] +$$

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

Replacing $n =$

$$M(n-3) = 2M(n-3-1)$$

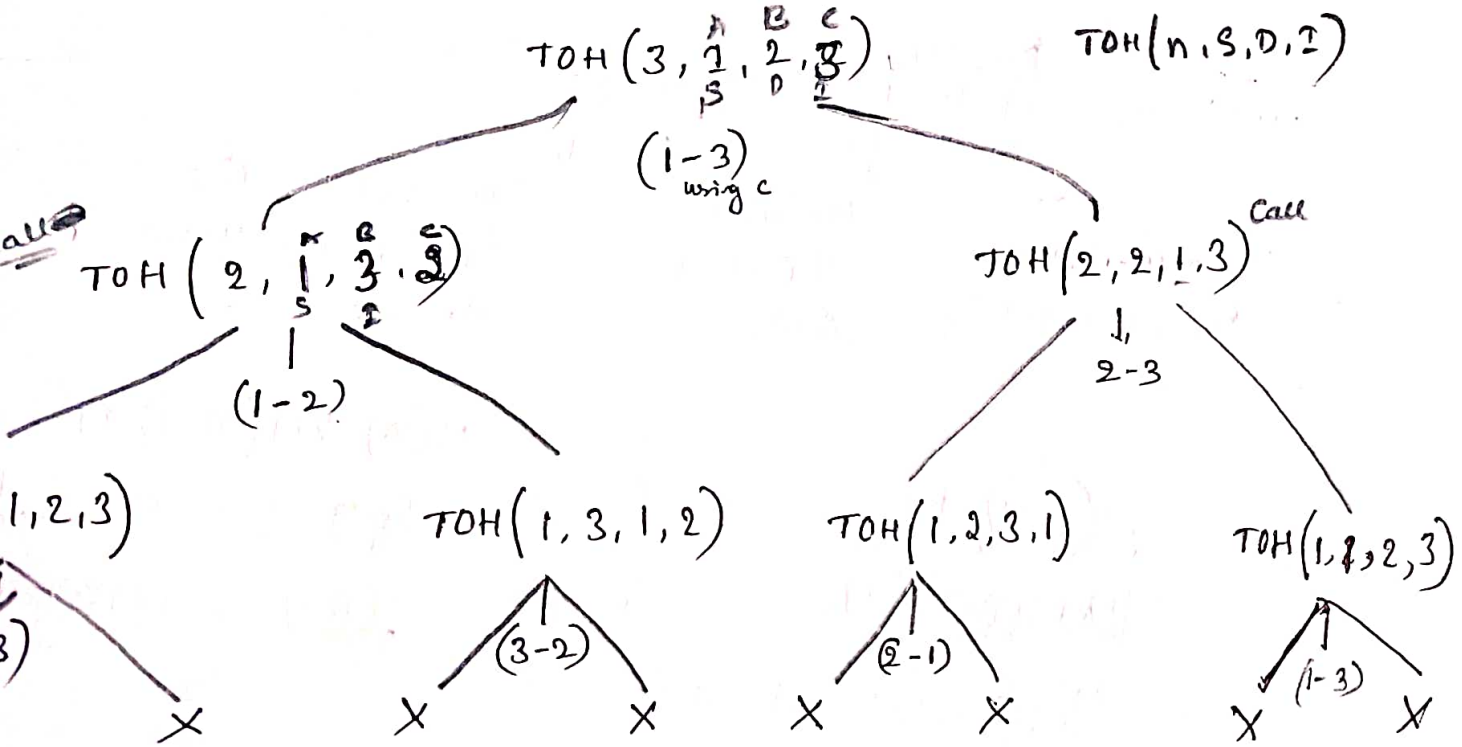
$$= 2M(n-4) -$$

Substituting $i = n-1$ in eqn ①

$$m(n) = 2^{n-1} m(n-n+3) + 2^{n-1} - 1$$

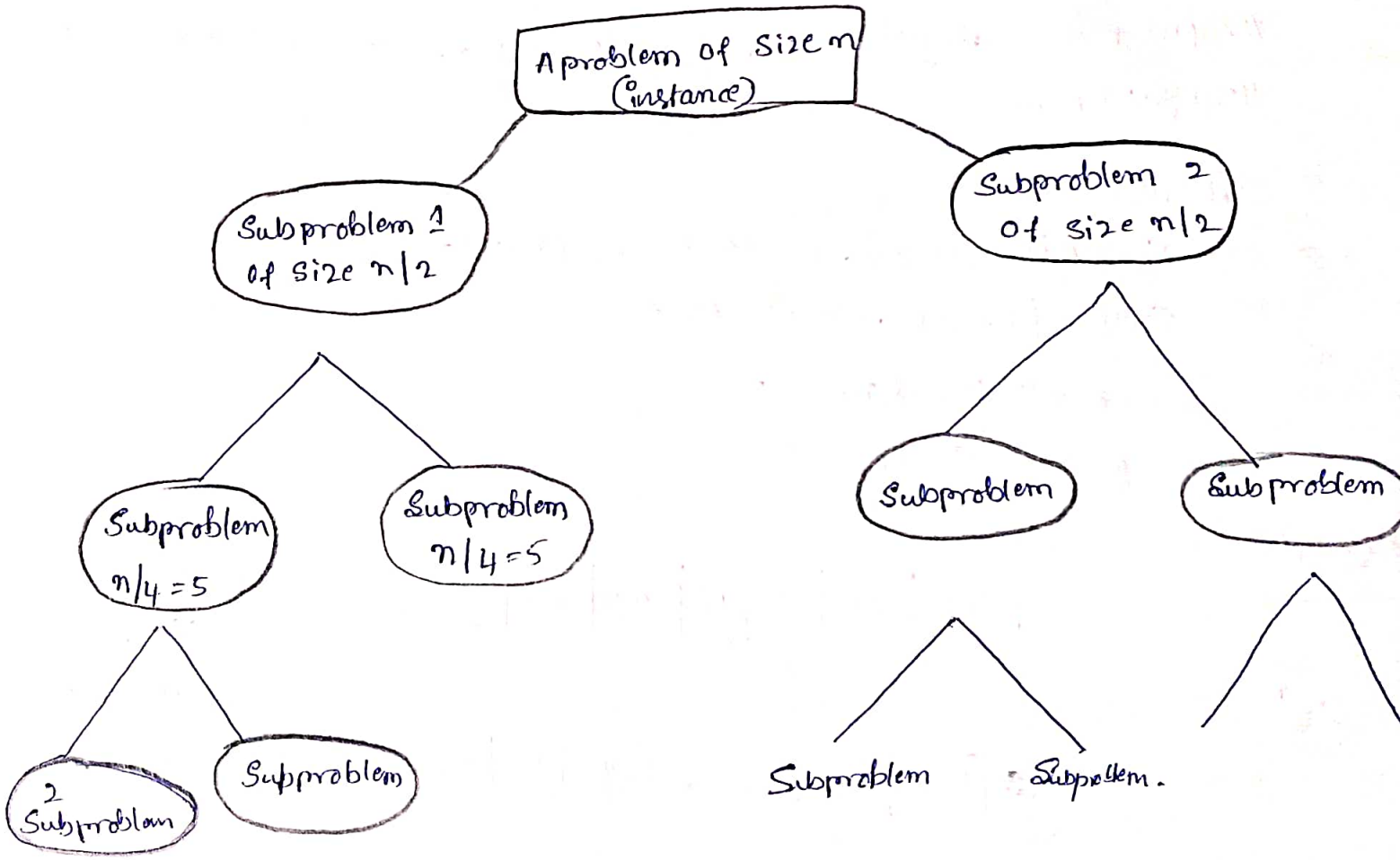
$$=$$

ecurenc Tree for Tower of Hanoi



Devide - and Conquer

1) Devide the instance of a problem into two or more smaller instances
2) Solve smaller instances recursively
3) Obtain solution to original (larger) instance by combining these solution



Upto - 1

Divide - and - Conquer Example

* Sorting
* Binary tree traversal
* Binary Search.
* Multiplication of large integers.
* Matrix Multiplication : Strassen's algorithm
* Closest - Pair and Convex - hull Algorithm

Mergesort

Algorithm Mergesort[A[0....n-1])

//Sort array A[0...n-1] by recursive Mergesort

//Input: An ~~algorithm~~ array A[0..n] of orderable element

//Output: Array C

if n>1,

Copy A[0...⌊n/2⌋-1] to B[0..⌊n/2⌋-1]

Copy A[⌊n/2⌋ .. n-1] to C

Mergesort(B..⌊n/2⌋ -1

mergesort (C..n/2) -1

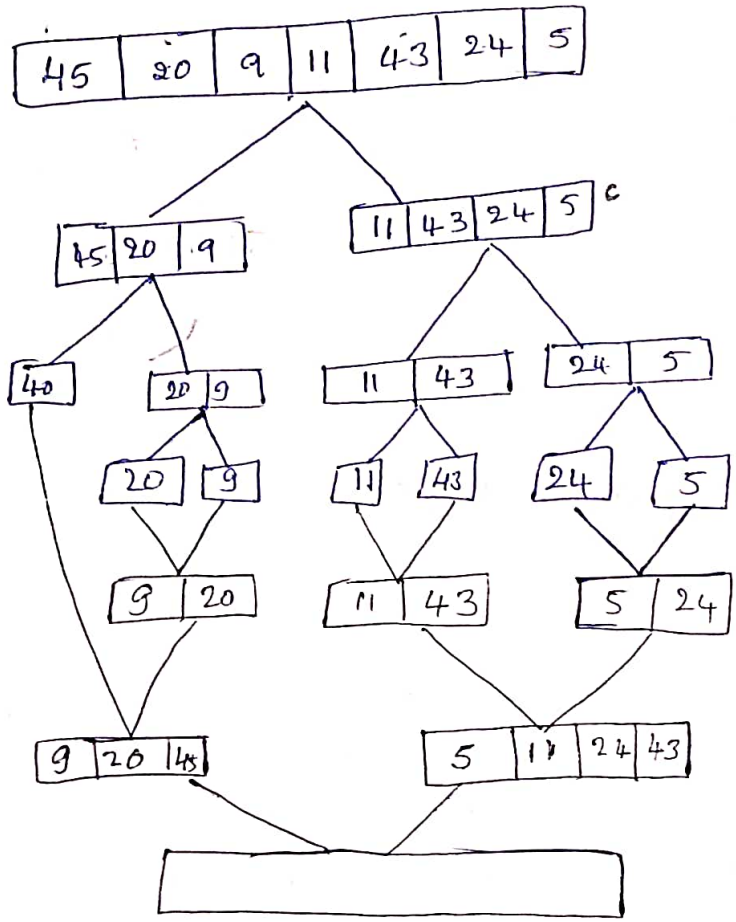
Example

$n = 7$ odd.

$\frac{7}{2} = 3.5$

floor of 3.5 = 3

Left side



T(1) = 0

Tracing should include how the conquier and combile is done with the help of algorithm.

Analysis of Merge Sort

All the cases have same time efficiency in case of merge sord let us assume that no. of items in the array are integral multiple of $n = 2^k$ for $k \geq 0$

$2 \times \frac{n}{2}$ cancel

The recurrency is $T(n) =$

if $n = 1$

$$T(n) = \begin{cases} 0 \\ T(n/2) + T(n/2) + n \end{cases}$$

$n-1$

$T(n/2)$ → Time required for sorting left part

$T(n/2)$ → Time required for sorting right part

n → Time required to devide merge one istance of a soln with n element

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

$$T(n) = 2T(n/2) + n \quad \frac{n}{2} \Big| \frac{n}{2}$$

Replace n by $n/2$

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

Masters Theorem Apply

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

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

$(2\times)+ 4\times 3 5 4-1$

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

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

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

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

Replace $n = n/2$

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

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

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

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

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

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

Addition bracket not consider

Taking the benifit of initial condition = 0

let $2^i = n \longrightarrow (1)$

$n = T(1) = 0$?

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

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

$$= n \times (1) + n.i = nT + nT$$

$$= n.i \longrightarrow \text{equation } (2)$$

From equn (1) $2^i = n$ Taking log on both side we get $i.\log_2^2 =$

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

$$i = \log_2 n \longrightarrow (3)$$

* Substitute the ~~log~~ value i from equn (3) in equation (2)

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

Properties of Sorting Algorithm

1) Stable Property : Relative positions of the elements of the array

2) In-place ~~Spac~~ Property : Space taken by algorithm.

Merge Sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

Process of Merge Sort is to divid the array into two ~~halves~~ halfs

$\frac{1}{8^1} = 8^{-1}$ $\quad 8^1 = \frac{1}{8^1} = 0.125$

$\frac{1}{8^1} = 1.8^{-1} = 8^{-1}$

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

$\frac{1}{8^1} = 1.8^{-1} = 8^{-1}$

$\frac{1}{2^3} = 1.2^{-3} = 2^{-3}$

Eg $T(n) = \underset{\geq 1}{8}\, T\left(\frac{n}{\underset{>1}{2}}\right) + n^2$

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

$= n^{\log_2^8}$

$= n^3\,[u(n)]$

$= n^3 \cdot 1$

$T(n) = n^3$

$h(n) = \frac{f(n)}{n^3} = \frac{n^2}{n^3} = \frac{1}{n} = n^{-1} = r$

Master Method

$T(n) = \underset{\geq 1}{a}\, T\left(\frac{n}{\underset{>1}{b}}\right) + f(n)$

$a \geq 1 \qquad b > 1$

Sol $\boxed{T(n) = n^{\log_b^a}\,[u(n)]}$

→ $u(n)$ depends on $h(n)$

→ $h(n) = \frac{f(n)}{n^{\log_b^a}}$

→ Relation between $h(n)$ and $u(n)$

1) $T(n) = T\left(\frac{n-1}{1}\right) + n$ Not allowed

2) $T(n) = 8\,T\left(\frac{n-1}{2}\right) + n$ allowed

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

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

Master Theorum

In Most Critical Cases of divide and Conquer an Instance of size n can be divided into B instance of size $\frac{n}{b}$ with 'a' instances of n need to be Sort.

The devide and Conquere recurrence relation is

$$\boxed{T(n) = aT(n/b) + f(n)}$$ where $f(n) \in \Theta(n^d)$ $d \geq 0$

a = constant 1 or >1

T = b>1

* If $a < b^d$, $T(n) \in \Theta(n^d)$
* If $a = b^d$, $T(n) \in \Theta(n^d \log n)$
* If $a > b^d$, $T(n) \in \Theta(n \log_b^a)$

Where 'a' and 'b' are the +ve constances such that $b > 1$, d is the power of n in $f(n)$

Time complexity analysis of using Masters Theorum

Pearrence Relation for merge sort is $2(T(n/2) + n$

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

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

from general recurrence relation of we get

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

$d = 1$

$\boxed{a = 2, \quad b^d = 2^1 = 2}$ is equal to b^d.

The time complexity of merge sort is $\Theta(n \log n)$

***

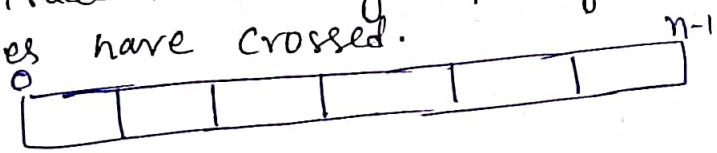
* Write a Master theorum, Explain the Devide and Conquer Algorithm and also explain Backwood Substitution Method.

QUICK SORT or Partision Exchange Sort

0	1	2	3	4	5
5	4	5	10	12	12

1) Select any element as pivotal
2) Comparing remain element with a pivotal
3) partitioning in the given array will result into Smaller elements move to left Part of the pivotal & Large element move to right part of the ~~pivotal~~. Selected pivotal element

There are 3 Situations araising Depending on Whether or NOT Scanning indices have crossed.



i = low

j = hegut j = i+1

* 2 Scaning indices i & j are needed and i is for left Scaning and "j" is for right Scanning.

Case 1

If Scanning indices i & j have not crossed i<j then exchange a[i] and a[j] and resume the Scanning

Resume the Sence by increame-nting "i" and "j" respectivly

P	all are ≤ P	≥ P	.....	≤ P	all ≥ P

Case 2

If i > j the we have to exchange with A[j] element.

P	all are < P	≤ P	≥ P	all are > P

Case 3.

i = j in this c

Algorithm For Quick Sort

```
//Sort a Subarray by Quicksort
//Input: Subarray of array A[0...n-1] defined by its left and right
// Indices l and r
//Output A[l...r] Sorted in non decreasing ordered
if l<r
    s ← partition (A[l..r]) //s is a split position
    Quicksort (A [l...s-1])
    Quicksort (A [s+1...r])
```

Method

Working And tracing of Quicksort.

Given array.

i 0	1	2	3	4	5	6	7
15	13	11	19	18	12	14	17

Greater | =

true → move
False → stop

15 13 11 19 18 12 14 17
pivotal, L+ startpoint, A>15, swap, Tend point

l > r
stop

15 13 11 14 18 12 19 17
pivotal, L+, Start point, swap, Tendpoint

≥
Lesstha

15 13 11 14 12 18 19 17
Tst, Tend

12 13 11 14 15 18 19 17
pivotal, go

12 13 13 14
Piv

11 12 13 14

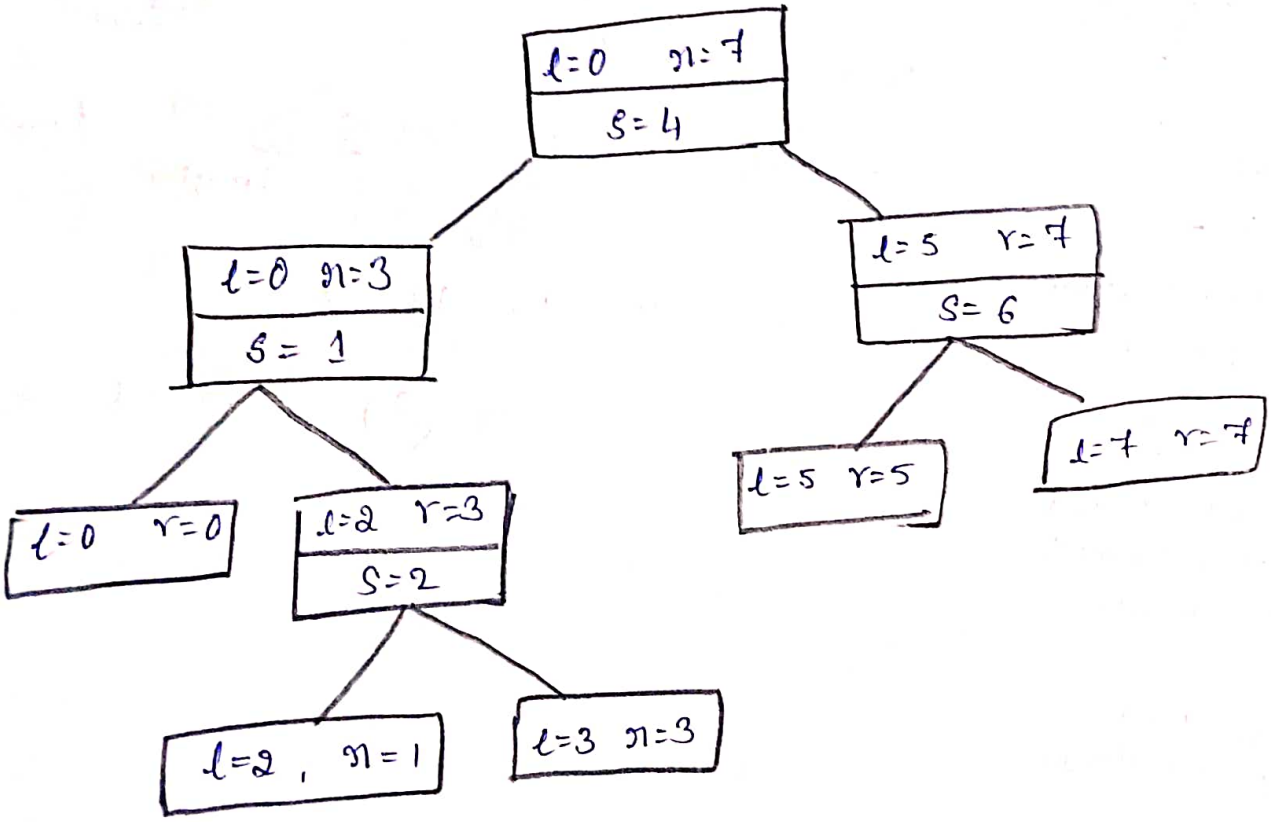
11 12 13 14

* Start variable crossed end variable then we are not going to cross these element. then we have to cross with ~~Start~~ end element & pivote element.

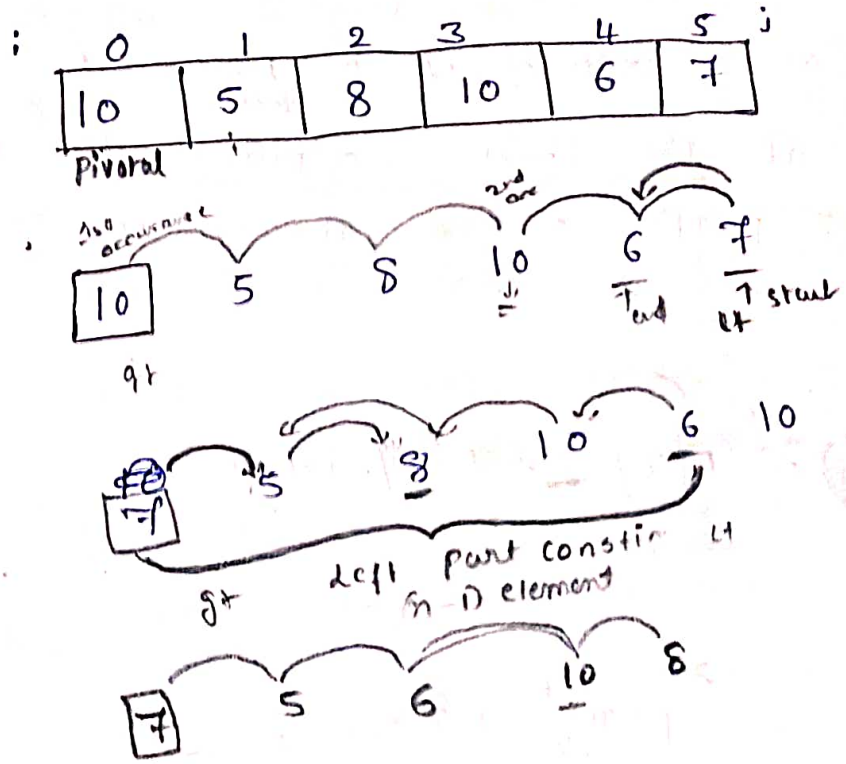
12 18
↳ crossed with pivote element

$l=0 \quad r=7$

Recursive Tree for quicksort



Example :



1st occurrance of 10

2nd occurrance of 10

Before Sorting $i=0$ $\left\{ \begin{matrix} i<j \\ i>j \end{matrix} \right\}$ $j=3$ $j=4$

After sorting $i=5$

Relative order is not maintained

Hence Quick Sort is not Stable in nature

Quick Sort

Best Case

Best case occurce if the pivot element choosen is placed Exactly at the center dividing the array into to equal parts so the recurrence relation that will be

$$T(n) = \underbrace{T(n/2)}_{\text{Sorting left part}} + \underbrace{T(n/2)}_{\text{Sorting right part}} + \underbrace{n}_{\text{Time Required for partioning array into subparts}}$$

$$= 2T(n/2) + n$$ // For Substitution Refer merge sort.

Now Refer to the merge Sort techique (go back that Page) and find the best Case

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

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

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

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

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

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

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

replace n by n/2

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

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

$$=$$

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

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

$$=$$

Worst Case

The worst case occurce when at each invocation of the procedure the current array is partiond into 2 sub array with one of them being empty the array can be considered as either left skew or Right skew.

In general If the array has the 'n' elements the in worst case situation after partioning (n-1) element will be moved towards 1 part of the array and zero element in another part, vise versa.

The recurrence relation is

worst case

$$\boxed{T(n) = \{T(0) + T(n-1) + 1\}}$$

if $n = 1$
otherwise.

Considering

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

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

$$= \frac{n(n+1)}{2} \approx \underbrace{n^2}_{g(n)}$$

$f(n)$

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

Replacing n with (n-1)

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

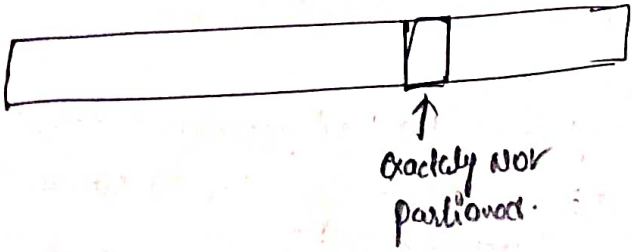
$T(n-2) = T(n-3) + (n-2)$ *
Replacing n with n-1

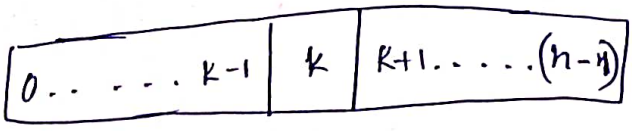
3+2+1 | $3^2 = 3$

$\boxed{6 = 9}$

Average Case

In Average case the given array may not be exactly partisioned into 2 sub arrays or may not the skewed arrays.





$l = 0$

$r = k-1$

$= u - l + 1$

$= k \cancel{-1} - 0 \cancel{+1}$

$= T(k) \rightarrow (1)$

$l = k+1$

$r = n-1$

$= u - l + 1$

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

$= n-1-k \cancel{-1} \cancel{+1}$

$= n-k-1$

$T(n-1-k) \rightarrow (2)$

The recurrence relation is

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ T(k) + T(n-1-k) + n & \text{otherwise} \end{cases}$$

Quick Sort Example 3.

	0	1	2	3	4	5	6
A	10	15	1	2	9	16	11

Pivotal

Partioning is the main part

After Partition

Partion 1 <	Pivote Element	> Partion 2

We can not guarantee that this partion is ordered. But ~~these~~ value are < pivote element

↓

again same fanda to this sub arrays

No need to sort this after partioning

We can not guarantee that this partion is order But the values are > pivotal

0	1	2	3	4	5	6.
10	15	1	2	9	16	11

Example 4

start, end

	0	1	2	3	4	5	6	7	8
	7	6	10	5	9	2	1	15	7

rt - pivore element

Lt pivore element

pivotal →

end (−)

	7	6	1	5	9	2	10	15	7

(+) start

all the element Lt 7 would be the left side & all the elements Rt 7 would be right side

start, end

	7	6	1	5	2	9	10	15	7

end, start

end, start

Swap a[lb] to a[end]

	2	6	1	5	7	9	10	15	7

Exactly middle

Partition Method

```
while (start < end)
{
  Partition(A, lb, ub)
  {
    pivotal = A[lb]
    Start = lb
    end = ub
    while(a[start] <= pivotal)
      {
        Start++;
      }
    while(a[end] > pivotal)
      {
        end--
      }
    if (Start < end)
        Swap(A[start], A[end])
  }
else
    Swap (a[lb], a[end])
    return end.
```

Code

```
QuickSort(A, lb, ub)
    if (lb < ub)
    loc = Partition(A, lb, ub)
        Quicksort(A, lb, loc-1)
        Quicksort(A, loc+1, ub);
```

BINARY SEARCH

BINARY SEARCH

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	67	89

Condition

* The array should be sorted
* If array is not sorted the we can't apply Binary Search.
* recursively Divide the array into two parts.

data = 59

Cases - 3

~~Cases : data == a[mid]~~

1) data == a[mid] return
2) data < a[mid] move r to
3) data > a[mid] move l to

25 < 59

Index condition

if (l > r) then data is not present

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	67	89

↑l → Move to (mid+1)

↑ ↑r

partition $\left[\frac{l+r}{2}\right]$

$= \frac{0+9}{2}$

$= 4.5$

Mind

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	67	89

l ↑mid r

data > a[mid]

Move l = mid+1

5	6	7	8	9
45	59	63	67	89

$= \frac{l+r}{2}$

$= \frac{5+9}{2}$

$= 7$

data < 63 a[mid]

(r = mid - 1)

5	6	7	8	9
45	59	63	67	89

↑l ↑r ↑mid

Mid

$= \frac{5+6}{2}$

$= 5.5$

data 59 > 45 the move l = mid+1 = 5+1 = 6 then

always consider (low) 4.5 = 4 = 7 6

6	7	8	9
45	59	63	67	89

↑l ↑ ↑r

Mid

data == a[mid]

return

if data != mid then l = mid + 1 = 6 + 1 = 7 → l > r → given data is not exist

Binary Search (A, n, data) { → Size of Array, → Given data to Search, → Array pointer

```
    l = 0;
    r = n-1;
    mid = l + r / 2;
  while (l<r) {
      if (data == a[mid])
        { return mid;
      else
      if (data < a[mid])
        r = mid - 1;
      else
        l = mid + 1;
  }
  return -1; // if l > r
}
```

repeating upto $l < r$

Binary Search algorithm can be implemented in two ways.
1) Iterative 2) Recursive Method – follows the divide and conquere

Steps :

1) The array in which searching is to be performed

3	4	5	6	7	8	9

→ Let $\boxed{X = 4}$ be the element to be searched

2) Set two pointers low and high as the lowest and the highest position respectively

3	4	5	6	7	8	9

↑ low ↑ high

$\boxed{arr[(low+high)/2]} = 6$

5) if $\boxed{x > mid}$ compare ~~elements~~ x with the middle element ~~on the~~ ~~searched with m.~~ right side of $\boxed{mid}$ This is done by setting low to $\boxed{low = mid+1}$

6) Else, $\boxed{\text{Compare } X}$ with the middle elements of the elements on the left side of mid, This is done by setting low to $\boxed{low = mid+1}$

* Binary Search is a Searching algorithms for finding an elements position in a Sorted array.

* In this approach, the element is always searched in the middle of the portion of an array.

* Binary Search Algorithm can be implemented in two ways which are dicussed below

  * Iterative Method
  * recursively method.

* The recursive Method follows the Divide and Conquer approach.

20 Marks

Algorithm

~~Binary Search(0... n-1~~

binarySearch (arr[0...n-1], x, low, high)
//search the key element from given sorted array.
// Input : The Sorted array and key element is x
//output: The element is found or not.

```
if low > high
    return False;
else
    mid = (low+high)/2 → ceil value
if x == arr[mid]
    return mid;
else if x > arr[mid]    // then x is on the right side.
    return binarySearch(arr, x, mid+1, high);
    else    //x is on the left side.
    return binarySearch(arr, x, low, mid-1);
```

0	1	2	3	4	5	6	7	8	9
4	10	16	24	32	46	76	112	114	182

Searching key 112

low 4 10 16 24 32 46 76 112 114 182 high

mid (3)

mid = $\frac{0+9}{2}$ 112 > 32

$= \lfloor 4.5 \rfloor$ Consider Right half part

$\boxed{= 4}$ low = mid+1 = 4+1 = 5

5 6 7 8 9

4 10 16 24 32 46 76 112 114 182

low high

$= \frac{l+h}{2} = \frac{5+9}{2} = \frac{14}{2} = 7$

key == ar(mid)

Key = arr[mid] hence key element 112 is found at index position 7.

Best Case Situation: The best accures the item to be search is present in the middle of array. The no. of comparisions required will be on 1 hence the time complexit in best case is $\Omega(1)$

low = 0 high = 9
mid = 4

low = 5 high
mid = 7.

Worst Case: occurs when the key elements to either at the 1st position or at the last position. So the maximum no. of comparisons and the time complexity is given by.

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

$1 \rightarrow$ for comparison

$T(n/2) \rightarrow$ Searching either Left/Right part.

$1 \rightarrow$ Comparting key with middle.

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

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

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

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

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

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

$\rightarrow$ Taking benifit of the initial condision

let $\boxed{2^i = n}$ becaus $\frac{n}{n} = 1$

Taking log on both size

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

$i = \log(n)$

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

$$= T(1) + k$$

$$= 1 + \log(n)$$

$$T(n) \approx \log(n)$$

DECREASE AND CONQUER.

The process of reducing a problem to a subproblem and solving it one step at a time is the core concept of the decrease-and-conquer algorithm.

Problem is transformed into a smaller subproblem and then that subproblem is solved first.

1) Reduce problem instance to smaller instance of $n-1$, $n/2$ etc of the same problem

Eg: Binary Search.

2) Conquer or solve smaller instances

3) Extend solution of smaller instances to obtain the solution to original instances

4) Can be implement either top-down (recursive) or bottom-up (Non-Recursive).

5) Also reffered to as inductive or incremental approach

6) The Three variations of decrease and conquere are
   * Decrease by a constance
   * Decrease by a factor
   * Variable size decrese.

Insertion Algorithm

Algorithm: Insertion sort (A[0... n-1])

// Sort a given array by insertion sort.

// Input: An array [0...n-1] of n orderable elements

// Output: Array A[0... n-1] sorted in nondecending order.

Insertion Sort

Insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card game, ~~and then~~ If the unsorted card is greater than the the first card, It will be placed at the right side; otherwise, It will be placed at the left side. Similarly, all unsorted cards are taken and put in thier exact place.

Steps - If the Element of ~~the~~ is the first Element, assume that it is already sorted. Return 1.

Step 2 - Pick the next element and store it separately in a key

Step 3 - Now compare key with all the Elements in the sorted array

Step 4 - If the element in the sorted is smaller than the current Element, then move to the next Element, Else. Shift greater elements in the array towards the right

Step 5 - Instert the value

Step 6 - Repeat until the array is sorted.

JTP

12	31	25	8	32	17

Initially first two elements are compared

31 is greater than 12 → Already sorted in ascending order

12	31	25	8	32	17

Move to Next two elements

25 is greter than 31 → Not Sorted

25 is smaller than 31

Now swap with 31 and 25. Along with swapping, insertion sort also check it with all elements in the sorted array.

12	25	31	8	32	17

Next Check

Now two element sorted in the array Now move forward to next elements that are 31 & 8.

is 8 ≥ 31 → Not Sorted

Swap

12	25	8	31	32	17

is 8 ≥ 25 - Not Sorted

Swap

12	8	25	31	32	17

is 8 ≥ 12 - Not Sorted

Swap

8	12	25	31	32	17

Sorted Unsorted

32 ≥ 31 - yes already sorted move to next element

8	12	25	31	32	17

is 17 ≥ 32 – Not Sorted.

8	12	25	31	17	32

17 ≥ 31 → Not Sorted.

8	12	25	17	31	32

17 ≥ 25 → Not Sorted.

8	12	17	25	31	32

Time Complexity.

1) Best Case → Partially Sorted.

$O(n)$

2) Avrage Case → Elements are in jumbled order

$O(n^2)$

3) Worst Case → Elements are in Decending order.

$O(n^2)$

Working of Insertion Sort.

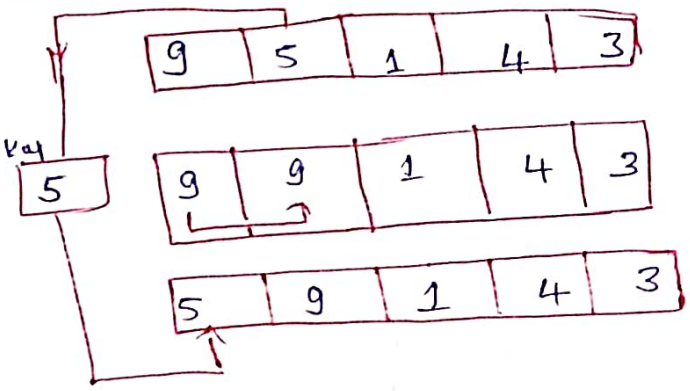
Suppose we need to sort the following array.

9	5	1	4	3

1. The first element in the array is assumed to be stored. Take the second it separately in $\boxed{\text{Key}}$.

Compare Key with the first element. If the first element is greterthan Key, then Key is placed in front of the first element.

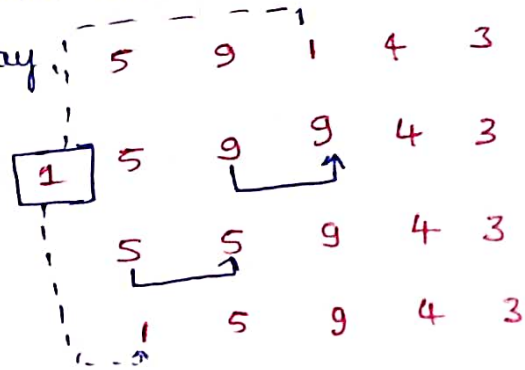
Step=1



* If the first element is greter than Key, then Key is placed in front of the element.

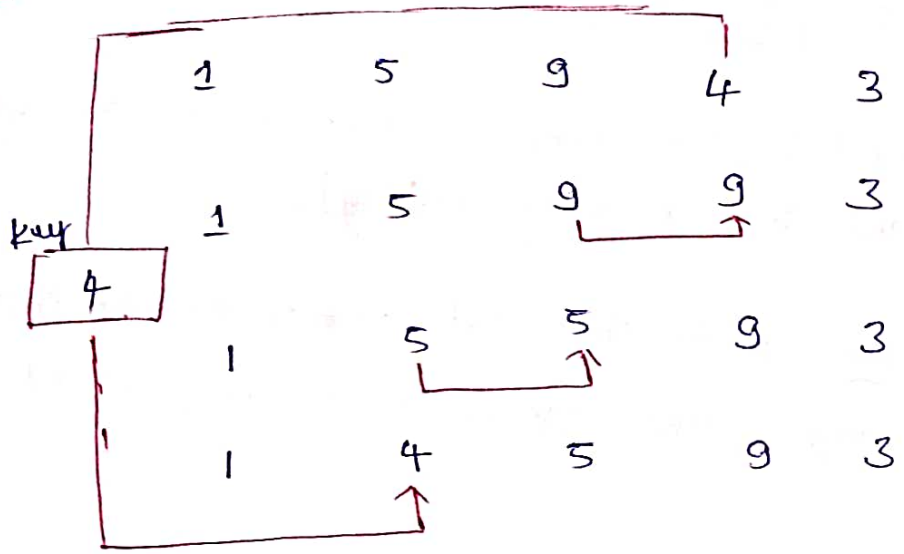
2. Now, the first two elements are stored.

Take the third element and compare it with the elements on the left of it. Placed it just behind the element smalled than it. if there is no element smaller than it. then place it at the begging of the array.



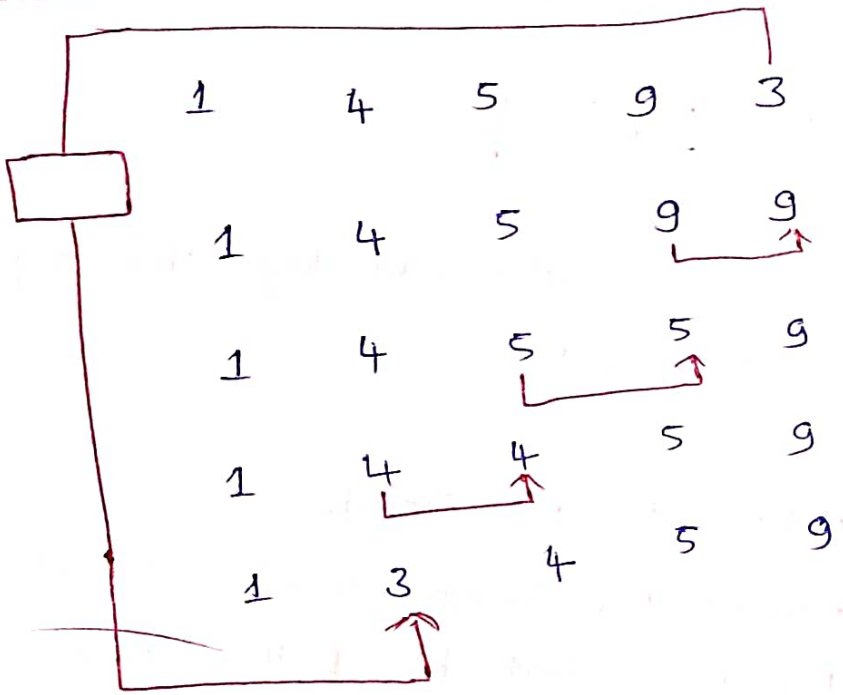
3. Similarly, place every unsorted element at its correct position.

Step-3



place 4 behind 1

Step -4



→ place 3 behind 1 and the array is Sorted

```
// Insertion sort in c
# include <stdio.h>

// function to print an array
void printArray (int array[], int size)
{
    for (int i=0; i< size; i++)
    {
        print("%d", array [i]);
    }
    print("\n");
}
void insertionSort(int array[], int size)
{
    for (int step=1; step <size ; step++)
    {
        int key = array [step];
        int j = step-1;
        // compare key with each element on the left of it untile an
          element smaller than
        // it is found.
        // for descending order, change key<array [j] to key> array[j]
        while (key < array [j] && j>=0)
        {
            array [j+1] = array [j];
        }
    }
    // Driver code
    int main() {
        int data[] = {9, 5, 1, 4, 3};
        int size = sizeof(data) / sizeof(data[0]);
        insertionSort (data, size);
        printf("Sorted array in ascending order:\n");
        printArray (data, size); }
```

Apply insertion sort to the given data set

Class Notes.

Array = 25 75 40 10 20

0	1	2	3	4
25	75	40	10	20

first element is sorted

25	75	40	10	20

Sorted

25	75	40	10	20

25	40	75	10	20

Sorted

10	25	40	75	20

sorted

Best Case The Best Case accurs when the items in the list are partial or nearly sorted.

the Conditional Expression $A[J] > V$ is executed for $n-1$ no. of times. So the order of growth is in denier

$\boxed{n-1 < n}$

Worst Case: The worst Case occurs when the Condision expression $a[j] > V$ is excuted for maximum no. of times. this situation occures when element of the sorted in reverse order.

for $i \leftarrow 1$ to $n-1$ do

tem $v = A[i]$ -

$j = i-1$

while($j >= 0$ and $A[j] > v$) do

$\vdots$

$$\sum_{i=1}^{n-1} \qquad \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=\phi}^{n-1} (i-\cancel{1}-0+\cancel{1}) = i-\cancel{1}+0+\cancel{1}$$

$$= \sum_{i=1}^{n-1} i \qquad n-1+1+1$$

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

$$= \frac{n(n-1)}{2} \longrightarrow \text{Sum of natural number.}$$

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

$T(n) \approx n$

The time effiency of insertion sort include.

Stable

Q M
P M2

5 15 75 30 15 10

already sorted.

5 15 75 30 15 10

5 10 15 75 30 15

5 10 15 75 30

Sorted

5 10 15 75 30

Sorted

5 10 15 15 30 75

Sorted.

Sorted & unsorted.

Jony's

Sorted Sublist

0	1	2	3	4	5
5	4	10	1	6	2

always sorted

Start

unsorted Sublist

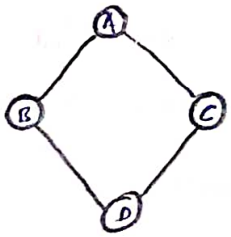
n=6

Graph Data Structure

* Directed Graph

	1	2	3
1	0	1	0
2	0	0	1
3	1	0	0

* Un Directed Graph



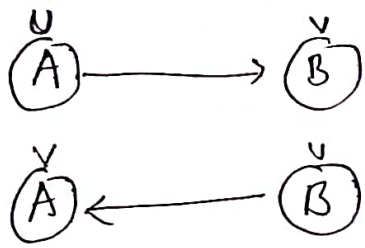
	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

Topological Order (Toprlogical Sorting)

TO is appliable for Directed graph ana acyclic graph only.

Build the graph - First

The Topological Sort of a Directed acyclic graph (DAG) $G = (V, E)$ is a linearing ordering all vertexes
Based on addjent
Such that for every edge (u, v) in a graph G, The Vertex "u" appear before "v" in the Ordering.

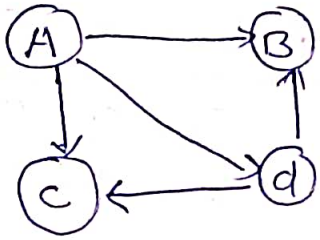


The Topological Sort of a graph can be viewed as ordering of vertices along the horizonal line so that all directed edges go from "left" to "Right"

Two ways to find out the TO

1) Source Removal Method.

2) DFS Method



Adjacency Matrix

	A	B	C	D
A	0	1	1	1
B	0	0	0	0
C	0	0	0	0
D	0	1	1	0

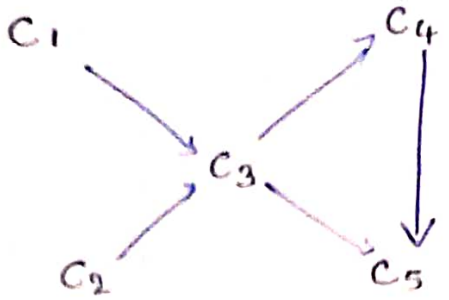
given the Matrix draw the graph or given the graph draw the Matrix.

Indegree → Incoming
outdegree → Outgoing.

* Source Removal Method.

Procedures

Source Removal Method.



Adjacency matrix

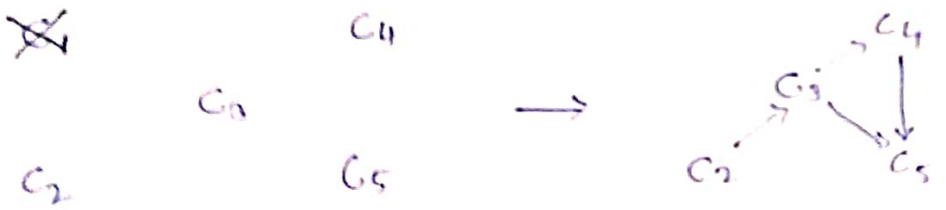
	C_1	C_2	C_3	C_4	C_5
C_1	0	0	1	0	0
C_2	0	0	1	0	0
C_3	0	0	0	1	1
C_4	0	0	0	0	1
C_5	0	0	0	0	0
Indegree	0	0	2	1	2

* For Indegree we check the column

Step 1 : Indegree of Vertices

$C_1 = 0$, $C_2 = 0$ $C_3 = 2$ $C_4 = 1$ $C_5 = 2$

As C_1 and C_2 Indegree 0 Choose any vertex to Source to be removed. Delete C_1 and its outgoing edges



Step2 Indegree for C_2

$C_2 = 0$ $C_3 = 1$ $C_4 = 1$ $C_5 = 2$

Delete

Step 3 : Indegree for C_3 C_4 C_5

$C_3 = 0$ $C_4 = 1$ $C_5 = 2$

$\cancel{C_3}$ C_4 C_5 $\longrightarrow$ $C_4 \to C_5$

Step 4 : Indegree for $C_4 \to C_5$

$C_4 = 0$ $C_5 = 1$

$\cancel{C_4} \to C_5$

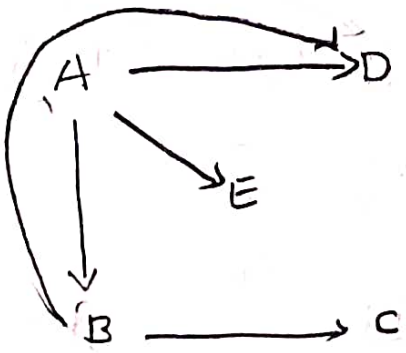
Step 5: Delete $\cancel{C_5}$

Topological order is ....

C_1 C_2 C_3 C_4 C_5

Not directly deleting C_3 to C_5 hence we show the dotted arrow mark.

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	1	1	0
C	1	1	0	0	0
D	0	1	0	1	1
E	1	0	0	0	0



Topic: Enhancing patients Experience and Efficiency of Health care service.

SDG: Good Health & well being.
Sustainable cities and communities

Empathy Stage

1) why patients are waiting for treatment too much time?
2) How can uneducated people get a good treatment?
3) How can we improve taking care of patients?
4) How do extended waiting periods affect patient?
5) what factors contribute most to patient impatience during Hospital visits?
6) what Strategies can Hospitals employ to reduce patient impatience?
7) what amenities or services would patients appreciate during waiting periods?
8) what role can technology play in managing patient impatience?
9) what are the problems for uneducated people in Hospital?
10) How can Hospitals address patients emotional needs during their Health care Journey.

Topological Order.

Jomics Lecture

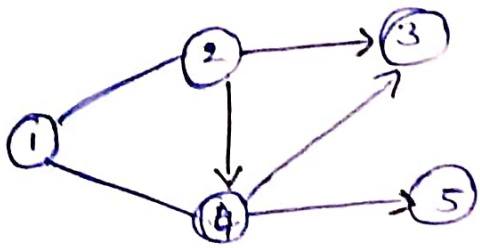
→ It is a linear ordering of its vertices such that for every directed edge uv for vertex u to v, u come before vertex v in the ordering

→ Graph should be DAG
→ every Directed Acyclic

$u \longrightarrow v$

↓

u come before vertex v

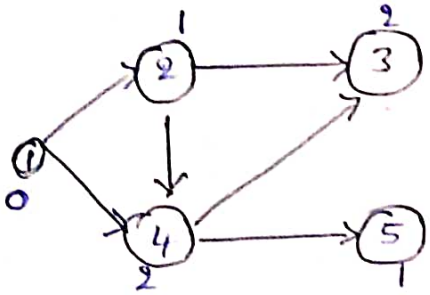


→ It is a directed graph every edges having some direction

→ This is Acyclic because only one direct is b/w the vertex

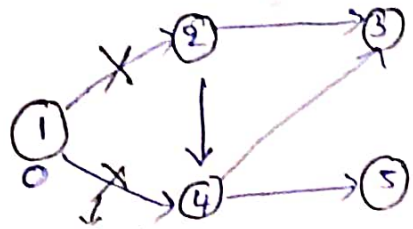
} Topology order is possible

Step 1 → Findout the Indegree of every vertex



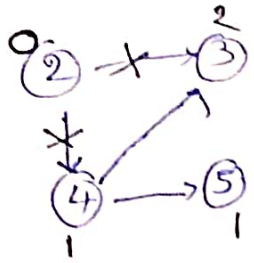
Step 2 → Select the vertex which is having '0' Indegree and delete the edges connected to them

I.

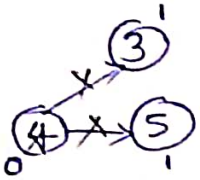


Delete the connected edges.

→



III



IV

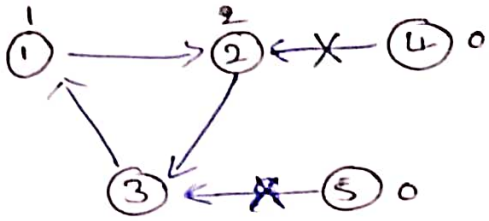
1 2 4 3 5

or

1 2 4 5 3.

} Two Topological order is possible.

2)

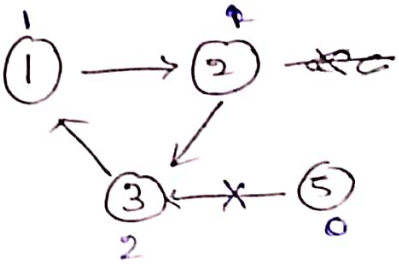


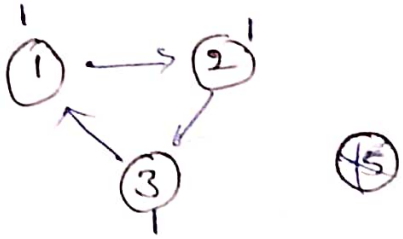
Case 1

4 5

Case 2 :

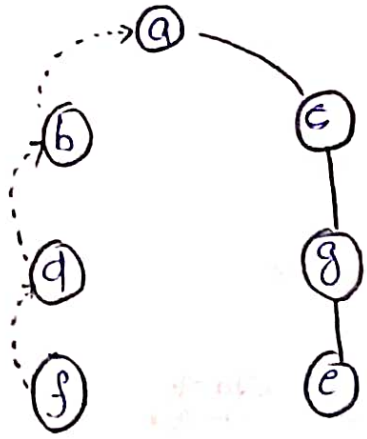
5 4





} In this case we can not complete the Topology ∴ ~~any vertex~~ Indegree of every vertex is one, hence we can't proceed further.

Tree representation



Topological order → a, b, d, f, c, g, e.

↳ In the graph there is cyclic motion hence the Topological order is not possible

↳ And this is not Unidirectional graph.

<u>Note</u>: If the root node is not given in the problem we have to assume <u>any one vertex/node</u> in the graph as root node.

BFS - Breadth First Search | Level block.

Queue DS → FIFO

For the BFS the data structure used is Queue and the nodes that will be visited is based on the levels or the distance to travel.

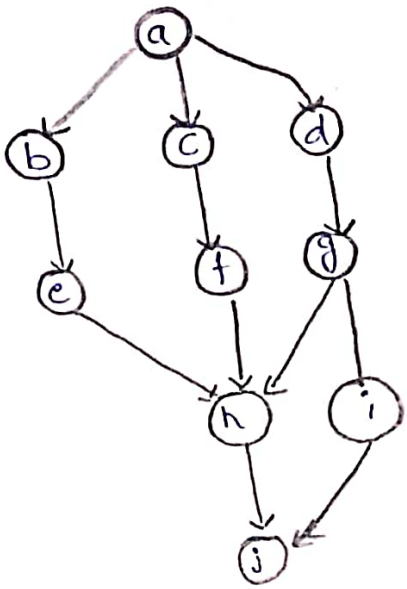
Level 0

Level 1

Level 2

Level 3

Level 4



Move to unvisited Node

Queue Data Structure would be used.

Queue U = deq(Q)	Representation For unvisited V = adj to U Moves to	Nodes visited (S)	queue (Q)	U - V output.
-	-	a	a	-
a	b, c, d	a, b, c d	b, c, d	a - b a - c a - d
b	e	a, b, c, d, e	c, d, e	b - e
c	f	a, b, c, d, e, f	d, e, f	c - f
d	g	a, b, c, d, e, f, e	e, f, g	d - g
e	h	a, b, c, d, e, f, g, h	f, g, h	e - h
f	-	a, b, c, d, e, f, g, h	g, h	-
g	i	a, b, c, d, f, g, h, i	h, i	g - i

'b' is inserted first. So it will be deleted in order to traverse the next adjacent node

order of node visited

g(U) is not considered because h already visited

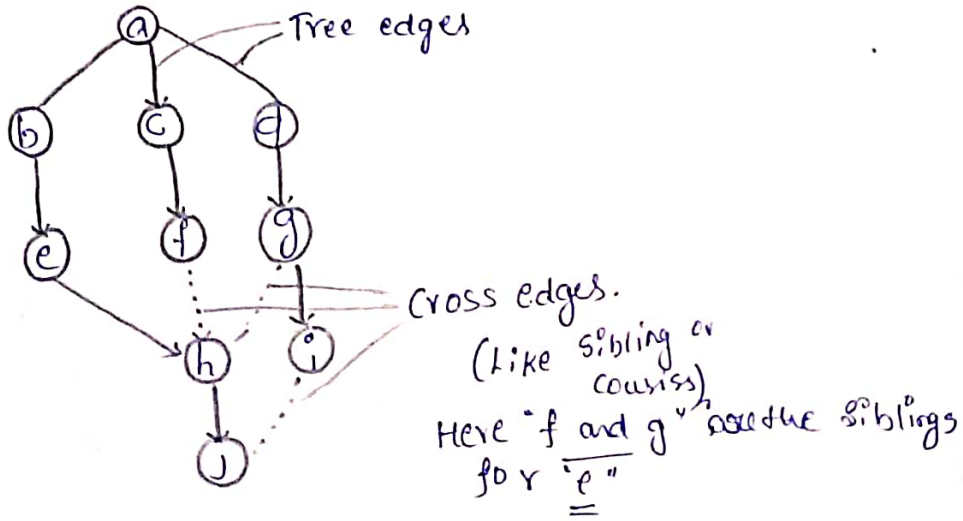
already visited to h

h	j	a, b, c, d, e, f, g, h, i	g, i, j	h - j
i	–	a, b, c, d, e, f, g, h, i	–	–
j	–	a, b, c, d, e, f, g, h, i, j	empty	–

Note : The ~~node~~ queue is empty at the last.

Order of Node visited

BFS Traversal



Cross edges connect to those vertices visited before but unlike back edges in BFS Tree they connect either to siblings or to cousins on the same or adjacent levels of BFS tree. Backedges in the DFS tree connect to the ancestors

Simple BFS

The Breadth First Search or BFS ~~for a graph~~ algorithm used to search a graph Data Structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on the nodes at the next depth level.

The only catch here is, that, Unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories.

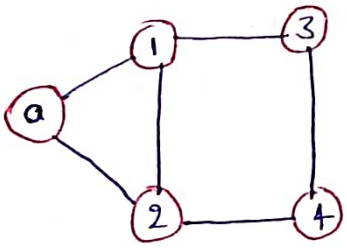
* Visited
* Not Visited.

How does BFS work.

Starting from the root, all the nodes at a perticular level are visited first and then the nodes of the next level are traversed ~~till all the members~~ till all the nodes are visited.

To do this a queue is used. All the adjecent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and poped from the queue.

Step 1: Initially queque and visited arrays are empty



Vistited

Queque

B.F.S. on Graph.

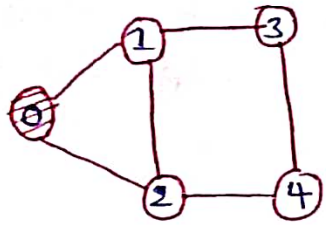
Queue and visited arrays are visited initially

Diff ~~rence~~ b/w

Step 2: Push node 0 into queue and mark it has visited

DS

TE



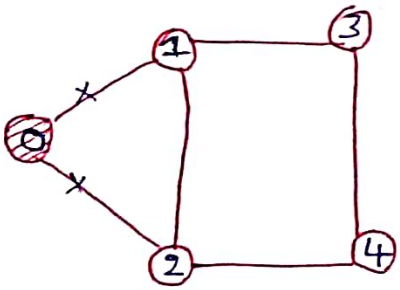
Visted | 0 | | | | |

Queue | 0 | | | | |

↑ Front.

Edge

Step 3: Remove node 0 from the front of queue and visited the unvisited neighbours and push them into queue.



Visted | 0 | 1 | 2 | | |

queue | 1 | 2 | | | |

/ remove 0 from queue.

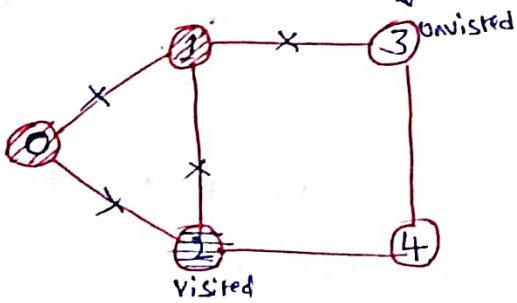
Front

Step 4: Remove node 1 from front of the queue and visit the unvisited neighbours and push them into queue.

Here unvisted neighbour

- Node removed

- visited

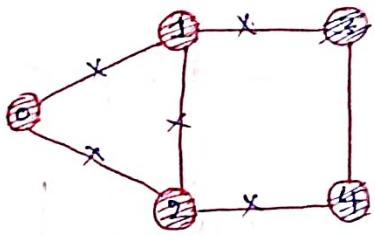


Visited | 0 | 1 | 2 | 3 | |

Queue | 2 | 3 | | | |

↑ Front

Step 5: Remove node 2 from the front of queque and visit the unvisted nighbours and push them into queque.



Visited | 0 | 1 | 2 | 3 | 4 |

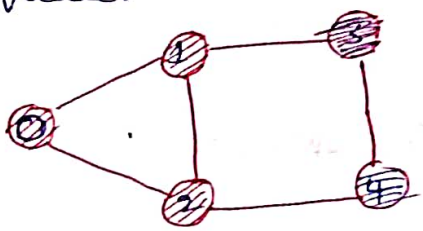
Queque | 3 | 4 | | | |

↑ front.

Step 6 : Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

→ As we can see that every neighbours of node 3 is visited.
~~nigbour and push them into que~~

Imp So move to the next node that are in the front of the queue.



Visited

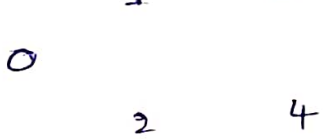
0	1	2	3	4

Queue

4				

↑ Front

Step 7 : Remove node 4 from the front of queue and visit the unvisited neighbour of node 4 and push them into queue.



Visited

0	1	2	3	4

Queue

Now, Queue becomes empty. So terminate these process of iteration.

C programing code.

```
#include <stdbool.h>
#include <stdio.h>
# ~~stdlib~~ Include <stdlib.h>
#define MAX_VERTICES 50

typedef struct Graph_t {
    int v;
    bool adj[MAX_VERTICES][MAX_VERTICES];
}Graph;

//Constructor
Graph* Graph_Create(int v)
{
    Graph* g = malloc(sizeof(Graph));
    g->v = v
    for (int i=0; i<v; i++)
    {
        for (int j=0; j<v; j++)
        {
            g -> adj[i][j] = false;
        }
    }
    return g;
}

//Destructor
void Graph_destroy (Graph * g)
{
    free(g);
}
//~~void~~ Function to add an edge to graph
void Graph_addEdge (Graph * g, int v, int w)
{
    //Add w to v's list
    g -> adj[v][w] = true;
}
```

Graph data type

vertices

~~Void Graph_BFS~~

```
//Prints BFS traversal from a Given Source s
Void Graph_BFS (Graph * g, int s)
{
    //Marks all the vertices as not visited.
    bool visited[MAX_VERTICES];
    for (int i=0; i < g->V; i++)
    {
        visited[i] = false;
    }
    //Create new Queue for BFS
    int queue [MAX_VERTICES];
    for (int i=0; i < g->V; i++)
    {
        visited[i] = false;
    }
    //Create a Queue for BFS.
    int queue [MAX_VERTICES];
    int front = 0, rear = 0;
    //Mark the current node as Visted and enqueue it.
    Visited [s] = true;
    queue [rear++] = s;
    while (front != rear)
    {
        s = queue [front++];
        printf("%d", s);
        //Get All adjacent vertices of the dequeued vertex s.
        //If an adjacent has not been visited
        //then mark it visited and enqueue it
        for(int adjacent = 0; adjacent < g->V; adjacent ++)
        {
            if (g->adj [s] [adjacent] && !visited [adjacent])
            {
                visited [adjacent] = true;
                queue [rear ++] = adjacent;
}   }   }   }
```

```
//Driver Code
int main()
{
    //create a graph
    Graph *g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 2);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 0);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 3, 3);
    printf("Following is Breadth First Traversal "" (Starting from
            Vertex 2)\n");
    Graph_BFS(g, 2);
    Graph_destroy(g);
    return 0;
}.
```

Output

```
Following is Breadth First Traversal (Starting From Vertex 2)
2 0 3 1
```

What is optimal Solution & feasible Solution
↓
maximisation
or
minimisation

Greedy Techique

- Spanning tree
  - Prim's Algorithm.
  - Kruskal's → $O(|E| \log |E|)$ → any edges are going to use.

  It is better than Kruskal's Algorithm

- Shortest Path - Dijkstra
- Knapsack → Soldier's bag

A greedy Algorith tries to find the best solution for each Sub-Problem with the hope that this will lean and optimal Solution for the hole problem.

Properties of Greedy Technique

1) Feasible
   The Choice made must satisfy the Constrain of the problem Such a Solution is called

2) Locally optimal
   The current choice made has to be the best local choice among all feasible Solution.

3) Irrevotable
   Once the choice is made it should not be change in the Subsequent Steps of the algorithm.

Optimal Solution is the feasible Solution that Satisfy the object of the problem

Knap Sack
- Brute force Method
- Greedy technique — fractional knapsack { ½ fraction partially.
- Dynamic programming — 0 and 1 knapsack (either full or empty) like washing machine
- Branch Bound.

Fractional knapsack Problem.

$n = 7$

$M = 15$

M = Maximum Capacity of Knapsack

$X = 1$ or $\frac{RC}{Wight}$

$RC = RC - (x_i * w_i)$ → remaining Capacity

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

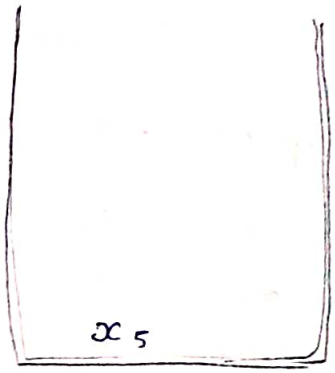
$C = \frac{\text{Profit}}{\text{Weight}}$ 5 — — — — — —

@ More profit Make it as 1

single dimensional

X → How much portion getting inserted between 0-1

	1		1		1	1	
	x_1	x_2	x_3	x_4	x_5	x_6	x_7



A Container having a Capacity of 15 kg.

Object	Wight	Profit	$X = 1$ or $\frac{RC}{Wight}$ (remaining Capacity)	Profit $x_i * P_i$	$RC = RC - (x_i * w_i)$
					$= 15 - (1 * 1) = 14$ — rm
x_5	1	6	$X = 1$	$1 * 6 = 6$	$= 14 - (1 * 2)$
x_1	2	10	$X = 1$	$1 * 10 = 10$	$= 12$
x_6	4	18	$X = 1$	$1 * 18 = 18$	$= 12 - (1 * 4)$
					$= 8$
x_3	5	15	$X = 1$	$1 * 15 = 15$	$= 8 - (1 * 5)$
					3
x_7	1	3	$X = 1$	$1 * 3 = 6$	$= 3 - (1 * 1) = 2$
x_4	3	5	$X = \frac{2}{3}$	$\frac{2}{3} \times 5 = 3.33$	$= 2 - (\frac{2}{3} \times 3) = 0$

Objective is to max $\sum_{i=1}^{n} x_i P_i$

$= (1 \times 2) + (\frac{2}{3} \times 3) + (1 * 5) + 0 + (1 * 1) + (1 \times 4) + (1 * 1)$

$= 2 + 2 + 5 + 1 + 4 + 1$

$= 15$

Algorithm Fractional_knapsack (M, n, w, x)

//purpose: Find the Solution Vector that shows the Fraction of object Selected.

// : M - Maximum capacity of knapsa
is an array that consits of weights of all n objects

//output : X - which Contains the Solution Vector

```
for i ← 1 to n
    x[i] ← 0
end for
rc ← M // initialy rc is M or the Maximum Capacity of knapsack

For i ← 1 to n
    if (w[i] > rc) break // ith object can not be selected
    x[i] = 1 // Select the ith object
    rc = rc - w[i] // Find the remaining capacity of knapsack
endfor
if (i <= n)
    x[i] = rc/w[i] // find the Fraction of object to be selected.
```

V = Vertices of the given Graph

E = Edges

V_T = Vertices of MST

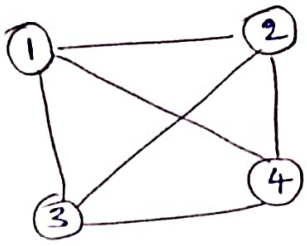
E_T = Edges of MST

V_0 = Starting Vertex from where we are begging.

Spanning tree

For a Given Graph $G = (V, E)$ the Spanning tree is an acyclic Subgraph $G' = (V', E')$ Such that $V' = V$ and $E = |V| - 1$

$\downarrow$ Cardinality of V.



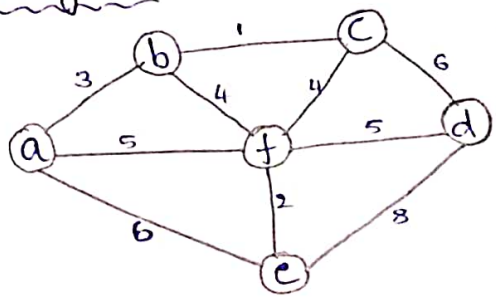
* More than 1 Spanning tree

2 Algorithms to Find

a) prim's - adjacent Node Algorithm.

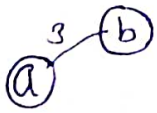
b) kruskal's Algorithm

Prim's Algorithm → Used to Find a minimum Spanning tree for a given weighted graph.



Tree Vertices	Running Vertices
begin with a (- -)	minimum weight select b(a,3), f(a,5)
	c(-,∞), d(a,5)
	e(a,6)

Illustration.

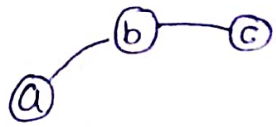


b(a,3)

min $\begin{cases} c(-,\infty) \\ c(b,1) \end{cases}$ d(-,∞),

min considered

min $\begin{cases} e(a,b) \\ e(b,\infty) \end{cases}$ min $\begin{cases} f(a,5) \\ f(b,4) \end{cases}$



$c(b,1)$ $\quad$ $\min\begin{Bmatrix} d(-,\infty) \\ d(c,6) \end{Bmatrix}$, $e(a,6)$

$\min\begin{Bmatrix} f(b,4) \\ f(c,4) \end{Bmatrix}$

$f(c,4)$ $\quad$ $\min\begin{Bmatrix} d(c,e) \\ d(f,5) \end{Bmatrix}$ $\min\begin{Bmatrix} e(a,6) \\ e(f,2) \end{Bmatrix}$

$e(f,2)$ $\quad$ $\min\begin{Bmatrix} d(e,8) \\ d(f,5) \end{Bmatrix}$ $\min\begin{Bmatrix} e(a,b) \\ e(f,2) \end{Bmatrix}$

$d(f,5)$ $\quad$ Empty.

<u>Algorithm</u>

// Prim's Algorithm for constructing a minimum spanning tree.

// Input A weighted connected graph $G = (V, E)$

// Output ~~output~~ E_T, the set of edges

$V_T \leftarrow (V_0)$

$E_T \leftarrow \emptyset$

for $i \leftarrow 1$ to $|V| - 1$ do

find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u) such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup (u^*)$

$E_T \leftarrow E_T \cup (e^*)$

return E_T

* Prim's Algorithm: Time Efficiency.

* Needs priority queue for locating classest vertex

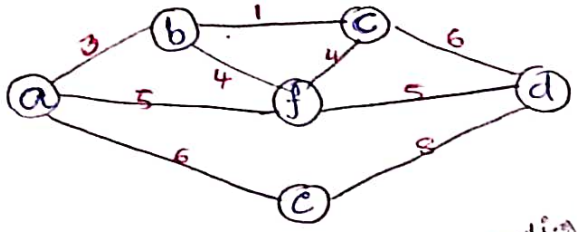
* Efficiency

→ $O(|V|^2)$ for weight matrix representation of graph ~~with n vertices~~ and array implimentation of priority queue.

→ $O(|E| \log |V|)$ for adjancency lists representation of graph with n vertices and m edges and min-heap implementation of the priority queue.

Kruskal's Algorithm for Finding MST

* Sort the edges of the given graph in increasing order
* Sorting with empty subgraph scan the sorted list
* Scan the sorted list by adding on the list to the current subgraph.
* If such an edge inclucion does't form a cycle then that edge can be consider as the part of the spaning tree otherwise just discard the edge if it breack cycle.



Tree edges

— initial empty set

Sorting every time.

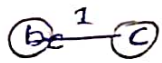
Sorted edges

$\overset{1}{bc}$, $\overset{3}{ab}$, $\overset{3}{bf}$, $\overset{4}{fc}$, $\overset{5}{ac}$, $\overset{5}{fd}$, $\overset{5}{ae}$, $\overset{6}{cd}$, $\overset{8}{ed}$

Shortest edge

the edges & sorted edges should be equal.

Illustration

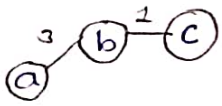


bc

bc, $\overset{3}{ab}$, bf, cf, af, df.

Shortest edge

ae, cd, de,

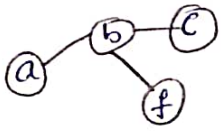


bf

bc, ab, $\overset{4}{bf}$, cf, af,

df, ae, cd, de

Shortest edge.



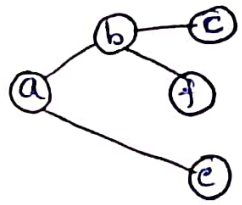
af

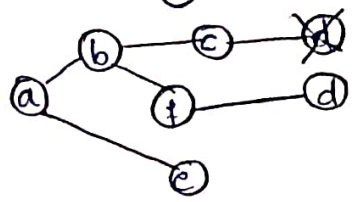
bc, ab, bf, cf, af,

df, ae, cd, de)

Disgard cf as it creates ~~cycle~~

af	bc, ab, bf, cf, af, df, de, ~~ae~~ de	Discard if as it creates
df	bc ab bf cf, df, ae → Shortest edge cd, de.	— 1 —
ae	bc, ab, bf, cf, af, df, ae, cd, de	
cd	bc, ab, bf, cf, af, df, ae, cd, de	
de	bc, ab, bf, cf, af, df, ae, cd, de	





Algorithm

// Krustral's algorithm for constructing a minimum spanning tree

//Input: A weighted connect graph $G=(V, E)$

//Output: E in nondeconcreasing order of the edge weight

$w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$

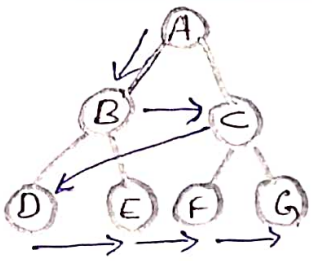
Sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \ldots \leq$

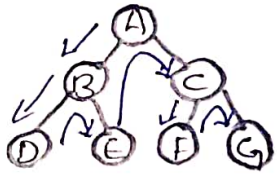
If the heap sort technique is used in all the cases this only.

Difference between prim's Algorithm and Kurskal's Algorithm.

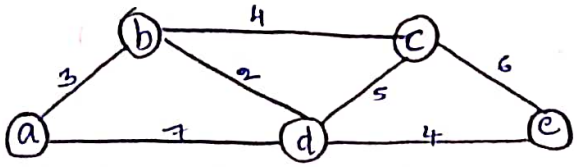
Difference between BFS and DFS.

BFS	DFS.
BFS Stands for Breadth First Search	DFS stands for Depth First Search
BFS uses Queue Data Structure for finding the Shortest path	DFS use Stack Data Structure
BFS can be used to find single source shortest path in an unweighted graph, because in BFS we reach a vertex with minimum no. of edges from a source vertex	In DFS, we might traverse through more edges to reach a destination vertex from a source.
BFS more suitable for searching vertices which are closed to the given source A B C D E F A, B, C, D, E, F	DFS is more suitable when there are solutions away from a source A B C D E F A, B, D, C, E, F

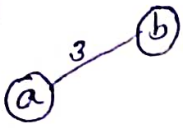




Shortest Path - Dijkstra's algorithm. - Single Source Shortest path

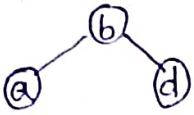


Source Vertices -> a

Tree Vertices	Remaining Vertices	Illustration
a(–, 0)	b(a, 3) c(–, ∞)	
by which vertices/ travelling it has come	d(a, 7) e(–, ∞)	
	b(a, 3) is the ~~smallest~~ list one.	

b(a, 3)

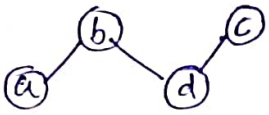
$$\min\begin{Bmatrix} c(b, 3+4) \\ e(-, \infty) \end{Bmatrix} \quad \min\begin{Bmatrix} d(a, 7) \\ d(b, 3+2) \end{Bmatrix}$$



Compare both whichever is min, that we have to consider.

d(b, 5)

$$\min\begin{Bmatrix} c(b, 7) \\ c(d, 5+5) \end{Bmatrix} \quad \min\begin{Bmatrix} e(-, \infty) \\ e(d, 5+4) \end{Bmatrix}$$



Shortest distance is the highest priority

c(b, 7)

$$\min\begin{Bmatrix} e(c, 7+6) \\ e(d, \end{Bmatrix}$$

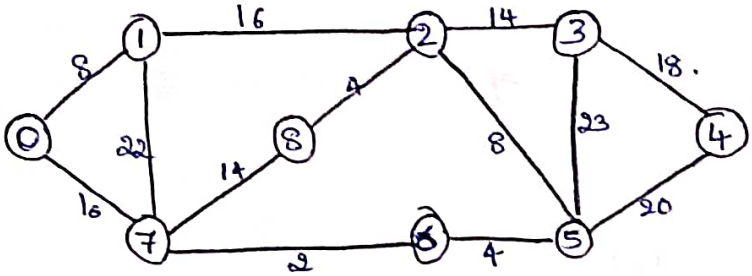
Shortest parts are.

a to b = $a \xrightarrow{3} b$ = 3

a to c = $a \xrightarrow{3} b \xrightarrow{4} c$ = 7

a to d = $a \xrightarrow{3} b \xrightarrow{2} d$ = 5

a to e = $a \xrightarrow{3} b \xrightarrow{2} d \xrightarrow{4} e$ = 9



Tree Vertices	Remaining Vertices	Illustration.	Illustration
3(8, 0)	2(3, 14), 5(3, 23), 4(3, 18), 1(8, ∞), 8(–, ∞), 7(–, ∞), 6(–, ∞), 0(–, ∞)		

Minimum consider

2 (3, 14)

$\min\begin{Bmatrix} 1(2, 16) \\ 1(-, \infty) \\ 1(2, 16) \end{Bmatrix}$, $\min\begin{Bmatrix} 8(2, 4) \\ 8(-, \infty) \\ 8(2, 4) \end{Bmatrix}$, $\min\begin{Bmatrix} 5(2, 14+8) \\ 5(3, 23) \\ 5(2, 23) \end{Bmatrix}$

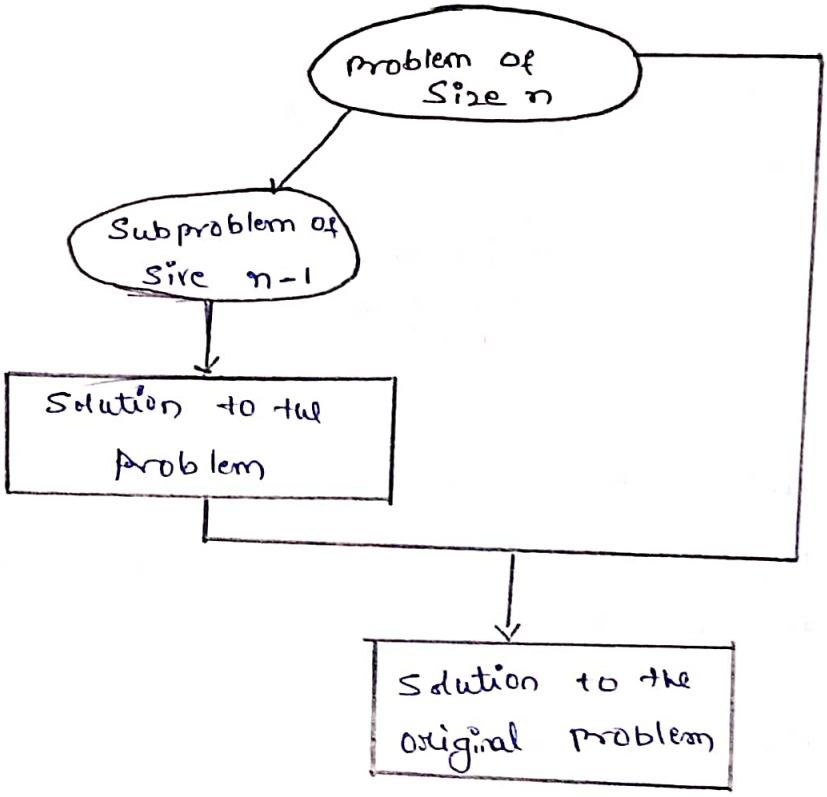
$\min\begin{Bmatrix} 4(2, \infty) \\ 4(3, 18) \end{Bmatrix}$, $\min\begin{Bmatrix} 6(2, \infty) \\ 6(-, \infty) \end{Bmatrix}$, $\min\begin{Bmatrix} 7(-, \infty) \\ 7(-, \infty) \end{Bmatrix}$

$\min\begin{Bmatrix} 0(-, \infty) \\ 0(-, \infty) \end{Bmatrix}$, ~~min~~

3 Types of Decrease and Conquer.

1) Decrease by Constant

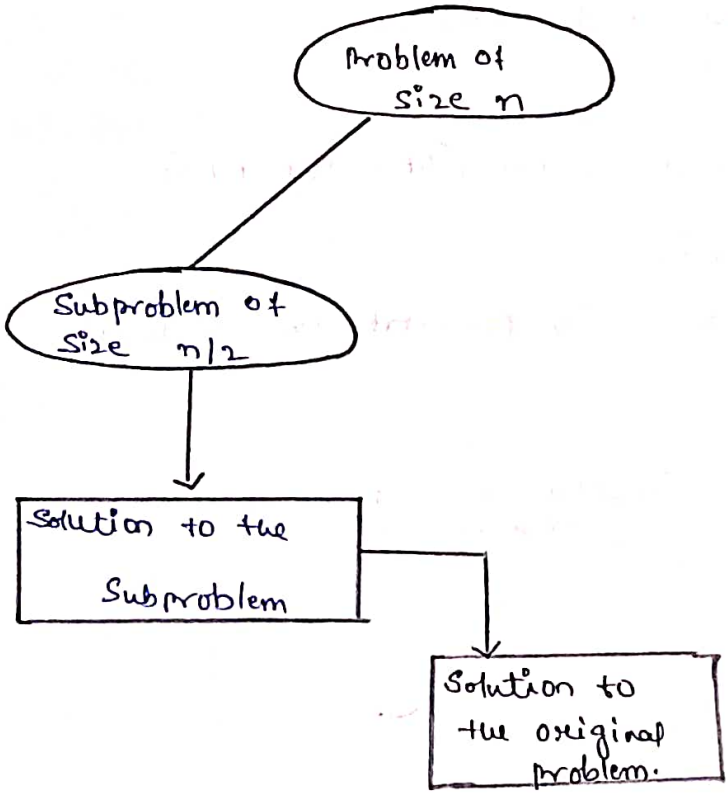
* Insertion Sort
* Graph traversal algorithms (DFS and BFS)
* Topological Sorting
* Algorithm for generating permutations, Subsets.



2) Decrease by Constant Factor.

→ Usually by half ($n/2$)

* Divide and Search
  1) binary search
  2) Exponential by Squaring
  3) Multiplication of Large integers.



3) Decrease by Variable size.

* The size of pattern varies from one iteration of an algorithm to another.

Eg: * Computing median and Selection problem.

* Interpolation Search

* Euclid algorithm.

Euclid algorithm

What's the Difference ?

→ Consider the problem of exponentiation : compute x^n

* Bruce Force → $n-1$ multiplication
* Divide & Conquer → $T(n) = 2 * T(n/2) + 1$
  $\boxed{= n-1}$
* Decrease by one → $T(n) = T(n-1) + 1 = \boxed{n-1}$
* Decrease by constant Factor → $T(n) = T(n/a) + a - 1$
  $= (a-1) \log_a n$
  $\boxed{= \log_2 n \quad \text{When } a = 2}$

Space and Time Tradeoffs.

An Algorithm must be time and space efficient. To achive both may not be possible for certain algorithm.

* Space may be important factor in some situation and time may be important factor in some other situations.

Thus time-space trade-off is the situation in which either time efficiency can be achived at the cost of space and space efficiency can be achived at the cost of time.

Three Varieties of Space-for-time algorithm:

* input enhancement – preprocess the input (or its part) to store some info to be used later in solving the problem
    - Counting Sort
    - String Searching Algorithm.
* prestructuring – uses additional space to facilitate faster and flexible access to the data.
    - hashing
    - indexing schemes
* Dynamic Programming: Recording the solutions to overlapping subproblems of a given problem in table from which the solution to original problem.
    - knapsack problem
    - Warshal-Floyd algorithm TSP etc.

Sorting by Counting.

- The method of arranging the elements in required order using input-enhancement technique.

- Two algorithms
  - → Sorting by Comparison (Comparison counting method)
  - → Sorting by Distribution (Distribution counting or Lenear counting)

* Input Enhancement. – preprocess the problem's input, in whole or in part, and store the additional information obtained to accelerate solving the problem. afterward.

Comparison Counting.

Eg: Array

A					

A	25	45	10	20	50	15
Index	0	1	2	3	4	5
Count	~~2~~ 3	1	0	0	1	0

	~~3~~ ~~2~~ 4	0	0	2	0

		0	1	3	1

			2	4	1

				5	1

```
for i ← 0 to n-2
    for j ← i+1 to n-1
```

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

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

$= n-1+i+1+1$

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

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

$\cancel{n-n+2+1}$

$3+(2+1)$

10	20	10	20	25
0	1	2	3	4
~~124~~	0	0	0	0
	01	1	1	0
		23	1	0
			2	0
0	1	2	3	4

	0				
	10	20	10	20	25
		0	0	0	
When i=0	~~123~~	0			
	4	1	1	1	1
	0				
				2	2
When i=1	0	3	1		
				2	
When i=2					

Input array = A[0..5] → 62 31 84 96 19 47

lessthan

	62	31	84	96	19	47
	0	1	2	3	4	5
Count	0	0	0	0	0	0
Count 1	1	0	1	1	0	0
Count 2	No use of first column it is already done	1	2	2	0	1
Count 3	3	1	4	3	0	1
	No use Now					
Count 4	3	1	4	5	0	1
Count 5	3	1	4	5	0	2

True = pivotal Increament

False = ~~i+1<0~~ same colum increamend.

(n-1) count happens.

	62	31	84	96	19	47
Count =	3	1	4	5	0	2

array according to count indexes

	0	1	2	3	4	5
Array = Final Sorted Array	19	31	47	62	84	96.

Algorithm

```
for i ← 0 to n-1 do
    Count[i] ← 0
for i ← 0 to n-2 do
    for j ← to n-1 do
        if (a[i] < A[j])
```

Distribution Counting.

- Input non hance counting technique

When to apply

12 13 10 12 10 12 11 10 13

Within the lower bound = 10 and upper bound = 13

Set = {10, 11, 12, 13}

	10	11	12	13
- frequency	3	1	3	2
	3	4	7	9

→ Distribution Value stores in D[] → single-D array

Given List :

A[ ] = 12, 13, 10, 12, 10, 12, 11, 10, 13 No. are Repeating.

	0	1	2	3
	3	4	7	9
A[8]	3	4	7	8
A[7]	2	4	7	8
A[6]	2	3	7	8
A[5]	2	3	6	8
A[4]	1	3	6	8
A[3]	1	3	5	8
A[2]	0	3	5	7
A[1]	0	3		
A[0]				

Distribution count Sort

Horspool's String patter Matching

textstring : patternstring

P = BARBER

length (P) = 6

4	2	6	....	1	....	3	...	6

When we do search for a string in a note pad / wordfile, browser, or Data base, patter searching algorithms are used to show the search results.

|| Not Matching

Matching

Test String

JIM_SAW_ME_IN_BARBER_SOHP

BARBER

Shift by 4 → BARBER

Shift by 2 → BARBER

Shift by 6 → BARBER

Shift by 3 → BARBER

The pattern is found.

RV _ COLLEGE _OF _ ENGINEERING

Teststring => ENGINE

ENGINE

ENGINE

ENGINE

ENGINE

Dynamic programming

→ It is used to solve optimization problem

① It Breaks down the complex problem into simpler subproblem

② find optimal solution to these subproblem

③ Store the result of subproblem (memoization)

④ reuse them so that same subproblem is not calculated more than once

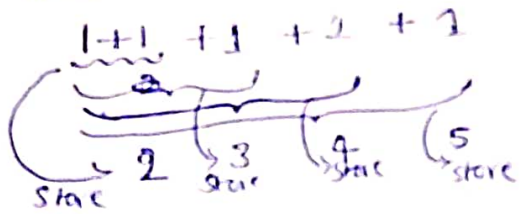
⑤ finally calculates the result of complex problem.

→ Applicable to problems are having properties of:-

→ overlaping subproblem & optimal substructure.

→ Space complexity is increased.

→ Time complexity is less.



Best example is Dynamic programing is Fibanacci Series

0 1 1 2 3 5 8 13 21 34

Alogorithm

```
int fin(n)
{ if (n==0)
      return 0
  if (n==1)
      return 1
}
if (n<0)
    error
sum = fib(n-1) + fib(n-2)
return sum
```

DYNAMIC PROGRAMMING

Dynamic Programming is a general for Solving problems defined by recurrences with overlapping sub problems.

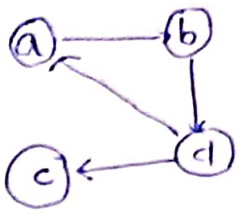
1) Warshall floyds → Transitive property of relation
floyed marshalls.

$(a,b) = (a,b)(b,c)$

aRc ; bRc then aRc

$a \to b \to c$

Input Graph:



D^0 →

Adjacency Matrix Representation (Distance Matrix)

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

Step 1: Choising vertex 'a' as the intermediate vertex

$(d,a) = 1$, $(a,b) = 1 \to (d,b) = 1$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

(b,d).

Step 2: Intermediate Vertex b is

$(a,b) = 1$ $(b,d) = 1$ $(a,d) = 1$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

Step3 : Intermediate Vertex 'c'

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

Step4 : Intermediate Vertex

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

(d, a) = 1

(d, a) = 1

(a, d) (d, d)

(d, a) (d, d) (d, b)

(b, d), (d, b)

Algorithm.

Algorithm Warshal(n, A, P)

For i ← 0 to n do ← Source Vertices

For i ← 0 to n do ← Destination

For k ← 0 to n do ← Intermediate

if(P[i, j] = 0 and if (P[i, k] = 1 and P[k, j] = 1)

P[i, j] = 1

n = No. of. Vertices

A =

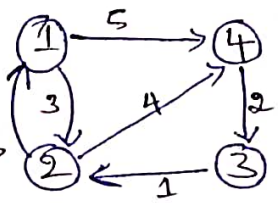
P = result & path matrix

All PAIR Shortest path

· Floyd-Warshall Agorithm: Floyd-Warshall Agorithm for finding the Shortest path between the all the pair of vertices in a weighred graph.

also called → Floyd's algorithm,
Roy-Floyd algorithm
Roy-Warshall algorithm or
WFI algorithm

Eg



Warshal floyd works on bidirectional

1) Create a Matrix

$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

→ These are Same.

→ The diagnal vertex are always 0

2) Now Create the Matrix D^1 using the matrix D^0. The elements in the first Column and the first row are left as they are. the remaining cells are filled.

(2,3) =

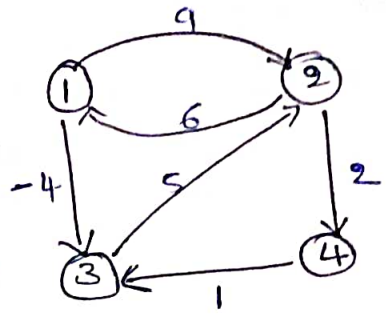
$$\begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \longrightarrow \begin{matrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & (32) & 0 & \\ \infty & & & 0 \end{matrix}$$

(2,3)

$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & \infty \\ 2 & 6 & 0 & \infty & 2 \\ 3 & \infty & 5 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

D^0 → Distance ~~Vector~~ Matrix

→ all the Diagnolsare 0



always (1,1)(2,2)(3,3)(4,4) have always 0. because we don't have 1-1 edges here

$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & \infty \\ 2 & 6 & 0 & 2 & 2 \\ 3 & \infty & 5 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

Shortest distance of each pair vertexes via 1.

$\boxed{2 \to 3}$

$D^0[2,3]$ $D^0[2,1]$ $D^0[1,3]$

∞ $(6) + (-4)$

$\infty > 2$

$\boxed{2 \to 4}$

$D^0[2,4]$ $D^0[2,1]$ $D^0[1,4]$

2 $6 + \infty$

$2 < \infty$

$\boxed{3 \to 2}$

$D^0[3,2]$ $D^0[3,1][1,2]$

5 $\infty + 9$

$5 < \infty$

$\boxed{3 \to 4}$

$D^0[3,4]$ D^0

∞

$\boxed{4 \to 2}$

$D^0[4,2]$ $D^0[4,1][1,2]$

∞ $\infty + 9$

$\boxed{4 \to 3}$

$D^0[4,3]$ $[4,1][1,3]$

$1 <$ $(\infty) + (-4)$

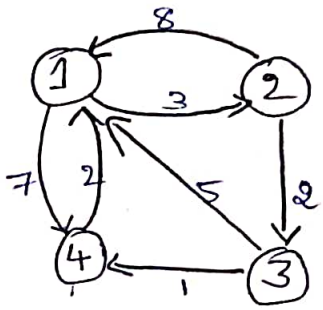
∞

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & \infty \\ 2 & 6 & 0 & 2 & 2 \\ 3 & \infty & 5 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

$$\begin{array}{cccc} 0 & 9 & -4 & 11 \\ 6 & 0 & 2 & 2 \\ 14 & 5 & 0 & \infty \\ \infty & \infty & 1 & 0 \end{array}$$

shortest pair of each pair vertex via two

Q.



$\longrightarrow D^0 =$

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

Distance Matrix $D^1 =$

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

Distance Matrix $D^2 =$

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

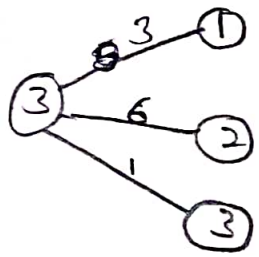
Distance Matrix $D^3 =$

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

$D4 =$

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 2 & 6 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

Shortest path



Varuty.

```
//Floyd-Warshall Algorithm in C
#include <stdio.h>

#define nV 4 //defining the no.of vertixes

#define INF 999

void printMartrix(int matrix[][nV]);
void floydwarshall(int graph[][nV]) { //Implementing floyd warshall
    int matrix[nV][nV], i, j, k;        Algorithm.

    for (i=0; i<nV; i++) {
        for(j=0; j<nV; j++) {
            matrix[i][j] = graph[i][j];

        //adding vertices individually
        for(k=0; k<nV; k++) {
            for(i=0; i<nV; i++) {
                for(j=0; j<nV; j++) {
                    if (matrix[i][k] + matrix[k,j] < Matrix[i][j])
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                }
            }
        }
        PrintMatrix(matrix);
    }

void PrintMatrix(int matrix[][nV]) {
    for (int i=0; i<nV; i++) {
        for (int j=0; j<nV; j++) {
            if (matrix[i][j] == INF)
                Printf("%4s", "INF");
            else
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
```

```
    }
}
int main() {
    int graph[nV][nV] = {{0, 3, INF, 7},
                         {8, 0, 2, INF},
                         {5, INF, 0, 1},
                         {2, INF, INF, 0}}

    floydWarshall(graph);
}.
```

Easy

0/1 knapsack problem — Dynamic program

No. of object = $M = 5$

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

$p = \{12, 10, 20, 15\}$

0/1 knapsack.
either pic the complete item or leave

Eg: laptop
↓
we want buy a complete new laptop it is not posible to bring the laptop half like display & other components only.

$$\left\{\begin{array}{l} \text{No. of rows} = \text{No. object} + 1 \\ \text{No. of columns} = \text{max Capacity of knapa sack} + 1 \end{array}\right\}$$

p_i	w_i		0	1	2	3	4	5
		0	0	0	0	0	0	0
12	2	1	0	0	12	12	12	12
10,	1	2	0	10	12	22	22	22
20	3	3	0	10	12	22	30	32
15	2	4	0	10	15	25	~~30~~	37

→ In the final cell we will get the ans

wieght should be written in the ascending order.

<u>Formula</u> = $Max[i, w] = max(m[i-1, w], m[i-1, w-w[i]) + P[i])$

$4, 6 = max(m(3,7), m(3, 7-6) + 1$

$= max(m(3,7), m(3,1) + 1)$

$= (5, 0+1)$

$\boxed{= 5}$

Weight = {3, 4, 6, 5}

Value / profit = {2, 3, 1, 4}.

$w = 8 \rightarrow$ bag having capacity of 8 kg

$n = 4$ ↓ no. of items

devide the problem into sequence of steps.

Colum = capacity of bag

row = No. of object + 1

consider to the value

P_i	W_i	↓ item / bag Wight →	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
2	3	1	0	0	0	2	2	2	2	2	2
3	4	2	0	0	0	2	3	3	3	5	5
4	5	3	0	0	0	2	3	4	4	5	6
1	6	4	0	0	0	2	3	4	4	5	$\boxed{6}$

→ we dont have any item to store

8 Substeps

Ans of Subproblem written the cells.

used to formula.

The Maximum profit is 6.

according to weight we take profit.

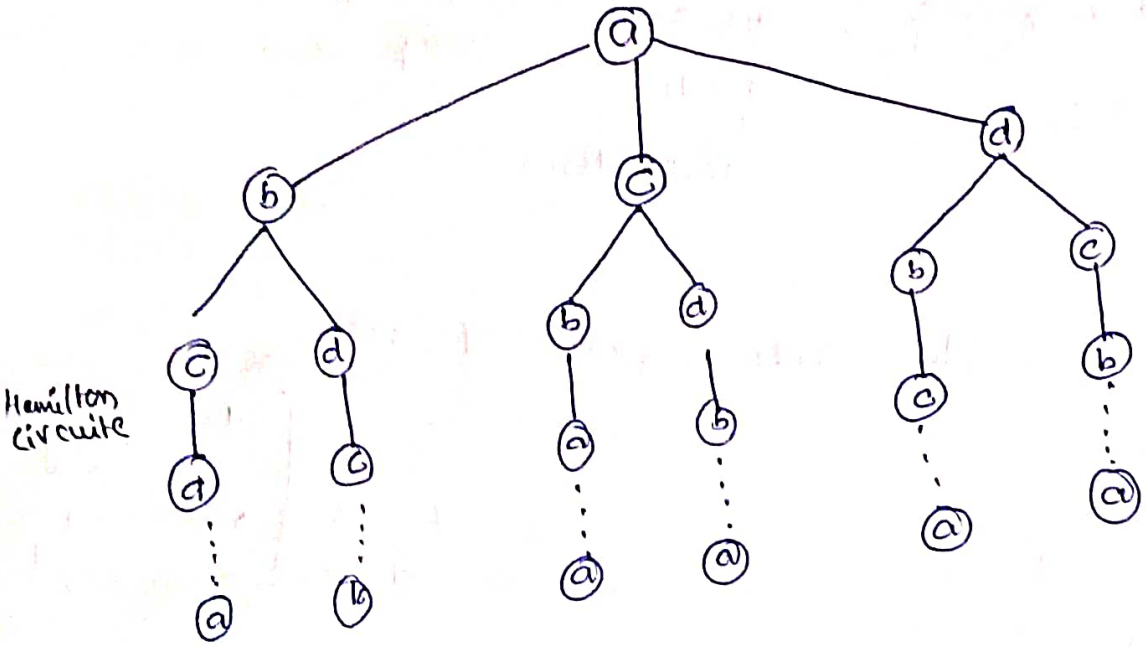
→ wight should be written in the ~~ascending~~ Increasing order.

First row and First column.

First row : If the capacity of the knapsack (bag) is 0 then there is no space to any items in the knapsack therefore the maximum value that can be put in the knap sack is also 0

First column :

If there are no items to put in the knapsack, ~~is also~~ then the maximum value that can be put in the knapsack is also 0.



$g(b, \phi) = 5$

$g(c, \phi) = 6$

$g(d, \phi) = 8$

$g(b, c) = 15$

$g(b, d) = 9$

$g(c, b) = 5$

$g(c, d) = 8$

$g(d, b) = 5$

$g(d, c) = 6$

$g(b, \{c, d\}) = 25$

$g(c, \{b, d\}) = 25$

$g(d, \{b, c\}) = 27$

$g(a, \{b, c, d\}) = 35.$

Subproblem : Here in the TSP using Dynamic programing after reaching the ith node finding the to that ith node is a a subproblem.

If we solve the recursive equatio we will get total of $(n-1) \times 2^{n-2}$ ~~this~~ many subproblems we will get. each such subproblem ~~On~~ times (it's find the path to reaching $\underbrace{n-1}$ nodes) $n-1 \approx n$

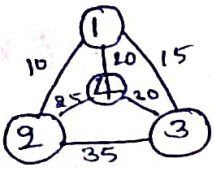
$$(n-1) * n^{n-2} \approx n 2^n$$

The total time complexity of TSP is ~~$n^2 \times 2^n$~~

$$\underbrace{O(n * 2^n)}_{\text{Space complexity.}} * O(n) = O(n^2 * 2^n)$$

Question:

Apply Dynamic program To find optimal tool for the following istance of TSP column.

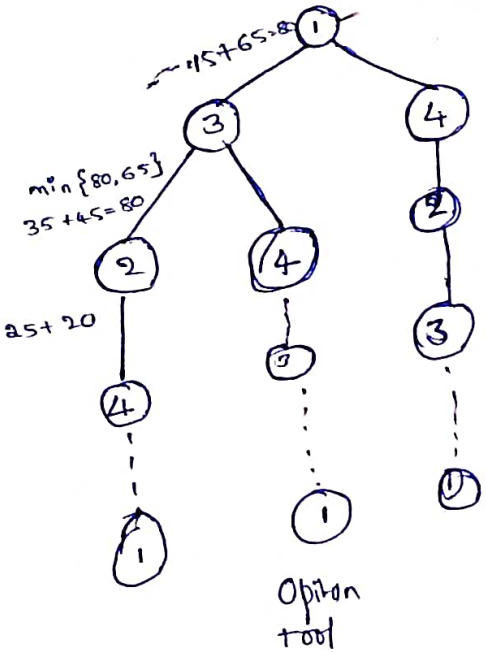


No. of cities | vertices = 4

Source Vertices = 1

Constraint → City 3 has to be visited City 2

Ans



$$g(i, S) = \min_{k \in S} \{ c_{ik} + g(k, S - \{k\}) \}$$ General formula.

$$g(a, \{b, c, d\}) = \min \{ c_{ak} + g\{b, c, d\} - k$$

Knapsack Problem - In Dynamic programming.

→ Divide problem into sequence of steps.

item — weights } .
profit

$!>20$ <=

$w = \frac{?}{20.}$ kg.

Given N Items where each item has some weight and profit associated with it and also given a back with capacity w, [i.e the bag can hold at most w weight in it]

Task: The task is to put the items into the bag such that the sum of profit associated with them is the maximum possible

2 - types

1) 0/1 → Dynamic Method.

2) Fractional → Greedy Method

Question

weight = {3, 4, 6, 5}

profits (values) = {2, 3, 1, 4}

→ capacity of the bag

W = 8

n = 4

→ No. of items.

N ↓ / W →	P_i	W_i	0	1	2	3	4	5	6	7	8
0			0	0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2	2
2	3	4	0	0	0	2	3	3	3	5	5
3	4	5	0	0	0	2	3	4	4	5	6
4	1	6	0	0	0	2	3	4	4	5	$\boxed{6}$

maximum profit.

max {3 + 0, 2}
= 3

max {3, +0, 2}
max {3+0, 2}
max {3+2, 2}
max {3+2, 2}

max {4+0, 3}
max {4+2, 5}
max {4+3, 5}
max {1+4, 4}
{1+4, 6}

{1+0, 5}
{1+0, 6}

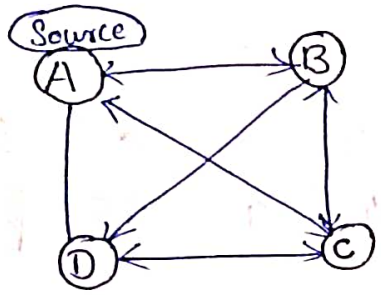
TSP using Dynamic programming. – Cycle

↓

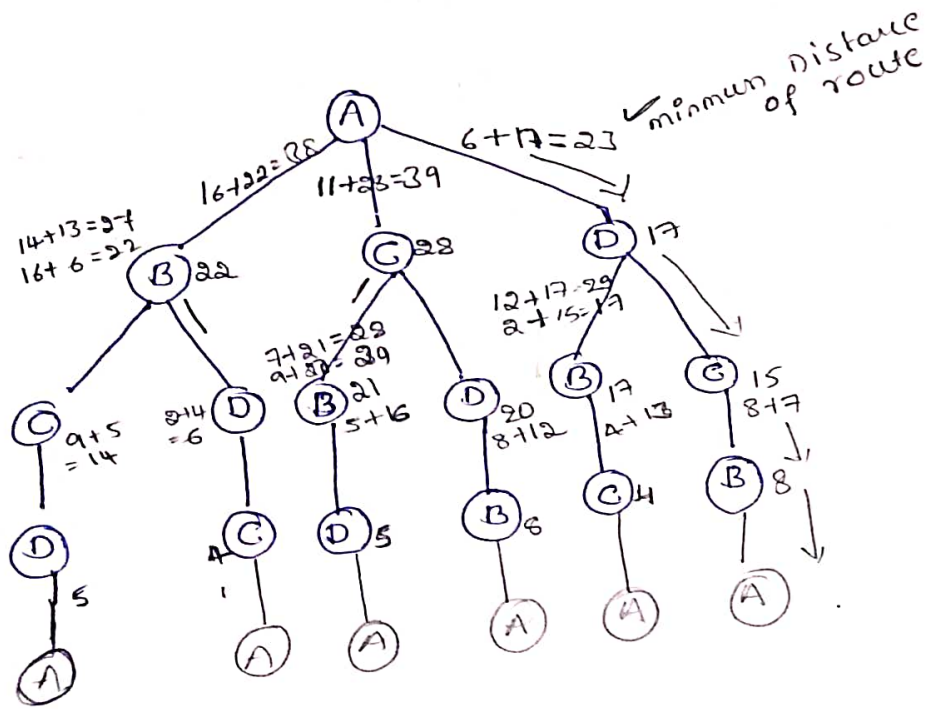
Travelling Salls Man problem.

Given a set of Cities and the distance b/w every pair of Cities, the problem is to find the shortest possible route that visit every City exactly once and returns to the point.

Question :



	A	B	C	D
A	0	16	11	6
B	8	0	13	16
C	4	7	0	9
D	5	12	2	0



Formula

$$\boxed{g(i, s) = \min \left[w(i, j) + g(j, \{s - j\}) \right]}$$

other Vertex

i = starting Vertex

s = Set of Vertexes that Salesman visit from Starting Vertices

$j \in s$ → other Vertex

In this problem

$i = A$

$$g(A, \{B, C, D\}) = \min \left[w(A, B) + g(B, \{C, D\}) \right.$$

Remaining except the B.

We have 3 options.

$$= \left[w(A, C) + g(C \{B, D\}) \right.$$

$$= \left[w(A, D) + g(D \{B, C\}) \right]$$

$$g(A, \{B, C, D\}) = \min \left[w(A, B) + g(B, \{C, D\}) \right] = 39$$

(1)

$$w(A, C) + g(C, \{B, D\}) = 39$$

(2)

$$w(A, D) + g(D, \{B, C\})$$

(3)

This will be recursive in Nature.

(i)

$$g(B, \{C, D\}) = \min \left[w(B, C) + g(C, \{D\}) \right] = 27$$

(22)

$$= w(B, D) + g(D \{C\}) \right] = 22 \text{ min}$$

$$g(C, \{D\}) = w(C, D) + g(D, \{\phi\})$$

$$= 9 + 5 = \boxed{14}$$

$$g(D \{C\}) = w(D, C) + g(C + \{\phi\})$$

$$2 + 4 = 6$$

(2)

any one of the set

$$g(C, \{B, D\}) = \min\left[\, w(C, B) + g(B, \{D\}) \;\; w(C, D) + g(D, \{B\}) \,\right]$$

$$7 + 21 = 28 \qquad 24 = 29$$

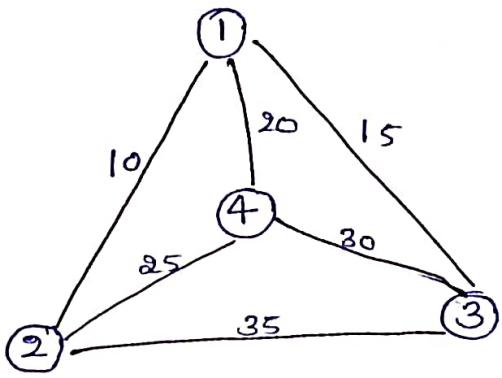
$$g(B, \{D\}) = \min\left[\, w(B, D) + g(D, \{\phi\}) \,\right]$$

$$16 + 5 = 21$$

$$g(D, \{B\}) = \min\left[\, w(D, B) + g(B, \{\phi\}) \,\right]$$

$$12 + 8 = 20$$

Travelling Salesman Problème



$$
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 10 & 15 & 20 \\
2 & 10 & 0 & 35 & 25 \\
3 & 15 & 35 & 0 & 30 \\
4 & 20 & 25 & 30 & 0
\end{array}
$$

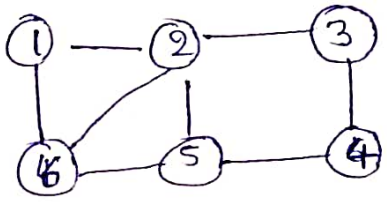
5. Backtracking & Branch-Bound.

Backtracking	Branch-Bound
non-optimisation problem	Optimisation problem.
DFS	Best-First Search.
Bounding Function is minimally used	Bounding function is used to find optimal solution
Ex: Hamitonian Circute Subset sum N-Queens	Ex: Assignment problem TSP 0/1 knapsack.

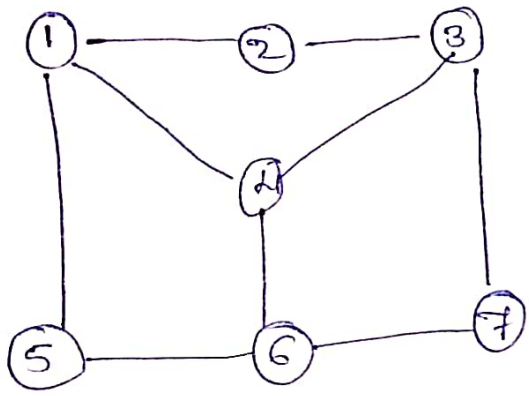
Quad → n^2
non-polinomial

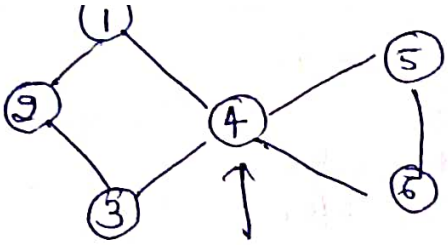
State Space tree → varies from problem to problem.

Hamiltonious circuit → No needs the wieghr for edges



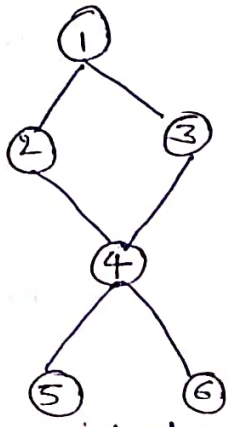
{1 2 3 4 5 6 1}
{6 5 4 3 2 1 2}





articulation point.

Hamiltonious circuit not possible in



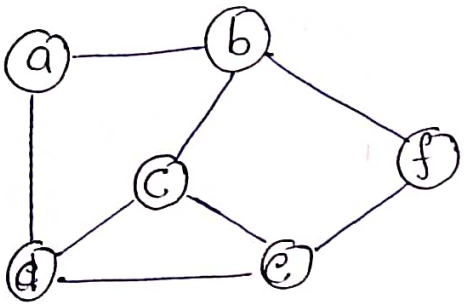
pendent nodes.

→ promising node

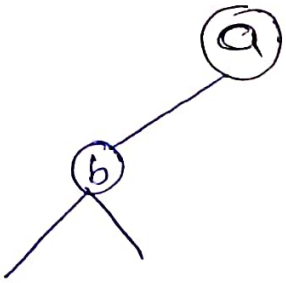
→ Non promising node.

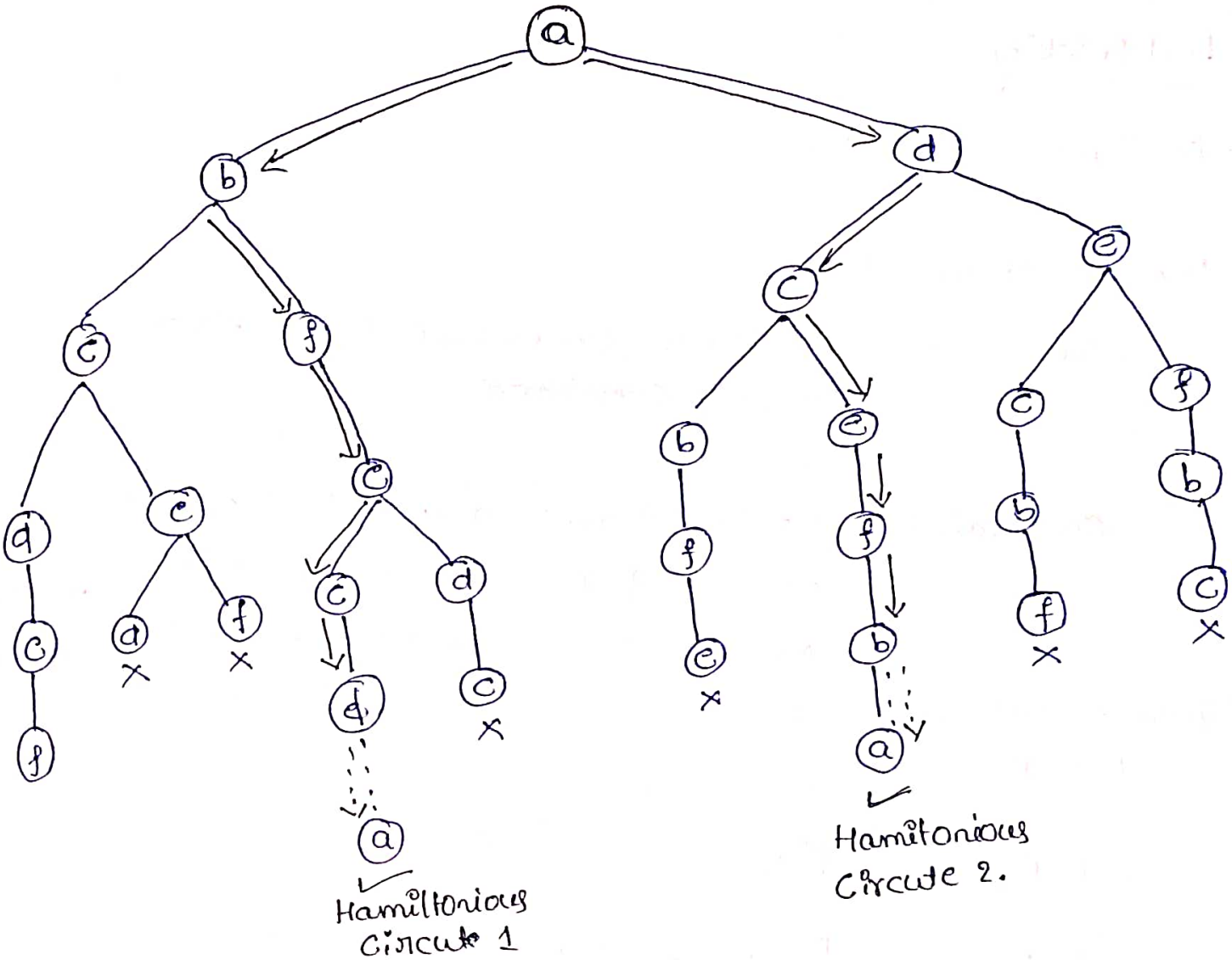
TC → is not polinomial

<u>Qusion</u> : Hamiltoneous circuite using backtracking.



assumming the Starting Vertex as (a)




a
b
d
c
f
c
e
b
e
c
f
a
c
c
d
b
f
b
c
f
c
a
f
c
d
c
e
f
b
f
a
a
Hamiltonious Circuit 1
Hamitonious Circuit 2.

Backtracking

N-Queens problem.

Problem Statement.

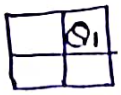
Given Data → 'n' no.of queens have to be placed on $n \times n$ chessboard

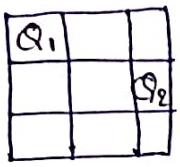
Constraints → No two queens attack horizontally
No two queens attack vertically
No two queens attack diagnolly < Left - Right, Right - Left

Different Types of Attack.

When n

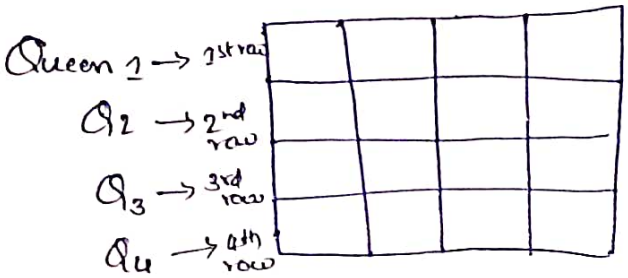
$n=1$ → Trivial Solution

$n=2$  → No Soln is posible

$n=3$ 

To place the Queens on $n \times n$ chess board

→ Horizonal or row attack.

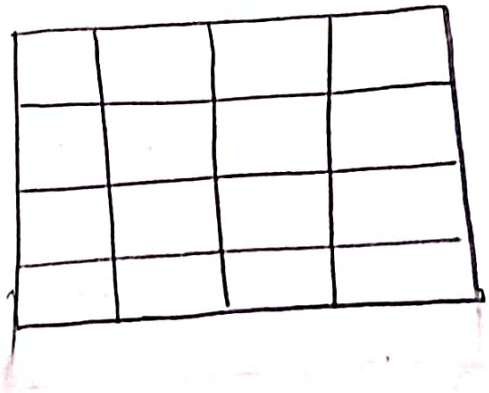


→ Column Attack

→ Diagnal Attack. — left top to Right bottom.

$i - x[i] = k - x[k]$

$i + k = x[i] - x[k]$ — ①



Secondary diagnal or Right top to Left bottom

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

$i + x[i] = k + x[k]$

$2 + 3 = 3 + 2$

$i - k = x[i] + x[k]$ — ②

Combining ① & ②

To Check for Diagnal Attack of two Queens

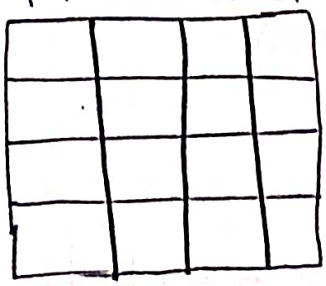
i & k

✓ $i - k = abs(x[i] - x[k])$ → ③

Turn the page horizontally

Statespace tree Representation for n-queens.

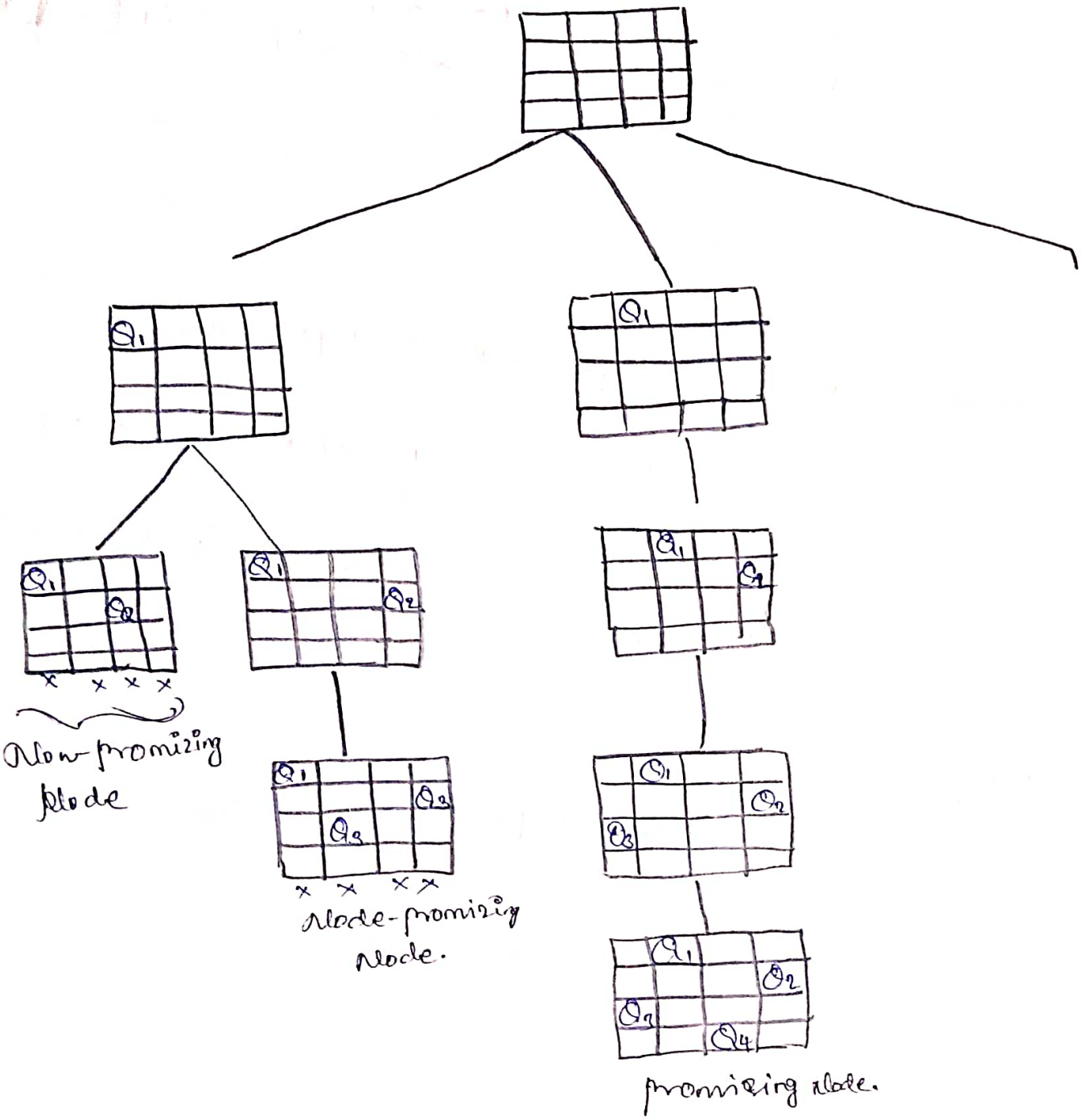
empty chess board.



4 Queens

Q_1 Q_2 Q_3 Q_4

We have place it



Algorithm place (x[ ], k)

// purpose. To check whether kth queen can be sucessfully placed on the chess board.

// Input. x constrains the column values for each of kth queens. & k is the current queen to be placed

// output. False : Queen can not be placed
True : Queen can be placed

```
for i ← 1 to k-1 //
  //check whether two queens attack vertically or Diagnolly.
  if(x[i]==x[k] || abs (i-k) = abs (x[i] -x[k]))
        return False //
  else
        retur true
```

any one of them or both of them

//Algorithm to check all ~~the~~ queens on the chess board.

```
k=1   // Select the first queen
x[k]=0 // But donot place in chess board
while (k!=0) // A queen exists
    x[k]+=1 // place the queen in next colum.
    while (x[k] ≤ n) && (!place(x, k))
        x[k]+=1
    end while
    if (x[k] ≤ n)
        if(k==n)
            print (x1, x2 .... xn)
        else
            k++;
            x[k]=0;
    endif
```

endWhile.

Sum of Subset problem.

Problem Statement:

A set consisting of 'n' elements

A sum value - d

Find all the possible subset such a way the the sum of elements of the subset is equal to the given sum 'd

Eg $S = \{1, 2, 3, 4, 5\}$ $d = 10$

Subset1 = $\{2, 3\ 5\}$

$2 = \{1, 4, 5\}$

$3 = \{1, 2, 3, 4\}$

$S = \{S_1, S_2, S_2 \ldots S_n\}$

$S_1 \leq S_2 \leq S_3 \leq \ldots \leq S_n$

Sum values = d

State Space tree

w or S = $\{3, 5, 6, 7, 8\}$

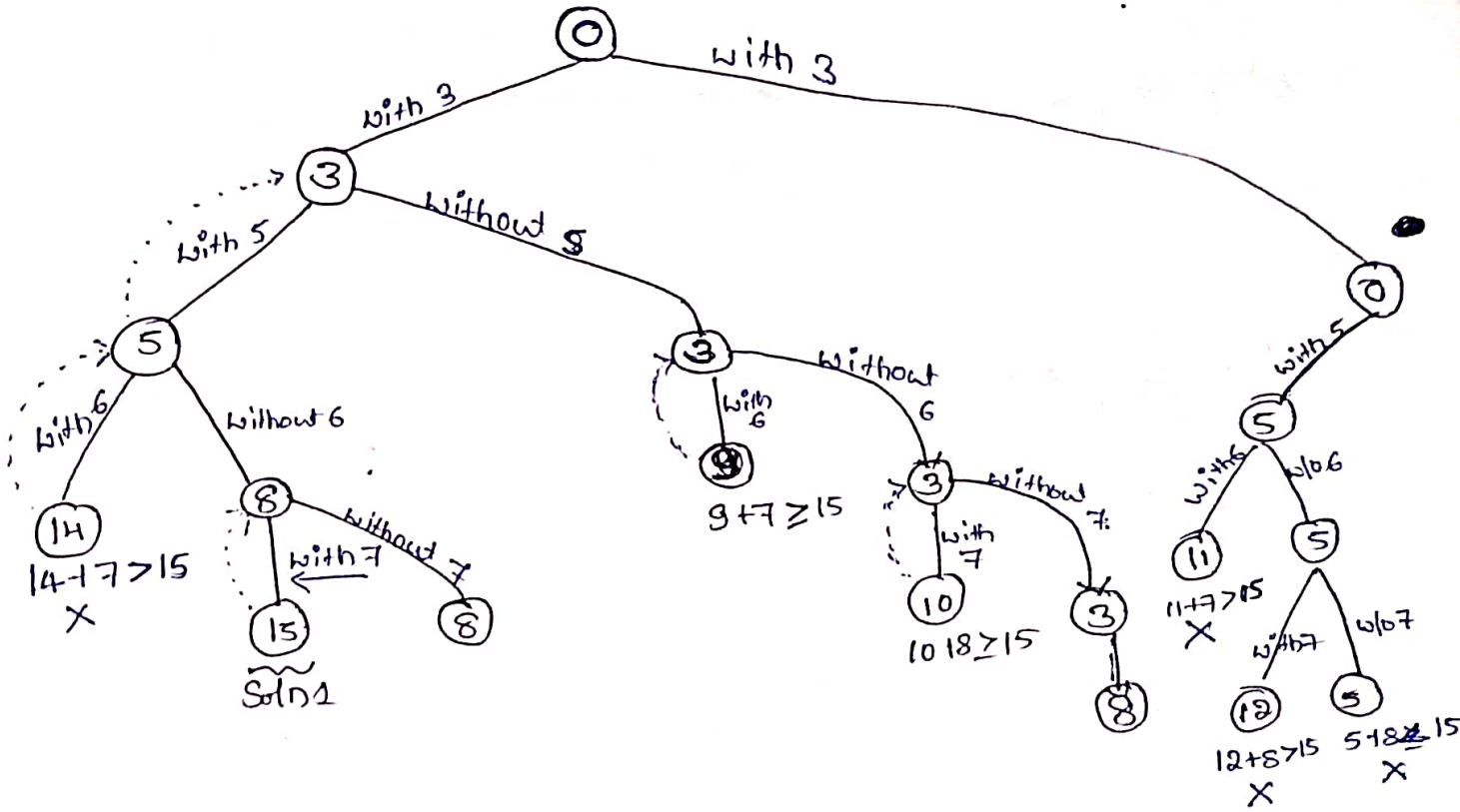
$d = 15$

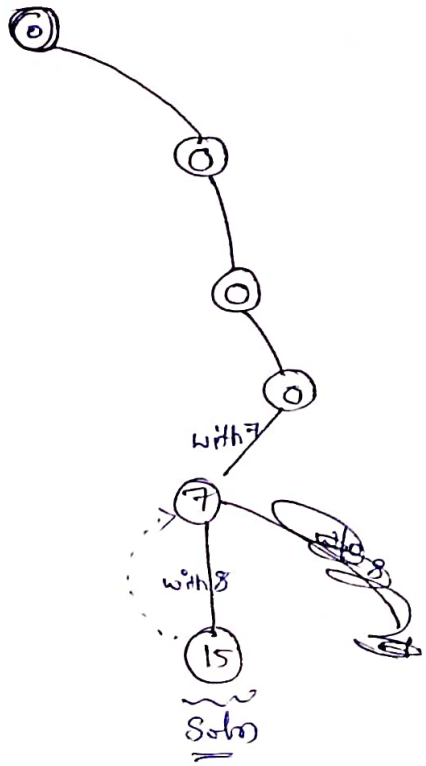
Node → Gives the sum value at the stage.

Alongedge → Represent the inclusion of element into subset

For a tree → inclusion of element

→ Without inclusion of element





Time Complexity 2^n

$S' + S_{i+1} > d$ $\longrightarrow$ Non promising nodes

Sum till the ith iteration

$S' + \sum_{j=i+1}^{n} < d$

Branch and Bound Techinques

↳ Assignment problem
↳ 0/1 knapsack problem
↳ TSP

BFS → Breadth First Search

→ Bound Values
- lower Bound - minimization problem
- upper Bound. - maximization problem.
  0/1 knapsack maximizing profit.

Assignment Problem using Branch & Bound Technique.

Problem Senario :1) There are n no. of Assigment & n. no. of tasks.

2) There is a Cost CIJ Associated with each task getting assigned to each assigning

assignee: $i = 1, 2 \ldots n$

Task: $j = 1, 2 \ldots n$

C_{ij} = Cost of assigning the task to an assignee

→ each assignee can carry out a single task.

Objective function: Determine the total cost of n assignments that has to be minimized

real world Exmple: ola, Uber, zomato etc

Problem

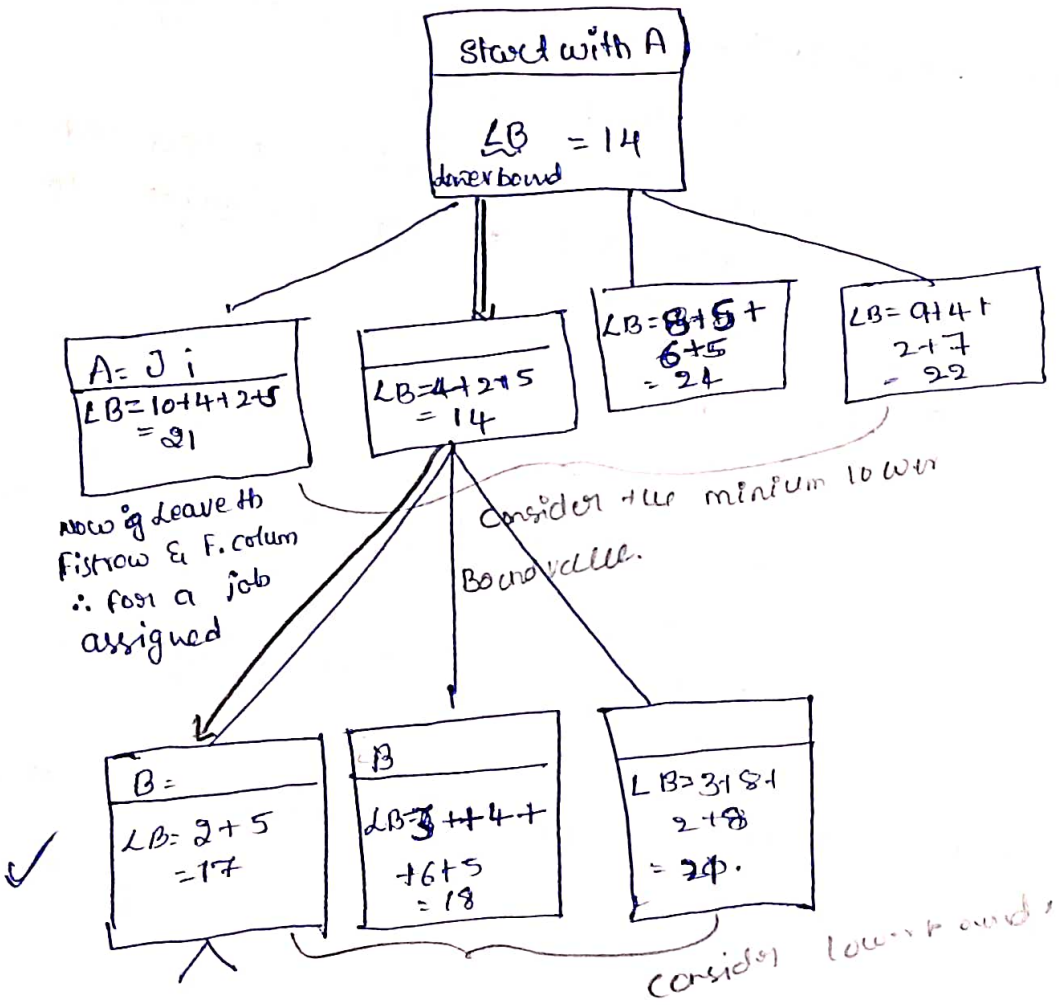
Assigned problem

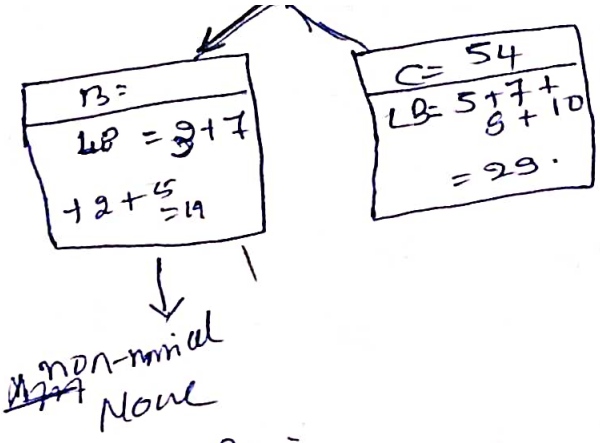
	Job1	Job2	Job3	Job4
A	10	3	8	9
B	7	5	4	8
C	6	9	2	9
D	8	7	10	5

Initial Lower Bound

take minimum job in each row

$3+4+2+5 = 14$





non-minimal
~~Min~~ None

optimal solution = Pa =

Pd = Job4

Minimum assignment = 17

TSP using Branch and Bound → Minimizing the cost | distance of tour

→ Find out the 2 nearest cities

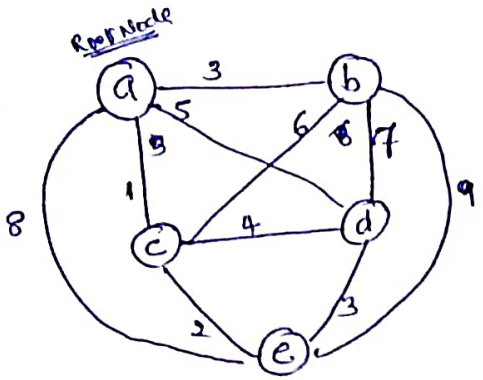
$\frac{2}{2}$ → to get initial lower bound value.

Lowerbound $= [S/2]$

$S =$ Sum of S_i

S_i is the sum of 2 nearest cities of value 'i'

$$S = \sum_{i=1}^{n} S_i$$

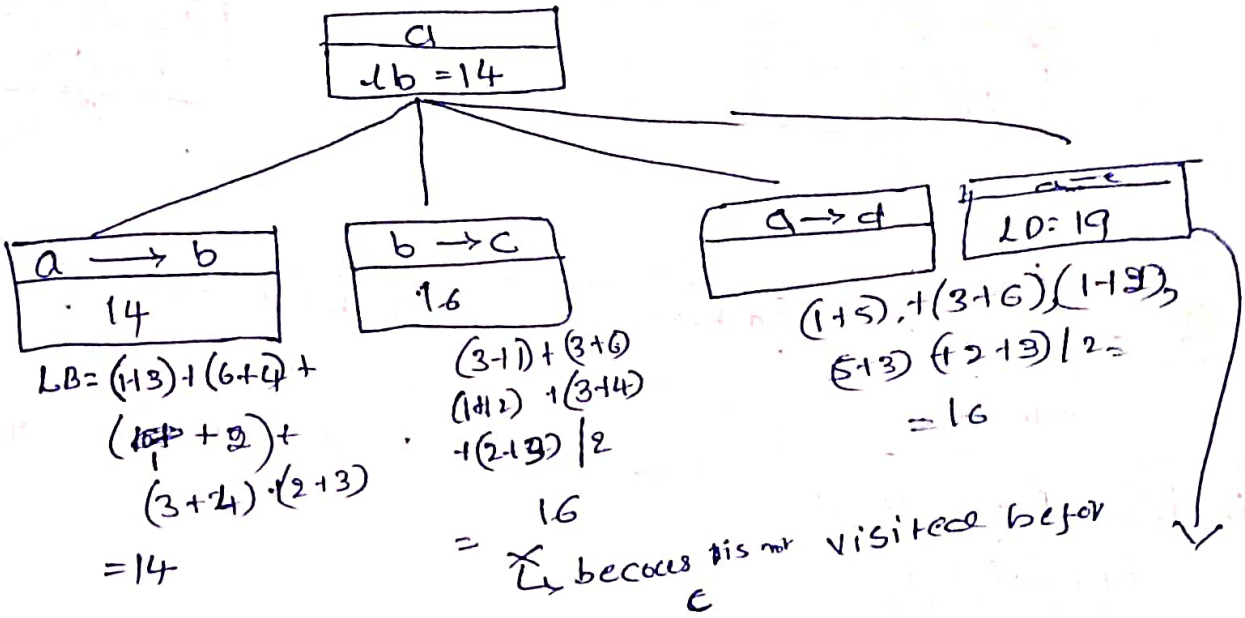


Initial Lower bound =

$= (1+3)$ for vertex a, $(3+6)$ b, $(1+2)$ c, $(3+4)$ d, $(2+3)$ e

$= \frac{28}{2} = \boxed{14}$ initial lower bound.

→ We have to come to source | Root vertex back



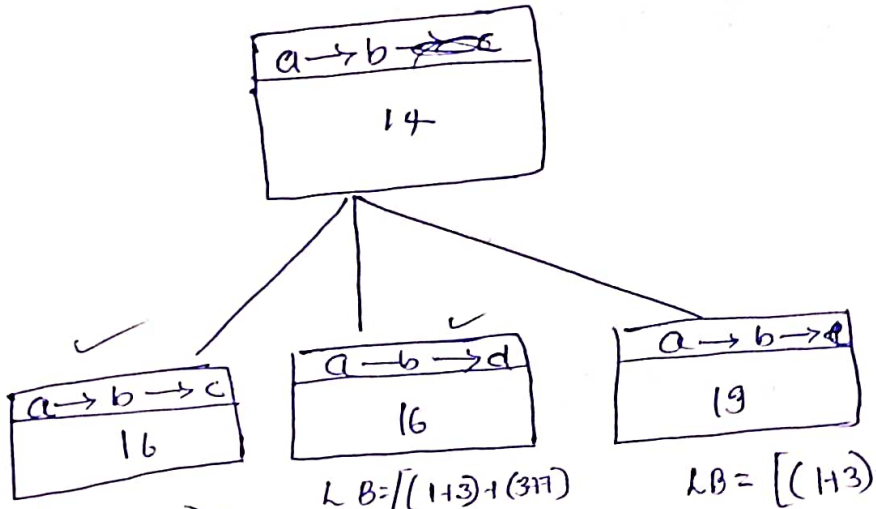
$LB = (1+3)+(6+4)+(1+2)+(3+4)+(2+3)$
$= 14$

$(3+1)+(3+6)+(1+2)+(3+4)+(2+3)/2$
16
$=$ ✗ becoues this not visited befor e

$(1+5)+(3+6)+(1+2)+(5+3)+(2+3)/2 =$
$= 16$

a → e

$LB = (1+8)+(3+6)+(1+2)+(5+3)+(8+2))/2 = 19.$

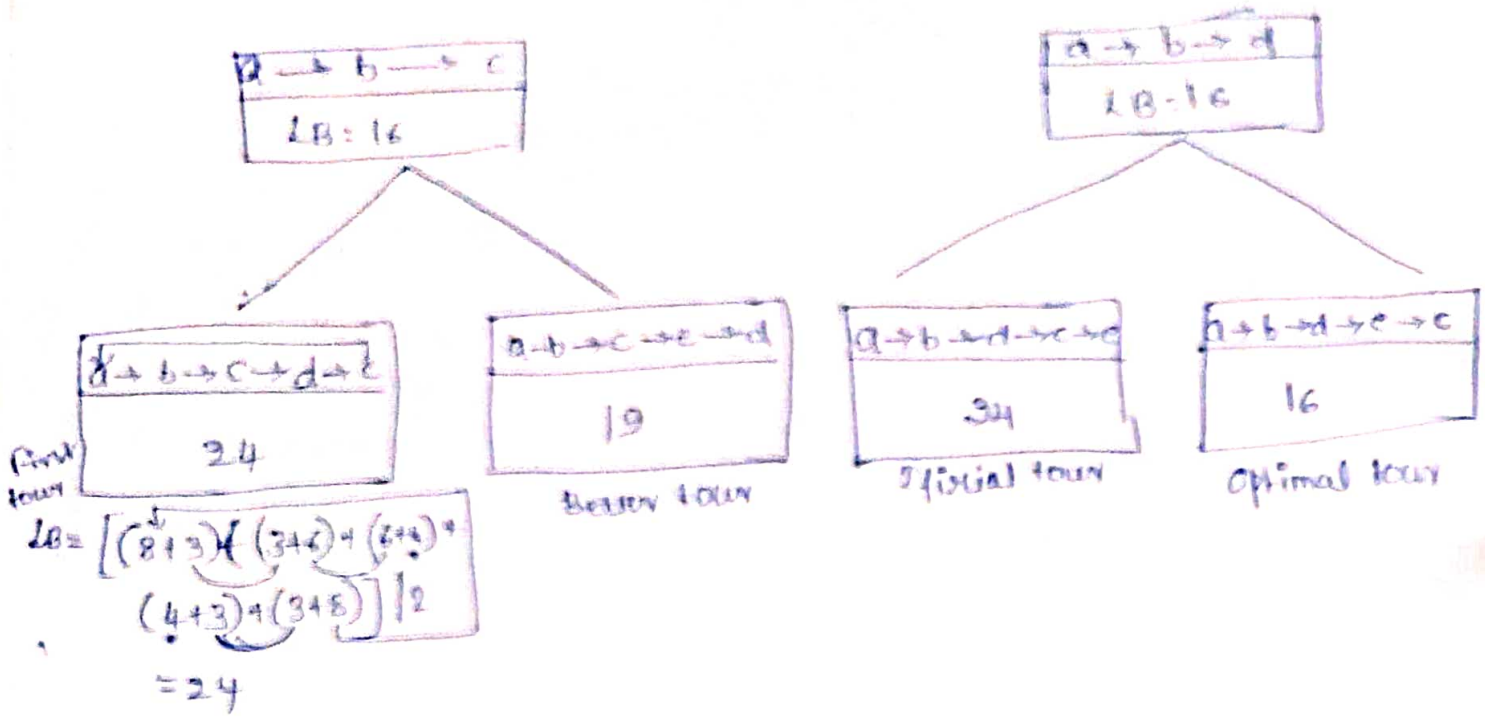


$LB = [(1+3)+(3+6)+(6+1)+(3+4)+(2+3)]/2$
$= 16.$

$LB = [(1+3)+(7+3)+(1+2)+(7+3)+(2+3)]/2 =$
$16.$

$LB = [(1+3)+(9+3)+(1+2)+(3+4)+(9+2)]/2 = 18.5$

✗



$LB = [(8+3)+(3+6)+(6+4)+(4+3)+(3+5)]/2$

$= 24$

* $a \to b \to c \to e \to d.$

$LB = (5+3)+(3+6)+(6+3)+(2+3)+(3+5)/2$

$= 38/2$

$= 19$

* $a \to b \to d \to c \to e$

$LB = (8+3)+(3+7)+(7+4)+(4+2)+(2+8)$

$= 48/2$

$= 24.$

* $a \to b \to d \to c \to e$

$LB = (1+3)+(3+7)+(7+3)+(3+2)+(2+1)$

$=$

Assignment Problem in Branch and Bound. → BFS

→ The objective of the assignment problem is set to assign 'n' jobs to 'n' persons such that the total cost of assignment is as small as possible
↳ Bounding function

1-1 mapping

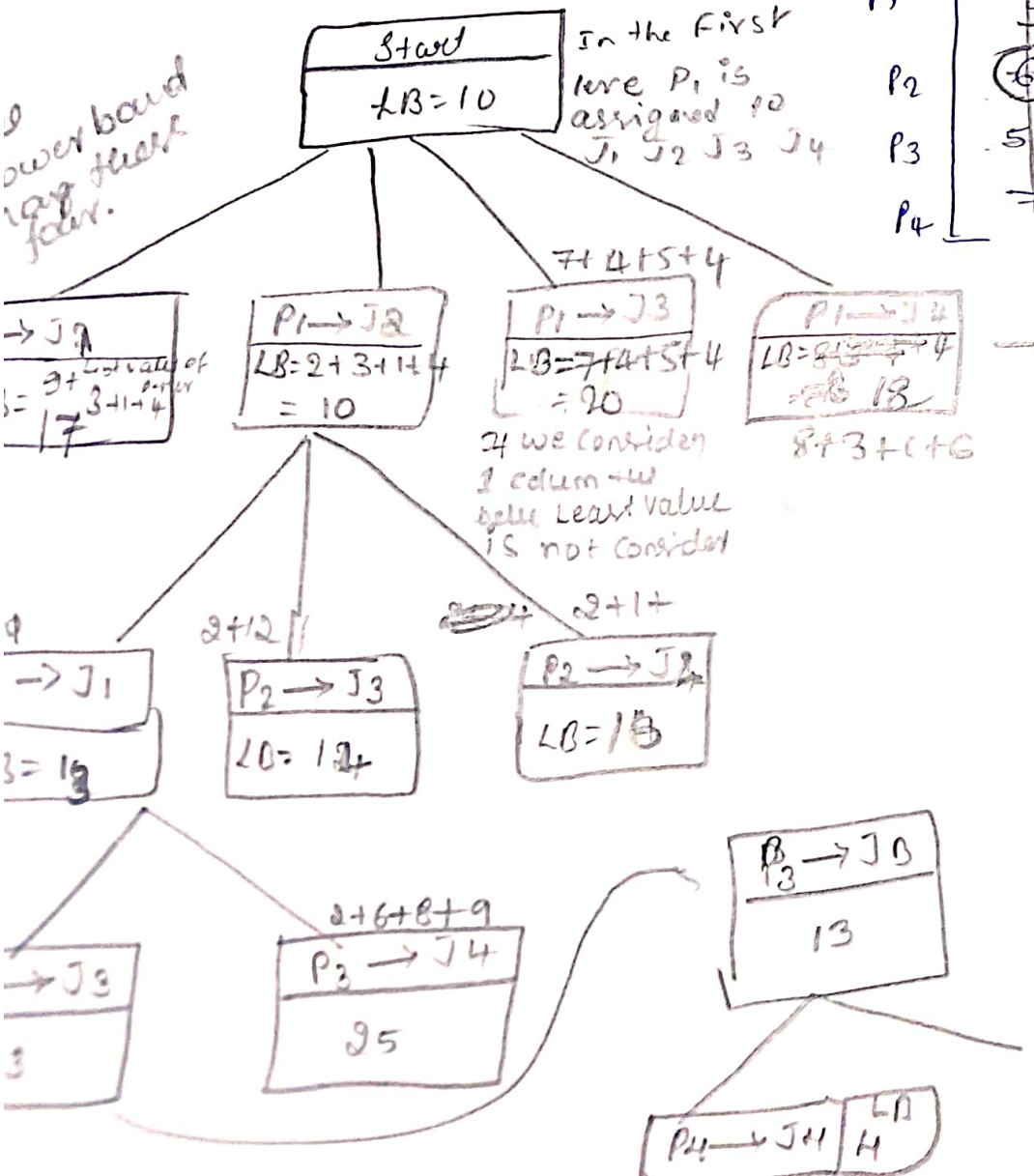
→ The assignment problem is a lower bound problem.

→ First we want to calculate lower bound

→ Lower bound is calculated as the summation of the lesser / least values in each row.

$LB = 2 + 3 + 1 + 4$
$= 10$

	J_1	J_2	J_3	J_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4



→ Here J_2 is least one hence the P_1 is assigned to J_2.
J_2 is assigned now & P_1, Now 2 is common for all

Note: If job is assigned we can't touch that row & columns

Delete
Backspace
End

Assignment Problem in Branch and Bound. → BFS

→ The objective of the assignment problem is set to assign 'n' jobs to 'n' persons such that the total cost of assignment is as small as possible
↳ Bounding function

1-1 mapping

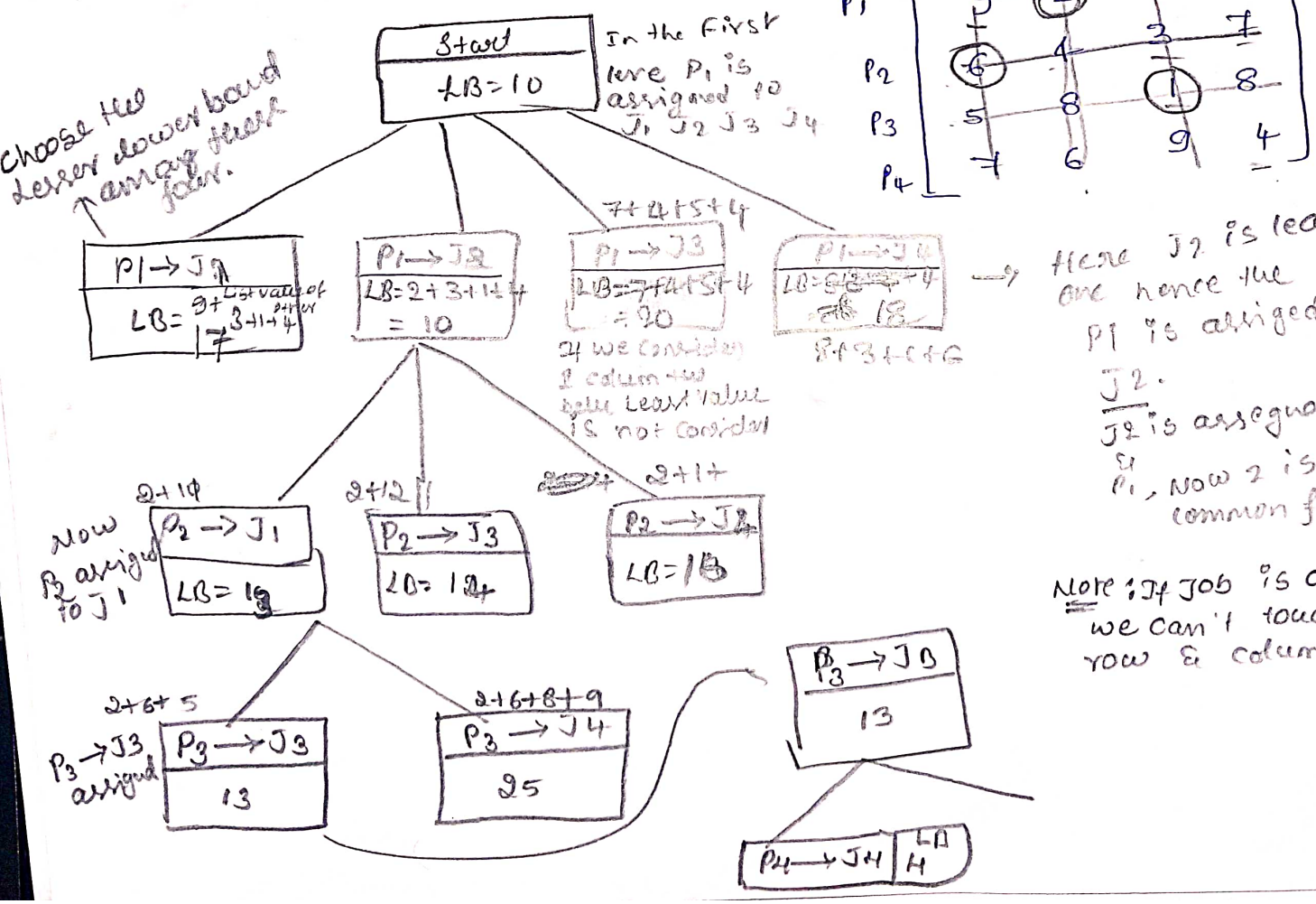
→ The assignment problem is a lower bound problem.

→ First we want to calculate lower bound

→ Lower bound is calculated as the summation of the lesser / least values in each row.

$LB = 2 + 3 + 1 + 4$
$= 10$

	J_1	J_2	J_3	J_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

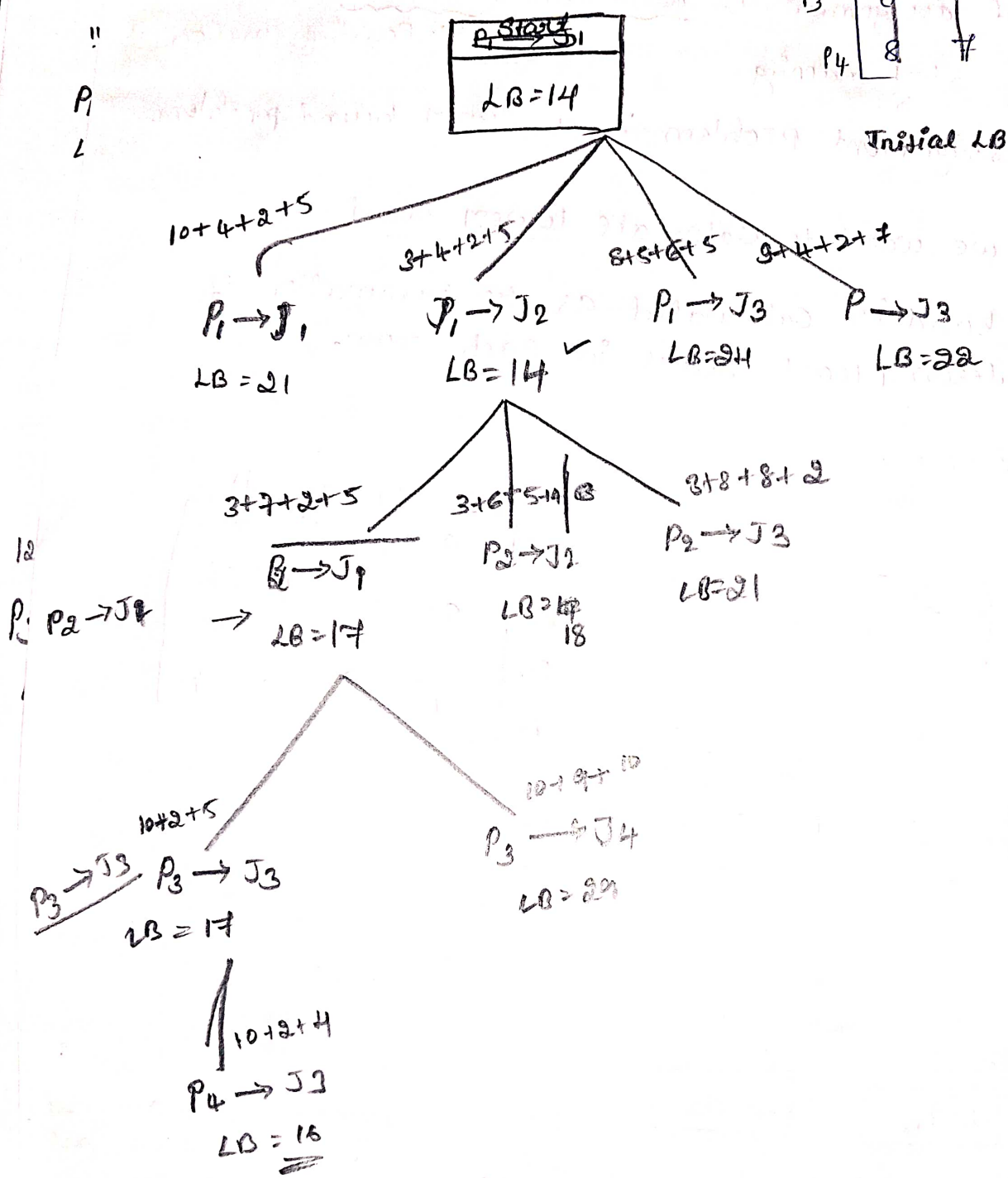


Job1 Job2 Job3 Job4

Class Notes

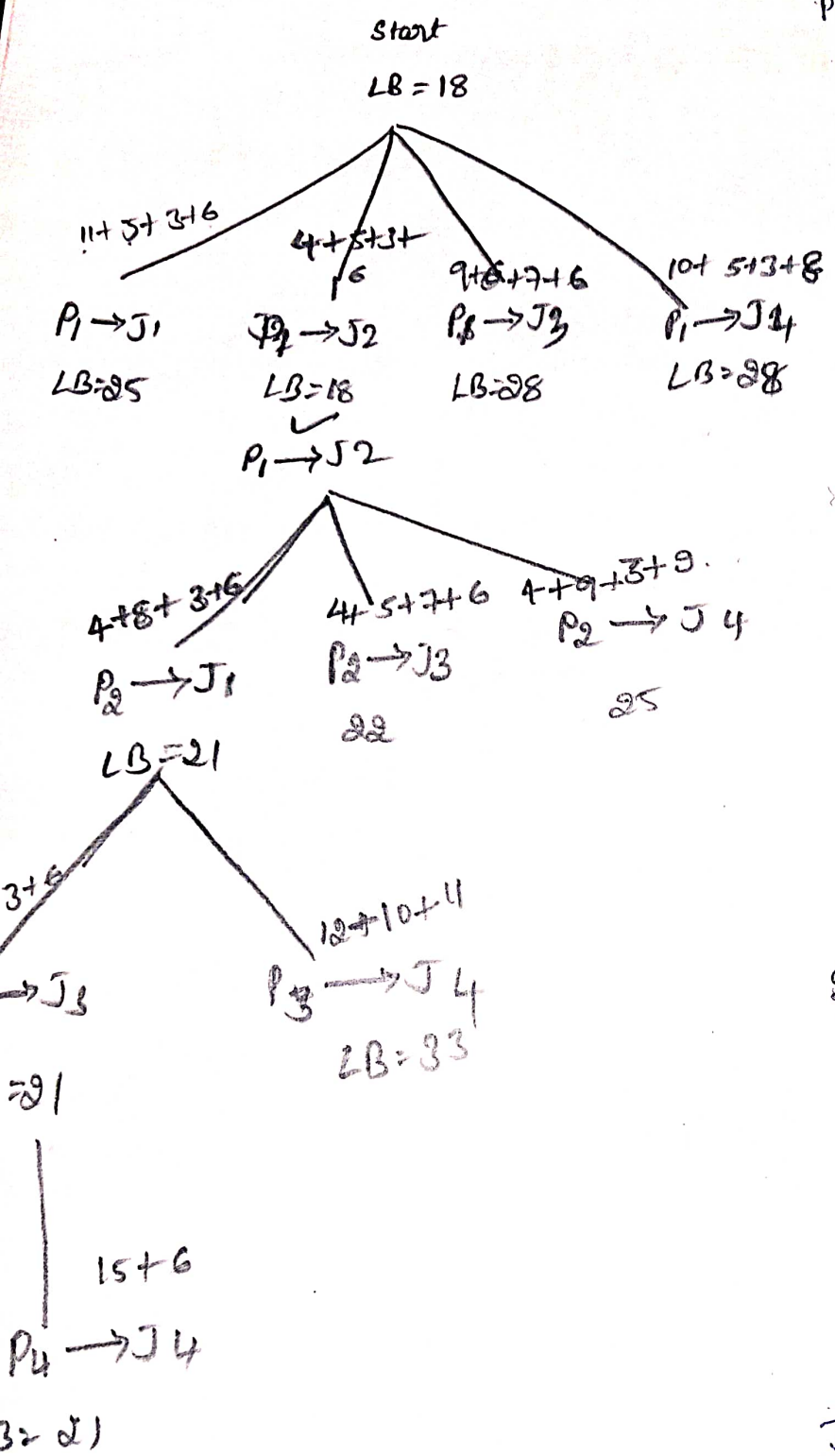
	J1	J2	J3	J4
P_1	10	3	8	9
P_2	7	5	4	8
P_3	6	9	2	9
P_4	8	7	10	5

Initial LB = 3 + 4 + 2 + 5
= 14



	Job1	Job2	Job3	Job4
P1	11	4	9	10
P2	8	6	5	9
P3	7	10	3	10
P4	9	8	11	6

Starting LB = 4 + 5 + 3 + 6.
= 18



Solution

$P_1 \rightarrow 4$

$P_2 \rightarrow 8$

$P_3 \rightarrow 3$

$P_4 \rightarrow 6$

Total cost = 21 is the

TSP using branch and Bound.

→ The objective of the TSP is to find a route such that a sales person visit all the cities and return back to the same city from where he started the tour with the minimum cost.

→ TSP is a minimization problem, hence we are supposed to find the lower bound. → Bounding function is lower bound.

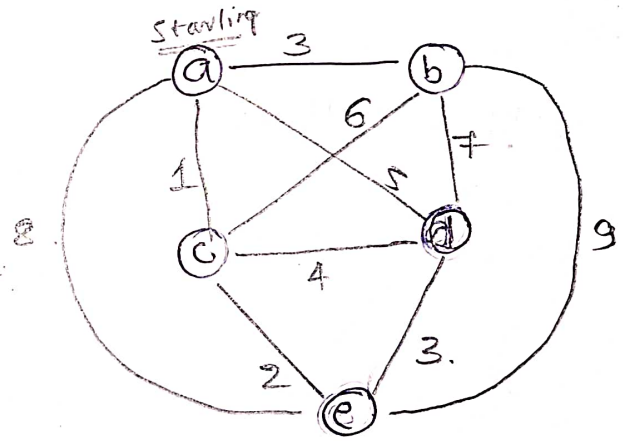
→ The lower bound (lb) is be denoted as $\left\lceil \frac{s}{2} \right\rceil$

s = the summazation of 2 closest cities of a city for all cities.

$$lb = \frac{\overset{(a)}{\underbrace{1+3}_{\substack{a\to b \\ a+c}}} + \overset{(b)}{\underbrace{3+6}_{\substack{b\to a \\ b+c}}} + \overset{(c)}{1+2} + \overset{(d)}{4+3} + \overset{(e)}{2+3}}{2}$$

$$= \frac{28}{2}$$

$$\boxed{\text{Initial} \quad lb = 14}$$



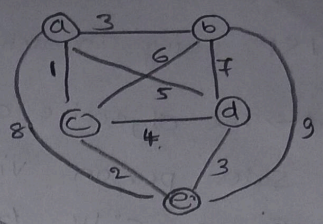
umption

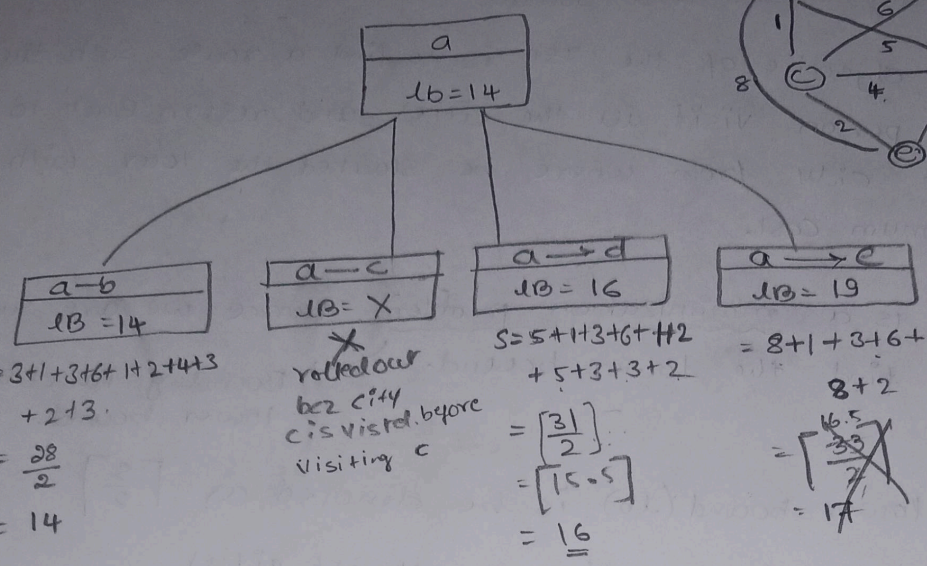
. (a) is started city

. city (b) is visited befor visting (c)

The cost of Tour is = a — b — d — e — c — a

TSP-B&B-②





$S = 3+1+3+6+1+2+4+3+2+3$

$= \frac{28}{2}$

$= 14$

rejected our bcz city c is visted before visiting c

$S = 5+1+3+6+1+2+5+3+3+2$

$= \left\lceil \frac{31}{2} \right\rceil$

$= \lceil 15.5 \rceil$

$= 16$

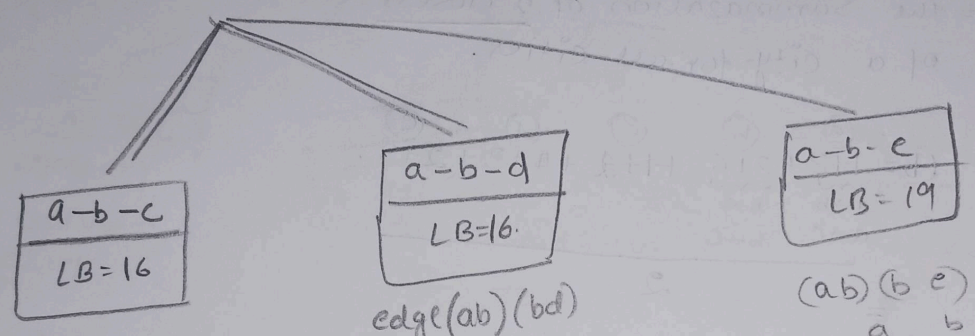
$= 8+1+3+6+1+2+4+3+8+2$

$= \cancel{\left\lceil \frac{33}{2} \right\rceil}$

$= \cancel{17}$

$= \frac{38}{2}$

$= 19$



edge (ab) (b c)

a b c d e

$S = 3+1+3+6+6+1+3+4+2+3$

$= \frac{32}{2}$

$= 16$

edge (ab) (bd)

a b c d e

$S = 3+1+3+7+1+3+7+3+2+3$

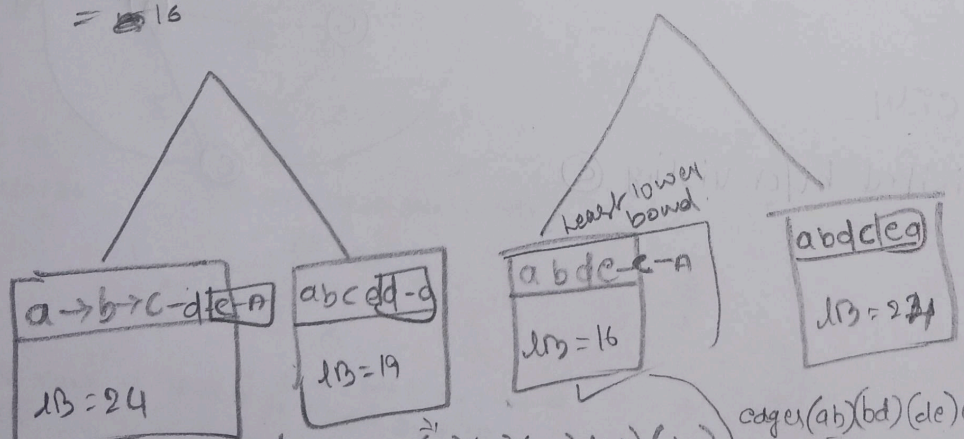
$= \frac{32}{2}$

(ab) (b e)

a b c d e

$S = 3+1+3+9+1+2+4+3+9+2$

$= \frac{37}{2} = \lceil 18.2 \rceil$



→ get the left out city.

edges (ab)(bc)(cd)(de)(ea)

a b c d e

$S = 3+8\ 3+6+6+4+4+3+3+8$

$= 48/2 = 24$

edges (ab)(bc)(ce)(ed)(da)

a b c d e

$S = 3+5\ 3+6+\ 6+2+3+5+8+3$

$= \frac{38}{2}$

$= 19$

edges (ab)(bd)(de)(ec)(cA)

a b c d e

$S = 3+1+3+7+2+1+7+3+3+2$

$= \left\lceil \frac{32}{2} \right\rceil$

$= \frac{6}{2}$

edges (ab)(bd)(de)(ce)(cA)

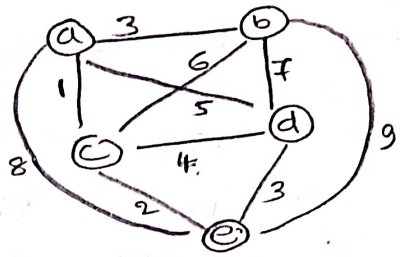
a b c d e

$S = 3+8\ 1+3+7+2+4+7+4+2+8$

$= \frac{48}{2}$

$= 24$

The cost of Tour is: a — b — d — e — c — a



a
lb = 14

- **a—b**, lB = 14
  - S = 3+1+3+6+1+2+4+3 +2+3.
  - = 28/2
  - = 14
- **a—c**, lB = X
  - rejected our bcz city c is visited before visiting c
- **a → d**, lB = 16
  - S = 5+1+3+6+1+2 + 5+3+3+2
  - = [31/2]
  - = [15.5]
  - = 16
- **a → e**, lB = 19
  - = 8+1+3+6+1+2+4+3+ 8+2
  - ~~= [33/2] = 17~~
  - = 38/2, = 19

a—b—c, LB = 16
- edge (ab) (b c)
- a b c d e
- S = 3+1+3+6+6+1+3+4+2+3
- = 32/2
- = 16

a—b—d, LB = 16.
- edge (ab) (bd)
- a b c d e
- S = 3+1+3+7+1+2+7+3+2+3
- = 32/2

a—b—e, LB = 19
- (ab) (b e)
- a b c d e
- S = 3+1+3+9+1+2+4+3+9+2
- = 37/2 = [18.5]

a → b → c — d — e — a, lB = 24
- edges (ab)(bc)(cd)(de)(ea)
- a b c d e
- S = 3+8 3+6+6+4+4+3+3+8
- = 38/2 = 24

abcdd-e, lB = 19

a b d e — c — a, lB = 16
- Lower bound

a b d c e a → get the left out city
- edges (ab)(bd)(dc)(ce)(ea)
- a b c d e
- S = 3+1+3+7+2+1+7+3+3+2
- = [32/2]
- = 16

edges (ab)(bc)(ce)(ed)(da)
- a b c d e
- S = 3+5 3+6+ 6+2+3+ 5+2+3
- = 38/2,
- = 19

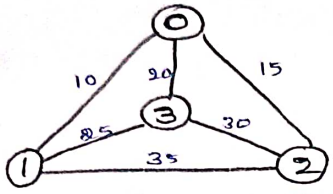
edges (ab)(bd)(de)(ec)(ca)
- a b c d e
- S = 3+8 +3+7+2+4+7+4+2+8
- = 48/2
- = 24

The cost of Tour is: a — b — d — e — c — a

$= 3 + 7 + 3 + 2 + 1$

$= \underline{\underline{16}}$

2)



Initial lower bound = Adding the two shortest edges connecting to each vertex to the MST / 2

$= \frac{(10+15)+(10+25)+(15+30)+(20+25)}{2}$

$= \frac{150}{2} = 75$

0 → 0

Starting
LB = 75

0 − 1
LB = 75

edge(0 − 1)

$S = 25+35+45+45$

$= \frac{150}{2} = 75$

0 − 3
LB = 78

cd(0 − 3)

$= (20+10)+45+35+45$

$= \frac{155}{2} = 78$

0 − 2
LB = 75

cd(0 − 2)

$= 15+30+45+35+45$

$= \frac{150}{2}$

$= 75$

0 − 1 − 2 − 3 − 0
LB = 85

cd(01)(12)(23)(30)

0 1 2 3 − 0

$= 10+20+10+35+35+30+30+20$

$= \frac{190}{2} = 95$

X

0 − 1 − 3 − 2 − 0
LB =

cd(01)(13)(32)(20)

0 1 2 3

$S = 10+15+10+35+30+15+35+30$

$= \frac{180}{2} = 90$

X

0 − 2 − 1 − 3 − 0
LB =

cd(02)(21)(13)(30)

0 1 2 3

$S = 15+25+35+15+35+25+20$

$= \frac{190}{2}$

X

0 − 2 − 3 − 1 − 0
LB = 80

cd(02)(23)(31)(10)

0 1 2 3

$S = 15+10+10+25+15+30+30+25$

$= \frac{160}{2} = 80$

✓

The cost of Tour = 0 − 2 − 3 − 1 − 0

$= 15+30+25+10$

$= \underline{\underline{80}}$