DAA.

I. Algorithm:

A step by step process performing some action

↓ i/p list

> A step by step method for solving a problem or doing a task.

↓ o/p list.

Common terms :-

1) Variables :- Specific loc in comp mem used to store 1 and only 1 value
2) Data types :- Set of variables takes its values
3) Statements :- line of codes.

II. Properties of Algorithm / Characteristics

1. input :- Alg uses values from a specific set Σ
2. output :- for each i/p the alg produces values from a specific task.
3. precision :- Steps are precisely defined
4. correctness :- i/p is defined for that o/p
5. fitness :- is correct, clear & unambiguous
   ↳ o/p after finite no of steps for each i/p
6. determination :- Result should be guarentee
7. Generality :- Procedure apply to all problem not a special subset.
8. effectiveness :- can be paper, effective

III Algo can be expressed in diff notations like

* Natural lang
* flow chart
* Pseudo code
* Programming lang.

IV Performance Analysis.

* By using space complexity and time complexity we can analyse the performance analysis
* An Algorithm is said to be effficient & fast if it takes less time to execute and consumes less memory space

① Space Complexity :-

* Amount of memory space required by an algorithm during course of execution.
* The algorithm generally requires space for

a) Instruction space :- Space required to store executable version of a program It depends on the no. of lines taken to execute a program.

b) Data Space :- The space required to store all the constants & variable values

c) Environment Space :- The space required to store the environment information needed to resume the

Suspended function

* The Space complexity can be calculated in two ways.

a) Constant Space complexity :-

Here, the algo requires fixed amt of space for all the i/p values so the space complexity is constant.

Eg:-
```
int Square (int a)
{
  return a*a;
}
```

b) Linear Space complexity :-

The space needed for a algo is
- Size variable 'n' = 1 word.
- array 'a' values = n word.
- loop variable 'i' = 1 word.
- Sum variable is = 1 word.

Eg:-
```
int sum (int a[], int n)
{
  int sum = 0, i;
  for (i=0; i<n; i++)
    sum = sum + a[i];
  return sum;
}
```

∴ The linear space complexity of the above example is (n+3) words

* Here the algo requires no fixed amount of space as, if the input size increases the CST will also increase.

② Time complexity:

* The total amount required by an algo to complete its execution.
* The time complexity can be calculated in 2 ways.

a) Constant time Complexity:

If a program requires fixed amount of time for all inputs values

Eg:-
```
int sum(int a, int b)
{
    return a+b;
}
```

b) Linear Time Complexity:

If the input values increases, then the time complexity will change.

~~Eg:-~~
- comments = 0 steps.
- Assignment stmt = 1 step.
- Condition stmt = 1 step.
- loop conditions for { = n+1 ~~step~~.
  n times
- body of the loop = n steps.

V. Issues in Study of Algorithm

(1) How to devise Algorithms
1) Divide & Conquer
2) Greedy
3) DP
4) Branch & Bound
5) Backtracking

(2) How to validate an Algo?
checking if it's producing correct results or not

(3) Analysis of Algorithm How?
How much computing time & storage space is req.

(4) How to test a program?
Using prog lang, writing expressions for instruction
Here testing requires — debugging
performance measurement (or) profiling.

∴ By considering these 4 issues, we can study the algorithm.

VI. Types of Time Complexities

There are basically 3 types of time complexities.

a) best case TC : If an algo requires minimum amt of time for its execution

b) worst case TC : maximum amt of time

c) Average TC :- Average amt of time

eg:- Linear Search, while searching for an element.
If the key element found is in
1st pos then - best case, last pos - worst
case.

VII. Approaches to calculate Time Complexities
Inorder to calculate TC, there are
2 approaches
1. Frequency count / Step count
2. Asymptotic notations -

Frequency count or Step Count:

→ It Specifies the no of times a statement to be executed.

→ Here we can give the count val dep on stmt.

→ We have 4 rules,

1. For comments & declarations, the step count is '0'
2. For return stmts and assignment stmts the step count is '1'
3. Ignore lower order exponents when higher order exponents are present.
   Eg:- $3n^4 + 4n^3 + 10n^2 + n + 10c$, here only take $3n^4$.
4. Ignore constant multipliers
   Eg. $3n^4 \rightarrow O(n^4)$.

Eg:- let's calculate the TC:

```
int sum (int a[], int n).
{
  S=0;                    → 1.
  for (i=0; i<n; i++)     → n+1
        S=S+a[i];         → n.
  return S; → 1
}
```

Eg n=3.
0<3 → 1
1<3 → 2.
2<3 → 3

$1+n+1+n+1 \Rightarrow 2n+3$.

$\therefore O(n)$.

n+1
n(n+1).
n(n+1)(n+1)
n*n

Lets calculate Space Complexity of the same example.

a → n
n → 1
S → 1
i → 1.

n+3

∴ The SC is O(n).

TC

eg 2). void mat_mul (int a[][], int b[][]).

```
{
  for (i=0; i<n; i++).  ⟶ n+1.
  {
    for (j=0; j<n; j++)  ⟶ n*(n+1)
    {
      c[i][j]=0;.  ⟶ n*n
      for (k=0; k<n; k++)  ⟶ n*n*(n+1)
      {
        c[i][j]+=a[i][k]*b[k][j];  ⟶ n*n*n.
      }
    }
  }
}
```

2n³ + 3n² + 2n + 1.
= O(n³).

SC

a → n²
b → n²
c → n².
n → 1

i → 1
j → 1
k → 1
∴ 3n² + 4 ⇒ O(n²)

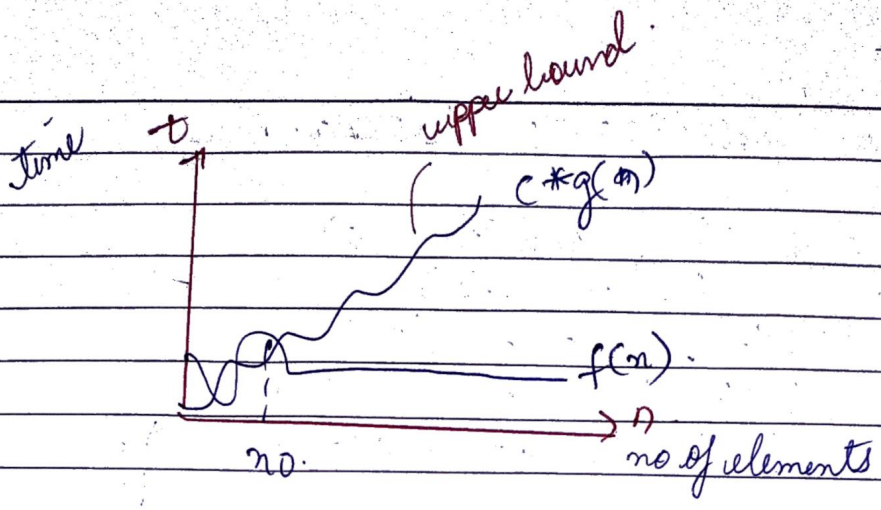
II). ASSYMPTOTIC NOTATIONS:

1. Big Oh Notation (O)
2. Big Omega (Ω)
3. Theta Notation (Θ)
4. Little Oh Notation (o)
5. Little Omega Notation (ω)

* By using Assymptotic notations, we can calculate the best case TC, worst case TC & avg case TC of an algo
* Best case: If an algo req min time for its execution
  Worst case: If an algo takes max time for its execution
  Avg case: If an algo takes avg time for its execution

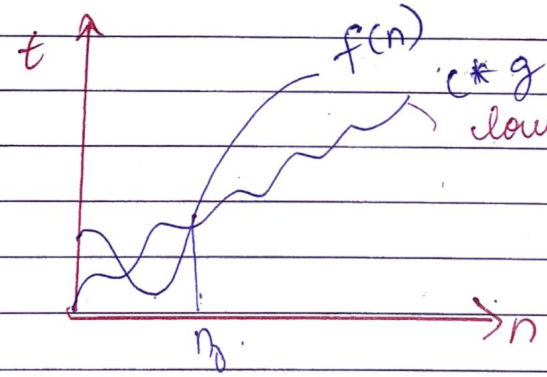
1. BIG Oh NOTATION:

* It represents the upper bound algorithms running time
* By using Big Oh Notation we can calculate maximum amt of time for its execution (worst case)
* Defination: Let $f(n), g(n)$ be 2 non-ve. Integers functions over $f(n) = O(g(n))$ if there exists 2 true constants C, n_0 such that $f(n) \leq C * g(n)$, $\forall\ n > n_0$.
* Graphical representation



2. Big Omega Notation :- (Ω) ... running time

* It represents lower bound of the algo.
* We can calculate minimum amt of time for its execution (best case).
* ∴ We can calculate best case TC
* Defination :- Let $f(n)$, $g(n)$ be 2 non-ve functions, over $f(n) = \Omega(g(n))$ if there exists 2 +ve constants C, n_0 such that $f(n) \geq C * (g(n))$, $\forall\ n > n_0$.
* Graphical representation of Big Omega.

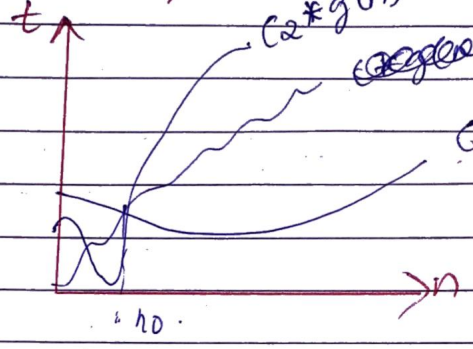


3). Theta Notation :- (Θ) ... running time

* It represents average bound of algo.
* We can calculate average amt of time for its execution.
* ∴ By using Theta Notation, we can calculate avg case TC

* Defination :- Let $f(n), g(n)$ be 2 non-ve functions over $f(n) = \Theta(g(n))$ if there exists 2 +ve constants c_1, c_2, n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ $\forall n > n_0$.

* Graphical representation of Θ notation.



4. Little Oh Notation (o) :-

Defination :- Let $f(n)$ & $g(n)$ be 2 non-ve functions then $f(n) = o(g(n))$ such that

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

5. Little Omega Notation :- (ω).

Let $f(n)$ & $g(n)$ be 2 non-ve func then $f(n) = \omega(g(n))$ such that

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0 \quad \text{or} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

VIII. SPACE COMPLEXITY OF AN ALGO IN DS.

* Space complexity means, the amt of space that an algo requires for its execution.
* The algo which requires less space, that algo becomes the best algo. So we need to develop a program based on that algo.
* Formula :- $S(P) = C + S_p$

where,

S = space complexity
P = problem.
C = constant part / Independent part.
S_p = instance characteristics / dependent part / variable part.

* Ex :-

```
Algorithm sum(a,b,c)
{
  a=10;
  b=20;
  c=a+b;
}
```

By formula $S(P) = C + S_p$.

C =

a → 1
b → 1
c → 1

$\therefore S(P) = 3+0$

= 3 time units $\therefore$ O(1)

IX. VARIOUS TYPES OF COMPUTING TIMES.
(OR).
TYPICAL GROWTH RATES.

$O(1)$ → Constant Computing time
$O(n)$ → Linear " "
$O(n^2)$ → Quadratic " "
$O(n^3)$ → Cubic " "
$O(2^n)$ → Exponential " "
$O(\log n)$, $O(n \log n)$ → logarithmic " "

n	n^2	n^3	2^n	$\log n$	$n \log n$
1	1	1	2	0	0
2	4	8	4	1	2
4	16	64	16	2	8
8	64	512	256	3	24

$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n^3)$.

X. DIVIDE AND CONQUER.

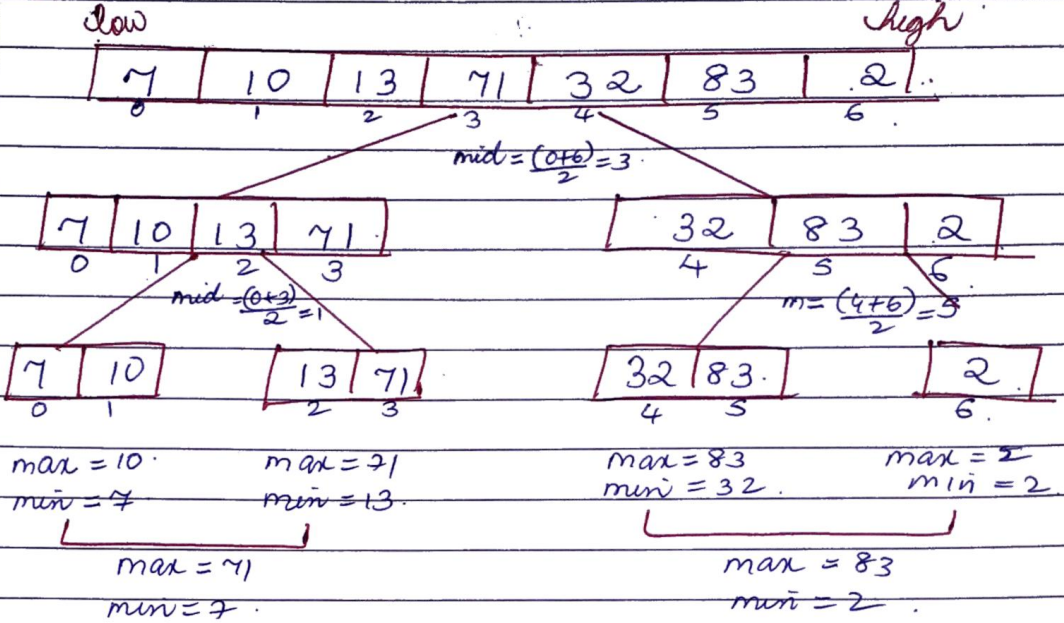
Step 1 : Divide
* The given array is divided into 2 parts
Step 2 : Conquer
* Each subproblem is solved recursively
Step 3 : Combine
* The solutions of the subproblems are combined

4. FINDING MAXIMUM & MINIMUM ELEMENT USING DIVIDE AND CONQUER ALGO.

* Let us take an array of size 7 elements
* The first element is treated as low and the last is treated as high
* Divide the array into 2 parts by calculating mid value.
  $mid = \frac{low + high}{2}$.
* Now we have 2 lists.
* the first list ranges from low to mid value.
* the second list ranges from mid+1 to high value.
* Now we have to conquer, which means to solve each list recursively.
* So again the first list is divided into 2 sub lists where the first sub list ranges from 1 to mid & the second sublist ranges from mid+1 to high.
* So again the second list is divided
* we have to solve the problem as long as the list contains either 1 or 2 elements
* Then we compare the max element with max element of the sublist1. then the min" with min element of sublist1, the same applies for sublist2

__/__/____

Ex:



Algo:

```
Algorithm maxmin (low, high, max, min)
{
  if (low == high).  // if list contains 1 element.
                     // then a[0] becomes both max & min
  {
    max = min = a[low];
  }
  else if (low == high-1) // if list contains 2 elements
  {
    if (a[low] > a[high])
    {
      max = a[low];
      min = a[high];
    }
    else {
      max = a[high];
      min = a[low]
    }
  }
```

1) Explain Divide and Conquer Technique?

* Divide and conquer is a general algorithm design paradigm.

Divide : Divide the given input problem 'P' into k distinct subproblem $1 \le k \le n$.

Recur: Solve the k distinct subproblems recursively

Conquer : The solution of the orginal problem is then forwarded from the solutions of the subprogram.

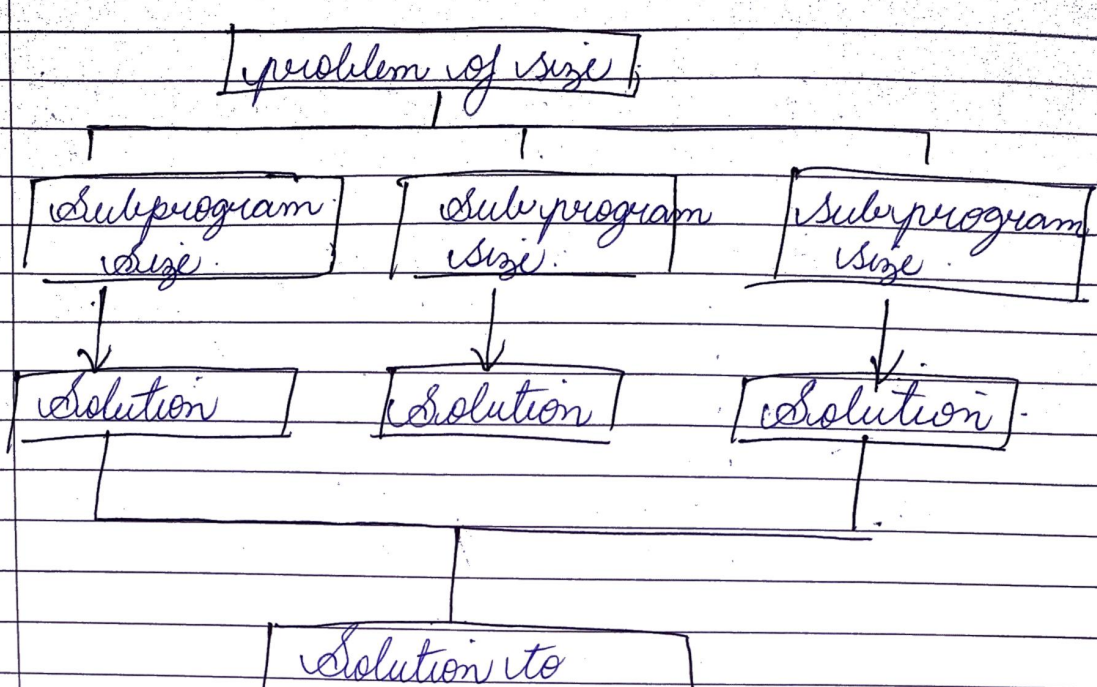
*The program is divided into smaller subprograms (or) algorithms.

* The subprograms are solved independently combining all the solution of the subprograms. if a solution of the core program.

→ $mid = \frac{start + end}{2}$.

→ A[mid] < key
start = mid + 1.

→ A[mid] > key
end = mid - 1



2) State & Explain graph coloring problem.

* Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color.
* The objective is to minimize the no. of colors while coloring the graph.
* The smallest no. of colors required to color a graph G is called chromatic number of graph.
* Graph coloring problem is a NP complete problem.

Step 1.

Method 4:- Arrange the vertices of in some order.

Step 2: Choose the first vertex and color it with first color

Step 3: Choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices are colored with this color, assign a new color to it. Repeat it until all the vertices are colored.

Appliation :-
- map coloring.
- making time table.
- register allocation.
- Mobile Radio frequency assignment.
- Bipartite Graph Checking

Q) Write the algo for Strassen Matrix Multiplication

* Let A and B the two $n \times n$ matrix.
* The product matrix $C = A \times B$.
* The result matrix is also $n \times n$ matrix.
* The matrix multiplication formula is.

$$C(i,j) = \sum_{1 \le k \le n} A(i,k)\, B(k,j)$$

* The reduce can be done by divide and conquer approach.
* We assume that n is a power of 2 that is there exists a non-negative

integer k such that $n = 2^k$

* In case 'n' is not a power of two, then enough rows & columns of zeroes can be added to both A & B so that the resulting dimentions are power of 2.

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

Then...

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

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

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

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

$$T(n) = \begin{cases} b & n \leq 2 \\ 8T(n/2) + cn^2 & n > 2 \end{cases}$$

$T(n) = O(n^3)$

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

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$P_2 = (A_{21} + A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12} \cdot B_{22})$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{12}) * B_{22}$

$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$P_7 = (A_{12} \cdot A_{22}) * (B_{21} * B_{22})$

N Queen Problem

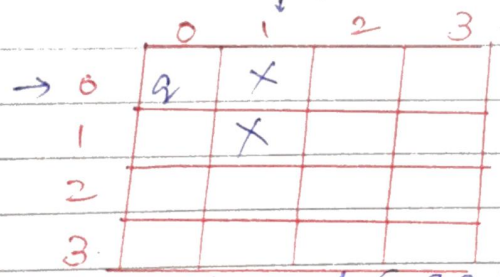
① * N Queen problem of placing N chess queens on N×N chessboard so that no two queens attack each other

② * We have to check that no two queens are placed in same row, column, diagnol

③ * We take the 4-Queen problem, as the generalized way to solve the N-Queen problem

④ * Example :-

Take a 4×4 chessboard



board [r][c]

⑤ → Step 0 :- c=0, r=0

check, board [0][0] = safe

Now increment [col] by 1

⑥ → Step 1 :- c=1, r=0

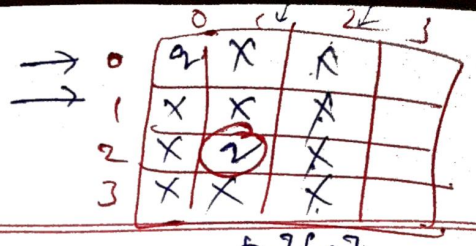
check, board [0][1] ≠ safe

so increment [row] by 1

⑦ → Step 1.1 :- c=1, r=1

check board [1][1] ≠ safe

so ↑ row by 1.



⑧ Step 1.2 : c=1, r=2.
board[2][1] = safe.
place queen
Increment col by 1.

⑨ Step 2.0 : c=2, r=0.
board[0][2] ≠ safe
↑ row by 1.

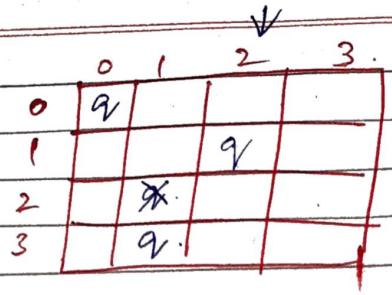
⑩ Step 2.1 : c=2, r=1.
board[1][2] ≠ safe.
↑ row by 1.

⑪ Step 2.2. c=2, r=2.
board[2][2] ≠ safe.
↑ row by 1.

⑫ Step 2.3 c=2, r=3.
board[3][2] ≠ safe.
~~Loop ends~~
we reached last row of col 2, there is no safe position.
So, we have to backtrack.

⑬ Note : We have to remove the last placed queen the one at position.
board [2][1] removed.
new row = 2+1 = 3.
col = 1.

(14)



(15)

Step 1.3 Set col = 1, row = 3.
check board [3][1] = safe.
↑ col by 1.

(16)

Step 2.0 c = 0, r = 0.
check board [0][2] ≠ safe.
↑ row by 1.

(17)

Step 2.1 c = 2, r = 1.
check board [1][2] = safe.
place Q.
↑ col by 1.

(18)

Step 3.0 c = 3, r = 0.
check board [0][3] ≠ safe.
↑ row by 1.

(19)

Step 3.1 c = 3, r = 1.
check board [1][3] ≠ safe.
↑ row by 1.

(20)

Step 3.2 c = 3, r = 2.
check board [2][3] ≠ safe.
↑ row by 1

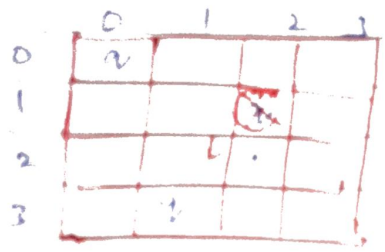
(21)

Step 3 : c = 3, r = 3
board [3][3] ≠ safe

(22) Ans

→ We reached last row and last column but don't place the 4th queen
→ So, again we do backtracking
→ remove last placed queen in board i.e. board [1][2]
and increment row by 1
→ Set col = 2,
row = 1+1 = 2

(23)



(24)

Step - 2 : c = 2, r = 2
board [2][2] ≠ safe
↑ r by 1

(25)

Step 2 3 : c = 2, r = 3
board [3][2] ≠ safe
again backtrack

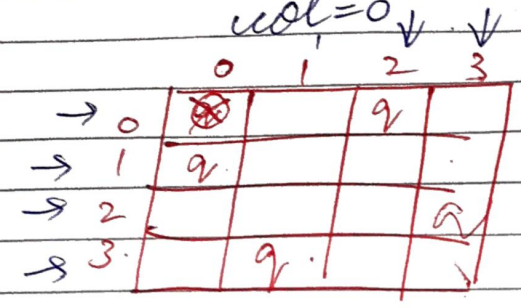
(26)

Note -

→ again backtrack because we have reached end of row.
→ remove the last placed last item placed i.e. board [0][0]
→ increment r = 1

→ new set row = 1.

col = 0



(28) Step 0.1 :- c=0, r=1.
board [1][0] = safe.
place queen
increment col by 1

(29) Step 1.0 c=1, r=0.
board [0][1] ≠ safe.
↑ r by 1

(30) Step 1.2 c=1, r=2.
board [2][1] ≠ safe.
↑ r by 1

(31) Step 1.3 c=1, r=3
board [3][1] = safe.
Place queen.
↑ c by 1.

(32) Step 2.0 c=2, r=0.
board [0][2] = safe.
place queen.
↑ c by 1.

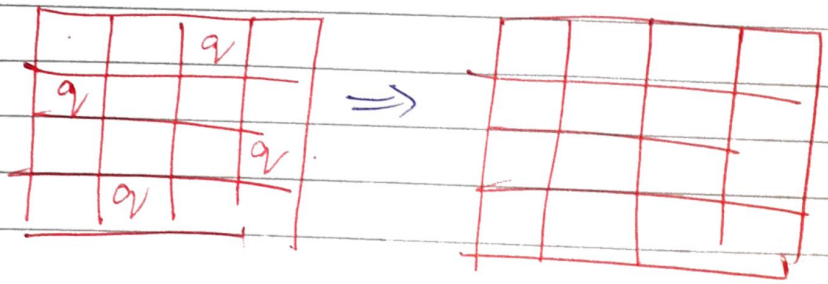
(33) Step 3.0 c=3, r=0.
board[0][3] ≠ safe
↑ r ++

(34) Step 3.1 c=3, r=1.
board[1][3] ≠ safe..
↑ r ++

(35) Step 3.2. c=3, r=2.
board[2][3] = safe
place queen.

(36) Finally we have placed all queens i.e 4.

(37)



(38)

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Matrix format

STRASSEN'S MATRIX MULTIPICATION

* It can be performed only on Square matrices
* Order of both matrices are n×n
* The idea of Strassen's method is to reduce the no of multiplication calls to 7
* There are 7 multiplication and 4 addition, Subtraction in total

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

Using Strassens algorithm, compute the following

$$M_1 = (A+C) \times (E+F)$$
$$M_2 = (B+D) \times (G+H)$$
$$M_3 = (A-D) \times (E+H)$$
$$M_4 = A \times (F-H)$$
$$M_5 = (C+D) \times E$$
$$M_6 = (A+B) \times H$$
$$M_7 = D \times (G-E)$$

Then, $I = M_2 + M_3 - M_6 - M_7$.

$J = M_4 + M_6$

$K = M_5 + M_7$

$Z = M_1 - M_3 - M_4 - M_5$.

* Time complexity - $O(n^{\log 7})$

Algo:

```
Strassen (x, y):
    if len(x) == 1:
        return x * y

    a, b, c, d = split(x)
    e, f, g, h = split(y)

    m1 = Strassen (a+c), (e+f)
    m2 = Strassen (b+d), (g+h)
    m3 = Strassen (a-d), (e+h)
    m4 = Strassen (a, f-h)
    m5 = Strassen (c+d, e)
    m6 = Strassen (a+b, h)
    m7 = Strassen (d, g-e)
```

$$\begin{matrix} i & \\ j & \\ k & \\ l & \end{matrix}$$

$$c = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

return c

0/1 Knapsack problem

* The knapsack is like a bag which has some weights associated with some values
* Here, the weight should not exceed the maximum weight
* Also, we must get more value
* ∴ less weight, more value
* The items in the knapsack are indivisible i.e you can't break an item, you either take an item or don't take any item

Example :- items $n=4$, $w=5$.

$\therefore w_1, w_2, w_3, w_4 = \{2, 3, 4, 5\}$

values $= \{3, 4, 5, 6\}$.

items	weight	value
1	2	3
2	3	4
3	4	5
4	5	6

Formula :- $T(i,j) = \max(T(i-1, j), (\text{value}) + T(i-1, j-(\text{weight})i))$.

i \ j	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0					
3	0					
4	0					

r = i
c = j

Step 1 $T(1,1)$ $i=1, j=1$

$(value)_i = (value)_j = 3.$

$(weight)_i = (weight)_1 = 2.$

$\therefore T(1,1) = \max\{T(1-1,1),\ 3+T(1-1,1-2)\}$

$= \max\{T(0,1),\ 3+T(0,-1)\}$ → ignore

$= T(0,1) = 0.$

Step 2 $T(1,2)$ $i=1, j=2.$

$(value)_i = (value)_i = 3$

$(weight)_i$ $(weight)_j = 2.$

$T(1,2) = \max\{T(1-1,2),\ 3+T(1-1,2-2)\}$

$= \max\{T(0,2),\ 3+T(0,0)\}.$

$= \max\{0,\ 3+0\} = 3$

Step 3 $T(1,3)$ $i=1, j=3.$

$(value)_1 = 3.$

$(weight)_1 = 2.$

$\therefore T(1,3) = \max\{(T(1-1,3),\ 3+T(1-1,2-2)\}$

$= \max\{T(0,3),\ 3+T(0,1)$

$= \max\{0,\ 3+0\} = 3.$

Step 4 $T(1,4)$ $i=1, j=4.$

$(value)_1 = 3$, $(weight)_1 = 2.$

$\therefore T(1,4) = \max\{T(1-1,4),\ 3+T(1-1,4-2$

$= \max\{T(0,4),\ 3+T(0,2)\}$

$= \max\{0, 3, 0\} = 3.$

Step 5:-

$T(1,5)$ = $i=1, j=5$.

$(value)_1 = 3$, $(weight)_1 = 2$.

$\therefore T(1,5) = \max\{T(0,5), 3+T(0,3)\}$

$= \max\{0, 3+0\} = 3$.

Step 2.1 $i=2, j=1$.

$(value)_2 = 4$, $(weight)_2 = 3$.

$\therefore T(2,1) = \max\{T(1,1), 4+T(1,-2)\}$ ↳ ignore

$= T(1,1) = 0$.

Step 2.2 $i=2, j=2$.

$(value)_2 = 4$, $(weight)_2 = 3$.

$T(2,2) = \max\{T(2-1,2), 4+T(2-1,2-3(2)$

$= \max\{T(1,2), 4+T(1,$

Step 2.3 $i=2, j=3$

$(value)_2 = 4$, $(weight)_2 = 3$

$T(2,3) = \max\{T(2-1,3), 4+T(2-1,3-3(2))$

$= \max\{T(1,3), 4+T(1,0) = 4$.

Step 2.4 $i=2, j=4$

$(value)_2 = 4$ $(weight)_2 = 3$.

$T(2,4) = \max\{(2-1,4), 4+T(2-1,3-4(2))$

$= \max\{(1,4), 4+T(1,$

Step 2.5 $i=2, j=5$

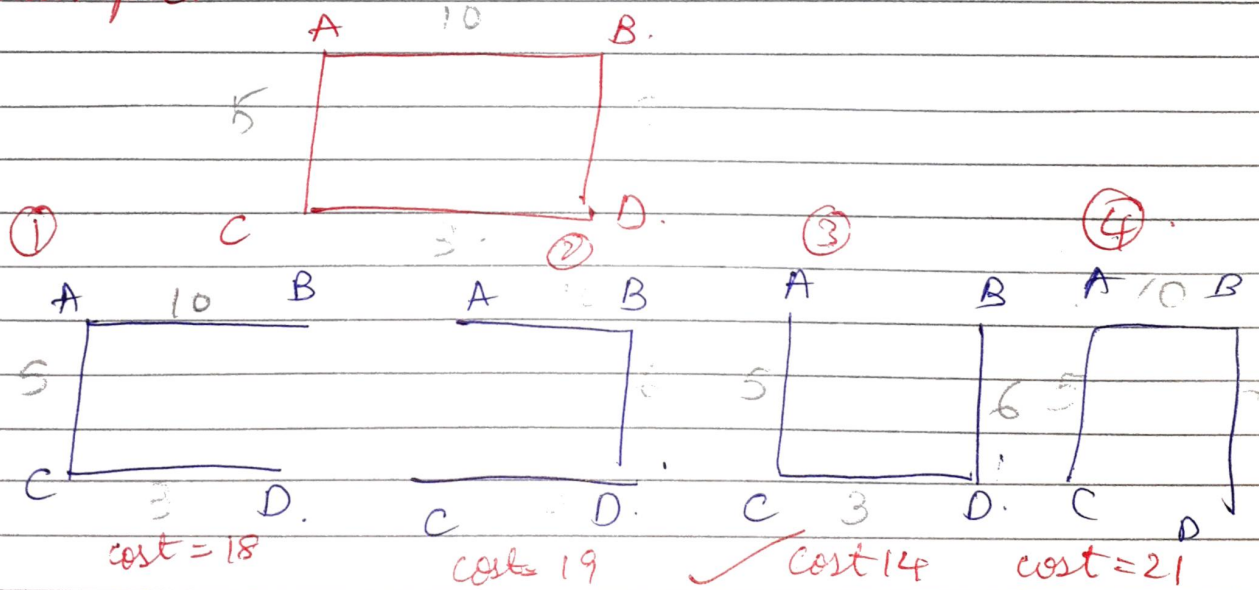
$(value)_2 = 4$ $(weight)_2 = 3$.

$T(2,5) = \max\{$.

Spanning Tree

* A Spanning Tree is a subgraph of a given graph
* It contains 3 properties
  a) It must contain all the vertices of a graph
  b) If the graph contains n vertices, then the Spanning tree must contain n-1 edges.
  c) Spanning tree should not contain any cycle

Example:-



* Therefore from the above graph we can construct 4 Spanning Trees
* ③ we can say that diag 4 is the minimum cost Spanning tree because it has minimum cost

Minimum Cost Spanning Tree :-

* If the spanning tree's weight is minimum, then it is said to be minimum spanning tree

PRIM'S ALORITHM.

* Prim's Algorithm is mainly used to calculate minimum cost spanning tree

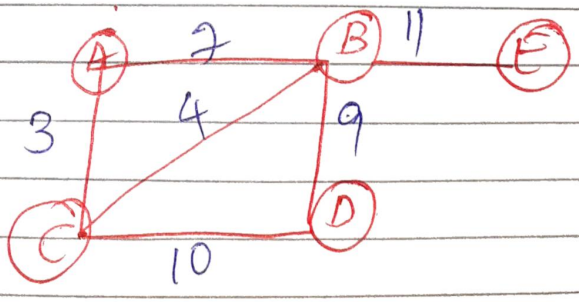
Step 1:- Start with any vertex of the graph

Step 2:- Find the edges associated with that vertex and add minimum cost edge to the spanning tree if it is not forming any cycle, suppose if it is forming any cycle then discard the edge.

Step 3:- Repeat Step 2 till all the vertices are completed.

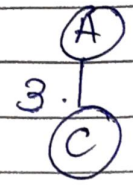
* Spanning Tree properties :-

* Example :-



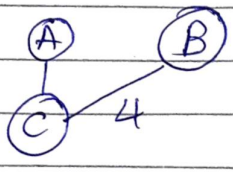
* Take (A).

(A).
A – B (7).
A – C (3) ✓ ⇒
Take 3.



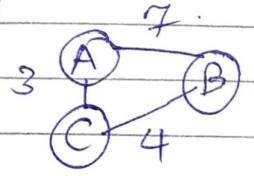
* Now the new edge is C, Take all the posibilies of C along with untaken A

(C)
A – B (7)
C – B (4) ✓ ⇒
C – D (10)
Take 4.



* Now, the new edge is B.
* Take all the posible edges of A, B, C.
remaing.

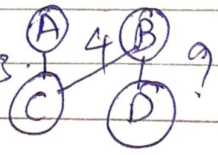
A – B (7) ✗ discard
C – D (10)
B – D (9) ⇒
B – E (11).



✗

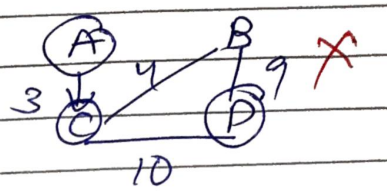
* Here we observe that if we take A – B (7) it forms a cycle
so we must take
remaing edges
* Take again all the posibilites of A, B, C.

C – D (10)
B – D (9) ✓ ⇒
B – E (11)

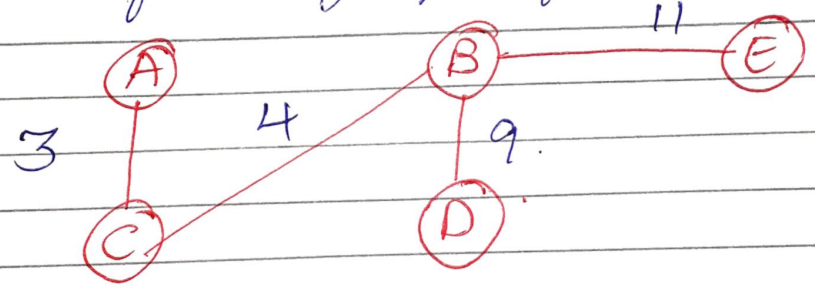


* Now take the new edge D remaining
* Take all the posible edges of A, B, C, D

C-D (10) ✗ discard.
B-E (11)



* Here, we discard C-D (10) because it is forming cycle.
* Take the other remaining posibilities i.e B-E (11). ✓
* ∴ The final graph formed is.



* ∴ This is the minimum cost spanning tree formed with the help of Prim's Algorithm.
* The cost of this MCST = 27.
* ∴ All the properties of the Spanning tree are satisfied.

__/__/____

Characteristics of Algo.

1) i/p & o/p — o/1 I/p ; atleast 1 o/p
2) finiteness = steps
3) definiteness (or) Unambiguous - clear. (not mistake)
4) effectiveness - necessary steps
5) generality - any type of i/p

Time Complexity / Space Complexity

i) best case ii) worst case iii) avg case
Eg explain all cases with linear search.

2 Approaches to calculate T.C

① frequency count & step count.
② Asymptotic notations

Components of TC

① Capacity of the System.
② Computer contains single processor or multiple processors

Components of SC

① Instruction space - store inst
② Env stack - partially executed func
③ Data space - variables & const

SC of algo can be calculated in 2 parts.

① fixed part - ∅ independent char.
② Variable part - dependent char.

$S(P) = C + S_p$ → variable part.

SC / const &

Quick Sort

① Divide & conquer.

② Pivot = 1st, last, random, median

③
```
QuickSort (arr[], l, r){
  if (l<r){
    int pi = partition (arr[], l, r)

    Quicksort (arr[], l, pi-1).
    Quicksak (arr[], pi+1, r).

  }
}
```

④

{6, 3, 9, 5, 2, 8, (7)}.

partition around 7.

{6, 3, 5, (2)} 7 {8, (9)}.

partition around 2. partition around 9.

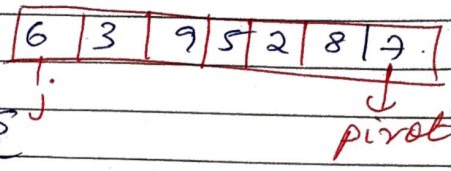
{} 2 {6, 3, (5)}. {8} 9 {}

partition around 5.

{3} 5 {6}.

S.A = {2, 3, 5, 6, 7, 8, 9}

③ partition algo.




```
Partition (arr[], l, r) {
    pivot = arr[r].
    i = l-1.
    for j = l to r-1.
        if arr[j] < pivot.
            i++;
            Swap(i, j).

    Swap(i+1, r).
    return i+1.
}
```

Code

```
#include <iostream>
using namespace std;

void quicksort (int arr[], int l, int r) {
    if (l < r) {

        int pi = partition (arr, l, r);
        quicksort (arr, l, pi-1);
        quicksort (arr, pi+1, r);

    }

}
```

```
int partition (int arr[], int l, int r){
    int pivot = arr[r];
    int i = l-1;

    for (int j=l; j<r; j++){
        if (arr[j] < pivot){
            i++;
            swap(arr, i, j);
        }
    }
    swap(arr, i+1, r);
    return i+1;
}

void swap(int arr[], int i, int j){
    int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}

int main(){

    int arr[5] = {5,4,3,2,1};
    quicksort(arr, 0, 4);
    for (int i=0; i<n; i++){
        cout << arr[i] << " ";
    } cout << endl;
return 0;
```

Complexity

Depends on pivot

1) In best case, pivot would be median element
2) In worest case, pivot would be end element

$O(n \log n)$ in best case
$O(N^2)$ in worest case

Analysis of Quick Sort

Best case

If there is only 1 element $- T(n) = 1$.

If $n > 1$ then $T(n) = 2T(n/2) + n$.

$= 2[2T(n/2/2 + n/2] + n$.

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

$= 4[2T(n/4/2 + n/4] + 2n$.

$= 8T(n/8) + 3n \Rightarrow 2^3 T(n/2^3) + 3 \cdot n$

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

let $2^k = \log n$.

$k = \log_2 = \log n$.

$k = \log n$.

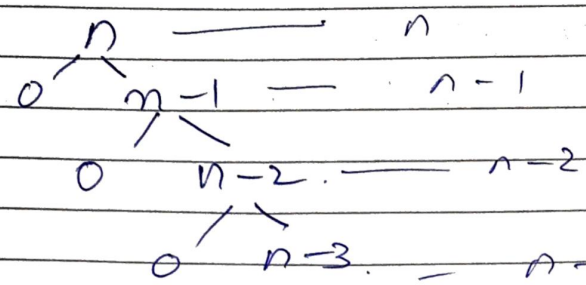
$T(n) = n \cdot T(n/n) + \log n \cdot n$.

$= n \cdot 1 + n \cdot \log n$.

$= n \log n$.

$O(n \log n)$

Worst case



$1+2+3+4+5 \Rightarrow \frac{n(n+1)}{2} = \frac{n^2+n/2}{2}$

$O(n^3)$.

Knapsack prob using Greedy.

n objects

profit (Pi) weight (wi).

Knapsack (Kg).

* The main objective is to place the corresponding object with the weight to obtain max profit.
* optimal sol of knapsack vector $0 \leq x_i \leq 1$
* If obj is placed in KS then $x_i = 1$. else $x_i = 0$.

or

$x_i = \frac{\text{Rem size of KS}}{\text{Actual size}}$

Algo

1. Calculate $\frac{P_i}{w_i}$ for every obj

2. Arrange all the objects in decreasing order based on $\frac{P_i}{w_i}$

3. Place objects in knapsack based on previous values/data

Example problem:

5 objects.

$m = 100$

$(P_1, P_2, P_3, P_4, P_5) = (20, 30, 66, 40, 60)$

$(w_1, w_2, w_3, w_4, w_5) = (10, 20, 30, 40, 50)$

1) $\frac{P_i}{w_i} = \frac{P_i}{w_i} =$

$= P_1/w_1 = 2$

$= P_2/w_2 = 3/2 = 1.5$

$= P_3/w_3 = 2.2$

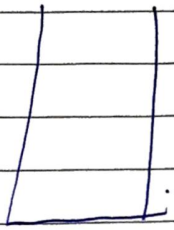
$= P_4/w_4 = 1$

$= P_5/w_5 = 6/5 = 1.2$

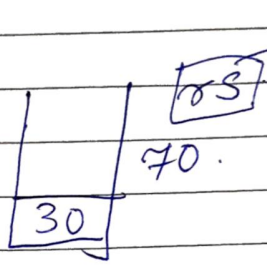
2) D.O.

$P_3/w_3 > P_1/w_1 > P_2/w_2 > P_5/w_5 > P_4/w_4$

3). $m = 100$

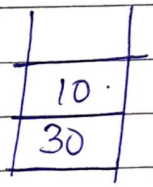


i). (P_3, W_3).
$= (66, 30)$



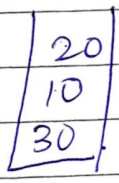
$x_3 = 1$.

(ii) (P_1, W_1)
$= (20, 10)$



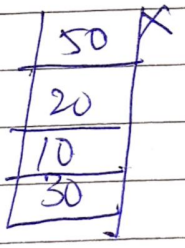
$x_1 = 1$.

(iii) $P_2 (W_2)$
$(30, 20)$



$x_2 = 1$.

(iv). (P_5, W_5)
$= (60, 50)$.



no suff space.

$\therefore x_5 = \frac{\text{rem size}}{\text{actual size}}$.

$x_5 = \frac{40}{50}$.

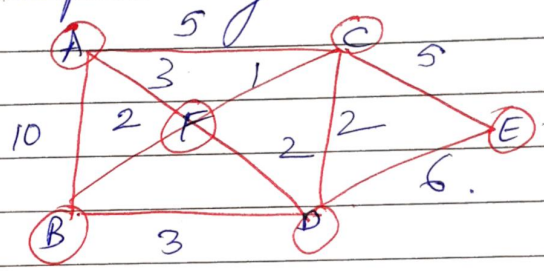
(v) $(P_4, W_4) = (40, 40)$ as $m = 0$. (bg empty)
$x_4 = 0$.

4) $profit = \sum P_i x_i = 20\times1 + 30\times1 + 66\times1 + 40\times0 + 60\times4/5$

$\Rightarrow 20+30+66+48 = 164.$

KRUSKAL'S ALGO.

Mainly used to construct Minimum cost Spanning tree.



n edges
n-1 vertices

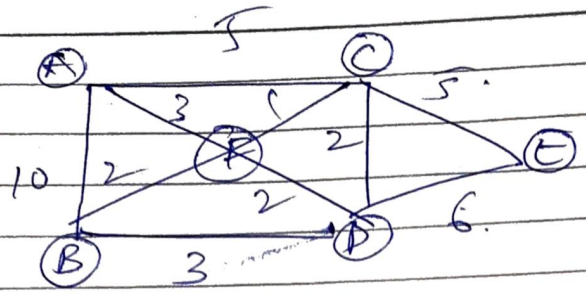
Algo

1) Arrange all the edges in A.O. based upon cost.

2) Select minimum cost edge from the edge list & add that edge to the Spanning tree. if its not forming any cycle.

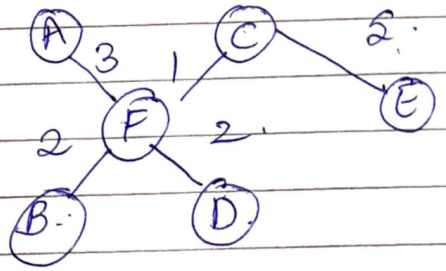
3) Suppose a cycle is formed then discard that edge.

4) Repeat Step 2 untill all the vertices are covered.



	edge	cost	
1).	C-F	1.	✓
	B-F	2.	✓
	D-F	2.	✓
	A-F	3.	✓
	B-D	3.	X
	A-C	5.	X
	C-E	5.	✓
	D-E	6.	X
	A-B.	10.	X
	C-D	12.	X

2). add C-F.



cost: 13.
6 vertices satisfied
5 edges satisfied

Job Sequencing With Deadlines using Greedy Approach

'n' jobs.

	profit (Pi)	Deadline (Di)

Goal: We can earn profit if and only if have completed the job by its deadline.

constraints

1) 1 machine.
2) 1 unit of time

Algorithm

1) Arrange all jobs in D.O. based on profit values.
2) Select the first job and execute it by assigning a slot.
3) Likewise also also we must execute all one by one

Ex:

5 jobs. $[n-1, n]$

$(P_1, P_2, P_3, P_4, P_5) = (100, 19, 38, 27, 52)$

$(d_1, d_2, d_3, d_4, P_5) = (2, 1, 2, 1, 3)$

1) D.O. $(P_1, P_5, P_3, P_4, P_2) = (100, 52, 38, 27, 19)$
$(d_1, d_5, d_3, d_4, d_2) = (2, 3, 2, 1, 1)$

2)

Assigned slot	Job selected	Action	profit
none.	J_1	$[1,2]$	100.
$[1,2]$.	J_5.	$[2,3]$	152.
$[1,2]$ $[2,3]$.	J_3.	$[0,1]$	190.
$[0,1]$ $[1,2]$ $[2,3]$	J_4.	reject it.	190.
$[0,1]$ $[1,2]$ $[2,3]$	J_2.	reject it.	190.

J_3 J_1 J_5
0 1 2 3.

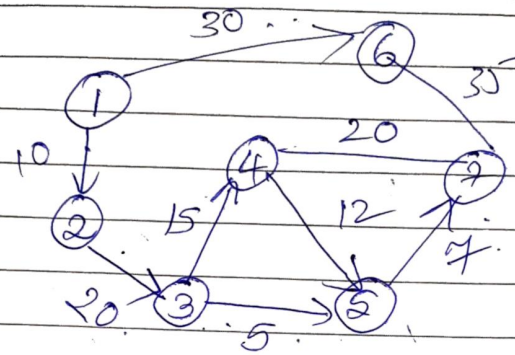
//

Single Source Shortest path Algo (Dijkstra's Algo).

1) Starts at the source node.
2) Assume we have a graph $G = (V, E)$ & it is directed weighted graph.

Aim: Select one vertex as

Source vertex & find the shortest path from s to all remaining all vertices in a graph



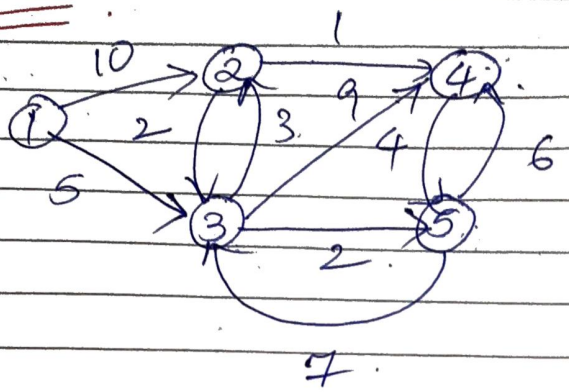
1 to 2 → 10.
1 to 6 → 30.

Selected Vertex	Visited Vertex	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
1.	{1}	10	∞	∞	∞	30	∞
2.	{1,2}	10	30	∞	∞	30	∞
3.	{1,2,3}	10	30	42	35	30	∞
~~6~~	{1,2,3,~~6~~}	10	30	~~45~~	~~35~~	30	65
5.	{1,2,3,6,5}	10	30	45	35	30	42
7.	{1,2,3,6,5,7}	10	30	45	35	30	42
4	{1,2,3,6,5,7,4}	10	30	45	35	30	42

1 to

2	10
3	30
4	45
5	35
6	30
7	42

Ex 2



SV	VS	d[2]	d[3]	d[4]	d[5]
1	{1}	10	5	∞	∞
3	{1,3}	8	5	14	7
~~7~~	{1,3,~~7~~}	8	5	13	7
2	{1,3,5,2}	8	5	9	7
4	{1,3,5,4}	8	5	9	7

2	8
3	5
4	9
5	7

Sum of Subsets

Using Backtracking

$n = 7$

$w_1, w_2, w_3 \cdots w_7 = (1, 2, 3, 4, 5, 5, 7)$

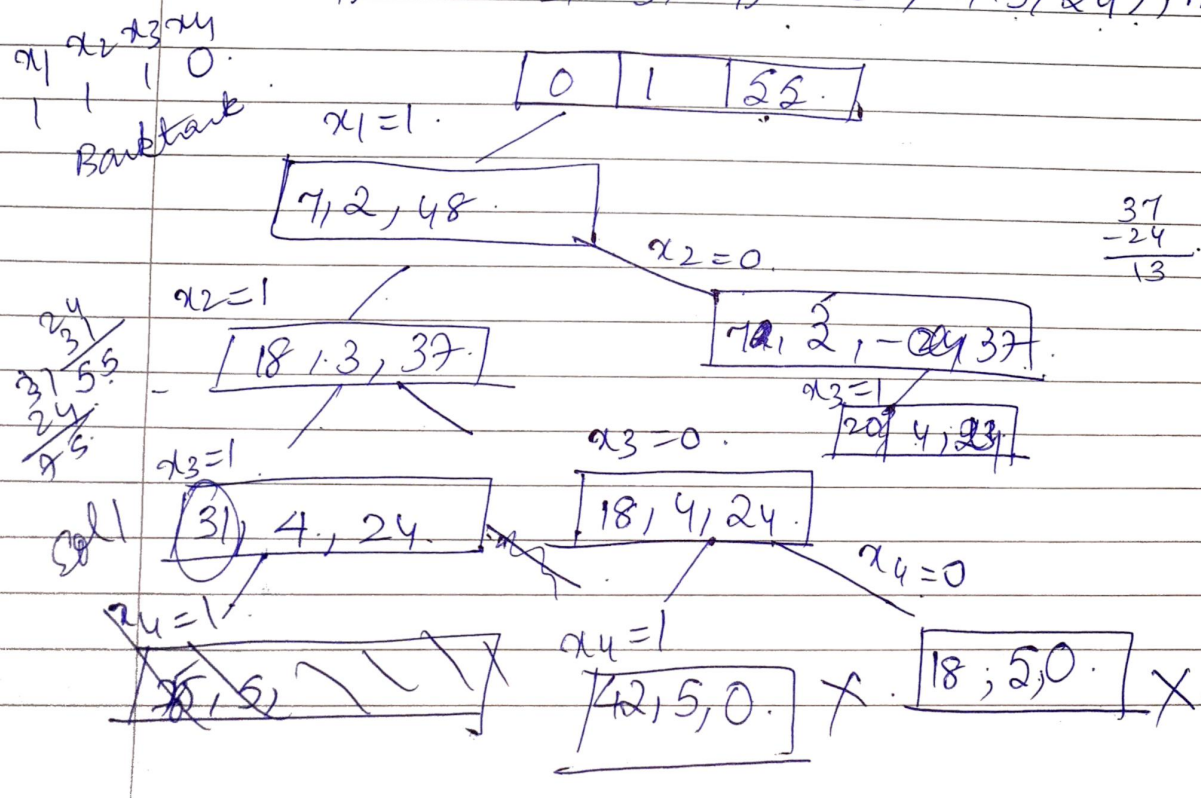
$m = 10$

Objects	x_1	x_2	x_3	x_4	x_5	x_6	x_7
	1	1	1	1	0	0	0
	0	0	0	1	0	1	0
	0	0	1	0	0	0	1

Ex.

$n = 4, (w_1, w_2, w_3, w_4) = (7, 11, 13, 24), m = 31$

$o_1 \; o_2 \; o_3 \; o_4$

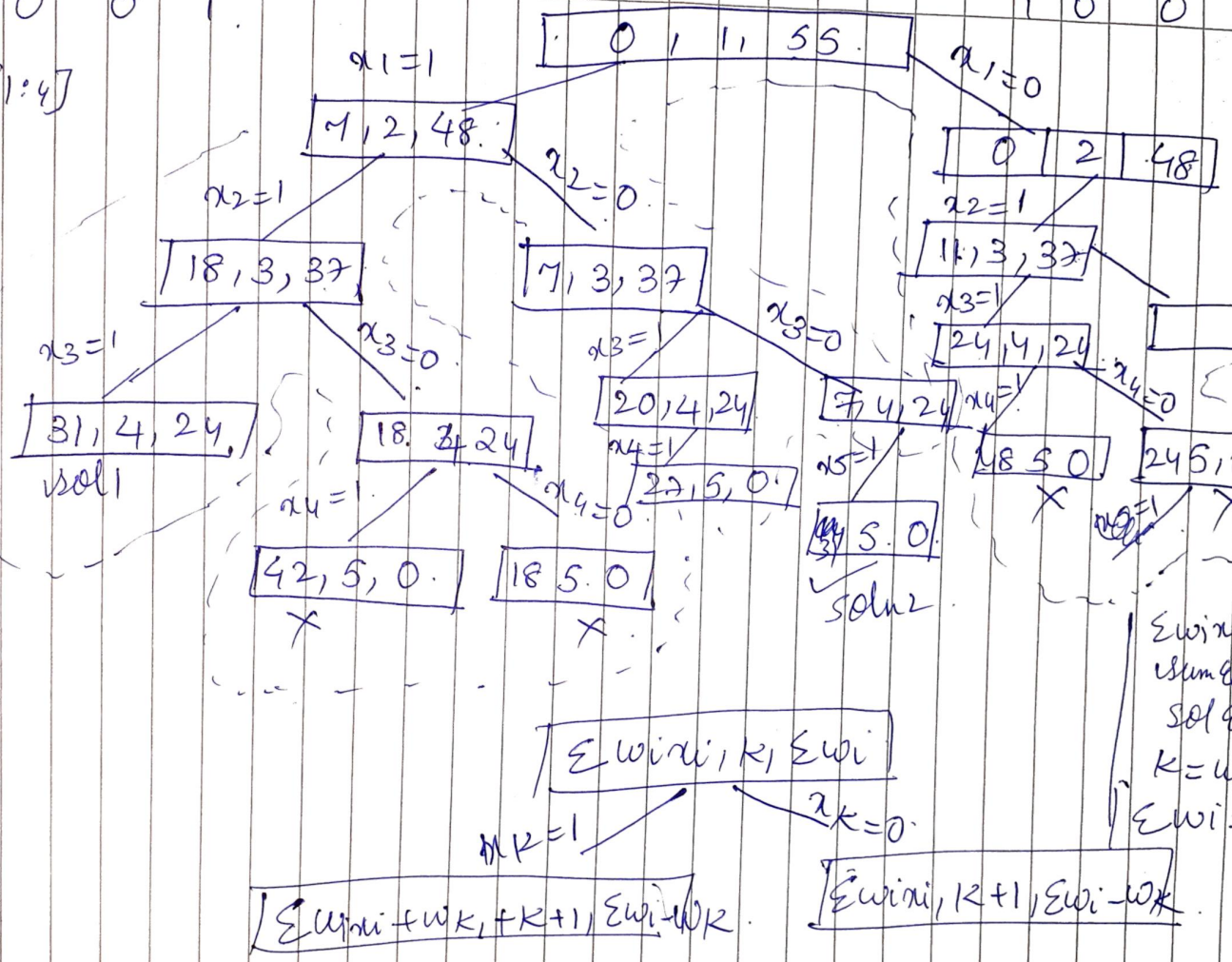


x_1	x_2	x_3	x_4	x_5
0	0	1	0	0
0	0	0	0	1

$w[1:4]$

o_1 o_2 o_3 .
7, 11, 13, 24 .

x_1	x_2	x_3	x_4
1	1	1	0
1	0	0	1



$\Sigma w_i x_i$
sum of prod of sol & wt vals
k = wt no
Σw_i = sum of wts

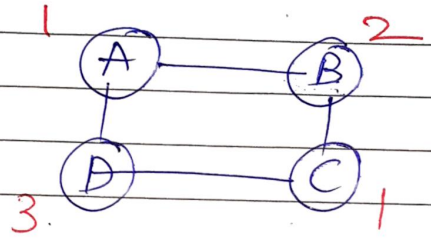
Algo

```
Sumofsub(S, k, r)
// find all subsets that sum to M.
# the values of x[j], 1 ≤ j ≤ k already determined
// s = Σ_{j=1}^{k-1} w[j]*x[j] && r = Σ_{j=k}^{n} w[j]
// The w[j] are in D.O
// Assumed that w[i] ≤ m && Σ_{i=1}^{n} w[i] ≥ m

{
  // generate left child
  // Note: S + w[k] ≤ m since B_{k-1} is true
  x[k] := 1;
  if (S + w[k] = m)
    then write (x[1:k]); // subset found
  // there is no recursive call as
  // w[j] > 0, 1 ≤ j ≤ n
  else if (S + w[k] + w[k+1] ≤ m)
    then Sumofsub(S + w[k], k+1, r - w[k]);

  // generate right child && evaluate Bk
  if ((S + r - w[k] ≥ m && (S + w[k+1] ≤ m))
    then {
      x[k] := 0;
      Sumofsub(S, k+1, r - w[k]);
    }
}
```

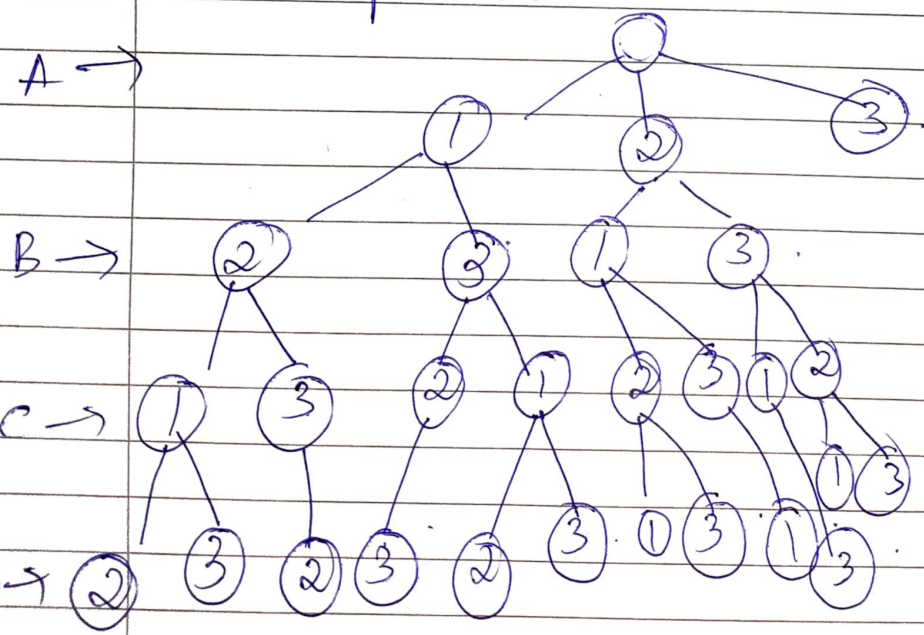
Graph Coloring Using Backtracking

We have to color all the vertices of a graph in such a way that no two adjacent edges should have the same color

Eg. m = 3. (Total clrs).
color names = 1, 2, 3.
G = (V, E).



State space tree.



Travelling Salesman Problem using Branch & Bound

$G = (V, E)$ (1, 2, 3, 4)

* TSM prob consists of salesman & a set of cities.
* The salesman has to visit each city starting from home & returning to same city
* Main challenge: The person wants to minimize the total length of trip
* Let $g(i, S)$ be the lt of shortest path starting at vertex i, going through all vertices in S & terminating at vertex 1
* The func $g(1, V - \{1\})$ is the lt of optimal sales persons

Eg

$$\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

$G = V, E$

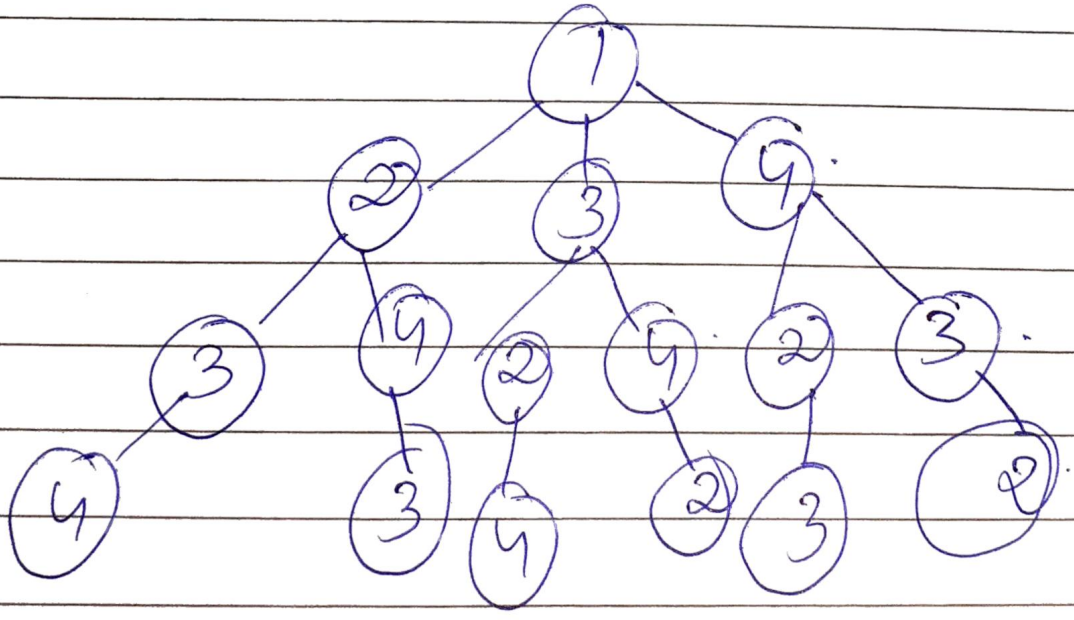
$$cost[i,j] = \begin{cases} 0 & \text{if } i = j \\ c_{ij} & \text{if } (i,j) \in G \\ \infty & \text{if } (i,j) \notin G \end{cases}$$

formula

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, \{S\} - j) \}$$

Eg:

$$g(1, \{1,2,3,4\}) = \min \{ c_{12} + g(2, \{3,4,5\}) \}$$



$|S| = \phi$ | $|S| = 1$