11-08-2025
Monday

* Unit-3. Divide And Conquer *

* Technique-1: Divide and conquer:-

Problems List:

1. Merge Sort
2. Quick sort
3. Max-Min
4. Strassen's matrix multiplication

* Technique-2: Greedy method:-

1. knapsack problem
2. Job sequencing with deadlines
3. Prims algorithm to find minimum spanning tree
4. kruskal's algorithm to find minimum spanning tree
5. Dijkstra's algorithm for single source shortest paths

* Divide & conquer method:-

- Divide & conquer is a general algorithm design technique. In general it has 3 steps:

1) Divide:-

Divide the problem into a no.of subproblems that are smaller instances of same problem.

2) Conquer:-

Conquer the subproblems by solving them recursively. If they are small enough, just solve them in a straight forward ~~matter~~ manne

3) Combine:-

Combine the subproblem solutions to give a solution to original problem.

* what is control abstraction (or) write C.A of divide & conquer?

Algorithm DAndC(P)

- what to do
  ↓
  cheppidhi
- How to do
  ↓
  cheppadu

```
{
    if small(P) then
        return S0
    else
    {
        Divide P into smaller instances P1, P2, .... Pk;
        Apply DAndC to each of these subproblems;
        return combine( DAndC(P1), DAndC(P2), .... DAndC(Pk));
    }
}
```

* **General divide & conquer reccurrence relation:-**

A problems instance of size n is divided into a subproblems of equal size of $\frac{n}{b}$.

then computing time of divide & conquer is

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ aT\left(\frac{n}{b}\right)+f(n) & \text{if } n>1 \end{cases}$$

where n : problem size

a : No. of smaller instances after division

$\frac{n}{b}$: Subproblem size

$f(n)$: Time spent on dividing the problem into smaller sub-problem & combining their solutions.

* **Recursion:-**

- Process in which a problem is defined in terms of itself.
- Recursion has 2 types:

① Base case ② Recursion case

* **Recursive Algorithm:-**

- A recursive algorithm is an algorithm that calls itself, one or more times on smaller inputs.
- To prevent an infinite chain of such calls, there has to be a value of the input for which the algorithm does not call itself.

* Methods to solve recurrence relation:-

Some of the methods to solve recurrence relations

1. Iteration method
2. Substitution method
3. Recursion tree method
4. Master theorem

* Calculate time complexity for recursive algorithm to find factorial of a number(n)?

```
Algorithm RFact(n)
{
   if (n==0)
      return(1);
   else
      return (n * RFact(n-1));
}
```

$$T.C \Rightarrow T(n) = \begin{cases} T(n-1)+3 & \text{if } n>0 \\ 2 & \text{if } n=0 \end{cases}$$

$T(n)$ — time to calculate $n!$

if $n=0$ $\quad T(n)=2$, $T(0)=2$

if $n=1$, $\quad T(n)=T(n-1)+3$

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

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

$\vdots$

$= T(n-k)+k(3)$ $\quad \boxed{\text{put } n-k=0 \\ k=n}$

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

$= T(0)+3n = 2+3n$

$= 3n+2 = O(n)$

* Recursive relation for Binary Search:-

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right)+5 & \text{if } n>1 \\ 4 & \text{if } n=1 \end{cases}$$

Sol

$T(n) = T\left(\frac{n}{2}\right)+5$

$= T\left(\frac{n/2}{2}\right)+2(5)$

$= T\left(\frac{n}{2^2}\right)+2(5)$

$= T\left(\frac{n}{2^3}\right)+3(5)$

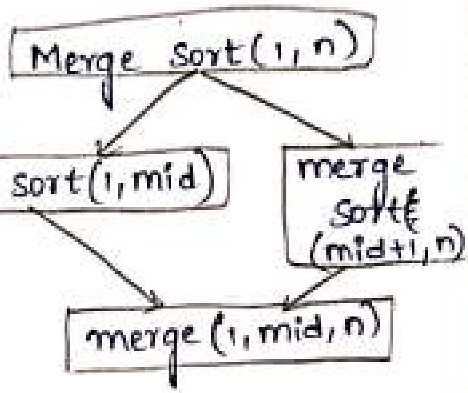
$$T(n) = T\left(\frac{n}{2^4}\right) + 4(5)$$
$$\vdots$$
$$= T\left(\frac{n}{2^k}\right) + k(5)$$
$$= T\left(\frac{n}{n}\right) + k(5)$$
$$= T(1) + (\log_2^n)5$$
$$= 4 + 5\log_2^n$$
$$T(n) = \Theta(\log_2^n)$$

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

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

13-06-2025 Wednesday

* Merge Sort:-



Main idea:

1. Divide the array into equal parts
2. Recursively sort two parts using merge sort
3. Combine two sorted parts using merge

* Merge sort: Pseudo code:-

```
//a: array of elements
//Initial call: MergeSort(a, 1, n)
Algorithm MergeSort(a, low, high)
{
    if (low<high)
    {
        mid = [(low+high)/2];
        Mergesort (a, low, mid);
        Mergesort (a, mid+1, high);
        Mergesort (a, low, mid, high);
    }
}
```

* Merge: Pseudo code

```
Algorithm Merge (a, low, mid, high)
{
    i=low; j=mid+1; K=low;
    while((i<=mid) and (j<=high)){
        if (a[i] < a[j]){
            b[K] = a[i];
        }
```

```
        else {
            b[K] = a[j];
            j = j+1 ;  K = K+1;
        }
    }
    while (i ≤ mid)
    {
        b[K] = a[i];
        i = i+1 ;  K = K+1;
    }
    while (j ≤ high)
    {
        b[K] = a[j];
        j = j+1 ;  K = K+1;
    }
    for k = low to high
        a[K] = b[K];
}
```

***Recurrence relation of merge sort:-**

$T(n)$: Time for merge sort of n numbers

1) Divide: Computes the middle of the subarray $\Theta(1)$
2) Conquer (Sort) the two sub lists recursively using merge sort. $2T\left(\frac{n}{2}\right)$
3) Combine (Merge) the two sorted sub lists to produce the sorted answer. $\Theta(n)$

So, the recurrence relation is

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

***Solve the recurrence relation for merge sort:**

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \text{if } n>1,\ T(1)=K,$$

where c, K are constants.

Solⁿ

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

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

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

$= 2 \; T \left(2^2 \right)$

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

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

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

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

$$\vdots$$

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

put $\frac{n}{2^x} = 1 \Rightarrow n = 2^x$ & $x = \log_2 n$

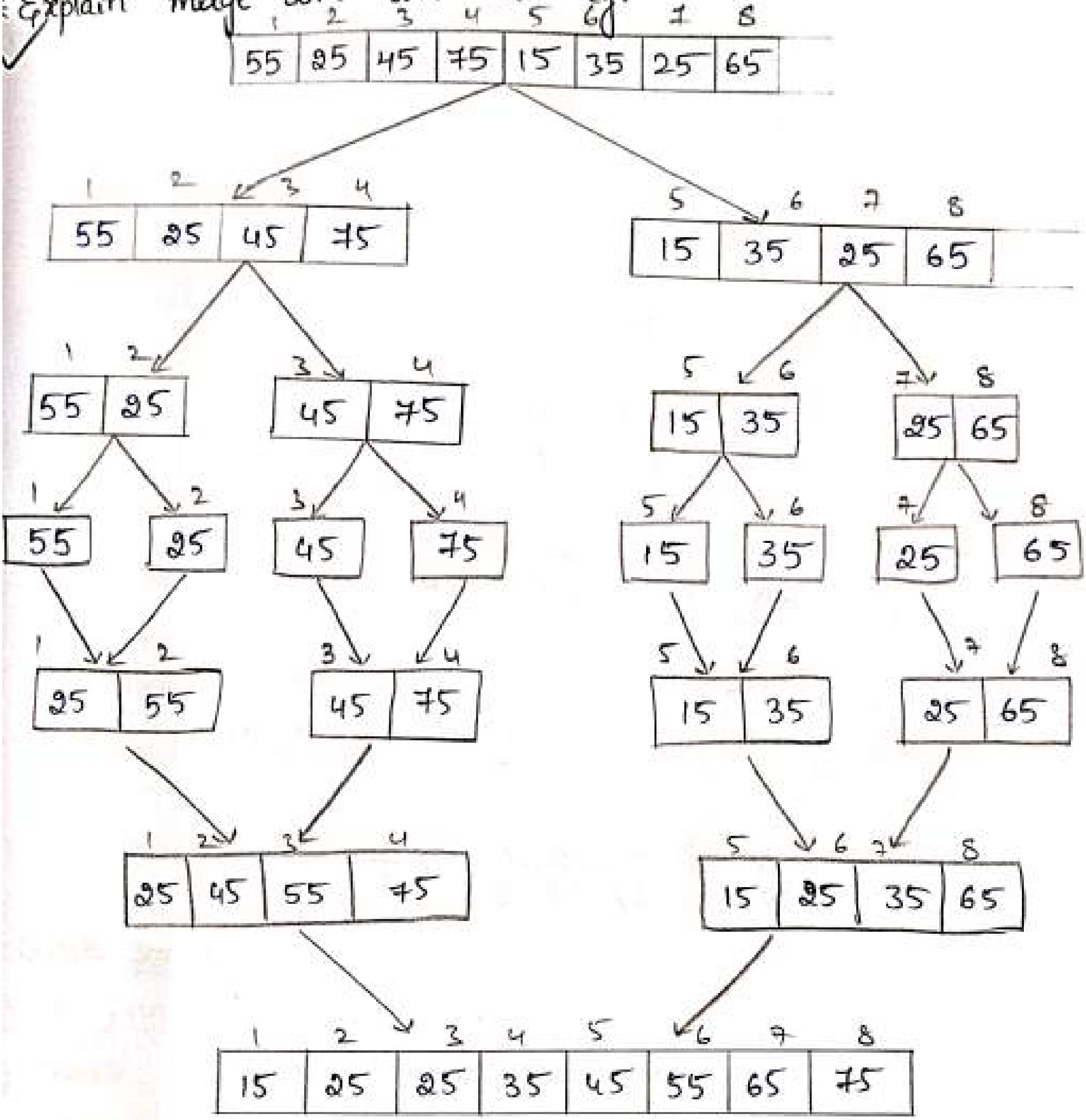
then $T(n) = n\,T\left(\frac{n}{n}\right) + (\log_2 n)\,cn$

$$= n\,T(1) + cn\log_2 n$$

$$= nk + cn\log_2 n$$

$$= \Theta(n\log_2 n).$$

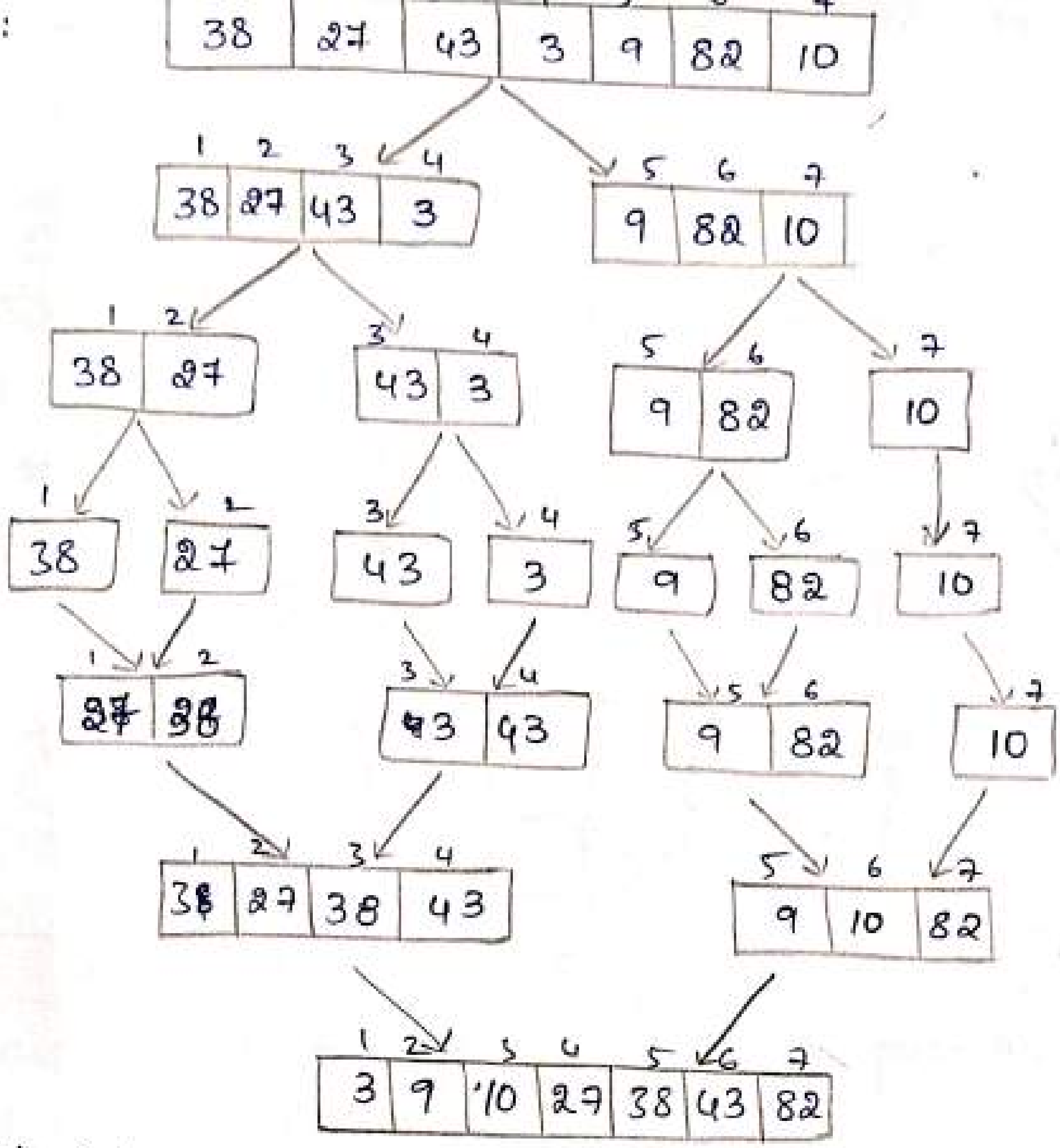
Explain merge sort with an eg:-



Note:-

If one path need to have more elements then we keep the more elements in left path.

g-②:



* **Quick Sort:-**

Main idea:-

1) Choose a pivot element from the array
2) Split the array into 3 sub arrays containing the items less than pivot, the pivot itself and the items bigger than pivot.
3) Recursively quicksort first and last sub array.

Q: Explain the quick sort with example:-

pivot $\boxed{50}$

50 40 20 60 80 100 45 70 105 30 90 75

i — i — i — i ⟵ swap ⟶ j — j — j

50 40 20 30 80 100 45 70 105 60 90 75

i — i ⟵ swap ⟶ j — j — j — j

50 40 20 30 45 100 80 70 105 60 90 75

i — i

j — j — j

Swap

| 45 40 20 30 | 50 | 100 80 70 105 60 90 75 |

Recursively applied to left part & right part. we get the sorted array as follows:

20 30 40 45 50 60 70 75 80 90 100 105

Eg-2:- 40 30 20 70 35 55 85 65 25 75 95

i — i — i — i

Swap

j — j — j

40 30 20 25 35 55 85 65 70 75 95

i — i — i

Swap

j — j — j

j — j

| 35 30 20 25 | 40 | 55 85 65 70 75 95 |

Recursively applied to left part & right part. we get the sorted array as follows:

20 25 30 35 40 55 65 70 75 85 95

Egr 3:- 23 89 45 32 9 56 84 93 66 19 83 64

i — i

swap

j — j — j

23 19 45 32 9 56 84 93 66 89 83 64

i — i

swap

j — j — j — j — j — j

23 19 9 32 45 56 84 93 66 89 83 64

i — i

j

swap

j — j

| 9 19 | 23 | 32 45 56 84 93 66 89 83 64 |

Recursively applied to left part and right part. we get the sorted array as follows: 9 19 23 32 45 56 64 66 83 84 89 93

* Quick sort: Pseudo code:-

```
//a: array of elements
//Initial call : Quicksort (a, 1, n)
Algorithm QuickSort ( a, low, high) {
   if (low< high){
        j= Partition (a, low, high);
        QuickSort (a, low, j-1);
        QuickSort (a, j+1, high);
   }
}
Algorithm Partition (a, low, high)
{
    pivot = a[low];
    i=low ; j= high;
    while( i<j ){
    while (i ≤ high and a[i] ≤ pivot)
        i=i+1;
    while (j≥ low and a[j] ≥ pivot)
        j=j-1;
    if (i<j)
        swap (a[i], a[j]);
    }
    swap (a[low] , a[j]);
    return(j);
}
```

* Quick Sort: Analysis:-

1) Time complexity

Best case	Worst case	Average case
$O(n \log n)$	$O(n^2)$	$O(n \log n)$

2) Not stable

3) In - place

4) Number of comparisons $\approx O(n \log n)$

5) Not adaptive

6) Number of swaps $\approx O(n \log n)$

* What is stable sorting?

. A sorting technique which ~~ma~~ preserves the relative order of duplicates is called stable sorting.

* why quick sort is not stable? Give an example:

10_B 50 10_R 20 10_y 80

↑ i — ↑ i ⟵ swap ⟶ ↑ j — ↑ j

pivot $\boxed{10}$

10_B 10_y 10_R 20   50   80

↑ i   ↑ j

∴ Not maintaining the relative order of 10's.

* Explain worst-case time complexity of quick sort. (2mts)

Happens for sorted and reverse sorted arrays

Occurs when the subarrays are completely unbalanced

Has 0 elements in one subarray and n-1 elements in the other subarray.

$T(n) = T(n-1) + T(0) + \theta(n)$, $T(1) = 1$

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

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

* write recurrence relation for worst-case of Time complexity of quick sort and solve it.

RR for worst-case time complexity of quick sort is

$T(n) = T(n-1) + cn$, $T(1) = 1$, where c is constant

Solution:

$$T(n) = \boxed{T(n-1)} + cn$$
$$= T(n-2) + c(n-1) + cn$$
$$= T(n-3) + c(n-2) + c(n-1) + cn$$
$$= T(n-4) + c(n-3) + c(n-2) + c(n-1) + cn$$
$$\vdots$$
$$= T(n-k) + c(n-(k-1)) + \dots + c(n-1) + cn$$

put $n-k = 1$

Then, $T(n) = T\left(n-(n-1)\right) + c\left(n-(n-2)\right) + \cdots\cdots + c(n-1) + cn$

$= T(1) + c2 + c3 + \cdots\cdots + c(n-1) + cn$

$= 1 + c\left[2 + 3 + \cdots\cdots + (n-1) + n\right] + c - c$

$= 1 - c + c\left[1 + 2 + 3 + \cdots\cdots + (n-1) + n\right]$

$= 1 - c + c\,\dfrac{n(n+1)}{2}$

$= \Theta(n^2)$

7

STRASSEN'S MATRIX MULTIPLICATION

* Matrix multiplication can be done in 3 ways:
  1. Straight approach (iteration approach) = $O(n^3)$
  2. Using divide and conquer approach = $O(n^3)$
  3. Using Strassen's equations = $O(n^{2.81})$

* Straight matrix multiplication: Pseudo code

```
// X, Y, Z are matrices of order nxn
// we are calculating Z = X * Y
StraightMatrixMul(X, Y, Z, n)
{
   for i=1 to n do          n times
     for j=1 to n do        n times
     {
        Z[i][j]=0;
        for k=1 to n do
            Z[i][j]= Z[i][j] + X[i][k] * Y[k][j];
     }
}
```

Time complexity : $O(n^3)$

* Master theorem:-

Let $a \geq 1$ and $b > 1$ be constants and $f(n)$ is an asymptotically positive function. Let $T(n)$ be defined on non-negative integers by the recurrences

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

1. Identify $a, b, f(n)$
2. Calculate $\log_b a$
3. Compare $n^{\log_b a}$, $f(n)$ asymptotically
   - 3.1. If $f(n)$ is big, then $T(n) = O(f(n))$
   - 3.2. If $n^{\log_b a}$ is big, then $T(n) = O(n^{\log_b a})$
   - 3.3. If $f(n)$ $\log^a$ ... then $T(n) = O(f(n).\log n)$.

1: $T(n) = 4T\left(\frac{n}{2}\right) + 10n^3$

Solⁿ $a=4$, $b=2$ $f(n)=10n^3$

$\log_b^a = \log_2^4 = 2$

$\therefore \log_y^x = \frac{\log x}{\log y}$

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

clearly $n^{\log_b^a} \le f(n)$

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

Eg2: $T(n) = 16T\left(\frac{n}{2}\right) + 10n^3$

$a=16$, $b=2$ $f(n)=10n^3$

$\log_b^a = \log_2^{16} = 4$

$\log_2^{4^2}$ $2\log_2^4$

$n^{\log_b^a} = n^4$, $f(n) = n^3$

$2(2\log_2^2)$

clearly $f(n) \le n^{\log_b^a}$

$2(2)$

$\therefore T(n) = O(n^4)$

4.

Eg3: Calculate time complexity for merge sort.

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

Sol $a=2$, $b=2$, $f(n)=cn$

$\log_b^a = \log_2^2 = 1$

By applying master theorem.

$n^{\log_b^a} = n^1$, $f(n) = n$

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

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

Eg4: $T(1) = 2$, $T(n) = 3T\left(\frac{n}{4}\right) + n$

Sol $a=3$, $b=4$, $f(n)=n$

$\log_b^a = \log_4^3 = 0.79$

$n^{\log_b^a} = n^{0.79}$, $f(n) = n^1$

clearly $n^{\log_b^a} < f(n)$ $\therefore T(n) = O(n)$

* Matrix multiplication using divide and conquer approach:-

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

$I = AE + BG$

$J = AF + BH$

$K = CE + DG$

$L = CF + DH$

Recurrence Relation for T.C:-

T(n): Time complexity to multiply X & Y matrices of order n×n

$$T(n) = \begin{cases} 8T(n/2) + \boxed{4(n/2)^2} & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n < 2 \end{cases}$$

$4\left(\frac{n}{2}\right)^2$

By applying master theorem;

$a = 8$, $b = 2$, $f(n) = 4n^2$ $\qquad \log_2^3$

$\log_b^a = \log_2^8 = 3$

$n^{\log_b^a} = n^3$, $f(n) = n^3$

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

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

* Explain Strassen's matrix multiplication:-

Suppose X, Y are matrices of order n & let us calculate $Z = X * Y$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1n} \\ z_{21} & z_{22} & \cdots & z_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nn} \end{bmatrix}$$

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

$M_1 = (A+C)(E+F)$

$M_2 = (B+D)(G+H)$

$M_3 = (A-D)(E+H)$

$M_4 = A(F-H)$

$M_5 = (C+D)E$

$M_6 = (A+B)H$

$M_7 = D(G-E)$

$I = M_2 + M_3 - M_6 - M_7$

$J = M_4 + M_6$

$K = M_5 + M_7$

$L = M_1 - M_3 - M_4 - M_5$

Recurrence relation for TC:-

T(n): Time complexity to multiply A & B matrices of order $n \times n$

7 submatrix multiplication of order $n/2 \times n/2$

18 matrix additions / subtractions

$$T(n) = \begin{cases} 7T(n/2) + 18(n/2)^2 & \text{if } n \geq 2 \\ O(1) & \text{if } n < 2 \end{cases}$$

$TC = O(n^{2.81})$

By applying master theorem:-

$a = 7$, $b = 2$, $f(n) = n^2$

$\log_b a = \log_2 7 = 2.81$

$n^{\log_b a} = n^{2.81}$; $f(n) = n^2$

$n^{\log_b a} > n^2$; So $TC = O(n^{2.81})$ //.

Give an example for Strassen's matrix multiplication:-

Suppose $X = \begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix}$ $Y = \begin{bmatrix} 3 & 7 \\ 2 & 4 \end{bmatrix}$

then $Z = XY = \begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix}\begin{bmatrix} 3 & 7 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 6+8 & 14+16 \\ 3+10 & 7+20 \end{bmatrix} = \begin{bmatrix} 14 & 30 \\ 13 & 27 \end{bmatrix}$

Let us apply Strassen's equations

$X = \left[\begin{array}{c|c} 2 & 4 \\ \hline 1 & 5 \end{array}\right] \Rightarrow A = 2,\ B = 4 \quad C = 1 \quad D = 5$

$Y = \left[\begin{array}{c|c} 3 & 7 \\ \hline 2 & 4 \end{array}\right] \Rightarrow E = 3 \quad F = 7 \quad G = 2 \quad H = 4$

$M_1 = (A+C)(E+F)$
$= (2+1)(3+7)$
$= (3)(10)$
$= 30$

$M_2 = (B+D)(G+H)$
$= (4+5)(2+4)$
$= (9)(6)$
$= 54$

$M_3 = (A-D)(E+H)$
$= (2-5)(3+4)$
$= (-3)(7)$
$= \cancel{-20}\ -21$

$M_4 = A(F-H)$
$= 2(7-4)$
$= 2(3)$
$= 6$

$M_5 = (C+D)E$
$= (1+5)3$
$= (6)(3)$
$= 18$

$M_6 = (A+B)H$
$= (2+4)4$
$= 6 \times 4$
$= 24$

$M_7 = D(G-E)$
$= 5(2-3)$
$= 5(-1)$
$= -5$

$I = M_2 + M_3 - M_6 - M_7 = 54 + (-21) - 24 + 5$
$= 14$

$J = M_4 + M_6 = 6 + 24 = 30$

$K = M_5 + M_7 = \cancel{30 - 5 = 25}\ 18 - 5 = 13$

$L = M_1 - M_3 - M_4 - M_5 = 30 + 21 - 6 - 18$
$= 51 - 24$
$= 27$

19-08-2025 Tuesday

* Maximum & minimum of n numbers: Divide & conquer approach:

MaxMin(1,n)

MaxMin(1,mid) MaxMin(mid+1,n)

Max1: ___
Min1: ___

Max2: ___
Min2: ___

```
if(max1>max2)
    max=max1
else
    max=max2

if(min1<min2)
    min=min1
else
    min=min2
```

* Explain Min Max algorithm with an example

Consider the following array of 8 numbers

70	30	40	20	50	60	25	75
1	2	3	4	5	6	7	8

MaxMin(1,8,75,20)

MaxMin(1,4,70,20) MaxMin(5,8,75,25)

MaxMin(1,2,70,30) MaxMin(3,4,40,20) MaxMin(5,6,60,50) MaxMin(7,8,75,25)

* Write Pseudo code for MaxMin Algorithm.

```
//a: Array of n ~~number~~ elements
// Initial call: MaxMin(a, 1, n, max, min) {
Algorithm MaxMin (a, i, j, max, min) {
    if(i==j)
        min=max=a[i];
        return;
```

```
if (i = j-1)
    if(a[i] < a[j])
        max = a[j];     min = a[i];
    else
        max = a[i];     min = a[j];
else
    mid = [(i+j)/2].
    MaxMin (a , i, mid, max1, min1);
    MaxMin(a, mid+1 , j , max2, min2);
    if (max1 > max2)
        max = max1;
    else
        max = max2;
    if (min1 < min2)
        min = min1;
    else
        min = min2;
}
```

* Number of comparisons = $\frac{3n}{2} - 2$

* Calculate Timecomplexity for minmax algorithm:-

Suppose $T(n)$ represents number of element comparisons,

The recurrence relation is

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

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

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

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

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

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

$$\vdots$$

$$= 2^K T\left(\frac{n}{2^K}\right) + 2^K + 2^{K-1} + \ldots\ldots\ldots 2^3 + 2^2 + 2^1 \quad \text{where } \frac{n}{2^K} = 2$$

$$2^K T(2) + 2\,\frac{(2^K - 1)}{2-1}$$

$$= 2^K T(2) + \left[2\left[2^K - 1\right]\right]$$

$$= 2^K \cdot 1 + \left[2\left[2^K - 1\right]\right]$$

$$= \frac{n}{2} + 2\left[\frac{n}{2} - 1\right]$$

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

20-08-2025
Wednesday.

* Greedy Methods:-

1) Knapsack problem
2) Job sequencing with dead lines
3) Minimal spanning tree using Kruskal's algorithm
4) Minimal spanning tree using Prim's algorithm
5) Single source shortest problem using Dijikstra's algorithm

* Optimization Problems:-

An optimization problem involves maximizing or minimizing an objective function subject to constraints.

* What is greedy method:-

(2m)

- The greedy method is an algorithm design technique used to solve the optimization problems.
- A greedy method arrives at a solution by making a sequence of choices where each choice is the one which looks the best at that moment.
- i.e, In the greedy method, the choice of the optional decision is made on the information

* Greedy Method : Terminology :-

1) Opti Objective function:

The function which has to be maximized (or minimized) subject to the given constraints is called objective function.

2) Feasible solution:

Any subset that satisfies the given constraints is called feasible solution.

3) Optimal solution:

Any feasible solution that maximies (or minimies) the given objective function is called optimal solution.

* write control abstraction of greedy method:-

```
//a: Array of size n which contains inputs
Algorithm Greedy (a, n)
{
   Solution = ∅;  // Initialize the Solution
   for i=1 to n
   {
      x = Select (a);
      if feasible (solution, x) then
           Solution = union (solution, x);
   }
   return (solution);
}
```

* Explain knapsack problem:

Given n items with their profits P_i, weights w_i and a knapsack with a capacity C. Fill the Knapsack with the possible items or fraction of items to get maximum profit.

~~Fixed~~ Formal representation of knapsack problem is

Maximize Profit $\sum_{i=1}^{n} x_i P_i$

Subject to $\sum_{i=1}^{n} x_i w_i \leq C$ where $0 \leq x_i \leq 1$, $1 \leq i \leq n$

where,

C : knapsack capacity

n : number of items

w_i: weight of i^{th} item

P_i: profit of i^{th} item

x_i: the fraction of i^{th} item placed in knapsack.

* Solve the following knapsack problem:

Items	I_1	I_2	I_3	I_4	I_5	I_6
profit	14	10	20	8	15	30
weight	5	8	4	6	5	15

knapsack capacity $c = 15$

1) Calculate $\frac{P_i}{W_i}$

Items	I_1	I_2	I_3	I_4	I_5	I_6
$\frac{P_i}{W_i}$	$\frac{14}{5} = 2.8$	$\frac{10}{8} = 1.25$	$\frac{20}{4} = 5$	$\frac{8}{6} = 1.3$	$\frac{15}{5} = 3$	$\frac{30}{15} = 2$

2) Arrange the items in decreasing order of $\frac{P_i}{W_i}$

Items	I_3	I_5	I_1	I_6	I_4	I_2
P_i	20	15	14	30	8	10
W_i	4	5	5	15	6	8

3) Greedy solution:-

Knapsack

$\frac{1}{15}[I_6]$
I_1
I_5
I_3

profit

$0 + 20 + 15 +$
$14 + \frac{1}{15}(30)$

$= 51$

Solution

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 1 & 0 & 1 & 0 & 1 & 1/15 \end{pmatrix}$$

Hence the optimal solution is $(x1, x2, x3, x4, x5, x6) =$
$= (1, 0, 1, 0, 1, 1/15)$ and the maximum profit is 51.

Eg 2:

Items	I_1	I_2	I_3	I_4	I_5	I_6
P_i	20	10	15	12	24	15
W_i	10	2	5	2	8	6

$n=6, \; c=15$

1) Calculate $\frac{P_i}{W_i}$

Items	I_1	I_2	I_3	I_4	I_5	I_6
$\frac{P_i}{W_i}$	$\frac{20}{10}=2$	$\frac{10}{2}=5$	$\frac{15}{5}=3$	$\frac{12}{2}=6$	$\frac{24}{8}=3$	$\frac{15}{6}=2.5$

2) Arrange the items in decreasing order of $\frac{P_i}{W_i}$

Items	I_4	I_2	I_3	I_5	I_6	I_1
P_i	12	10	15	24	15	20
W_i	2	2	5	8	6	10

3) Greedy ~~method~~ Solution:-

Knapsack	
$\frac{6}{8}[I_5]$	$\cancel{15}$
I_3	$\cancel{13}$
I_2	$\cancel{11}$
I_4	6

profit

$$0+12+10+15+\frac{6}{8}\overset{3}{\boxed{24}}$$

$= 55$

Solution

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & 1 & 1 & 1 & 6/8 & 0 \end{pmatrix}$$

$12 + 10 + 15 = 37$, $37 + 18 = 55$

Hence the optimal solution is $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 1, 1, 1, 6/8, 0)$ and the maximum profit is 55.

Eg3:

Items	I_1	I_2	I_3	I_4	I_5
P_i	12	32	40	30	50
W_i	3	10	10	4	10

$C = 20$

1) Calculate $\frac{P_i}{W_i}$

Items	I_1	I_2	I_3	I_4	I_5
$\frac{P_i}{W_i}$	$\frac{12}{3}=4$	$\frac{32}{10}=3.2$	$\frac{40}{10}=4$	$\frac{30}{4}=7.5$	$\frac{50}{10}=5$

2) Arrange the items in descending order of $\frac{P_i}{W_i}$

Items	I_4	I_5	I_1	I_3	I_2
P_i	30	50	12	40	32
W_i	4	10	3	10	10

3) Greedy solution:-

Knapsack | profit | Solution

Knapsack	
$\frac{3}{10}[I_3]$	~~20~~
I_1	~~16~~
I_5	~~6~~
I_4	3

$0+30+50+12+\frac{3}{10}(40)$ $= 104$

$$\begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \\ (1 & 0 & \frac{3}{10} & 1 & 1) \end{array}$$

50
30
12
12
104

Hence the optimal solution is $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 3/10, 1, 1)$ and maximum profit is 104.

* write pseudo code for greedy knapsack problem: Pseudo code

//n: number of items
//c: knapsack capacity
//P: Array of profits of size n
//w: Array of weights of size n
//x: Array to store the fraction of items placed in knapsack.

Algorithm ~~Fraction kno~~ Fractional knapsack (p, ω, n, c)

```
{
  for i=1 to n
      x[i] = 0.0;
  W=c;
  for i=1 to n
  {
    if(ω[i] ≤ W)
    {
       x[i]=1.0;
       W=W-ω[i];
    }
    else
       break;
  }
  if(i≤n)
     x[i] = W/ω[i];
  profit = 0.0;
  for i=1 to n
     profit = profit + x[i]* p[i];
  return (profit);
}
```

<u>Time complexity</u>:

1) If items given in sorted order then TC = O(n)
2) If items are not given in sorted order then TC = O(n log n)

21-08-2025
Thursday

* Find the optimal solution to the following knapsack problem

i	1	2	3	4	5	6	7	8	c = 75
P_i	30	20	10	15	18	35	25	51	
ω_i	10	5	6	7	12	25	8	30	

1) Calculate $\frac{P_i}{\omega_i}$

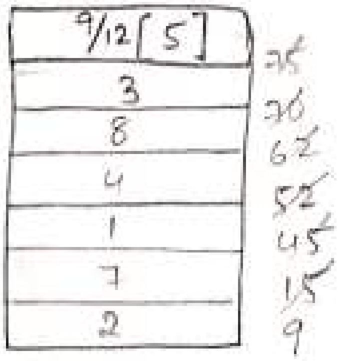
i	1	2	3	4	5	6	7	8
$\frac{P_i}{\omega_i}$	$\frac{30}{10}=3$	$\frac{20}{5}=4$	$\frac{10}{6}=1.66$	$\frac{15}{7}=2.14$	$\frac{18}{12}=1.5$	$\frac{35}{25}=1.4$	$\frac{25}{8}=3.125$	$\frac{51}{30}=1.7$

2) Arrange the i in descending order of $\frac{p_i}{w_i}$

i	2	7	1	4	8	3	5	6
p_i	20	25	30	15	51	10	18	35
w_i	5	8	10	7	30	6	12	25

3) Greedy solution:-

Knapsack   profit   Solution



$0+20+25+30+15+51+10+\frac{9}{12}(18)$ $\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ 1 & 1 & 1 & 1 & 9/12 & 0 & 1 & 1 \end{pmatrix}$

$= 151 + 13.5$

$= 164.5$

Hence the optimal solution $(x1, x2, x3, x4, x5, x6, x7, x8) =$
$= (1, 1, 1, 1, 9/12, 0, 1, 1)$ and maximum profit is 164.5

22-05-2025 Friday (5m)

*Explain job sequencing with deadlines problem

· we are given a set of n jobs. Associated with job i is an integer deadline $d_i \geq 0$ and a profit $p_i > 0$. For only job i, the profit p_i is earned if the job is completed by its deadline. To complete a job, one has to process the job on a machine for one unit. of time only one machine is available for processing jobs.

· A feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline.

· The value of a feasible solution J is the sum of the profits of the jobs in J

that is $\sum_{i \in J} P_i$

An optimal solution is a feasible solution with maximum value.

Example for job sequencing with deadlines:

Let $n=4$, $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ and $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

The feasible solutions and their values are given below

S.NO	Feasible Solution J	Processing Sequence	Value
1	{1, 2}	2, 1	10 + 100 = 110
2	{1, 3}	3, 1 or 1, 3	15 + 100 = 115
3	{1, 4}	4, 1	27 + 100 = 127
4	{2, 3}	2, 3	10 + 15 = 25

Clearly the solution 3 is optimal. In this solution only jobs 1 and 4 are processed value is 127.

*Q: Find an optimal solution using greedy method to the following instances of job sequencing with deadlines & profit problem

$n=7$, $(P_1, P_2, \ldots P_7) = (3, 5, 20, 18, 1, 6, 30)$ and $(d_1, d_2, \ldots d_7) = (1, 3, 4, 3, 2, 1, 2)$

Q: Find optimal solution of following:-

i	1	2	3	4	5	6	7	8
P_i	15	18	2	9	16	21	23	10
d_i	5	4	6	4	2	4	6	5

J_7	J_6	J_2	J_5	J_1	J_8	J_4	J_3
23	21	18	16	15	10	~~9~~	~~2~~
6	4	4	2	5	5	~~4~~	~~6~~

$\underline{J_8}$ $\underline{J_5}$ $\underline{J_2}$ $\underline{J_6}$ $\underline{J_1}$ $\underline{J_7}$

Profit $= 23 + 10 + 16 + 18 + 21 + 15$

$= 103$

Q: $n = 5$, $(P_1, P_2 \dots P_5) = (20, 13, 10, 4, 1)$, $(d_1, d_2, \dots d_5) = (2, 1, 2, 3, 3)$

J_2	J_1	J_4
1	2	3

$= 20 + 13 + 4$

$= 37$

Q: $n = 7$, J 1 2 3 4 5 6 7

P_i

d_i

J_6 J_7 J_4 J_3

$= 6 + 30 + 18 + 20$

$= 74$

Q: $P_i = 99, 67, 45, 34, 23, 10$; $d_i = (2, 3, 1, 4, 5, 2)$.

$\overline{45}$ $\overline{99}$ $\overline{67}$ $\overline{34}$ $\overline{23}$ $= 268.$

Q: Solve the following job sequencing problem $(P_1, P_2, \dots P_5) = (10, 3, 33, 11, 40)$ & $(d_1, d_2 \dots d_5) = (3, 1, 1, 2, 2)$

Arange the jobs in non increasing order of profits.

Jobs	5	3	4	1	2
profits	40	33	11	10	3
deadlines	2	1	2	3	1

Consider the jobs in the above non-increasing order of profits. Add a job i to solution J if $J \cup \{i\}$ is feasible otherwise discard it.

S.No	Job considered	Action taken	Feasible solution	Processing sequence	Profit Value $\sum_{i \in J} P_i$
1	—	—	$J = \phi$	—	0
2	5	added to J	$J = \{5\}$	5	40
3	3	added to J	$J = \{3, 5\}$	3, 5	33+40=73
4	4	discarded	$J = \{3, 5\}$	3, 5	73
5	1	added to J	$J = \{1, 3, 5\}$	3, 5, 1	33+40+10=83
6	2	discarded	$J = \{1, 3, 5\}$	3, 5, 1	83

Optimal sequence of jobs is J_3, J_5, J_1

Optimal profit is 83.

02-09-2025 Tuesday

* Job Sequencing Problem: Pseudo code

//n: number of jobs   J: Array of job numbers of size n

//P: Array of profits of size n   //w: Array of deadlines of size n

//slot: Array to store the jobs which can be completed.

```
Algorithm JobSequence (J, P, d, n) {
    for i=1 to n
        slot[i] = 0;
    profit = 0;
    for i=1 to n
    {
        K = d[i];
        while (K ≥ 1)
            if (slot[K] = 0)
            {
                slot[K] = J[i];
                profit = profit + p[i];
                break;
            }
            else
                K = K-1;
    }
    return profit;
}
```

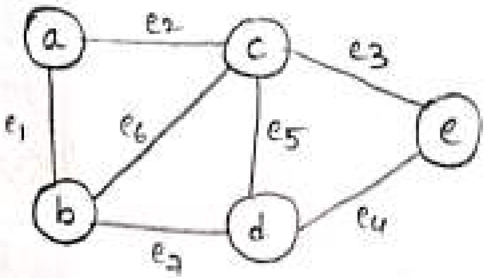
$TC = O(n^2)$

05-09-2025 Friday

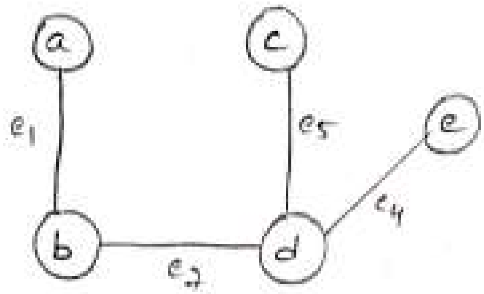
* Spanning tree:-

A tree (i.e, connected, acyclic graph) which contains all the vertices of the graph is called spanning tree.

Egr



Graph G



Spanning Tree T

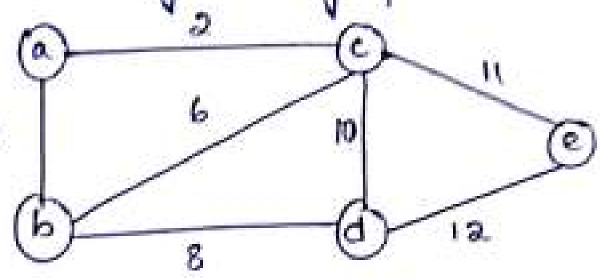
Note:- • A Graph can have more than one spanning tree.

· Every spanning tree with n vertices has n-1 edges.

* Weighted Graph:-

A graph in which each edge has an associated weight is called weighted graph.

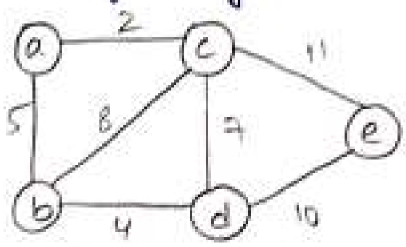
Egr



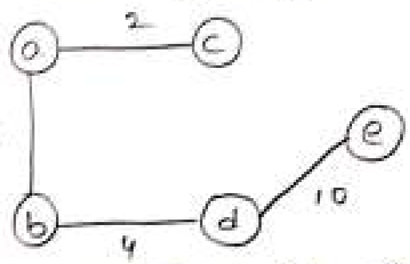
* Minimal Spanning Tree:-

Spanning tree with the minimum sum of weights is called minimal spanning tree or minimum cost spanning tree.

Egr



Weighted graph



Minimal Spanning Tree

weight (MST) = 2+5+4+10 = 21

* Two algorithms to find MST

1) Prim's algorithm

2) Kruskal's algorithm

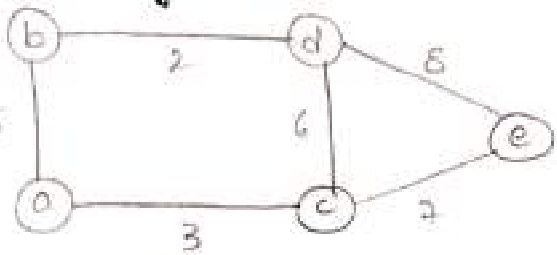
* Prim's Algorithm:-

If A is a subset of a minimal spanning tree, then the edges of A always from a single tree.

* Kruskal's Algorithm:-

If A is a subset of a minimal spanning tree, then the edges of A need not form a single tree.

*Explain Kruskal's algorithm:—

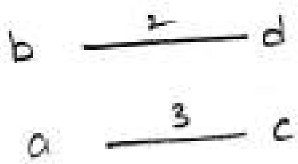


1) Arrange the edges in non-decreasing order of their weights

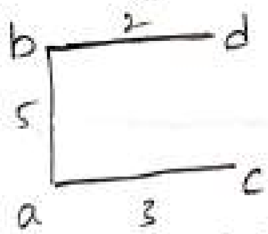
edge	bd	ac	ab	cd	ce	de
weight	2	3	5	6	7	8
Status	✓	✓	✓	×	✓	×

1. Initially spanning tree $T = \phi$
2. The edge 'bd' is having minimum weight. So, add the edge 'bd' to T

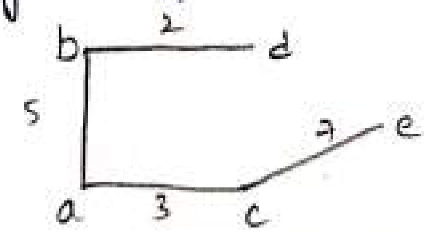
3. Add the edge ac to T



4. ab is the edge with next minimal weight add 'ab' to T



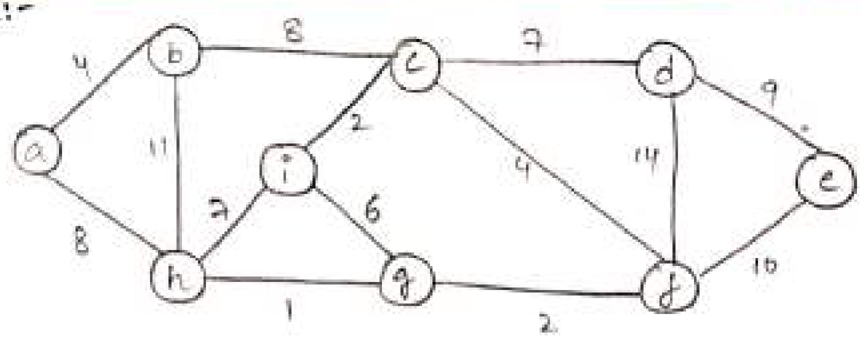
5. The next minimal weight edge is cd but the addition of it to T forms a cycle. So don't add it. The next minimum weight edge is ce. So, add it to T



we got n-1 edges so the minimial ST = 2+3+5+7 = 17

06-09-2025 Saturday

Example:-

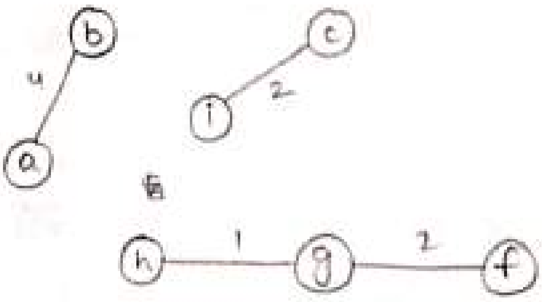


1) $h \xrightarrow{1} g$

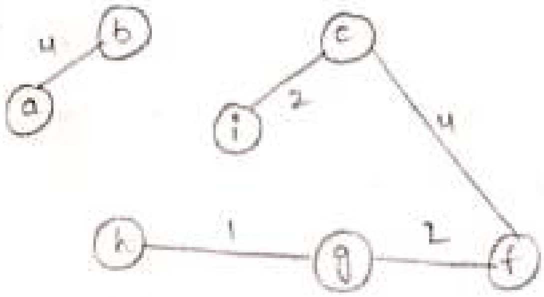
2) $h \xrightarrow{1} g \xrightarrow{2} f$

3)

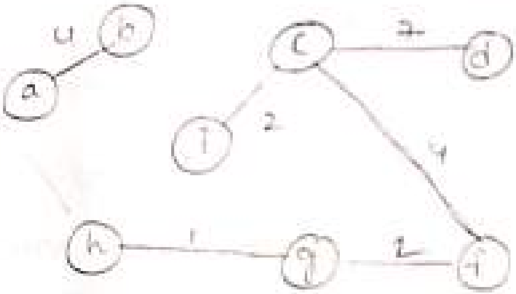
4)



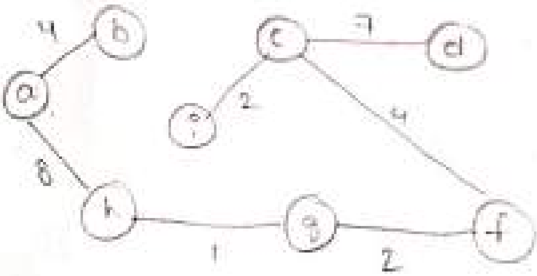
5)



6)

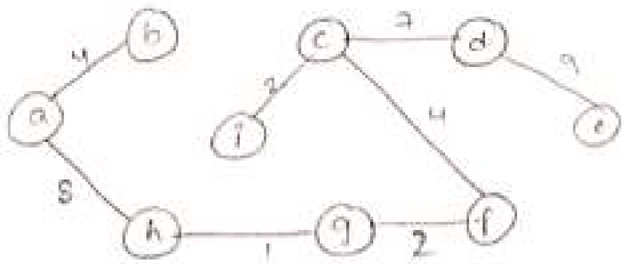


7)



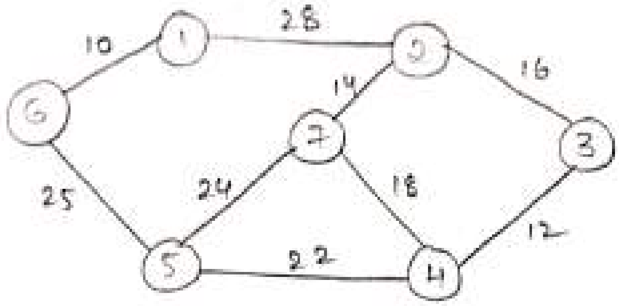
8 lines

8)

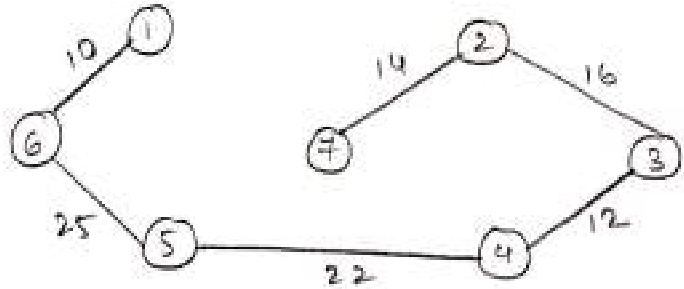


$w = 4+8+1+2+2+4+7+9$
$= 37$

* Exampler



Ans:



Weight of MST = 99

* Kruskal's algorithm:-

```
Kruskal (V, E, ω)
{
   A = ∅;
   for each vertex v ∈ V
        MAKE-SET(v);
   sort E into non-decreasing order by weights ω;
   for each (u,v) taken from the sorted list
        if (FIND-SET(u) ≠ FIND-SET(v))
             A = A ∪ {(u,v)};
             UNION (u,v);
   return A;
}
```

Time complexity: $O(|E| \log |V|)$

or

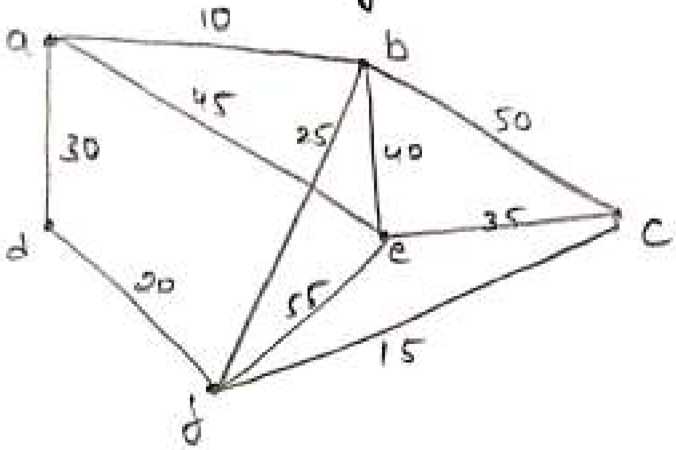
$\Rightarrow O(e \log e)$

or

$\Rightarrow O(e \log v)$

05-09-2025
Monday

* Explain 'Prim's Algorithm:- with an example?



1. Start with vertex 'a'

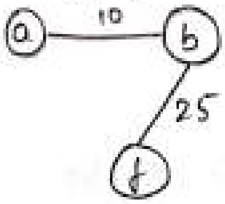
(a)

2. Fringe edges : ab, ad, ae

Add ab to MST

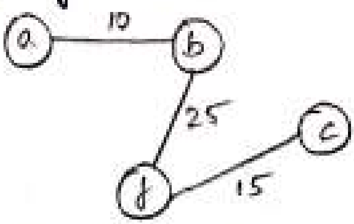
3. Fringe edges : ad, ae, bc, bf, be

Add bf to MST



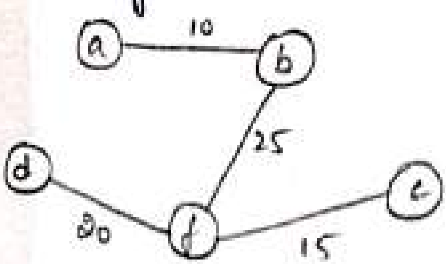
4. Fringe edges : ad, ae, bc, be, fd, fc, fe

Add fc to MST



5. Fringe edges: ad, ae, bc, be, fd, fe, ce

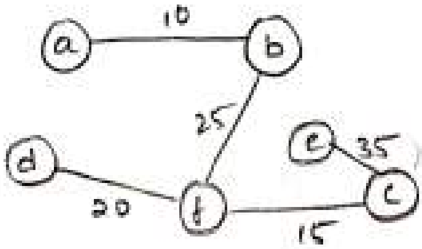
Add fd to MST



6. Fringe edges: ad, ae, bc, be, fe, ce

ad forms a cycle

add ce to MST

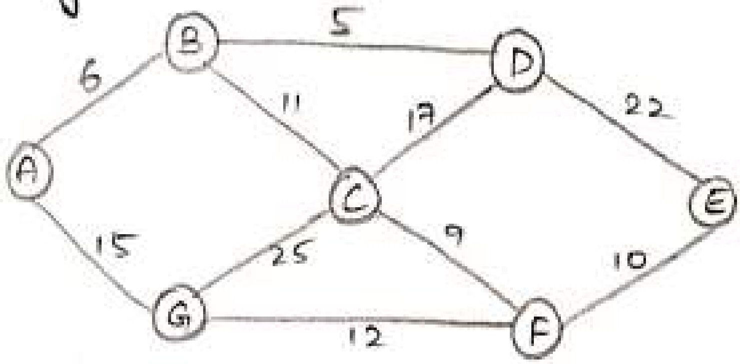


∴ The cost of the minimial Spanning tree T

$= 10 + 25 + 20 + 15 + 35$

$= 105$

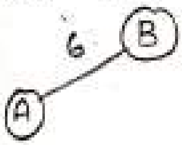
* Find the minimial spanning tree of following graph using prim's algorithm?



1. Start with vertex 'A'

2. fringe edges: AB, AG

Add AB to MST

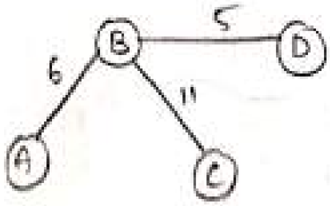


3. Fringe edges: AG, BD, BC

Add BD to MST

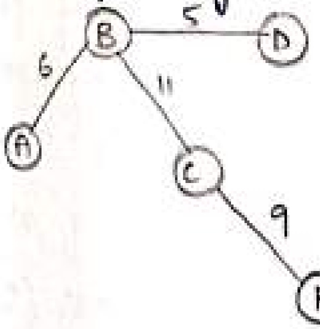
4. Fringe edges: AG, BC, DC, DE

Add BC to MST



5. Fringe edges: AG, DC, DE, CG, CF

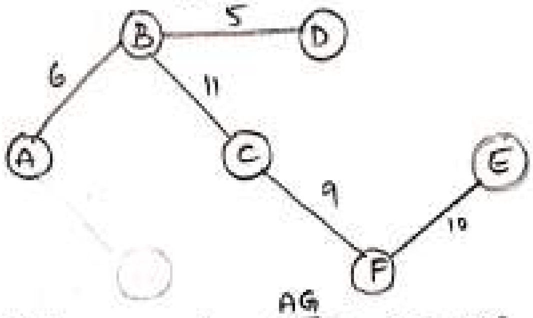
Add CF to MST



6. Fringe edges: AG, DC, DE, CG, FE

15 17 22 25 10

Add ~~FE~~ to MST



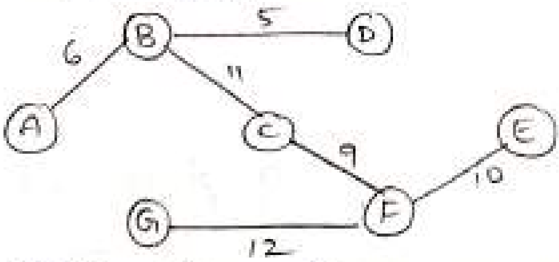
7. Fringe edges: AG, DC, DE, CG, FG, ED

15 × 17 22 25 12 22

~~Add DC to~~

~~DC forms a cycle.~~

Add ~~FG~~ to MST



8. Fringe edges:- AG, DC, DE, CG, ED

Now number of edges T is 6. The no. of vertices n = 7

∴ edges in T = n − 1

So stop process

The required MST is

The cost of MST = 6+5+11+9+12+10

= 53

* Prim's Algorithm:-

//r: initial vertex or root vertex //ω: weight matrix

```
Algorithm Prim(V, E, r, ω){
    Q= ∅; // Vertices not included in tree forms a priority Queue Q
    for each u ∈ V
    {
        u.key = ∞;      u.π = NIL;
        INSERT (Q,u);
    }
    DECREASE - KEY (Q,r,0);   // key[r]=0
    while (Q ≠ ∅)
    {
        u = EXTRACT - MIN(Q);
        for each v ∈ Adj [u]
            if (v ∈ Q and ω(u,v) < v.key)
            {
                v.π = u;
                DECREASE - KEY (Q,v,ω (u,v));  // v.key = ω(u,v)
            }
    }
}
```

$TC = O(e \log v)$

Q: write difference between kruskal's and prim's algorithm

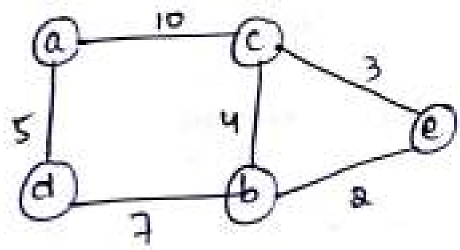
Kruskal's Algorithm	Prim's Algorithm
1.) If A is a subset of a minimal spanning tree, then the edges of A need not to form a single tree.	1. If A is a subset of MST the edges of A always form a single tree.
2.) Starts with all the vertices of the graph as a forest and every addition of edge	2.) Starts with any vertex of the graph arbitrarily. At any point of times the answer

takes a forest one step further towards a complete tree.	is a tree.
3.) The concept of Kruskal's algorithm is based on the acyclic nature of the graph.	3.) The concept of prim's algorithm is based on the connectedness.
4.) Adding of an edge is performed by selecting the sorted ~~array~~. edge.	4.) Always a new vertex is joined to an old vertex.
5.) Next edge is always determined by the edgelist.	5.) Addition of nodes is based on the concept of shortest distance or weight.

* Weight matrix:-

The matrix which represents weights of edges of a weighted graph is called a weight matrix.

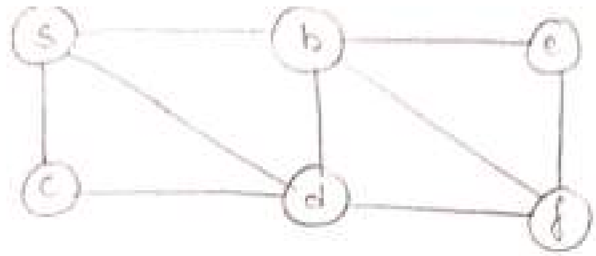
Eg:-



$$
\begin{array}{c c}
 & \begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} &
\begin{bmatrix}
0 & \infty & 10 & 5 & \infty \\
\infty & 0 & 4 & 7 & 2 \\
10 & 4 & 0 & \infty & 3 \\
5 & 7 & \infty & 0 & \infty \\
\infty & 2 & 3 & \infty & 0
\end{bmatrix}
\end{array}
$$

Q: Explain single source shortest path problem:

From a given source vertex, we need to find the shortest paths and shortest distances to all the remaining vertices. This is called a single source shortest path problem.



Vertex	Shortest path	Shortest distance
b	s-b	5
c	s-c	4
d	s-b-d	6
e	s-b-d-f-e	10
f	s-b-d-f	8

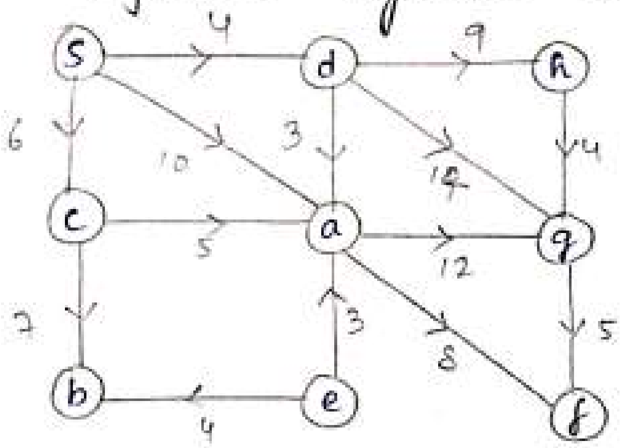
There are mainly two algorithms to solve single source shortest path problem.

1. Dijkstra's algorithm 2. Bellman-ford algorithm.

Q: Write differences b/w Dijkstra's algorithm & Bellman-ford algorithm:

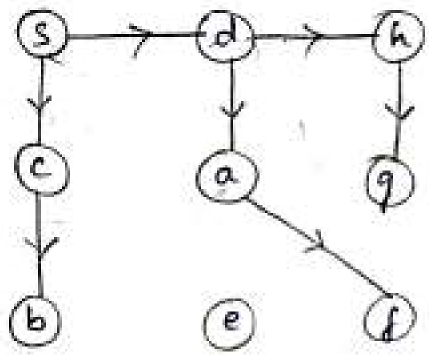
Dijkstra's Algorithm	Bellman-Ford Algorithm
1.) It is a greedy method	1.) It follows dynamic program
2.) It can be applied only when all edge weights are positive.	2.) It can be applied to the graphs with negative weights also.
3.) It doesn't work well when the graph has negative edge weights.	3.) It don't work when the graph has a cycle of negative length.
4.) Time complexity:- $O(n^2)$	4.) Time complexity:-

Q: Explain Dijkstra's algorithm with an example.

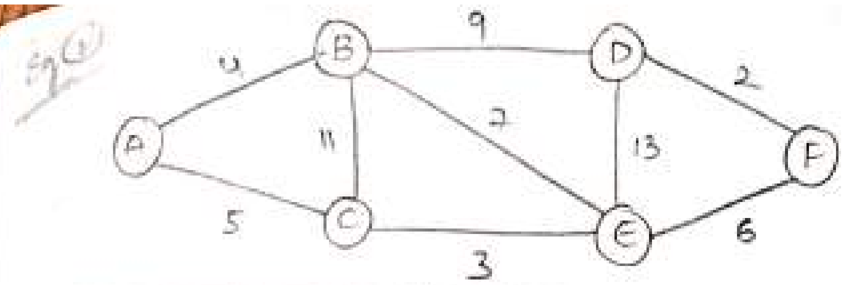


Finalized vertices	w.r.t vertex	s	a	b	c	d	e	f	g	h
—	—	∞	∞	∞	∞	∞	∞	∞	∞	∞
{s}	s	0	10	∞	6	4	∞	∞	∞	∞
{s, d}	d	0	7	∞	6	4	∞	∞	18	13
{s, d, c}	c	0	7	13	6	4	∞	∞	18	13
{s, d, c, a}	a	0	7	13	6	4	∞	15	18	13
{s, d, c, a, b}	b	0	7	13	6	4	∞	15	18	13
{s, d, c, a, b, h}	h	0	7	13	6	4	∞	15	17	13
{s, d, c, a, b, h, f}	f	0	7	13	6	4	∞	15	17	13
{s, d, c, a, b, h, f, g}	g	0	7	13	6	4	∞	15	17	13

The shortest path tree is ⟶



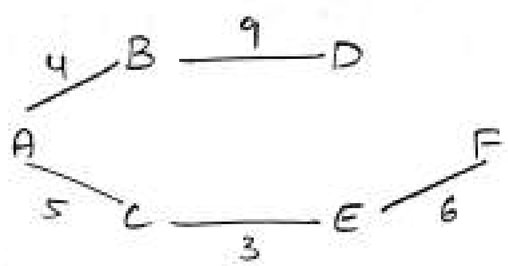
Vertex	Shortest path	Shortest distance
a	s-d-a	7
b	s-c-b	13
c	s-c	6
d	s-d	4
e	unreachable	∞
f	s-d-a-f	15
g	s-d-h-g	17
h	s-d-h	13



Finalised vertices	w.r.t vertex	A	B	C	D	E	F
—	—	∞	∞	∞	∞	∞	∞
{A}	A	0	4	5	∞	∞	∞
{A,B}	B	0	4	5	13	11	∞
{A,B,C}	C	0	4	5	13	8	∞
{A,B,C,E}	E	0	4	5	13	8	14
{A,B,C,E,D}	D	0	4	5	13	8	14
{A,B,C,E,D,F}	F	0	4	5	13	8	14

Shortest path

B	A – B	4
C	A – C	5
D	A – B – D	13
E	A – C – E	8
F	A – C – E – F	14



* Dijkstra's algorithm:–

Dijkstra (a, ω, S)

// n: number of vertices

// src: source vertex

// w: weight matrix of order n×n

// d: Array to store length of shortest path.

// S: Boolean array of n vertices which indicate shortest distance to vertex finalised or not.

```
Algorithm Dijkstra (w, src, n)
{
    for i=1 to n
    {
        s[i]:=false;
        d[i]:= w[src][i];
    }
    s[src]:= true;
    d[src]=0;
    for i=2 to n
    {
        choose u among non-finalised vertices such that d[u] is
        minimum;
        s[u]:= true;
        for(each adjacent vertex v of u)
        if((s[v]==false) and (d[v]>d[u]+w[u][v]))
            d[v]:= d[u]+w[u][v];
    }
}
```

T.C = O(n^2)

```
⇒ Static void Dijkstra's(int[][]w, int src, int n)
{
    int i,j,u;
    int d[] = new int[n];
    boolean[] s = new boolean[n];
    for(i=0; i<n; i++)
        d[i]= 9999;
    d[src]=0;
    for(i=0; i<n; i++)
    {
        u=minVertex(s,d,n);
        s[u]=true;
        for(j=0; j<n; j++)
            if(w[u][j]!=0)
                if(s[j]==false && d[j]>d[u]+w[u][j])
                    d[j]= d[u]+w[u][j];
    }
    PrintAns(d,n);
}
```