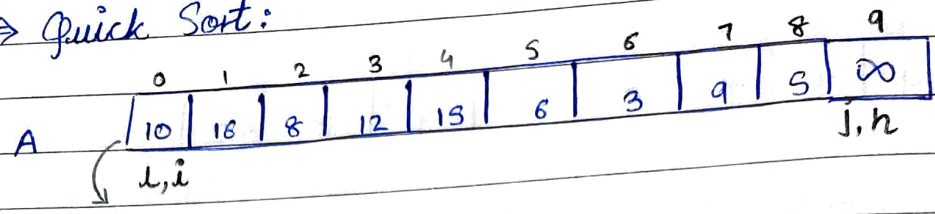
⇒ Quick Sort:

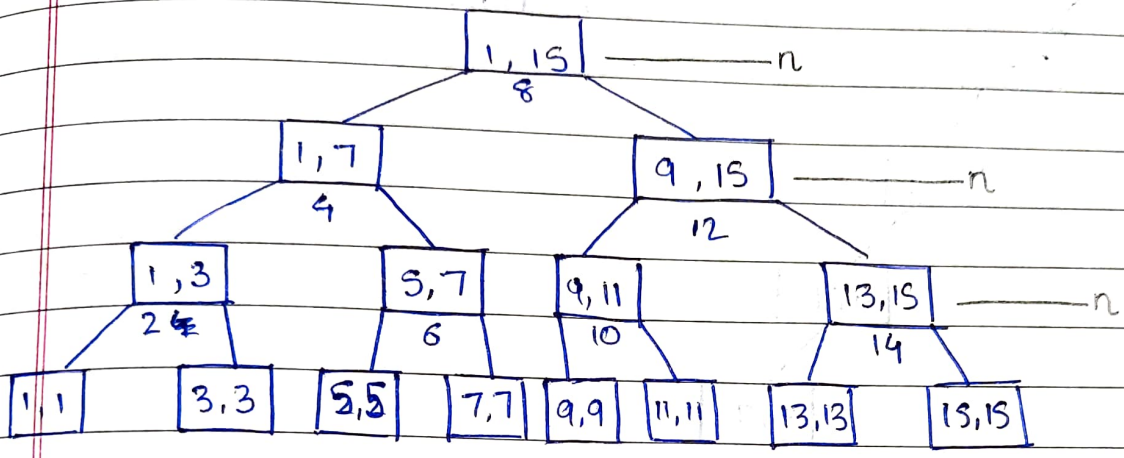



```
Partition (l, h) {
    pivot = A[l];
    i = l; j = h;
    while (i < j) {
        do {
            i++;
        } while (A[i] ≤ pivot);
        do {
            j--;
        } while (A[j] > pivot);
        if (i < j) {
            swap (A[i], A[j]);
        }
    }
    swap (A[l], A[j]);
    return j;
}

QuickSort (l, h) {
    if (l < h) {
        j = partition (l, h);
        QuickSort (l, j);
        QuickSort (j+1, h);
    }
}
```

Time Complexity :

Eg, A = [1, ... , 15] [If the partition is in the middle]



K steps, time per step = n

$\therefore T(n) = nK$

Assume, $\frac{n}{2^K} = 1$

$n = 2^K$

$K = \log_2 n$

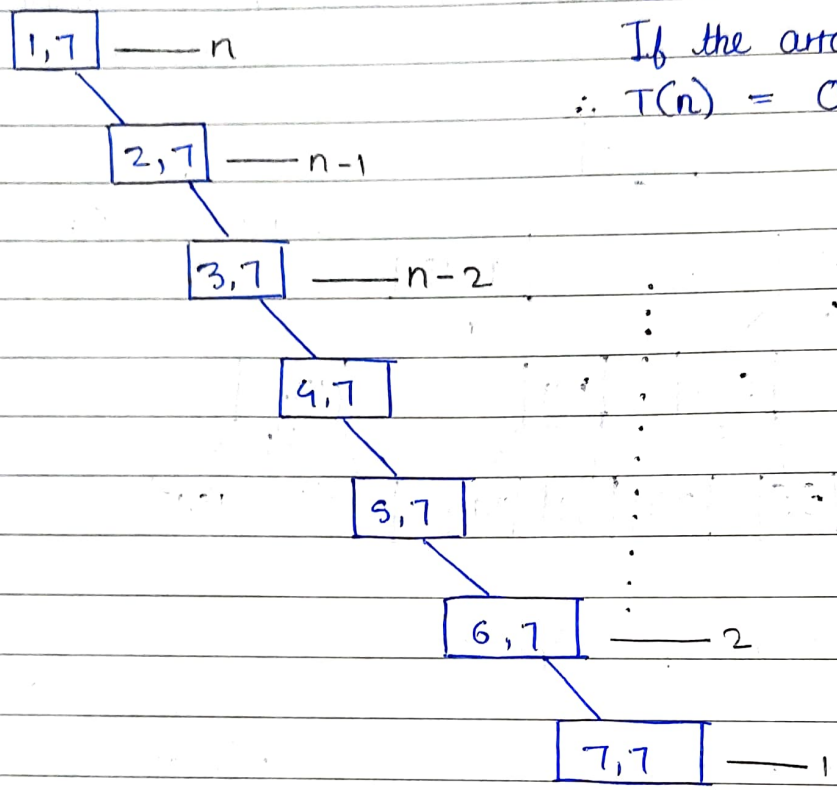
$\therefore T(n) = n \log n$

$\therefore O(n \log n)$

$\therefore$ Best Case:
If the partition is done in the middle,
$\therefore T(n) = O(n \log n)$

Eg, A = [2 4 8 10 16 18 17]

Worst Case:
If the array is already sorted
$\therefore T(n) = O(n^2)$



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

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

$T(n) = O(n^2)$

Note: To improve worst case to best case:
If the array is sorted,
(i) Select middle element at pivot [More accurate]
(ii) Select random element as pivot [Less accurate]

⇒ Binary Search :

- The array / list should be already sorted.

A =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	8	12	14	17	25	29	31	36	42	47	53	55	62

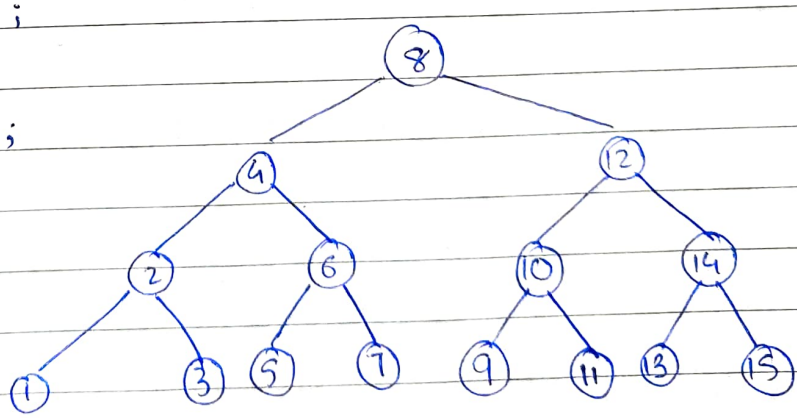
l, mid, l_2, mid_2, h_2, mid_{-2}, h

key = 42

l	h	mid = $\frac{l+h}{2}$
1	15	$\frac{15+1}{2} = 8$
9	15	$\frac{9+15}{2} = 12$
9	11	$\frac{9+11}{2} = 10$
11	11	$\frac{11+11}{2} = 11$

[Iterative Method]

```
BinarySearch (A, n, Key) {
    l = 1 , h = n ;
    while (l ≤ h) {
        mid = (l+h) / 2;
        if ( Key == A [mid ]) {
            return mid ;
        } else if ( Key < A [mid ]) {
            h = mid - 1 ;
        } else {
            l = mid + 1 ;
        }
    } return 0 ;
}
```



Time complexity :

min ⟶ O(1)

max ⟶ O(log n)

```
RBinarySearch (l, h, key) {                    ——— T(n)
  if (l==h) {
    if (A[l] == key) {
      return l;
    else {
      return 0;
    }
  else {
    mid = l+h/2 ;
    if (key ==A[mid]) {
      return mid;
    if (key < A[mid]) {
      return RBinarySearch(l, mid-1, key)
    }
    else {
      return RBinarySearch(mid+1, h, key)
    }
  }
}
```

1

1

T(n/2)

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

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

⇒ Min - Max :

A

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

[Iterative method]

```
Max-Min (A, n) {
    max = min = A[0];
    for (i = 1; i < n; i++) {          ——— n+1
        if (a[i] > max) {
            max = a[i];
        }
        else if (a[i] < min) {
            min = a[i];
        }
        return (max, min);
    }
}
```

∴ $t(n) = O(n)$

```
Max - Min (A, i, j, max, min) {
    if (i == j) {
        max = min = A[i];
    } else if (i == j-1) {
        if (A[i] < A[j]) {
            min = A[i], max = A[j];
        } else {
            min = A[j], max = A[i];
        }
    } else {
        mid = i + j / 2;
        Min - Max (A, i, mid, max, min);
        Min - Max (A, mid+1, j, max1, min1);
        if (max < max1) then max = max1;
        if (min <= min1) then min = min1;
```

Time Complexity :

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

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

⇒ Greedy Method :

- Used for solving optimization problems.
- Solving a problem can have ~~mult~~ multiple solutions, but only few feasable solutions depending on the problem condition.
- With multiple feasable solutions, there can be only a single optimal solution.

```
Algo Greedy (A, n) {
    for (i = 1 to n) {
        x = Select (a);
        if Feasible (x) {
            Solution = Solution + x;
        }
    }
}
```

⇒ Knapsack:

	1	2	3	4	5	6	7
1. Objects :	0	1	2	3	4	5	6	7
Profits :	10	5	15	7	6	18	3
Weights :	2	3	5	7	1	4	1

$n = 7$, $m = 15$

P/W :	5	1.3	3	1	6	4.5	3

x (1 x_1 | 2/3 x_2 | 1 x_3 | 0 x_4 | 1 x_5 | 1 x_6 | 1 x_7)

Total weight = 15 Kg

$\therefore$ $15 - 1 = 14$

$14 - 2 = 12$

$12 - 4 = 8$

$8 - 5 = 3$

$3 - 1 = 2$

$2 - 2 = 0$

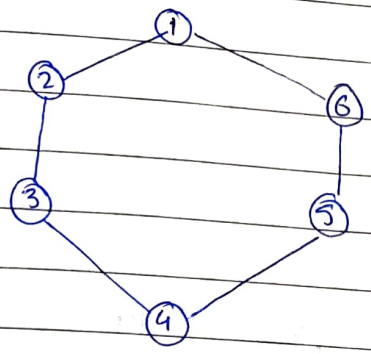
$$\sum x_i \cdot w_i = (1 \times 2) + \left(\frac{2}{3} \times 3\right) + (1 \times 5) + (0 \times 7) + (1 \times 1) + (1 \times 4) + (1 \times 1)$$

$$= 15$$

$$\sum x_i \cdot p_i = (1 \times 10) + \left(\frac{2}{3} \times 5\right) + (1 \times 15) + (0 \times 7) + (1 \times 6) + (1 \times 18) + (1 \times 3)$$

$$= 54.6$$

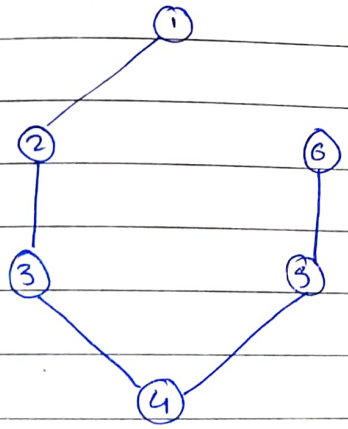
⇒ Minimum Cost Spanning Tree :



$G = (V, E)$

$V = \{1, 2, 3, 4, 5, 6\}$

$E = \{(1,2), (2,3), (3,4), \ldots\}$



$|V| = n = 6$

$E = |V| - 1 = 5$

- Spanning tree is the sub-graph of a graph having all vertices but only $(n-1)$ edges.
- Tree will not have a complete cycle.

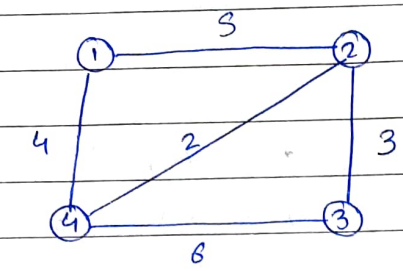
$|E| = 6$

$= {}^{6}C_{5} = 6$

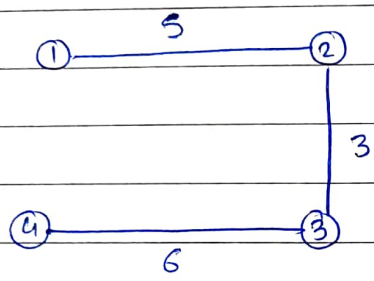
No. of possible spanning trees :

${}^{|E|}C_{|V|-1}$ - no. of complete cycles

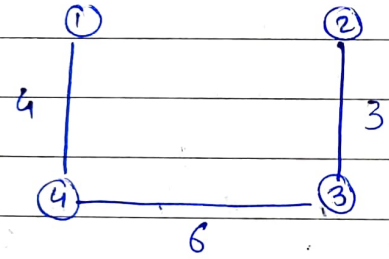
1.



Draw Spanning Tree.



cost = 14



cost = 13

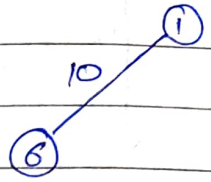
For finding minimum spanning tree of weighted graphs:

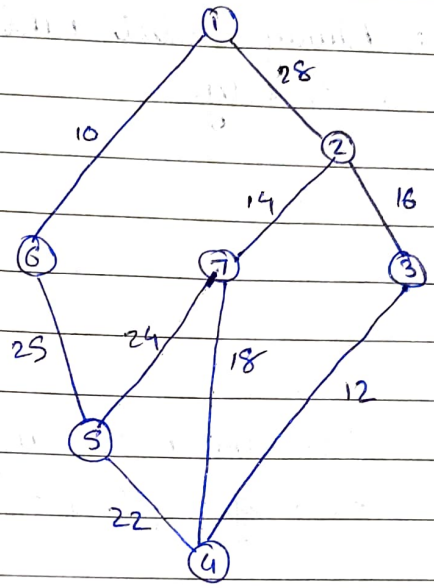
(i) Prim's

(ii) Kruskal's

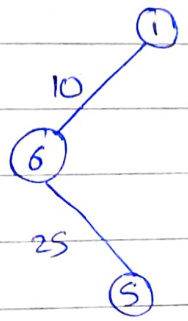
⇒ Prim's Algorithm :

1. Step 1: Select minimum cost edge

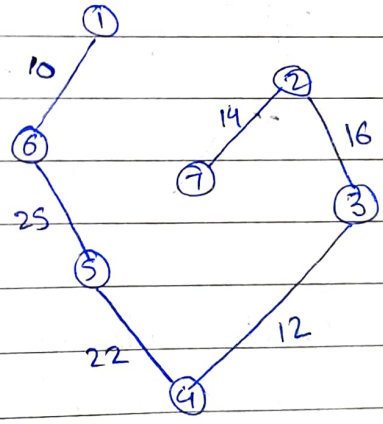




Step 2: Always select minimum cost edge, making sure it is connecting to already selected vertices.



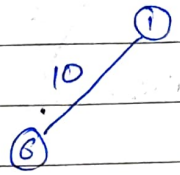
⇒

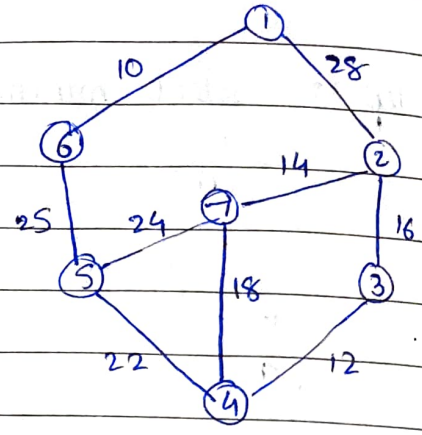


∴ Cost = 99

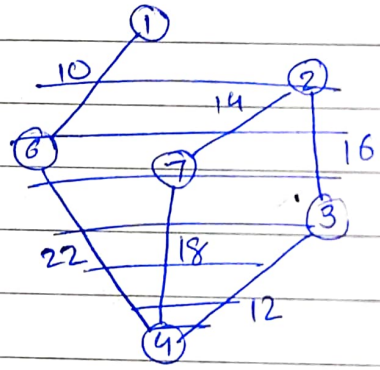
⇒ Kruskal's Algorithm:

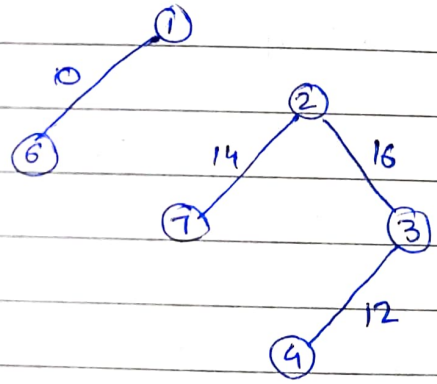
Step 1: Always select minimum cost edge.



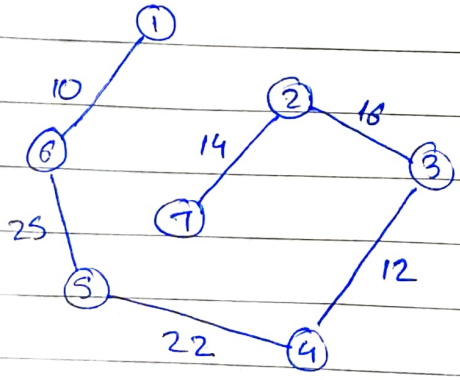


Step 2: Select the next minimum cost edge. [May or may not be connected]





Note: Next minimum cost edge is (7, 4), but it will make a closed loop. ∴ Dont include it



Cost = 99

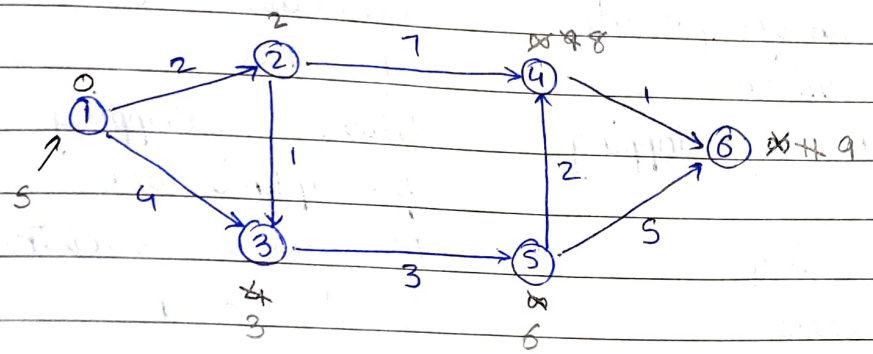
$t(n) = O(|V| . |E|)$

$= O(n.e)$

$= O(n^2)$

⇒ Dijkstra Algorithm :

- We have to find the shortest path from one vertices to all vertices.



∴

V	d[V]
2	2
3	3
4	8
5	6
6	9

$n = |V|\ |V|$

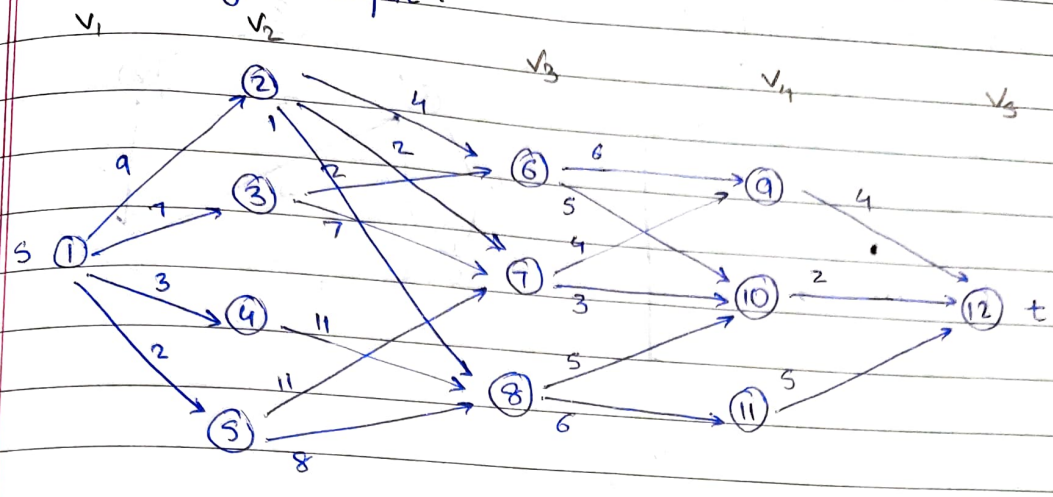
$t(n) = \Theta(|V|^2)$

$= \Theta(n^2)$ [Worst Case]

⇒ Dynamic Programming

Greedy Method	Dynamic Programming
• May not always provide optimal solution	Guarantees to find optimal sol^n if exists
• Does not handle overlapping subproblems.	Utilizes overlapping subproblems for optimization.
• No backtracking	May involve backtracking
• Generally faster	Can be slower
• Requires less memory.	Requires more memory.

⇒ Multistage Graph :



V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2/3	7	6	8	8	10	10	10	12	12	12	12

stage ↑ Vertex

$$cost(3,6) = \min [\, c(6,9) + cost(4,9),\ c(6,10) + cost(4,10) \,]$$
$$= \min [\, 6+4,\ 5+2 \,]$$
$$= 7$$

$$\therefore \boxed{cost(i,j) = \min [\, c(j,l) + cost(i+1,l) \,]}$$

stage ↑ Vertex

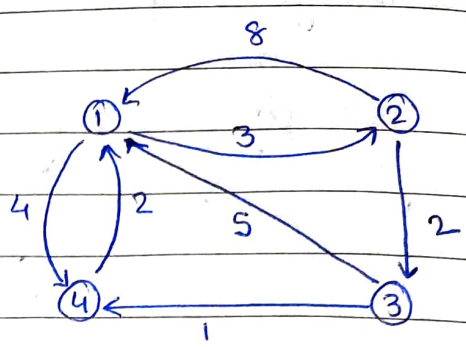
Now, $d(1,1) = 2$ $\quad d(1,1) = 3$

$d(2,2) = 7$ $\quad d(2,3) = 6$

$d(3,7) = 10$ $\quad d(3,6) = 10$

$d(4,10) = 12$ $\quad d(4,10) = 12$

⇒ All Pair Shortest Path :



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & \infty & 0 \end{bmatrix} \end{array}$$

$A^0[2,3]$ $\quad$ $A^0[2,1] + A^0[1,3]$

$2 \quad < \quad 8 + \infty$

$A^0[2,4]$ $\quad$ $A^0[2,1] + A^0[1,4]$

$\infty \quad\quad 8 + 7$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & 7 & 0 \end{bmatrix} \end{array}$$

$A^1[1,3]$ $\quad$ $A^1[1,2] + A^1[2,3]$

$\infty \quad > \quad 3 + 2$

A^3 =

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

$A^2[1,2]$ $<$ $A^2[1,3] + A^2[3,2]$

3 $<$ 5 + 8

Similarly, A^4 =

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

$$A^K[i,j] = \min [A^{K-1}[i,j] , A^{K-1}[i,K] + A^{K-1}[K,j]]$$

```
Algo APSP(i, j, K) {
    for (K = 1 ; K ≤ n ; K++) {
        for (i = 1 ; i ≤ n ; i++) {
            for (j = 1 ; j ≤ n ; j++) {
                A[i,j] = min {A[i,j], A[i,K] + A[K,j]}
            }
        }
    }
}
```

$t^{(n)} = O(n^3)$

⇒ Matrix Chain Multiplication:

A_1	A_2	A_3	A_4
(5×4)	(4×6)	(6×2)	(2×7)

m

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

m[1,2]

$A_1 \cdot A_2$

$(5\times4)\ (4\times6)$

$5\times4\times6 = 120$

m[1,3]

$A_1.(A_2.A_3)$	$(A_1.A_2).A_3$
$(5\times4)\ (4\times6)\ (6\times2)$	$(5\times4)\ (4\times6)\ (6\times2)$
$m[1,1] + m[2,3] +$	$m[1,2] + m[3,3]$
$5\times4\times2$	$+ 5\times6\times2$
$= 88$	$= 180$

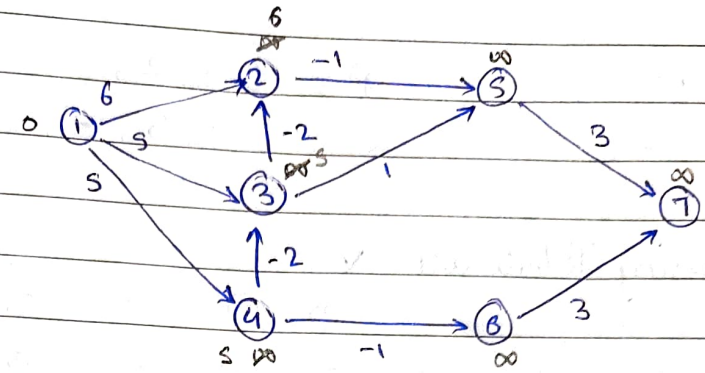
S	1	2	3	4
1	0	1	1	3
2			2	3
3				3
4				

$$m[1,4] = \min\{\ m[1,1] + m[2,4] + 5\times4\times7,$$
$$m[1,2] + m[3,4] + 5\times6\times7,$$
$$m[1,3] + m[4,4] + 5\times2\times7\ \}$$

$=$

⇒ Single Source Shortest Path:

Basically, Dijkstras hi hai, but you have to repeat the whole procedure [|V|-1] times.



Ye question mein, repeat the procedure ~~(6-1)~~ (7-1) = 6 times.

:)

⇒ 0/1 Knapsack : [Cannot take fractions]

$m = 8$ $\quad P = \{1, 2, 5, 6\}$

$n = 4$ $\quad w = \{2, 3, 4, 5\}$

Using Tabulation V

		w								
		0	1	2	3	4	5	6	7	8
P_i	w_i	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3
5	4	3	0	0	1	2	5	5	6	7
6	5	4	0	0	1	2	5	6	6	7

$$V[i, w] = \max \{ V[i-1, w], V[i-1, w - w[i]] + P[i] \}$$

x_1	x_2	x_3	x_4	
0	1	0	1	$8 - 6 = 2$
				$2 - 2 = 0$

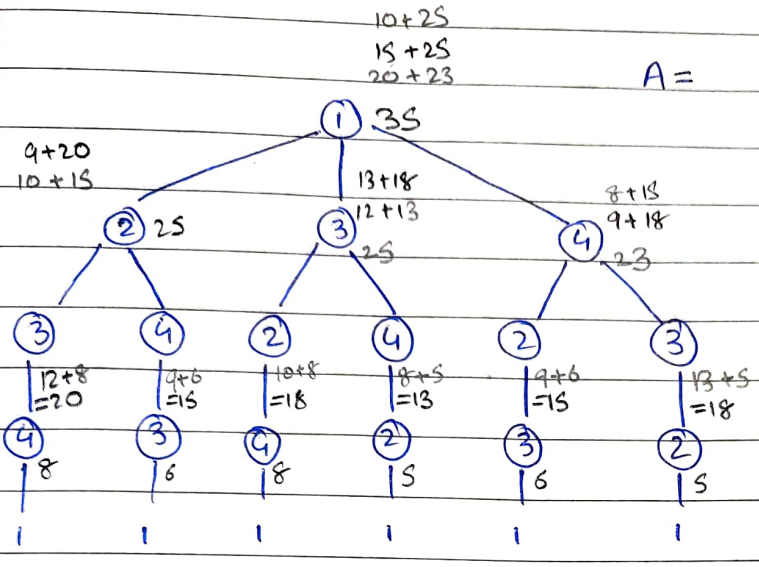
Using Sets,

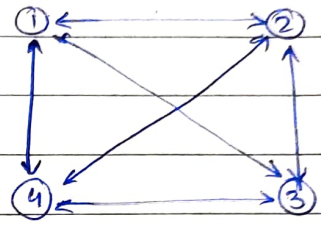
Watch Abdul Bari. Thanks ::

⇒ Travelling Salesman Problem :

$A =$

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0





$$g(1, \{2,3,4\}) = \min \{ C_{1K} + g(K, \{2,3,4\} - \{K\}) \}$$

$$g(i, S) = \min_{K \in S} \{ C_{iK} + g(K, S - \{K\}) \}$$