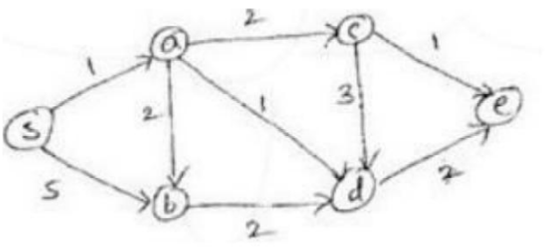
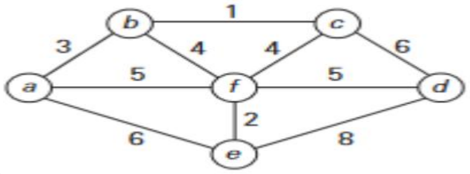
USN

Internal Assessment Test 1 – March 2025

Sub:	Analysis & Design of Algorithms			Sub Code:	BCS401	Branch:	AIDS & CSE(AIDS)	
Date:	**26/3/2025**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**IV -A, B & C**	**OBE**

Answer any FIVE FULL Questions			**MARKS**	**CO**	**RBT**
1	a	Obtain the topological sort for the graph (take DAG with 7 vertices) by using source removal method and DFS method	4	1	L2
	b	Define asymptotic notations for worst case, best case and average case time complexities with example.	6	1	L1
2	a	Solve it by recursive tree method $T(n) = 2T(n/2)+n^2$.	5	1	L3
	b	Write a recursive algorithm to search for a key element in an array of size n. Derive an equation for the best-case and worst-case complexity of your algorithm.	5	1	L3
3	a	Solve the given graph using Dijistra's method where Source is B.  2 1 1 3 2 1 5 2 2 a c s e b d	6	3	L2
	b	Construct minimum cost spanning tree using Prims algorithm for the following graph.  1 3 4 4 6 5 5 2 6 8 a b c d e f ii.	4	4	L2
4	a	With neat diagram explain different steps in designing and analyzing an algorithm	5	1	L2
	b	Find the optimal tour of the following given graph in 3.a using travelling salesman problem(using exhaustive search method)	5	2	L2
5	a	Design an insertion sort algorithm and obtain its time complexity. Apply insertion sort on these elements. 25,75,40,10,20,	5	1	L2
	b	What are Huffman Trees? Construct the Huffman tree for the following data. Character A, B, C, D ,E - Probability 0.5, 0.35, 0.5, 0.1, 0.4, 0.2 Encode DAD-CBE using Huffman Encoding.	5	3	L2
6	a	Define transitive closure of a graph. Apply Warshalls algorithm to compute transitive closure of a directed graph $\begin{array}{ccccc} & a & b & c & d \\ a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 0 & 1 & 0 \end{array}$	7	4	L2
	b	What is algorithm and write its properties	3	1	L1

CI **CCI** **HoD**

Topological sorting:

→ A Topological sort of a DAG G = (V, E), is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before v in the ordering.

→ If the graph form a cycle then no linear ordering is possible.

→ Topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

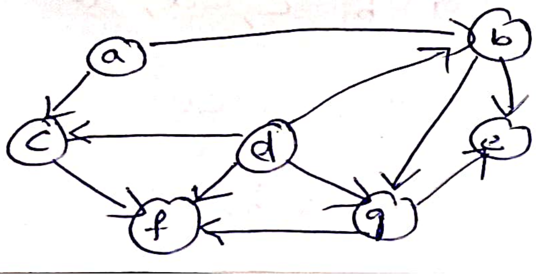
→ The topological sorting can be done using two methods

1. DFS method
2. Source Removal method.

DFS Method:

1. select any orbitary vertex.
2. When a vertex is visited for the first time, push onto the stack.
3. When a vertex become a dead end, it is removed from the stack.
4. Repeat ~~Steps 2~~ Step 2 & 3 for all the vertices in the graph.
5. Reverse the order of deleted items to get the topological sequence.

Ex:



Step	Stack	Adjacent vertex	Node visited	Stack
Initial	a	–	a	–
1.	a	b	a, b	–
2.	a, b	e	a, b, e	–
3	a, b, e	–	a, b, e	'e'
4.	a, b	g	a, b, e, g	–
5.	a, b, g	f	a, b, e, g, f	–
6.	a, b, g, f	–	a, b, e, g, f	'f'
7.	a, b, g	–	a, b, e, g, f,	'g'
8.	a, b	–	a, b, e, g, f	'b'
9.	a,	c	a, b, e, g, f, c	–
10.	a, c	–	a, b, e, g, f, c	c
11.	a	–	a, b, e, g. f, c	a
12.	d	–	a, b, e, g, f, c, d	–
13.	d	–	a, b, e, g, f, c, d	'd'
14	–	–	a, b, e, g, f, c, d.	–

Order:

d e, f, g, b, c, a, d

Topological sequence: reverse the order

d, a, c, b, g, f, e

~pological Sorting - Source Removal Method

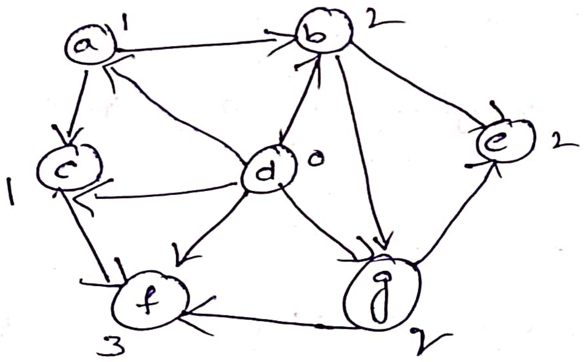
Method is based on Decrease & conquer technique.

Topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

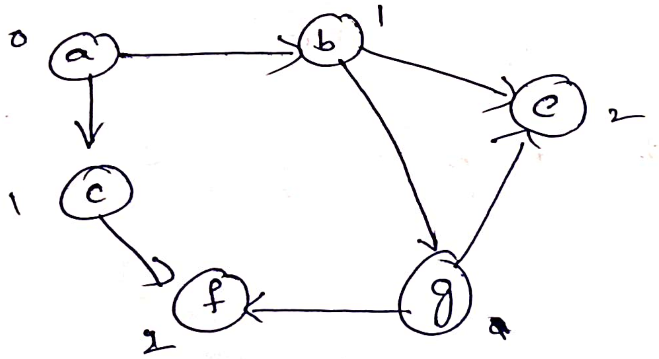
Design:

1. In the given graph identify the vertex with no incoming edges and delete along with the outgoing edges.
2. If There are several vertices with no incoming edges break the tie arbitrarily.
3. The order in which the vertices are visited & deleted one by one results in topological sorting

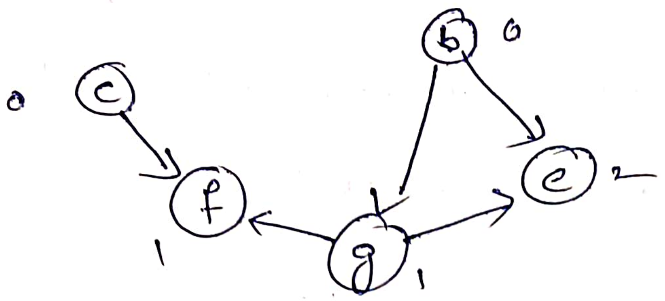
Ex:



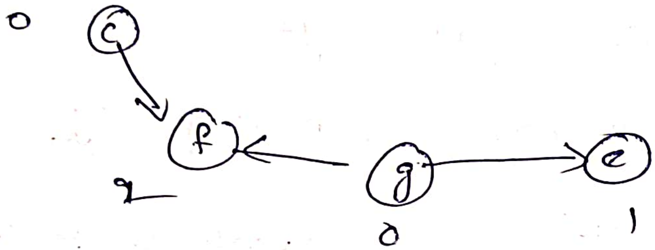
Step 1: Node with indegree 0 is d. remove d.



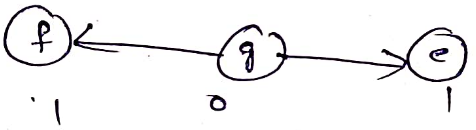
2. Next node with indegree 0 is a, remove a.



3. Next node with indegree 0 is b & c. Break the tie & remove b



Step 4: Next node with indegree '0' is c. remove c.



Step 5: Next node with indegree 0 is g. remove g

Step 6: Next node with indegree 0 is e & f. ~~remove~~ break the tie & remove 'e'

Step 7: remove 'f'

Topological sequence.

d a b c g f e

Asymptotic Notations & Basic efficiency classes.

(O, Ω, Θ)

Big "oh":

The function $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \le c * g(n)$ for all n, $n \geq n_0$.

Ex:

1) $3n+2 = O(n)$

as $3n+2 \le 4n \ \forall n \geq 2$

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

$3n+3 \le 4n$ for all $n \geq 3$

3) $100n+6 = O(n)$ as

$100n+6 \le 101n \ \forall n \geq 6.$

4) $10n^2+4n+2 = O(n^2)$

as $10n^2+4n+2 \le 11n^2 \ \forall n \geq 5$

5) $1000n^2 + 100n - 6 = O(n^2)$ as

$1000n^2 + 100n - 6 \leq 1001n^2 \ \forall \ n \geq 100$

6) $6 * 2^n + n^2 = O(\cancel{n}2^n)$

$6 * 2^n + n^2 \leq 7 * 2^n \ \forall \ n \geq 4$

7) $3n + 3 = O(n^2)$

$\cancel{3n+3}$ $\leq 3n^2 \ \forall \ n \geq 2$
$3n+3$

8) $10n^2 + 4n + 2 = O(n^4)$

$10n^2 + 4n + 2 \leq 10n^4 \ \forall \ n \geq 2$.

9) $3n + 2 \neq O(1)$ as $3n+2$ is not less than & equal to c for any constant c. ~~and~~ all $n \geq n_0$.

10) $10n^2 + 4n + 2 \neq O(n)$

Omega (Ω)

The function $f(n) = \Omega(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all n, $n \geq n_0$.

Ex:

$3n + 2 = \Omega(n)$

$3n + 2 \geq 3n \ \forall \ n \geq 1$

$100n + 6 = \cancel{e}\Omega(n)$ as

$100n + 6 \geq 100n \ \forall \ n \geq 1$.

3) $10n^2+4n+2 = \Omega(n^2)$

$10n^2+4n+2 \geq n^2$ for $n \geq 1$

4) $6 * 2^n + n^2 = \Omega(2^n)$

$6 * 2^n + n^2 \geq 2^n$ for $n \geq 1$.

5) $3n+3 = \Omega(1)$,

$10n^2+4n+2 = \Omega(n)$

$10n^2+4n+2 = \Omega(1)$

$6 * 2^n + n^2 = \Omega(n^{100})$

$6 * 2^n + n^2 = \Omega(n^{50.2})$

$6 * 2^n + n^2 = \Omega(n^2)$

$6 * 2^n + n^2 = \Omega(n)$

$6 * 2^n + n^2 = \Omega(1)$

Theta

The function $f(n) = \Theta(g(n))$ iff there exist positive constants c_1, c_2 and n_0 such that $c_1 \cdot g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.

Ex: $3n+2 = \Theta(n)$

$3n+2 \geq 3n \;\forall\; n \geq 2$

$3n+2 \leq 4n \;\forall\; n \geq 2$ $c_1 = 3$, $c_2 = 4$ and $n_0 = 2$

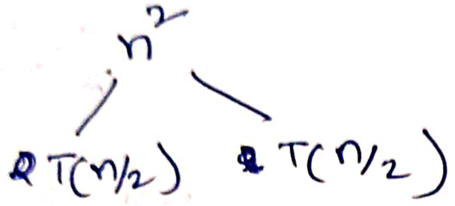
$3n+3 = \Theta(n)$,

$10n^2+4n+2 = \Theta(n^2)$

②

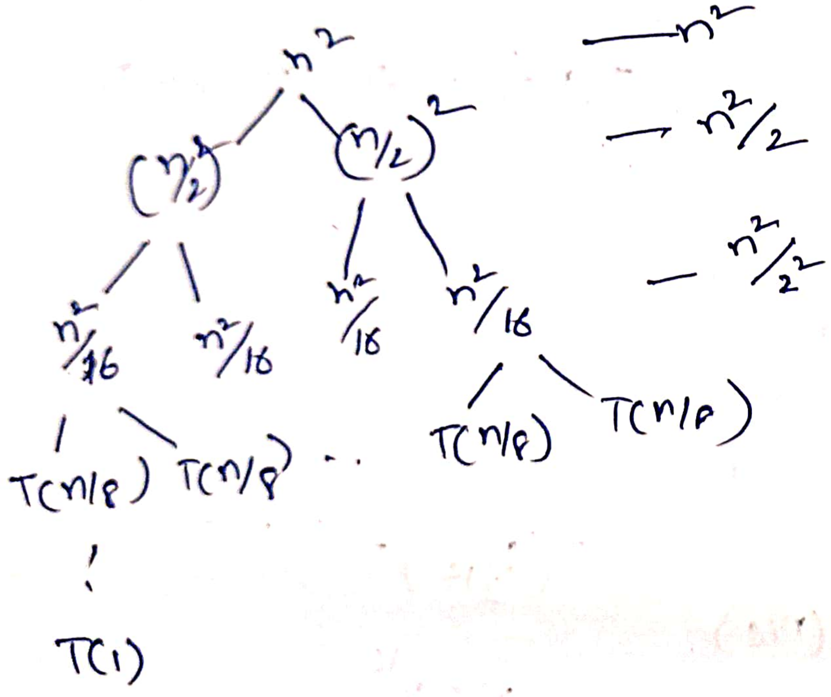
$$T(n) = \begin{cases} 1 & n=1 \\ 2T(n/2) + n^2 & n>1 \end{cases}$$

Sol:



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

$T(n/4) = 2T(n/8) + (n/4)^2$



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

$n^2 = 2^K$

$K = \log^n_2$

$LC = 2^k = n$

$IC = \cancel{K \cdot \frac{n^2}{2^{K-1}}} \quad n^2 + \frac{n^2}{2} + \frac{n^2}{2^2} + \dots \frac{n^2}{2^{K-1}}$

$IC = n^2 \left((\tfrac{1}{2})^0 + (\tfrac{1}{2})^1 + (\tfrac{1}{2})^2 + \dots (\tfrac{1}{2})^{K-1} \right]$

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

$T(n) = O(n^2+n) = \underline{O(n^2)}$

Recursive binary search

```
Algorithm Binsch(a, i, l, x)
{
 if (l = = i) then
 {
   if (x = a[i]) then return i;
   else return 0;
 }
 else
  { mid := (i+l)/2;
   if (x = a[mid]) then return mid;
  }
 else if (x < a[mid]) then
        return Binsrch(a, i, mid-1, x);
    else return Binsrch(a, mid+1, l, x);
  }
```

Successful search

$O(1)$ $\Theta(\log n)$ $\Theta(\log n)$

Best Avg. Worst

unsuccessful search

$O(\log n)$

best, avg & worst.

Analysis for worst case :

Analysis for worst case:

$T(0) = 0$

$T(n) = 1 \qquad x = a(mid)$

$= 1+(T(n+1)/2-1) \qquad x < a(mid)$

$= 1+T(n-(n+1)/2) \qquad x > a(mid)$

$= 1+T(n/2) \qquad x \neq a(mid)$

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

$= 1+1+T(n/4)$

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

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

.

.

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

$= \log_2 n + T(n/2^{\log_2 n})$

$= \log_2 n + T(1)$

$= \log_2 n + 1$

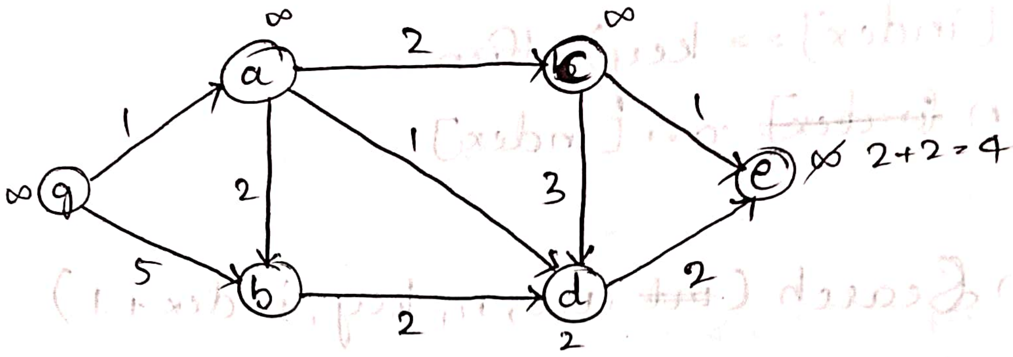
$= O(\log_2 n)$

Best case $= O(1)$

Avg case $= O(\log n)$

Worst case $= O(\log n)$

3
a)



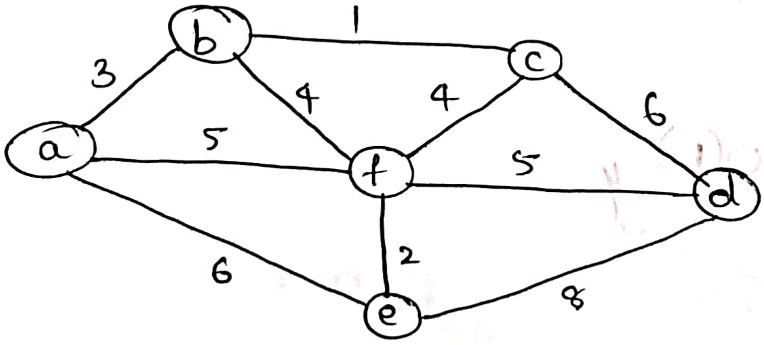
v (vertex)	distance (d[v]	Path.
b	0	-
d	2	b-d
e	4	b-d-e

Since there is no other path from node e or nod
it cannot traverse the entire graph.

Condition:

if (d[i, j] > d[j, ~~k~~] + cost[i, j]) then
d[i,j] = d[j] + cost[i, j]

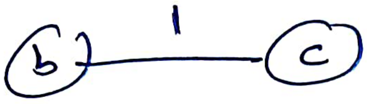
b)



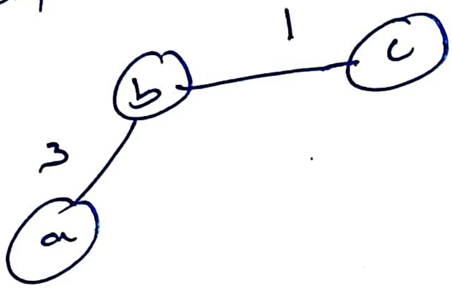
Steps:

1. Start with arbitary node

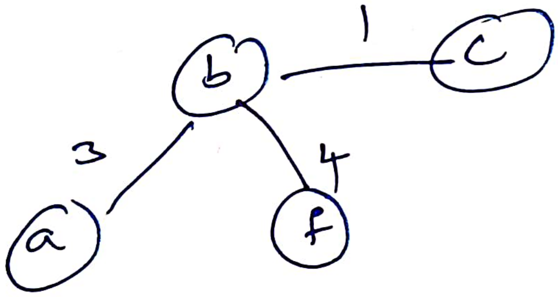
step1: Take edge with min - cost



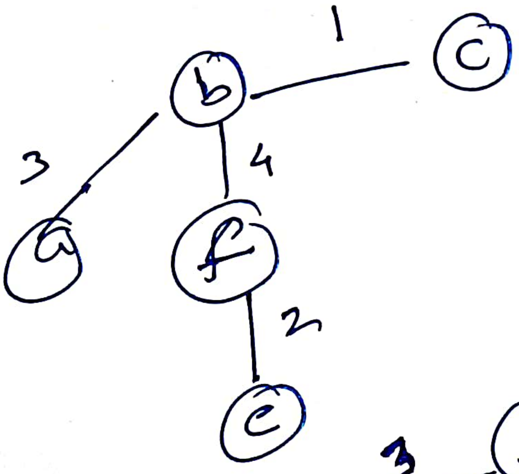
step2: From b & c adjacent min cost edge to be taken



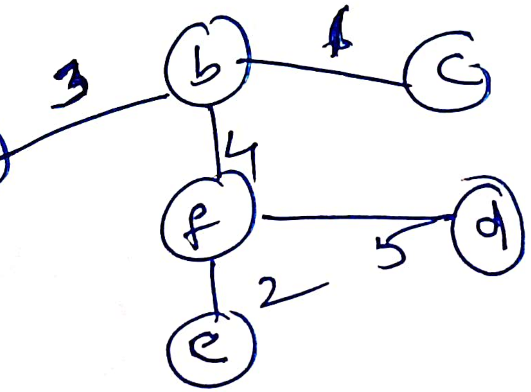
step3: Repeat Above step to get prim's MST until n-1 edges i.e 5 edges



Step 4:



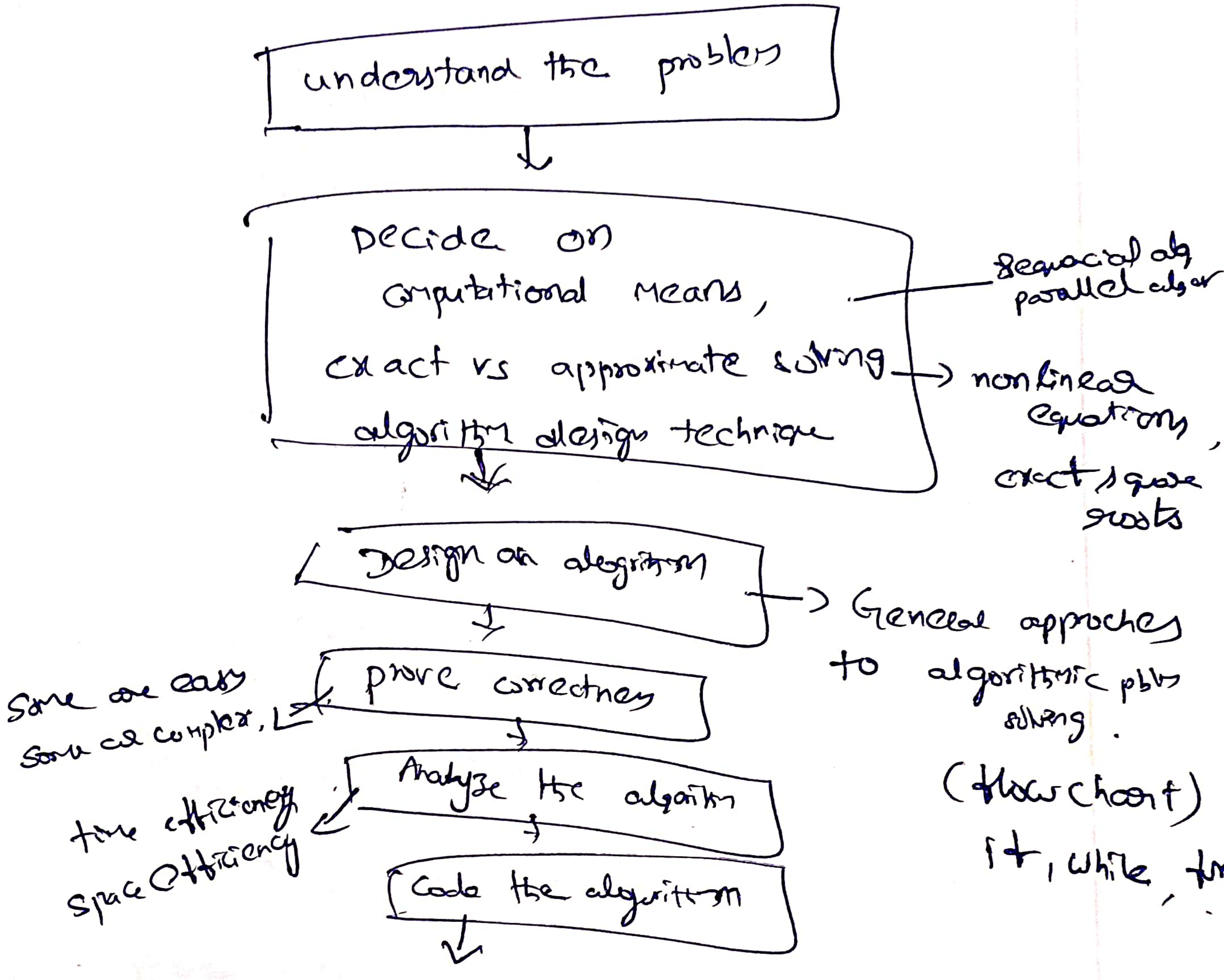
Step5:



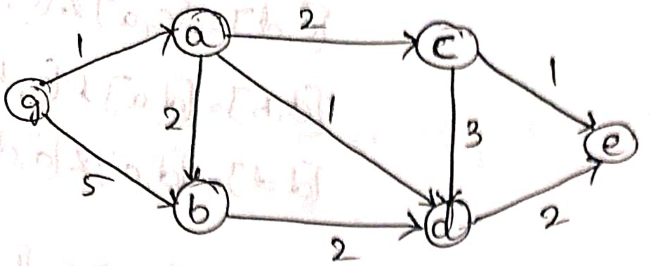
MST

3+1+4+5
+2 = 15

Fundamentals of Algorithmic problem solving



b)



Travelling salesman problem ~~is~~ uses hamiltanion circuit to be found where each node is visited once and return back to the same node.

Since there is no complete path and any path starting from a particular node doesn't reach the origin node, hamiltanion circuit i.e optimal tour can not be found.

Insertion sort

Algorithm:

```
InsertionSort (A[0 . . n-1])

for i = 1 to n-1 do
{  V = A[i]
   j = i - 1
   while  j ≥ 0  and  A[j] > v  do
   {
       A[j+1] = A[j],
   }   j = j-1 ;
   A[j+1] = V ;
}
```

Time complexity

Best case : $\Omega O(n$

Worst case : $O(n^2$

Avg case : $\Theta(n^2)$

for i = 1 to 6   0   1   2   3   4   5   6

Sorted list

unsorted list

25 | 75 40 10 20

25 75 | 40 10 20

25 40 75 | 10 20

10 25 40 75 | 20 1

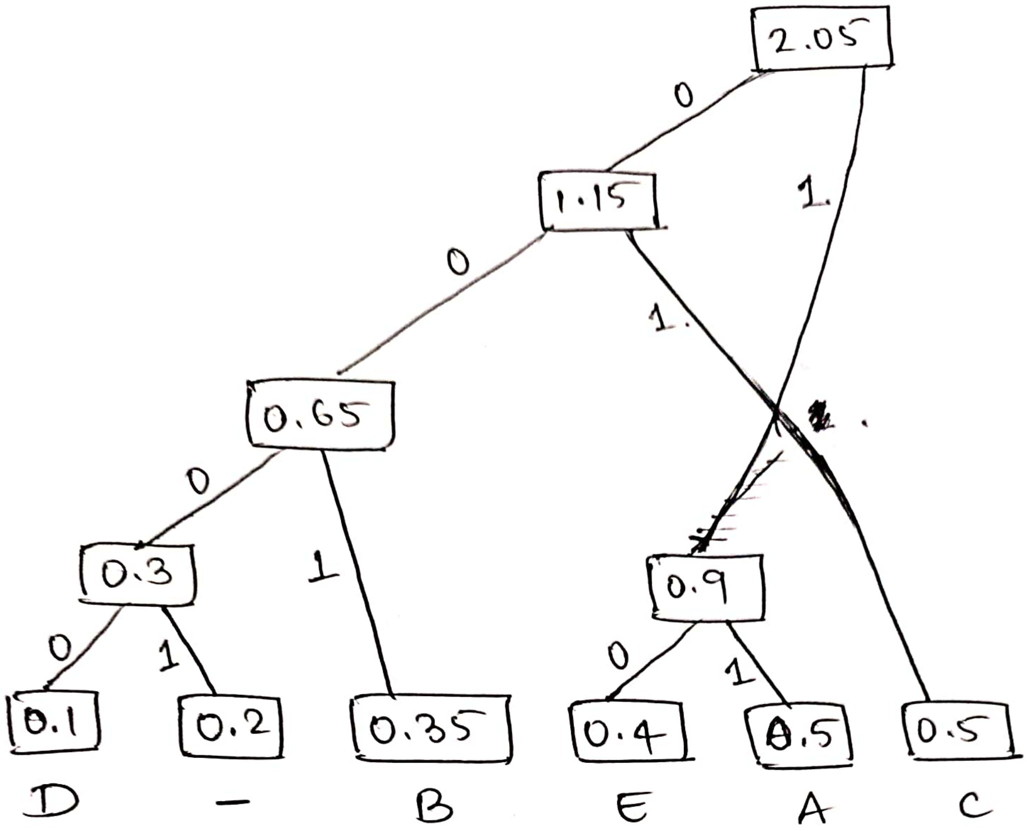
10 20 25 40 75

b) Huffman Trees are the binary trees which is used in Huffman coding which assigns variable length binary code to characters based on their frequencies.

Note! Low frequency variables have longer binary cod

~~Sh~~ High frequency variables have ~~to~~ shorter binary code

char	A	B	C	D	E	_
prob.	0.5	0.35	0.5	0.1	0.4	0.2



Make all the left subtree 0 & right subtree 1.

Encode:

D = 0000

A = 11

D = 0000

- = 0001

C = 01

B = 001

E = 10

DAD - CBE

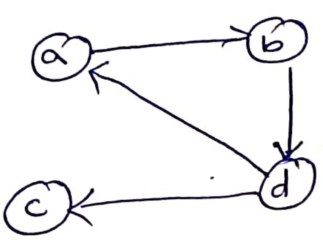
000011000000010100110

Warshall's algorithm using Dynamic programming

-> To find the existence of path b/w all vertices in a given weighted connected graph

-> To determine transitive closure of a directed graph ~~of~~ ~~an~~ in a

directed graph



$$R^0 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 0 & 1 & 0 \end{array}$$

Step 2: Consider path through vertex a

$$R^1 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 0 \end{array}$$

b -> b = b -> a & a -> b
0 & 0 = 0

b -> c = b -> a & a -> c
0 & 0 = 0

c -> b = c -> a & a -> b
0 & 1 = 0

c -> c = c -> a & a -> c
0 & 0
= 0

c -> d = c -> a & c -> d
0 & 0 = 0

d -> b = d -> a & a -> b
1 & 1

d -> d = d -> a & a -> d.
1 & 0

Step 3: Consider path through vertex 'b'

$$R^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 1 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

a -> a = a -> b & b -> a
1 & 0

a -> c = a -> b & b -> c
1 & 0

a -> d = a -> b & b -> d
1 & 1

c -> a = c -> b & b -> a
0

c -> c = c -> b
0

c -> d => c -> b

d -> d = d -> b & b -> d
1 & 1

Step 3: consider path through vertex c

$$P^3 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 0 & 1 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

a → a = a → c & c → a = 0 & 0 = 0

a → C = a → c & c → b = 0 & 0 = 0

b → b = b → c & c → b = 0 & 0 = 0

~~b → a~~

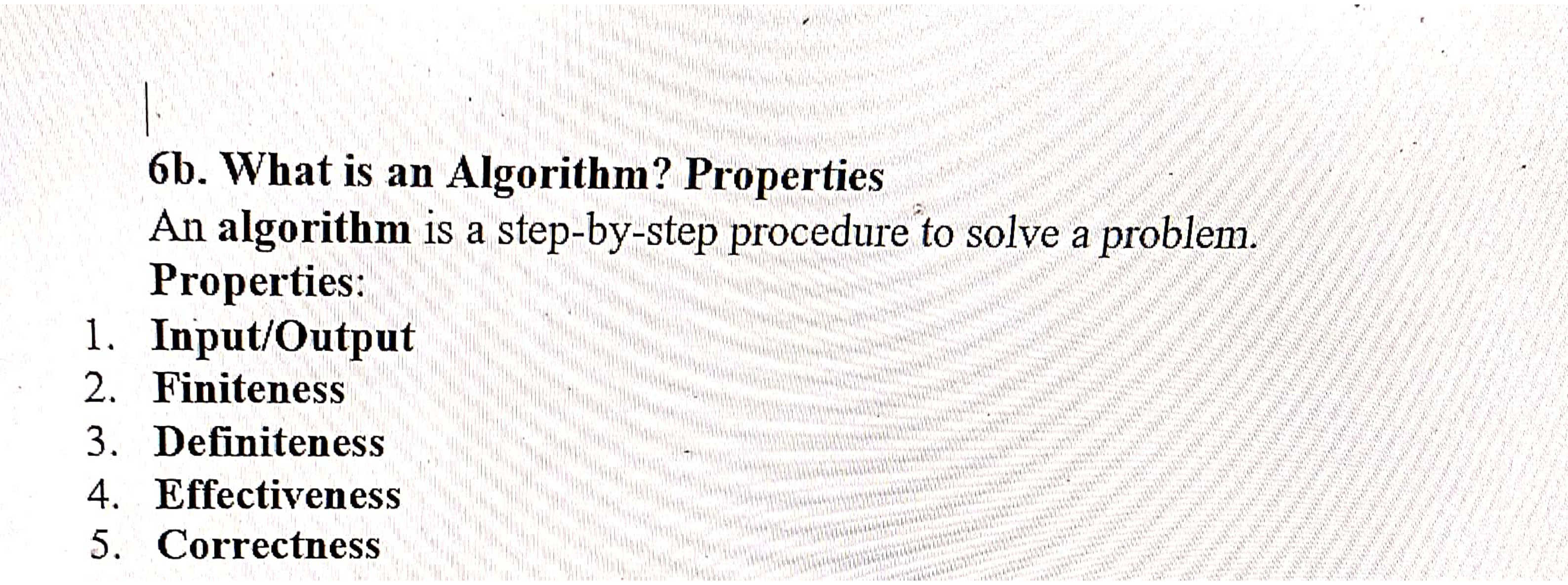
Step 4: consider path through vertex d

$$P^4 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 1 & 1 \\ b & 1 & 1 & 1 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

a → a (=) a → d & d → a = 1 & 1 = 1

Warshall's algorithm

```
for k = 1 to n-1 do
    for i = 0 to n-1 do
    {
        for j = 0 to n-1 do
    {
        if (P[i,j] = 0 & if (P[i,k] = 1 and P[k,j] = 1)) then
            {   P[i,j] = 1
            }
      }
  }
}
```



6b. What is an Algorithm? Properties

An **algorithm** is a step-by-step procedure to solve a problem.

Properties:

1. **Input/Output**
2. **Finiteness**
3. **Definiteness**
4. **Effectiveness**
5. **Correctness**