APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

B.Tech Degree S6 (R, S) / S6 (PT) (R) Examination June 2023 (2019 Scheme)

Course Code: CST306

Course Name: ALGORITHM ANALYSIS AND DESIGN

PART A

1. Show that for any real constants a and b, where b > 0, $(n + a)^b = O(n^b)$

Ans:

We need to find c and n_0 such that $(n + a)^b \leq c\, n^b$, for all $n >= n_0$

$(n + a) \leq (n + |a|)$

$(n + a)^b \leq (n + |a|)^b$

$(n + |a|)^b = n^b (1 + \frac{|a|}{n})^b$

$(1 + \frac{|a|}{n})^b \leq c$

Assume that $c=2^b$ and $n_0=\max(1,|a|)$

Now the following relation is true.

$(n + a)^b \leq 2^b n^b$, for all $n >= \max(1,|a|)$

Therefore, $(n + a)^b = O(n^b)$

2. Solve the following recurrence equations using Master theorem

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

b. $T(n) = 2T(n/2) + n \log n$

Ans:

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

a=3 b=2 $n^2 = \Theta(n^2 \log^0(n))$ k=2 p=0

$b^k = 2^2 = 4$

Here $a < b^k$ and $p \geq 0$

$T(n) = \Theta(n^k \log^p(n))$

$= \Theta(n^2 \log^0(n))$

$= \mathbf{\Theta(n^2)}$

b) $T(n) = 2T(n/2) + n \log n$

a=2 b=2 $n \log n = \Theta(n^1 \log^1(n))$ k=1 p=1

$b^k = 2^1 = 2$

Here $a = b^k$ and $p > -1$

$T(n) = \Theta(n^{(\log_b a)} \log^{p+1}(n))$

$= \Theta(n^{(\log_2 2)} \log^2(n))$

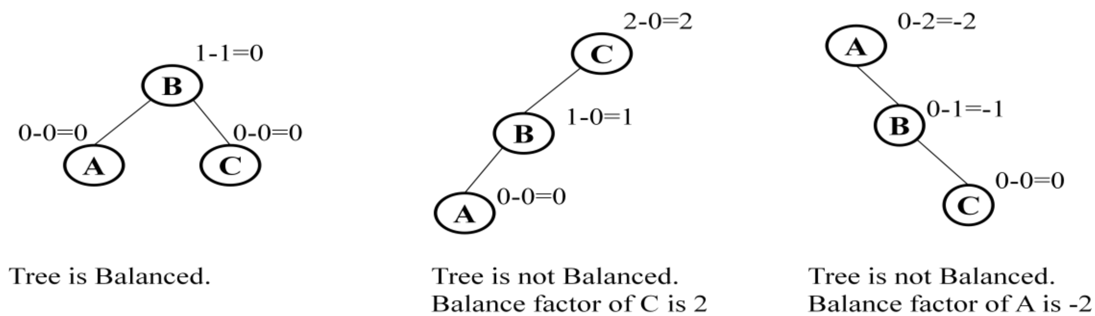
$= \mathbf{\Theta(n\log^2(n))}$

3. Define AVL tree. Explain the rotations performed for insertion in AVL tree.

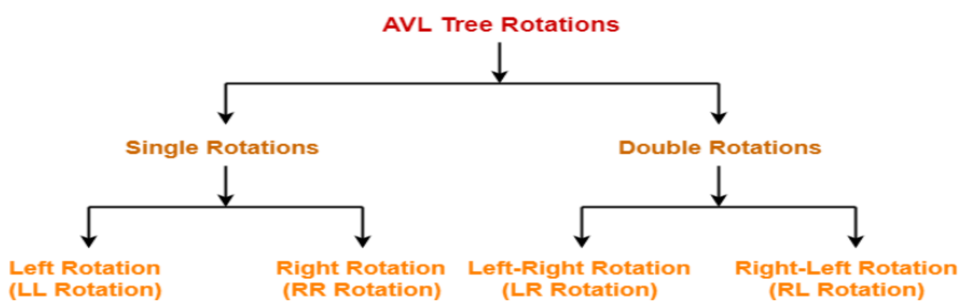
Ans:

- AVL Tree can be defined as **height balanced binary search tree** in which each node is associated with a balance factor.

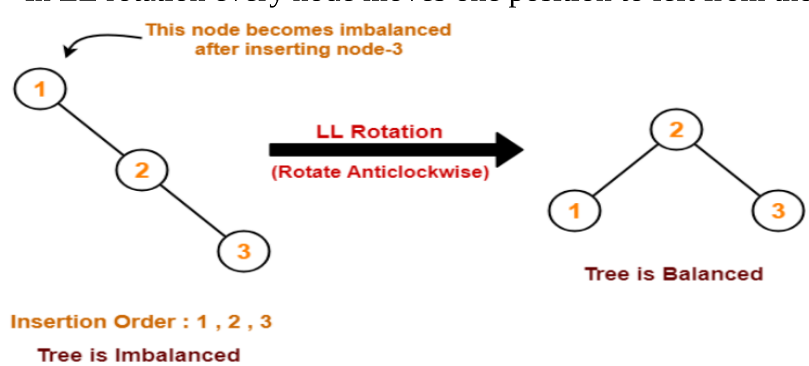
- **Balance Factor**
  - Balance Factor of a node = height of left subtree – height of right subtree
  - In an AVL tree balance factor of every node is -1,0 or +1
  - Otherwise the tree will be unbalanced and need to be balanced.



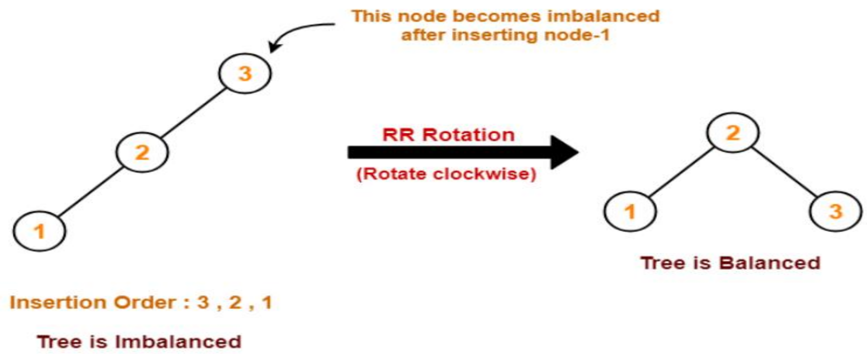
- An AVL tree becomes imbalanced due to some insertion or deletion operations
- We use rotation operation to make the tree balanced.
- There are 4 types of rotations



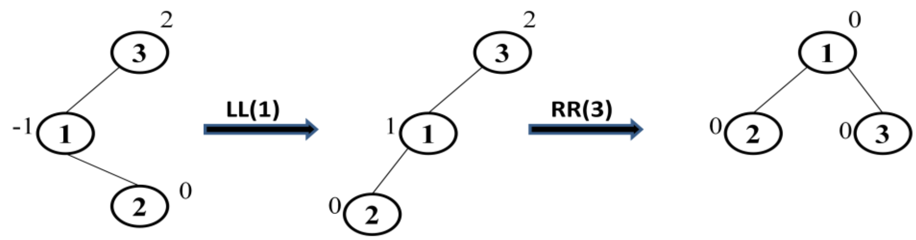
  - **Single Left Rotation(LL Rotation)**
    - In LL rotation every node moves one position to left from the current position



  - **Single Right Rotation(RR Rotation)**
    - In RR rotation every node moves one position to right from the current position

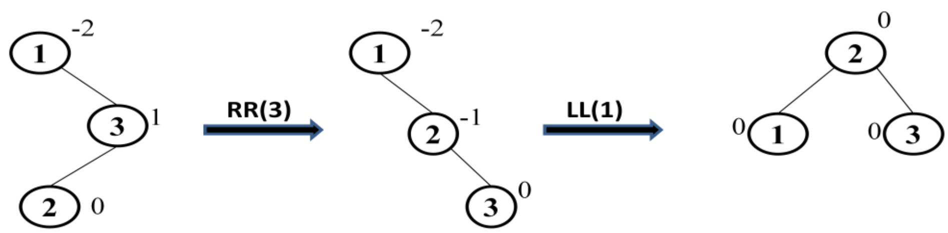


- **Left-Right Rotation(LR Rotation)**
  - The LR rotation is the combination of single left rotation followed by single right rotation.



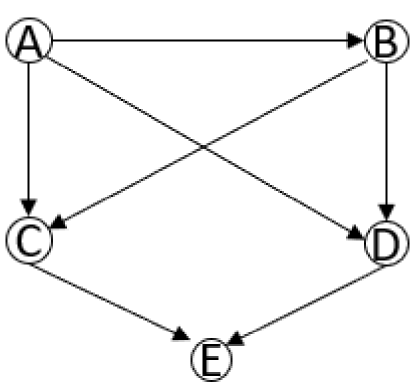
Insertion Order: 3,1,2

- **Right-Left Rotation(RL Rotation)**
  - The RL rotation is the combination of single right rotation followed by single left rotation.



Insertion Order: 1,3,2

4. Find the different topological ordering of the given graph



Ans:

A, B, C, D, E

A, B, D, C, E

5. Write the control abstraction of divide and conquer strategy

Ans:

```
Algorithm DAndC(P)
{
        if Small(P) then
                return S(P)
        else
        {
                Divide P into smaller instances P1, P2, . . . . Pk,  k≥1;
                apply DAndC to each of these sub-problems;
                return Combine(DAndC(P1), DAndC(P2), . . . . , DAndC(Pk));
        }
}
```

6. Compare Strassen’s matrix multiplication with ordinary matrix multiplication

Ans:

- **Ordinary Matrix Multiplication**
  - For multiplying two matrices of size n x n, we make n^3 matrix multiplications.
  - So the time complexity = $\mathbf{O(n^3)}$
- **Strassen’s matrix multiplication**
  - For multiplying two matrices of size n x n, we make 7 matrix multiplications and 10 matrix additions and subtractions
  - Addition/Subtraction of two matrices takes $O(n^2)$ time.
  - Time complexity = $7\,T(n/2) + O(n^2) = O(n^{\log 7}) = \mathbf{O(n^{2.81})}$

7. Differentiate backtracking technique from branch and bound technique

Ans:

Backtracking	**Branch and Bound**
Backtracking is a problem-solving technique so it solves the decision problem.	Branch n bound is a problem-solving technique so it solves the optimization problem.
Backtracking uses a Depth first search.	Branch and bound uses Depth first search/D Search/Least cost search.
In backtracking, all the possible solutions are tried. If the solution does not satisfy the constraint, then we backtrack and look for another solution.	In branch and bound, based on search; bounding values are calculated. According to the bounding values, we either stop there or extend.
Applications of backtracking are n-Queens problem, Sum of subset.	Applications of branch and bound are knapsack problem, travelling salesman problem, etc.
Backtracking is more efficient than the Branch and bound.	Branch n bound is less efficient.

8. What is Principle of Optimality?

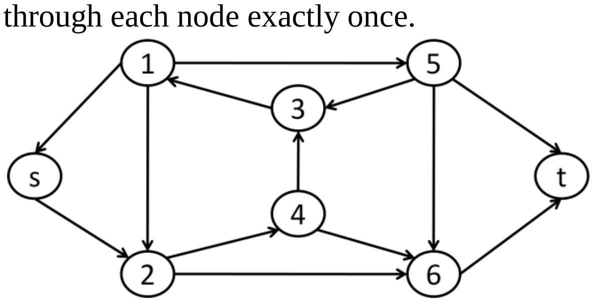
Ans:

- **Definition**: The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.
- A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themesleves optimal solutions for their subproblems.
- Examples:
  - The shortest path problem satisfies the Principle of Optimality.
  - This is because if $a,x_1,x_2,...,x_n,b$ is a shortest path from node a to node b in a graph, then the portion of x_i to x_j on that path is a shortest path from x_i to x_j.

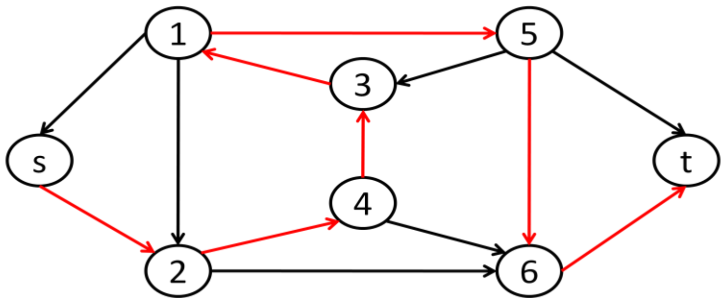
9. Differentiate P and NP problems. Give one example to each

Ans:

- **Class P**
  - Class P consists of those problems that are solvable in polynomial time.
  - P problems can be solved in time $O(n^k)$. Here n is the size of input and k is some constant.
  - Example:
    - **PATH Problem:** Given directed graph G, determine whether a directed path exists from s to t.
      - Complexity of this algorithm = O(n).
      - This is a polynomial time algorithm.
- **Class NP**
  - Some problems can be solved in exponential or factorial time. Suppose these problems have no polynomial time solution. We can verify these problems in polynomial time. These are called NP problems.
  - NP is a class of problem that having only non-polynomial time algorithm and a polynomial time verifier.
  - Example:
    - **Hamiltonian path(HAMPATH) Problem**
      - A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.



      - The Hamiltonian path of the above graph is as follows

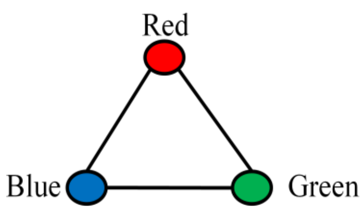


- There is no polynomial solution to find the Hamiltonian path from s to t in a given graph. But we can verify these problems in polynomial time. So **HAMPATH** problem is a NP problem.

10. Define graph coloring problem

Ans:

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 number of colors while coloring a graph. The smallest number of colors required to color a graph is called its chromatic number of that graph. Graph coloring problem is a NP Complete problem.



3 Colorable graph

PART B

11.

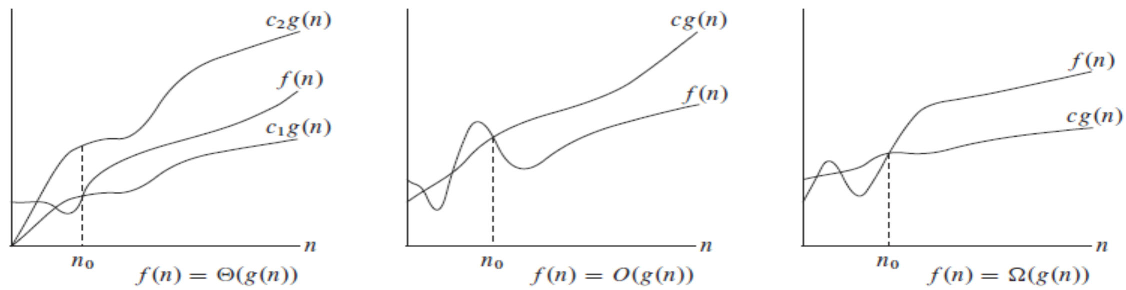
a) Define Big Oh, Big Omega and Theta notations and illustrate them graphically.

Ans:

- **Asymptotic Notations**
    - It is the mathematical notations to represent frequency count.
    - **Big Oh (O)**
        - The function f(n) = O(g(n)) iff there exists 2 positive constants c and n_0 such that $0 \leq f(n) \leq c\ g(n)$ for all $n \geq n_0$
        - It is the measure of longest amount of time taken by an algorithm(Worst case).
        - It is asymptotically tight upper bound
    - **Omega (Ω)**
        - The function f(n) = Ω (g(n)) iff there exists 2 positive constant c and n_0 such that $f(n) \geq c\ g(n) \geq 0$ for all $n \geq n_0$
        - It is the measure of smallest amount of time taken by an algorithm(Best case).

- It is asymptotically tight lower bound

- **Theta (Θ)**
  - The function $f(n) = \Theta(g(n))$ iff there exists 3 positive constants c_1, c_2 and n_0 such that $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ for all $n \geq n_0$
  - It is the measure of average amount of time taken by an algorithm(Average case).



b) Find the time complexity of following code segment

i)

```
for (int i = 1; i <= n; i *= c) {
    // some O(1) expressions
}
for (int i = n; i > 0; i /= c) {
    // some O(1) expressions
 }
```

(ii)

```
for (int i = 1; i <= n; i += c) {
    // some O(1) expressions
}
for (int i = n; i > 0; i -= c) {
    // some O(1) expressions
}
```

Ans:

i)

Time complexity of First for loop = $O(\log_c n)$
Time complexity of Sevond for loop = $O(\log_c n)$
Altogether the time complexity = $\mathbf{O(\log_c n)}$

ii)

Frequency count of first for loop = n/c
Frequency count of second for loop = n/c
Total frequency count = 2n/c
Time complexity = **O(n)**

12.

a) Find the best case, worst case and average case time complexity of binary search

Ans:

```
Algorithm BinarySearch(A, low, high, search_data)
{
    flag=0
    while low<=high do
    {
            mid= (low+high)/2
            if A[mid]=search_data then
            {
                    flag = 1
                            break
            }
            else if A[mid] >search_data then
                    high=mid-1
            else
                    low=mid+1
    }
    if flag=0 then
            Print "Search data not found"
    else
            Print "Search data found at index "mid
}
```

■ Best Case Time Complexity of Binary Search

- The search data is at the middle index.
- So total number of iterations required is 1
- Therefore, Time complexity = O(1)

■ Worst Case Time Complexity of Binary Search

- Assume that length of the array is n
- At each iteration, the array is divided by half.
- At Iteration 1, Length of array = n
- At Iteration 2, Length of array = n/2
- At Iteration 3, Length of array = $(n/2)/2 = n/2^2$
- At Iteration k, Length of array $=n/2^{k-1}$
- After k divisions, the length of array becomes 1

  $n/2^{k-1} = 1$

  $n = 2^{k-1}$
- Applying log function on both sides:

  $\log_2 (n) = \log_2 (2^{k-1})$

  $\log_2 (n) = (k-1) \log_2 (2)$

  $k = \log_2 (n) +1$
- Hence, the time complexity = $O(\log_2 (n))$

■ Average Case Time Complexity of Binary Search

- Total number of iterations required = $k/2 = \log_2(n)/2$
- Hence, the time complexity = $O(\log_2(n))$

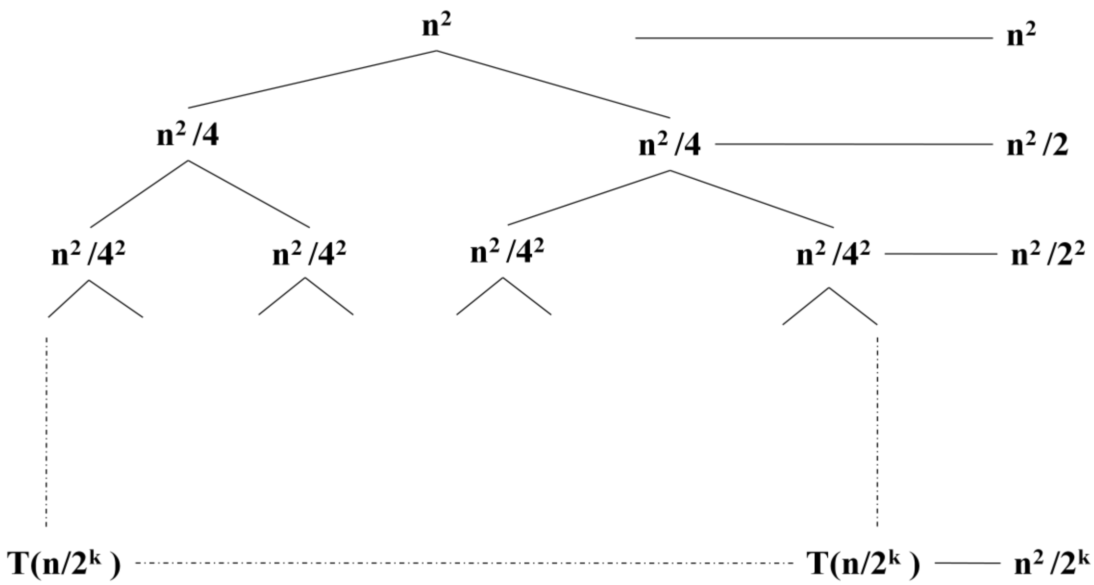
b) Find the time complexity of following function using recursion tree method

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

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

Ans:

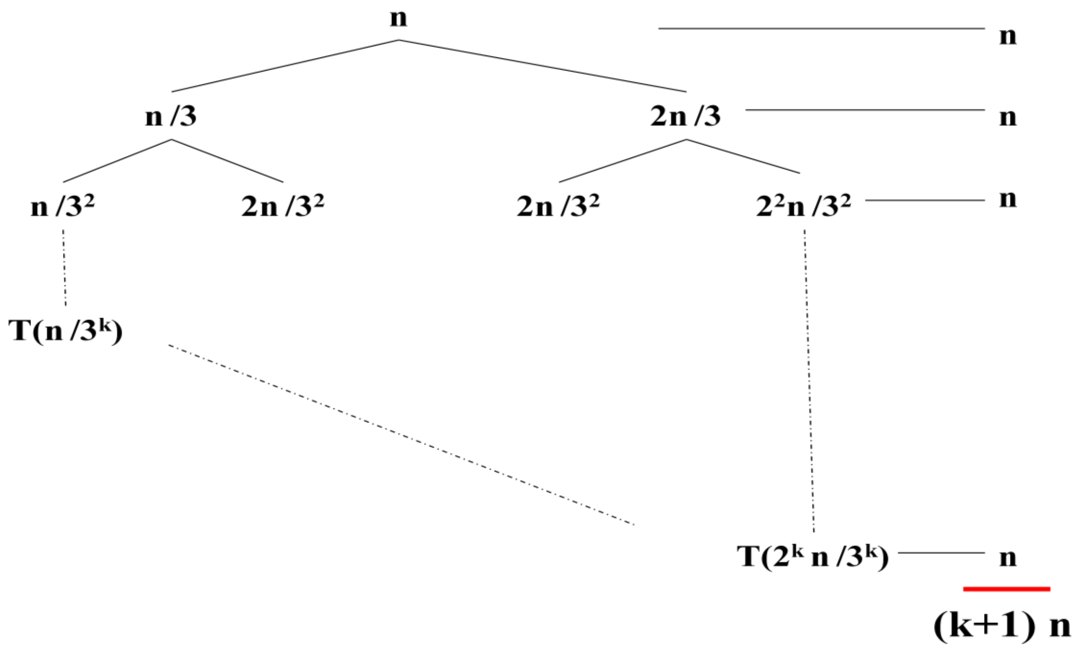
(i) $T(n) = 2\,T(n/2) + n^2$



Assume $n/2^k=1$ ➔ $2^k=n$ ➔ $k=\log_2 n$

$$
\begin{aligned}
T(n) &= n^2+(n^2/2)+(n^2/2^2)+\dots\dots+(n^2/2^k) \\
&= n^2[1+(1/2)+(1/2)^2+\dots\dots+(1/2)^k] \\
&= n^2[\,[1-(1/2)^{k+1}]\,/\,[1-(1/2)]\,] \\
&= 2n^2[\,1-(1/2)\text{x}(1/2)^k] \\
&= 2n^2[\,1-(1/2)\text{x}(1/2^k)] \\
&= 2n^2[\,1-(1/2)\text{x}(1/n)] \\
&= 2n^2 - n \\
&= \mathbf{O(n^2)}
\end{aligned}
$$

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



Assume that $2^k n/3^k = 1$ ➔ $(3/2)^k = n$ ➔ $k = \log_{(3/2)} n$

$T(n) = (k+1)\,n$

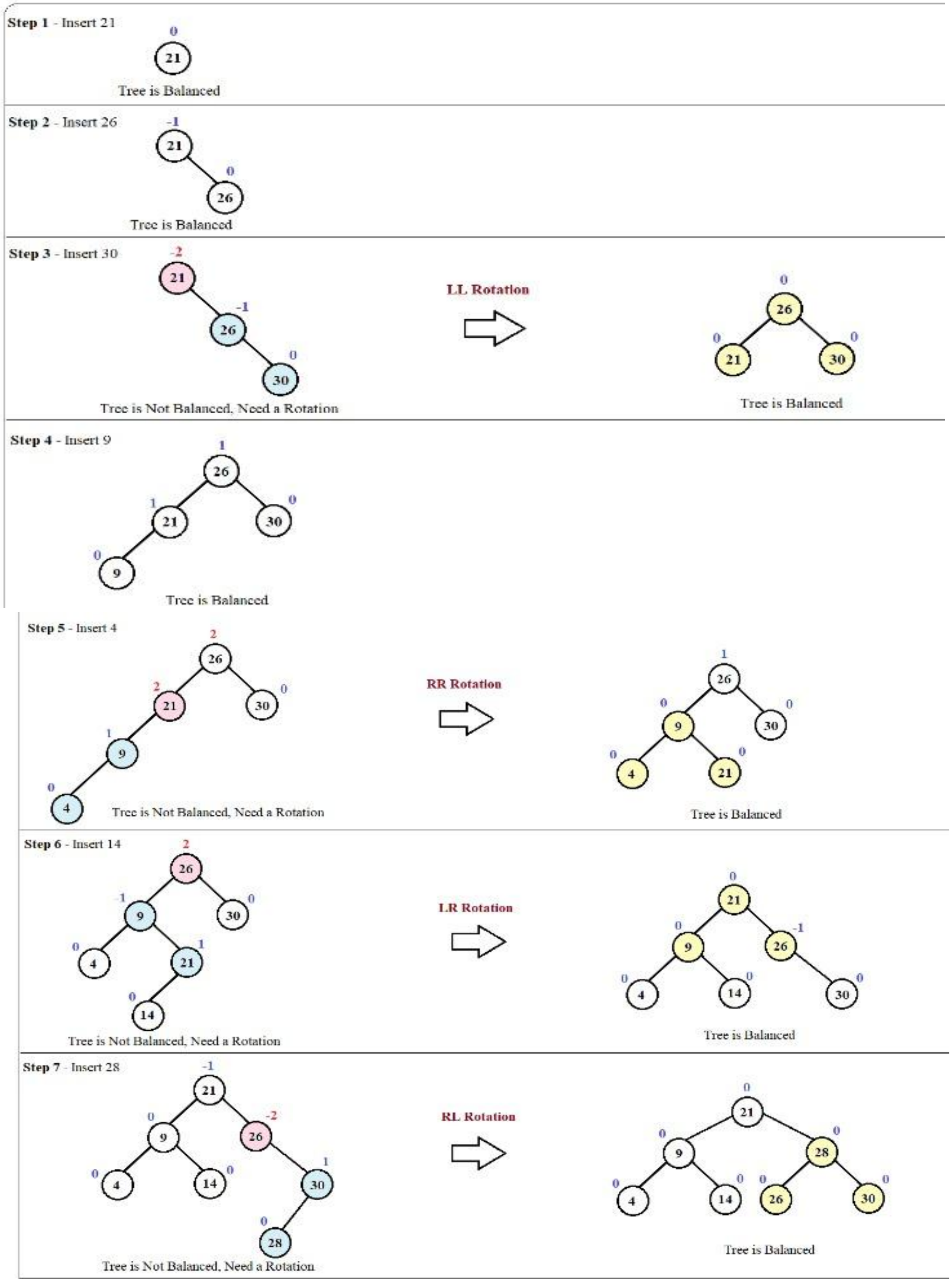
$= (\log_{(3/2)} n + 1)\,n$

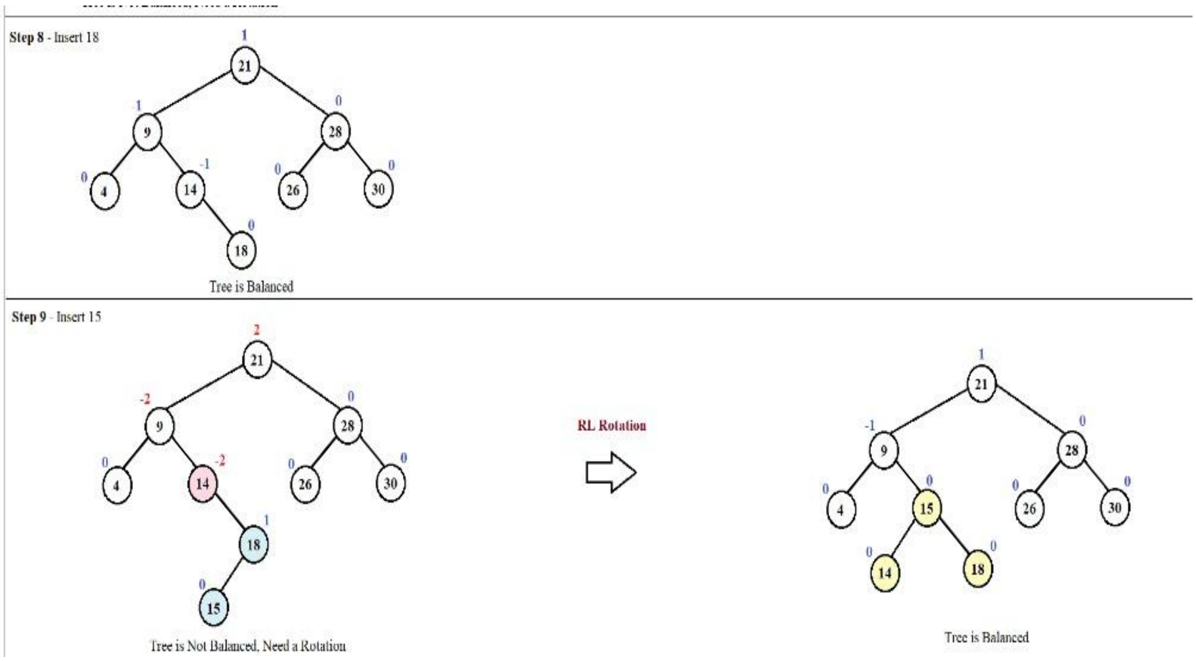
$= n \log_{(3/2)} n + n$

$= \mathbf{O(n \log_{(3/2)} n)}$

13.

a) Construct AVL tree by inserting following elements appeared in the order.
21, 26, 30, 9, 4, 14, 28, 18,15
Ans:

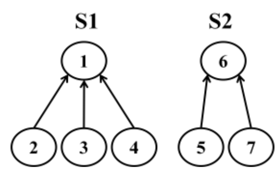

Step 1 - Insert 21
Tree is Balanced
Step 2 - Insert 26
Tree is Balanced
Step 3 - Insert 30
Tree is Not Balanced, Need a Rotation
LL Rotation
Tree is Balanced
Step 4 - Insert 9
Tree is Balanced
Step 5 - Insert 4
Tree is Not Balanced, Need a Rotation
RR Rotation
Tree is Balanced
Step 6 - Insert 14
Tree is Not Balanced, Need a Rotation
LR Rotation
Tree is Balanced
Step 7 - Insert 28
Tree is Not Balanced, Need a Rotation
RL Rotation
Tree is Balanced



b) Explain union and find algorithms in disjoint datasets

Ans:

- **Find Operation**
  - Determine which subset a particular element is in.
  - This will return the representative(root) of the set that the element belongs.
  - This can be used for determining if two elements are in the same subset.



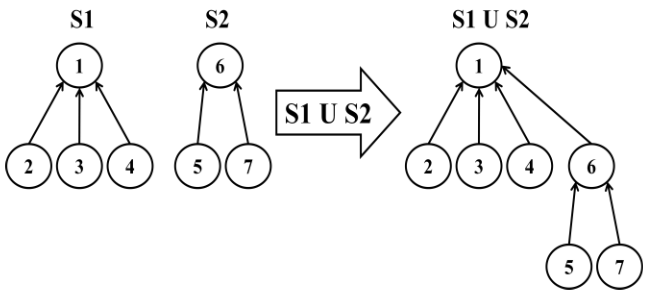
Find(3) will return 1, which is the root of the tree that 3 belongs
Find(6) will return 6, which is the root of the tree that 6 belongs

- **Find Algorithm**

  Algorithm Find(n)
  1. while n→parent != NULL do
     1.1 n = n→parent
  2. return n

  Worst case Time Complexity = O(d), where d is the depth of the tree

- **Union Operation**
  - Join two subsets into a single subset.
  - Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union



i	1	2	3	4	5	6	7
P	-1	1	1	1	6	1	6

- **Union Algorithm**

  Algorithm Union(a, b)
  1. X =Find(a)
  2. Y = Find(b)
  3. If X != Y then
     1. Y→parent = X

  Worst case Time Complexity = O(d), where d is the depth of the tree

14.

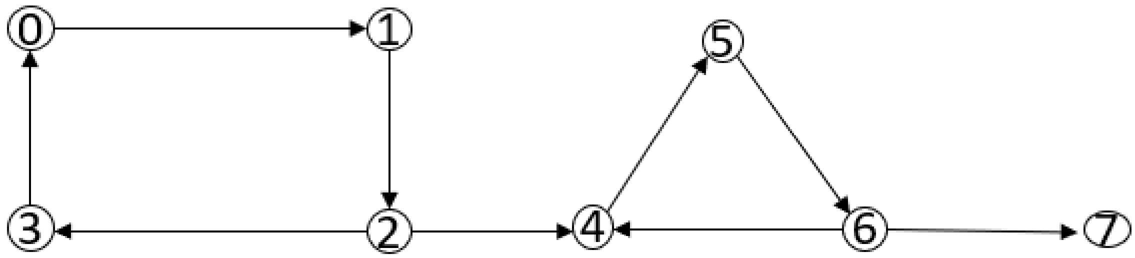
a) Write DFS algorithm for graph traversal. Also derive its time complexity.
Ans:

Algorithm DFS(G, u)
1. Mark vertex u as visited
2. For each adjacent vertex v of u
   2.1 if v is not visited
   2.1.1 DFS(G, v)

Algorithm main(G,u)
1. Set all nodes are unvisited.
2. DFS(G, u)
3. For any node x which is not yet visited
   3.1 DFS(G, x)

- **Complexity**
- If the graph is represented as an adjacency list
  - Each vertex is visited atmost once. So the time devoted is **O(V)**
  - Each adjacency list is scanned atmost once. So the time devoted is **O(E)**
  - Time complexity of DFS = **O(V + E)**.
- If the graph is represented as an adjacency matrix
  - There are V^2 entries in the adjacency matrix. Each entry is checked once.
  - Time complexity of DFS = $\mathbf{O(V^2)}$

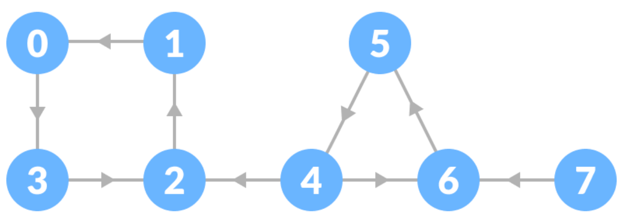
b) Find the strongly connected components of the given directed graph.



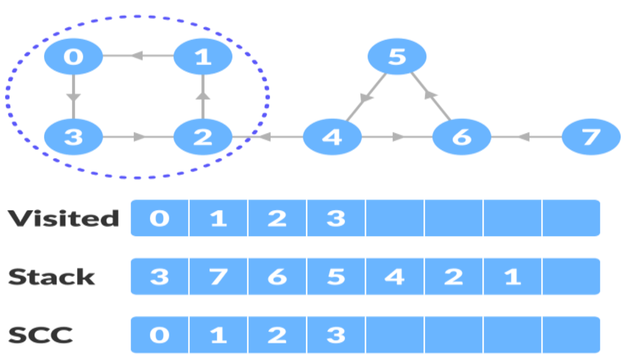
Ans:

PASS-1
Perform a depth first search on the whole graph. If a vertex has no unvisited neighbor, then push this vertex to the stack.
Final stack will look like:

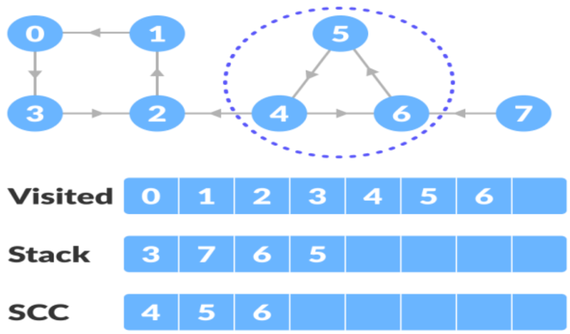
PASS-2
Now reverse the original graph.



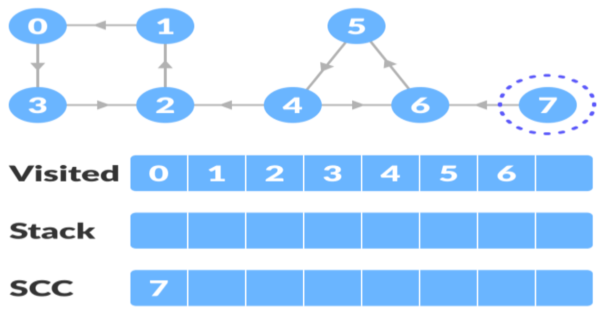
Set all nodes are unvisited.
Pop an item from the stack. If it is unvisited and perform DFS on the reversed graph. It will generate the first strongly connected component.



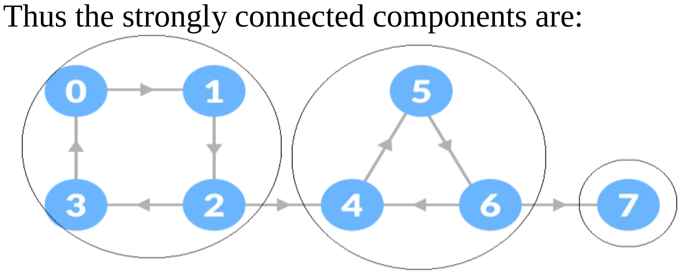
Again pop the next item from the stack and if it is unvisited perform DFS. It will generate the next strongly connected component.



Again pop the next item from the stack and if it is unvisited perform DFS. It will generate the next strongly connected component.



Thus the strongly connected components are:



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

15.

a) Explain 2- way merge sort algorithm with an example and derive its time complexity
Ans:

```
Algorithm MergeSort(low, high)
{
        mid = (low + high )/2;
        MergeSort(low, mid);
        MergeSort(mid+1, high);
        Merge(low, mid, high);
}
Algorithm Merge(low, mid, high)
{
```

```
        i= low; x= low;  y= mid + 1;
        while((x ≤ mid) and (y ≤ high)) do
        {
                if ( a[x] ≤ a[y] ) then
                {
                        b[i] = a[x];
                        x = x+1;
                }
                else
                {
                        b[i] = a[y];
                        y = y+1;
                }
                i=i+1;
        }
        if( x ≤ mid) then
        {
                for k=x to mid do
                {
                        b[i] = a[k];
                        i =i+1;
                }
        }
        else
        {
                for k=y to high do
                {
                        b[i] = a[k];
                        i =i+1;
                }
        }
        for k= low to high do
                a[k] = b[k];
}
```

- **Complexity**

$$\mathbf{T(n)} = \begin{cases} \mathbf{a} & \textbf{if n=1} \\ \mathbf{2\ T(n/2) + cn} & \textbf{Otherwise} \end{cases}$$

a is the time to sort an array of size 1
cn is the time to merge two sub-arrays
2 T(n/2) is the complexity of two recursion calls

$$\begin{aligned} T(n) &= 2\ T(n/2) + c\ n \\ &= 2(2\ T(n/4)+c(n/2)) + c\ n \end{aligned}$$

$= 2^2T(n/2^2) + 2\ c\ n$
$= 2^3T(n/2^3) + 3\ c\ n$
. . . .
$= 2^kT(n/2^k) + k\ c\ n$ [Assume that $2^k = n$, $k = \log n$]
$= n\ T(1) + c\ n \log n$
$= a\ n + c\ n \log n$
$=$ **O(n log n)**

Best Case, Average Case and Worst Case Complexity of Merge Sort **= O(n log n)**

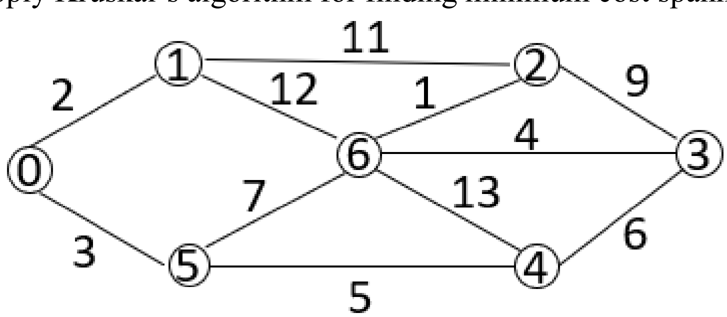
b) Find the optimal solution for the following Fractional Knapsack problem. Given the number of items(n) = 7, capacity of sack(m) = 15,
W={1, 3, 5, 4, 1, 3, 2} and P = {10, 15, 7, 8, 9, 4}

Ans:

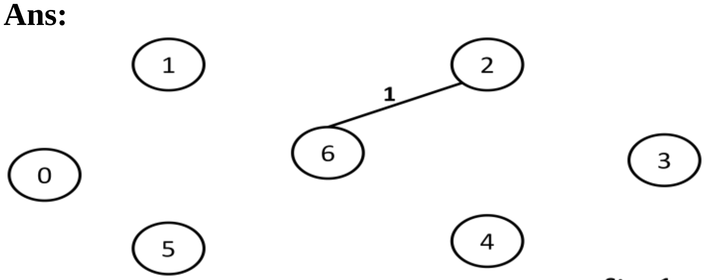
There are 7 weights and 6 profits. So we cannot solve this problem.

16.

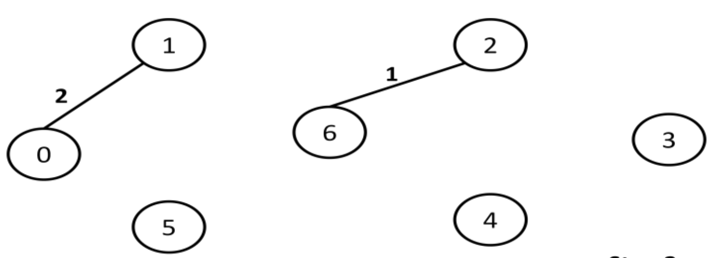
a) Apply Kruskal's algorithm for finding minimum cost spanning tree



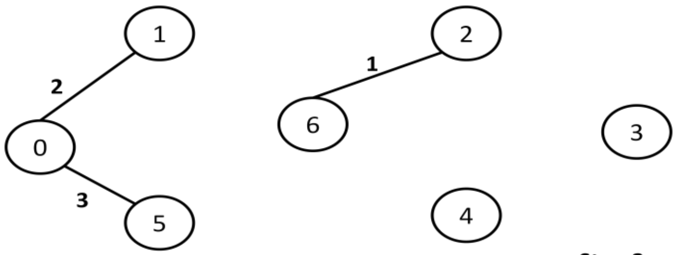
Ans:



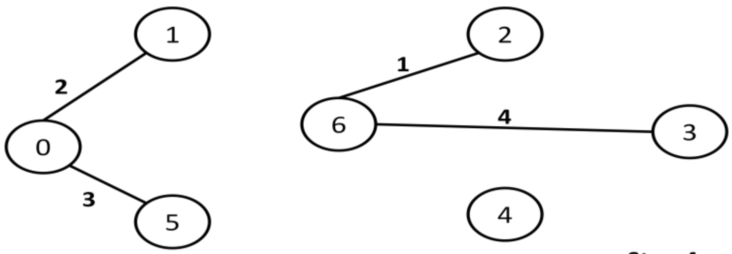
Step 1



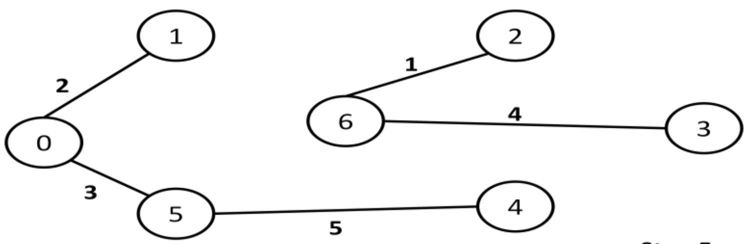
Step 2



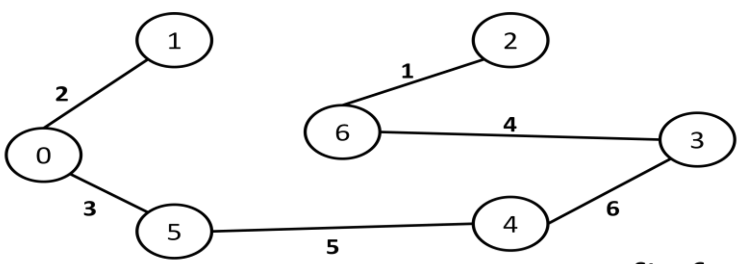
Step 3



Step 4



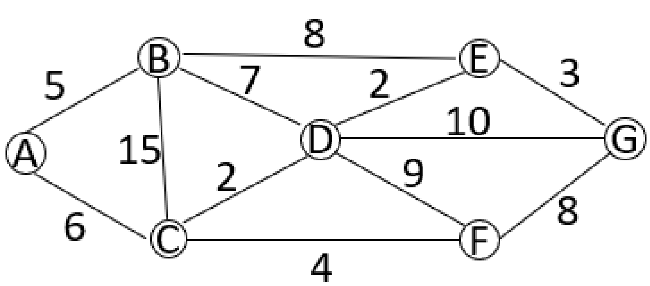
Step 5



Step 6

This is minimum cost spanning tree and its cost = 21

b) Apply Dijikstra's algorithm for finding the shortest path from vertex A to all other vertices.



Ans:

	A	B	C	D	E	F	G
A	0	∞	∞	∞	∞	∞	∞
B		5	∞	∞	∞	∞	∞
C			4	12	13	∞	∞
D			6	6	13	10	∞
F					10	10	18
E					10		13
G							13
				C	D	C	E

PATH	SHORTESTPATH	SHORTESTDISTANCE
A→B	A→B	5
A→C	A→C	6
A→D	A→C→D	8
A→E	A→C→D→E	10
A→F	A→C→F	10
A→G	A→C→D→E→G	13

17.

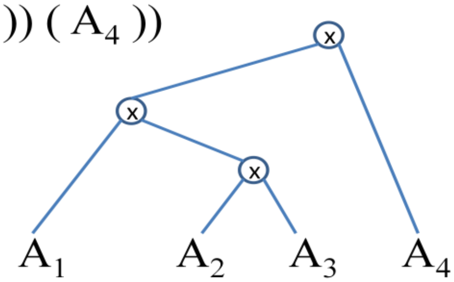
a) Find the optimal parenthesis of matrix chain product whose sequence of dimensions is 5 x 4, 4 x 6, 6 x 2, 2 x 7

Ans:

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

s	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

$((A_1 (A_2 \; A_3)) (A_4))$

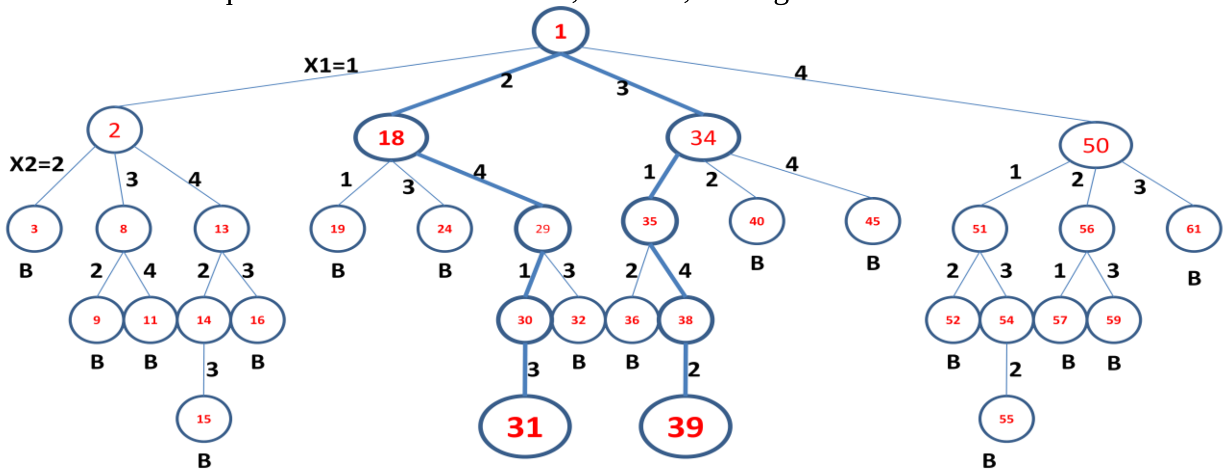


Minimum number of scalar multiplications = **158**

b) Explain 4 queen problem. Draw the state space tree for 4 queen problem.

Ans:

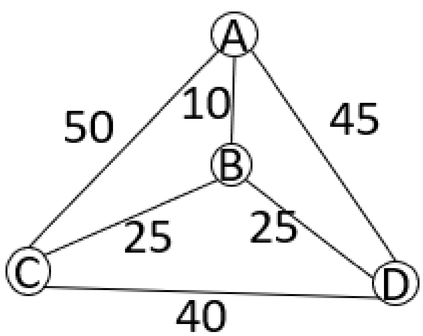
- **4-Queens Problem**
  - 4 queens are to be placed on a 4 x 4 chessboard so that no two attack. That is, no two queens are on the same row, column, or diagonal.



- **State Space Tree of 4 Queens Problem**

18.

a) Define TSP problem. Apply branch and bound algorithm for solving TSP.



Ans:

Adjacency Matrix :
$$\begin{pmatrix} \infty & 10 & 50 & 45 \\ 10 & \infty & 25 & 25 \\ 50 & 25 & \infty & 40 \\ 45 & 25 & 40 & \infty \end{pmatrix}$$

Perform row reduction and column reduction

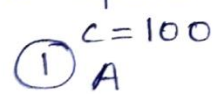
$$\begin{array}{c c}
\begin{bmatrix} \alpha & 10 & 50 & 45 \\ 10 & \alpha & 25 & 25 \\ 50 & 25 & \alpha & 40 \\ 45 & 25 & 40 & \alpha \end{bmatrix} & \begin{array}{c} -10 \\ -10 \\ -25 \\ -25 \end{array}
\end{array}
\xrightarrow[\text{reduction}]{\text{row}}
\begin{array}{c}
\begin{bmatrix} \alpha & 0 & 40 & 35 \\ 0 & \alpha & 15 & 15 \\ 25 & 0 & \alpha & 25 \\ 20 & 0 & 15 & \alpha \end{bmatrix} \\
\begin{array}{cccc} 0 & 0 & -15 & -15 \end{array}
\end{array}$$

$$\xrightarrow[\text{reduction}]{\text{column}}
\begin{bmatrix} \alpha & 0 & 25 & 20 \\ 0 & \alpha & 0 & 0 \\ 25 & 0 & \alpha & 10 \\ 20 & 0 & 0 & \alpha \end{bmatrix}$$

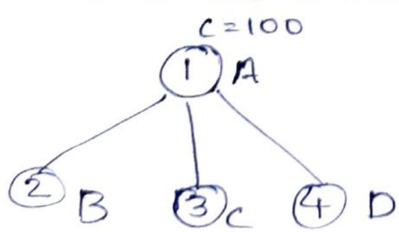
M_1

Total cost reduced $= 10+10+25+25+15+15 = 100$

The state space tree is



Generate the children of node 1



Find the matrix and cost of node 2

* Set row A and column B elmts are α
* Set $M_1[B,A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & 0 \\ 25 & \alpha & \alpha & 0 \\ 20 & \alpha & 0 & \alpha \end{bmatrix}$$

* Perform row reduction & column reduction

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & 0 \\ 25 & \alpha & \alpha & 0 \\ 20 & \alpha & 0 & \alpha \end{bmatrix} \end{array} \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \xrightarrow[\text{reduction}]{\text{row}} \begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & 0 \\ 25 & \alpha & \alpha & 0 \\ 20 & \alpha & 0 & \alpha \end{bmatrix} \\ \begin{array}{cccc} -20 & 0 & 0 & 0 \end{array} \end{array}$$

$$\xrightarrow[\text{reduction}]{\text{column}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & 0 \\ 5 & \alpha & \alpha & 0 \\ 0 & \alpha & 0 & \alpha \end{bmatrix}$$

M_2

cost reduced $= r = 0 + 20 = 20$.

M_2 is the matrix for node 2

cost of node 2 $= 100 + M_1[A,B] + r = 100 + 0 + 20 = 120$

Find the matrix and cost of node 3

* set row A and coloumn c elmts are α

* Set $M_1[C,A] = \alpha$

* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 0 \\ \alpha & 0 & \alpha & 10 \\ 20 & 0 & \alpha & \alpha \end{bmatrix}$$

* Perform row reduction & column reductn

$$\begin{array}{c} \left[\begin{array}{cccc} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 0 \\ \alpha & 0 & \alpha & 10 \\ 20 & 0 & \alpha & \alpha \end{array}\right] \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \end{array} \xrightarrow[\text{reduction}]{\text{row}} \begin{array}{c} \left[\begin{array}{cccc} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 0 \\ \alpha & 0 & \alpha & 10 \\ 20 & 0 & \alpha & \alpha \end{array}\right] \\ \begin{array}{cccc} 0 & 0 & 0 & 0 \end{array} \end{array}$$

$$\xrightarrow[\text{reduction}]{\text{column}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 0 \\ \alpha & 0 & \alpha & 10 \\ 20 & 0 & \alpha & \alpha \end{bmatrix}$$

M_3

cost reduced $= r = 0$

M_3 is the matrix for node 3

cost of node 3 = cost of node 1 + $M_1[A, C]$ + r

$= 100 + 25 + 0 = 125$

Find the matrix and cost of node 4

* Set row A and column D to α.
* set $M_1[D, A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 0 & \alpha \\ 25 & 0 & \alpha & \alpha \\ \alpha & 0 & 0 & \alpha \end{bmatrix}$$

* Perform row reduction & column reduction

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 0 & \alpha \\ 25 & 0 & \alpha & \alpha \\ \alpha & 0 & 0 & \alpha \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & \;\; 0 & \;\; 0 & \;\; 0 \end{matrix} \end{array} \xrightarrow[\text{column reduction}]{\text{row reduction}} \underset{M_4}{\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 0 & \alpha \\ 25 & 0 & \alpha & \alpha \\ \alpha & 0 & 0 & \alpha \end{bmatrix}}$$

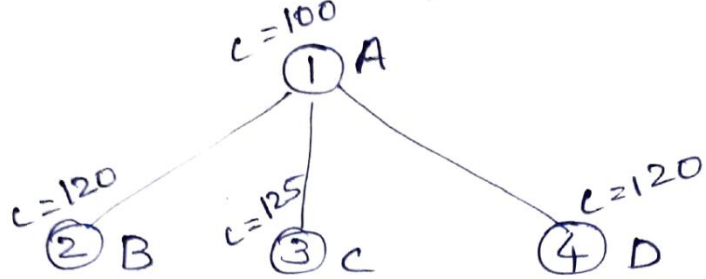
cost reduced $= \gamma = 0$

M_4 is the matrix for node 4

cost of node 4 = ~~At~~ cost of node 1 + $M_1[A,D]$ + γ

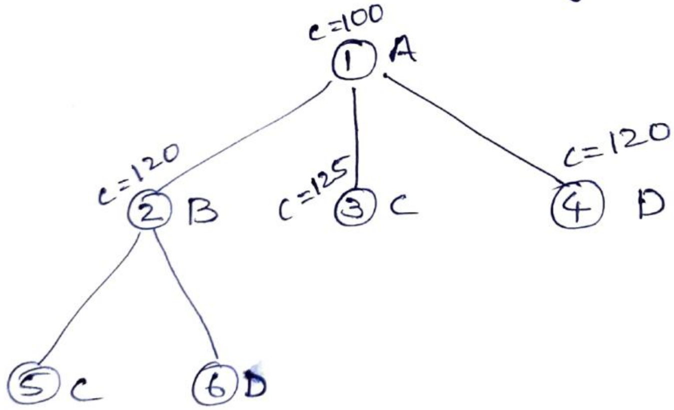
$= 100 + 20 + 0 = 120$

Now the state space tree is



Minimum cost Live nodes are 2 & 4.

Choose node 2 as the next E-node.
Generate child nodes of node 2



Find the matrix and cost of nodes-

* Set row B and column C elmts are α
* Set $M_2[C, A] = \alpha$
* Now the matrix is

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

* Perform row reduction & column reduction

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{bmatrix} \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \\ \begin{array}{cccc} 0 & 0 & 0 & 0 \end{array} \end{array} \xrightarrow[\text{column redtn}]{\text{row redtn}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{bmatrix}$$

M_5

cost reduced $= \gamma = 0$

M_5 is the matrix for node 5

cost of node 5 $=$ cost of node 2 $+ M_2[B,C] + \gamma$

$= 120 + 0 + 0 = 120$

Find the matrix and cost of node 6.

* Set row B and column D elmts are α
* set $M_2[D,A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{bmatrix}$$

* Perform row reduction & column redtn

$$\begin{array}{cc} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{bmatrix} & \begin{array}{c} \\ \\ 5 \\ \\ \end{array} \end{array} \xrightarrow[\text{redn}]{\text{row}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{bmatrix} \xrightarrow[\text{redtn}]{\text{column}} \underbrace{\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{bmatrix}}_{M_6}$$

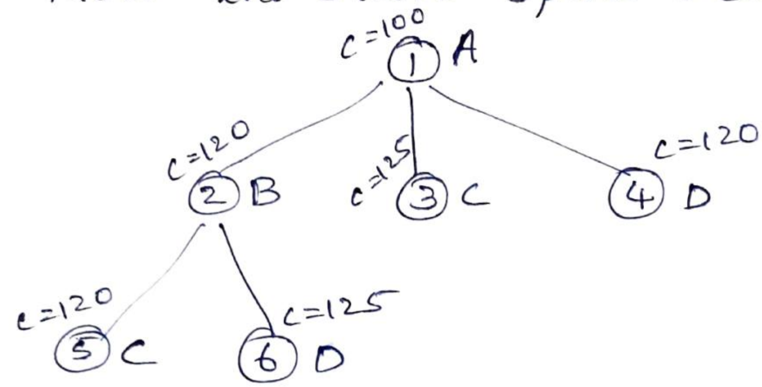
cost reduced $= \gamma = 5$

M_6 is the matrix for node 6.

cost of node 6 $=$ ~~no~~ cost of node 2 $+ M_2[B,D] + \gamma$

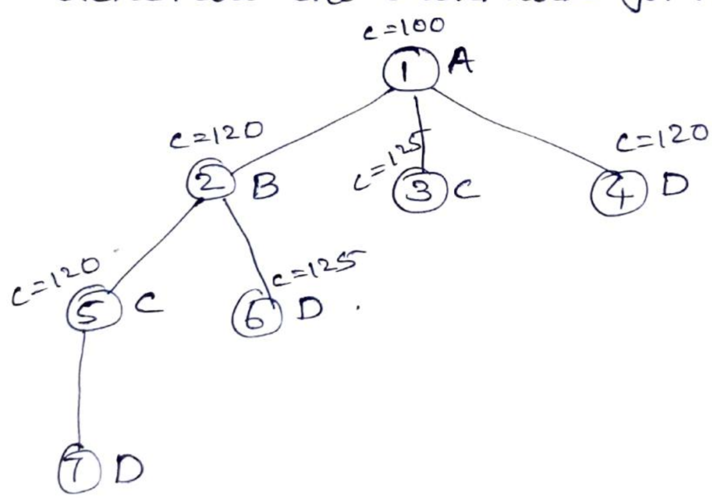
$= 120 + 0 + 5 = 125$

Now the state space tree is



Minimum cost live nodes are node 4 and node 5.
Select node5 as the next E-node.

Generate the child node for node 5.



Find the matrix and cost of node 7

* Set row C and column D elmts are α
* set $M5[D,A] = \alpha$.
* The matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{bmatrix}$$

* Perform row redn & column redn.

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{bmatrix} \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \\ \begin{array}{cccc} 0 & 0 & 0 & 0 \end{array} \end{array} \xrightarrow[\text{column redn}]{\text{row redn}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{bmatrix}$$

M_7

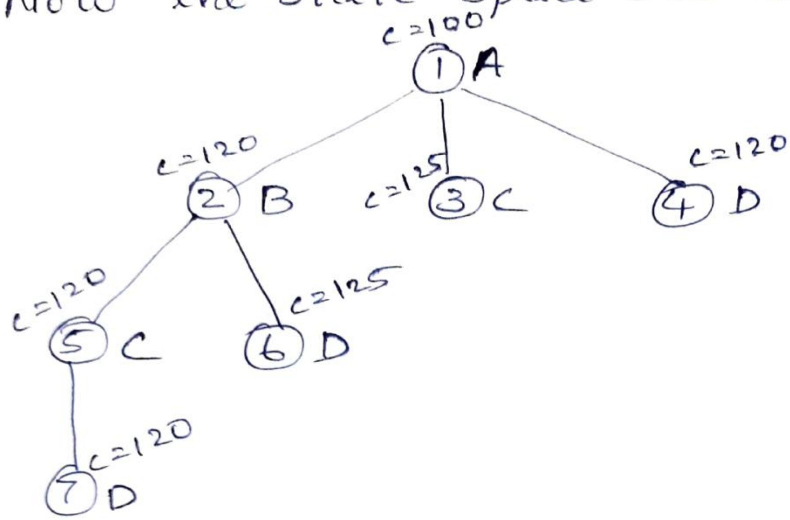
cost reduced = $r = 0$

M_7 is the matrix of node 7

cost of ~~node~~ node 7 = cost of node 5 + $M5[C,D] + r$

$= 120 + 0 + 0 = 120$

Now the state space tree is



Minimum cost live nodes are node 4 & 7.

Select node 7 as the next E-node.

It has no child node.

So the TSP path is <u>A-B-C-D-A</u>

TSP cost = cost of node 7 = <u>120</u>

b) Write Floyd Warshall’s algorithm for finding all pairs shortest path algorithm.
Ans:

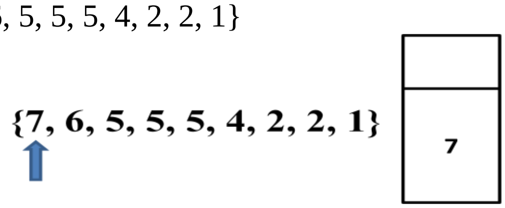
```
Algorithm FloydWarshall(cost[][], n)
{
        for i=1 to n do
                for j=1 to n do
                        D[i, j] = cost[i, j]
        for k := 1 to n do
                for i := 1 to n do
                        for j := 1 to n do
                                D[i, j] = min{D[i, j] , D[i, k] + D[k, j]}
        Return D
}
```

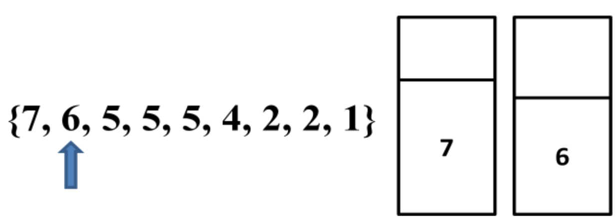
- **Time Complexity**
  - Floyd Warshall Algorithm consists of three loops over all the nodes. Each loop has constant complexities.
  - Hence, the time complexity of Floyd Warshall algorithm = $O(n^3)$, where n is the number of nodes in the given graph.

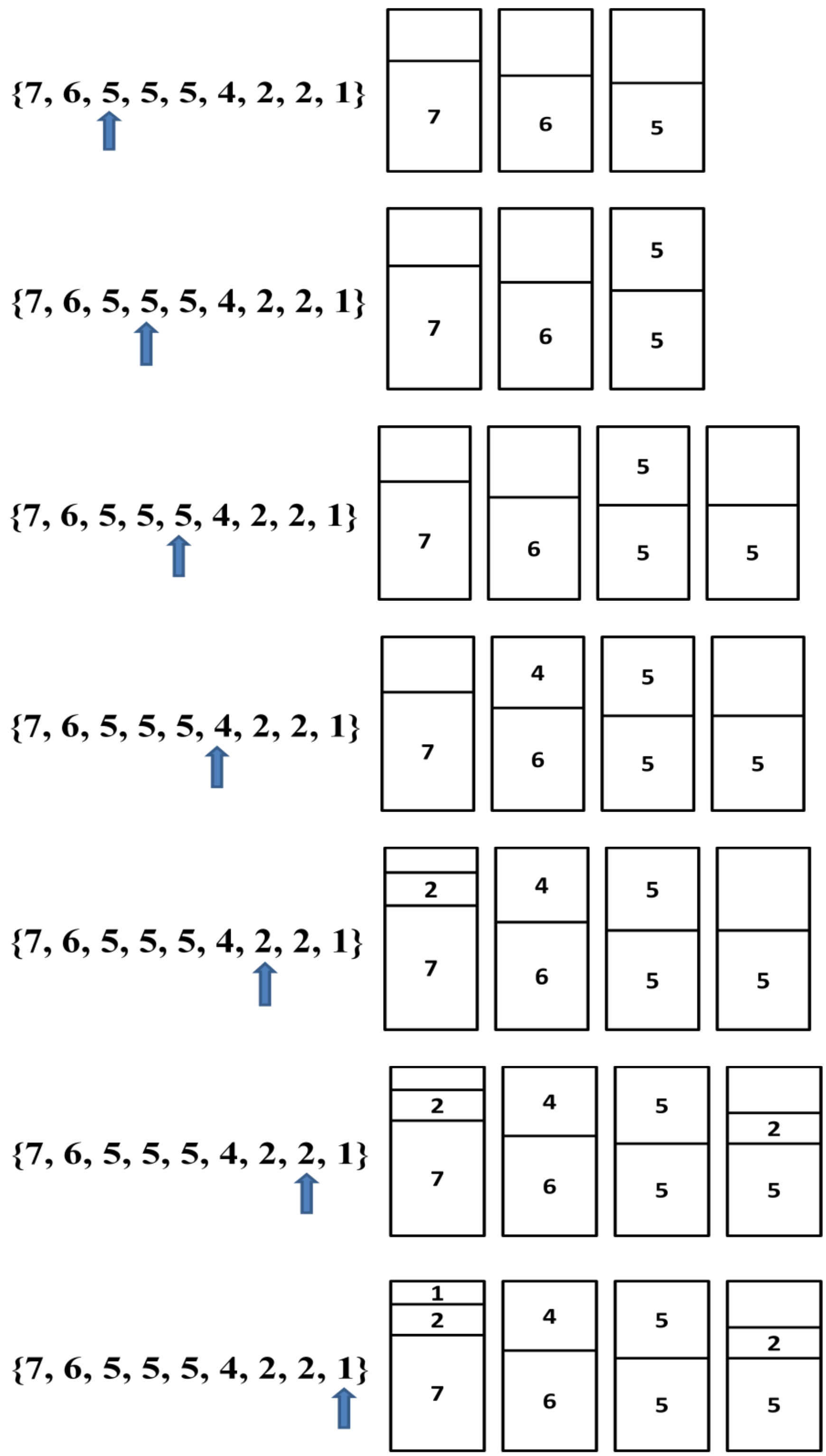
19.

a) Explain the first fit-decreasing strategy of bin packing algorithm.
Ans:

- **First Fit Decreasing Algorithm**
  - Sort the items in the descending order of their size
  - Apply First fit algorithm
  - Time Complexity
    - Best case Time Complexity = $\theta(n \log n)$
    - Average case Time Complexity = $\theta(n^2)$
    - Worst case Time Complexity = $\theta(n^2)$
- **Example:** bin capacity=10, sizes of the items are {5, 7, 5, 2, 4, 2, 5, 1, 6}.
  - Arrange the items in the decreasing order of the weight
    {7, 6, 5, 5, 5, 4, 2, 2, 1}







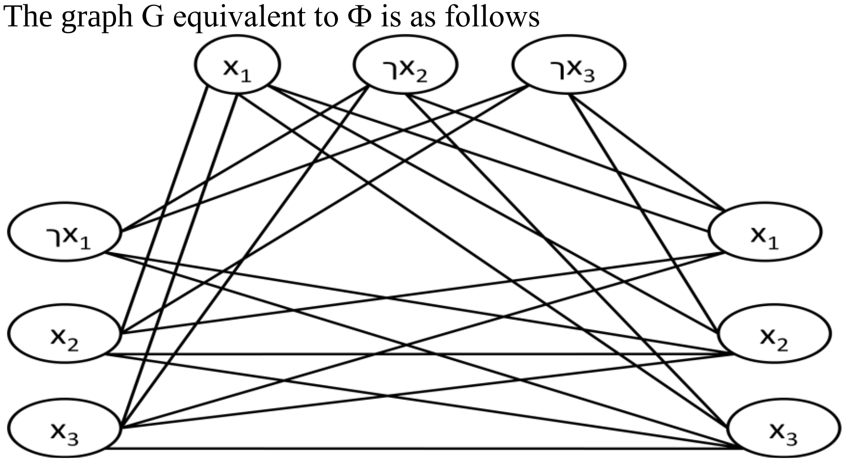
Number of bins required = 4

b) Prove that Clique Decision problem is NP-complete

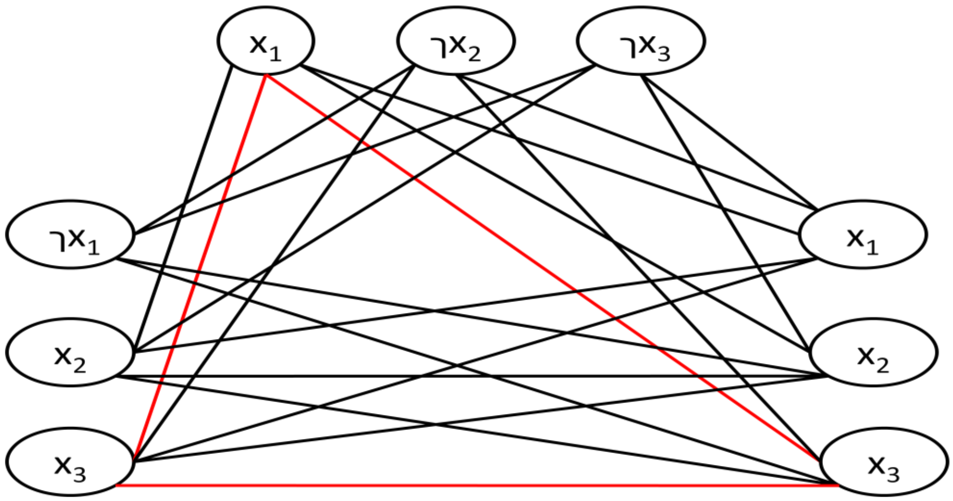
Ans:

- **CLIQUE problem is NP Complete: Proof**
  - Step 1: Write a polynomial time verification algorithm to prove that the given problem is NP
    - Algorithm: Let G= (V,E), we use the set V' ⊆ V of k vertices in the clique as a certificate of G
      1. Test whether V' is a set of k vertices in the graph G
      2. Check whether for each pair (u,v) ∈ V', the edge (u,v) belongs to E.
      3. If both steps pass, then accept. Otherwise reject.
    - This algorithm will execute in polynomial time. Therefore **CLIQUE problem is a NP problem.**
  - Step 2: Write a polynomial time reduction algorithm from 3-CNF-SAT problem to CLIQUE problem(3-CNF-SAT $\leq_p$ CLIQUE)
    - Algorithm
      - Let $\Phi = C_1 \wedge C_2 \ldots\ldots \wedge C_k$ be a Boolean formula in 3CNF with k clauses
      - Each clause C_r has exactly three distinct literals l^r_1, l^r_2, l^r_3.
      - Construct a graph G such that Φ is satisfiable iff G has a click of size k.
      - The graph G is constructed as follows
        - For each clause $C_r = (l^r_1 \vee l^r_2 \vee l^r_3)$ in Φ, we place a triple of vertices V^r_1, V^r_2 and V^r_3 in to V.
        - Put an edge between V^r_i to V^s_i if following two conditions hold
          - V^r_i and V^s_i in different triples(that is r!=s)
          - l^r_i is not a negation of l^s_i.

  - Example: $\Phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
    - The graph G equivalent to Φ is as follows



    - If G has a clique of size k, then Φ has a satisfying assignment. Here k=3.



- G can easily be constructed from Φ in polynomial time.
- So **CLIQUE problem is NP Hard**.

- Conclusion
  - CLIQUE problem is NP and NP Hard. So it is NP-Complete

20.

a) Differentiate Las Vegas and Monte Carlo algorithms
Ans:

- **Randomized Las Vegas Algorithms**
  - Output is always correct and optimal.
  - Running time is a random number
  - Running time is not bounded
  - Example: Randomized Quick Sort
- **Randomized Monte Carlo Algorithms:**
  - May produce correct output with some probability
  - A Monte Carlo algorithm runs for a fixed number of steps. That is the running time is deterministic
- **Example**: Finding an '*a*' in an array of *n* elements
  - **Input**: An array of $n \geq 2$ elements, in which half are '*a*'s and the other half are '*b*'s
  - **Output**: Find an '*a*' in the array

  - **Las Vegas algorithm**

```
Algorithm findingA_LV(A, n)
{
        repeat
        {
                Randomly choose one element out of n elements
```

```
	}until('a' is found)
}
```

- The expected number of trials before success is 2.
- Therefore the time complexity = O(1)

- **Monte Carlo algorithm**

```
Algorithm findingA_MC(A, n, k )
{
	i=0;
	repeat
	{
		Randomly select one element out of n elements
		i=i+1;
	}until(i=k or 'a' is found);
}
```

- This algorithm does not guarantee success, but the run time is bounded. The number of iterations is always less than or equal to k.
- Therefore the time complexity **= O(k)**

b) Explain randomized quick sort with the help of suitable examples

Ans:

Algorithm randQuickSort(A[], low, high)

1. If low >= high, then EXIT
2. While pivot 'x' is not a Central Pivot.
   2.1. Choose uniformly at random a element from A[low..high]. Let the randomly picked element be x.
   2.2. Count elements in A[low..high] that are smaller than x. Let this count be sc.
   2.3. Count elements in A[low..high] that are greater than x. Let this count be gc.
   2.4. Let n = (high-low+1). If sc >= n/4 and gc >= n/4, then x is a central pivot.
3. Partition A[low..high] into two subarrays. The first subarray has all the elements of A that are less than x and the second subarray has all those that are greater than x. Now the index of x be pos.
4. randQuickSort(A, low, pos-1)
5. randQuickSort(A, pos+1, high)

Example:

Consider an unsorted array : [3, 6, 8, 10, 1, 2, 4]

Randomly choose one element as pivot element with the condition that n/4 elements are greater than that pivot element and n/4 elements are less than that pivot element.

Suppose we select element 4 as the pivot element. Here 3 elements are greater than 4 and 3 elements are less than 4.

$n/4 = 7/4 \approx 1$

So the condition satisfied and the element 4 is the pivot element.

Swap the pivot with the 1st element of the array.

Now the array is : [4, 6, 8, 10, 1, 2, 3]

Partition the array

4	**6**	**8**	**10**	**1**	**2**	**3**
	low					**high**

4	**3**	**8**	**10**	**1**	**2**	**6**
	low					**high**

4	**3**	**8**	**10**	**1**	**2**	**6**
		low			**high**	

4	**3**	**2**	**10**	**1**	**8**	**6**
		low			**high**	

4	**3**	**2**	**10**	**1**	**8**	**6**
			low	**high**		

4	**3**	**2**	**1**	**10**	**8**	**6**
			low	**high**		

4	**3**	**2**	**1**	**10**	**8**	**6**
			high	**low**		

1	**3**	**2**	**4**	**10**	**8**	**6**
			high	**low**		

1	**3**	**2**	**4**	**10**	**8**	**6**

Now the location of 4 is fixed. It partition the array into 2 subarrays
Subarray1: [1, 3, 2]
Subarray2: [10, 8, 6]

Then recursively call the two subarrays and perform the above operations.
First consider the subarray1: [1, 3, 2]
Suppose 2 is the randomly selected pivot element. Swap it with the 1st element in that array. Now the array becomes [2, 3, 1].
Partition this array

2	**3**	**1**
	low	**high**

2	**1**	**3**
	low	**high**

2	**1**	**3**
	high	**low**

1	**2**	**3**
	high	**low**

1	**2**	**3**

Now 2 is placed in its actual location. It partition the array in to 2 subarrays. These subarrays contain only one element. So no need for further sorting.

Now consider the subarray2: [10, 8, 6]
Suppose 8 is the randomly selected pivot element. Swap it with the 1st element in that array. Now the array becomes [8, 10, 6].

Partition this array

8	**10**	**6**
	low	**high**

8	**6**	**10**
	low	**high**

8	**6**	**10**
	high	**low**

6	**8**	**10**
	high	**low**

6	**8**	**10**

Now 8 is placed in its actual location. It partition the array in to 2 subarrays. These subarrays contain only one element. So no need for further sorting.

The resultant sorted array is :

1	**2**	**3**	**6**	**8**	**10**