# APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

**Sixth Semester B.Tech Degree Supplementary Examination May 2023 (2019 Scheme)**

**Course Code: CST306**

**Course Name: ALGORITHM ANALYSIS AND DESIGN**

## PART A

1. Let $T(n)=7n+4$. Prove that this is of the order of $\Omega(n)$.

**Ans:**

$f(n) = 7n+4$

$7n+4 \geq 7n$ for all $n \geq 1$

Here $c=7$ $n_0=1$ $g(n)= n$

$7n+4 = \mathbf{\Omega(n)}$

2. Solve the following recurrence using Master theorem

a) $T(n)=8T(n/2)+n^2$
b) $T(n)=2T(n/2)+n$

**Ans:**

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

$a=8$ $b=2$ $n^2 = \Theta(n^2 \log^0(n))$ $k=2$ $p=0$

$b^k = 2^2 = 4$

Here If $a>b^k$ then $T(n) = \theta(n^{(\log_b a)}) = \theta(n^{(\log_2 8)}) = \theta(n^3)$

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

$a=2$ $b=2$ $n = \Theta(n^1 \log^0(n))$ $k=1$ $p=0$

$b^k = 2^1 = 2$

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

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

$= \Theta(n^{(\log_2 2)} \log^1(n))$

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

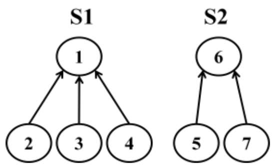
3. Define MAKE_SET(x), UNION(x,y) and FIND_SET(x) operations of disjoint set data structure with a suitable example.

**Ans:**

- **Make Set Operation**
  - Make-set(x) function will create a set that containing one element x.
  - **Algorithm Make-set(x)**
    1. Create a new linked list that contains one node n
    2. n→data=x
    3. n→parent = NULL

- **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.



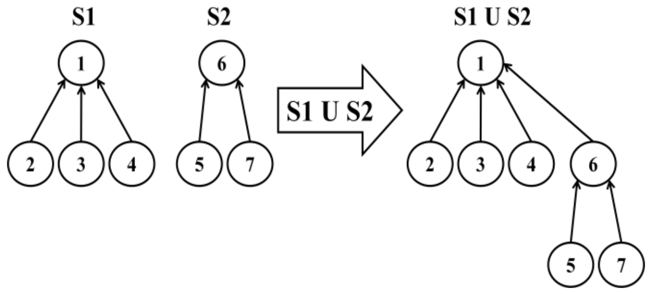
Find(3) will return 1, which is the root of the tree that 3 belongs
Find(6) will return 6, which is the root of the tree that 6 belongs

- **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

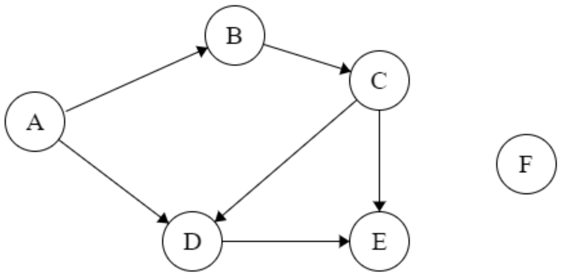


- **Union Algorithm**

```
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

4. Find any ONE topological ordering of the following the graph

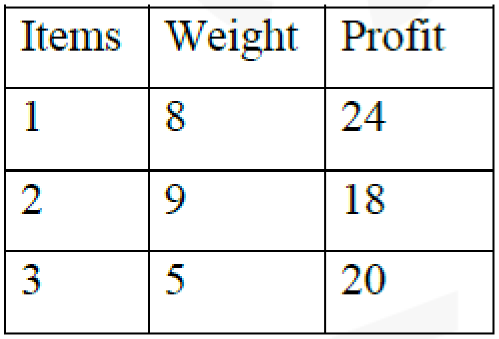


**Ans:**
A, B, C, D, E, F

5. Give the control abstraction of Divide and Conquer algorithm design strategy
**Ans:**

```
Algorithm DAndC(P)
{
    if Small(P) then
        return S(P)
    else
    {
        Divide P into smaller instances $P_1, P_2, . . . . P_k$,  $k \geq 1$;
        apply DAndC to each of these sub-problems;
        return Combine(DAndC($P_1$), DAndC($P_2$), . . . . , DAndC($P_k$));
    }
}
```

6. Apply greedy algorithm for fractional knapsack to find the optimal ordering for loading the items in the knapsack. Let the knapsack capacity, M=15



| Items | Weight | Profit |
|---|---|---|
| 1 | 8 | 24 |
| 2 | 9 | 18 |
| 3 | 5 | 20 |

**Ans:**

m = 15
n = 3
i ➔ { 1, 2, 3 }
P = {24, 18, 20 }
W = {8, 9, 5 }
P/W= {3, 2, 4 }

**Sort p[i]/w[i] ≥ p[i+1]/w[i+1]**
i ➔ { 3, 1, 2 }
P = {20, 24, 18 }
W= {5, 8, 9 }

| Item | Pi | Wi | Xi | U = U-Wi |
|---|---|---|---|---|
| 3 | 20 | 5 | 1 | 10 |
| 1 | 24 | 8 | 1 | 2 |
| 2 | 18 | 9 | 2/9 | 0 |

Total Profit = Σ Pi * Xi

=24x1 + 18x2/9 + 20x1 = **48**

Solution vector **X ={1, 2/9 , 1}**

7. Explain about the structure of an optimal paranthesization of matrix-chain multiplication problem

   **Ans:**

   - $A_{i..j}$ denote the matrix that results from evaluating the product $A_iA_{i+1} \dots A_j$ where i<= j
   - If i< j, we must split the problem into two subproblems ($A_i A_{i+1} \dots A_k$ and $A_{k+1}A_{i+1} \dots A_j$ ), for some integer k in the range i<= k < j.
   - That is, for some value of k, we first compute the matrices $A_{i..k}$ and $A_{k+1..j}$. Then multiply them together to produce the final product $A_{i..j}$ .
   - Total cost = Cost of computing the matrix $A_{i..k}$+ Cost of computing $A_{k+1..j}$+ Cost of multiplying them together.

8. Distinguish the branch-and-bound technique from the backtracking 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. |

9. Discuss the need for approximation algorithm.

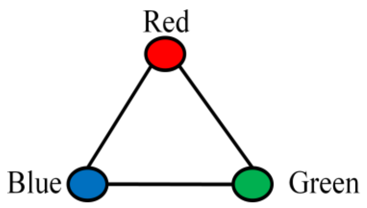
   **Ans:**

   - **Approximate Solution:** A feasible solution with value close to the value of optimal solution is called an approximate solution
   - **Approximation Algorithms**: An algorithm that returns near optimal solution is called Approximation Algorithm.
   - Approximation algorithms have two main properties:
     - They run in polynomial time
     - They produce solutions close to the optimal solutions
   - Approximation algorithms are useful to give approximate solutions to NP complete optimization problems.
   - It is also useful to give fast approximations to problems that run in polynomial time.

10. Define graph colouring 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

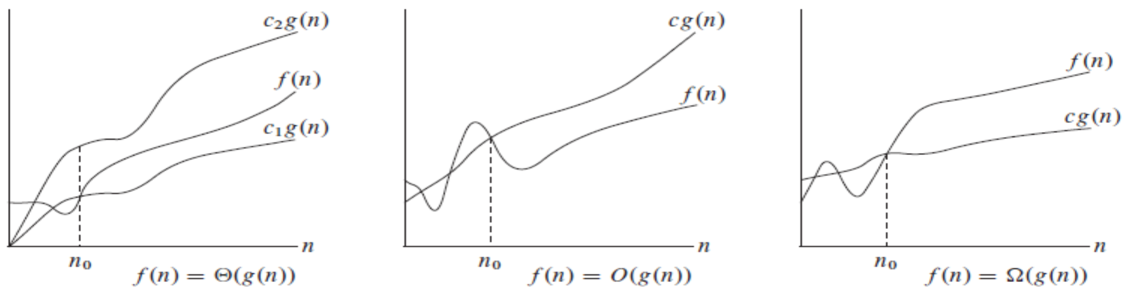
## PART B

11.

a) Define the asymptotic notations: Big Oh, Big Omega, Theta, little oh and little omega

**Ans:**

- **Asymptotic Notations**
  - It is the mathematical notations to represent frequency count. 5 types of asymptotic notations
    - **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 (Ө)**
      - The function f(n) = Ө (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).



- **Little Oh (o)**
  - The function f(n) = o(g(n)) iff for any positive constant c>0, there exists a constant $n_0>0$ such that $0 \le f(n) < c\, g(n)$ for all $n \ge n_0$
  - It is asymptotically loose upper bound

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

  g(n) becomes arbitrarily large relative to f(n) as n approaches infinity

- **Little Omega (ω)**
  - The function f(n) = ω(g(n)) iff for any positive constant c>0, there exists a constant $n_0>0$ such that $f(n) > c\, g(n) \ge 0$ for all $n \ge n_0$
  - It is asymptotically loose lower bound

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

  f(n) becomes arbitrarily large relative to g(n) as n approaches infinity

b) Solve the following recurrence using recursion tree method:

1)T $(n)=T(n/2)+1$, $T(1)=1$
2) T$(n)=2T(n/2)+n^2$, $T(1)=1$
**Ans:**
1)

T(n) = 1 + T(n/2)
= 1 + [1+ T($n/2^2$)]  = 2+ T($n/2^2$)
= 2+ [1+ T($n/2^3$)]  = 3+ T($n/2^3$)
……………………
= k + T($n/2^k$)  $k^{th}$ term
Assume $n/2^k = 1$ ➔ $2^k = n$ ➔ $k = \log_2(n)$
T(n) = $\log_2(n)$ + T(1)  = $\log_2(n) + 1$
= **O($\log_2$(n) )**

2)

T(n) = $n^2 + 2T(n/2)$
= $n^2 + 2[(n/2)^2+2T(n/2^2)]$  = $n^2+n^2/2 + 2^2\,T(n/2^2)$
= $n^2 + n^2/2 + 2^2\,[(n/2^2)^2+2T(n/2^3)]$

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

……………………

$= n^2[1 + (1/2) + (1/2)^2 + \ldots + (1/2)k\text{-}1^2\,] + 2^k\, T(n/2^k)$ kth term

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

T(n) $= n^2\, [(1-(1/2)^k) / (1-(1/2))] + n\, T(1)$

$= 2\, n^2\, [\, 1-(1/2^k)] + n = 2\, n^2\, [\, 1-(1/2^{\log n})] + n$

$= 2\, n^2\, [\, 1-(1/n^{\log 2})] + n = 2\, n^2\, [\, 1-(1/n)] + n$

$= 2\, n^2 - 2n + n$

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

12.

a) Perform complexity analysis for the following code segments

```
1) For i=1 to n do

     For j = 1 to n do
             A=B * C
     End for
   End for

2) Function F(n)
   {
     If(n==0)
             return(1)
     Else
             return (n*F(n-1))
   }
```

**Ans:**

1)

Most frequently executing statement is A=B*C. It will execute $n^2$ times.

So the time complexity = $\mathbf{O(n^2)}$

2)

| | **Step/Execution** | **Frequency Count** | | **Total Frequency Count** | |
|---|---|---|---|---|---|
| | | **n==0** | **n>0** | **n≤0** | **n>0** |
| Function F(n) | 0 | 0 | 0 | 0 | 0 |
| { | 0 | 0 | 0 | 0 | 0 |
| If(n==0) | 1 | 1 | 1 | 1 | 1 |
| return(1) | 1 | 1 | 0 | 1 | 0 |
| Else | 0 | 0 | 0 | 0 | 0 |
| return (n*F(n-1)) | 1 + T(n-1) | 0 | 1 | 0 | 1 + T(n-1) |
| } | 0 | 0 | 0 | 0 | 0 |
| | | | | **2** | **2 + T(n-1)** |

$$\textbf{Time Complexity} = T(n) = \begin{cases} \mathbf{2} & \textbf{if } n<=0 \\ \mathbf{2 + T(n-1)} & \textbf{Otherwise} \end{cases}$$

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

=2 + 2 + T(n-2)
=2 + 2 + 2+ T(n-3)
=2x3 + T(n-3)
. . .
=2xn +T(n-n)
=**2n + 2**

b) Discuss the concept of best case, worst case and average case complexity of an algorithm with linear search algorithm.

**Ans:**

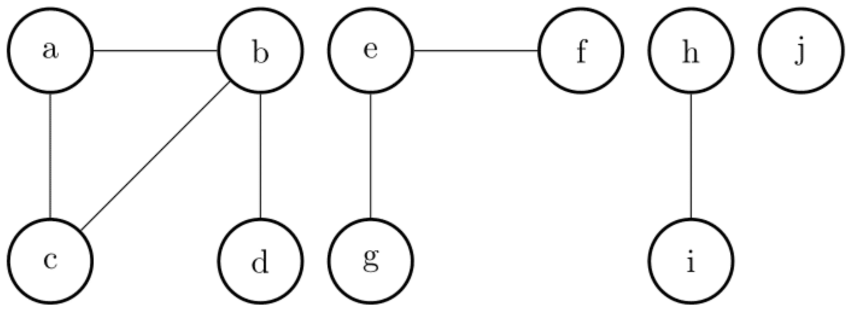
- In certain case we cannot find the exact value of frequency count. In this case we have 3 types of frequency counts
  - Best Case : It is the minimum number of steps that can be executed for a given parameter
  - Worst Case: It is the maximum number of steps that can be executed for a given parameter
  - Average Case: It is the average number of steps that can be executed for a given parameter
- Example: Linear Search
  - Best Case: Search data will be in the first location of the array.
  - Worst Case: Search data does not exist in the array
  - Average Case: Search data is in the middle of the array.

| | **Best Case** | | | **Worst Case** | | | **Average Case** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **S/E** | **FC** | **TFC** | **S/E** | **FC** | **TFC** | **S/E** | **FC** | **TFC** |
| Algorithm Search(a,n,x) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| { | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| for i:=1 to n do | 1 | 1 | 1 | 1 | n+1 | n+1 | 1 | n/2 | n/2 |
| if a[i] ==x then | 1 | 1 | 1 | 1 | n | n | 1 | n/2 | n/2 |
| return i; | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| return -1; | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| } | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **3** | | | **2n + 2** | | | **n+1** |

Best Case Complexity = **3** = **Ω(1)**
Worst Case Complexity = **2n + 2** = **O(n)**
Average Case Complexity = **n+1** = **ϴ(n)**

13.

a) What are the operations supported by Disjoint Data Structure? Explain the working of Disjoint Set Data Structure for computing Connected Components of an undirected graph given in the following Figure.

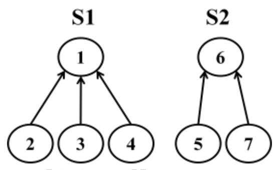


**Ans:**

- **Disjoint Set operations**
    - Make set
    - Union
    - Find

- **Make Set Operation**
    - Make-set(x) function will create a set that containing one element x.
    - **Algorithm Make-set(x)**
        1. Create a new linked list that contains one node n
        2. n→data=x
        3. n→parent = NULL

- **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.



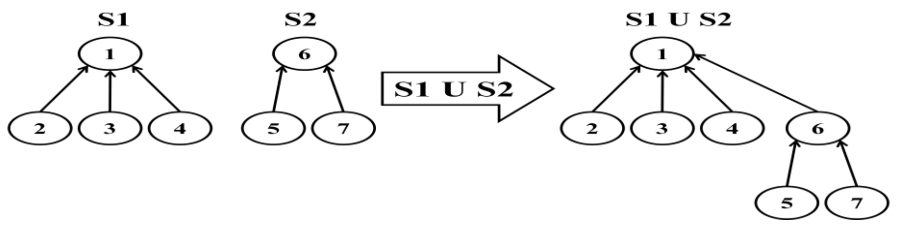
Find(3) will return 1, which is the root of the tree that 3 belongs
Find(6) will return 6, which is the root of the tree that 6 belongs

- **Find Algorithm**

```
Algorithm Find(n)
3.  while n→parent != NULL do
        1.1 n = n→parent
4.  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



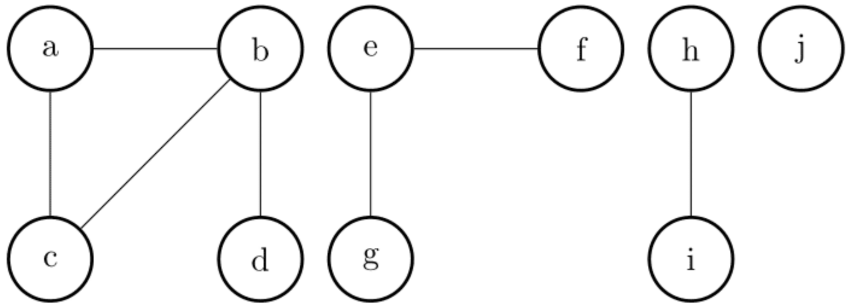
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| P | -1 | 1 | 1 | 1 | 6 | 1 | 6 |

- **Union Algorithm**

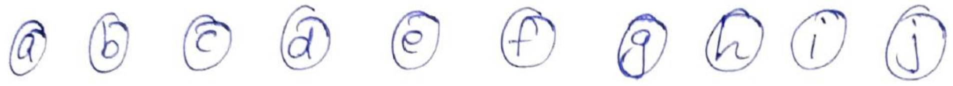
```
Algorithm Union(a, b)
4. X =Find(a)
5. Y = Find(b)
6. If X != Y then
    1. Y→parent = X
```

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

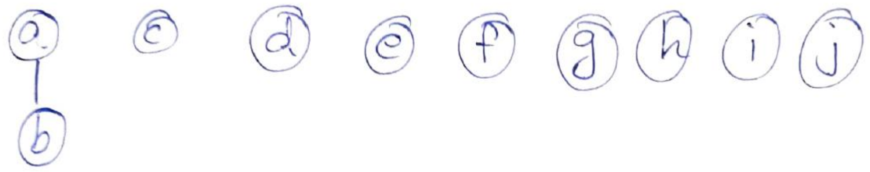
- The Disjoint Set Data Structure is a way to efficiently represent a collection of disjoint sets and perform operations on them. It's commonly used for tasks like finding connected components in an undirected graph.
- Here's how it works:
  - **Initialization**: Each vertex in the graph is initially placed in its own set.
  - **Union Operation (Merge)**: When an edge between two vertices is encountered, we merge the sets containing those vertices. This is done by finding the root of the sets each vertex belongs to and making one of the roots the parent of the other. This effectively merges the two sets into one.
  - **Find Operation (Parent)**: This operation determines which set a particular vertex belongs to. It's done by recursively traversing through the parent pointers until the root of the set is reached. The root of each set serves as a unique identifier for that set.
  - **Path Compression (Optimization)**: To improve the efficiency of the Find operation, we apply path compression. This means that when we find the root of a set for a vertex, we make all the vertices along the path from that vertex to the root point directly to the root. This flattens the structure of the tree representing the sets, reducing the time complexity of future Find operations.
- Using these operations, we can efficiently find connected components in the graph:



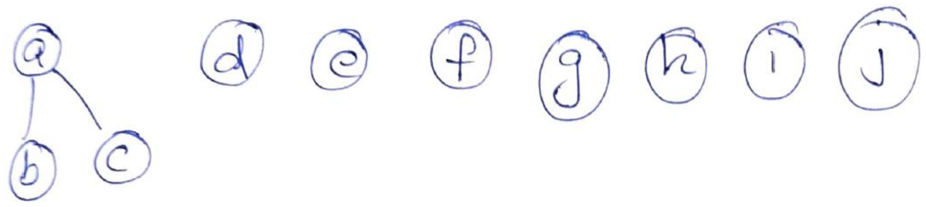
Initially all vertices are belong to different sets



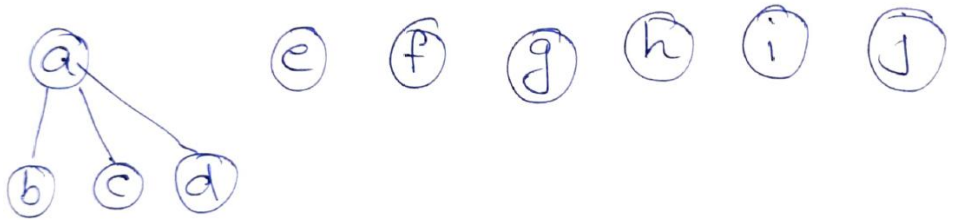
Consider all edges one at a time. Initially consider the edge (a, b). Using FIND operation we came to know that a and b are belongs to different sets. So perform UNION operation. The resultant sets are



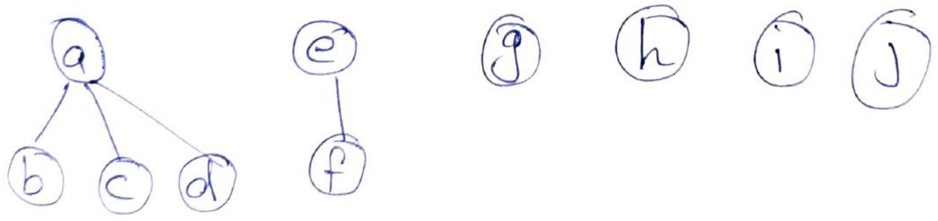
Then consider the edge (a, c). Using FIND operation we came to know that a and c are belongs to different sets. So perform UNION operation. The resultant sets are



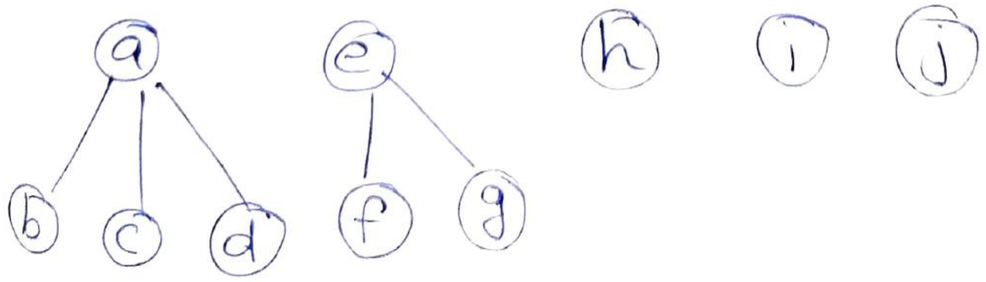
Then consider the edge (b, c). Using FIND operation we came to know that b and c are belongs to the same set. So do nothing.
Then consider the edge (b, d). Using FIND operation we came to know that b and d are belongs to different sets. So perform UNION operation. The resultant sets are



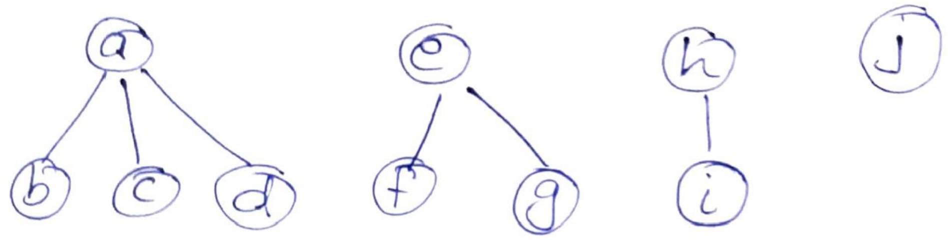
Then consider the edge (e, f). Using FIND operation we came to know that e and f are belongs to different sets. So perform UNION operation. The resultant sets are



Then consider the edge (e, g). Using FIND operation we came to know that e and g are belongs to different sets. So perform UNION operation. The resultant sets are



Then consider the edge (h, i). Using FIND operation we came to know that h and i are belongs to different sets. So perform UNION operation. The resultant sets are



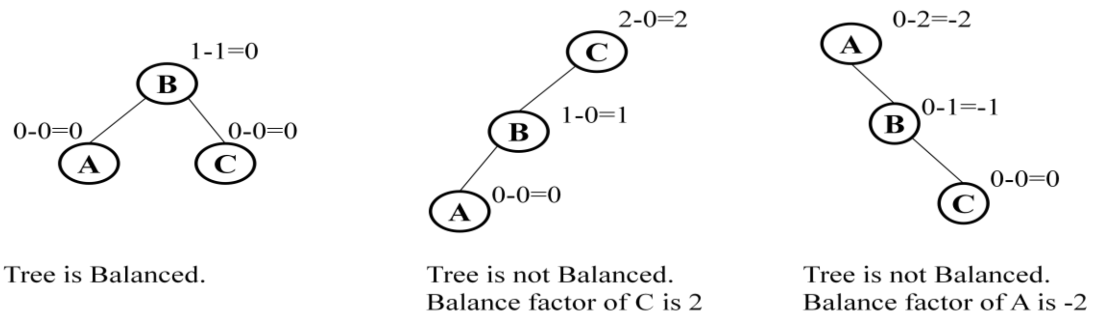
All edges are completed. From the above sets we can conclude that there are 4 connected components.
{a,b,c,d}
{e,f,g}
{h,i}
{j}

b) Illustrate the advantage of AVL trees with a suitable example. Discuss the various rotations required to balance the height of AVL tree during insertion and deletion

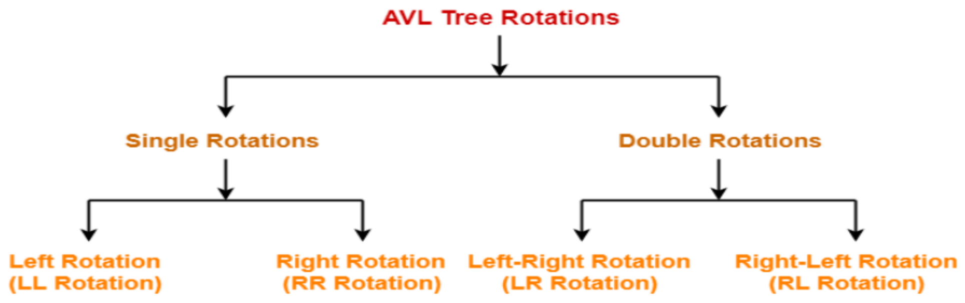
**Ans:**

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

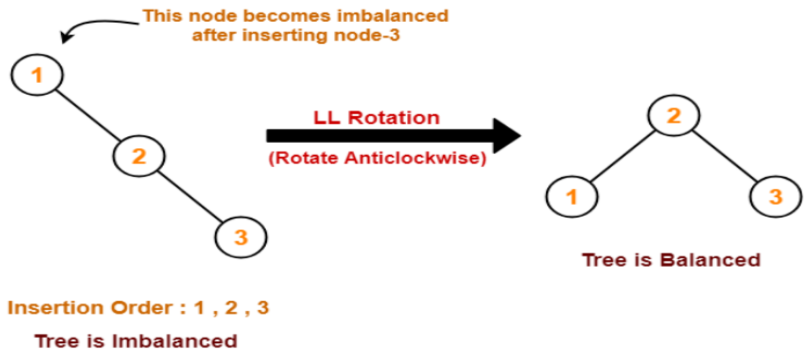
- **Balance Factor**
  - Balance Factor of a node = height of left subtree – height of right subtree
  - In an AVL tree balance factor of every node is -1,0 or +1
  - Otherwise the tree will be unbalanced and need to be balanced.



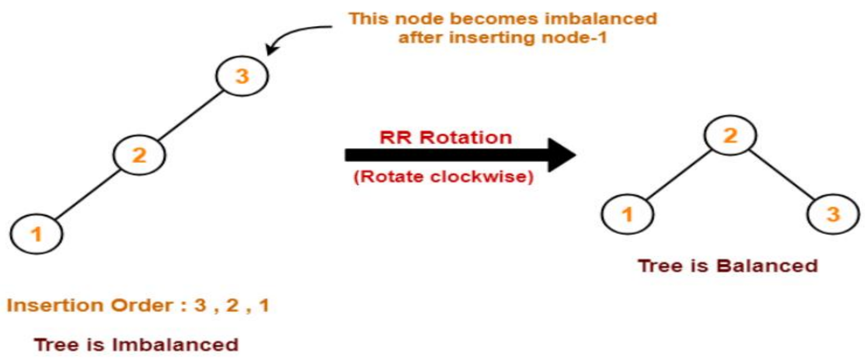
- **Why AVL Tree?**
  - Most of the Binary Search Tree(BST) operations (eg: search, insertion, deletion etc) take O(h) time where h is the height of the BST.
  - The minimum height of the BST is log n
  - The height of an AVL tree is always O(log n) where n is the number of nodes in the tree.
  - So the time complexity of all AVL tree operations are O(log n)
- 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



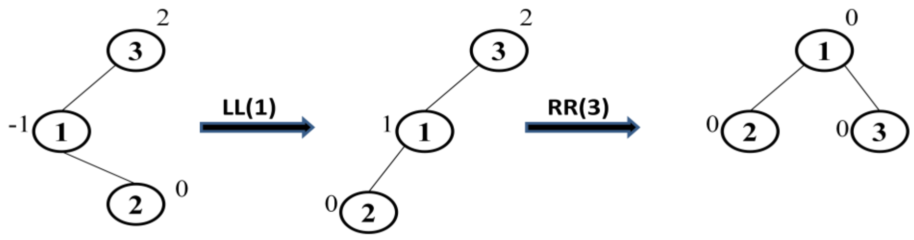
- **Single Left Rotation(LL Rotation)**
  - In LL rotation every node moves one position to left from the current position



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

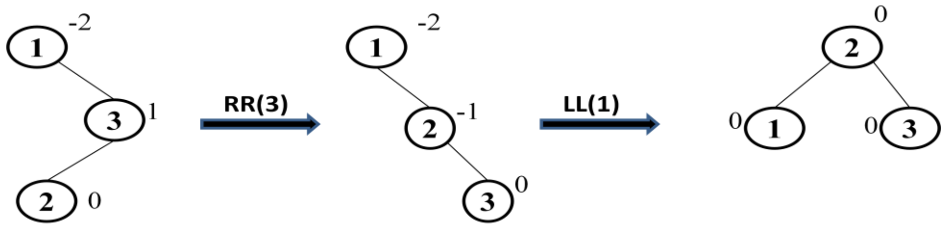

This node becomes imbalanced after inserting node-1
3
2
1
RR Rotation
(Rotate clockwise)
2
1
3
Tree is Balanced
Insertion Order : 3 , 2 , 1
Tree is Imbalanced

- **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

14.

a) Give BFS algorithm for graph traversal and perform its complexity analysis
**Ans:**

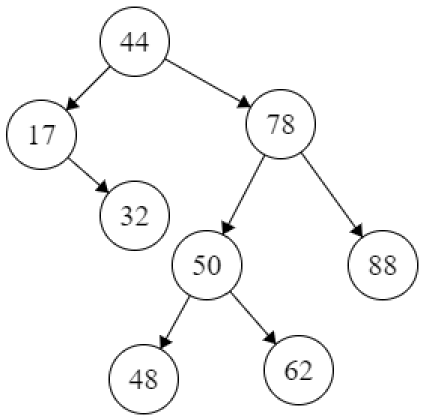
- **Algorithm BFS(G, u)**

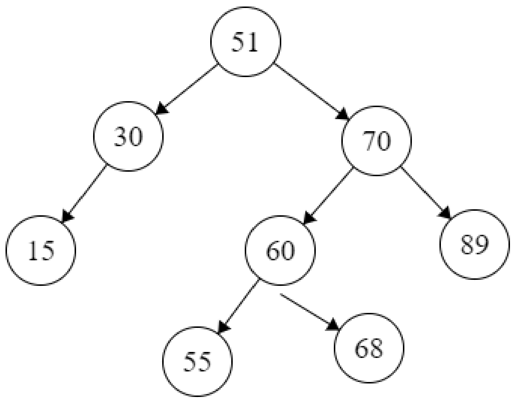
```
1. Set all nodes are unvisited
2. Mark the starting vertex u as visited and put it into an empty Queue Q
3. While Q is not empty
   3.1 Dequeue v from Q
   3.2 While v has an unvisited neighbor w
       3.2.1 Mark w as visited
       3.2.2 Enqueue w into Q
4. If there is any unvisited node x
   4.1 Visit x and Insert it into Q. Goto step 3
```

- **Complexity**
  - If the graph is represented as an adjacency list
    - Each vertex is enqueued and dequeued atmost once. Each queue operation take O(1) time. So the time devoted to the queue operation is **O(V).**
    - The adjacency list of each vertex is scanned only when the vertex is dequeued. Each adjacency list is scanned atmost once. Sum of the lengths of all adjacency list is |E|. Total time spend in scanning adjacency list is **O(E).**
    - Time complexity of BFS = O(V) + O(E) = **O(V+ E).**
    - **In a dense graph:**
      - $E=O(V^2)$
      - Time complexity= $O(V) + O(V^2) = \mathbf{O(V^2)}$
  - 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 BFS = $\mathbf{O(V^2)}$

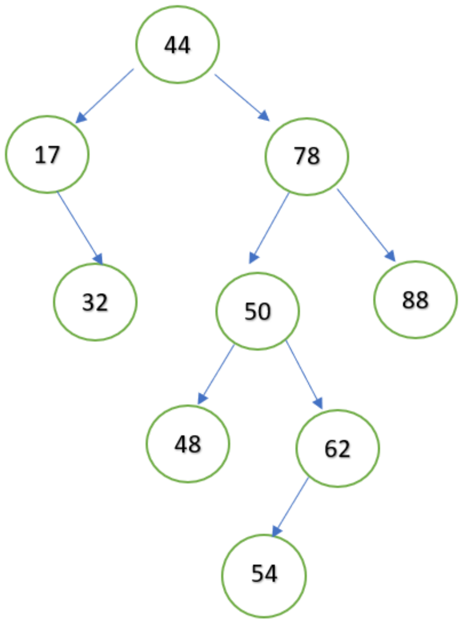
b) Perform the following operations in the given AVL trees.

1) Insert 54 in Tree 1 2) Delete 15 from Tree 2.





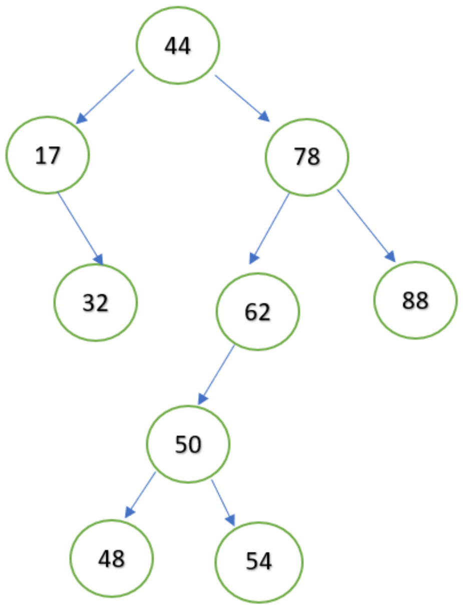
**Ans:**
1)
Inset 54→



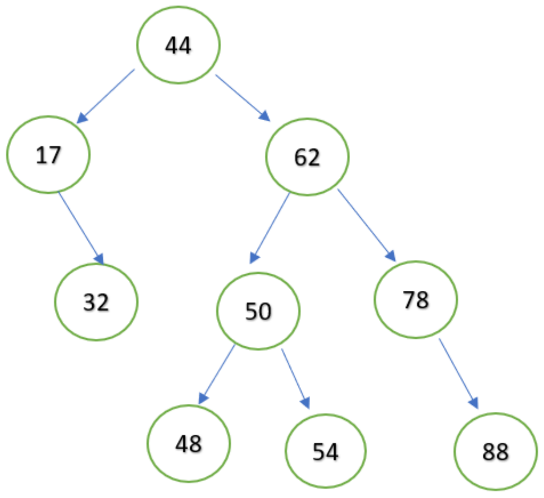
X=78,Y=50,Z=54

PERFORM RIGHT LEFT ROTATION

LL(50)→

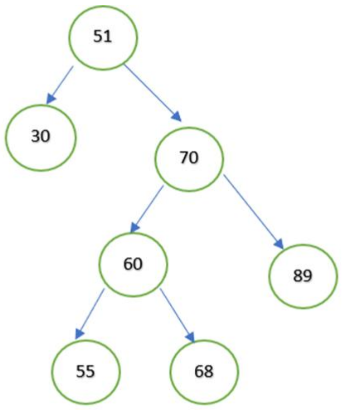


RR(78)→



2)

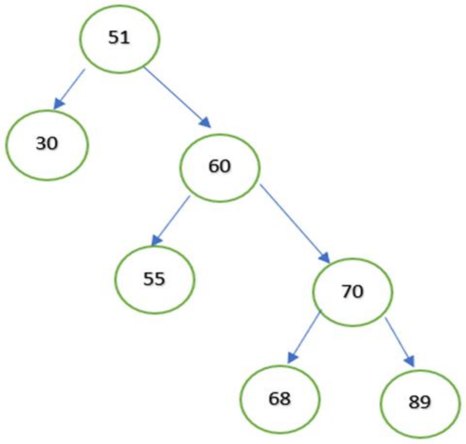
Delete 15→



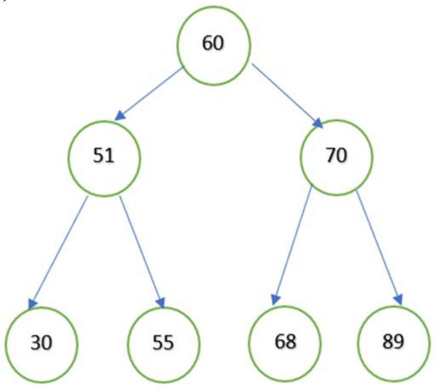
X=51,y=10,z=60

PERFORM RIGHT LEFT ROTATION

RR(70)→



LL(60)→



15.

a) Write a divide and conquer algorithm for 2-way merge sort and perform its complexity analysis.

**Ans:**

- Given a sequence of n elements a[l],…...a[n]. Split this array into two sets a[l],,.a.[n/2] and a[(n/2)+1],…a[n]. Each set is individually sorted, and the resulting sorted sequences are merged to produce a single sorted sequence of n element.

```
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($(x \le mid)$ and $(y \le high)$) do
        {
                if ( $a[x] \le a[y]$ ) then
                {
                        b[i] = a[x];
                        x = x+1;
                }
                else
                {
                        b[i] = a[y];
                        y = y+1;
                }
                i=i+1;
        }
        if( $x \le mid$) 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**

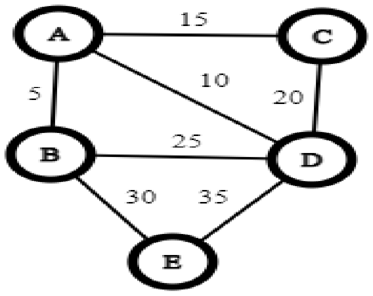
  $$T(n) = \begin{cases} a & \text{if } n=1 \\ 2\,T(n/2) + cn & \text{Otherwise} \end{cases}$$

  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

  $T(n) = 2\,T(n/2) + c\,n$
  $= 2(2\,T(n/4)+c(n/2)) + c\,n$
  $= 2^2T(n/2^2) + 2\,c\,n$
  $= 2^3T(n/2^3) + 3\,c\,n$
  . . . . . . . . . . . . . .
  $= 2^kT(n/2^k) + k\,c\,n$ [Assume that $2^k = n$ □ $k=\log n$]
  $= n\,T(1) + c\,n\log n$
  $= a\,n + c\,n\log n$
  $=$ **$O(n \log n)$**

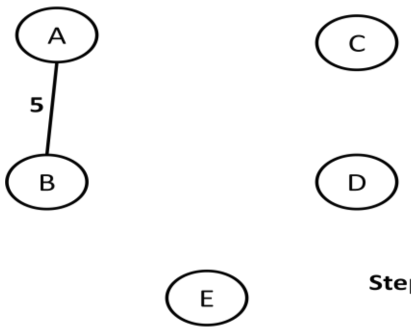
  Best Case, Average Case and Worst Case Complexity of Merge Sort **= O(n log n)**

b) Give Kruskal's algorithm for minimum cost spanning tree computation. Apply the algorithm to find the minimum cost spanning tree for the following graph

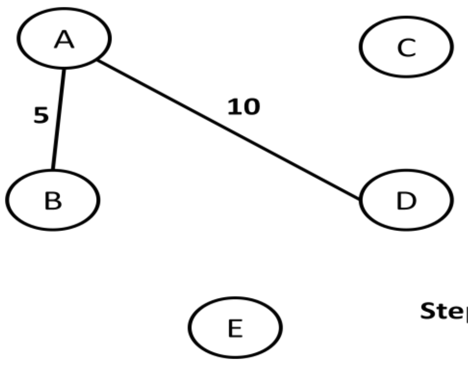


**Ans:**

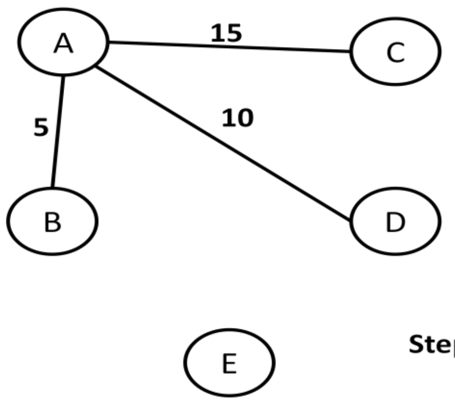
```
Algorithm Kruskals(E, cost, n, t)
{
    Construct a heap out of edge costs using Heapify();
    for i=1 to n do
        parent[i] = -1;
    i=0;
    mincost = 0.0;
    while (i < n-1) and (heap not empty) do
    {
        Delete a minimum cost edge (u, v) from the heap and reheapify using Adjust();
        j = Find(u);
        k = Find(v);
        if j ≠ k then
        {
            i = i+1;
            t[i, 1] = u;    t[i, 2] = v;
            mincost = mincost + cost[u, v];
            Union(j, k);
        }
    }
    if i ≠ n-1 then
        Write ("No Spanning Tree");
    else
        return mincost;
}
```



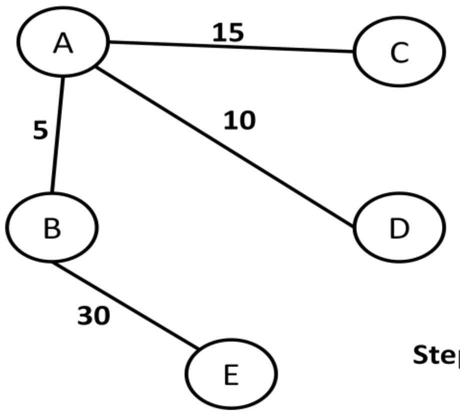
Step 1



Step 2



Step 3



Step 4

**This is the minimum cost spanning tree and its cost = 60**

16.

a) Write Dijkstra’s algorithm for single source shortest path. Perform its complexity analysis.

**Ans:**

- **Algorithm Dijkstra(G,W, S)**

```
1. For each vertex v in G
   1.1 distance[v] = infinity
   1.2 previous[v] = Null
2. distance[S] = 0
```

```
3. Q = set of vertices of graph G
4. While Q is not empty
   4.1 u = vertex in Q with minimum distance
   4.2 remove u from Q
   4.3 for each neighbor v of u which is still in Q
       4.3.1  alt = distance[u] + W(u,v)
       4.3.2  if alt < distance[v]
              4.3.2.1 distance[v] = alt
              4.3.2.2 previous[v] = u
5. Return distance[], previous[]
```

- **Complexity**
  - The complexity mainly depends on the implementation of Q
  - The simplest version of Dijkstra's algorithm stores the vertex set *Q* as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in *Q*. In this case, the running time is $O(E + V^2)$ = $\mathbf{O(V^2)}$
  - Graph represented using adjacency list can be reduced to **O(E log V)** with the help of binary heap.

b) Write the control abstraction for Greedy design technique. Give a greedy algorithm for fractional knapsack problem.
**Ans:**

- **Control Abstraction**

```
Greedy(a, n)   //a[1..n] contains n inputs
{
        solution = Φ;
        for i=1 to n do
        {
                x = Select(a);
                if Feasible(solution, x) then
                        solution = Union(solution, x);
        }
        return solution;
}
```

  - Select() selects an input from the array a[] and remove it. The selected input value is assigned to x.
  - Feasible() is a Boolean valued function that determines whether x can be included into the solution subset.
  - Union() combines x with the solution and updates the objective function.

- **Fractional Knapsack Problem**
  - We are given with *n* objects and a knapsack(or bag) of capacity *m*. The object *i* has weight $W_i$ and profit $P_i$. If a fraction $X_i$ is placed into the knapsack, then a profit $P_iX_i$ is obtained. The objective is to obtain an optimal solution of the knapsack that maximizes the total profit earned.
  - The total weight of all the chosen objects should not be more than *m*.

- Fractional knapsack problem can be stated as

$$\text{Maximize } \sum_{i=1}^{n} PiXi \tag{1}$$

$$\text{Subject to } \sum_{i=1}^{n} WiXi \leq \text{m} \tag{2}$$

$$0 \leq Xi \leq 1 \text{ and } 1 \leq i \leq n \tag{3}$$

  The profits and weights are positive numbers.
- A feasible solution is one that satisfies equation 2 and 3.
- An optimal solution is a feasible solution that satisfies equation 1.
- In greedy strategy we are arranging the objects in the descending order of profit/weight.
- **Algorithm**

```
Algorithm GreedyKnapsack(m, n)
//p[1:n] is the profits and w[1:n] is the weights of n objects such that p[i]/w[i] ≥ p[i+1]/w[i+1].
{
        for i= 1 to n do
                x[i] = 0.0;          // x[1:n] is the solution vector
        U = m;                       // m is the knapsack capacity
        for i=1 to n do
        {       if w[i] > U then
                        break;
                x[i] = 1.0
                U = U – w[i];
        }
        If i ≤ n then
               x[i] = U / w[i];
}
```

- **Time Complexity**
  - The for loop will execute maximum n times. So the time complexity = **O(n)**

17.

a) Write Floyd Warshall algorithm for all pair shortest path problem and perform its complexity analysis

**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.

b) Define Travelling Salesman Problem. Solve the following instance of TSP using branch and bound technique. The cost matrix is given below:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | - | 10 | 8 | 9 | 7 |
| B | 10 | - | 10 | 5 | 6 |
| C | 8 | 10 | - | 8 | 9 |
| D | 9 | 5 | 8 | - | 6 |
| E | 7 | 6 | 9 | 6 | - |

**Ans:**

The adjacency matrix is

$$\begin{bmatrix} \alpha & 10 & 8 & 9 & 7 \\ 10 & \alpha & 10 & 5 & 6 \\ 8 & 10 & \alpha & 8 & 9 \\ 9 & 5 & 8 & \alpha & 6 \\ 7 & 6 & 9 & 6 & \alpha \end{bmatrix}$$

Perform row reduction and column reduction

$$\begin{bmatrix} \alpha & 10 & 8 & 9 & 7 \\ 10 & \alpha & 10 & 5 & 6 \\ 8 & 10 & \alpha & 8 & 9 \\ 9 & 5 & 8 & \alpha & 6 \\ 7 & 6 & 9 & 6 & \alpha \end{bmatrix} \begin{matrix} 7 \\ 5 \\ 8 \\ 5 \\ 6 \end{matrix} \xrightarrow[\text{redn}]{\text{row}} \begin{array}{c} \begin{bmatrix} \alpha & 3 & 1 & 2 & 0 \\ 5 & \alpha & 5 & 0 & 1 \\ 0 & 2 & \alpha & 0 & 1 \\ 4 & 0 & 3 & \alpha & 1 \\ 1 & 0 & 3 & 0 & \alpha \end{bmatrix} \\ \begin{matrix} 0 & 0 & 1 & 0 & 0 \end{matrix} \end{array}$$

$$\xrightarrow[\text{redn}]{\text{column}} \underset{M_1}{\begin{bmatrix} \alpha & 3 & 0 & 2 & 0 \\ 5 & \alpha & 4 & 0 & 1 \\ 0 & 2 & \alpha & 0 & 1 \\ 4 & 0 & 2 & \alpha & 1 \\ 1 & 0 & 2 & 0 & \alpha \end{bmatrix}}$$

Total cost reduced = 31 + 1 = 32

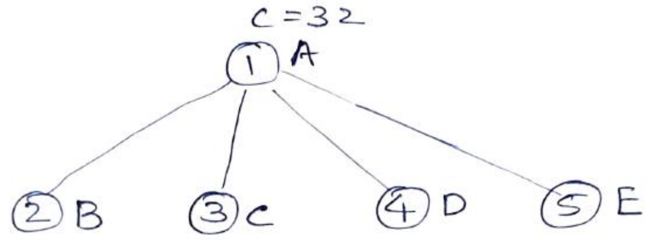
The state space tree is

$c=32$

① A

$M_1$ is the matrix for node 1

Generate the children of node 1



Find the matrix and cost of node 2

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

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 4 & 0 & 1 \\ 0 & \alpha & \alpha & 0 & 1 \\ 4 & \alpha & 2 & \alpha & 1 \\ 1 & \alpha & 2 & 0 & \alpha \end{bmatrix}$$

* Perform row reduction & colum redn

$$\left[\begin{array}{ccccc|c} \alpha & \alpha & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & 4 & 0 & 1 & 0 \\ 0 & \alpha & \alpha & 0 & 1 & 0 \\ 4 & \alpha & 2 & \alpha & 1 & 1 \\ 1 & \alpha & 2 & 0 & \alpha & 0 \end{array}\right] \xrightarrow[\text{redn}]{\text{row}} \begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 4 & 0 & 1 \\ 0 & \alpha & \alpha & 0 & 1 \\ 3 & \alpha & 1 & \alpha & 0 \\ 1 & \alpha & 2 & 0 & \alpha \end{bmatrix} \\ \begin{array}{ccccc} & & 1 & & \end{array} \end{array}$$

$$\xrightarrow[\text{reductn}]{\text{Column}} \underset{M_2}{\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 3 & 0 & 1 \\ 0 & \alpha & \alpha & 0 & 1 \\ 3 & \alpha & 0 & \alpha & 0 \\ 1 & \alpha & 1 & 0 & \alpha \end{bmatrix}}$$

M12

cost reduced = $\gamma = 2$

$M_2$ is the matrix of node 2

cost of node 2 = cost of node1 + $M_1[A,B]$ + $\gamma$

= 32 + 3 + 2 = 37

Find the matrix and cost of nod 3

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

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & \alpha & 0 & 1 \\ \alpha & 2 & \alpha & 0 & 1 \\ 4 & 0 & \alpha & \alpha & 1 \\ 1 & 0 & \alpha & 0 & \alpha \end{bmatrix}$$

* Perform row and column reduction

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & \alpha & 0 & 1 \\ \alpha & 2 & \alpha & 0 & 1 \\ 4 & 0 & \alpha & \alpha & 1 \\ 1 & 0 & \alpha & 0 & \alpha \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \end{array} \xrightarrow[\text{redn}]{\text{row}} \begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & \alpha & 0 & 1 \\ \alpha & 2 & \alpha & 0 & 1 \\ 4 & 0 & \alpha & \alpha & 1 \\ 1 & 0 & \alpha & 0 & \alpha \end{bmatrix} \\ \begin{matrix} 1 & 0 & 0 & 0 & 1 \end{matrix} \end{array}$$

$$\xrightarrow[\text{redn}]{\text{column}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & 0 & 0 \\ \alpha & 2 & \alpha & 0 & 0 \\ 3 & 0 & \alpha & \alpha & 0 \\ 0 & 0 & \alpha & 0 & \alpha \end{bmatrix}$$

$M_3$

cost reduced = 1

$M_3$ is the matrix for node 3.

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

= 32 + 0 + 1 = 33

Find the matrix and cost of node 4.

* Set row A and column D elmts are $\alpha$

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

* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & 4 & \alpha & 1 \\ 0 & 2 & \alpha & \alpha & 1 \\ \alpha & 0 & 2 & \alpha & 1 \\ 1 & 0 & 2 & \alpha & \alpha \end{bmatrix}$$

* Perform row reduction and column reductn

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & 4 & \alpha & 1 \\ 0 & 2 & \alpha & \alpha & 1 \\ \alpha & 0 & 2 & \alpha & 1 \\ 1 & 0 & 2 & \alpha & \alpha \end{bmatrix} 1 \xrightarrow[\text{redn}]{\text{row}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & 3 & \alpha & 0 \\ 0 & 2 & \alpha & \alpha & 1 \\ \alpha & 0 & 2 & \alpha & 1 \\ 1 & 0 & 2 & \alpha & \alpha \end{bmatrix}$$

2

$$\xrightarrow[\text{redn}]{\text{column}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & 1 & \alpha & 0 \\ 0 & 2 & \alpha & \alpha & 1 \\ \alpha & 0 & 0 & \alpha & 1 \\ 1 & 0 & 0 & \alpha & \alpha \end{bmatrix}$$

$M_4$

cost reduced = $\gamma$ = 3

$M_4$ is the matrix for node 4

cost of node 4 = cost of node 1 + $M_1[A,D]$ + $\gamma$

$= 32 + 2 + 3 = 37$

Find the matrix and cost of node 5

* set row A and column E elmts are $\alpha$
* set $M_1[E,A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & 4 & 0 & \alpha \\ 0 & 2 & \alpha & 0 & \alpha \\ 4 & 0 & 2 & \alpha & \alpha \\ \alpha & 0 & 2 & 0 & \alpha \end{bmatrix}$$

* Perform row and column reduction

$$\begin{array}{c|c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & 4 & 0 & \alpha \\ 0 & 2 & \alpha & 0 & \alpha \\ 4 & 0 & 2 & \alpha & \alpha \\ \alpha & 0 & 2 & 0 & \alpha \end{bmatrix} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 2 & 0 & 0 \end{matrix} & \end{array} \xrightarrow[\text{colum reddn}]{\text{row redn}} \underset{M_5}{\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 5 & \alpha & 2 & 0 & \alpha \\ 0 & 2 & \alpha & 0 & \alpha \\ 4 & 0 & 0 & \alpha & \alpha \\ \alpha & 0 & 0 & 0 & \alpha \end{bmatrix}}$$

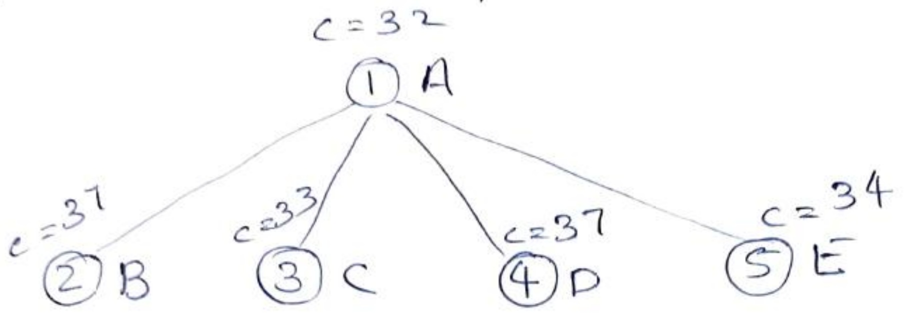
cost reduced $= \gamma = 2$

$M_5$ is the matrix for node 5

cost of node 5 = cost of node 1 + $M_1[A,E]$ + $\gamma$

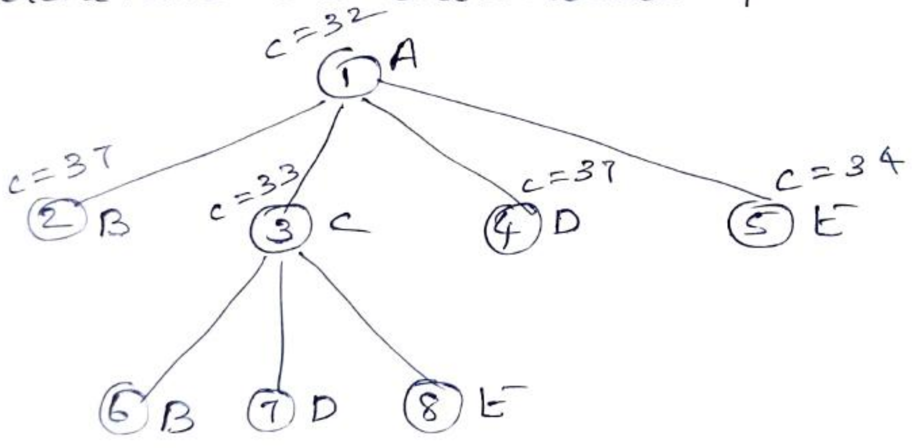
$= 32 + 0 + 2 = 34$

Now the state space tree is



Minimum cost live node is node 3

Generate the child nodes of node 3



Find the matrix and cost of node ~~B~~6

* Set row c and column ~~B~~ to $\alpha$.
* set $M_3[B,A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & 0 & \alpha \end{bmatrix}$$

* Perform row reduction & column reduction

$$\begin{array}{c}\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & 0 & \alpha\end{array}\right]\begin{array}{c}0\\0\\0\\0\\0\end{array} \\ \begin{array}{ccccc}0 & 0 & 0 & 0 & 0\end{array}\end{array} \xrightarrow[\text{redn}]{\text{row/colun}} \underset{M_6}{\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & 0 & \alpha\end{array}\right]}$$

cost reduced $= 0$

$M_6$ is the matrix of node 6

cost of node 6 = cost of node 3 + $M_3[C,B]$ + r

$= 33 + 2 + 1 = 36$

Find the matrix and cost of node 7

* Set row C and column D elmts are α
* Set $M_3[D,A] = \alpha$
* Now the matrix is

$$\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & 0 \\ 0 & 0 & \alpha & \alpha & \alpha\end{array}\right]$$

* Perform row and column reduction

$$\begin{array}{c} \left[\begin{array}{ccccc} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & 0 \\ 0 & 0 & \alpha & \alpha & \alpha \end{array}\right]\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \\ \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \end{array} \end{array} \xrightarrow[\text{redn}]{\text{row/colm}} \underset{M_7}{\left[\begin{array}{ccccc} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & 0 \\ 0 & 0 & \alpha & \alpha & \alpha \end{array}\right]}$$

cost reduced = 0

$M_7$ is the matrix of node 7

$$\text{cost of node } 7 = \text{cost of node } 3 + M_3[C,D] + \gamma$$

$$= 33 + 0 + 0 = 33$$

Find the matrix and cost of node 8

* set row C and column E elmts are $\alpha$
* set $M_3[E,A] = \alpha$
* Now the matrix is

$$\left[\begin{array}{ccccc} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & 0 & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & 0 & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & 0 & \alpha \end{array}\right]$$

* Perform row reduction & column redn

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & 0 & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & 0 & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & 0 & \alpha \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \xrightarrow[\text{rdn}]{\text{row}} \begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & 0 & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 3 & 0 & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & 0 & \alpha \end{bmatrix} \\ \begin{matrix} 3 & 0 & 0 & 0 & 0 \end{matrix} \end{array}$$

$$\xrightarrow[\text{redn}]{\text{column}} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 1 & \alpha & \alpha & 0 & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 0 & 0 & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & 0 & \alpha \end{bmatrix}_{M_8}$$

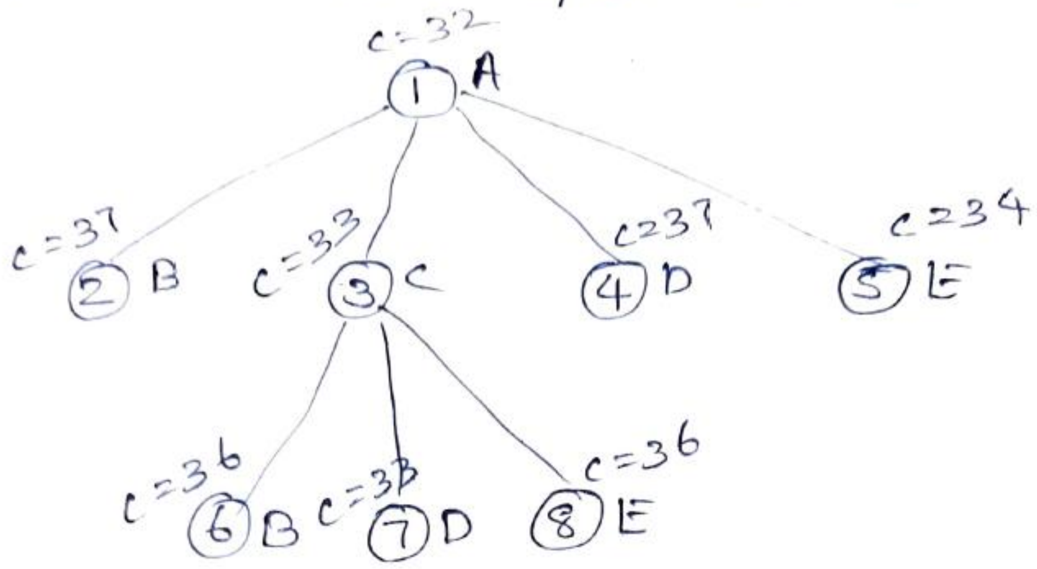
cost reduced $= r = 3$

$M_8$ is the matrix for node 8

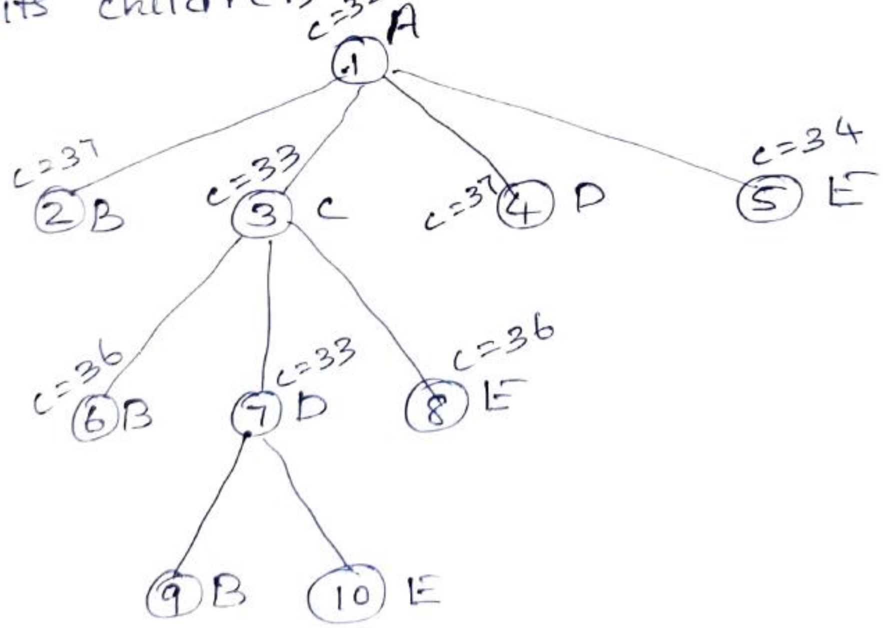
cost of node 8 = cost of node 3 + $M_3[C,E]$ + $r$

$= 33 + 0 + 3 = 36$

Now the state space tree is



Minimum cost live ~~d~~ node is node 7.
Now node 7 is the E-node. Generate its children



Find the matrix and cost of node 9.

* set row D and colum B elmts are α
* Set M7[B,A] = α
* Now the matrix is

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

* Perform row and column reduction

$$\begin{array}{c} \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha & \alpha \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \end{array} \xrightarrow[\text{redn}]{\text{row/colum}} \underset{M9}{\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha & \alpha \end{bmatrix}}$$

cost reduced $= r = 0$

M9 is the cost of node 9.

cost of node 9 = cost of node 7 + $M7[D,B] + r$

$= 33 + 0 + 0 = 33$

Find the matrix and cost of node 10

* set row D and column E elmts are $\alpha$
* set $M7[E,A] = \alpha$
* Now the matrix is

$$\begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & \alpha \end{bmatrix}$$

* Perform row & column reduction

$$\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ 4 & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & \alpha\end{array}\right]\begin{array}{c}0\\4\\0\\0\\0\end{array} \xrightarrow[\text{redn}]{\text{row}} \begin{array}{ccccc}\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & \alpha\end{array}\right] \\ \bullet \quad 0 \quad 0 \quad 0 \quad 0\end{array}$$

$$\xrightarrow[\text{redn}]{\text{colunn}} \underset{M_{10}}{\left[\begin{array}{ccccc}\alpha & \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & 0 & \alpha & \alpha & \alpha\end{array}\right]}$$

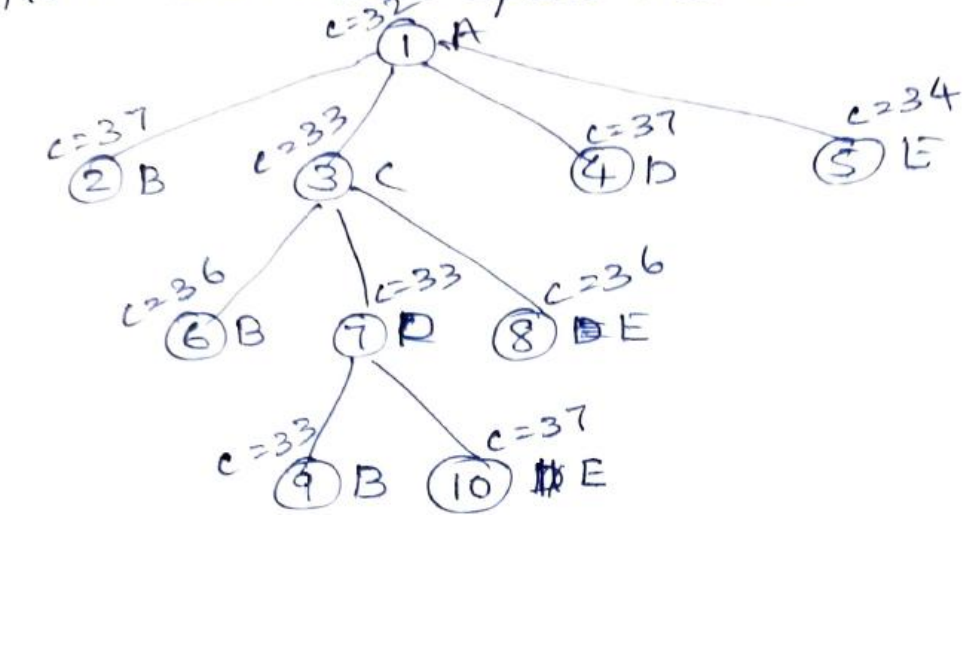
cost reduced = $r = 4$

$M_{10}$ is the matrix for node 10

cost of node 10 = cost of node D + $M_7[D,B]$ + $r$

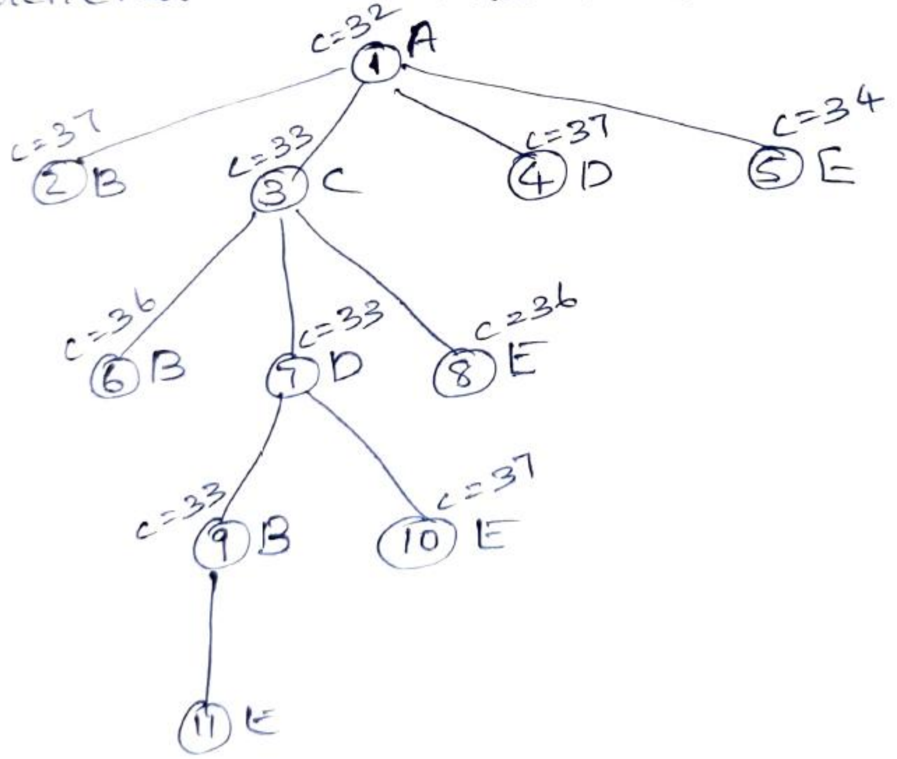
= $33 + 0 + 4 = 37$

Now the state space tree is



Minimum cost live node is node 9.

Generate its child node.



Find the matrix and cost of node 11

* set row B and col~~umn~~umn E elmts are $\alpha$
* set $M9[E,A]=\alpha$
* Now the matrix is

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

* Perform row & column reduction

$$\begin{array}{c} \left[\begin{array}{ccccc} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \end{array}\right]\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \\ \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \end{array} \end{array} \xrightarrow[\text{redn}]{\text{row/colum}} \underset{M_{11}}{\left[\begin{array}{ccccc} \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha & \alpha \end{array}\right]}$$

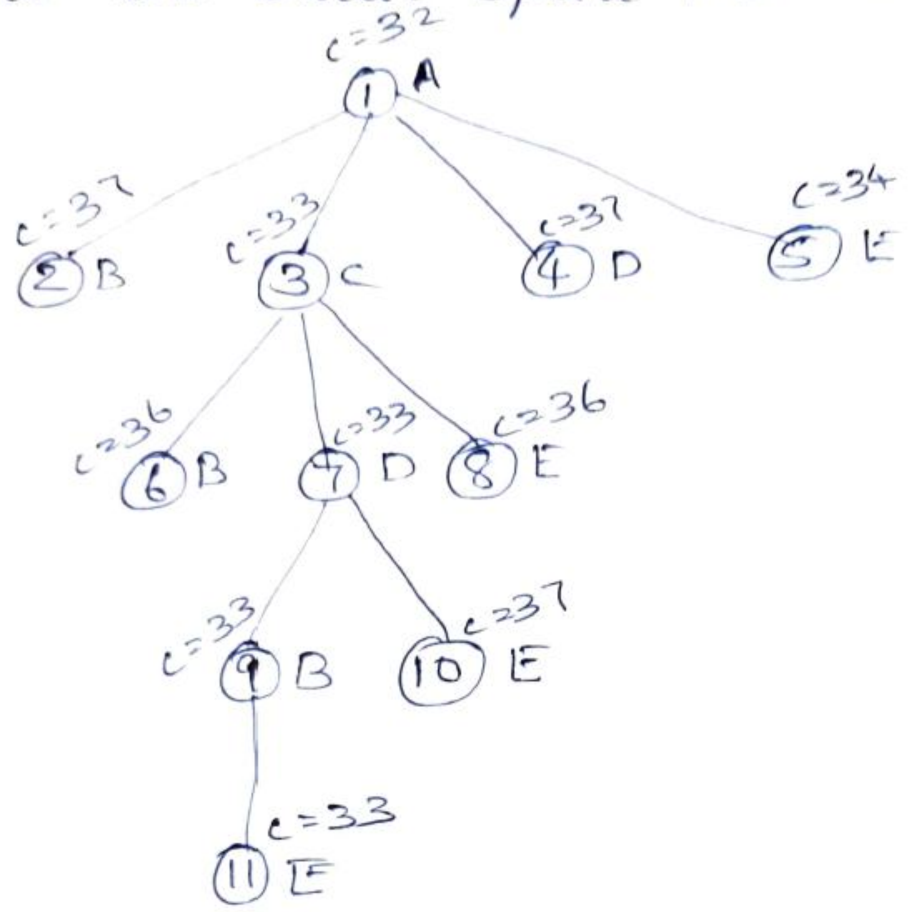
cost reduced = $r = 0$

$M_{11}$ is the matrix of node 11

cost of node 11 = cost of node 9 + $M_9[B,E]$ + $r$

$= 33 + 0 + 0 = 33$

Now the state space tree is



Minimum cost live node is node 11.

It has no child node.

$\therefore$ TSP path = $\underline{\underline{A-C-D-B-E-A}}$

TSP cost = cost of node 11 = $\underline{\underline{33}}$

18.

a) Consider the following four matrices and perform chain matrix multiplication using the dynamic programming approach. Finally give the optimal cost of multiplication and optimal parenthesization

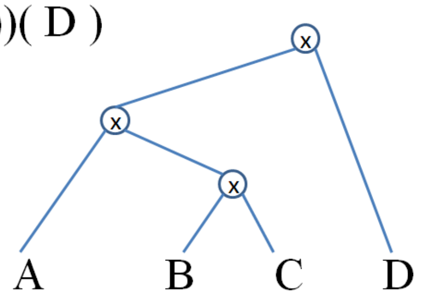
| A | B | C | D |
|---|---|---|---|
| 4x5 | 5x3 | 3x2 | 2x7 |

**Ans:**

| m | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 60 | 70 | 126 |
| 2 |  | 0 | 30 | 100 |
| 3 |  |  | 0 | 42 |
| 4 |  |  |  | 0 |

| s | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 |  | A | A | C |
| 2 |  |  | B | C |
| 3 |  |  |  | C |
| 4 |  |  |  |  |

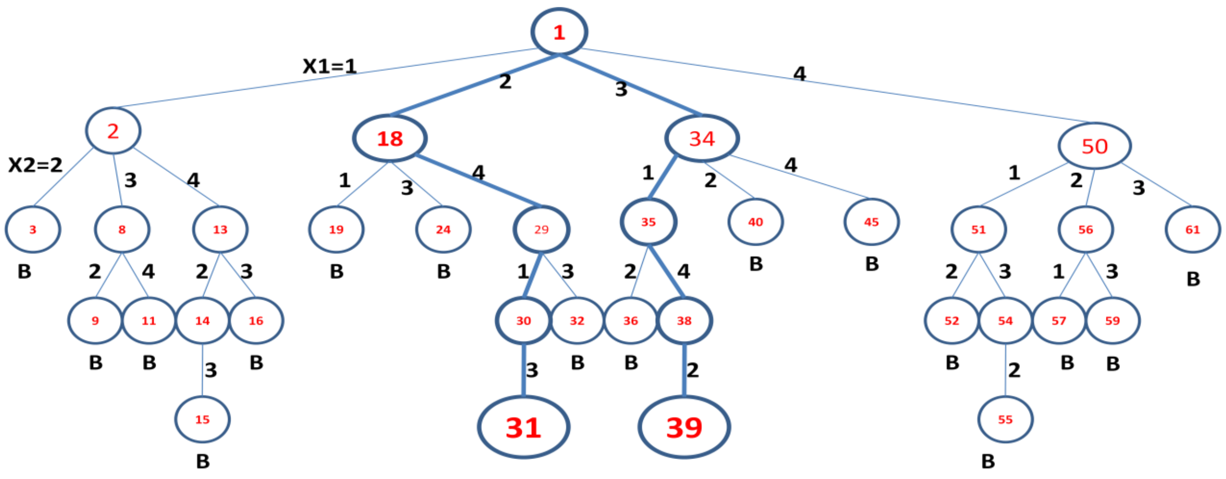
( (A) (B C ))( D )



b) Define n-queens problem. Draw the state space tree for 4-queens problem using backtracking method

**Ans:**

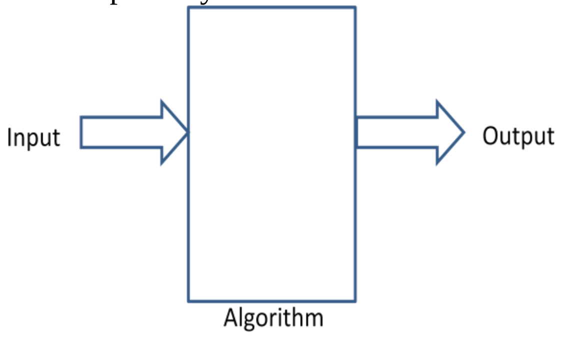
- **n-Queens Problem**
  - n queens are to be placed on a n x n chessboard so that no two attack. That is, no two queens are on the same row, column, or diagonal.


**State Space Tree of 4 Queens Problem**

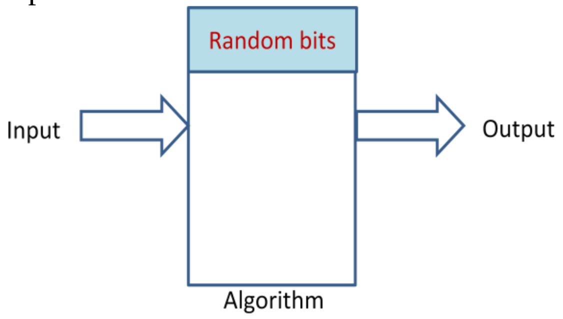
19.

a) Explain the benefits of randomized algorithm over deterministic algorithm. Discuss briefly the major categories of randomized algorithms. Give example for each category.
**Ans:**

- **Randomized Algorithm**
    - **Deterministic Algorithm**: The output as well as the running time are functions of the input only.



    - **Randomized Algorithm**: The output or the running time are functions of the input and random bits chosen



        - An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm

- Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms
- It hopes to achieve good performance in the "average case" over all possible choices of random bits.

- **Type of Randomized Algorithms**

    - **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
  - **Example1**: 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)**

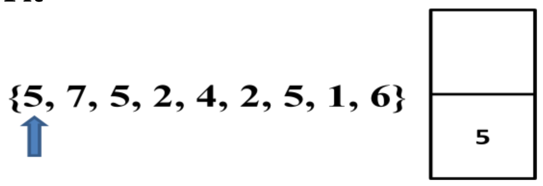
b) Define bin packing problem. Discuss the first fit strategy for solving it. State the approximation ratio of the algorithm
**Ans:**

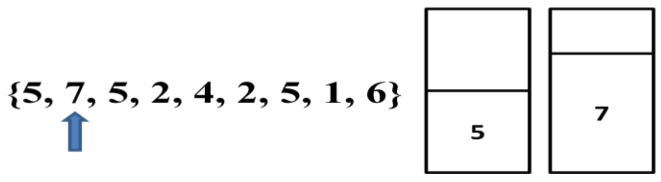
- **Bin packing problem**: Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity

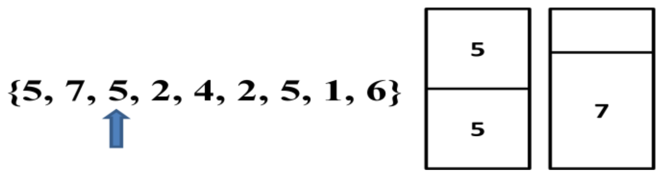
- **First Fit Algorithm**
  - Scan the previous bins in order and find the first bin that it fits.
  - If such bin exists, place the item in that bin
  - Otherwise use a new bin.
  - Time Complexity

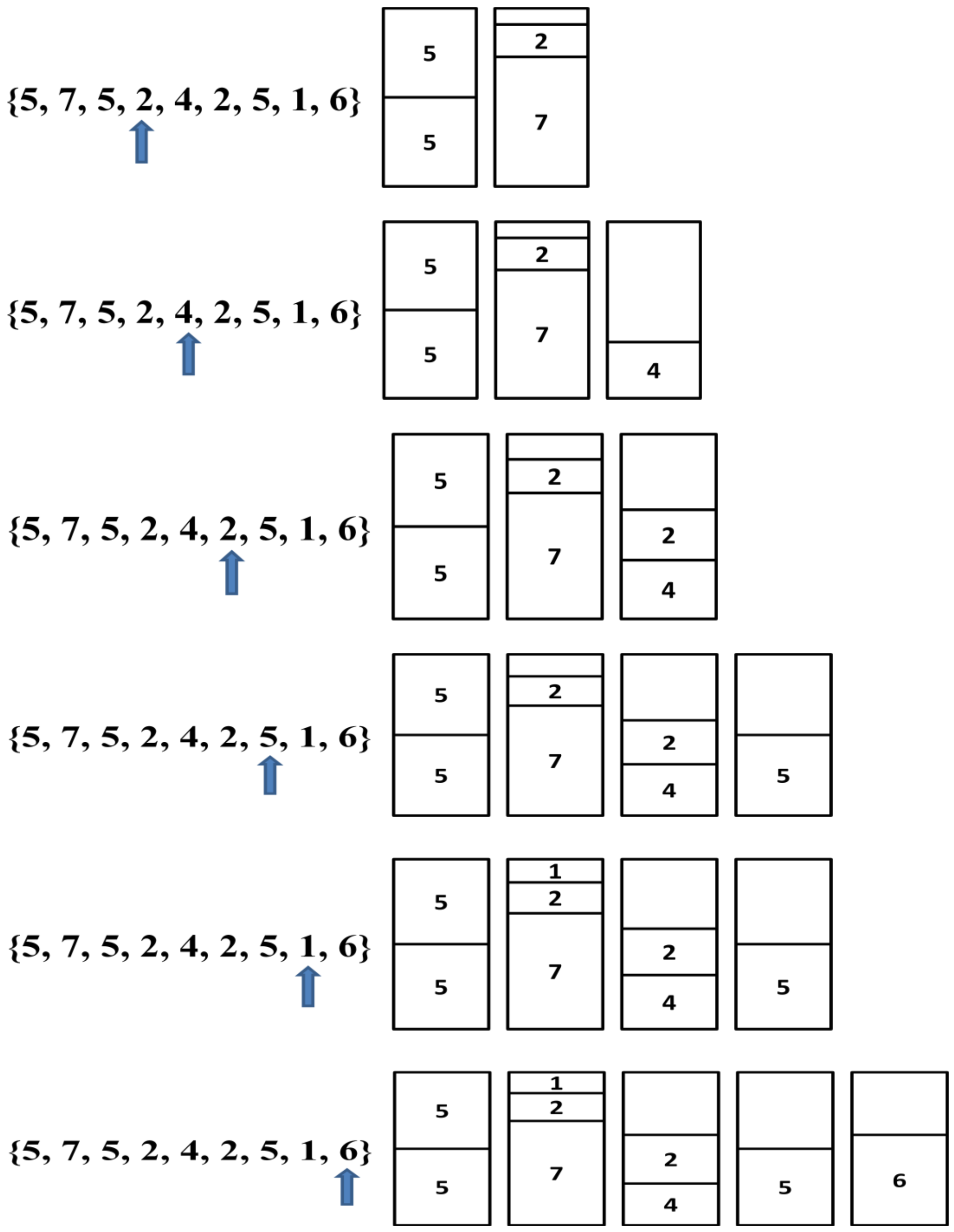
o Best case Time Complexity = $\theta(n \log n)$
o Average case Time Complexity = $\theta(n^2)$
o Worst case Time Complexity = $\theta(n^2)$

- **Example:** Apply different Bin packing approximation algorithms on the following items with bin capacity=10. Assuming the sizes of the items be {5, 7, 5, 2, 4, 2, 5, 1, 6}.
  - Minimum number of bins >= Ceil ((Total Weight) / (Bin Capacity))
    = Ceil (37 / 10) = 4
  - **First Fit**









Number of bins required = 5

20.

a) Give a randomized version of quicksort algorithm and perform its expected running time analysis.

**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)
```

**Number times while loop runs before finding a central pivot?**

- The probability that the randomly chosen element is central pivot is 1/n.
- Therefore, expected number of times the while loop runs is n.
- Thus, the expected time complexity of step 2 is O(n).

b) Prove that vertex cover problem is NP Complete.
**Ans:**

- The vertex Cover of a graph is defined as a subset of its vertices, such for every edge in the graph, from vertex **u** to **v,** at least one of them must be a part of the vertex cover set.
- Vertex cover problem is to find the minimum sized vertex cover of the given graph.

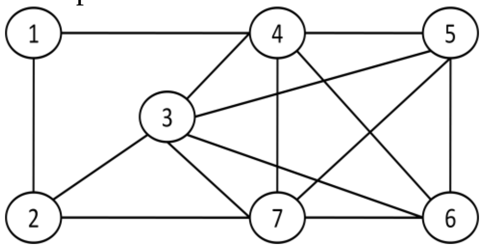
- **Steps to prove that Vertex Cover is a NP-Complete problem**

- Step 1:Write a polynomial time verification algorithm to prove that the given problem is NP
    - **Inputs:** <G, k,V'>
    - **Verifier Algorithm**:
        1. count = 0
        2. for each vertex v in V' remove all edges adjacent to v from set E
            a. increment count by 1
        3. if count = k and E is empty then the given solution is correct
        4. else the given solution is wrong
    - This algorithm will execute in polynomial time. Therefore **VERTEX COVER problem is a NP problem.**

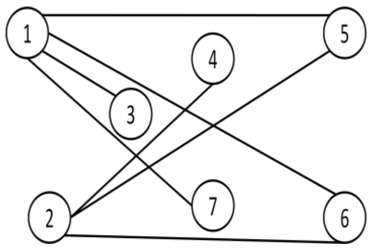
- Step 2: Write a polynomial time reduction algorithm from CLIQUE problem to VERTEX COVER problem

- **Algorithm**
  **Inputs: <G=(V,E), k>**
  1. Construct a graph G’, which is the complement of Graph G
  2. If G’ has a vertex cover of size |V| - k, then G has a clique of size k.

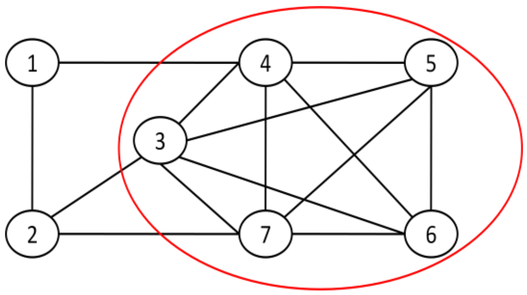
- Example:

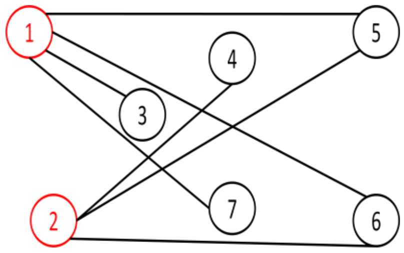


G=(V, E)



G’=(V, E’)





Vertex cover of G’ is {1,2}
Size of vertex cover of G’ is 2.
If so G has a clique of size |V| - 2 = 5

- This reduction algorithm(CLIQUE to VERTEX COVER) is a polynomial time algorithm
- So **CLIQUE problem is NP Hard**.

• Conclusion

- VERTEX COVER problem is NP and NP-Hard.
- So it is NP-Complete