APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

B.Tech Degree S6 (S, FE) / S6 (PT) (S) Examination January 2024 (2019 Scheme)

Course Code: CST306

Course Name: ALGORITHM ANALYSIS AND DESIGN

PART A

1. What are the characteristics of a good algorithm?

   Ans:

   - **Input:** Zero or more inputs are externally supplied.
   - **Output:** At least one output is produced.
   - **Definiteness:** Each instruction is clear and unambiguous.
     - "add 6 or 7 to x" , "compute 5/0" etc. are not permitted.
   - **Finiteness:** The algorithm terminates after a finite number of steps.
   - **Effectiveness:** Every instruction must be very basic so that it can be carried out by a person using only pencil and paper in a finite amount of time. It also must be feasible.

2. Solve $T(n)=4T(n/2) + n^3$ using master method

   Ans:

   $T(n)=4T(n/2) + n^3=4T(n/2) + \theta(n^3 \log^0(n))$

   a=4 b=2 k=3 p=0

   Here $a<b^k$ and $p\geq 0$, then $T(n)= \theta(n^k \log^p(n))= \theta(n^3 \log^0(n))=\boldsymbol{\theta(n^3)}$

3. Can we use DFS to detect cycles in a graph? Justify your answer

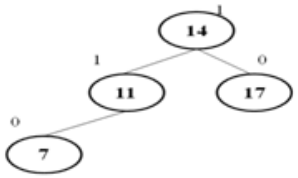
   Ans:

   Yes. After performing DFS, Check whether any back edge is present in the graph. Presence of back edge indicates a cycle in a graph.

4. Define AVL tree. What is the advantage of AVL tree? Give an example

   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.
     - Example:



- **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)

5. Strassen's multiplication method is used to multiply two nxn matrices when n is a power of 2. How it can be modified when n is not a power of 2?

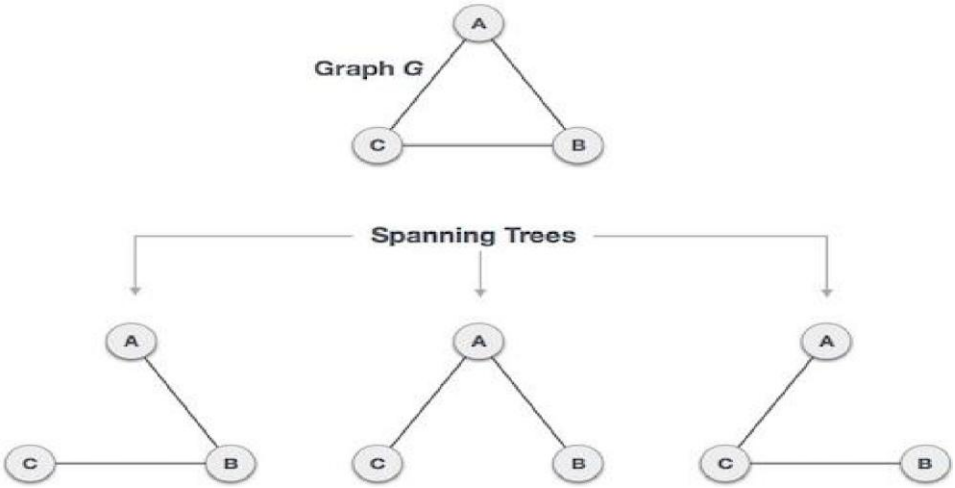
   Ans:

   If n is not a power of 2, then enough rows and columns of 0's can be added to both matrices so that the resulting dimensions are the power of two.

6. Define spanning tree of a graph. Write the total number of spanning trees possible for a complete graph with 4 vertices

   Ans:

   - A spanning tree is a subset of undirected connected Graph G=(V,E), which has all the vertices covered with minimum possible number of edges.



Total number of spanning trees possible for a complete graph with 4 vertices

$= \mathbf{n^{n-2} = 4^{4-2} = 4^2 = 16}$

7. Write a recurrence to represent the number of ways to parenthesize a chain of n matrices.

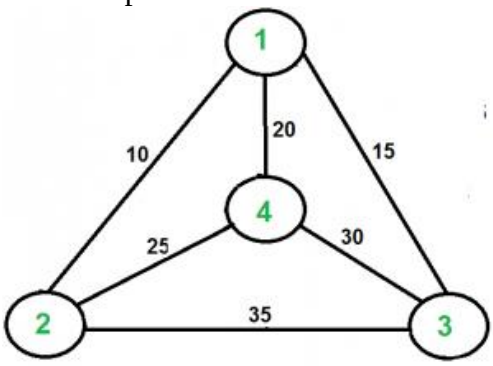
Ans:

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases}$$

8. Define Travelling Salesman problem

Ans:

- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.
- Example:



TSP route is 1-2-4-3-1
TSP Cost = 10+ 25 + 30 + 15 = 80

9. What do you mean by tractable problem? Give an example.

Ans:

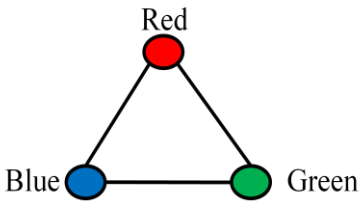
When the complexity is expressed as some polynomial function over input size, then the concerned problem is tractable.When the complexity is expressed as some exponential function over input size, then the concerned problem is intractable.
Tractable problem solutions are implemented in practice. They have polynomial time complexity.
Example of tractable problem

- PATH problem: Given directed graph G, determine whether a directed path exists from vertex s to vertex t.
  - Time complexity = O(n), where n is the total number of vertices

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) Solve the following recurrence using iteration method
T(n)=2T(n/2) + n

Ans:

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

= n + 2 [(n/2)+ 2 T(n/2^2)] = 2n + 2^2T(n/2^2)

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

…….

=k n + 2^kT(n/2^k) k^{th} term

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

T(n) =n$\log_2$ (n) + nT(1)

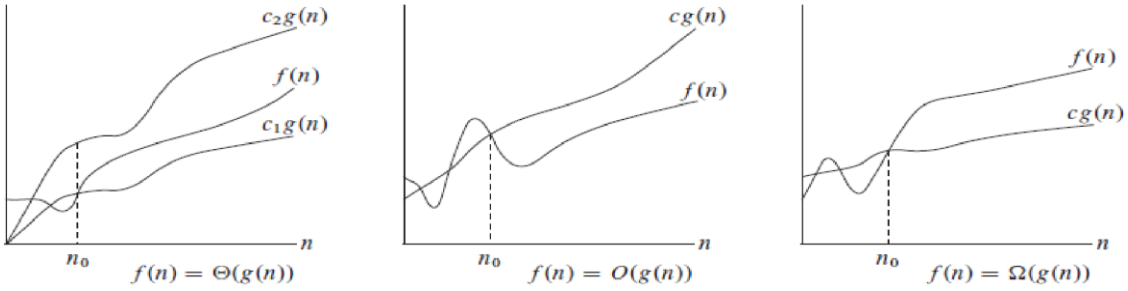
=n$\log_2$ (n) + n

=**O(n$\log_2$(n))**

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

Ans:

- Asymptotic notationsare the mathematical notations to represent frequency count. Few asymptotic notations are
- **Big Oh (O)**
  - The function f(n) = O(g(n)) iff there exists 2 positive constants c and n_0 such that $0 \le f(n) \le c\ g(n)$ for all $n \ge n_0$
  - 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 $f(n) \ge c\ g(n) \ge 0$ for all $n \ge n_0$
  - 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) = Θ (g(n)) iff there exists 3 positive constants c_1, c_2 and n_0 such that $0 \le c_1\, g(n) \le f(n) \le c_2\, g(n)$ for all $n \ge n_0$
  - It is the measure of average amount of time taken by an algorithm(Average case).
  - 

- **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

12.

a) Illustrate best case, average case and worst-case complexity with insertion sort algorithm

Ans:

```
Algorithm InsertionSort(A,n)
{
        for i=1 to n-1 do
        {
                j=i
                while j>0 and A[j-1] > A[j] do
                {
                        Swap(A[j], A[j-1])
                        j=j-1
                }
        }
}
```

Best case complexity: Best case situation occurs when the array itself is in sorted order. In this case the while loop will not execute successfully. So the time complexity is proportional to number of times the for loop will execute. It will execute O(n) time.

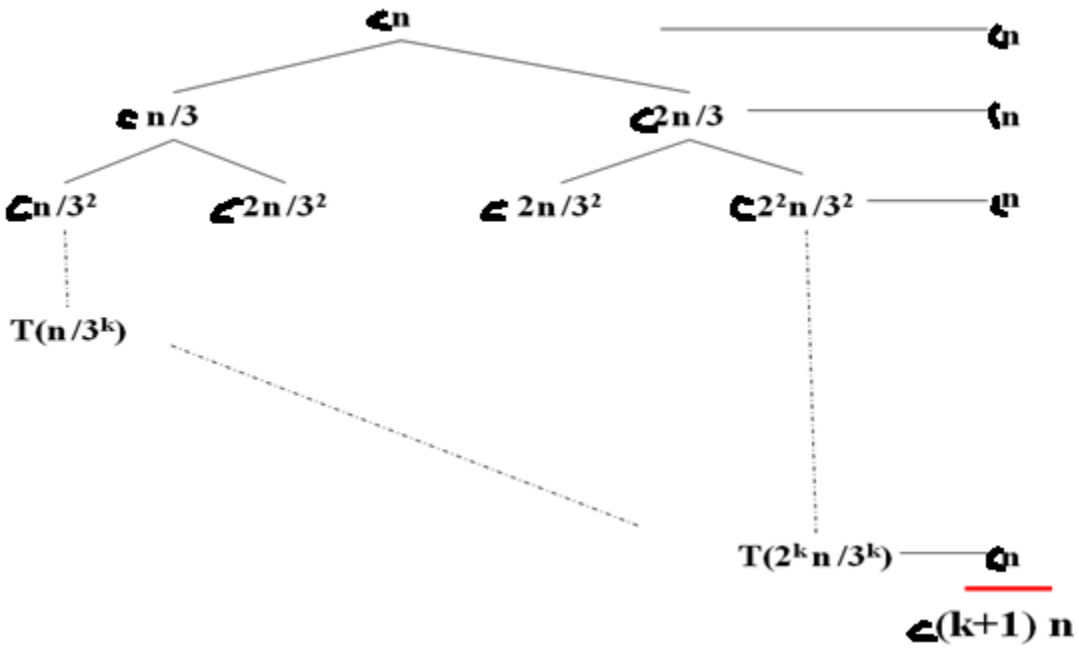
The best case time complexity = **Ω(n)**

Worst case complexity: Worst case situation occurs when the array itself is in reverse sorted order. In this case the while loop will execute 1+2+3+ . .. .+(n-1)=n(n+1)/2 times.
The time complexity is proportional to number of times the while loop will execute. It will execute $O(n^2)$ time.
The worst case time complexity = **$O(n^2)$**

Average case complexity: Average case situation occurs when the while loop will iterate half of its maximum iterations. In this case the while loop will execute [1+2+3+ . .. .+(n-1)]/2=n(n+1)/4 times.
The time complexity is proportional to number of times the while loop will execute. It will execute $O(n^2)$ time.
The average case time complexity = **$\Theta(n^2)$**

b) Solve the following recurrence using recursion tree method
T(n)=T(n/3) + T(2n/3)+cn

Ans:



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

T(n) = c(k+1) n
= $(\log_{(3/2)} n + 1)c\ n$
= $cn \log_{(3/2)} n + cn$
= **$O(n \log_{(3/2)} n)$**

13.

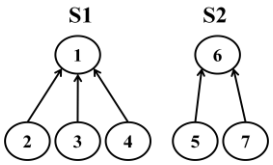
a) Explain the different operations possible on disjoint sets. Implement UNION using linked list representation of disjoint sets.

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.



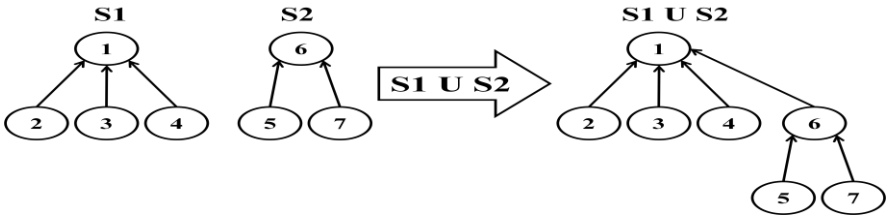
Find(3) will return 1, which is the root of the tree that 3 belongs
Find(6) will return 6, which is the root of the tree that 6 belongs

  - **Find Algorithm**

Algorithm Find(n)
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

b) Give Breadth First Search algorithm for graph traversal. 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 = $\mathbf{O(V^2)}$

14.

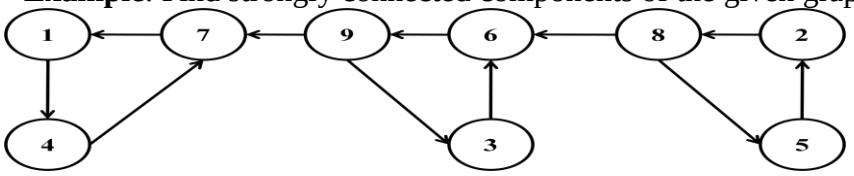
a) Write an algorithm to find strongly connected components of a graph. Illustrate with an example

Ans:

Algorithm

1. Set all vertices of graph G are unvisited.
2. Create an empty stack S.
3. Do DFS traversal on unvisited vertices and set it as visited. If a vertex has no unvisited neighbor, push it in to the stack.
4. Perform the above step until all vertices are visited
5. Reverse the graph G.
6. Set all nodes are unvisited.
7. While S is not Empty
   - 7.1.1 POP one vertex v'
   - 7.1.2 If v' is not visited
     - 7.1.2.1 Set v' as visited
     - 7.1.2.2 Call DFS(v'). It will print strongly connected component of v'.

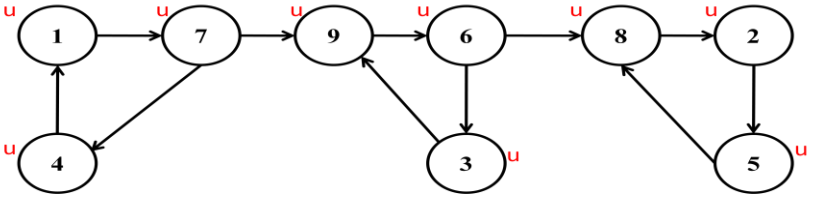
Example: Find strongly connected components of the given graph



Answer:

Pass 1: DFS: 1,4,7,9,3,6,8,5,2

Reverse the graph



Perform DFS

Connected Components: **8-2-5** **9-6-3** **1-7-4**

b) Give Depth First Search algorithm for graph traversal. How the edges of a graph are classified based on DFS?

Ans:

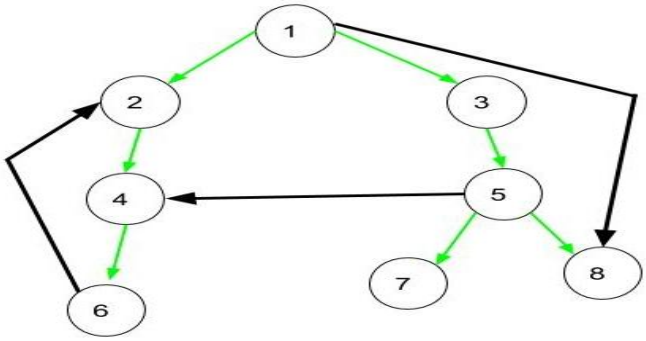
Algorithm DFS(G, u)

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)

- **Classification of Edges**



  - The DFS traversal of the above graph is 1 2 4 6 3 5 7 8
  - **Tree Edge:** It is a edge in tree obtained after applying DFS on the graph
    - Eg: (1,2), (2,4), (4,6), (1,3), (3,5), (5,7) and (5,8)
  - **Forward Edge:**
    - It is an edge (u, v) such that v is descendant but not part of the DFS tree
    - Eg: (1,8)
  - **Backward Edge**
    - It is an edge (u, v) such that v is ancestor of edge u but not part of DFS tree
    - Eg: (6,2)
  - **Cross Edge**
    - It is a edge which connects two node such that they do not have any ancestor and a descendant relationship between them.
    - Eg: (5,4)

15.

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

Ans:

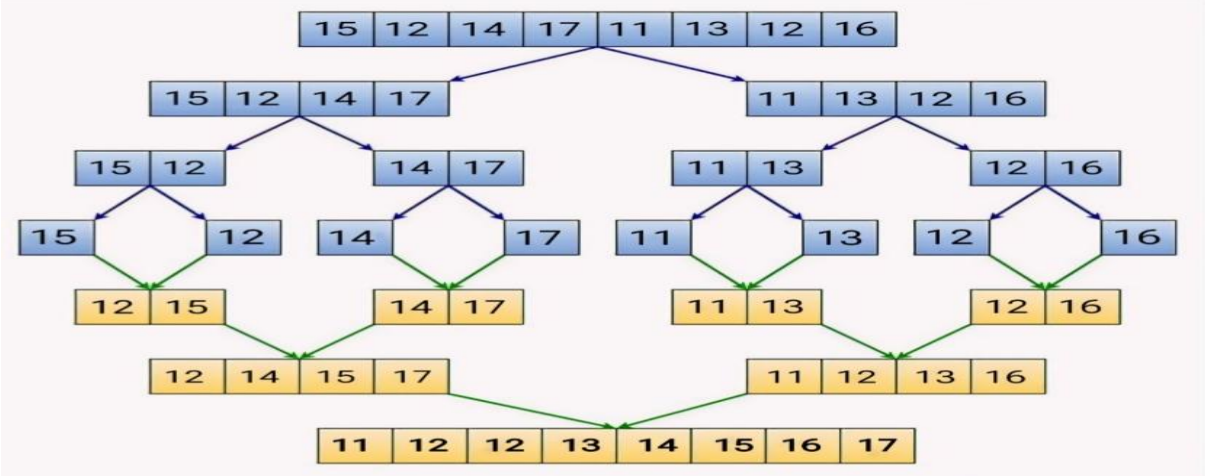
```
Greedy(a, n)   //a[1..n] contains n inputs
{       solution = Φ;
        for i=1 to n do
        {       x = Select(a);
                if Feasible(solution, x) then
                        solution = Union(solution, x);
        }
        return solution;
}
```

- 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.

```
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];
}
```

b) Illustrate the divide and conquer approach by applying 2 way merge sort for the input array: [15,12,14,17,11,13,12,16]. Write the recurrence for merge sort and give the complexity.

Ans:



Recurrence Relation of Merge Sort

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

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 $2^k = n \Rightarrow k=\log n$]
$= n\,T(1) + c\,n \log n$
$= a\,n + c\,n \log n$
$=$ **O(n log n)**

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

16.

a) Find an optimal solution to the fractional knapsack instance n=5, m=60, (p1,p2,.....p5) = (30,20,100,90,160) and (w1,w2,.....w5) = (5,10,20,30,40), Where m is the knapsack capacity, p_i is the profit of i^{th} item and w_i is the weight of i^{th} item

Ans:

- Arrange the objects in the descending order of profit/weight

i ={ 1, 2, 3, 4, 5 }
P ={ 30, 20, 100, 90, 160 }
W ={ 5, 10, 20, 30, 40 }

Pi/Wi ={ 6, 2, 5, 3, 4 }

Now the i, P and W arrays are

i ={ 1, 3, 5, 4, 2 }
P ={ 30, 100, 160, 90, 20 }
W ={ 5, 20, 40, 30, 10 }

Initially U=m=60

Item	Pi	Wi	Xi	U = U-Wi
1	30	5	1	55
3	100	20	1	35
5	160	40	35/40 = 7/8	0
4	90	30	0	0
2	20	10	0	0

- Total weight of the chosen objects are
  $\sum_{i=1}^{n} WiXi$ = 5x1 + 10x0 + 20x1 + 30x0 +40x7/8 = 60
- Profit earned is $\sum_{i=1}^{n} PiXi$ = 30x1 + 20x0 + 100x1 + 90x0 + 160x7/8 = **270**
- Solution vector **X={1, 0, 1, 0, 7/8}**

b) 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 $O(E + V^2)$ = **$O(V^2)$**
  - Graph represented using adjacency list can be reduced to **O(E log V)** with the help of binary heap.

17.

a) Find an optimal paranthesization of a matrix-chain product whose sequence of dimensions is 4x10,10x3,3x12,12x20,20x7 using dynamic programming

Ans:

The values of m and s tables are computed as follows:

m[1][1] = 0, m[2][2]=0, m[3][3]=0, m[4][4]=0, m[5][5]=0

m[1][2] = 4*10*3 =120, m[2][3] = 360, m[3][4] =720, m[4][5]=1680

m[1][3] = min{m[1][1]+m[2][3]+4*10*12 , m[1][2]+m[3][3]+4*3*12}=264

m[2][4] = 1320, m[3][5] = 1140

m[1][4] = 1080, m[2][5] = 1350, m[1][5] = 1344

s[1][2] = 1, s[2][3] =2, s[3][4]=3, s[4][5] =4, s[1][3]=2, s[2][4]= 2, s[3][5]=4, s[1][4]=2, s[2][5]=2, s[1][5]=2

m	1	2	3	4	5
1	0	120	264	1080	1344
2		0	360	1320	1350
3			0	720	1140
4				0	1680
5					0

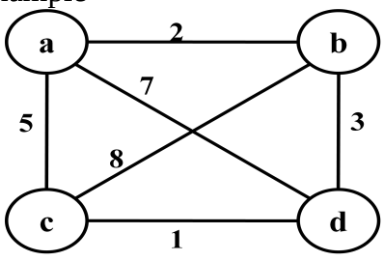
s	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	4
4					4
5					

$$(A_1\ A_2)\ ((A_3\ A_4)\ A_5)$$

b) Explain how Travelling Salesman Problem can be solved using Branch and Bound method

Ans:

- **Travelling Salesman Problem**
  - Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.
  - Example



The adjacency matrix is

	a	b	c	d
a	α	2	5	7
b	2	α	8	3
c	5	8	α	1
d	7	3	1	α

Perform row reduction, then column reduction

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

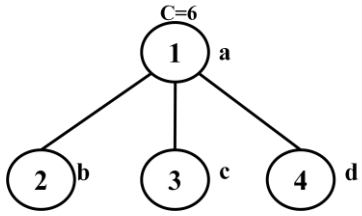
Row reduction → Column reduction → M_1

Total cost reduced = 2+2+1+1+0+0+0+0 = 6

The state space tree is

M_1 is the matrix for node 1 is

Generate the child node of node 1



Find the matrix and cost of node 2

- Set row a and column b elements are α
- Set $M_1[b, a]= \alpha$
- The resultant matrix is

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 6 & 1 \\ 4 & \alpha & \alpha & 0 \\ 6 & \alpha & 0 & \alpha \end{pmatrix}$$

Perform row reduction, then column reduction

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 6 & 1 \\ 4 & \alpha & \alpha & 0 \\ 6 & \alpha & 0 & \alpha \end{pmatrix} \begin{matrix} 0 \\ -1 \\ 0 \\ 0 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 5 & 0 \\ 4 & \alpha & \alpha & 0 \\ 6 & \alpha & 0 & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 5 & 0 \\ 0 & \alpha & \alpha & 0 \\ 2 & \alpha & 0 & \alpha \end{pmatrix} = M_2$$

$$\begin{matrix} & -4 & 0 & 0 & 0 \end{matrix}$$

Cost reduced = r = 5

M_2 is the matrix for node 2

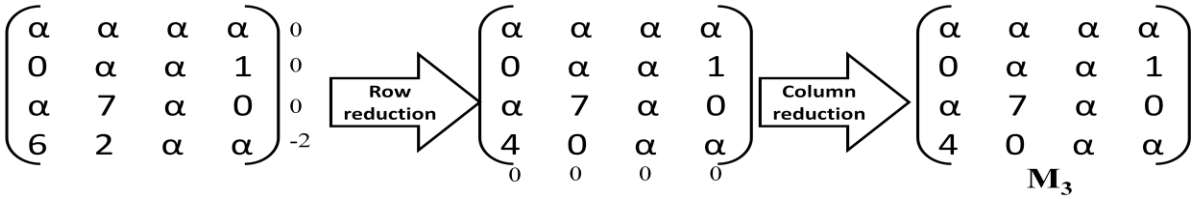
Cost of node 2 = Cost of node 1 + $M_1[a, b]$ + r = 6 + 0 + 5 = 11

Find the matrix and cost of node 3

- Set row a and column c elements are α
- Set $M_1[c, a]= \alpha$
- The resultant matrix is

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 1 \\ \alpha & 7 & \alpha & 0 \\ 6 & 2 & \alpha & \alpha \end{pmatrix}$$

Perform row reduction, then column reduction



Cost reduced = r = 2

M_3 is the matrix for node 3

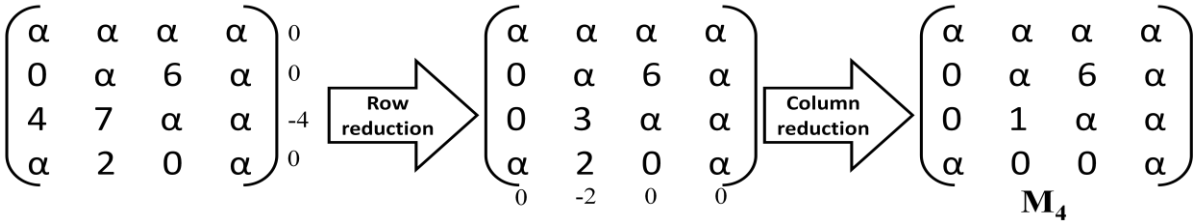
Cost of node 3 = Cost of node 1 + M_1[a, c] + r = 6 + 3 + 2 = 11

Find the matrix and cost of node 4

- Set row a and column d elements are α
- Set M_1[d, a]= α
- The resultant matrix is

α	α	α	α
0	α	6	α
4	7	α	α
α	2	0	α

Perform row reduction, then column reduction

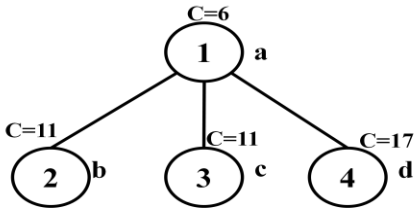


Cost reduced = r = 6

M_4 is the matrix for node 4

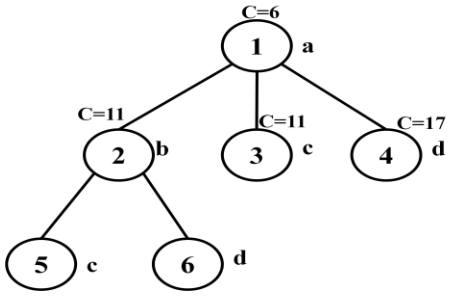
Cost of node 4 = Cost of node 1 + M_1[a, d] + r = 6 + 5 + 6 = 17

Now the state space tree is



Now the live nodes are 2, 3 and 4. Minimum cost is for node 2 and 3. Choose one node(say node 2) as the next E-node.

Generate the child node of node 2



Find the matrix and cost of node 5

- Set row b and column c elements are α
- Set $M_2[c, a]= \alpha$
- The resultant matrix is

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 2 & \alpha & \alpha & \alpha \end{pmatrix}$$

Perform row reduction, then column reduction

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 2 & \alpha & \alpha & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ -2 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{pmatrix}$$

$0 \quad 0 \quad 0 \quad 0$

M_5

Cost reduced = r = 2

M_5 is the matrix for node 5

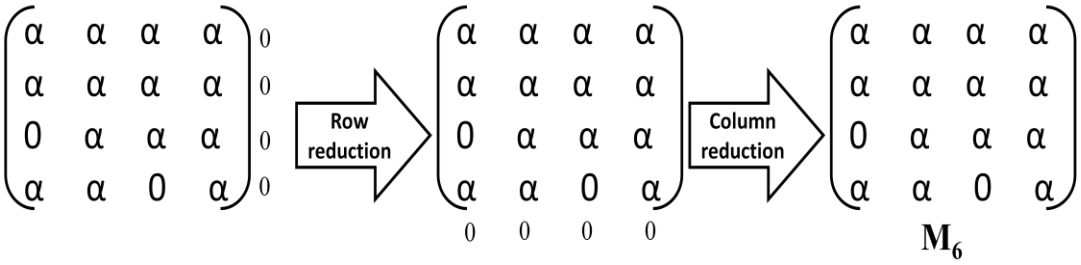
Cost of node 5 = Cost of node 2 + $M_2[b, c]$ + r = 11 + 5 + 2 = 18

Find the matrix and cost of node 6

- Set row b and column d elements are α
- Set $M_2[d, a]= \alpha$
- The resultant matrix is

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

Perform row reduction, then column reduction

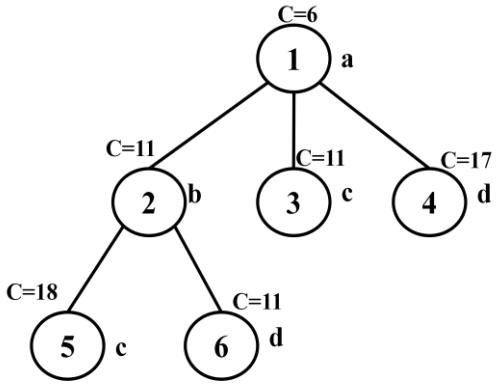


Cost reduced = r = 0

M_6 is the matrix for node 6

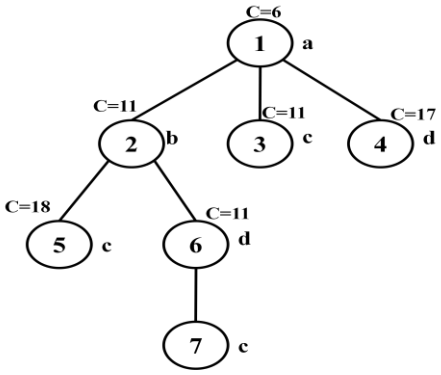
Cost of node 6 = Cost of node 2 + M_2[b, d] + r = 11 + 0 + 0 = 11

Now the state space tree is



Now the live nodes are 5, 6, 3 and 4. Choose one node which having minimum cost(say node 6) as the next E-node.

Generate the child node of node 6

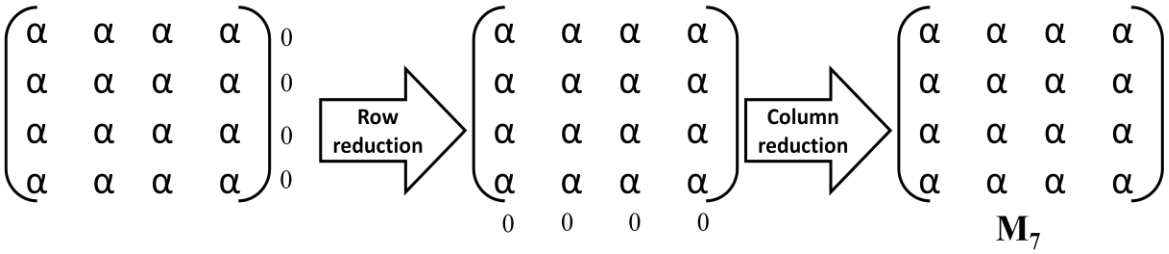


Find the matrix and cost of node 7

- Set row d and column c elements are α
- Set M_6[c, a]= α
- The resultant matrix is

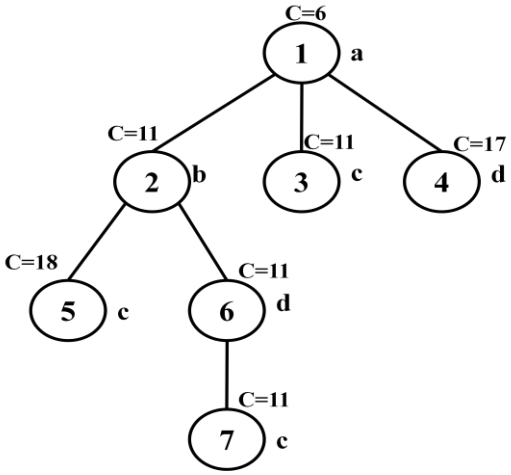
$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{pmatrix}$$

Perform row reduction, then column reduction



Cost reduced = r = 0
M_7 is the matrix for node 7
Cost of node 7 = Cost of node 6 + M_6[d, c] + r = 11 + 0 + 0 = 11

Now the state space tree is



Now the live nodes are 5, 7, 3 and 4. Choose one node which having minimum cost(say node 7) as the next E-node. Node 7 having no child node. Now we can say that node 1 to node 7 path is the Traveling salesperson path.

The TSP path = a-b-d-c-a
The TSP cost = 11

18.

a) Explain backtracking technique. How 4-queens problem can be solved using backtracking?

Ans:

Backtracking Control Abstraction

- $(x_1, x_2.,\ldots\ldots x_i)$ be a path from the root to a node in a state space tree.
- **Generating Function** $T(x_1, x_2.,\ldots\ldots x_i)$ be the set of all possible values for x_{i+1} such that $(x_1, x_2.,\ldots\ldots x_{i+1})$ is also a path to a problem state.
- $T(x_1, x_2.,\ldots\ldots x_n) = \varphi$
- **Bounding function** $B_{i+1}(x_1, x_2.,\ldots\ldots x_{i+1})$ is false for a path $(x_1, x_2.,\ldots\ldots x_{i+1})$ from the root node to a problem state, then the path cannot be extended to reach an answer node.
- Thus the candidates for position i+1of the solution vector $(x_1, x_2.,\ldots\ldots x_n)$ are those values which are generated by T and satisfy $B_{i+1.}$
- The recursive version is initially invoked by **Backtrack(1)**.

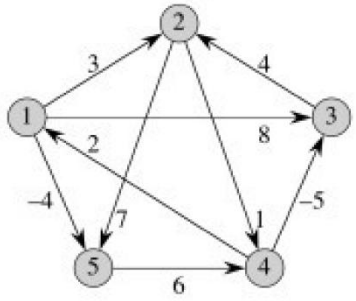
```
Algorithm Backtrack(k)
{       for (each x[k] ϵ T(x[1], . . . .x[k-1])
        {
            if(Bk(x[1], x[2], . . . . . ., x[k]) != 0) then
            {
                if(x[1], x[2], . . . . . ., x[k] is a path to an answer node)
                        then write(x[1:k])
                if(k<n) then Backtrack(k+1)
            }
        }
}
```

N-Queens Problem

```
Algorithm NQueens(k,n)
{
        for i=1 to n do
        {       if Place(k,i) then
                {       x[k] = i
                        if(k==n) then write(x[1:n])
                        else    NQueens(k+1, n)
                }
        }
}
Algorithm Place(k,i)
{
        for j=1 to k-1 do
        {
                if( (x[j]==i) or ( Abs(j-k)==Abs(x[j]-i) )  then
                        Return false
        }
        Return true
}
```

- **Place(k,i)** returns true if the k^{th} queen can be placed in column i.
  - i should be distinct from all previous values x[1],x[2], ..., x[k-1]
  - And no 2 queens are to be on the same diagonal
  - **Time complexity** of Place() is **O(k)**
- **NQueen()** is initially invoked by NQueen(1,n)
  - Time complexity of NQueen() is **O(n!)**

b) Explain Floyd-Warshall Algorithm. Using the algorithm find all pair of shortest paths in the following graph.



Ans:

$$\mathbf{D^0} = \begin{pmatrix} 0 & 3 & 8 & \alpha & -4 \\ \alpha & 0 & \alpha & 1 & 7 \\ \alpha & 4 & 0 & \alpha & \alpha \\ 2 & \alpha & -5 & 0 & \alpha \\ \alpha & \alpha & \alpha & 6 & 0 \end{pmatrix}$$

$$\mathbf{D^1} = \begin{pmatrix} 0 & 3 & 8 & \alpha & -4 \\ \alpha & 0 & \alpha & 1 & 7 \\ \alpha & 4 & 0 & \alpha & \alpha \\ 2 & 5 & -5 & 0 & -2 \\ \alpha & \alpha & \alpha & 6 & 0 \end{pmatrix}$$

$$\mathbf{D^2} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \alpha & 0 & \alpha & 1 & 7 \\ \alpha & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \alpha & \alpha & \alpha & 6 & 0 \end{pmatrix}$$

$$D^3 = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \alpha & 0 & \alpha & 1 & 7 \\ \alpha & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \alpha & \alpha & \alpha & 6 & 0 \end{pmatrix}$$

$$D^4 = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^5 = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

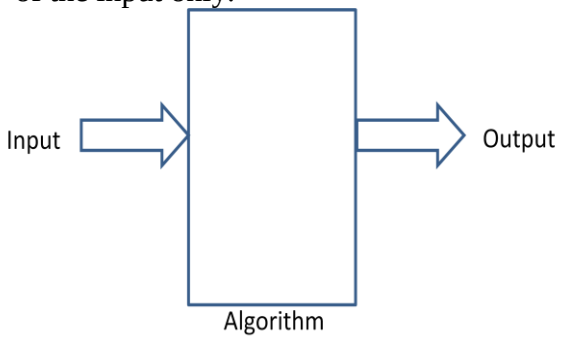
$D^{(5)}$ represents the shortest distance between each pair of the given graph

19.

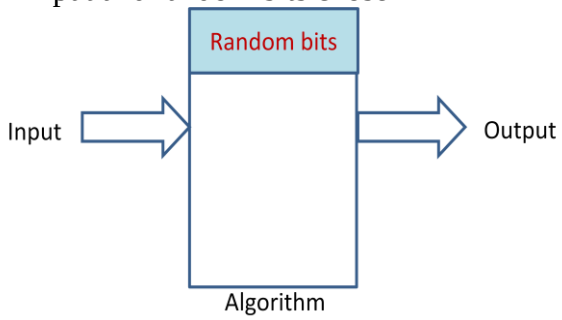
a) Discuss the advantages of randomized algorithms over deterministic algorithms. Discuss Las Vegas and Monte Carlo algorithms with a suitable example

Ans:

- **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

- **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

- **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) 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).

20.

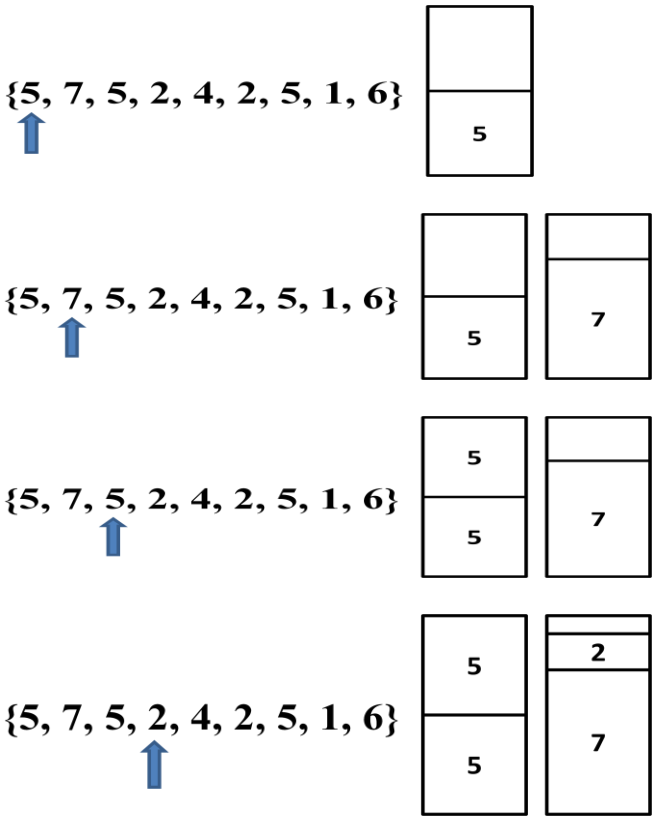
a) Define bin packing problem. Discuss the first fit strategy for solving it. State the approximation ratio of the algorithm

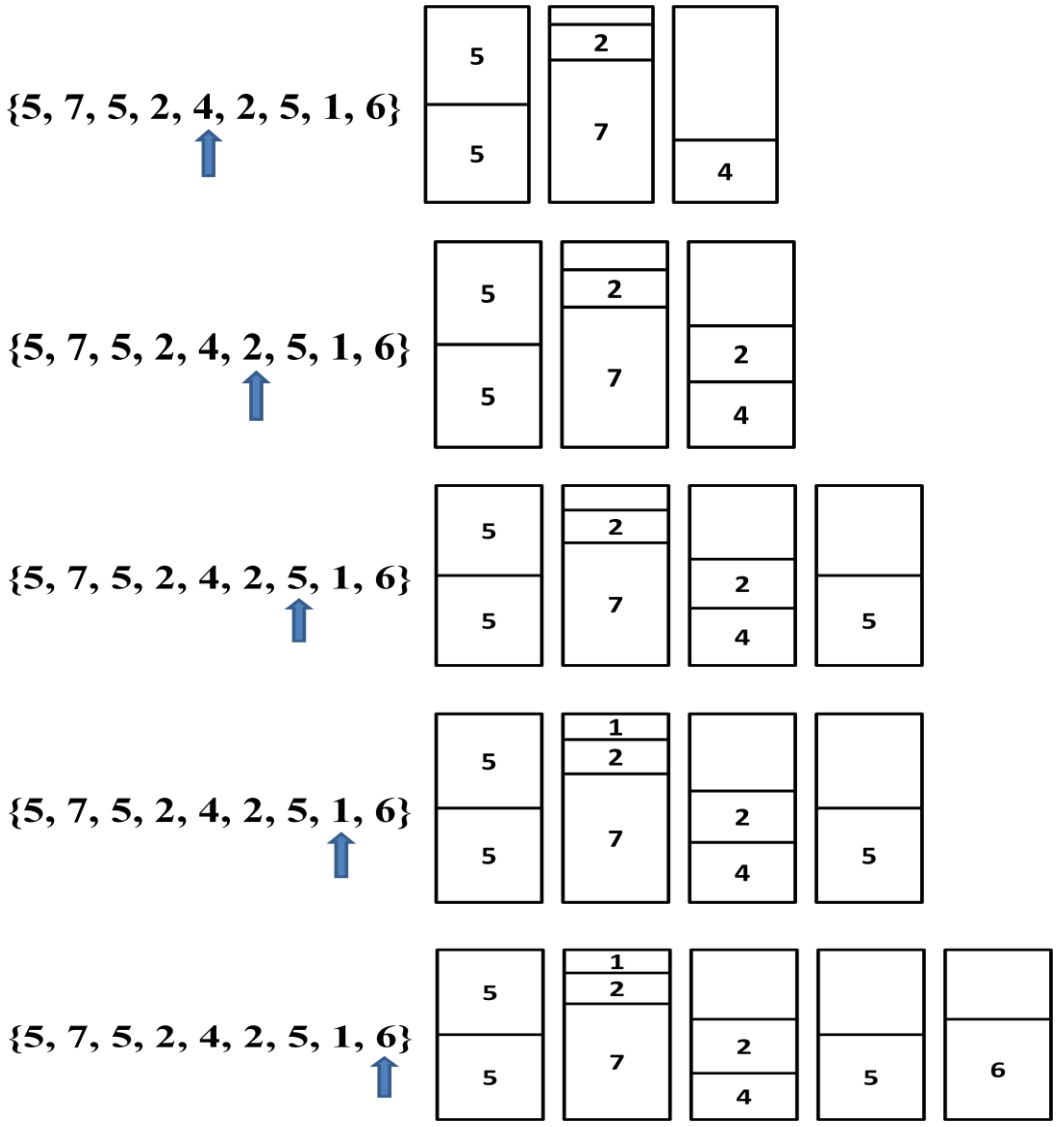
Ans:

- **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
    - Best case Time Complexity = $\theta(n \log n)$
    - Average case Time Complexity = $\theta(n^2)$
    - 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

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.

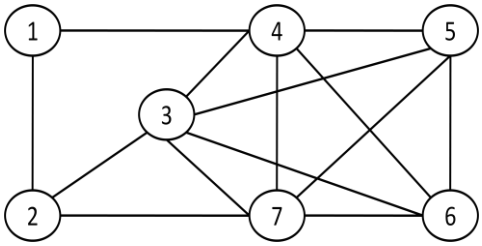
- **Steps to prove that Vertex Cover is a NP-Complete problem**
  - Step 1:Write a polynomial time verification algorithm to prove that the given problem is NP

- **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.**

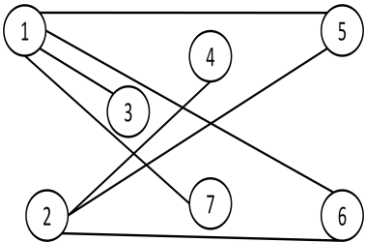
- Step 2: Write a polynomial time reduction algorithm from CLIQUE problem to VERTEX COVER problem
  - **Algorithm**

    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.

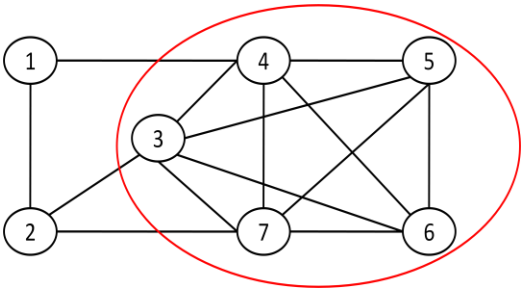
  - Example:

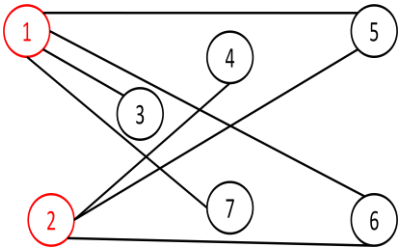


G=(V, E)



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