Analysis of Algorithm
(DEC 2018)

Q.P. Code – 55800

Q 1

A) Explain Strassen's matrix multiplication concept with an example, derive it's complexity . **10M**

- Strassen has proposed divide and conquer strategy-based algorithm, which takes less numbers of multiplications compare to this traditional way of matrix multiplication.
- Using Strassen's method, multiplication operation is defined as,

$\begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} * \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix}$

C11 = S1+ S4 – S5 + S7

C12 = S3+ S5

C21 = S2 + S4

C22 = S1 + S3 – S2 +S6

Where,

S1 = (A11 + A22) * (B11 + B22)

S2 =(A21 + A22) *B11

S3 = A11 *(B12 - B22)

S4 = A22 *(B21 – B11)

S5 = (A11 + A12) * B22

S6 = (A21 -A11) * (B11 +B12)

S7 = (A12 -A22) * (B21 +B22)

Let us check if it same as conventional approach.

C12 = S3 + S5

= A11 * (B12 – B22) + (A11 + A12) * B22

= A11 * B12 + A12 * B22

This is same as C12 derived using conventional approach. Similarly we can derive all Cij for Strassen's matrix multiplication. Algorithm for Strassen's multiplication.

Complexity:

Conventional approach performs eight multiplications to multiply two matrices of size 2*2. Whereas Strassen's approach performs seven multiplications on problem of size 1 * 1, which in turn finds the multiplication of 2 * 2 matrices using addition.

Strassen's approach creates seven problems of size (n-2).

Recurrence equation for Strassen's approach is given as,

$T(n) = 7T\left(\frac{n}{2}\right)$

Two matrices of size 1 * 1 needs only one multiplication, so best case would be, T(1) =1.

Let us find solution using iterative approach. By substituting n =(n/2) in above equation,

$$T\left(\frac{n}{2}\right) = 7T\left(\frac{n}{4}\right)$$

$$T(n) = 7^2 T\left(\frac{n}{2^2}\right)$$

.

.

$$T(n) = 7^k T\left(\frac{n}{2^k}\right)$$

Let's assume $n = 2^k$

$$T(2^k) = 7^k T\left(\frac{2^k}{2^k}\right) = 7^k.\mathrm{T}(1) = 7^k = 7^{log_2 n}$$

$$= n^{log_2 7} = n^{2.81} < n^3$$

Thus, running time of Strassen's matrix multiplication algorithm **O($n^{2.81}$)**

B) Apply the quick sort algorithm to sort the list E,X,A,M,P,L,E in alphabetical order. Analyze the best case, worst case and average case and complexities of quick sort. 10M

Example:

Given List:

E	X	A	M	P	L	E
0	1	2	3	4	5	6

Step1:

E	X	A	M	P	L	E

pivot low high

E	E	A	M	P	L	X

Pivot low high

E	E	A	M	P	L	X

Pivot low high

E	E	A	M	P	L	X

pivot low high

E	E	A	L	P	M	X

pivot Low high

E	E	A

Left partition

P	M	X

Right partition

L

pivot

SO, the final list will be

A	**E**	**E**	**L**	**M**	**P**	**X**

Complexities:

- Worst-case: The worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with n−1 elements and one with 0 elements.

- The partitioning costs θ(n) time.
- T(n) = T(n−1)+T(0)+ θ (n)
  = T(n−1)+θ (n).
  T(n) = $\theta(n^2)$
- Therefore the worst-case running time of quicksort is no better than that of insertion sort.

- Best-case: In the most even possible split, PARTITION produces two subproblems, each of size no more than n/2, since one is of size(n/2)and one of size(n/2)−1.
- T(n) ≤ 2T(n/2)+θ (n)
- T(n) = O(nlgn)

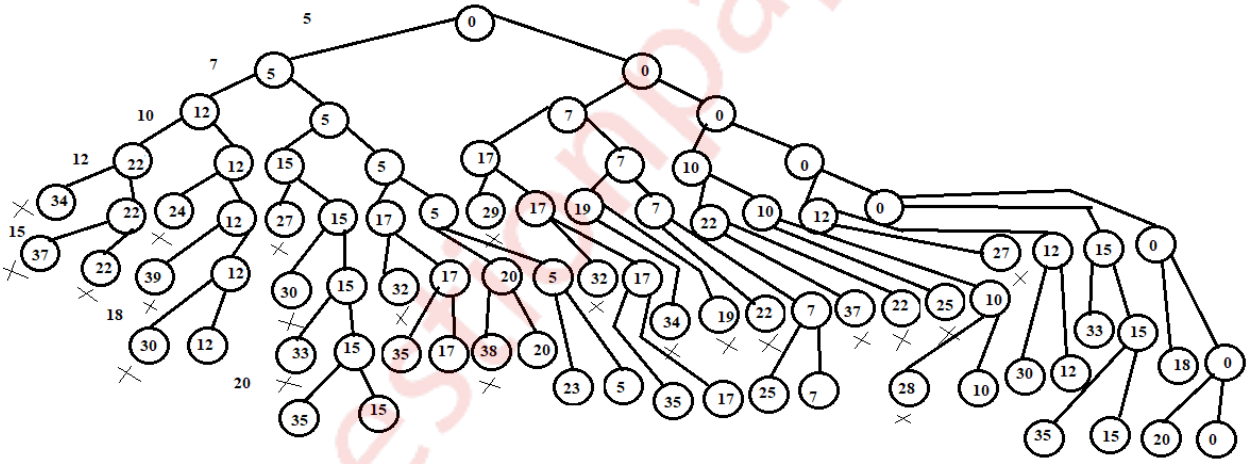
- Average-case: The average-case running time of quicksort is much closer to the best case than to the worst case.
- In the average case, PARTITION produces a mix of "good" and "bad" splits. In a recursion tree for anaverage-case execution of PARTITION, the good and bad splits are distributed randomly throughout the tree.
- O(nlgn).\

Q 2

A) Solve the following problem of sum of subset and draw portion of state space tree. W= (5, 7, 10, 12, 15, 18, 20) m=35

Find all possible subset of w that sum to m.

10M



Solution:

1) (5,8,12)
2) (15,20)
3) (5,10,20)
4) (5,12,18)

B) What is single source shortest path algorithm. Write an algorithm to find single source path using greedy methods. **10M**

- In a shortest-paths problem a weighted, directed graph G = (V, E), with weight function w : E → R mapping edges to real-valued weights.
- The weight of path p =(v0, v1, ..., vk) is the sum of the weights of it's constituent edges

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Bellman-Ford algorithm is used to find the shortest path from the source vertex to remaining all other vertices in the weighted graph.
- It is slower compared to Dijkstra algorithm but it can handle negative weights also.
- If the graph contains negative-weight cycle, it is not possible to find the minimum path, because on every iteration of cycle gives a better result.
- Bellman-Ford algorithm can detect a negative cycle in the graph but it cannot find the solution for such graphs.
- The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
- Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.

- Algorithm:

//Initialization
for each v ϵ V do
d[v]← ∞
π[v]← NULL
end
d[s]←0
//Relaxation
i←1 to |V| - 1 do
for each edge (u, v) ϵ E do
if d[u] + w (u,v) <d[v] then
d[v]← d[u] + w(u, v)
π[v]← u
end
end
end
//check for negative cycle
For each edge (u, v) ϵ E do
If d[u] +w(u, v)<d[v] then
Error "graph contains negative cycle"
end
end
return d, π

Q 3

A) Prove that vertex cover problem is NP complete. **10M**

Given that the Independent Set (IS) decision problem is N P-complete, prove that Vertex Cover (VC) is N P-complete.

Solution: 1. Prove Vertex Cover is in N P. I Given VC ,

vertex cover of G = (V, E), |VC | = k I We can check in O(|E| + |V|) that VC is a vertex cover for G.

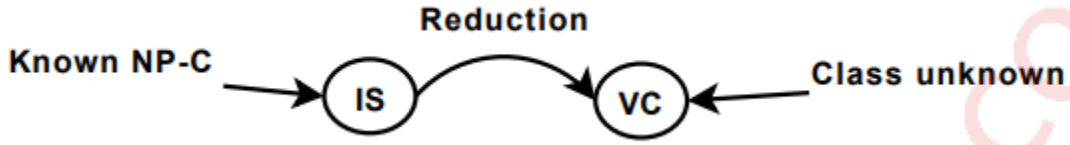
For each vertex ∈ VC , remove all incident edges.

Check if all edges were removed from G.

Thus, Vertex Cover ∈ N P

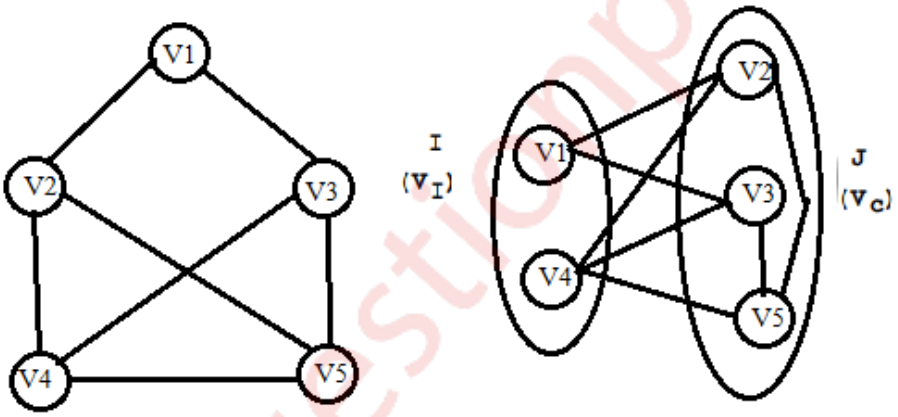
Select a known N P-complete problem.

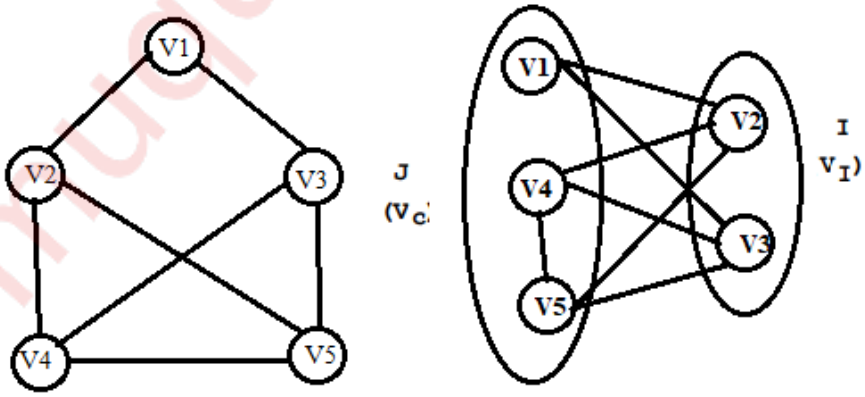
- Independent Set (IS) is a known N P-complete problem.
- Use IS to prove that VC is N P-complete.



3. Define a polynomial-time reduction from IS to VC:

- Given a general instance of IS: G 0=(V 0 , E 0), k 0
- Construct a specific instance of VC: G=(V, E), k I V =V 0 I E=E 0 I (G=G 0) I k=|V 0 | − k 0
- This transformation is polynomial:
- Constant time to construct G=(V, E)
- O(|V|) time to count the number of vertices
- Prove that there is a VI (|VI | = k 0) for G 0 iff there is an VC (|VC | = k) for G.





B) Explain various String matching algorithm. **10M**

There are various String matching algorithms listed below.

A] Naive:

i) It is the simplest method which uses brute force approach.

ii) It is a straight forward approach of solving the problem.

iii)It compares first character of pattern with searchable text. If match is found, pointers in both strings are advanced. If match not found, pointer of text is incremented and pointer of pattern is reset. This process is repeated until the end of the text.

iv)It does not require any pre-processing. It directly starts comparing both strings character by character.

v) Time Complexity = O(m*(n-m))

Algorithm -- **NAVE_STRING_MATCHING (T, P)**

```
for i  0 to n-m do
if P[1…….m] = = T[i+1…..i+m] then
print “Match Found”
end
end
```

B] Rabin-Karp:

i) It is based on hashing technique.

ii) It first compute the hash value of pattern and text. If hash values are same,i.e if hash(p) = hash(t). we check each character if characters are same pattern is found. If hash value are not same no need of comparing string.

iii) Strings are compared using brute force approach. If pattern is found then it is called as Hit. Otherwise it is called as Spurious Hit.Time Complexity = O(n), for worst case sometimes it is O(mn) when prime number is used very small.

Algorithm – **RABIN_KARP (T, P)**

```
n = T.length
m = P.length
 hᵖ = hash(T
 hᵗ = hash(T) (0……m-1)
 for S=0 to n-m
 if (hᵖ = hᵗ)
 if (P(0…..m-1) == T(0…..m-1))
 print “Pattern Found”
 if (S < n-m)
  hᵗ = hash(S+1……S+m-1)
```

C) Finite Automata:

i) Idea of this approach is to build finite automata to scan text T for finding all occurrences of of pattern P.

ii) This approach examines each character of text exactly once to find the pattern. Thus it takes linear time for matching but preprocessing time may be large.

iii) It is defined by tuple M = {Q, ∑, q_o, F, ∂}

Where Q = Set of States in finite automata

∑ = Sets of input symbols

q_o = Initial state

F = Final State

∂ = Transition function iv) Time Complexity = $O(M^3|\sum|)$

Algorithm – **FINITE_AUTOMATA (T, P)**

```
State ← 0
for I ← 1 to n
State ← ∂(State, t_i)
If State == m then
Match Found
end
end
```

D) Knuth Morris Pratt (KMP)

i) This is first linear time algorithm for string matching. It utilizes the concept of naïve approach in some different way. This approach keeps track of matched part of pattern.

ii) Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.

iii) This algorithm builds a prefix array. This array is also called as ∏ array.

iv) Prefix array is build using prefix and suffix information of pattern.

v) This algorithm achieves the efficiency of O(m+n) which is optimal in worst case.

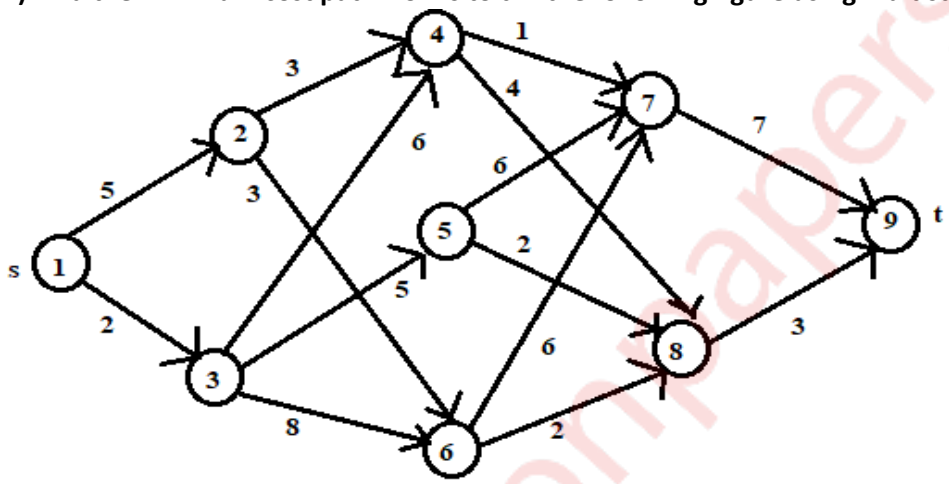
Algorithm – **KNUTH_MORRIS_PRATT (T, P)**

```
n = T.length
m = P.length
∏ = Compute prefix
q  0
for i = 1 to n
while q > 0 and P[q+1] ≠ T[i]
q = ∏ [q]
if P[q+1] = = T[i]
p= q+1
if q = = m
Print “pattern found”
q = ∏ [q]
COMPUTE_PREFIX (P)
M = P.length
Let ∏ [1……m] be a new array
```

∏ [1] = 0
K = 0
for k = 0 to m
while k > 0 and P[k+1] ≠ T[q]
k = ∏ [k]
if P[k+1] = = T[q]
k = k + 1
∏ [q] = k
return ∏

Q 4

A) Find the minimum cost path from s to t in the following figure using multistage graph. 10M



Stage 1:
Vertex 1 is connected to 2 and 3
Cost[1] = min{c[1, 2] , c[1, 3] }
= min {5, 2}
= 2

Stage 2:
Vertex 2 is connected to 4 and 6
Cost[2] = min {c[2,4], c[2, 6]}
= min{3, 3}
= 3
Vertex 3 is connected to 4, 5 and 6
Cost [3]= min{c[3,4], c[3,5], c[3, 6]}
= min{ 6, 5, 8}
= 5

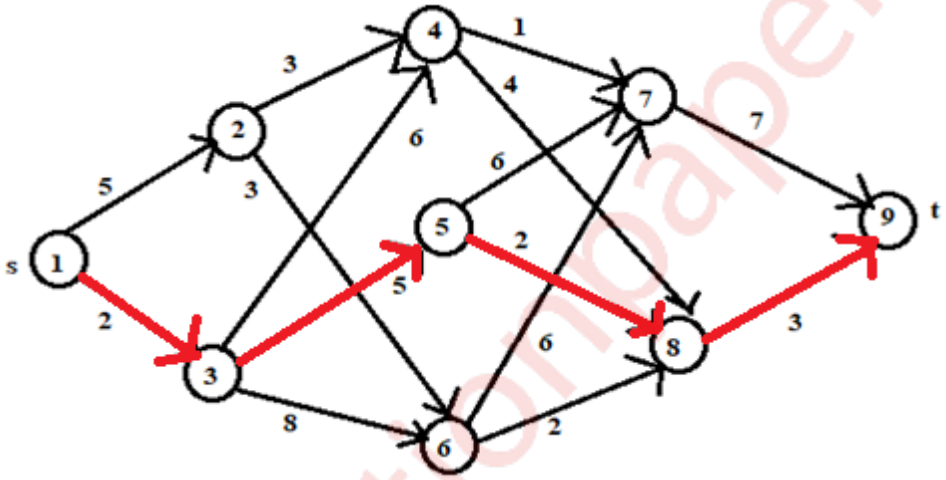
Stage 3:
Vertex 4 is connected 7 and 8
Cost [4] = min{c[4,7], c[4,8]}
=min {1, 4}
= 1

Vertex 5 is connected to 7 and 8
Cost [5]= min{c[5,7], c[5, 8]}
= min{6,2}
= 2
Vertex 6 is connected to 8
Cost [6]= min{c[6, 8]}
= 2

Stage 4:

Vertex 7 is connected to 9
Cost [7]= {c[7, 9]}
= 7
Cost [8]= {c[8, 9]}
= 3

Minimum cost path from s to t



B) Describe the travelling sales person problem and discuss how to solve it using dynamic programming with example. **10M**

- In the traveling-salesman problem given a complete undirected graph G = (V, E) that has a nonnegative integer cost c(u,v) associated with each edge (u,v) ∈ E, and we must find a Hamiltonian cycle (a tour) of G with minimum cost.
- As an extension of our notation, let c(A) denote the total cost of the edges in the subset A ⊆ E

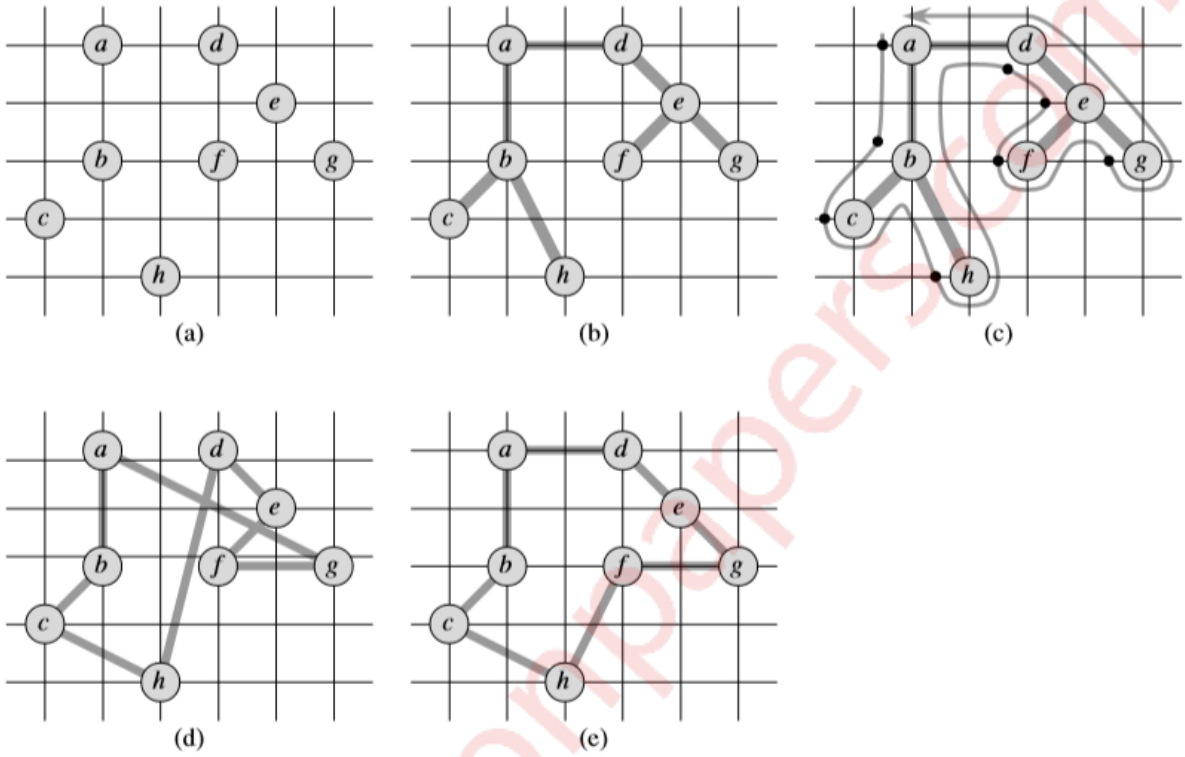
$$c(A) = \sum_{(u,v)\epsilon A} c(u,v)$$

- We formalize this notion by saying that the cost function c satisfies the triangle inequality if for all vertices u,v,w∈ V,
- c(u,w)≤c(u,v)+c(v,w)
- The triangle inequality is a natural one and in many applications it is automatically satisfied
- Algorithm for travelling sales man problem:

APPROX-TSP-TOUR(G,c)
1 select a vertex r ∈ V[G] to be a“root” vertex

2 compute a minimum spanning tree T for G from root r using MST-PRIM(G,c,r)
3 let L be the list of vertices visited in a preorder tree walk of T
4 return the hamiltonian cycle H that visits the vertices in the order L

Example:



Q 5

A) What is longest common subsequence problem? find the LCS for the following problem.

S1= abcdaf

S2= acbcf **10M**

i) The longest common sequence is the problem of finding maximum length common subsequence from given two string A and B.

ii) Let A and B be the two string. Then B is a subsequence of A. a string of length m has 2^m subsequence.

iii) This is also one type of string-matching technique. It works on brute force approach.

iv) Time complexity = O(m*n)

Algorithm **LONGEST_COMMON_SUBSEQUENCE (X, Y)**

// X is string of length n and Y is string of length m

for i ← 1 to m do

LCS [i, 0] ← 0

end

```
for j ← 0 to n do
LCS [0, j] ← 0
end
for i ← 1 to m do
for j ← 1 to n do
if Xi = = Yj then
LCS [i, j] = LCS [i-1, j-1] +
else if LCS [i-1, j] ≥ LCS [i, j-1]
 LCS [i, j] = LCS [i-1, j]
 else
 LCS [i, j] = LCS [i, j-1]
 end
 end
 end
 end
 return LCS
```

Example:

S1= abcdaf

S2= acbcf

Formula:

$$LCS(i,j) = \begin{cases} 0 & ,if\ i = 0\ or\ j = 0 \\ 1 + LCS[i-1, j-1] & ,if\ \ p_i = Q_j \\ \max(LCS[i, j-1], LCS[i-1, j]) & ,p_i \neq Q_j \end{cases}$$

LCS[1,1] ➔ i= 1, j=1, $p_i = A$, Q_i=B

LCS[1,1]=1	LCS[2,1]=1	LCS[3,1]=1	LCS[4,1]=1	LCS[5,1]=2
LCS[1,2]=1	LCS[2,2]=1	LCS[3,2]=2	LCS[4,2]=2	LCS[5,2]=2
LCS[1,3]=1	LCS[2,3]=1	LCS[3,3]=2	LCS[4,3]=3	LCS[5,3]=2
LCS[1, 4]=1	LCS[2,4]=1	LCS[3,4]=2	LCS[4,4]=3	LCS[5,4]=2
LCS[1,5]=2	LCS[2,5]=2	LCS[3,5]=2	LCS[4,5]=3	LCS[5,5]=2
LCS[1,6]=2	LCS[2,6]=3	LCS[3,6]=2	LCS[4,6]=3	LCS[5,6]=2

p_i ↓ Q_i →

			1	**2**	**3**	**4**	**5**	**6**
			a	**b**	**c**	**d**	**a**	**F**
		0	0	0	0	0	0	0
1	**a**	0	1	1	1	1	2	2
2	**c**	0	1	1	1	1	2	2
3	**b**	0	1	2	2	2	2	2
4	**c**	0	1	2	3	3	3	3
5	**f**	0	1	2	3	3	3	4

S1= abcdaf

S2= acbcf

So, LCS = abcf

B) Write short note on 8 queen problem, write an algorithm for the same. 10M

i) 8-queens problem is a problem in which 8 queens are arranged 8*8 chess board in such a way that no 2 queens should attack each other.

ii) 2 queens can attack each other if they are in same row, column or diagonal.

iii) Queen 1 is placed in a 1st column in the 1st row. All the position is closed in which queen 1 is attacking. In next level, queen 2 is placed in 3rd Column in row 2 and all cell are crossed which are attacked by already placed 1 and 2. This procedure keeps on going if we don't get feasible solution we backtrack and change the position of previous queen.

X	X	Q1	X	X	X	X	X
X	X	X	X	X	Q2	X	X
X	Q3	X	X	X	X	X	X
X	X	X	X	X	X	Q4	X
Q5	X	X	X	X	X	X	X
X	X	X	Q6	X	X	X	X
X	X	X	X	X	X	X	Q7
X	X	X	X	Q8	X	X	X

iv) 8 queen problem has $^{64}C_8$ = 4, 42, 61, 65, 368 different arrangements, out of these only 92 arrangements are valid solutions. Out of which only 12 are fundamental solution. Rest of 80 solutions can be generated by reflection or rotation.

v) Time Complexity = O(n!)

Algorithm **Queen (n)**

```
for column ← 1 to n do
{
if (Place(row, column)) then
{
Board [row]=column;
if (row = = n) then
Print_board (n)
else
Queen(row+1, n)
}
}
Place(row, column)
```

```
{
for i ← row -1 do
{
if (board [i] = column) then
return 0;
else if (abs(board [i] = column)) = abs(i-row) then
return 0;
}
```

Q 6 write a short note.

A) Branch and Bound strategy **10M**

- Branch and bound builds the state space tree and find the optional solution quickly by pruning few of the tree branches which does not satisfy the bound.
- Backtracking can be useful where some other optimization techniques like greedy or dynamic programming fail.
- Such algorithms are typically slower than their counterparts. In the worst case, it may run in exponential time, but careful selection of bounds and branches makes an algorithm to run reasonably faster.
- In branch and bound, all the children of E nodes are generated before any other live node becomes E node.
- Branch and bound technique in which E node puts its children in the queue is called FIFO branch and bound approach.
- And if E node puts its children in the stack, then it is called LIFO branch and bound approach.
- Bounding functions are a heuristic function.
- Heuristic function computes the node which maximize the probability of better search or minimizes the probability of worst search.
- Used to solve optimization problems.
- Nodes in tree may be explored in depth-first or breadth-first order.
- Next move is always towards better solution.
- Entire state space tree is searched in order to find optimal solution.
- Application: Travelling salesman problem, Knapsack problem

-Branch and bound technique generate all the child nodes of E-node before another node becomes E node.

-Let c(x) represent the cost of answer node x. The aim is to find minimum cost answer node.

-Search strategies like FIFO, LIFO and LC search differs in terms of the sequence in which they explore the node in state space tree.

Branch and bound method employ either BFS or DFS search. During BFS, expanded nodes are kept in a queue, whereas in DFS nodes are kept on the stack.

- Example: FIFO branch and bound approach.

B) Algorithms to find minimum spanning tree **10M**

- A spanning tree of a connected undirected graph G, is a sub-graph of G which is a tree that connects all the vertices together.
- A graph G can have many different spanning trees. We can assign weights to each edge.

- Use it to assign a weight to a spanning tree by calculating the sum of the weights of the edges in that spanning.
- A minimum spanning tree (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree.

- Algorithms:

Prim's algorithm:

- Prim's algorithm is a greedy algorithm that used to form a minimum spanning tree for a connected weighted undirected graph. In other words, the algorithm builds a tree that includes every vertex and a subset of the edges in such a way that the total weight of all the edges in the tree is minimized.
  - Tree vertices: Vertices that are a part of the minimum spanning tree T.
  - Fringe vertices: Vertices that are currently not a part of T, but are adjacent to some tree vertex.
  - Unseen vertices: Vertices that are neither tree vertices nor fringe vertices fall under this category.

- The steps involved in the Prim's algorithm:

Step 1: Select a starting vertex
Step 2: Repeat Steps3and4until there are fringe vertices
Step 3: Select an edge connecting the tree vertex and fringe vertex that has minimum weight
Step 4: Add the selected edge and the vertex to the minimum spanning tree T [END OF LOOP]
Step 5: EXIT

Kruskal's algorithm:

- Kruskal's algorithm is used to find the minimum spanning tree for a connected weighted graph.
- The algorithm aims to find a subset of the edges that forms a tree that includes every vertex.
- The total weight of all the edges in the tree is minimized.
- However, if the graph is not connected, then it finds a minimum spanning forest.
- Kruskal's algorithm is an example of a greedy algorithm, as it makes the locally optimal choice at each stage with the hope of finding the global optimum.
- Algorithm:

Step 1: Createaforest in suchaway that each graph isaseparate tree.
Step 2: Createapriority queueQthat contains all the edges of the graph.
Step 3: Repeat Steps4and5whileQis NOT EMPTY
Step 4: Remove an edge from Q
Step 5: IF the edge obtained in Step4connects two different trees, then Add it to the forest (for combining two trees into one tree). ELSE Discard the edge
Step 6: END

C) Recurrences **10M**

Definition:

Recurrence equation recursively defines a sequence of function with different argument, behavior of recursive algorithm is better represented using recurrence equations.

- Recurrence are normally of the form

$T(n) = T(n-1) + f(n)$, for $n > 1$
$T(n) = 0$, for $n = 0$

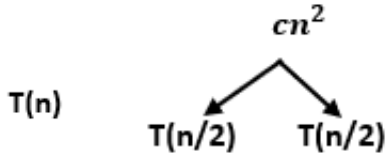
- The function f(n) may represented constant or any polynomial in n.
- T(n) is interpreted as the time required to solve the problem of size n.
- Recurrence of linear search

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

- Recurrence of selection/ bubble sort
- Recurrence is used in to represent the running time of recursive algorithms.
- Time complexity of certain recurrence can be easily solved using master methods.
- Substitution Method: Linear homogeneous recurrence of polynomial order greater than 2 hardly arises in practice.
- Two ways to solve unfolding method

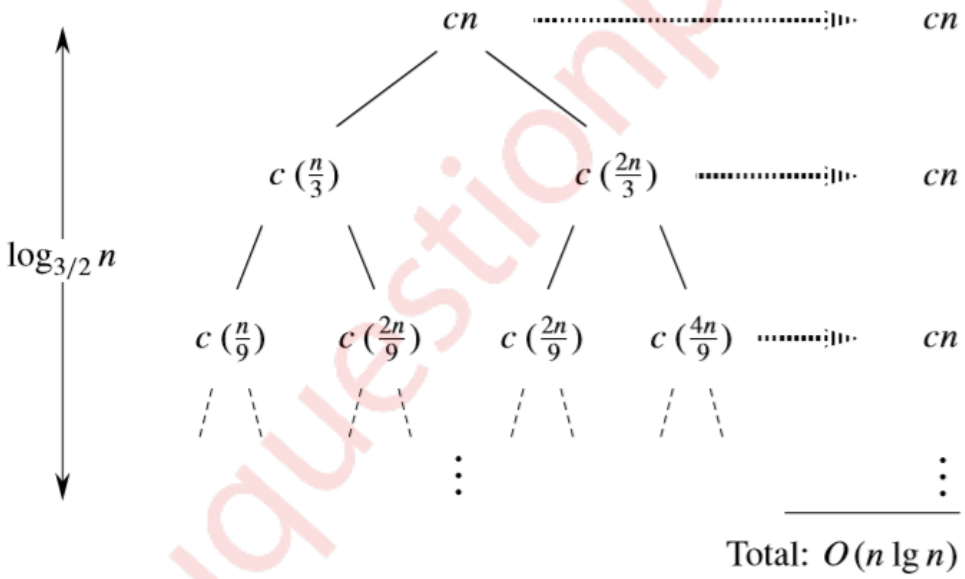
i) Forward substitution ii) Backward substitution

- **Recursion Tree:**
- Recurrence tree method provides effective is difficult for complex recurrence.
- Ultimately, recurrence is the set of functions, each branch in recurrence tree represents the cost of solving one problem from the family of problems belonging to given recurrence.



a) T (n) b) First level expansion of T(n)

- A recursion tree for the recurrence T(n)= T(n/3)+T(2n/3)+cn.



****