Analysis of Algorithm
(May 2018)

Q.P. Code - 38841

Q 1 Answer the following

a. Write the difference between greedy method and dynamic programming. 5M

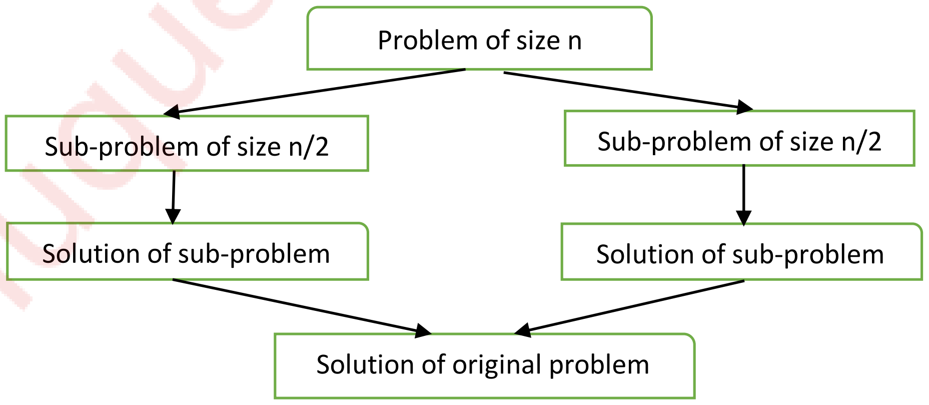
Greedy method	Dynamic programming
1. Greedy method does not guarantee an optimal solution.	1. Dynamic programming guarantees an optimal solution.
2. Sub-problems do not overlap.	2. Sub-problems overlap.
3. It does little work.	3. It does more work.
4. Only considers the current choices.	4. Considers the future choices.
5. Construct the solution from the set of feasible solutions.	5. There is no specialized set of feasible choices.
6. Select choice which is locally optimum.	6. Select choices which is globally optimum.
7. There is no concept of memorization.	7. Employ memorization.
8. Examples- Knapsack problems, Job scheduling with deadlines, Kruskal , Prim's.	8. Examples- All pair shortest path, 0/1 Knapsack, Travelling salesman problem, LCS.

b. Explain the general procedure of divide and conquer method. 5M

Divide and Conquer method: This is the most widely applicable technique for designing efficient algorithms. It works in three stages as shown below.

1) **Divide**: Recursively divide the bigger problem of size n into smaller sub-problems of size n/2
2) **Solve**: Sub-problems are solved independently.
3) **Combine**: Combine solutions of smaller sub-problems to derive the solution of larger big problem of size n.

Smaller sub-problems are similar to the larger problem with smaller arguments. Hence such Problems can be solved easily using recursion. Divide and conquer is multi-branched, Top-Down recursive approach. Each branch indicates one sub-problems and it calls itself with the Smaller argument.



Above diagram shows the working of Divide and Conquer method. Sub-problem may or may not be of size n/2.

c. **Determine the frequency counts for all statement in the following algorithm Segment.** **5M**

```
I=1;
While(I<=n)
{
X=X+I;
I=I+1;
}
```

	S/e	Frequency	Total Steps
I=1;	1	1	1
While(I<=n)	1	I + n	I + n
{	0	-	-
X=X+I;	1	I	I
I=I+1;	1	1	1
}	0	-	-
			2(I+1) + n.

d. **What is backtracking Approach? Explain how it is used in Graph Coloring.** **5M**

Backtracking Approach:

1) Backtracking is the process where the entire problem is divided into several stages. The algorithm then attempts to find the solution to the problem by constructing partial solutions remain consistent with the requirements of the problem.
2) However, when an inconsistency with the requirement of the problem occurs the algorithm backs up (backtracks) by removing the most recently constructed part of the solution and trying another possibility.
3) For example, the algorithm will first find solution for stage one and stage two and so on till stage N. However, if it cannot find the solution for stage N+1, then it will go back to stage N, remove the solution it found for stage N and will come up with a new solution for stage N.
4) Examples: Graph Coloring & n-Queens problem.

Graph Coloring:

1) Coloring the vertices of the graph in such a way such that no two adjacent vertices have the same color. This is called Graph Coloring.
2) It is also called as Vertex coloring or K-coloring.
3) The smallest number of color required for coloring the graph is called as Chromatic number.
4) Example:

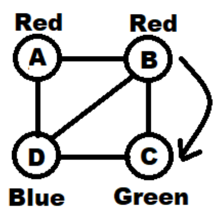


Fig. A

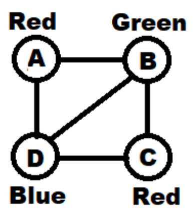


Fig. B

In Fig. A same color(Red) is assigned to adjacent vertices (A & B) which is against rule so we Backtrack and change the color as shown in the Fig. B. We cannot assign another color on the

Place of Red because chromatic color should be minimum. If we assign another color on the Place of Red Chromatic Number Will be 4 for Fig. A. and if we see Fig B. chromatic number is 3 Which is minimum. Fig. B is Right solution for Graph coloring,

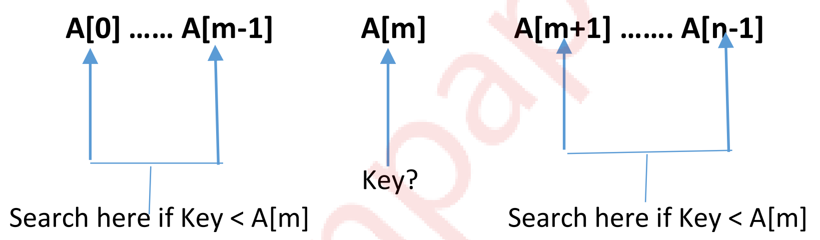
Time Complexity – O(m^n)

Q 2.a. Explain with example how divide and conquer strategy is used in binary Search? 10M

Binary search is an efficient searching method. While searching the elements using this method the most essential thing that the elements in the array should be sorted one. An element which Is to be sorted from the list of elements sorted in array should be sorted one. An element which is to be searched from the list of elements stored in array A [0…n-1] is called KEY element. Let A [m] be the mid element of array A. then these are three conditions that needs to be satisfied while searching the array using this method.

1) If KEY=A[m] then desired element is present in the list.
2) Otherwise if KEY<A[m] then search the left sub list
3) Otherwise if KEY>A[m] then search the right sub list.

This can be represented as



As shown above the array is divided into two part left and right sub list according to key element We decide in which list we can find the element. This is how divide and conquer strategy is used in Binary search.

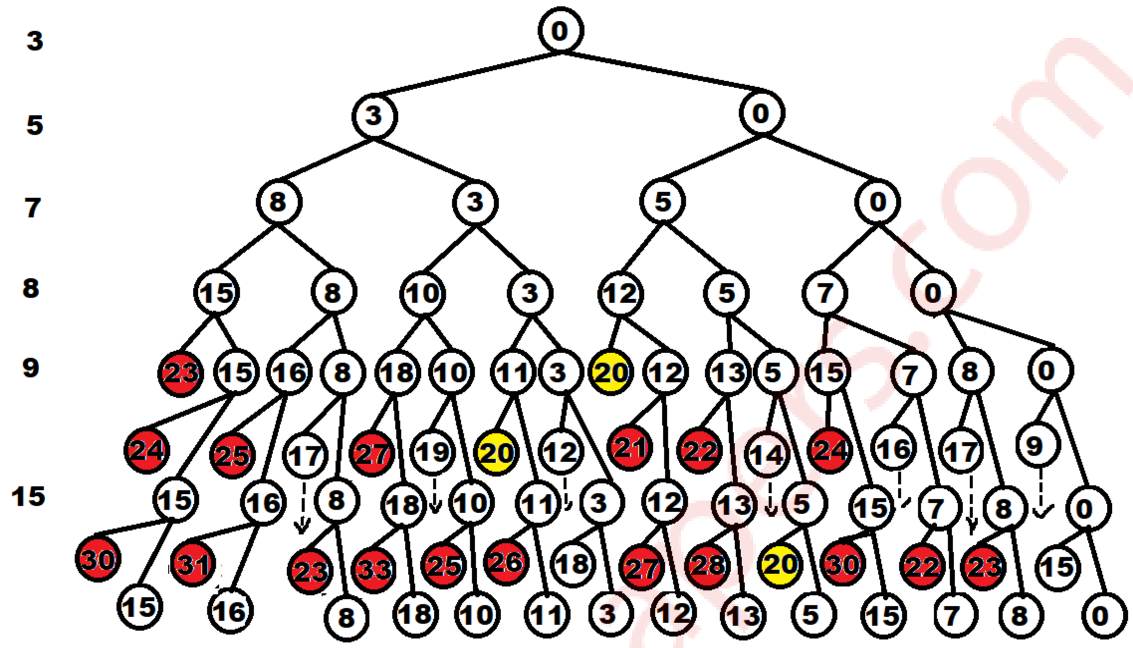
Algorithm: **BINARY_SEARCH(A, Key)**

```
Low <- 1
High <- n
While low < high do
    Mid <- (low + High) / 2
  If A[Mid] == Key then
      return Mid
  else if A[Mid] < Key then
      Low <- mid + 1
  else
      High <- Mid – 1
     end
   end
return 0
```

Time Complexity – Best case =O(1), Average case & worst case = O($\log_2 n$)

b. Solve sum of subsets problem for following **10M**

N=6 W={3,5,7,8,9,15} & M=20 Also write the Algorithm for it.



Solutions : Subset **1) {5, 7, 8}**

2) {9, 8, 3}

3) {5, 15}

<u>Algorithm</u>:

Let W be a set of elements & M be the expected sum of subset then

Step1: Start with empty set.

Step2: Add to the subset, the next element from the list.

Step3: If the subset is having sum equal to M then Stop with that subset as solution,

Step4: If the subset is not matching with the M or if we have reached to the end of the set Then backtrack through that subset until we find suitable value.

Step5: If the subset is less then M then repeat Step2.

Step6: If we have visited all the elements without finding suitable & No backtrack is possible Then stop without solution.

Time Complexity – O(2^n)

Q.3.a. Obtain the solution to knapsack problem by Greedy method n=7, m=15 (p1, P2,…P7)=(10,5,15,7,6,18,3), (W1,W2,…W7)=(2,3,5,7,1,4,1) **10M**

Item	Weight	Value	Value / Weight
P1	2	10	5
P2	3	5	1.667
P3	5	15	3
P4	7	7	1
P5	1	6	6
P6	4	18	4.5
P7	1	3	3

Arrange the Item in increasing order of value / Weight Ratio
P5, P1, P6, P7, P3, P2, P4.
Capacity of Bag is 15.

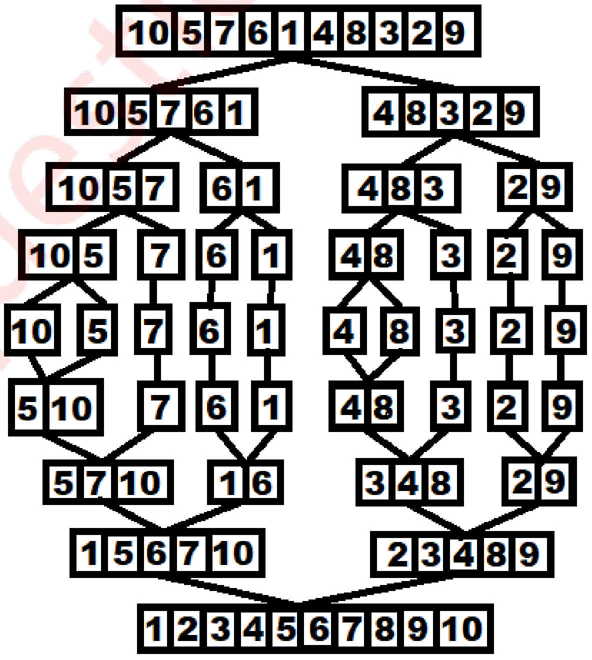
Capacity of Bag	Items	value
15	-----	0
14	P5	6
12	P5, P1	16
8	P5, P1, P6	34
7	P5, P1, P6, P7	37
2	P5, P1, P6, P7, P3	52

Now, P2 arrives but capacity of bag is only 2 and weight is 3 so we are going to take fraction

Maximum Profit = $52 + \frac{2}{3} * 5$ = **55.33**

b. Sort the list of the elements 10,5,7,6,1,4,8,3,2,9 using merge sort algorithm and show its computing time is O(n logn). **10M**

Merge sort 10, 5, 7, 6, 1, 4, 8, 3, 2, 9



Complexity Analysis

Using Master method: T(n) = 2 T$\left(\frac{n}{2}\right)$ + n

Comparing with : T(n) = a T$\left(\frac{n}{b}\right)$ + f(n)

a = 2, b = 2 & f(n) = n with k = 1

where k is degree pf polynomial function f(n)

a = b^K, 2 = 2^1.

So from the case I variant 2

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

T(n) = O(n log n)

"Hence Proved"

Time Complexity - T(n) = O(n $\log_2$n)

Q.4.a. Explain different string matching algorithms. **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.

iv) 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
$h^p$ = hash(T
$h^t$ = hash(T) (0………m-1)
for S=0 to n-m
 if ($h^p$ = $h^t$)
if (P(0…..m-1) == T(0…..m-1))
 print "Pattern Found"
if (S < n-m)
 $h^t$ = 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.

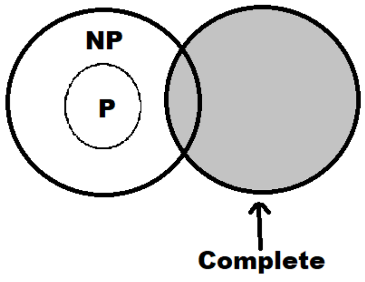
```
Algorithm – KNUTH_MORRIS_PRATT (T, P)
        n = T.length
        m = P.length
        ∏ = Compute prefix
        q ← 0
        for i = 1 to n
        while q > 0 and P[q+1] ≠ T[i]
           q = ∏ [q]
        if P[q+1] = = T[i]
             q = q+1
          if q = = m
        Print "pattern found"
           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 ∏
```

b. What do you understand by NP Complete? Explain is subset sum problem NP complete? If so explain. **10M**

i) NP complete is the combination of both NP and NP hard problem.

ii) Decision problem C is called NP complete if it has following two properties.

- C is in NP, and
- Every problem X in NP is reducible to C in polynomial time, i.e. For every X ε NP, X $\leq_p$ C
  This two factor prove that NP-complete problems are the harder problems in class NP. They often referred as NPC.



NP Complete

Sum of Subset:

i) Sum of subset is NP complete.

ii) It satisfy the above two condition. The problem cannot be solved in polynomial time but can be verified in polynomial time hence it is NP. Every problem in NP can be reduced in polynomial time hence it is NP hard. Therefore, sum of subset id NP complete.

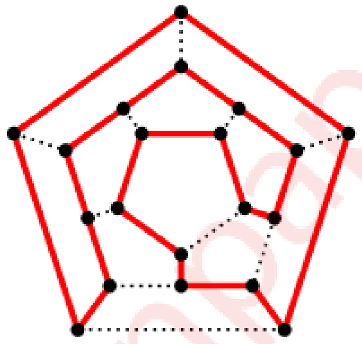
Example: set- { 3, 5, 7, 8, 9, 15}

Time complexity = $O(2^n) = 2^6 = 64$ which is quiet large.

So it requires more time to compute then polynomial time. But its verification is easy it Can be done into polynomial time.

Q.5.a. write a detailed note on Hamiltonian cycles. **10M**

- The Hamiltonian cycle of undirected graph G = <V, E> is the cycle containing each vertex in V.
- If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian.



The dodecahedron shows the Hamiltonian, and the Hamiltonian cycle is shown by thick red lines. Whereas the bipartite graph with an odd number of vertices as shown above.

- One way of checking if the graph is Hamiltonian or not is to list out all possible permutation of Vertices and check them one by one. There are m! different permutation of m vertices, and hence The running tome of algorithm would be $\Omega(m!)$ time. By encoding the graph using its adjacency Matrix representation, we can reduce the time $\Omega(\sqrt{N}) = \Omega(2^{\sqrt{N}})$ which is not polynomial.
- Here n represents the length of encoding of graph G. Thus, the given problem cannot be solved in Polynomial time.

Verification

- If given the solution string of Hamiltonian graph, it is very easy to prove the correctness of it. We Just need to check if the solution contains all the vertices of V and there must be an edge Between two consecutive vertices. This can be done in $O(n^2)$ time. Thus, the solution can be Verified in polynomial time., where n is the length of graph G. Verification algorithm A(x, y) is defined by two arguments, where x is the input string and y is the Binary string, called certificate y such that A(x, y) = 1, we say that algorithm verifies the string x. And the language defined by the algorithm is, $L = \{x \in \{0, 1\}^* :$ there exist $y \in \{0, 1\}$ such that $A(x, y) = 1\}$
- For each legal string $x \in L$, A must verify the string and produce the certificate y. And for any string x which is not in L, must not verify the string, and no certificate can prove that $x \in L$. For example if the graph is Hamiltonian, the proof can be checked in polynomial time as discussed earlier. But no vertices sequence should exist which can fool the algorithm to prove the non-hamiltonian graph as Hamiltonian.

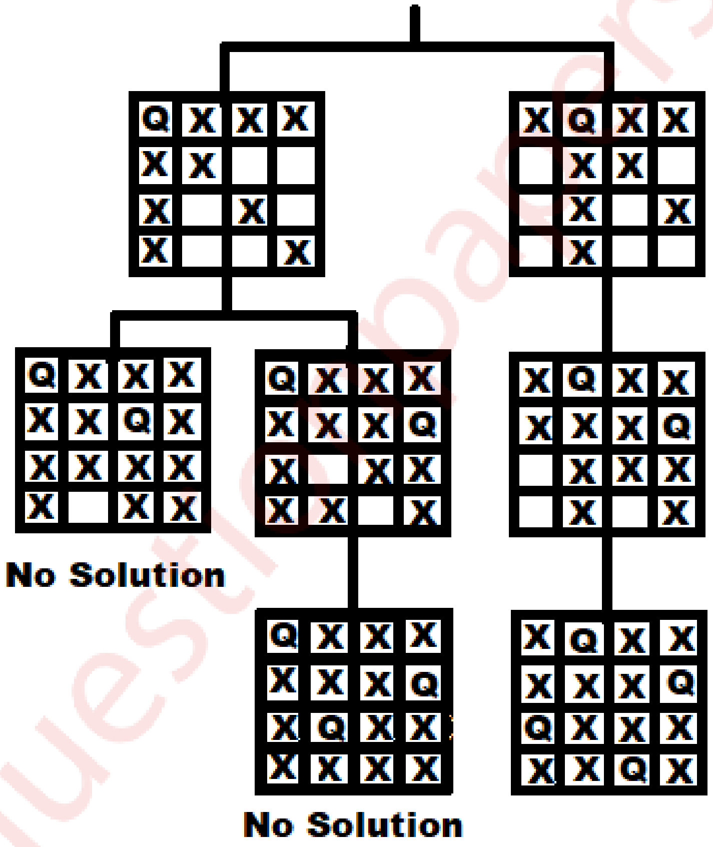
b. **Explain how backtracking is used for solving n- queens problem. Show the state space tree.** 10M

i) n-queens problem is a problem in which n-queens are placed in n*n chess board, such that no 2 queen should attack each other.

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

iii) For simplicity, State space tree is shown below. 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. As can be seen below. No place is left to place neat queen in row 3, so queen 2 backtracks to next possible and process continue.

iv) In a similar way, if (1, 1) position is not feasible for queen 1, then algorithm backtracks and put the first queen cell (1, 2), and repeats the procedure. For simplicity, only a few nodes are shown below in state space tree.



v) Complete state space tree for the 4 – queen problem is shown above. 2 – queen problem is not feasible.

vi) This is how backtracking is used to solve n-queens problems.

Q 6 Write short notes on (any two) **20M**

a. Job sequencing with deadlines.

i) Job sequencing is a problem in which n jobs are arranged in such a way that we get maximum Profit. The job should complete their task within deadlines.

ii) It is also called as Job scheduling with deadlines.

iii) Time complexity = $O(n^2)$

Algorithm

Step1: Sort the jobs J_i into non-increasing order.

Step2: Assign Empty slot as Maximum deadline value i.e. Empty slot = Dmax.

Step3: Take i index value compute value of k

K = min (Dmax, deadline(i))

Step4: If value of K is greater and equal to one check empty slot. Empty slot = k

If empty slot is full. check previous slot if it is free assign job to that slot.

If slot is full ignore that job.

Step5: increment value of i till all the jobs are not finished. Repeat Step3.

Step6: Stop.

Example

Let n=4, (J1, J2, J3, J4)=(100, 10, 15, 27), (D1, D2, D3, D4)=(2, 1, 2, 1) find feasible solution. With job scheduling with deadlines.

Index	1	2	3	4
Job	J1	J2	J3	J4
Value	100	10	15	27
Deadlines	2	1	2	1

Arrange the jobs in non-increasing value.

Index	1	2	3	4
Job	J1	J4	J3	J2
Value	100	27	15	10
Deadlines	2	1	2	1

Dmax = 2

Time slot	1	2
Status	Empty	Empty

i=1

K= min (Dmax, Deadline(i))

K= min (2, 2)

K=2 ($k \geq 1$)

Time slot = k = 2 (empty)

Time slot	1	2
Status	Empty	J1

i=2
K= min (Dmax, Deadline(i))
K= min (2, 1)
K=1 (k ≥ 1)
Time slot = k = 1 (empty)

Time slot	1	2
Status	J4	J1

Maximum Profit = J1 + J4
= 27 + 100
= **127**

b. 8 Queen problem

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;
           }
         return 1;
        }
```

c. **Longest common subsequence**

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 X_i = = 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 A = ABCF, B = ACF

A =

0	1	2	3
A	B	C	F

O	1	2
A	C	F

Draw matrix

0	0	0	0
0			
0			
0			
0			

r = 1, c = 1

x [0] = y [0] (true A = A)

LCS [1, 1] = L [0, 0] + 1 = 0+1 = 1

0	0	0	0
0	1		
0			
0			
0			

r = 1, c = 2

x [0] ≠ y [0] (true A = C)

LCS [0, 1] ≥ LCS [1, 1] (0 ≥ 1 (not true))

LCS [1, 2] = LCS [1, 2-1]

LCS [1, 2] = L [1, 1] = 1

0	0	0	0
0	1	1	
0			
0			
0			

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

0	0	0	0
0	↘ 1	1	1
0	1	1	1
0	1	↘ 2	2
0	1	2	↘ 3

LCS =

0	1	2
A	C	F

****