Digital Assignment

DAA

M.V.S. PRAGNYA BALA

22BKT0164

Q1. Find the minimized number of multiplication required for A1 * A2 * A3 * A4. Find optimal parenthesisation of a chain of matrices to be multiplied such that number of scalar multiplications is minimized

<u>Given</u>:

Matrices

$A1 = 10 \times 100 \quad A2 = 100 \times 5 \quad A3 = 5 \times 50 \quad A4 = 50 \times 7$

Matrix A_i has dimension $P_{i-1} * P_i$

$A_1 = 10 \times 100 = P_{1-1} * P_1 = P_0 * P_1$

$A_2 = 100 \times 5 = P_{2-1} * P_2 = P_1 * P_2$

$A_3 = 5 \times 50 = P_{3-1} * P_3 = P_2 * P_3$

$A_4 = 50 \times 7 = P_{4-1} * P_4 = P_3 * P_4$

$P_0 = 10 \quad P_1 = 100 \quad P_2 = 5 \quad P_3 = 50 \quad P_4 = 7$

$$m[i,j] = \begin{cases} 0 \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + P_{i-1} \cdot P_k \cdot P_j\} \end{cases}$$

if $i = j$

i \ j	1	2	3	4
1	0			
2		0		
3			0	
4				0

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

if $i < j$

$m[1,2] = \min\limits_{i \le k < 2} \{m[1,1] + m[2,2] + P_{i-1} \cdot P_1 \cdot P_2$

$= 0 + 0 + 10 \times 100 \times 5 = 5000$

$s[1,2] = 1$

$m[2,3] = \min\limits_{2 \le k < 3} \{m[2,2] + m[3,3] + P_{2-1} \cdot P_2 \cdot P_3\}$

$= 0 + 0 + P_1 \cdot P_2 \cdot P_3$

$= 100 \times 5 \times 50 = 25000$

$s[2,3] = 2$

$$\min[3,4] = \min_{3 \le k < 4} \{ m[3,3] + m[4,4] + P_{3-1} \cdot P_3 \cdot P_4 \}$$

$$= 0 + 0 + P_2 \cdot P_3 \cdot P_4$$

$$= 5 \times 50 \times 7 = 1750$$

$s[3,4] = 3$

	1	2	3	4
1		1		
2			2	
3				3
4				

$s(i,j)$

	1	2	3	4
1	0	5000		
2		0	25000	
3			0	1750
4				0

$m(i,j)$

$$\min[1,3] = \min_{i \le k < j} \{ \min[i,k] + m[k+1,j] + P_{i-1} \cdot P_k \cdot P_j \}$$

$$= \min_{k=1,2} \begin{cases} m[1,1] + m[2,3] + P_0 \cdot P_1 \cdot P_3 \\ m[1,2] + m[3,3] + P_0 \cdot P_2 \cdot P_3 \end{cases}$$

$$= 0 + 25000 + 10 \cdot 100 \cdot 50,\ 5000 + 0 + 10 \cdot 5 \cdot 50$$

$$= 75000,\ 7500$$

$$= 7500$$

$$\min[2,4] = \min_{\substack{2 \le k < 4 \\ k = 2,3}} \begin{cases} m[2,2] + m[3,4] + P_{2-1} \cdot P_2 \cdot P_4, \\ m[2,3] + m[4,4] + P_{2-1} \cdot P_3 \cdot P_4 \end{cases}$$

$$= 0 + 1750 + 100 \cdot 5 \cdot 7,\ 25000 + 0 + 100 \cdot 50 \cdot 7$$

$$= 5250,\ 60000$$

$$= 5250.$$

$$m[1,4] = \min_{\substack{i \le k < j \\ k = 1,2,3}} \{ \min[i,k] + m[k+1,j] + P_{i-1} \cdot P_k \cdot P_j \}$$

$$= \{ m[1,1] + m[2,4] + P_0 \cdot P_1 \cdot P_4,\ m[1,2] + m[3,4] + P_0 \cdot P_2 \cdot P_4,\ m[1,3] + m[4,4] + P_0 \cdot P_3 \cdot P_4 \}$$

$$= 0 + 5250 + 10 \cdot 100 \cdot 7,\ 5000 + 1750 + 10 \cdot 5 \cdot 7,\ 7500 + 0 + 10 \cdot 50 \cdot 7$$

$$= 7000 + 5250,\ 7100,\ 7500 + 3500$$

$$= 7100$$

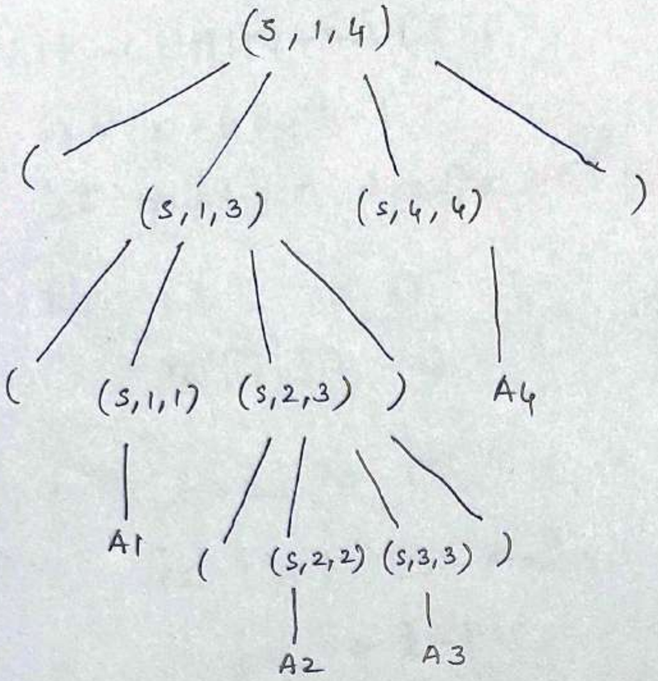
j →, i ↓

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

$s(i,j)$

	1	2	3	4
1	0	5000	7500	7100
2		0	25000	5250
3			0	1750
4				0

$m(i,j)$



Answer : ((A1 (A2 . A3)) A4)

Optimal cost = 7100 .

```
MATRIX-CHAIN-MULT(p)                                              TC
    n = p.length-1                                                1
    let m[1...n, 1...n] and s[1...n-1, 2...n]
    for i = 1 to n                                                n
        m[i, i] = 0
    for l = 2 to n  // l is chain length                          n
        for i = 1 to n-l+1                                        n^2
            j = i+l-1                                             n^2
            m[i,j] = ∞                                            n^2
            for k = i to j-1                                      n^3
                q = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j          n^3
                if q < m[i,j]                                     n^3
                    m[i,j] = q
                    s[i,j] = k
    return m and s                                                1
```

Asymptotically we get dominant as n^3. $\therefore$ TC is $O(n^3)$

```
Print-Optimal-Output(s, i, j)
    if i == j
        print "A_i"
    else print "("
        Print-Optimal-Output(s, i, s[i, j])
        Print-Optimal-Output(s, s[i,j]+1, j)
        print ")"
```

Time complexity analysis

Outer loop runs from 2 to n, where n is length of chain. This loop has c_1 (cost) & runs $n-1$ times

$\therefore$ Total cost $= c_1(n-1)$

$\therefore$ This loop has time complexity $O(n)$

Within outer loop, there are two nested loops.

The second loop runs from 1 to $n-l+1$

and innermost loop from i to $j-1$.

and second loop runs $n-l+1$ times

Let cost is c_2

$\therefore$ total cost $c_2(n-l+1)$

and inner loop runs i to $j-1$ i.e $j-i$ times

Let cost is c_3

$\therefore$ Combined time complexity of these nested loops is $O(n^2)$

Outer loop runs for n times $\Rightarrow O(n)$

Inside innermost loop, the algorithm performs constant-time operations.

Due to 3 nested loops we can say that asymptotically time complexity $(n * n^2 \approx n^3)$ is $O(n^3)$

We can also analyse in another way
Consider a general $n \times n$ matrix
Here I am representing only a 4×4 matrix

0	x_1	x_4	x_6
0	0	x_2	x_5
0	0	0	x_3
0	0	0	0

4×4

To compute the rightmost top value (here x_6) we ensure k ranges from 1 to $n-1$ for an $n \times n$ matrix
(Here $n=4$)

we are using the formula

$$m_{ij} = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \} & \text{if } i<j \end{cases}$$

the cost of accessing is constant so to evaluate above expression constant time c is taken
Like this we need to evaluate $n-1$ expressions
$\therefore$ total cost is $c(n-1)$ for 1st level value computation
Similarly for next level of values (here x_4, x_5) we need k to range from $k=1$ to $k=n-2$ and we have 2 such values here that we have to compute.

$\therefore$ total cost of computation for 2nd level is

$$2 \times c \times (n-2)$$

$\vdots$

so on we will have $n-1$ levels for $n \times n$ matrix

$(n-1)^{th}$ level the cost of computation will be

$$(n-1) \times c \times (n-(n-1))$$

$\therefore$ total cost for all levels =

$$1.c(n-1) + 2c(n-2) + 3c(n-3) + \cdots (n-1)c(n-(n-1))$$

$$= c[1(n-1) + 2(n-2) + 3(n-3) + \cdots (n-1)[n-(n-1)]]$$

$$= c[n + 2n + 3n + \cdots (n-1)n - \{1\times1 + 2\times2 + 3\times3 + \cdots (n-1)(n-1)\}]$$

$$\approx c\left[n \times \frac{n(n-1)}{2} - \left(1^2 + 2^2 + 3^2 \cdots + (n-1)^2\right)\right]$$

$$\approx c\left[\frac{n^3}{2} - \frac{n^3}{3}\right] \qquad \left[\because 1^2 + 2^2 + 3^2 + \cdots (n-1)^2 = \frac{(n-1)(n)(2(n-1)+1)}{6} = \frac{(n-1)(n)(2n-1)}{6} \approx \frac{2n^3}{6} \approx \frac{n^3}{3}\right]$$

$$\approx c \times \frac{\cancel{2}n^3}{6}$$

$$\approx n^3$$

which is dominant than the cost for other parts of algorithm (asymptotically)

Q2. Select the set of activities that can be accomplished

Given :

ACTIVITY	1	2	3	4	5	6	7	8	9
START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

Activity selection problem can be solved using Greedy Approach
Our task is to maximise the number of non-conflicting activities

Two activities A1 and A2 are said to be non-conflicting if $s_i \geq f(i-1)^{th}$ activity where s and f denote start and end time respectively.

	1	2	3	4	5	6	7	8	9
START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

A1 selected

	1	2	3	4	5	6	7	8	9
START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

$s_i \leq f(i-1)$ rejected

	1	2	3	4	5	6	7	8	9
START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

$s_i \leq f(i-1)$ rejected

START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

$s_i \geq f(i-1)$ selected

START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	72

$s_i \leq f_{(i-1)}$ rejected

Similar for activities 5, 6, 7

START	1	3	0	5	3	5	6	8	8
FINISH	4	5	6	7	8	9	10	11	12

$s_i \geq f_{(i-1)}$ selected

Last activity is rejected as condition not satisfied.

Activities Selected = $\{A_1, A_4, A_8\}$

ALGORITHM [We are taking sorted activities] TC

```
{                                                                        1
  n = LENGTH (s); // n is total number of activities //                 1
  A = {a1}; // A is set of selected activities and initialized to a1) //  1
  i = 1; // i represents the recently selected activity //             n+1
    for (j = 2; j <= n; j++)
                                                                        n-1
    { if (sj >= fi) {
          A = A ∪ {am};
          i = j;
      }
                                                                        1
    } Return A;
}
```

If we take by step count method & see the count for number of times loop runs when we took sorted array, we get approximately $O(n)$

Time Complexity

For sorting it takes $O(n \log n)$ [if not sorted]

For selecting activities it takes $O(n)$

Total = $O(n \log n) + O(n)$

as $n \log n > O(n)$

so

$T(n) = O(n \log n)$

If it is sorted then $T(n) = O(n)$

Q3. Let us consider two sequences X = (C, R, O, S, S) and Y = (R, O, A, D, S) and the objective is to find the LCS and its length

Given: X = C, R, O, S, S Y = R, O, A, D, S

X↓ \ Y→		R	O	A	D	S
	0	0	0	0	0	0
C	0	0↑	0↑	0↑	0↑	0↑
R	0	1↖	1←	1←	1←	1←
O	0	1↑	2↖	2←	2←	2←
S	0	1↑	2↑	2↑	2↑	3↖
S	0	1↑	2↑	2↑	2↑	3↖

Create a table of dimension $n+1 * m+1$ where m and n are lengths of X and Y

If character corresponding to current row and current column are matching, then fill current cell by adding one to diagonal element.

Else take maximum value from previous column and previous row for filling current cell. Point arrow to cell with maximum value.

After table filled, the value in last row and last column is length of LCS.

Here length of LCS = 3

Longest common subsequence is "ROS"

ALGORITHM LCS_LENGTH (X,Y)

```
{                                              TC
  m = length [X]                               1
  n = length [Y]                               1
  for (i=1; i<=m; i++)                         m
      c [i,0] = 0;
  for (j=0; j<n; j++)                          n
      c [0,j] = 0;
  for (i=1; i<m; i++) {                        m-1
    for (j=1; j<=n; j++) {                     (m-1) n
      if (x[i] == y [j]) {
        c [i,j] = 1+ c [i-1, j-1];
        b [i,j] = '↖';
      else
        if (c [i-1, j] >= c [i, j-1])
          c[i,j] = c [i-1, j];
          b [i,j] = '↑';
        else
          c [i,j] = c [i,j-1];
          b [i,j] = '←'
      }
    return c and b;                            1
  }
```

Asymptotically mn is higher
So TC is $O(mn)$

Time and Space Complexity Analysis

Time complexity = Time complexity of initializing the table
+
Time complexity of filling table in a bottom up manner.

$\therefore$ Time Complexity = $O(m+n) + O(mn)$
$= O(mn)$

Space complexity = $O(mn)$ for storing the table size $(m+1) * (n+1)$

Q4. Let there be n number of objects and each object is having a weight and contribution to profit. Knapsack capacity is M given. Fill the knapsack in such a way that profit will be maximum. We allow a fraction of item to be added to the knapsack.

<u>Given:</u>

Maximum capacity of bag = 20

Item	i_1	i_2	i_3
profit	25	24	15
weight	18	15	10

per kg profit $\frac{25}{18}$ $\frac{24}{15}$ $\frac{15}{10}$

= 1.38 1.6 1.5

Sort according to maximum profit

	x_2	x_3	x_1
	1.6	1.5	1.38
Portion of item kept in bag	1	$\frac{1}{2}$	0
Weight kept in bag	15	$5 (\frac{1}{2} \times 10)$	0

$\therefore$ Total weight

$$\Sigma w_i x_i = (18 \times 0) + (15 \times 1) + (10 \times \tfrac{1}{2})$$
$$= 20.$$

$\therefore$ Total profit

$$\Sigma P_i x_i = (25 \times 0) + (24 \times 1) + (15 \times \tfrac{1}{2})$$
$$= 31.5$$

ALGORITHM

1. Take the ratio of profits / weights $\quad 1$
2. Arrange ratios in decreasing order of profit $\quad n\log n$
3. Start selecting items to the knapsack $\quad n$

   if $W_i < M$ then change profit to $P = P + P_i$

   if $(W_i \le m)$ {

   max_profit = max_profit + P_i

   $M = M - W_i$

   }

   else {

   max_profit = max_profit + $P_i \times \frac{M}{W_i}$

   $m = 0$;

   }
4. Then return max_profit $\quad 1$

Asymptotically $n\log n$ is more
$\therefore$ TC is $O(n\log n)$

Time Complexity

Sorting step : $T_{sort}(n)$
Sorting items based on value to weight ratio is typically $O(n \log n)$ using efficient sorting algorithms like quick sort or merge sort

Greedy selection step : It involves iterating through the sorted items and making decisions based on the value to weight ratio. It is of $O(n)$.

$\therefore$ Overall time complexity $= O(n\log n) + O(n)$

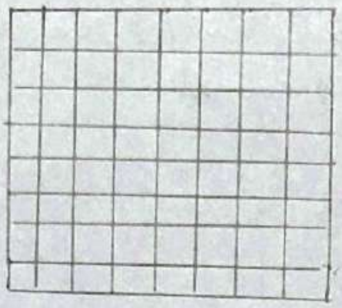
Since $n \log n > n$

$\therefore T(n) = O(n \log n)$

Q5) Let us consider there are 8 queens in the chessboard (8×8), place the queens in appropriate position with the following conditions (Backtracking) : Not more than one queen in the same row, in the same column, in their diagonal.

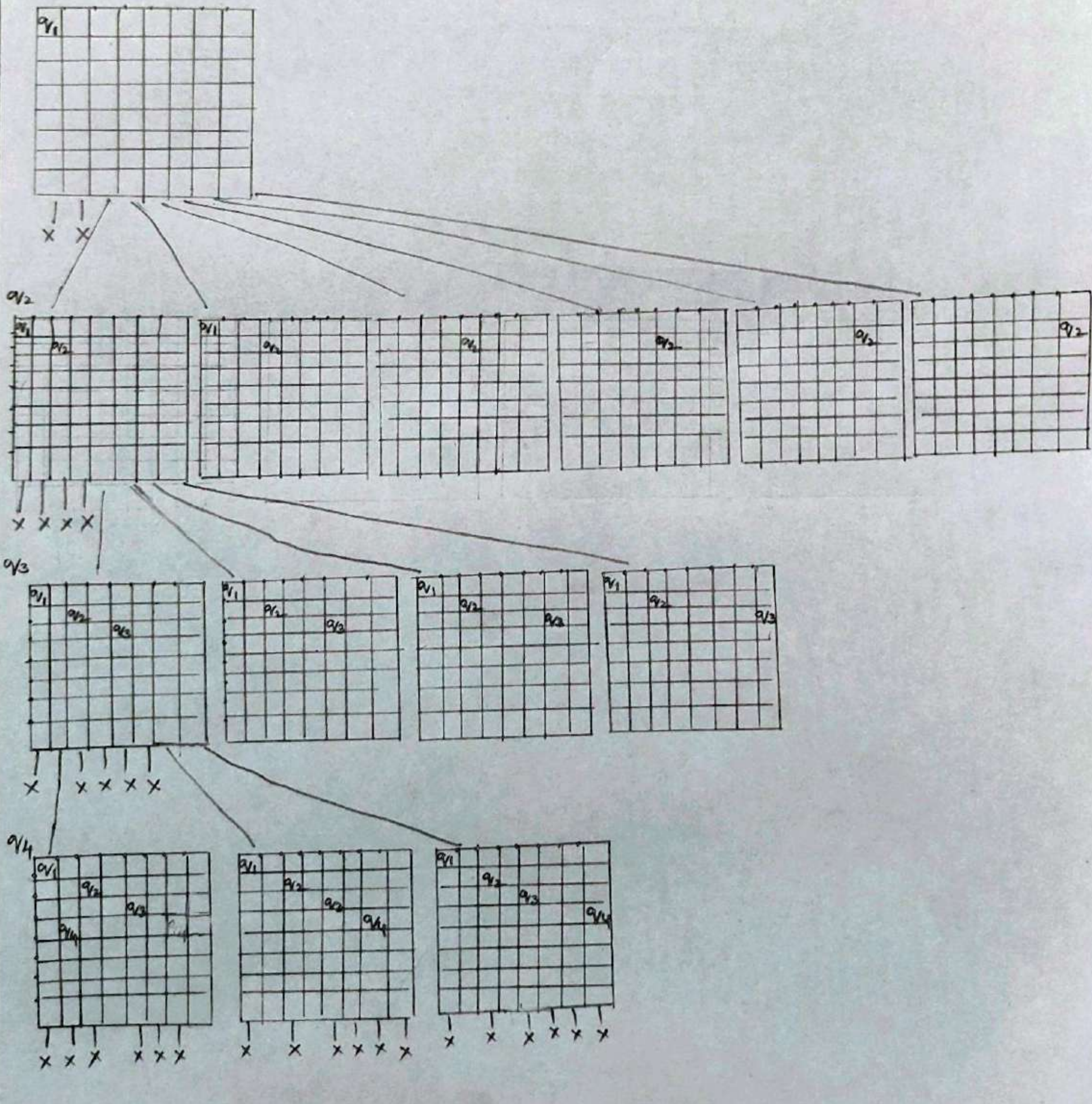
State Space Tree

q_1 placed in all columns of first row



8×8 matrix

No two queens are on same row, column or diagonal



Two types of constraints

Explicit constraints $\left\{ \begin{array}{l} S_i = \{1,2,3,4,5,6,7,8\} \\ x_i : 1 \leq i \leq 8 \end{array} \right\}$

Implicit constraints : No two queens are on same column, row or diagonal.

<u>Solution</u>:

	1	2	3	4	5	6	7	8
1				q_1				
2						q_2		
3								q_3
4		q_4						
5							q_5	
6	q_6							
7			q_7					
8					q_8			

ALGORITHM is queen safe (board, row, col)

```
for i from 0 to col - 1 do
    if board [row] [i] == Q then
        return false
  for i, j from row, col to 0, 0 to step 1 do
      if board [i] [j] == Q then
          return false
   for i, j from row, col to N-1, 0, to step 1 do
       if board [i] [j] == Q then
           return false
   else
  return true
```

Time Complexity Analysis

Back tracking approach reduces time complexity as it eliminates dead ends.

Main program calls function queen (n) which calls function place (k). 'k' checks already placed queens. Since function queen iterates n times and for each iteration it calls function place (row) function, the time complexity would be $O(N^2)$.

Queen (k, n) is recursive function. So $T(n-1)$ will run n times because it runs only for safe cells.

We started by filling up rows so there won't be more than n safe cells in the row.

$$
\begin{array}{ccccccc}
 & O(N^2) & & O(N^2) & & O(N^2) & \\
T(n) \longrightarrow & \searrow & \longrightarrow & \searrow & \longrightarrow & \searrow & \longrightarrow \ldots \\
 & n^* T(n-1) & & n^* O(N^2) & & n^* O(N^2) & \\
 & & & \downarrow & & \downarrow & \\
 & & & n(n-1)\, T(n-2) & & n(n-1)\, O(N^2) & \\
 & & & & & \downarrow & \\
 & & & & & n(n-1)(n-2)\, T(n-3) &
\end{array}
$$

Replacing $T(n-1)$ with $O(N^2) + (n-1)\,T(n-2)$

$$T(n) = O(N^2) + n\,(O(N^2) + (n-1)\,T(n-2))$$
$$= O(N^2) + n\,(ON^2) + (n-1)\,T(n-2))$$

Replacing $T(n-2)$ with $O(N^2) + (n-2)\,T(n-3)$

$$T(n) = O(N^2) + n\,O(N^2) + n(n-1)\,(O(N^2) + (n-2)\,T(n-3))\,T(n)$$
$$= O(N^2) + n\,O(N^2) + n(n-1)\,(O(N^2) + (n-2)\,T(n-3))$$

Similarly

$$T(n) = O(N^2)\,(1 + n + n(n-1) + n(n-2) + \cdots) + n(n-1)(n-2)(n-3)(n-4)$$
$$T(0)\,T(n)$$
$$= O(N^2)\,(O((n-2)!)) + n(n-1)(n-2)(n-3)\ldots\ T(0)\,T(n) = O(N^2)$$
$$(O((n-2)!)) + n(n-1)(n-2)(n-3)\,T(0)$$
$$= O(N^2)\,(O((n-2)!)) + O(n!) = O(n!)$$

$\therefore$ Time complexity $= O(n!)$