1) Multiply a chain of matrices

A	A	A	A	A	A
7×4	4×3	3×12	12×3	3×5	5×50
d_0 d_1	d_1 d_2	d_2 d_3	d_3 d_4	d_4 d_5	d_5 d_6

$$N_{i,j} = \min_{i \le k \le j} \{ N_{i,k} + N_{k+1,j} + d_i \times d_{k+1} \times d_{j+1} \}$$

sequence = { 7, 4, 3, 12, 3, 5, 50 }

Algorithm Matrix Chain ($d_0, \ldots, d_n$):
Input: Sequence $d_0, \ldots, d_n$ of integers
Output: For $i, j = 0, \ldots n-1$, the minimum number of multiplications N_{ij} needed to compute the product $A_i \cdot A_{i+1} \cdots A_j$ where A_k is a $d_k \times d_{k+1}$ matrix

```
for i ← 0 to n-1 do
    N[i,i] ← 0
for b ← 1 to n-1 do
    for i ← 0 to n-b-1 do
        j ← i+b
        N[i,j] ← +∞
        for k ← i to j-1 do
            N[i,j] ← min{ N[i,j], N[i,k] + N[k+1,j] + d[i]d[k+1]d[j+1] }
```

∴ we have $d_0 = 7$ $d_1 = 4$ $d_2 = 3$ $d_3 = 12$
$d_4 = 3$ $d_5 = 5$ $d_6 = 50$

6	5	4	3	2	1	
2028	308	228	336	84	0	1
1168	168	144	144	0		2
903	153	108	0			3
2550	180	0				4
750	0					5
0						6

(1,1) ..... (6,6) are initialized to '0'

$$m[1,2] = \min_{\substack{K = 1 \text{ to } 2 \\ i \le k \le j-1}} \left\{ m[1,1] + m[2,2] + d_0 \times d_1 \times d_2 \right\}$$

$$= \min \{ 0 + 0 + 7 \times 4 \times 3 \} = 84$$

$$m[2,3] = \min \{ m[2,2] + m[3,3] + d_1 d_2 d_3 \}$$

$$= \min \{ 0 + 0 + 4 \times 3 \times 12 \} = 144$$

$$m[3,4] = \min \{ m[3,3] + m[4,4] + d_2 d_3 d_4 \}$$

$$\min \{ 0 + 0 + 3 \times 12 \times 3 \} = 108$$

$$m[4,5] = \min \{ m[4,4] + m[5,5] + d_3 d_4 d_5 \}$$

$$\min \{ 0 + 0 + 12 \times 3 \times 5 \} = 180$$

$$m[5,6] = \min \{ m[5,5] + m[6,6] + d_4 d_5 d_6 \}$$

$$\min \{ 0 + 0 + 3 \times 5 \times 50 \} = 750$$

$$n[1,3] = \min_{k=1 \text{ to } 2} \begin{cases} n[1,1] + n[2,3] + d_0 d_1 d_3 \rightarrow k=1 \\ n[1,2] + n[3,3] + d_0 d_2 d_3 \rightarrow k=2 \end{cases}$$

$$\min \begin{cases} 0 + 144 + 7 \times 4 \times 12 \rightarrow 480 \\ 84 + 0 + 7 \times 3 \times 12 \rightarrow 336 \end{cases}$$

$= 336 \rightarrow (k=2)$

∴ The parenthesis will (1 2)(3) means it is 1 to 2 and 3. So the minimum number $k=2$ which is 108.

$$n[2,4] = \min_{k=2 \text{ to } 3} \begin{cases} n[2,2] + n[3,4] + d_1 d_2 d_4 \rightarrow (k=2) \\ n[2,3] + n[4,4] + d_1 d_3 d_4 \rightarrow k=3 \end{cases}$$

$$= \min \begin{cases} 0 + 108 + 4 \times 3 \times 3 = 144 \\ 144 + 0 + 4 \times 12 \times 3 = 288 \end{cases} = 144 \quad (k=2)$$

$$n[3,5] = \min_{k=3 \text{ to } 4} \begin{cases} n[3,3] + n[4,5] + d_2 d_3 d_5 \\ n[3,4] + n[5,5] + d_2 d_4 d_5 \end{cases}$$

$$= \min \begin{cases} 0 + 180 + 3 \times 12 \times 5 = 360 \\ 108 + 0 + 3 \times 3 \times 5 = 153 \end{cases} = 153 \quad (k=4)$$

$$n[4,6] = \min_{k=4 \text{ to } 5} \begin{cases} n[4,4] + n[5,6] + d_3 d_4 d_6 \\ n[4,5] + n[6,6] + d_3 d_5 d_6 \end{cases}$$

$$= \min \begin{cases} 0 + 750 + 12 \times 3 \times 50 = 2550 \\ 180 + 0 + 12 \times 5 \times 50 = 3180 \end{cases}$$

$= 2550$

$(k=4)$

$$m[1,4] = \min_{k=1 \text{ to } 3} \begin{cases} m[1,1] + m[2,4] + d_0 d_1 d_4 \\ m[1,2] + m[3,4] + d_0 d_2 d_4 \\ m[1,3] + m[4,4] + d_0 d_3 d_4 \end{cases}$$

$$= \min \begin{cases} 0 + 144 + 7 \times 4 \times 3 = 228 \\ 84 + 108 + 7 \times 3 \times 3 = 255 \\ 336 + 0 + 7 \times 12 \times 3 = 588 \end{cases}$$

$$= \min (228) \quad (k=1)$$

$$m[2,5] = \min_{k=2 \text{ to } 4} \begin{cases} m[2,2] + m[3,4] + d_1 d_2 d_5 \\ m[2,3] + m[4,5] + d_1 d_3 d_5 \\ m[2,4] + m[5,5] + d_1 d_4 d_5 \end{cases}$$

$$= \min \begin{cases} 0 + 108 + 4 \times 3 \times 5 = 168 \\ 144 + 180 + 4 \times 12 \times 5 = 564 \\ 144 + 0 + 4 \times 3 \times 5 = 204 \end{cases}$$

$$= \min (168) \quad k=2$$

$$m[3,6] = \min_{k=3 \text{ to } 5} \begin{cases} m[3,3] + m[4,6] + d_2 d_3 d_6 \\ m[3,4] + m[5,6] + d_2 d_4 d_6 \\ m[3,5] + m[6,6] + d_2 d_5 d_6 \end{cases}$$

$$= \min \begin{cases} 0 + 2550 + 3 \times 12 \times 50 = 4350 \\ 108 + 750 + 3 \times 4 \times 50 = 1458 \\ 153 + 0 + 3 \times 5 \times 50 = 903 \end{cases}$$

$$= \min (903) \quad k=5$$

$$n[1,5] = \min_{k=1 \text{ to } 5} \begin{cases} n[1,1] + n[2,5] + d_0 d_1 d_5 \\ n[1,2] + n[3,5] + d_0 d_2 d_5 \\ n[1,3] + n[4,5] + d_0 d_3 d_5 \\ n[1,4] + n[5,5] + d_0 d_4 d_5 \end{cases}$$

$$= \begin{cases} 0 + 168 + (7 \times 4 \times 5) = 308 \\ 84 + 153 + (7 \times 3 \times 5) = 342 \\ 336 + 180 + (7 \times 12 \times 5) = 936 \\ 228 + 0 + (7 \times 3 \times 5) = 333 \end{cases}$$

$$\min = 308 \quad (k = 1)$$

$$n[2,6] = \min_{k=1 \text{ to } 6} \begin{cases} n[2,2] + n[3,6] + d_1 d_2 d_6 \\ n[2,3] + n[4,6] + d_1 d_3 d_6 \\ n[2,4] + n[5,6] + d_1 d_4 d_6 \\ n[2,5] + n[6,6] + d_1 d_5 d_6 \end{cases}$$

$$= \begin{cases} 0 + 903 + (4 \times 3 \times 50) = 1503 \\ 144 + 2550 + (4 \times 12 \times 50) = 5094 \\ 144 + 750 + (4 \times 3 \times 50) = 1494 \\ 168 + 0 + (4 \times 5 \times 50) = 1168 \end{cases}$$

$$\min = 1168 \quad (K = 6)$$

$$m[1,6] = \min_{k=1 \text{ to } 6} \begin{cases} m[1,1] + m[2,6] + d_0 d_1 d_6 \\ m[1,2] + m[3,6] + d_0 d_2 d_6 \\ m[1,3] + m[4,6] + d_0 d_3 d_6 \\ m[1,4] + m[5,6] + d_0 d_4 d_6 \\ m[1,5] + m[6,6] + d_0 d_5 d_6 \end{cases}$$

$$= \min \begin{cases} 0 + 1168 + 7 \times 4 \times 50 = 2568 \\ 84 + 903 + 7 \times 3 \times 50 = 2037 \\ 336 + 2550 + 7 \times 12 \times 50 = 7086 \\ 228 + 750 + 7 \times 3 \times 50 = 2028 \\ 308 + 0 + 7 \times 5 \times 50 = 2058 \end{cases}$$

$\min = 2028 \quad (k = 5)$

→ So minimum multiplication required is 2028

∴ Now we get the final parenthesis for the calculated matrix.

1st : $m[1,6]$ where $k = 5$

So $k=5$ we divide from A_5 which says $m[1,5]$ and $m[6,6]$. So parenthesis

$((A_1 A_2 A_3 A_4 A_5)(A_6))$

2nd : $m[1,5]$ where $k = 1$

So $k=1$ we divide from A_1 which says $m[1,1]$, $m[2,5]$ and $m[6,6]$. So parenthesis

$[((A_1)(A_2 A_3 A_4 A_5))(A_6)]$

3rd : $n(2,5)$ where $k = 2$

So $k=2$ we divide from A_2 which says $n[1,1]$, $n[2,2]$, $n[3,5]$, and $n[6,6]$. So parentheis $[((A_1)((A_2)(A_3\ A_4\ A_5)))\ A_6)]$

4th : $n(3,5)$ where $k = 4$

So $k=4$ we divide from A_4 which says $n[1,1]$, $n[2,2]$, $n[3,4]$, $n[5]$ and $n[6,6]$. So parentheis $[((A_1)((A_2)(((A_3\ A_4))\ (A_5)))(A_6)]$

So therefore the order in which we get the optimal number of multiplications is

→ $[((A_1)((A_2)(((A_3\ A_4))\ (A_5))))\ (A_6)]$

2) Design an efficient algorithm for the matrix chain multiplication problem that outputs a fully parenthesized expression for how to multiply the matrices in the chain using the minimum number of operations. Recall that one of the notes show how to keep track of back pointer s[i,j]'s, use these to output the fully parenthesized expression.

Algorithm:

→ defining variable value [], i, j
→ check if the value of i = j
→ declaring the kth element for the matrix
→ Recursive loop for kth element until increment
→ Matrix chain multiplication function
→ Int min assign to find the minimum
→ check the function again and find the minimum value among others
→ Assigning the value to the function and finding the minimum
→ Return the last minimum value.

Pseudo Code :-

```
int MatrixChainMultiplication (int value[], int i, int j)
{ if (i == j)
        return 0;
    int kth;
    int minimum = Intmin;
    int counter;
    for (kth = i; kth < j; kth++)
    {
        counter =
MatrixChainMultiplication (value, i, kth) +
        MatrixChainMultiplication (value, kth+1, j) +
            value (i-1 * value [kth] * value [j];
        if (counter < minimum)
            minimum = counter;
    }
    return minimum;
}
```

3. Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit - weights values, a: (10, 4), b: (8, 6), c: (9, 4), d: (12, 6), e: (13, 4), f: (6, 1), g: (9, 2). What is an optimal solution to the 0-1 knapsack problem for S assuming we have a sack that can hold objects with total weight 14? Show your work using 2 different solutions. First is the naive solution where one enumerates the possible subproblems and second is the DP formulation as discussed in class.

→ Given $S = \{a, b, c, d, e, f, g\}$

benefits = {10, 8, 9, 12, 13, 6, 9}

weights = {4, 6, 4, 6, 4, 1, 2}

Total weight of the bag = 14

The 2 methods of solving the 0-1 knapsack problem by fill the maximum benifit possibe with the weight 14.

⟹ Going for dynamic approach -

→ making an array [i][j] which is filling in bottom approach.

→ i denotes the rows and j is the coloumn.

→ Saying Array [a] [c] representing as j = c, Array [a][c] and finding max of (Array[i][c-1] for i = {1 | 4 of Array[a][c-1] + Array[a-1][c-a]

→ Making an Array [i][j] for the given weight of benefit.

P_i	W_i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	4	1	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10
8	6	2	0	0	0	0	10	10	8	8	8	8	18	18	18	18	18
9	4	3	0	0	0	0	9	9	9	9	19	19	19	19	19	19	27
12	6	4	0	0	0	0	0	0	12	12	12	12	22	22	22	22	31
13	4	5	0	0	0	0	13	13	13	13	22	22	25	25	32	32	35
6	1	6	0	6	6	6	13	19	19	19	22	29	29	31	32	38	38
9	2	7	0	6	9	15	15	19	22	28	28	29	32	38	38	40	41

A simpler Equation :-

$$V = [i, w] = \max \left[V[i-1, w], V[i-1, w - w[i]] + P[i] \right]$$

$w = 7$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	0	1	0	1

(x_4, x_5, x_6, x_7) is included.

Total profit = 41

→ The time complexity = $O(N \times W)$
where N is max weight &
W is the knapsack bag capacity.

→ Dynamic approach is the best as it reduce work load to find an optimized solution.

⇒ (P,W) Naive Approach.

$S^0 = \{0,0\}$

$S_1^0 = \{10,4\}$

$S^1 = \{(0,0), (10,4)\}$

$S_1^1 = \{(8,6), (18,10)\}$

$S^2 = \{(0,0), (10,4), (8,6), (18,10)\}$

$S_1^2 = \{(9,4), (19,8), (17,10), (27,14)\}$

$S^3 = \{(0,0), (10,4), (8,6), (18,10), (9,4), (19,8), (17,10), (27,14)\}$

└ Dominance rule

$S_1^3 = \{(12,6), (22,10), (30,16), (30,16), (31,14), (29,16), (39,20)\}$

exceeding bag ┘

$S^4 = \{(0,0)\ (10,4)\ (8,6)\ (18,10), (19,8), (17,10), (27,14), (12,6), (22,10), (30,16)$
$(31,14)\}$

$S_1^4 = \{(13,4), (23,8), (21,10), (31,14), (30,14), (40,18), (25,10), (35,14), (44,18)\}$

$S^5 = \{(0,0)\ (10,4)\ (8,6)\ (18,10), (19,8), (17,10), (27,14), (12,6), (22,10), (31,14), (13,4), (23,8)$
$(21,10)\ (31,14)\ (30,14)\ (25,10)\ (35,14)\}$

$S_1^5 = \{(13,4), (23,8), (21,10), (31,14), (32,12), (30,14), (40,18), (35,10), (35,14), (44,18), (36,8),$
$(36,12), (34,14), (44,18), (43,18), (28,14), (48,18)\}$

$S^6 = \{(0,0)\ (10,4)\ (8,6)\ (18,10)\ (19,8)\ (17,10)\ (27,14)\ (12,6), (22,10), (31,14), (13,4)\ (23,8)\ (21,10),$
$(13,4), (23,8), (31,14), (32,12), (30,14), (35,10), (35,14), (26,8), (36,12), (34,14),$
$(31,14)\ (30,14)\ (25,10)\ (35,14)\ (28,14)\}$

$S_1^6 = \{(6,1)\ (16,5)\ (14,7)\ (26,9)\ (23,11)\ (33,15)\ (18,7)\ (28,11)\ (37,15)\ (19,5)\ (29,9)$
$(19,5)\ (29,9)\ (37,15)\ (38,13)\ (48,11)\ (47,20)\ (32,9)\ (41,16)\ (40,15)\ (37,15)\ (36,15)$
$(31,11)\ (41,15)\}$

$S^7 = \{(0,0)(10,4)(8,6)(19,8)(19,10)(27,14)(12,6)(22,10)(31,14)(13,4)(23,8)$
$(13,4)(3,4)(23,8)(13,4)(23,8),(31,14)(32,12)(35,10)(35,14)$
$(26,8)(36,12)(34,14)(31,14)(30,14)(25,10)(35,14)(6,1)$
$(16,5)(14,7)(25,9)(23,11)(33,15)(18,7)(28,11)(19,5)$
$(29,9)(19,5)(22,9)(37,15)(38,13)(40,13)(40,20)(32,9)$
$(41,15)(37,15)(31,11)\}$

$S_1^7 = \{(9,2),(19,8),(16,8)(28,10)(31,12)(36,16)(21,8)(41,14)$
$(40,16)(22,6)(32,10)(41,16)(44,16)(35,10)$
$(44,16)(43,16)(39,16)(34,12)(15,3)(25,7)(23,9)$
$(34,11)(32,13)(27,9)(37,13)(28,7)(38,11)(43,11)$
$(49,15)(41,12)(40,13)$

$S^8 = \{(0,0)(10,4)(8,6)(19,8)(19,10)(27,14)(12,6)(31,14)(13,4)(23,8)$
$(32,12)(35,10)(26,8)(36,12)(34,14)(30,14)(25,10)(6,1)$
$(16,5)(14,7)(25,9)(23,11)(18,7)(28,11)(19,5)(29,9)$
$(38,13)(40,13)(32,9)(31,11)(9,2)(19,6)(16,8)(28,10)(31,12)$
$(36,16)(21,8)(41,14)(22,6)(32,10)(35,10)(34,12)(15,3)$
$(25,7)(23,9)(34,11)(32,13)(27,9)(37,13)(28,7)(38,11)$
$(41,12)(40,13)$

→ Checking by reverse method :-

① $(41,14) \in S^8$

but $(41,14) \notin S^7$

$[(41-9),(14-2)]$

$= (32,12)$

② $(32,12) \in S^6$

$(32,12) \notin S^5$

$[(32-13),(12-4)]$

③ $(19,8) \in S^4$

$19,8 \notin S^3$

④ $[(19-9),(8-4)]$

$(10,4) \in S^2$

$(10,4) \notin S^1$

⑤ $[(10-10),(4-4)]$

$(0,0) \in S^0$

→ This naive approach checks all porible sets for the give benefit of theight. Then we check which subset has the weight that are equal to the knapsack capacity.

→ This exponential growth 2^n subsets is very time consuming so the time complexity is $O(2^n)$.

The knapsack benefit is = 41.

4) Sally is hosting an Internet auction to sell n widgets. She receives m bids, each of the form "I want k_i widgets for d_i dollars," for $i = 1, 2 \ldots m$. Characterize this optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?

Answer.

→ So recalling what knapsack fractional is where we can break items to get the maximum profit and by using the greedy approach.

→ whereas for the 0-1 knapsack problem we cannot break items for the maximum profit. It is a take it or leave it situation kinda thing. That is where Dynamic programming comes in.

→ So Sally was hosting an Internet auction where if she were to implement a fractional knapsack the bid can break for the maximum total value of the knapsack if the all the bids are in a d_i/k_i from the given statement is arranged in decending order and start selecting from the highest ratio.

→ If Sally were to take 0-1 knapsack the last maximum bid is taken if there is space or left if not in the knapsack. The widget must fullfil for a bid to be considered. This approach is used for dynamic programming to find optimized.

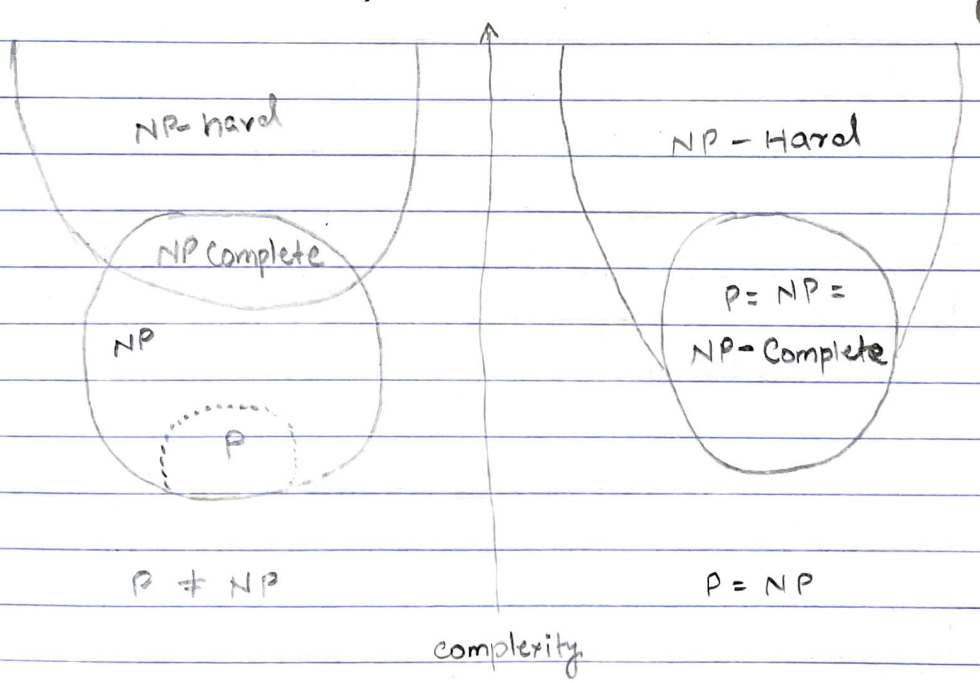
5) Implement the backtracking algorithm of the n-queens problem on page 2 in notes of Erickson @ Illinois using your favorite high-level language. Tabulate and plot the time taken by your implementation for various values of n. Find the maximum value of n your code can handle on your machine. For each n, how many nodes of the recursion tree does your algorithm explore before finding a solution? tabulate these values.

→ The code is attached with the type script in the zip file.

Thank You.

6) Problem R-17.1 of the GT book: Professor Amongus has shown that a decision problem L is polynomial-time reducible to an NP-complete problem M. Moreover, after 80 pages of dense machimatics, he has also just proven that L can be solved in polynomial time. Has he just proven that P = NP? Why, or why not? No credit without proper justification.

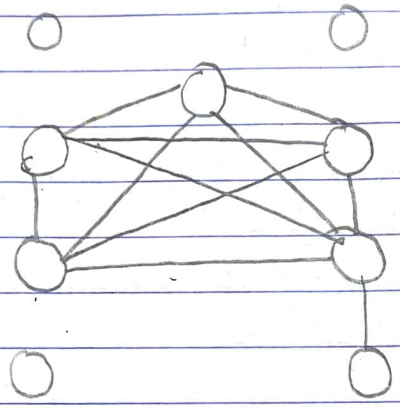
→ The Eucler diagram for P, NP, NP-complete and NP-hard problem. There are 2 assumptions where P = NP & P ≠ NP



→ So, he has proved that L can be solved in polynomial time. P = NP is valid, but as discussed in class not every Polynomial time problem of NP is NP complete as any problem in NP has a Polynomial - time reduction to every NP complete problem. There can be only one out of every odd issue in P or NP which will be NP Complete

7. Similar to [Problem R-17.11 of the GT book: Draw an example of a graph with 9 vertices and 11 edges that has a clique of size 5.

→ Clique is the subgraph of a graph which is complete.



$|V| = 9 \quad |E| = 11 \quad K = 5$

→ So Complete graph = $[n \times (n-1)] / 2$ where n is the vertices.

Finding the clique = $(5 \times (5-1)] / 2$

$= \frac{5 \times 4}{2} = 10$ //

→ So there must be 10 vertices present the rest 4 can be left as it is. And we have 10 edges in the clique where in the question we have 11 so, one edge is left as it is

8. Similar to problem R-17.12 of the GT book: Professor Amongus has just designed an algorithm that can take any graph G with n vertices and determine in $O(n^s)$ time whether G contains a clique of size s. Does Professor Amongus deserve the Turing Award for having just shown that P=NP? Why or why not.

→ So to prove we need to show that clique belong to NP-Complete, we show that by proving it to NP & NP-hard.

→ For NP if that problem belongs to NP then it should verify in polynomial time. By checking the problem has a solution in the polynomial time.

→ So let a graph be a set of N nodes in the clique where the N is the subgraph of the graph G. So now we can check if the graph G has the S subgraph and the complexity to check takes O(1) time. It requires $O(n^s)$ time to check if each vertex has an out degree $(n-1)$. We know that each vertex in a complete graph is connected through an edge so the formula ${}^{n}C_2$ where $n=(n-1)/2$ takes $O(n^2)$ time to check the complete graph.

∴ The Clique Decision Problem has polynomial time and it belongs to NP

→ So to prove we also need to proof that clique problem is NP-hard,
By taking a solved NP-hard problem we can reduce to clique problem.
We know that Independent Set problem is NP-Complete ∴ NP-hard.
Assume we have a graph G(V,E) ∴ S ⊆ V is an independent set if there are no edges between vertices in S taking an integer K ≤ |V| so the G have an independent set of size S and considering this an NP problem
So Ind. set ⟺ clique in complement.
→ ∴ The time complexity of this will be O(V+E)
→ We can further reduce it down by 2 methods.
→ The graph G has a clique k and every vertices is connected by an edge with the left over vertices. So if these edges are present in G then they wont be in G compliment. by this we can say that K vertices are not adjacent in the G compliment that forms Independent Set of size S.
→ we can also complement the G which has independent sets. when we do that these K vertices will share an edge and will be adjacent to each other. then the graph G will have a S size clique.

→ Therefore any instance of clique can be reduced to an instance of the clique.
Thus the clique will be NP- hard.

→ Finally : The clique decision problem is NP- Complete and he deserve the award for the algorithm to solve clique problem in $O(n^5)$ by this we can say $N = NP$ which is NP Complete.