Name : Sonal Dilip Gholap

UID : 2021600020

Branch : CSE-AIML

Batch : B

Experiment : 4

Aim : Experiment on dynamic programming approach.

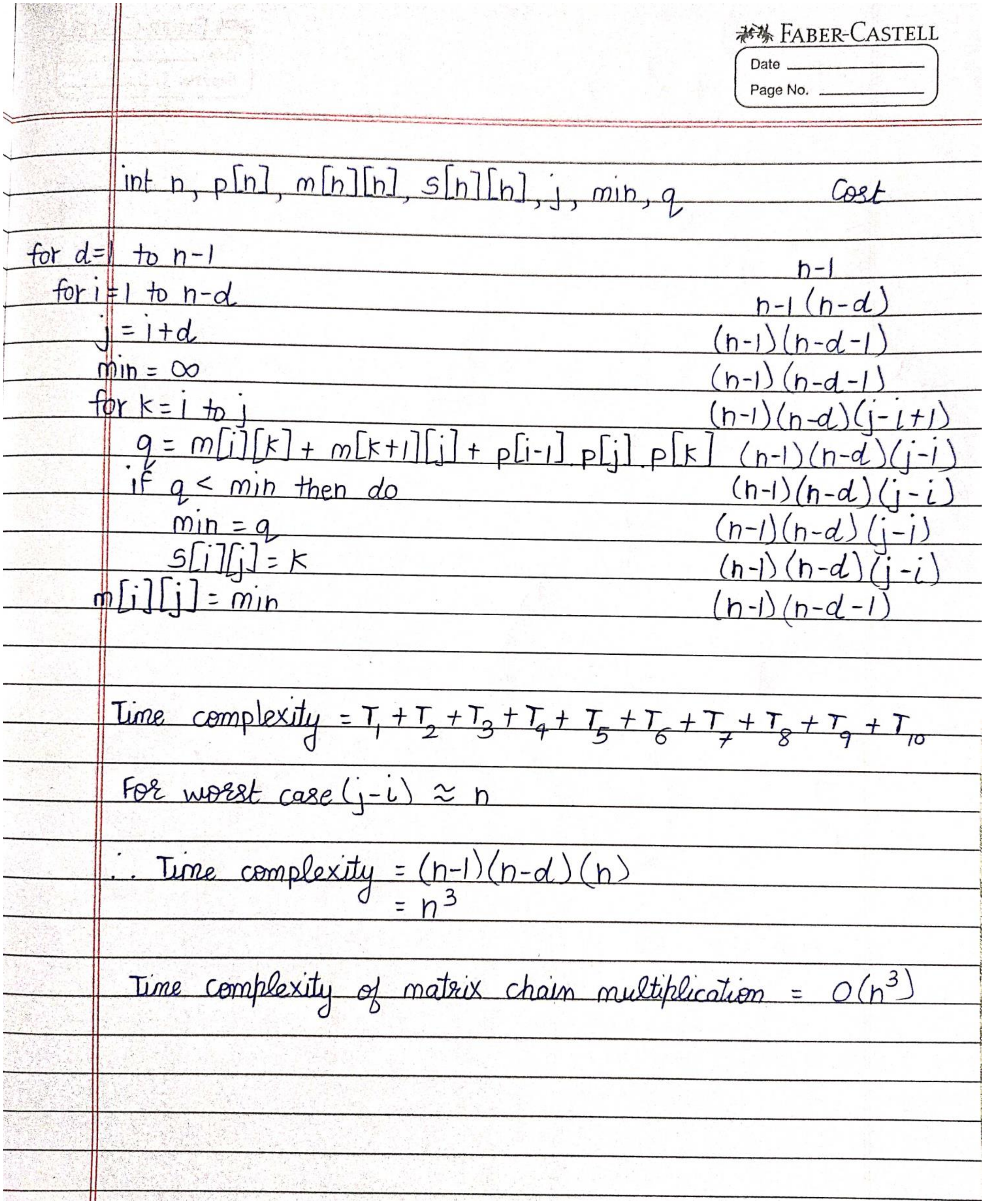
- **Objectives :**

1. To study Dynamic Programming Approach by implementing Matrix Chain Multiplication.
2. To understand the Overlapping Subproblems property and the Optimal Substructure property.
3. To derive the time complexity of Matrix Chain Multiplication.
4. To understand the difference between Base case and Recursive case.

- **Theory :**

1. Overlapping Subproblems property – This property implies that a subproblem is repeated. Unless the solution to a subproblem is used again, there is no use of storing it.
2. Optimal Substructure property – This property implies that the optimal solution of a problem can be found by using the solution of the optimal subproblem.
3. In Matrix Chain Multiplication, the outer loop (the d loop) will only run for the diagonal elements and the elements above it.
4. Initially, the 2d matrix m is initialized to 0.
5. Using the formula, we find the minimum computed value and store it in m[i][j] and store the corresponding k value in s[i][j]
6. The desired output is stored in m[1][n-1]
7. Whenever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using dynamic programming. The idea is to simply store the results of the subproblems so that we do not have to recompute them when needed later.

- **Algorithm and Derivation of Time Complexity**



int n, p[n], m[n][n], s[n][n], j, min, q	Cost
for $d=1$ to $n-1$	$n-1$
for $i=1$ to $n-d$	$n-1(n-d)$
$j=i+d$	$(n-1)(n-d-1)$
$min=\infty$	$(n-1)(n-d-1)$
for $k=i$ to j	$(n-1)(n-d)(j-i+1)$
$q = m[i][k] + m[k+1][j] + p[i-1].p[j].p[k]$	$(n-1)(n-d)(j-i)$
if $q < min$ then do	$(n-1)(n-d)(j-i)$
$min = q$	$(n-1)(n-d)(j-i)$
$s[i][j] = k$	$(n-1)(n-d)(j-i)$
$m[i][j] = min$	$(n-1)(n-d-1)$

Time complexity $= T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7 + T_8 + T_9 + T_{10}$

For worst case $(j-i) \approx n$

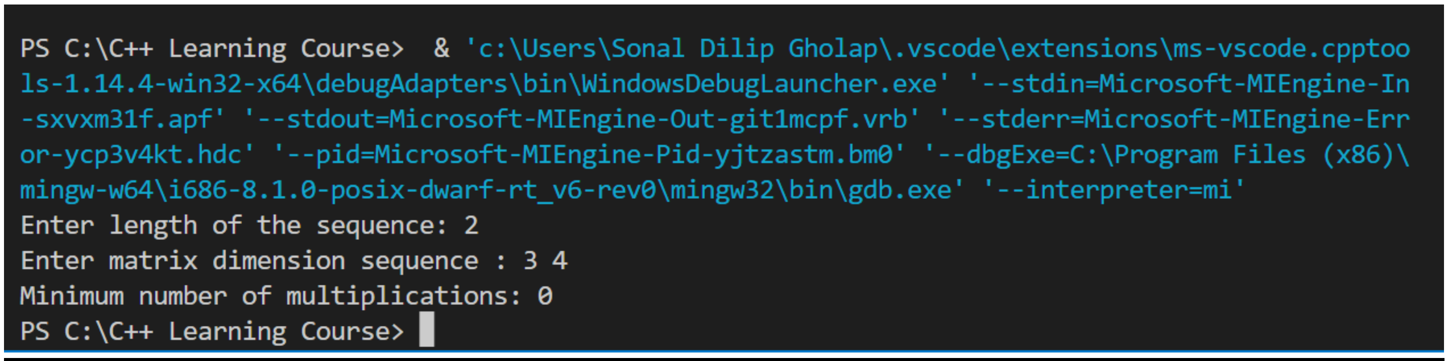
$\therefore$ Time complexity $= (n-1)(n-d)(n)$
$= n^3$

Time complexity of matrix chain multiplication $= O(n^3)$

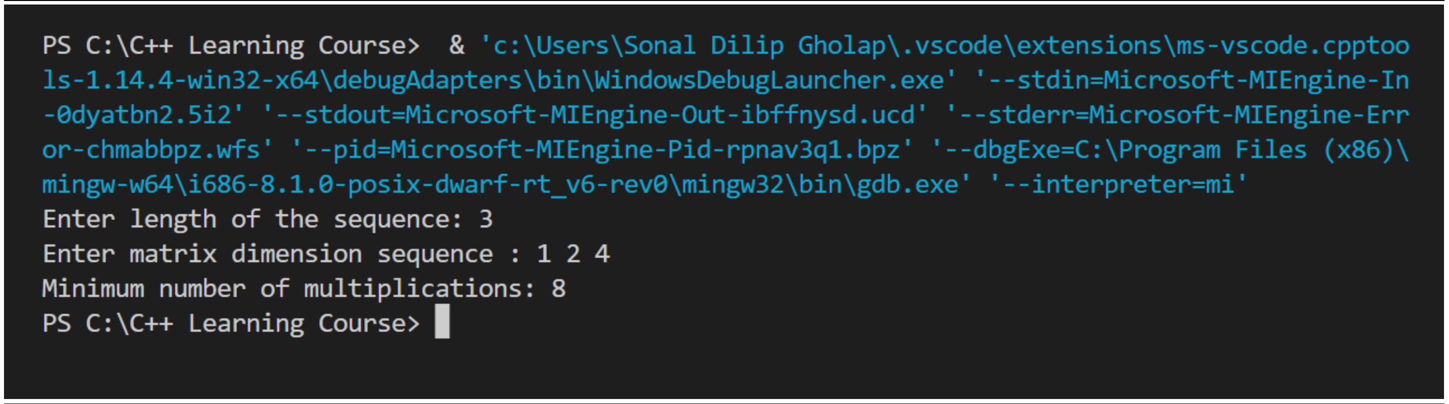
Code :

```cpp
#include<iostream>
#include <bits/stdc++.h>

using namespace std;

int main(){

    cout<<"Enter length of the sequence: ";
    int n;
    cin>>n;
    int a[n];
    cout<<"Enter matrix dimension sequence : ";
    for(int i=0;i<n;i++){
        cin>>a[i];
    }

    int m[n][n];
    int kval[n][n];
    memset(m,0,sizeof(m));
    memset(kval,0,sizeof(kval));

    for(int d=1;d<n-1;d++){                     //will run for the upper diagonals only
        for(int i=1;i<n-d;i++){                 //for row number
            int j = i+d;                        //for column number
            int min = INT_MAX;

            for(int k=i;k<=j-1;k++){
                int q = m[i][k] + m[k+1][j] + a[i-1]*a[k]*a[j];

                if(q<min){
                    min=q;
                    kval[i][j]=k;
                }
            }
            m[i][j]=min;
        }
    }

    cout<<"Minimum number of multiplications: "<<m[1][n-1];

    return 0;
}
```

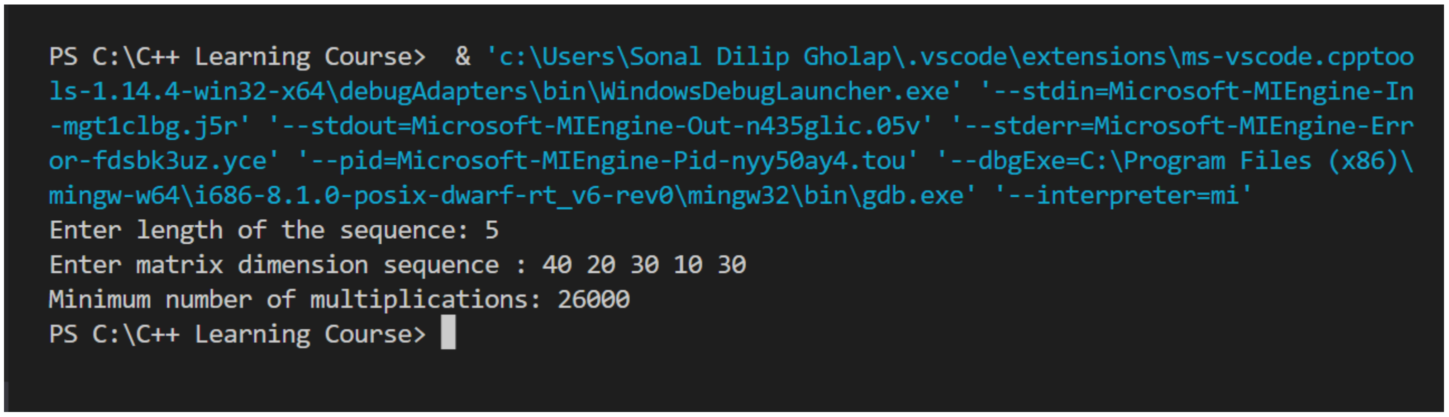
Output :



```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptoo
ls-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In
-sxvxm31f.apf' '--stdout=Microsoft-MIEngine-Out-git1mcpf.vrb' '--stderr=Microsoft-MIEngine-Err
or-ycp3v4kt.hdc' '--pid=Microsoft-MIEngine-Pid-yjtzastm.bm0' '--dbgExe=C:\Program Files (x86)\
mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Enter length of the sequence: 2
Enter matrix dimension sequence : 3 4
Minimum number of multiplications: 0
PS C:\C++ Learning Course>
```



```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptoo
ls-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In
-0dyatbn2.5i2' '--stdout=Microsoft-MIEngine-Out-ibffnysd.ucd' '--stderr=Microsoft-MIEngine-Err
or-chmabbpz.wfs' '--pid=Microsoft-MIEngine-Pid-rpnav3q1.bpz' '--dbgExe=C:\Program Files (x86)\
mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Enter length of the sequence: 3
Enter matrix dimension sequence : 1 2 4
Minimum number of multiplications: 8
PS C:\C++ Learning Course>
```



```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptoo
ls-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In
-mgt1clbg.j5r' '--stdout=Microsoft-MIEngine-Out-n435glic.05v' '--stderr=Microsoft-MIEngine-Err
or-fdsbk3uz.yce' '--pid=Microsoft-MIEngine-Pid-nyy50ay4.tou' '--dbgExe=C:\Program Files (x86)\
mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Enter length of the sequence: 5
Enter matrix dimension sequence : 40 20 30 10 30
Minimum number of multiplications: 26000
PS C:\C++ Learning Course>
```

Conclusion :

1. Time complexity of Matrix chain multiplication – **O(n^3)**
2. Space complexity of Matrix chain multiplication – **O(n^2)**
3. Whenever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using dynamic programming.
4. The idea is to simply store the results of the subproblems so that we do not have to recompute them when needed later.