**CSE2012-LAB5**

**NAME:** O G RAGAVI

**REG NO**: 20BCE1988

**TITLE:** MATRIX CHAIN MULTIPLICATION

**EX NO:** 5

**IN\_LAB PRACTICE :**

**1.CODE:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

//Given 3 matrices A1(10x100),A2(100x5),A3(5x50)--->l=10,m=100,n=5,p=50;

//brute force approach: compute (A1)(A2A3) and (A1A2)A3, the computation which has minimum cost or scalar mutiplications is the result.

//for (A1)(A2A3)

int res1,res2,l=10,m=100,n=5,p=50;

res1=m\*n\*p+l\*m\*p;

res2=l\*m\*n+l\*n\*p;

if(res1<res2)

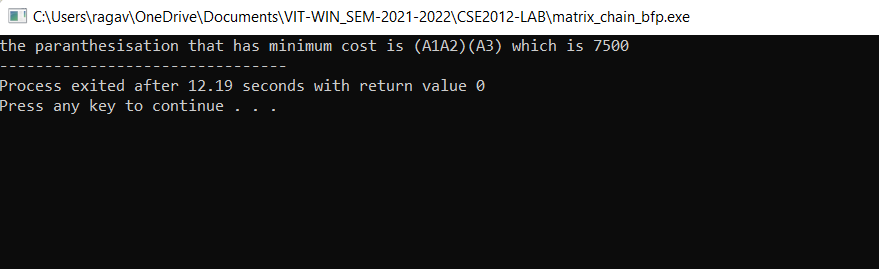
printf("the paranthesisation that has minimum cost is (A1)(A2A3) which is %d",res1);

else

printf("the paranthesisation that has minimum cost is (A1A2)(A3) which is %d",res2);

}

**OUTPUT:**



**LAB PRACTICE SHEET:**

**1.NAIVE MATRIX\_CHAIN MULTIPLICATION:**

CODE:

#include <bits/stdc++.h>

using namespace std;

int MatrixChainOrder(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int min = INT\_MAX;

int count;

for (k = i; k < j; k++)

{

count = MatrixChainOrder(p, i, k)

+ MatrixChainOrder(p, k + 1, j)

+ p[i - 1] \* p[k] \* p[j];

if (count < min)

min = count;

}

return min;

}

// Driver Code

int main()

{

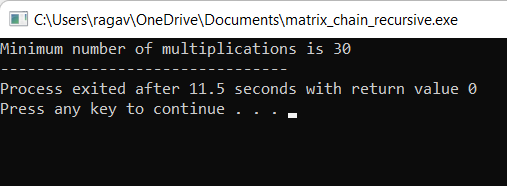
int arr[] = { 1, 2, 3, 4, 3 };

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Minimum number of multiplications is "

<< MatrixChainOrder(arr, 1, n - 1);

}

**OUTPUT:**

**2.MATRIX CHAIN MULTIPLICATION USING DYNAMIC PROGRAMMING:**

**CODE:**

#include<iostream>

using namespace std;

#include<vector>

#include<limits.h>

void matrix\_chain\_order(vector<int> p, vector<vector<int> >& m, vector<vector<int> >& s)

{

cout<<"entered"<<endl;

int n,l,i,j,k,q;

n = p.size()-1;

// try length of chain from 2 to n

for(l=2;l<=n;l++)

{

// From where does the chain starts is in 'i'

//cout<<"l is "<<l<<endl;

for(i=1;i<=n-l+1;i++)

{

// till what position does the chain goes is in 'j'

j = i+l-1;

//cout<<"i is "<<i<<" j is "<<j;

m[i-1][j-1] = INT\_MAX;

for(k=i;k<=j-1;k++)

{

//cout<< " k is "<<k<<endl;

q = m[i-1][k-1] + m[k][j-1] + p[i-1]\*p[k]\*p[j];

if(q<m[i-1][j-1])

{

m[i-1][j-1] = q;

s[i-1][j-1] = k;

}

}

}

}

}

void print\_Optimal\_Parens(vector<vector<int> >& s, int i, int j)

{

if(i==j)

cout<<"A"<<i;

else

{

cout<<"(";

print\_Optimal\_Parens(s,i,s[i-1][j-1]);

print\_Optimal\_Parens(s,s[i-1][j-1]+1,j);

cout<<")";

}

}

int main()

{

int n,i,j;

cin>>n;

vector<int> p(n);

for(i=0;i<n;i++)

cin>>p[i];

vector<vector<int> > m(n-1,vector<int>(n-1,0));

vector<vector<int> > s(n-1,vector<int>(n-1,0));

matrix\_chain\_order(p,m,s);

cout<<"m table is "<<endl;

for(i=0;i<n-2;i++)

{

for(j=1;j<n-1;j++)

cout<<m[i][j]<<" ";

cout<<endl;

}

cout<<"s table is "<<endl;

for(i=0;i<n-2;i++)

{

for(j=0;j<n-1;j++)

cout<<s[i][j]<<" ";

cout<<endl;

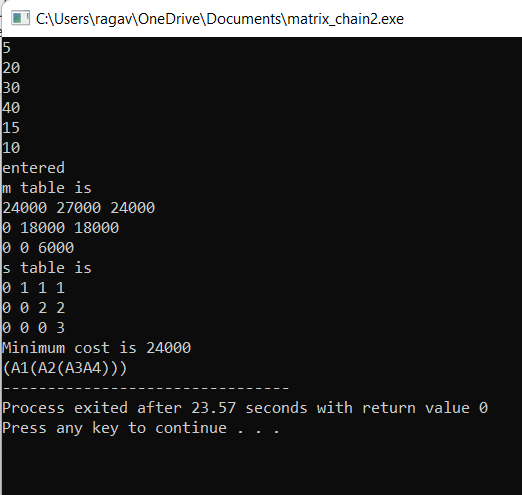
}

cout<<"Minimum cost is "<<m[0][n-2]<<endl;

print\_Optimal\_Parens(s,1,n-1);

}

**OUTPUT:**

****

**3.DYP FOR A1,A2,A3:**

**P={10,100,5,50}**

**CODE:**

#include<iostream>

using namespace std;

#include<vector>

#include<limits.h>

void matrix\_chain\_order(vector<int> p, vector<vector<int> >& m, vector<vector<int> >& s)

{

cout<<"entered"<<endl;

int n,l,i,j,k,q;

n = p.size()-1;

// try length of chain from 2 to n

for(l=2;l<=n;l++)

{

// From where does the chain starts is in 'i'

//cout<<"l is "<<l<<endl;

for(i=1;i<=n-l+1;i++)

{

// till what position does the chain goes is in 'j'

j = i+l-1;

//cout<<"i is "<<i<<" j is "<<j;

m[i-1][j-1] = INT\_MAX;

for(k=i;k<=j-1;k++)

{

//cout<< " k is "<<k<<endl;

q = m[i-1][k-1] + m[k][j-1] + p[i-1]\*p[k]\*p[j];

if(q<m[i-1][j-1])

{

m[i-1][j-1] = q;

s[i-1][j-1] = k;

}

}

}

}

}

void print\_Optimal\_Parens(vector<vector<int> >& s, int i, int j)

{

if(i==j)

cout<<"A"<<i;

else

{

cout<<"(";

print\_Optimal\_Parens(s,i,s[i-1][j-1]);

print\_Optimal\_Parens(s,s[i-1][j-1]+1,j);

cout<<")";

}

}

int main()

{

int n,i,j;

cin>>n;

vector<int> p(n);

for(i=0;i<n;i++)

cin>>p[i];

vector<vector<int> > m(n-1,vector<int>(n-1,0));

vector<vector<int> > s(n-1,vector<int>(n-1,0));

matrix\_chain\_order(p,m,s);

cout<<"m table is "<<endl;

for(i=0;i<n-2;i++)

{

for(j=1;j<n-1;j++)

cout<<m[i][j]<<" ";

cout<<endl;

}

cout<<"s table is "<<endl;

for(i=0;i<n-2;i++)

{

for(j=0;j<n-1;j++)

cout<<s[i][j]<<" ";

cout<<endl;

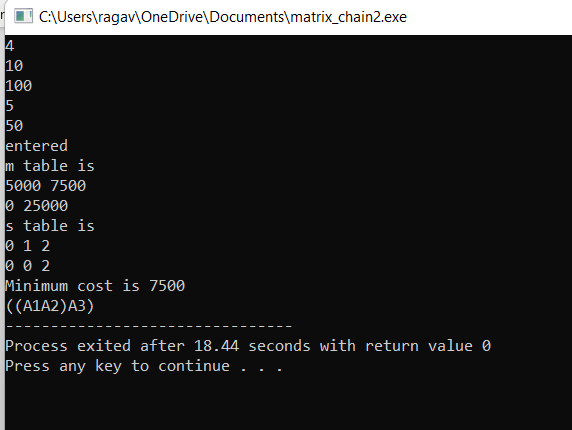
}

cout<<"Minimum cost is "<<m[0][n-2]<<endl;

print\_Optimal\_Parens(s,1,n-1);

}

**OUTPUT:**



**4.BFMCMP:**

**CODE:**

#include<iostream>

using namespace std;

#include<vector>

#include<limits.h>

int matrix\_chain\_mul(int i, int j, vector<int> &p)

{

int min\_cost = INT\_MAX,k,left\_cost,right\_cost,total\_cost;

if(i==j)

return 0;

for(k=i;k<j;k++)

{

left\_cost = matrix\_chain\_mul(i,k,p);

right\_cost = matrix\_chain\_mul(k+1,j,p);

total\_cost = left\_cost + right\_cost + p[i-1] \* p[k] \* p[j];

if(total\_cost<min\_cost)

{

min\_cost = total\_cost;

}

}

return min\_cost;

}

int main()

{

int n,ele,i;

vector<int> p;

cin>>n;

for(i=0;i<n;i++)

{

cin>>ele;

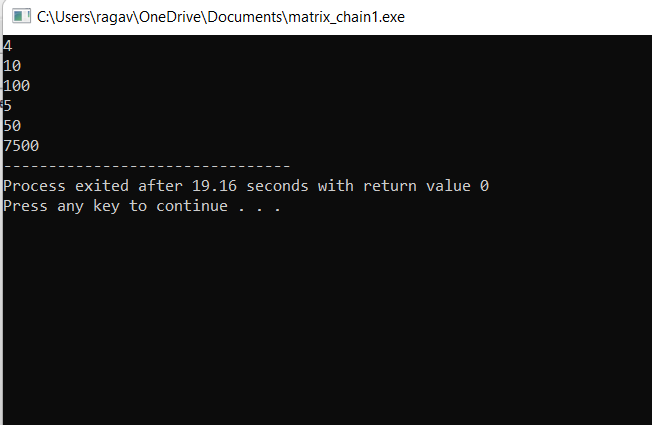
p.push\_back(ele);

}

cout<<matrix\_chain\_mul(1,n-1,p);

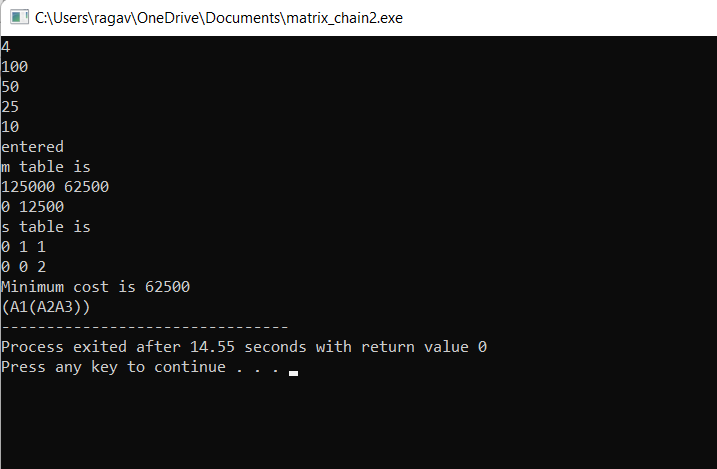
}

**OUTPUT:**



**5.DYP: P={100,50,25,10}**

**OUTPUT:**

****

**6.**The values obtained through BFP and DYP are the same but the execution time and the complexity varies.

7**.CATALAN NUMBER:**

**CODE:**

#include <iostream>

using namespace std;

// A recursive function to find nth catalan number

unsigned long int catalan(unsigned int n)

{

// Base case

if (n <= 1)

return 1;

// catalan(n) is sum of

// catalan(i)\*catalan(n-i-1)

unsigned long int res = 0;

for (int i = 0; i < n; i++)

res += catalan(i)

\* catalan(n - i - 1);

return res;

}

// Driver code

int main()

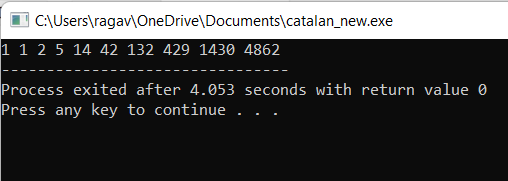
{

for (int i = 0; i < 10; i++)

cout << catalan(i) << " ";

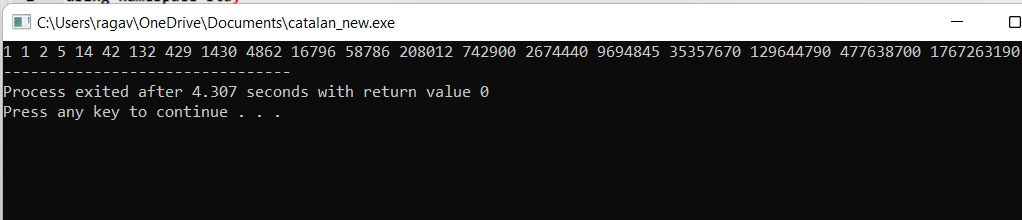
return 0;

}

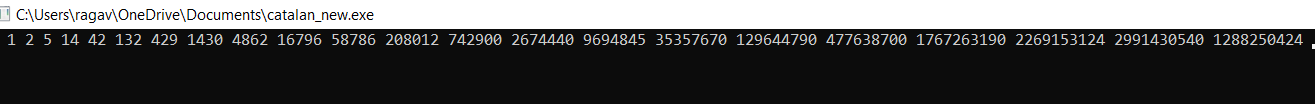
**OUTPUT:  
**

**9.CATALAN NUMBER :**

**FOR n=20:**

****

**for n=30:**

****