PRACTICAL 4:

**Aim: Write a program to implement Strassen’s matrix multiplication using divide and conquer strategy.**

* **PSEUDO CODE**



* **TIME COMPLEXITY**

Addition and Subtraction of two matrices takes O(N2) time. So time complexity can be written as:

T(N) = 7T(N/2) + O(N2)

From Master's Theorem, time complexity of above method is

**O(NLog7)** which is approximately **O(N2.8074)**

* **C PROGRAM**

#include<stdio.h>

#define MAX 64

typedef struct mat

{

int t[MAX][MAX];

}m;

m plus(m a,m b,int n)

{

int i,j;

m c;

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

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

c.t[i][j]=a.t[i][j]+b.t[i][j];

return c;

}

m minus(m a,m b,int n)

{

int i,j;

m c;

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

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

c.t[i][j]=a.t[i][j]-b.t[i][j];

return c;

}

m strass(m a,m b,int n)

{

m p1,p2,p3,p4,p5,p6,p7,t1,t2,t3,t4,t5,t6,t7,t8,sub1,add1,add2;

m c;

int i, j, q;

if(n==1)

c.t[0][0] = a.t[0][0]\*b.t[0][0];

else

{

q = n/2;

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

for(j=0;j<q;j++)

t1.t[i][j] = a.t[i][j];

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

for(j=q;j<n;j++)

t2.t[i][j-q] = a.t[i][j];

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

for(j=0;j<q;j++)

t3.t[i-q][j] = a.t[i][j];

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

for(j=q;j<n;j++)

t4.t[i-q][j-q] = a.t[i][j];

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

for(j=0;j<q;j++)

t5.t[i][j] = b.t[i][j];

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

for(j=q;j<n;j++)

t6.t[i][j-q] = b.t[i][j];

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

for(j=0;j<q;j++)

t7.t[i-q][j] = b.t[i][j];

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

for(j=q;j<n;j++)

t8.t[i-q][j-q] = b.t[i][j];

sub1 = minus(t6,t8,q);

p1 = strass(t1,sub1,q);

add1 = plus(t1,t2,q);

p2 = strass(add1,t8,q);

add1 = plus(t3,t4,q);

p3 = strass(add1,t5,q);

sub1 = minus(t7,t5,q);

p4 = strass(t4,sub1,q);

add1 = plus(t1,t4,q);

add2 = plus(t5,t8,q);

p5 = strass(add1,add2,q);

sub1 = minus(t2,t4,q);

add1 = plus(t7,t8,q);

p6 = strass(sub1,add1,q);

sub1 = minus(t1,t3,q);

add1 = plus(t5,t6,q);

p7 = strass(sub1,add1,q);

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

for(j=0;j<q;j++)

c.t[i][j] = p5.t[i][j] + p4.t[i][j] - p2.t[i][j] + p6.t[i][j];

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

for(j=0;j<q;j++)

c.t[i][j+q] = p1.t[i][j] + p2.t[i][j];

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

for(j=0;j<q;j++)

c.t[i+q][j] = p3.t[i][j] + p4.t[i][j];

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

for(j=0;j<q;j++)

c.t[i+q][j+q] = p1.t[i][j] + p5.t[i][j] - p3.t[i][j] - p7.t[i][j];

}

return c;

}

int main()

{

int in[4];

int i,j,k=0,n=1;

m m1,m2,m3;

printf("Enter number of rows and columns of first matrix\n");

scanf("%d%d", &in[0], &in[1]);

printf("Enter elements of first matrix\n");

for (i = 0; i < in[0]; i++)

for (j = 0; j < in[1]; j++)

scanf("%d", &m1.t[i][j]);

printf("\nEnter number of rows and columns of second matrix\n");

while(1)

{

scanf("%d%d", &in[2], &in[3]);

if(in[1]==in[2])

break;

printf("\n\*\*ENTER VALID DIMENSIONS\*\*\n");

}

printf("Enter elements of second matrix\n");

for (i = 0; i < in[2]; i++)

for (j = 0; j < in[3]; j++)

scanf("%d", &m2.t[i][j]);

for(i=0;i<4;i++)

if(in[i]>k)

k=in[i];

while(1)

{

if(n>=k)

break;

n\*=2;

}

m3 = strass(m1,m2,n);

printf("\nResult of A x B\n");

for(i=0;i<in[0];i++)

{

for(j=0;j<in[3];j++)

printf("%d\t",m3.t[i][j]);

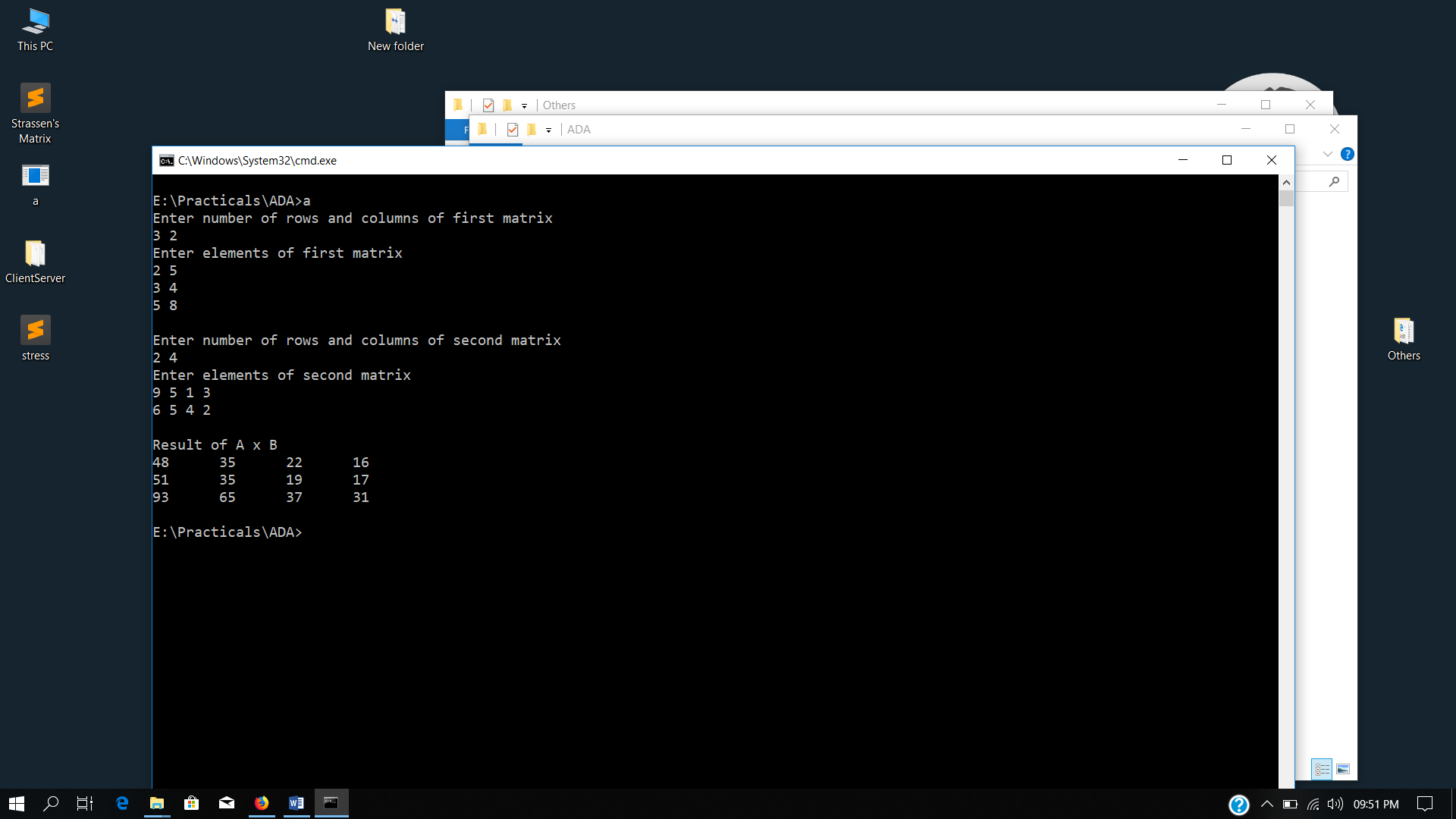
printf("\n");

}

return 0;

}

-**OUTPUT**



PRACTICAL 5:

**Aim: Knapsack problem using dynamic programming.**

* **PSEUDO CODE**

// Input:

// Values (stored in array v)

// Weights (stored in array w)

// Number of distinct items (n)

// Knapsack capacity (W)

for j from 0 to W do

m[0, j] := 0

end for

for i from 1 to n do

for j from 0 to W do

if w[i] <= j then

m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])

else

m[i, j] := m[i-1, j]

end if

end for

end for

* **TIME COMPLEXITY**

Time Complexity: O(nW) where n is the number of items and W is the capacity of knapsack.

* **C PROGRAM**

#include <stdio.h>

#define N 100

typedef struct myItem

{

int profit;

int weight;

} ListItem;

int main()

{

int i,j, n, op, W, t, temp;

printf("Enter number of items: ");

scanf("%d",&n);

int taken[n];

ListItem List[n];

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

{

printf("Enter profit and weight of item %d: ", i+1);

scanf("%d%d",&List[i].profit,&List[i].weight);

taken[i]=0;

}

printf("Enter the Knapsack capacity: ");

scanf("%d",&W);

int matrix[n+1][W+1];

for(i=0;i<n+1;i++)

matrix[i][0]=0;

for(i=0;i<W+1;i++)

matrix[0][i]=0;

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

for(j=1;j<W+1;j++)

{

if(j<List[i-1].weight)

matrix[i][j]=matrix[i-1][j];

else

{

if(matrix[i-1][j]>=List[i-1].profit + matrix[i-1][j-List[i-1].weight])

matrix[i][j]=matrix[i-1][j];

else

matrix[i][j]=List[i-1].profit+matrix[i-1][j-List[i-1].weight];

}

}

i--;

j--;

while(i>0)

{

if(matrix[i][j]==matrix[i-1][j])

i--;

else

{

taken[i-1]=i;

j-=List[i-1].weight;

i--;

}

}

printf("Name\tprofit\tweight\n");

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

if(taken[i]!=0)

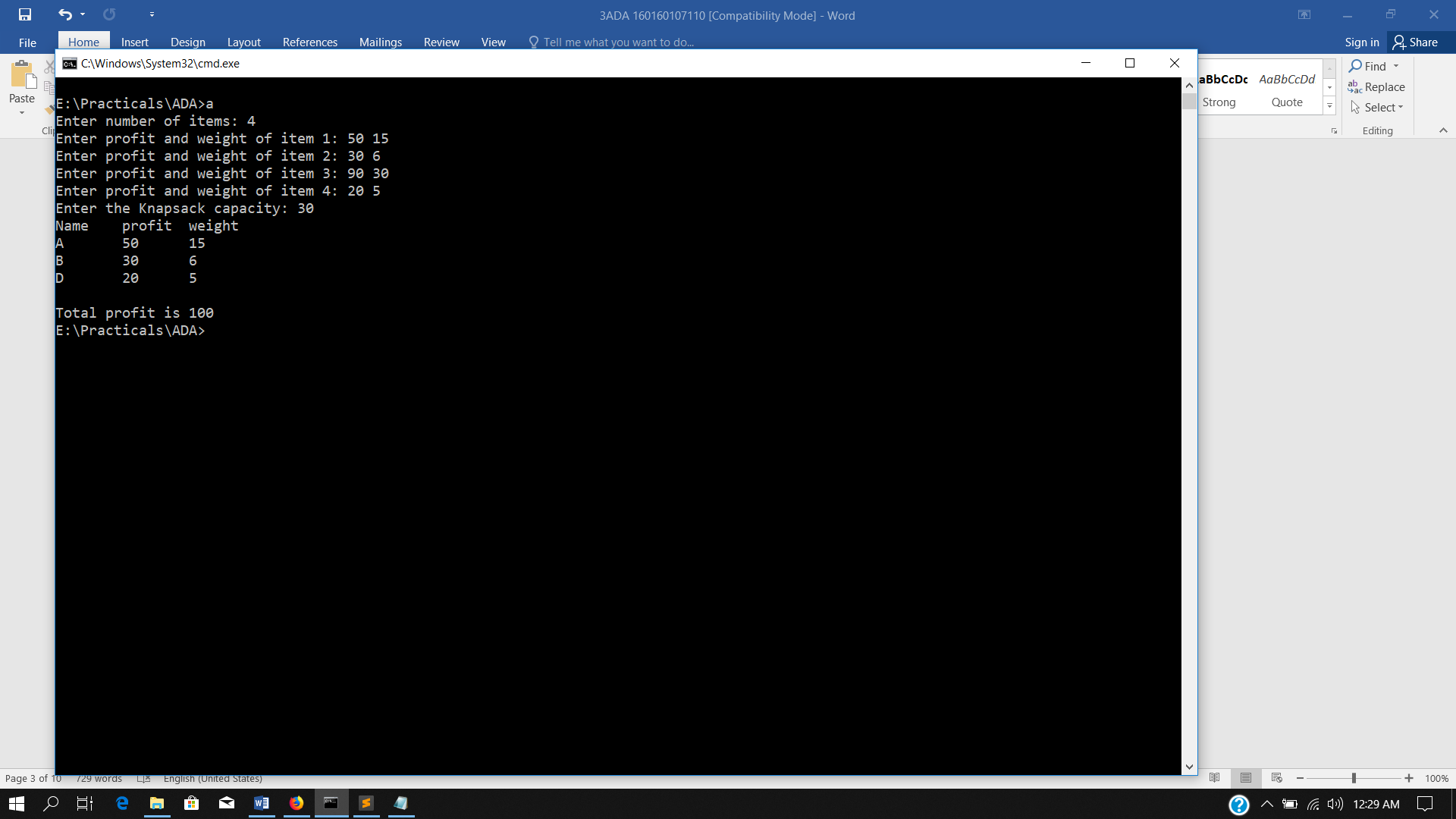
printf("%c\t%d\t%d\n", 'A'+i,List[i].profit,List[i].weight);

printf("\nTotal profit is %d",matrix[n][W]);

return 0;

}

-**OUTPUT**



PRACTICAL 6:

**Aim: Write a program to implement matrix multiplication of 2 matrix.**

* **PSEUDO CODE**

for i=1 to n

for j=1 to n

c[i][j]=0

for k=1 to n

c[i][j] = c[i][j]+a[i][k]\*b[k][j]

* **TIME COMPLEXITY**

Time Complexity: O(n3) as 3 ‘for’ loops are used.

* **C PROGRAM**

#include <stdio.h>

int main()

{

int m, n, p, q, c, d, k, sum = 0;

int first[10][10], second[10][10], multiply[10][10];

printf("Enter number of rows and columns of first matrix\n");

scanf("%d%d", &m, &n);

printf("Enter elements of first matrix\n");

for (c = 0; c < m; c++)

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

scanf("%d", &first[c][d]);

printf("Enter number of rows and columns of second matrix\n");

scanf("%d%d", &p, &q);

if (n != p)

printf("The matrices can't be multiplied with each other.\n");

else

{

printf("Enter elements of second matrix\n");

for (c = 0; c < p; c++)

for (d = 0; d < q; d++)

scanf("%d", &second[c][d]);

for (c = 0; c < m; c++) {

for (d = 0; d < q; d++) {

for (k = 0; k < p; k++) {

sum = sum + first[c][k]\*second[k][d];

multiply[c][d] = sum;

sum = 0;

}

}

printf("Product of the matrices:\n");

for (c = 0; c < m; c++) {

for (d = 0; d < q; d++)

printf("%d\t", multiply[c][d]);

printf("\n");

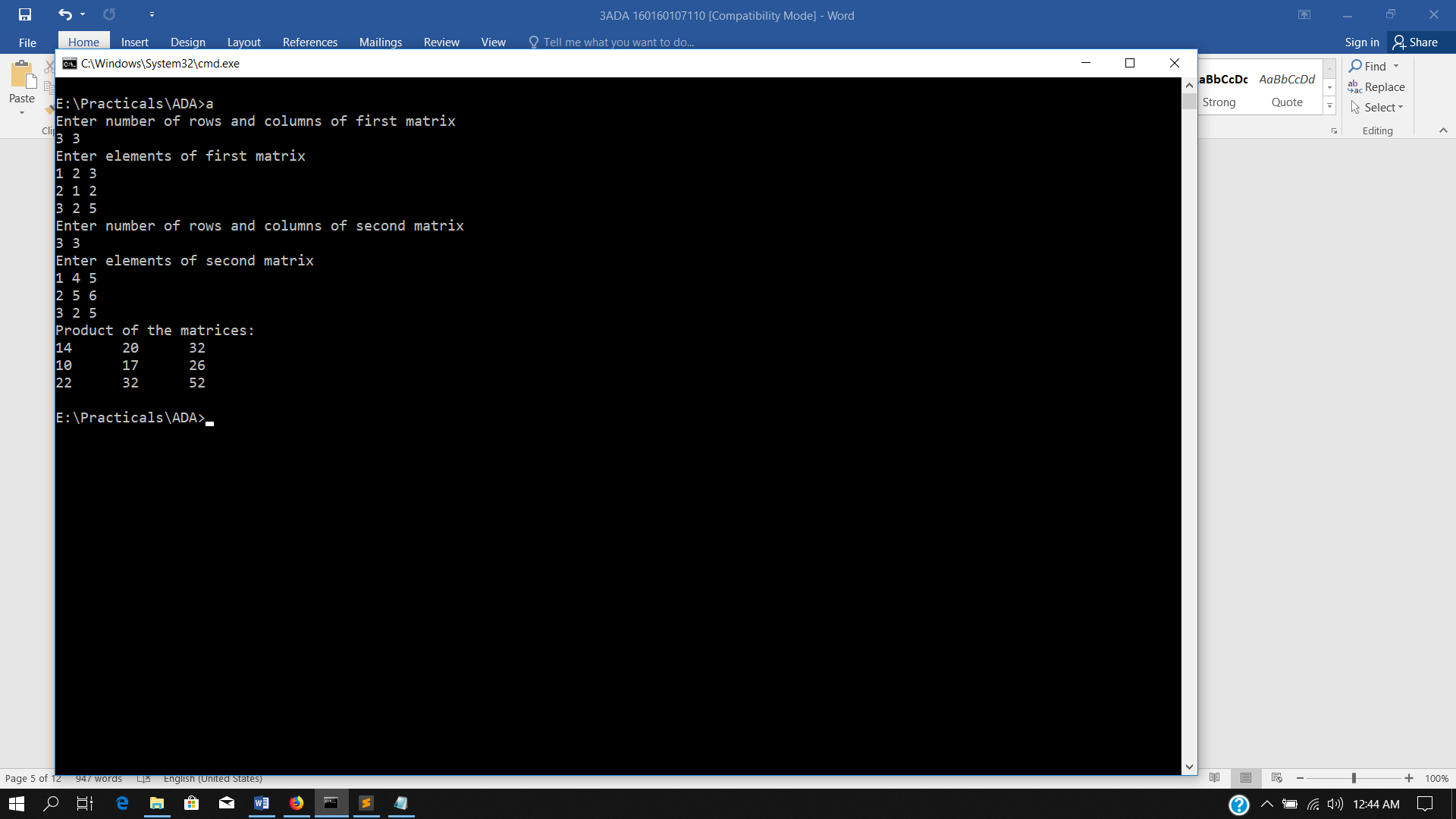
}

}

return 0;

}

-**OUTPUT**



PRACTICAL 7:

**Aim: Chain Matrix multiplication using dynamic programming.**

* **PSEUDO CODE**

Matrix-Chain(array p[1 .. n], int n) {

Array s[1 .. n − 1, 2 .. n];

FOR i = 1 TO n DO m[i, i] = 0;

FOR L = 2 TO n DO {

FOR i = 1 TO n − L + 1 do {

j = i + L − 1;

m[i, j] = infinity;

FOR k = i TO j − 1 DO {

q = m[i, k] + m[k + 1, j] + p[i − 1] p[k] p[j];

IF (q < m[i, j]) {

m[i, j] = q;

s[i, j] = k;

}

}

}

}

return m[1, n](final cost) and s (splitting markers);

}

* **TIME COMPLEXITY**

Since, the three for-loops are nested three deep, and each one of them iterates at most n times. Therefore, the running time of this procedure is Ο(n3).

* **C PROGRAM**

#include<stdio.h>

#include<string.h>

#define MAX 100

int matrix[MAX][MAX];

void insertAtPos(char list[], int val, char pos)

{

char temp[MAX] = {};

int i, a=0, b=0, c=0, index;

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

if(list[i]=='A'+val-1)

{

index = i+1;

break;

}

if(pos=='L')

{

do{

if(index-1==b)

temp[b++] = '(';

else

temp[b++] = list[a++];

}

while(list[a]!=0);

}

else

{

do{

if(index==b)

temp[b++] = ')';

else

temp[b++] = list[a++];

}

while(list[a]!=0);

}

while(temp[c]!=0)

list[c]=temp[c++];

}

void divideNcon(char r[], int low, int high)

{

if(high-low>1)

{

int select = matrix[low][high];

if(low!=select)

{

insertAtPos(r,low,'L');

insertAtPos(r,select,'R');

}

if(select+1!=high)

{

insertAtPos(r,select+1,'L');

insertAtPos(r,high,'R');

}

divideNcon(r,low,select);

divideNcon(r,select+1,high);

}

}

int main()

{

int i, j, k, L, q, n, c=0, m[MAX][MAX];

char result[MAX] = {};

printf("Enter number of matrices: ");

scanf("%d",&n);

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

{

result[c++] = 'A'+i;

if(i<n-1)

result[c++] = '\*';

}

result[c] = ' ';

int arr[++n];

printf("Enter dimensions \n");

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

{

printf("Enter d%d : ",i);

scanf("%d",&arr[i]);

}

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

for (L=2; L<num; L++)

for (i=1; i<num-L+1; i++)

{

j = i+L-1;

m[i][j] = 9999;

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

{

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

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

{

m[i][j] = q;

matrix[i][j] = k;

}

}

}

printf("\nMinimum number of multiplications is %d \n", m[1][num-1]);

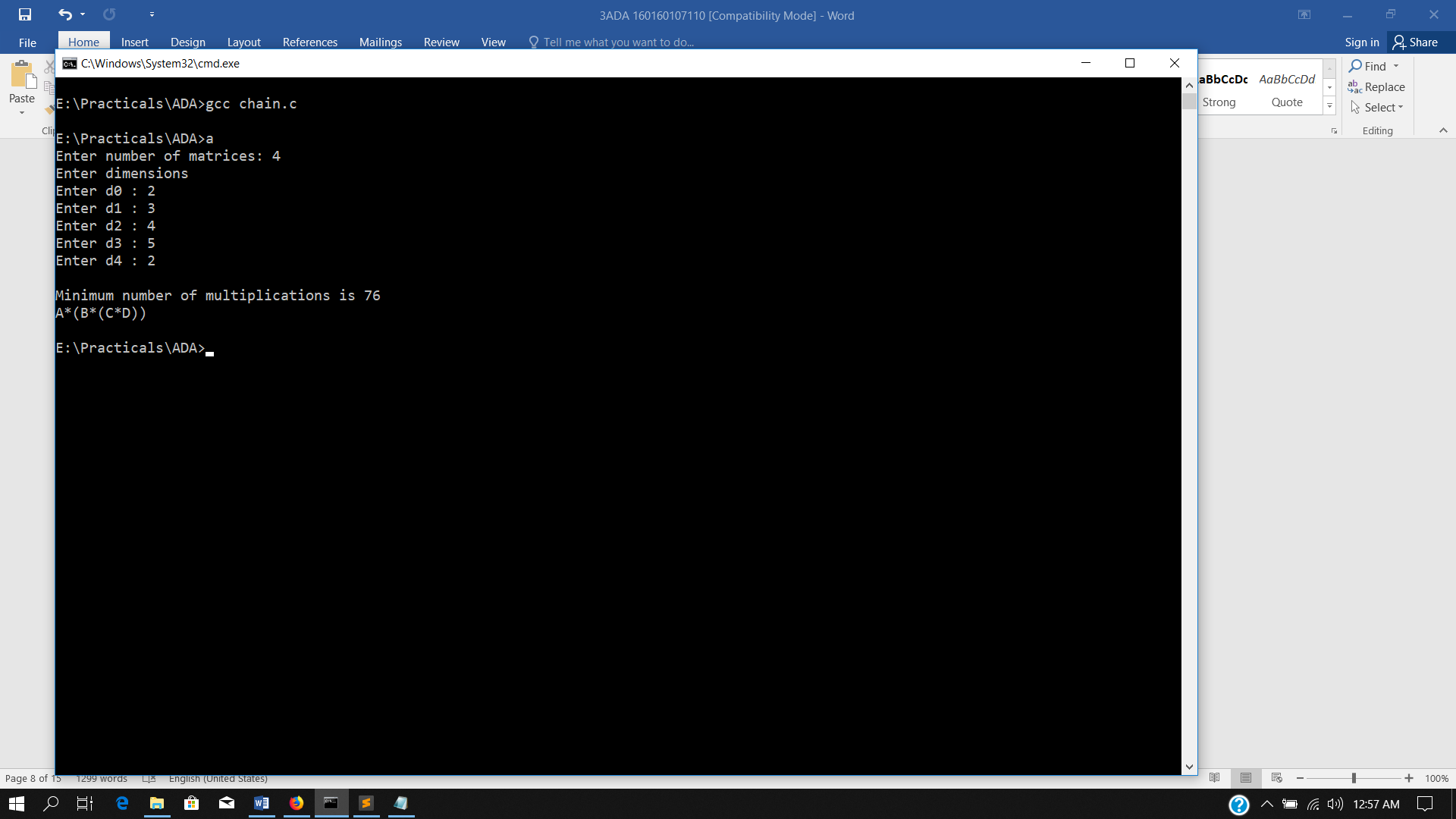
divideNcon(result,1,n-1);

printf("%s\n",result);

return 0;

}

-**OUTPUT**



PRACTICAL 8:

**Aim: Making a change using dynamic programming.**

* **PSEUDO CODE**

func count( n, m )

for i from 0 to n+1

for j from 0 to m

if i equals 0

table[i, j] = 1

else if j equals 0

if i%S[j] equals 0

table[i, j] = 1

else

table[i, j] = 0;

else if S[j] greater than i

table[i, j] = table[i, j - 1]

else

table[i, j] = table[i - S[j], j] + table[i, j-1]

return table[n, m-1]

* **TIME COMPLEXITY**

Time Complexity: O(mn), as there are two for loops running to m and n respectively.

* **C PROGRAM**

#include <stdio.h>

int main()

{

int i,j,n,num,t,temp,counter=0,flag=0;

printf("\nEnter amount: ");

scanf("%d",&num);

printf("Enter number of coins: ");

scanf("%d",&n);

int d[n][2];

printf("Enter %d denominations: ",n);

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

{

scanf("%d",&d[i][0]);

d[i][1]=0;

if(num%d[i][0]==0)

flag=1;

}

if(flag==0)

printf("Change is not possible in given denominations\n");

else

{

int matrix[n+1][num+1];

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

{

t=i;

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

if(d[j][0]<d[t][0])

t=j;

temp = d[i][0];

d[i][0] = d[t][0];

d[t][0] = temp;

}

for(i=0;i<num+1;i++)

matrix[0][i]=99;

for(i=0;i<n+1;i++)

matrix[i][0]=0;

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

for(j=1;j<num+1;j++)

{

if(i==1 && j<d[i-1][0])

matrix[i][j]=99;

else if(i==1)

matrix[i][j]=1+matrix[i][j-d[i-1][0]];

else if(j<d[i-1][0])

matrix[i][j]=matrix[i-1][j];

else

{

if(matrix[i-1][j]>=1+matrix[i][j-d[i-1][0]])

matrix[i][j]=1+matrix[i][j-d[i-1][0]];

else

matrix[i][j]=matrix[i-1][j];

}

}

i=n, j=num;

while(i!=0 || j!=0)

{

if(matrix[i][j]==matrix[i-1][j])

i--;

else

{

j-=d[i-1][0];

d[i-1][1]++;

}

}

printf("\nCoin\tCount\n");

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

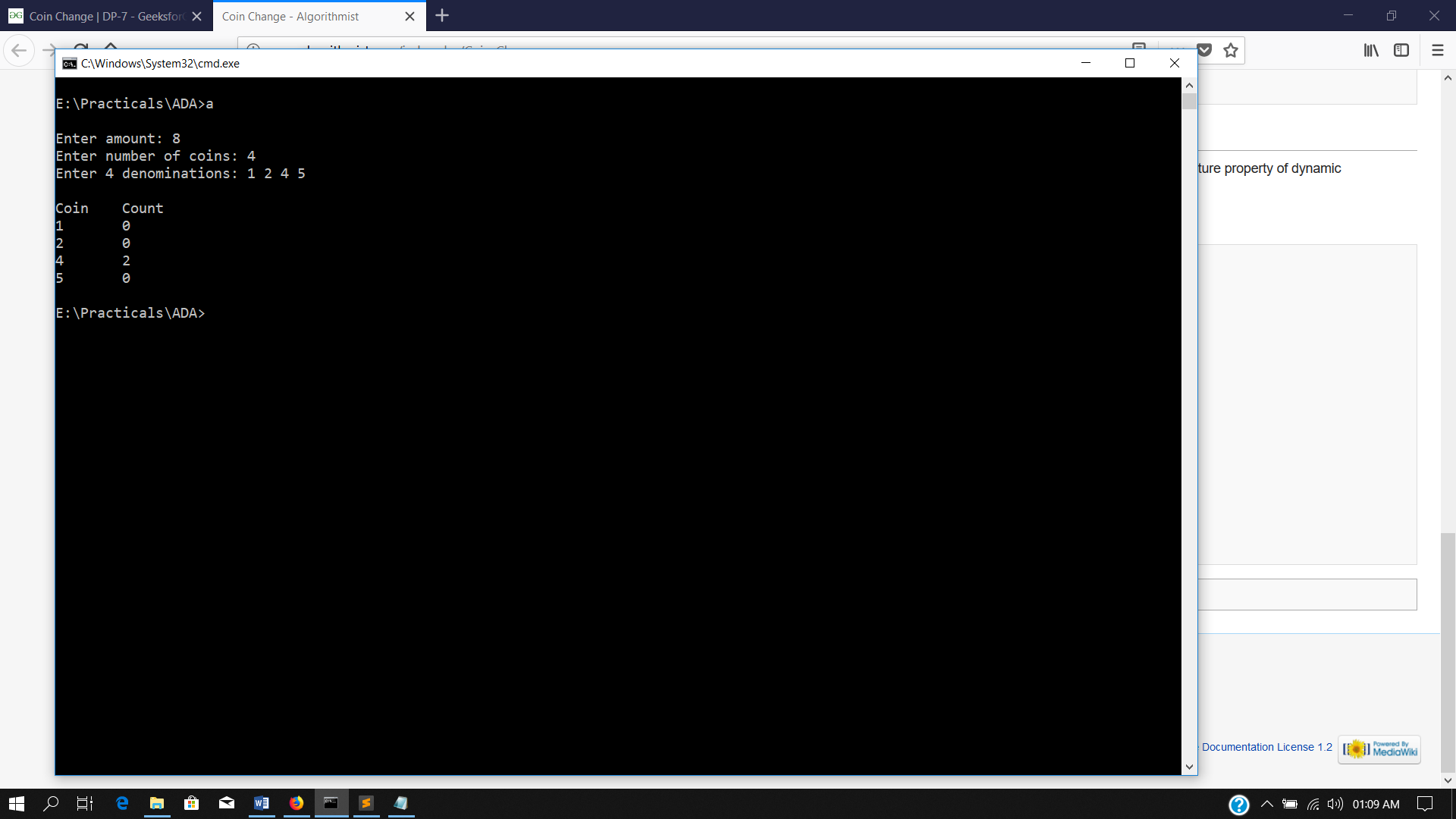
printf("%d\t%d\n", d[i][0], d[i][1]);

}

return 0;

}

-**OUTPUT**



PRACTICAL 9:

**Aim: Knapsack problem using greedy algorithm.**

* **PSEUDO CODE**

Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)

for i = 1 to n

do x[i] = 0

weight = 0

for i = 1 to n

if weight + w[i] ≤ W then

x[i] = 1

weight = weight + w[i]

else

x[i] = (W - weight) / w[i]

weight = W

break

return x

* **TIME COMPLEXITY**

If the provided items are already sorted into a decreasing order of pi/wi, then the while loop takes a time in O(n); Therefore, the total time including the sort is in O(n logn).

* **C PROGRAM**

#include <stdio.h>

#define N 100

typedef struct myItem

{

char name;

int profit;

int weight;

float value;

} ListItem;

int main()

{

int i,j, n, op, W, t, temp;

printf("Enter number of items: ");

scanf("%d",&n);

ListItem List[n];

float taken[n];

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

{

printf("Enter profit and weight of item %d: ", i+1);

scanf("%d%d",&List[i].profit,&List[i].weight);

List[i].value=(float)List[i].profit/(float)List[i].weight;

taken[i]=0.0;

List[i].name='A'+i;

}

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

{

t=i;

ListItem te;

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

if(List[j].value>List[t].value)

t=j;

te = List[i];

List[i] = List[t];

List[t] = te;

}

printf("Enter the Knapsack capacity: ");

scanf("%d",&W);

printf("\nChoose Method: \n1.Binary method\n2.Fraction method\nEnter (1 or 2): ");

scanf("%d",&op);

int count=0;

temp=W;

while(temp>0 && count<n)

{

if(temp>=List[count].weight)

{

temp-=List[count].weight;

taken[count]=List[count].weight;

}

else

{

if(op==2)

{

taken[count]=temp;

temp=0;

}

}

count++;

}

printf("Item\tprofit\tweight\tselected\tearndProfit\n");

float w, totProfit=0;

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

{

float p;

p = ((float)taken[i] == (float)List[i].weight) ? List[i].profit:taken[i]\*List[i].value;

w+=taken[i];

printf("Item %c \t%d\t%d\t%f\t%f\n", List[i].name,List[i].profit,List[i].weight,taken[i],p);

totProfit+=p;

}

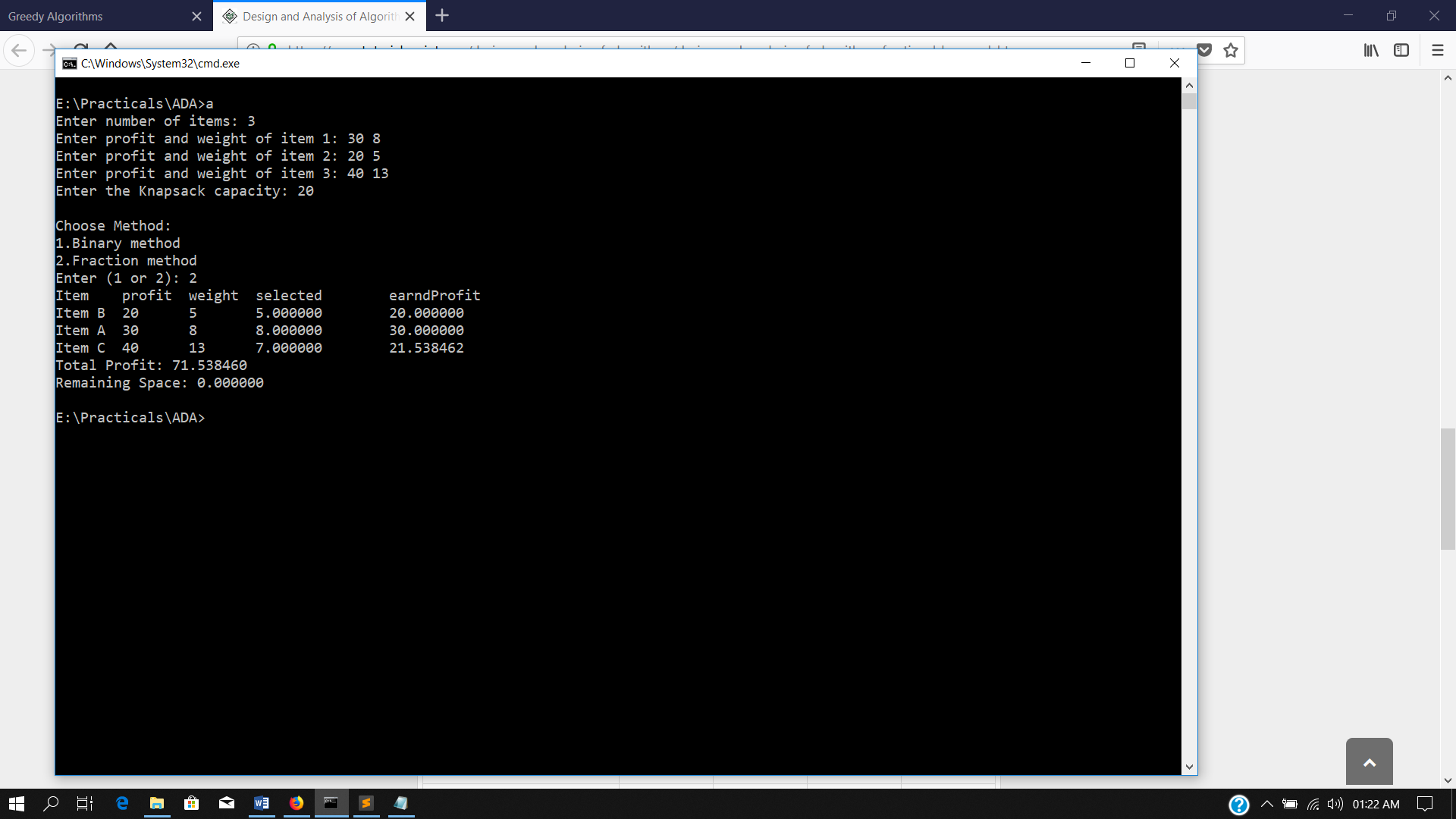
printf("Total Profit: %f\n",totProfit);

printf("Remaining Space: %f\n", W-w);

return 0;

}

-**OUTPUT**



PRACTICAL 10:

**Aim: Making change problem using greedy methods.**

* **PSEUDO CODE**

min\_coins(coin\_value[],n,amount)

{

for( i= 1 to n )

while amount > = to coins[i]

{

amount = amount - coin\_value[i]

print coin\_value[i]

}

}

* **TIME COMPLEXITY**

Time complexity for this algorithm is O( Amount \* numofdenominations )

* **C PROGRAM**

#include <stdio.h>

int main()

{

int i,j,n,num,t,temp,counter=0;

printf("Enter number of coins: ");

scanf("%d",&n);

int d[n][2];

printf("Enter %d denominations: ",n);

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

{

scanf("%d",&d[i][0]);

d[i][1]=0;

}

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

{

t=i;

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

if(d[j][0]>d[t][0])

t=j;

temp = d[i][0];

d[i][0] = d[t][0];

d[t][0] = temp;

}

printf("\nEnter amount: ");

scanf("%d",&num);

while(num>=d[n-1][0])

{

if(num>=d[counter][0])

{

num-=d[counter][0];

d[counter][1]+=1;

}

else

counter++;

}

printf("\n");

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

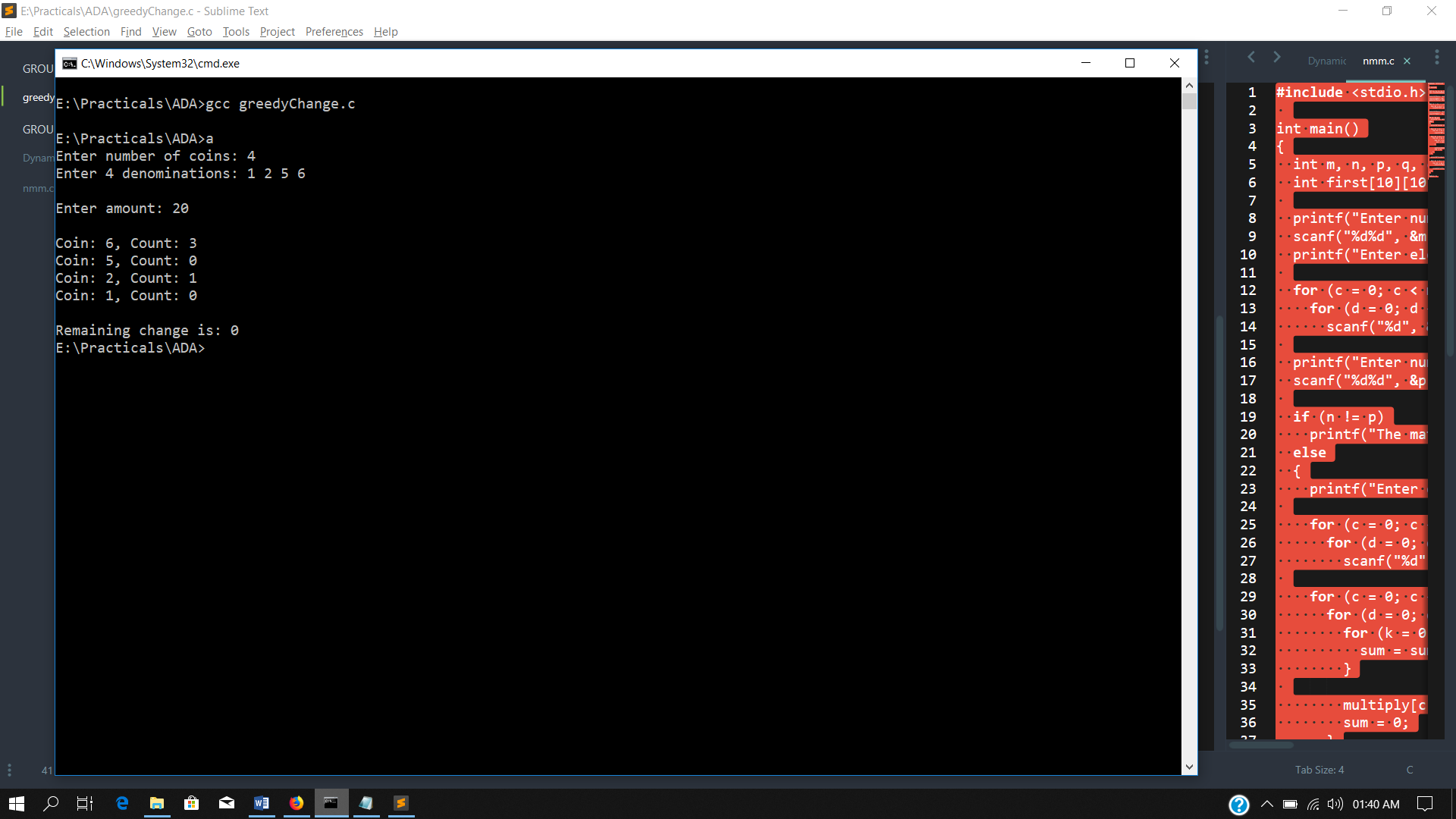
printf("Coin: %d, Count: %d\n", d[i][0], d[i][1]);

printf("\nRemaining change is: %d", num);

return 0;

}

-**OUTPUT**



PRACTICAL 11:

**Aim: Write a C program for Prim’s algorithm.**

* **PSEUDO CODE**

MST-PRIM(G, w, r)

for each u ∈ G.V

u.key ← ∞

u.π ← NIL

r.key ← 0

Q ← G.V

while Q ≠ Ø

u ← EXTRACT-MIN(Q)

for each v ∈ G.Adj[u]

if v ∈ Q and w(u, v) < v.key

v.π ← u

v.key ← w(u, v)

* **TIME COMPLEXITY**

The time complexity is O(VlogV + ElogV) = O(ElogV), making it the same as Kruskal's algorithm. However, Prim's algorithm can be improved using Fibonacci Heaps (cf Cormen) to O(E + logV).

* **C PROGRAM**

#include<stdio.h>

#define N 100

typedef struct myEdge

{

int x;

int y;

int w;

int used;

} edge;

typedef struct myEdgeList

{

edge data[N];

int num;

} edgeList;

int main()

{

int n,i,j,v;

edgeList eL,fL;

eL.num=0;

fL.num=0;

printf("Enter number of nodes : ");

scanf("%d",&n);

int matrix[n][n];

printf("Enter Matrix (NxN):\n");

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

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

{

if(i==j)

matrix[i][i]=0;

else

{

printf("Enter distance between %d and %d ",i,j);

scanf("%d",&matrix[i][j]);

matrix[j][i]=matrix[i][j];

eL.data[eL.num].x=i;

eL.data[eL.num].y=j;

eL.data[eL.num].w=matrix[i][j];

eL.num++;

}

}

int node[n];

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

node[i]=0;

printf("\nEnter initial vertex: ");

scanf("%d",&v);

node[v]=1;

int min=99;

int parent=-1;

int index = -1;

while(fL.num!=n-1)

{

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

{

if(node[i]==1)

{

for(j=0;j<eL.num;j++)

{

if((eL.data[j].x==i && node[eL.data[j].y]==0) || (eL.data[j].y==i && node[eL.data[j].x]==0))

{

if(eL.data[j].w<min)

{

min=eL.data[j].w;

if(eL.data[j].x==i)

{

parent = eL.data[j].x;

index = eL.data[j].y;

}

else

{

parent = eL.data[j].y;

index = eL.data[j].x;

}

}

}

}

}

}

node[index]=1;

fL.data[fL.num].x=parent;

fL.data[fL.num].y=index;

fL.data[fL.num].w=min;

fL.num++;

parent=-1;

index=-1;

min=99;

}

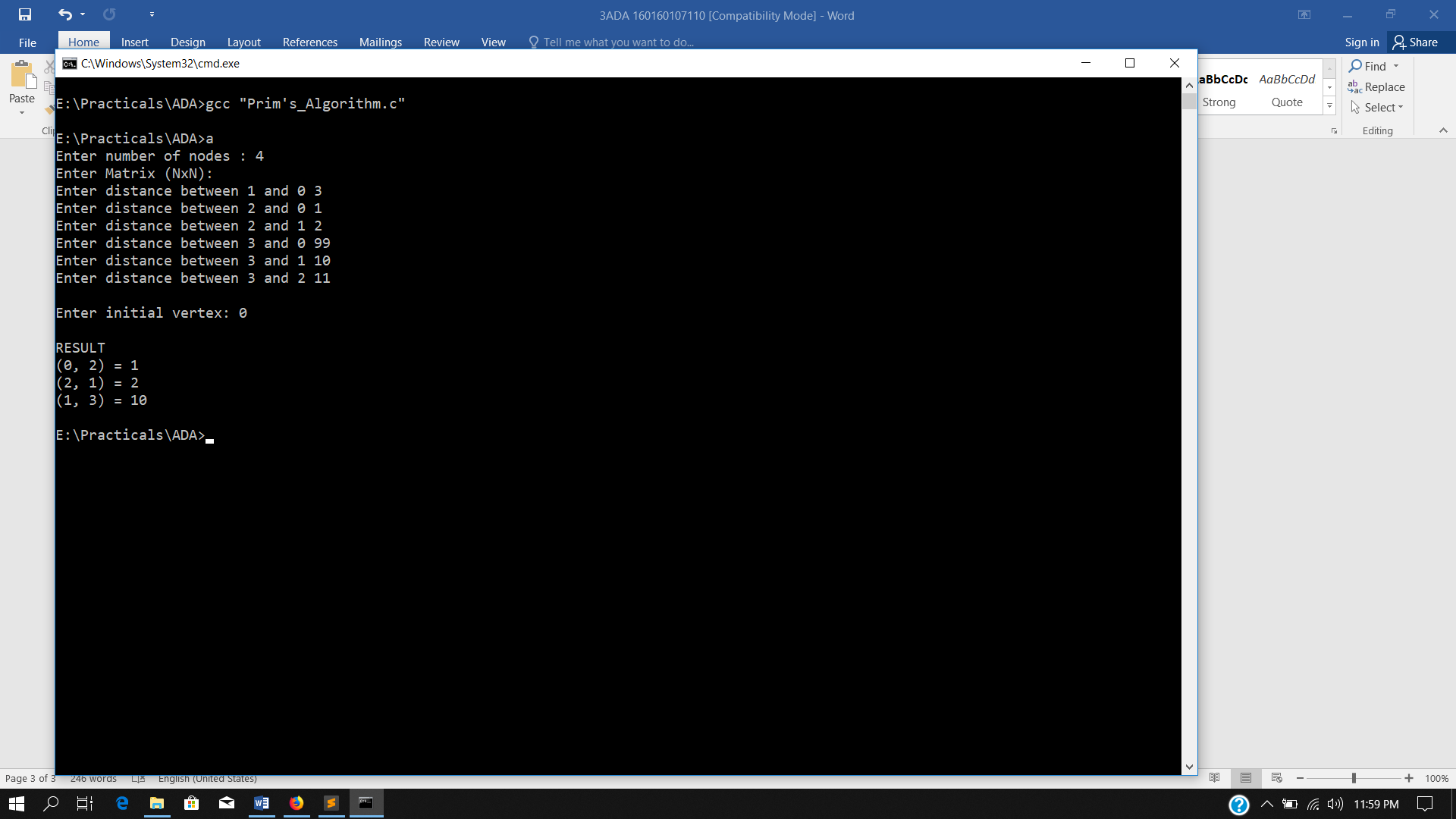
printf("\nRESULT\n");

for(i=0;i<fL.num;i++)

printf("(%d, %d) = %d\n", fL.data[i].x, fL.data[i].y, fL.data[i].w);

}

-**OUTPUT**



PRACTICAL 12:

**Aim: Write a C program for Kruskal’s algorithm.**

* **PSEUDO CODE**

MST-KRUSKAL(G, w)

A ← Ø

>for each vertex v ­ V[G]

>do MAKE-SET(v)

>sort the edges of E into nondecreasing order by weight w

>for each edge (u, v) ­ E, taken in nondecreasing order by weight

>do if FIND-SET(u) ≠ FIND-SET(v)

then A ← A ­ {(u, v)}

UNION(u, v)

>return A

* **TIME COMPLEXITY**

O(ElogE) or O(ElogV). Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost O(LogV) time. So overall complexity is O(ELogE + ELogV) time. The value of E can be atmost O(V2), so O(LogV) are O(LogE) same. Therefore, overall time complexity is O(ElogE) or O(ElogV).

* **C PROGRAM**

#include<stdio.h>

#define N 100

typedef struct myEdge

{

int x;

int y;

int w;

int used;

} edge;

typedef struct myEdgeList

{

edge data[N];

int num;

} edgeList;

edgeList sortlist(edgeList l)

{

edge temp;

int i,j,t;

for(i=0;i<l.num-1;i++)

{

t=i;

for(j=i;j<l.num;j++)

if(l.data[j].w<l.data[t].w)

t=j;

temp = l.data[i];

l.data[i] = l.data[t];

l.data[t] = temp;

}

return l;

}

int magicArray[N];

int main()

{

int n,i,j,a,b;

edgeList eL;

eL.num=0;

printf("Enter number of nodes : ");

scanf("%d",&n);

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

magicArray[i]=i;

int matrix[n][n];

printf("Enter Matrix (NxN):\n");

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

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

{

if(i==j)

matrix[i][i]=0;

else

{

printf("Enter distance between %d and %d ",i,j);

scanf("%d",&matrix[i][j]);

matrix[j][i]=matrix[i][j];

eL.data[eL.num].x=i;

eL.data[eL.num].y=j;

eL.data[eL.num].w=matrix[i][j];

eL.num++;

}

}

printf("\nSorted matrix\n");

eL=sortlist(eL);

for(i=0;i<eL.num;i++)

{

printf("(%d, %d) = %d\n",eL.data[i].x,eL.data[i].y,eL.data[i].w);

eL.data[i].used = 0;

}

for(i=0;i<eL.num;i++)

{

a = magicArray[eL.data[i].x];

b = magicArray[eL.data[i].y];

if(a!=b)

{

eL.data[i].used = 1;

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

if(magicArray[j]==b)

magicArray[j]=a;

}

}

printf("\nRESULT\n");

for(i=0;i<eL.num;i++)

if(eL.data[i].used==1)

printf("(%d, %d) = %d\n", eL.data[i].x, eL.data[i].y, eL.data[i].w);

}

-**OUTPUT**

