SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
Lab Assignment - II, Fall Semester 2021-22
Course Code : CSE2012 Slot : L3 + L4
Course Name: Design and Analysis of Algorithms
Marks : 15
Last Date : 10-10-21

NAME: ARYA DUBEY

REG NO:20BCE0908

FACULTY: PROFESSOR SRIVANI A

Question 1

1. **Matrix Multiplication by passing the 2D array to a function.**

```
#include <iostream>
using namespace std;

void enterData(int firstMatrix[][10], int secondMatrix[][10], int rowFirst, int columnFirst, int rowSecond, int columnSecond);
void multiplyMatrices(int firstMatrix[][10], int secondMatrix[][10], int multResult[][10], int rowFirst, int columnFirst, int rowSecond, int columnSecond);
void display(int mult[][10], int rowFirst, int columnSecond);

int main()
```

```
{
    int firstMatrix[10][10], secondMatrix[10][10], mult[10][10], rowFirst,
columnFirst, rowSecond, columnSecond, i, j, k;

    cout << "Enter rows and column for first matrix: ";
    cin >> rowFirst >> columnFirst;

    cout << "Enter rows and column for second matrix: ";
    cin >> rowSecond >> columnSecond;

    // If colum of first matrix in not equal to row of second matrix, asking
user to enter the size of matrix again.
    while (columnFirst != rowSecond)
    {
        cout << "Error! column of first matrix not equal to row of
second." << endl;
        cout << "Enter rows and column for first matrix: ";
        cin >> rowFirst >> columnFirst;
        cout << "Enter rows and column for second matrix: ";
        cin >> rowSecond >> columnSecond;
    }

    // Function to take matrices data
   enterData(firstMatrix, secondMatrix, rowFirst, columnFirst, rowSecond,
columnSecond);

   // Function to multiply two matrices.
   multiplyMatrices(firstMatrix, secondMatrix, mult, rowFirst, columnFirst,
rowSecond, columnSecond);

   // Function to display resultant matrix after multiplication.
   display(mult, rowFirst, columnSecond);
```

```
    return 0;
}

void enterData(int firstMatrix[][10], int secondMatrix[][10], int rowFirst, int
columnFirst, int rowSecond, int columnSecond)
{
    int i, j;
    cout << endl << "Enter elements of matrix 1:" << endl;
    for(i = 0; i < rowFirst; ++i)
    {
        for(j = 0; j < columnFirst; ++j)
        {
            cout << "Enter elements a"<< i + 1 << j + 1 << ": ";
            cin >> firstMatrix[i][j];
        }
    }

    cout << endl << "Enter elements of matrix 2:" << endl;
    for(i = 0; i < rowSecond; ++i)
    {
        for(j = 0; j < columnSecond; ++j)
        {
            cout << "Enter elements b" << i + 1 << j + 1 << ": ";
            cin >> secondMatrix[i][j];
        }
    }
}

void multiplyMatrices(int firstMatrix[][10], int secondMatrix[][10], int
mult[][10], int rowFirst, int columnFirst, int rowSecond, int columnSecond)
```

```
{
	int i, j, k;

	// Initializing elements of matrix mult to 0.
	for(i = 0; i < rowFirst; ++i)
	{
		for(j = 0; j < columnSecond; ++j)
		{
			mult[i][j] = 0;
		}
	}

	// Multiplying matrix firstMatrix and secondMatrix and storing in array
mult.
	for(i = 0; i < rowFirst; ++i)
	{
		for(j = 0; j < columnSecond; ++j)
		{
			for(k=0; k<columnFirst; ++k)
			{
				mult[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
			}
		}
	}
}

void display(int mult[][10], int rowFirst, int columnSecond)
{
	int i, j;
```

```
        cout << "Output Matrix:" << endl;
        for(i = 0; i < rowFirst; ++i)
        {
                for(j = 0; j < columnSecond; ++j)
                {
                        cout << mult[i][j] << " ";
                        if(j == columnSecond - 1)
                                cout << endl << endl;
                }
        }
}
```

```
Enter rows and column for second matrix: 3 4

Enter elements of matrix 1:
Enter elements a11: 5
Enter elements a12: 8
Enter elements a13: 8
Enter elements a21: 7
Enter elements a22: 9
Enter elements a23: 45
Enter elements a31: 1
Enter elements a32: 45
Enter elements a33: 78

Enter elements of matrix 2:
Enter elements b11: 98
Enter elements b12: 7
Enter elements b13: 6
Enter elements b14: 54
Enter elements b21: 2
Enter elements b22: 4
Enter elements b23: 25
Enter elements b24: 85
Enter elements b31: 25
Enter elements b32: 122
Enter elements b33: 13
Enter elements b34: 59
Output Matrix:
706 1043 334 1422

1829 5575 852 3798

2138 9703 2145 8481



...Program finished with exit code 0
```

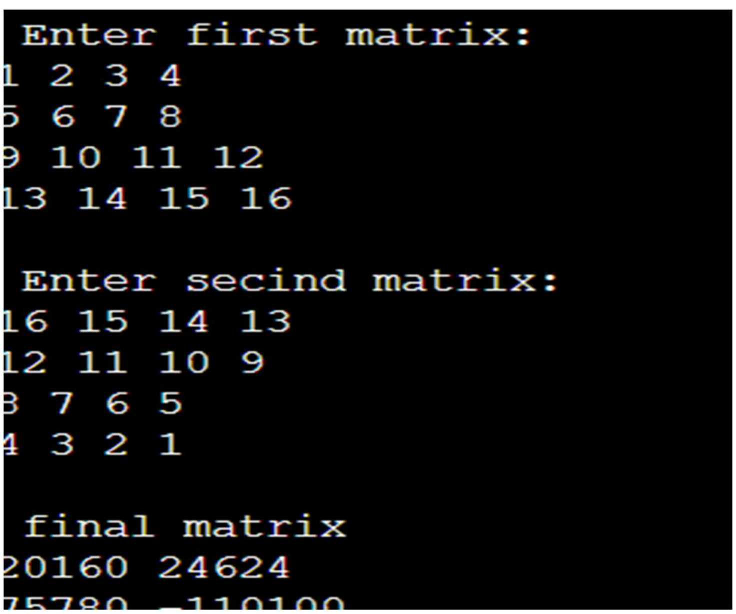
Question 2.

Divide and Conquer Approach : Strassen's Matrix Multiplication

```
#include<iostream>
using namespace std;
double a[4][4];
double b[4][4];

void insert(double x[4][4])
{
	double val;
	for(int i=0;i<4;i++)
	{
		for(int j=0;j<4;j++)
		{
			cin>>val;
			x[i][j]=val;
		}
	}
}
double cal11(double x[4][4])
{
	return (x[1][1] * x[1][2])+ (x[1][2] * x[2][1]);
}
double cal21(double x[4][4])
{
	return (x[3][1] * x[4][2])+ (x[3][2] * x[4][1]);
}

double cal12(double x[4][4])
{
	return (x[1][3] * x[2][4])+ (x[1][4] * x[2][3]);
```

```
}

double cal22(double x[4][4])
{
        return (x[2][3] * x[1][4])+ (x[2][4] * x[1][3]);
}

int main()
{
        double a11,a12,a22,a21,b11,b12,b21,b22,a[4][4],b[4][4];
        double p,q,r,s,t,u,v,c11,c12,c21,c22;
        //insert values in the matrix a
        cout<<"\n Enter first matrix: \n";
        insert(a);
        //insert values in the matrix a
        cout<<"\n Enter secind matrix: \n";
        insert(b);

        //dividing single 4x4 matrix into four 2x2 matrices
        a11=cal11(a);
        a12=cal12(a);
        a21=cal21(a);
        a22=cal22(a);
        b11=cal11(b);
        b12=cal12(b);
        b21=cal21(b);
        b22=cal22(b);

        //assigning variables acc. to strassen's algo
        p=(a11+a22)*(b11+b22);
```

```
        q=(a21+a22)*b11;
        r=a11*(b12-b22);
        s=a22*(b21-b11);
        t=(a11+a12)*b22;
        u=(a11-a21)*(b11+b12);
        v=(a12-a22)*(b21+b22);

    //outputting the final matrix
cout<<"\n final matrix";
        cout<<"\n"<<p+s-t+v<<" "<<r+t;
        cout<<"\n"<<q+s<<" "<<p+r-q+u;
    return 0;
}
```



Question 3)

Greedy Approach :
i) Any known problem

```
#include <bits/stdc++.h>
using namespace std;


class Edge {
public:
    int src, dest, weight;
};

class Graph {
public:


    int V, E;

    Edge* edge;
};



Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
```

```
    graph->edge = new Edge[E];

    return graph;
}


class subset {
public:
    int parent;
    int rank;
};


int find(subset subsets[], int i)
{

    if (subsets[i].parent != i)
        subsets[i].parent
            = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}


void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
```

```
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;



        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}



int myComp(const void* a, const void* b)
{
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}



void KruskalMST(Graph* graph)
{
    int V = graph->V;
    Edge result[V];
    int e = 0;
    int i = 0;

    qsort(graph->edge, graph->E, sizeof(graph->edge[0]),
          myComp);
```

```
subset* subsets = new subset[(V * sizeof(subset))];


for (int v = 0; v < V; ++v)
{
    subsets[v].parent = v;
    subsets[v].rank = 0;
}


while (e < V - 1 && i < graph->E)
{

    Edge next_edge = graph->edge[i++];

    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);


    if (x != y) {
        result[e++] = next_edge;
        Union(subsets, x, y);
    }

}


cout << "Following are the edges in the constructed "
        "MST\n";
```

```
    int minimumCost = 0;
    for (i = 0; i < e; ++i)
    {
        cout << result[i].src << " -- " << result[i].dest
             << " == " << result[i].weight << endl;
        minimumCost = minimumCost + result[i].weight;
    }
    // return;
    cout << "Minimum Cost Spanning Tree: " << minimumCost
         << endl;
}


int main()
{

            4 */
    int V = 4;
    int E = 5;
    Graph* graph = createGraph(V, E);


    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;


    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;
```

```
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;



    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;



    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;




    KruskalMST(graph);


    return 0;
}
```

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19


...Program finished with exit code 0
```

ii) Fractional Knapsack Problem.

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

using namespace std;


struct Item {
    int value, weight;


    Item(int value, int weight)
    {
       this->value=value;
       this->weight=weight;
    }
};


bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / (double)a.weight;
    double r2 = (double)b.value / (double)b.weight;
    return r1 > r2;
}


double fractionalKnapsack(int W, struct Item arr[], int n)
```

```
{
    sort(arr, arr + n, cmp);

    int curWeight = 0;
    double finalvalue = 0.0;

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

        if (curWeight + arr[i].weight <= W) {
            curWeight += arr[i].weight;
            finalvalue += arr[i].value;
        }

        else {
            int remain = W - curWeight;
            finalvalue += arr[i].value
                          * ((double)remain
                             / (double)arr[i].weight);
            break;
        }
    }

    return finalvalue;
}
```

```
int main()
{
    int W = 50;
    Item arr[] = { { 150, 90 }, { 105, 20 }, { 10, 130 } };

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

    // Function call
    cout << "Maximum value we can obtain = "
         << fractionalKnapsack(W, arr, n);
    return 0;
}
```

```
Maximum value we can obtain = 155

...Program finished with exit code 0
Press ENTER to exit console.
```

Question 4.

Dynamic Programming:

i) 0/1 Knapsack Problem

```
#include<stdio.h>
// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapsack(int val[], int weight[],int w,int n)
{
    if(n==0 || w==0 ) return 0;

    if(weight[n-1]>w)
        return knapsack(val,weight,w,n-1);
    else
        return max(val[n-1]+knapsack(val,weight,w-weight[n-1],n-1),
        knapsack(val,weight,w,n-1) );



}
// Driver program to test above function
int main()
{
         int val[] = {60, 100, 120};
         int wt[] = {10, 20, 30};
         int W = 50;
         int n = sizeof(val)/sizeof(val[0]);
```

```
	printf("%d", knapsack(val,wt,W,n));
	return 0;
}
```

```
220

...Program finished with exit code 0
```

---

ii) Travelling Salesman Problem

```
#include<iostream>

using namespace std;

int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
int i,j;

cout<<"Enter the number of cities: ";
cin>>n;

cout<<"\nEnter the Cost Matrix\n";

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

```
for( j=0;j < n;j++)
cin>>ary[i][j];

completed[i]=0;
}

cout<<"\n\nThe cost list is:";

for( i=0;i < n;i++)
{
cout<<"\n";

for(j=0;j < n;j++)
cout<<"\t"<<ary[i][j];
}
}

int least(int c)
{
int i,nc=999;
int min=999,kmin;

for(i=0;i < n;i++)
{
if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
{
min=ary[i][0]+ary[c][i];
kmin=ary[c][i];
nc=i;
```

```
}
}

if(min!=999)
cost+=kmin;

return nc;
}

void mincost(int city)
{
int i,ncity;

completed[city]=1;

cout<<city+1<<"--->";
ncity=least(city);

if(ncity==999)
{
ncity=0;
cout<<ncity+1;
cost+=ary[city][ncity];

return;
}

mincost(ncity);
}
```

```
int main()

{

takeInput();


cout<<"\n\nThe Path is:\n";

mincost(0); //passing 0 because starting vertex


cout<<"\n\nMinimum cost is "<<cost;


return 0;

}
```

```
Enter the number of cities: 4

Enter the Cost Matrix
0 2 5 1
8 0 5 9
1 3 0 7
3 9 7 0


The cost list is:
        0       2       5       1
        8       0       5       9
        1       3       0       7
        3       9       7       0

The Path is:
1--->4--->3--->2--->1

Minimum cost is 19

...Program finished with exit code 0
```

iii) Matrix Chain Multiplication

```
#include<iostream>
#include<stdio.h>
#include<limits.h>
using namespace std;

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainOrder(int p[], int n)
{

   /* For simplicity of the program, one extra row and one extra column are
      allocated in m[][].  0th row and 0th column of m[][] are not used */
   int m[n][n];

   int i, j, k, L, q;

   /* m[i,j] = Minimum number of scalar multiplications needed to compute
      the matrix A[i]A[i+1]...A[j] = A[i..j] where dimention of A[i] is
      p[i-1] x p[i] */

   // cost is zero when multiplying one matrix.
   for (i = 1; i < n; i++)
       m[i][i] = 0;

   // L is chain length.
   for (L=2; L<n; L++)
   {
```

```
        for (i=1; i<=n-L+1; i++)
        {
            j = i+L-1;
            m[i][j] = INT_MAX;
            for (k=i; k<=j-1; k++)
            {
                // q = cost/scalar multiplications
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n-1];
}

int main()
{
    int size;
    cout<<"Enter size"<<endl;
    cin>>size;
    int arr[size];
    cout<<"Enter elements"<<endl;
    for(int i=0;i<size;i++)
    cin>>arr[i];


    printf("Minimum number of multiplications is %d ",
                   MatrixChainOrder(arr, size));
```

```
    getchar();
    return 0;
}
```

```
Enter size
4
Enter elements
5 6 7 8
Minimum number of multiplications is 490

...Program finished with exit code 0
```