Part A

Aim:

1. Design algorithms for following problems.
   Euclid Algorithm, Matrix Multiplication, Matrix addition
2. Find time and space complexity of algorithms.
3. Implement the algorithms in any programming language.

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Procedure:

1. Design algorithm and find complexity
2. Implement algorithms in any programming language.
3. Paste output

Practice Exercise:

S.no	Query statement
1	Design and analysis Euclid algorithm for finding GCD of two numbers.
2	Design and analysis algorithm for matrix addition.
3	Design and analysis algorithm for matrix multiplication.

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

1)Design and analysis Euclid algorithm for finding GCD of two numbers.

ALGORITHM: Euclid algorithm for finding gcd of 2 numbers a,b

INPUT : Two integers a,b
OUTPUT : Largest integer that divides both the input numbers a,b (i.e gcd of a,b)

```
begin
gcd_euclid(a,b)
    begin while(b%a!=0)
        rem=b%a
        b=a
        a=rem
    end while
    return a
end
```

Amortized Analysis: (How much resource(time and space) our algorithm takes to execute)

Time Analysis:

In every iteration the reduction in sum (a+b) at minimum is 1.5.
so on solving $a+b/(3/2)^k=1$
We can get the value of k as $O(\log_{3/2}(a+b))$.
Time complexity of this algorithm will be in $O(\log_{3/2}(a+b))$.

Space Analysis:

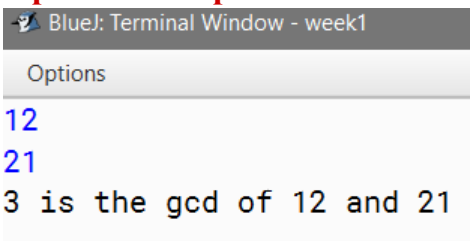
No of variables used here are 3 ,i.e a,b,rem where each variable takes a space of O(1) (so, 1+1+1=3 which is a constant) .
Therefore, space complexity will become O(1)

Code:

```
import java.util.*;
class euclid{
    public static int gcd_euclid(int a,int b)
    {
        int rem;
        while(b%a!=0)
        {
            rem=b%a;
            b=a;
            a=rem;
        }
        return a;
    }

    public static void main(String args[])
    {
        int a,b;
        Scanner sc=new Scanner(System.in);
        a=sc.nextInt();
        b=sc.nextInt();
        sc.close();
        System.out.println(gcd_euclid(a,b)+" is the gcd of "+a+" and "+b);
    }
}
```

Input and Output:



2) Design and analysis algorithm for matrix addition.

ALGORITHM: Addition of 2 matrices

Input : 2 matrices arr1 and arr2 of order n1*m1 and n2*m2
Output: Equivalent sum of matrices obtained of order n1*m1

```
begin
Matrix_addition( arr1[ ],arr2[ ])
    define addm matrix as (n1 * m1)
     if row size and column size of arr1==arr2:
          for(i=0;i<n1;i++)
              for(j=0;j<m1;j++)
                  addm[i][j]=arr1[i][j]+arr2[i][j];
                  System.out.print(addm[i][j]+"  ");
              System.out.println();
           return ;
    else
        System.out.println("Can't add the 2 matrices");
        return ;
end
```

Amortized Analysis: (How much resource(time and space) our algorithm takes to execute)

Time analysis:

Here in this algorithm we have 2 for loops where each loop if of O(n1),O(m1) and the loops are nested.So the time complexity becomes O(n1*m1) . For a square matrix where n1=m1 the order becomes O(n1^2) => **O(n^2)**

Space Analysis:

Here in this algorithm we used 4 variables(n1,m1,i,j) 3 matrices data structures of order n1*m1 n1*m1 and n1*m2 . So the space complexity will be (n1*m1) + (n1*m1) + (n1*m1) +4 i.e. O((n1*m1) + (n1*m1) + (n1*m1) +4) which is equal to O((n1*m1) + (n1*m1) + (n1*m1)).
If n1=m1 then space complexity will be in **O(n^2)**

Code:

```
import java.util.*;
class Matrix_add{
    public static void main(String args[])
    {
        int n1,m1,n2,m2,i,j;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter row size and column size of 2 matrices");
        n1=sc.nextInt();
        m1=sc.nextInt();
        n2=sc.nextInt();
        m2=sc.nextInt();
```

```
        if(n1!=n2 && m1!=m2){
            System.out.println("Can't add the 2 matrices");
            return ;
        }
        int arr1[][]=new int[n1][m1];
        int arr2[][]=new int[n2][m2];
        int addm[][]=new int[n1][n2];
        System.out.println("Enter elements of matrix 1");
        for(i=0;i<n1;i++){
            {
                for(j=0;j<m1;j++){
                    arr1[i][j]=sc.nextInt();
                }
            }
        }
        System.out.println("Enter elements of matrix 2");
        for(i=0;i<n1;i++){
            {
                for(j=0;j<m1;j++){
                    arr2[i][j]=sc.nextInt();
                }
            }
        }
        sc.close();
        System.out.println("Sum of both the matrices is ");

        for(i=0;i<n1;i++){
           {
                for(j=0;j<m1;j++){
                    addm[i][j]=arr1[i][j]+arr2[i][j];
                System.out.print(addm[i][j]+"  ");

                }
                System.out.println();
            }
        }
    }
}
```

Input & Output:

```
Enter row size and column size of 2 matrices
3 3
3 3
Enter elements of matrix 1
1 2 3
4 5 6
7 8 9
Enter elements of matrix 2
9 8 7
6 5 4
3 2 1
Sum of both the matrices is
10  10  10
10  10  10
10  10  10
```

3) Design and analysis algorithm for matrix multiplication.

Algorithm: Multiplication of matrices

Input : 2 matrices arr1 and arr2 of order n1*m1 and n2*m2

Output: Equivalent product of matrices obtained of order n1*m2

```
begin
Matrix_multiplication( arr1[ ],arr2[ ])
    define addm matrix as (n1 * m2)
     if m1==n2:
          for(i=0;i<n1;i++)
              for(j=0;j<m2;j++)
                  mul[i][j]=0;
                  for(k=0;k<m1;k++)
                      mul[i][j]+=arr1[i][k]*arr2[k][j];
                  System.out.print(mul[i][j]+"  ");
              System.out.println();
           return ;
    else
        System.out.println("Can't multiply the 2 matrices");
        return ;
end
```

Amortized Analysis: (How much resource(time and space) our algorithm takes to execute)

Time analysis:

Here in this algorithm we have 3 for loops where each loop if of O(n1),O(m2),O(m1) and the loops are nested.So the time complexity becomes O(n1*m2*m1) . For a square matrix where n1=m2=m1 the order becomes O(n1^3) =>

O(n^3)

Space Analysis:

Here in this algorithm we used 7 variables(n1,m1,n2,m2,i,j,k) 3 matrices data structures of order n1*m1 n2*m2 and n1*m2 . So the space complexity will be (n1*m1) + (n2*m2) + (n1*m2) +7 i.e. O((n1*m1) + (n2*m2) + (n1*m2) +6) which is equal to O((n1*m1) + (n2*m2) + (n1*m2)).

If n1=n2=m2 then space complexity will be in **O(n^2)**

CODE:

```
import java.util.*;
class Matrix_mul{
    public static void main(String args[])
    {
        int n1,m1,n2,m2,i,j,k;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter row size and column size of 2 matrices");
        n1=sc.nextInt();
        m1=sc.nextInt();
        n2=sc.nextInt();
        m2=sc.nextInt();
        if(m1!=n2){
            System.out.println("Can't multiply the 2 matrices");
            return ;
        }
        int arr1[][]=new int[n1][m1];
        int arr2[][]=new int[n2][m2];
        int mul[][]=new int[n1][n2];
        System.out.println("Enter elements of matrix 1");
        for(i=0;i<n1;i++){
            {
                for(j=0;j<m1;j++){
                    arr1[i][j]=sc.nextInt();
                }
            }
        }
        System.out.println("Enter elements of matrix 2");

        for(i=0;i<n2;i++){
            {
                for(j=0;j<m2;j++){
                    arr2[i][j]=sc.nextInt();
                }
            }
        }
```

```
        sc.close();
        System.out.println("product of both the matrices is ");
        for(i=0;i<n1;i++){
            {
                for(j=0;j<m2;j++){
                    mul[i][j]=0;
                    for(k=0;k<m1;k++)
                    {
                        mul[i][j]+=arr1[i][k]*arr2[k][j];
                    }
                System.out.print(mul[i][j]+"  ");
                }
                System.out.println();
            }
        }
    }
}
```

Input and output:

```
Enter row size and column size of 2 matrices
2 3
3 2
Enter elements of matrix 1
1 2 3
4 5 6
Enter elements of matrix 2
1 2
3 4
5 6
product of both the matrices is
22  28
49  64
```

Observation & Learning:

Learned about Amortized Analysis of an algorithm and found the time , space complexities of Euclid Algorithm, Matrix Multiplication, Matrix addition.

Implemented the algorithms in java programming language.

Conclusion:

Wrote algorithms for the 3 problem statements and executed them in Java programming language successfully

Questions:

1. What is the goodness criteria for algorithms?
2. How will two algorithms that solve the same problem be compared?

Answers:

1. Goodness of an algorithm is determined by:
   a. Characteristics of that algorithm (Should have 0/more inputs and at least 1 output, finiteness ,definiteness, effectiveness)
   b. Growth of the algorithm
   c. Time and space complexity of algorithm (The one with less time and space complexity is good)

2. They can be compared by finding time and space complexities of algorithms.