190905513 **MOHAMMAD DANISH EQBAL**

LAB 2

Lab Exercises:

1). Write a program to find GCD using consecutive integer checking method and analyze its time efficiency.
2). Write a program to find GCD using middle school method and analyze its time efficiency.
3). Write a program to find GCD using Euclid's algorithm and analyze its time efficiency.

Algorithms:

Euclid's algorithm for computing gcd(m, n)
Step 1 If n = 0, return the value of m as the answer and stop; otherwise, proceed to Step 2.
Step 2 Divide m by n and assign the value of the remainder to r.
Step 3 Assign the value of n to m and the value of r to n. Go to Step 1

Consecutive integer checking algorithm for computing gcd(m, n)
Step 1 Assign the value of min{m, n} to t.
Step 2 Divide m by t. If the remainder of this division is 0, go to Step 3; otherwise, go to Step 4.
Step 3 Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step 4.
Step 4 Decrease the value of t by 1. Go to Step 2

Middle-school procedure for computing gcd(m, n)
Step 1 Find the prime factors of m.
Step 2 Find the prime factors of n.
Step 3 Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If p is a common factor occurring pm and pn times in m and n, respectively, it should be repeated min{pm, pn} times.)
Step 4 Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

Program:
```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


int opcount1, opcount2, opcount3;
```

```
int euclidGCD(int n, int m)
{
  unsigned int r;
  int opcount = 0;
  while (n != 0)
  {
    opcount++;
    r = m % n;
    m = n;
    n = r;
  }
  printf("Operation count of Euclid's GCD method = % d\n", opcount);
  return m;
}

int consecutiveInt(int n, int m)
{
  int t = m > n ? n : m;
  int opcount = 0;
  while (t > 0)
  {
    if (m % t == 0)
      if (n % t == 0)
      {
        printf("Operation count of Consecutive Integer method = %d\n", opcount);
        return t;
      }
    t--;
    opcount++;
  }
```

```
}

int *Sieve(int n)
{
  int p, j, *A, *L;
  opcount1 = 0;
  A = (int *)malloc(sizeof(int) * (n+1));
  L = (int *)malloc(sizeof(int) * 200);
  for(int i=0; i<200; i++)L[i]=0;
  for (p = 2; p <= n; p++)
  {
    A[p] = p;
  }
  for (int p = 2; p < sqrt(n); p++)
  {
    opcount1++;
    if (A[p] != 0)
    {
      j = p * p;
      while (j <= n)
      {
        A[j] = 0;
        j += p;
      }
    }
  }
  int i = 0;
  for (p = 2; p <= n; p++)
  {
    if (A[p] != 0)
```

```
    {
      L[i] = A[p];
      i++;
    }
  }
  return L;
}

int *Divide(int m, int *Prime)
{
  int *PrimeFactor, i;
  PrimeFactor = (int *)malloc(sizeof(int) * 200);
  for(int i=0; i<200; i++)PrimeFactor[i]=0;
  opcount2 = 0;
  i = 0;
  int j = 0;
  while (m > 0 && j <= 9 && Prime[j]!=0)
  {
    opcount2++;
    while (m % Prime[j] == 0 && Prime[j]!=0)
    {
      PrimeFactor[i] = Prime[j];
      i++;
      m /= Prime[j];
    }
    j++;
  }
  return PrimeFactor;
}
```

```
int middleSchool(int m, int n)
{
  int *PrimesM, *PrimesN, *PrimeFactorsM, *PrimeFactorsN, *CommonFactors;
  PrimesM = (int *)malloc(sizeof(int) * 200);
  PrimesN = (int *)malloc(sizeof(int) * 200);
  PrimeFactorsM = (int *)malloc(sizeof(int) * 200);
  PrimeFactorsN = (int *)malloc(sizeof(int) * 200);
  CommonFactors = (int *)malloc(sizeof(int) * 200);
  for(int i=0; i<200; i++)CommonFactors[i]=0;
  PrimesM = Sieve(m);
  for (int i = 0; i < 200 && PrimesM[i]!=0; i++)
    printf("Prime Number = %d\n", PrimesM[i]);
  PrimeFactorsM = Divide(m, PrimesM);
  for (int i = 0; i < 200 && PrimeFactorsM[i]!=0; i++)
    printf("Prime Factor of M = %d\n", PrimeFactorsM[i]);
  PrimeFactorsN = Divide(n, PrimesM);
  for (int i = 0; i < 200 && PrimeFactorsN[i]!=0; i++)
    printf("Prime Factor of N = %d\n", PrimeFactorsN[i]);
  int k = 0;
  opcount3 = 0;
  for (int i = 0, j = 0; (i < 200 && PrimeFactorsM[i]!=0) || (j < 200 &&
PrimeFactorsN[i]!=0);)
  {
    opcount3++;
    if (PrimeFactorsM[i] > PrimeFactorsN[j])
      j++;
    else if (PrimeFactorsM[i] < PrimeFactorsN[j])
      i++;
    else if (PrimeFactorsM[i] == PrimeFactorsN[j])
    {
      CommonFactors[k] = PrimeFactorsM[i];
```

```
      k++;
      i++;
      j++;
    }
  }
  for (int i = 0; i < 200 && CommonFactors[i]!=0; i++)
  {
    printf("Common Factor = %d\n", CommonFactors[i]);
  }
  int gcd = 1;
  for (int i = 0; i<k; i++)
  {
    gcd *= CommonFactors[i];
  }
  if (opcount3 > opcount2 && opcount3 > opcount1)
    printf("OPCOUNT3 = %d\n", opcount3);
  else if (opcount2 > opcount3 && opcount2 > opcount1)
    printf("OPCOUNT2 = %d\n", opcount2);
  else
    printf("OPCOUNT1 = %d\n", opcount1);
  return gcd;
}

int main()
{
  int m, n;
  printf("Enter the numbers whose GCD needs to be calculated : \n");
  printf("Number 1 : ");
  scanf("%d", &n);
  printf("Number 2 : ");
```

```
  scanf("%d", &m);
  printf("GCD using Euclid's GCD method = %d\n", euclidGCD(n, m));
  printf("GCD using Consecutive Integer method = %d\n", consecutiveInt(n, m));
  printf("GCD using Middle School method = %d\n", middleSchool(n, m));
}
```

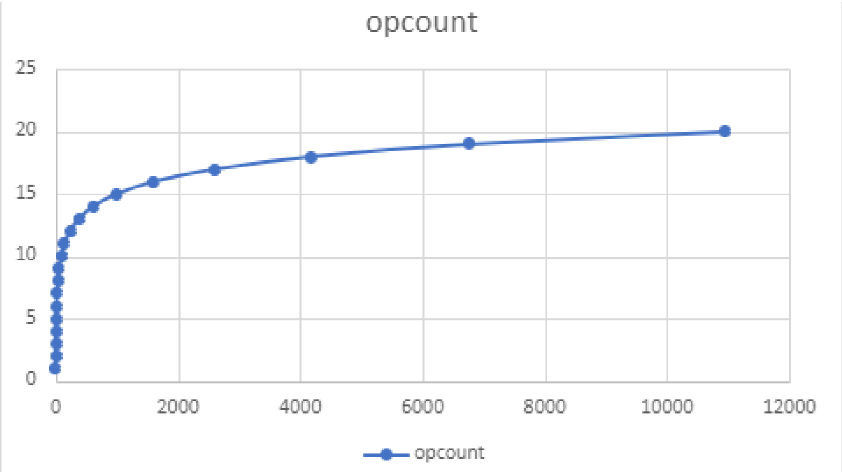
Output:

```
Enter the numbers whose GCD needs to be calculated :
Number 1 : 2
Number 2 : 3
Operation count of Euclid's GCD method =  2
GCD using Euclid's GCD method = 1
Operation count of Consecutive Integer method = 1
GCD using Consecutive Integer method = 1
Prime Number = 2
Prime Factor of M = 2
OPCOUNT3 = 204
GCD using Middle School method = 1

Process returned 0 (0x0)   execution time : 10.527 s
Press any key to continue.
```

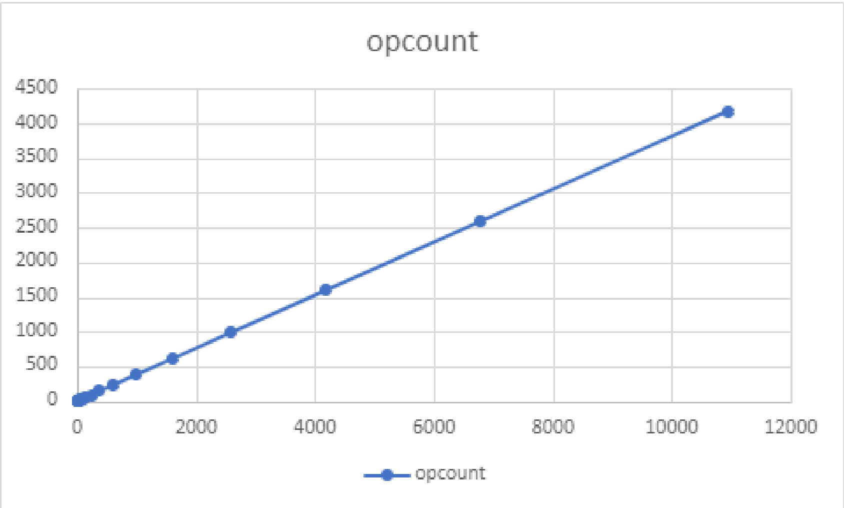
```
Enter the numbers whose GCD needs to be calculated :
Number 1 : 1
Number 2 : 1
Operation count of Euclid's GCD method =  1
GCD using Euclid's GCD method = 1
Operation count of Consecutive Integer method = 0
GCD using Consecutive Integer method = 1
OPCOUNT1 = 0
GCD using Middle School method = 1

Process returned 0 (0x0)   execution time : 4.051 s
Press any key to continue.
```

Graphs

1. Euclid



2. Consecutive Integer Checking



3. Middle school