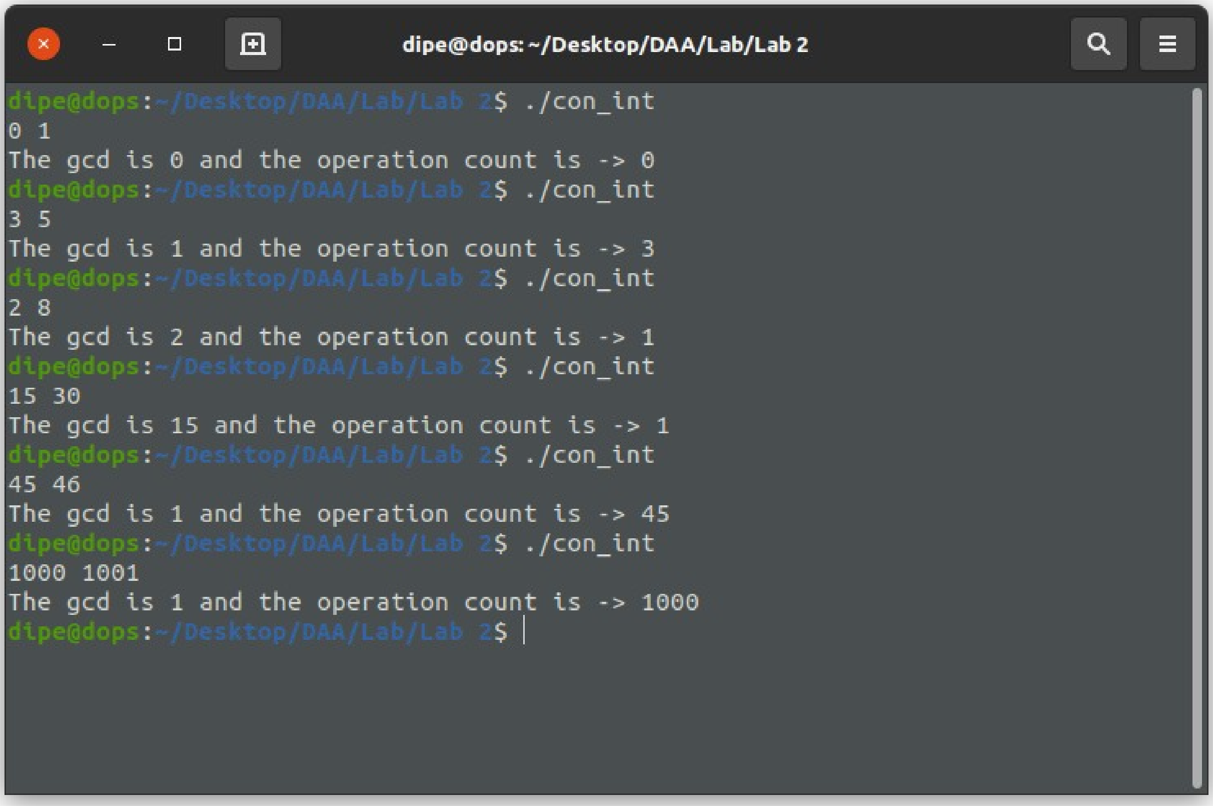
Lab 2 :

Question 1 :

Write a program to find GCD using consecutive integer checking method and analyze its time efficiency.

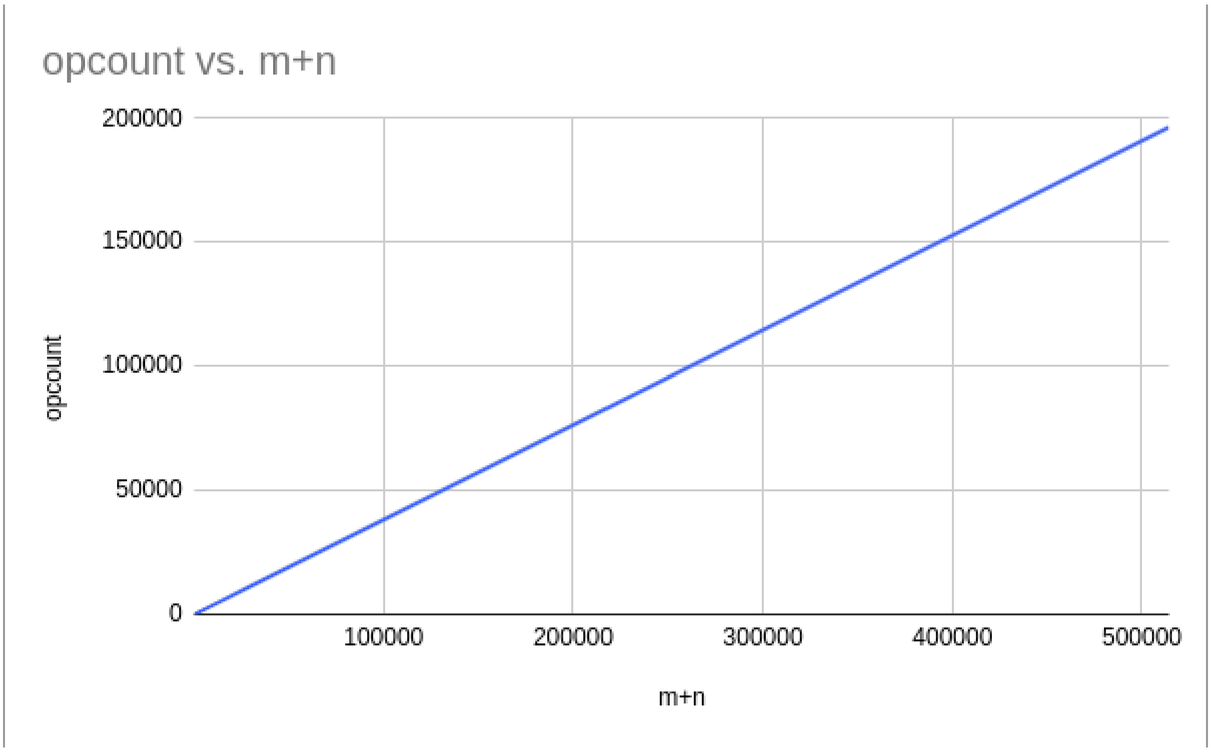
```
#include <stdio.h>
#include <stdlib.h>

int gcd(int m, int n, int *opr)
{
for (int i = (m > n ? n : m); i > 0; i--)
{
(*opr)++;
if (m % i == 0 && n % i == 0)
{
return i;
}
}
}

int main()
{
int m, n;
scanf("%d %d", &m, &n);
int opr = 0;
int result = gcd(m, n, &opr);
printf("The gcd is %d and the operation count is -> %d\n", result, opr);
}
```



```
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
0 1
The gcd is 0 and the operation count is -> 0
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
3 5
The gcd is 1 and the operation count is -> 3
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
2 8
The gcd is 2 and the operation count is -> 1
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
15 30
The gcd is 15 and the operation count is -> 1
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
45 46
The gcd is 1 and the operation count is -> 45
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./con_int
1000 1001
The gcd is 1 and the operation count is -> 1000
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ 
```

The basic operation for this algorithm is **comparison.**

m+n	opcount
1	0
2	1
5	2
8	3
13	5
21	8
34	13
55	21
89	34
144	55
233	89
377	144
28657	10946
514229	196418



As we can observe from the graph plot that the time complexity for the following algorithm is **O(min(m,n))** which belongs to **O(n+m).**

Question 2 :

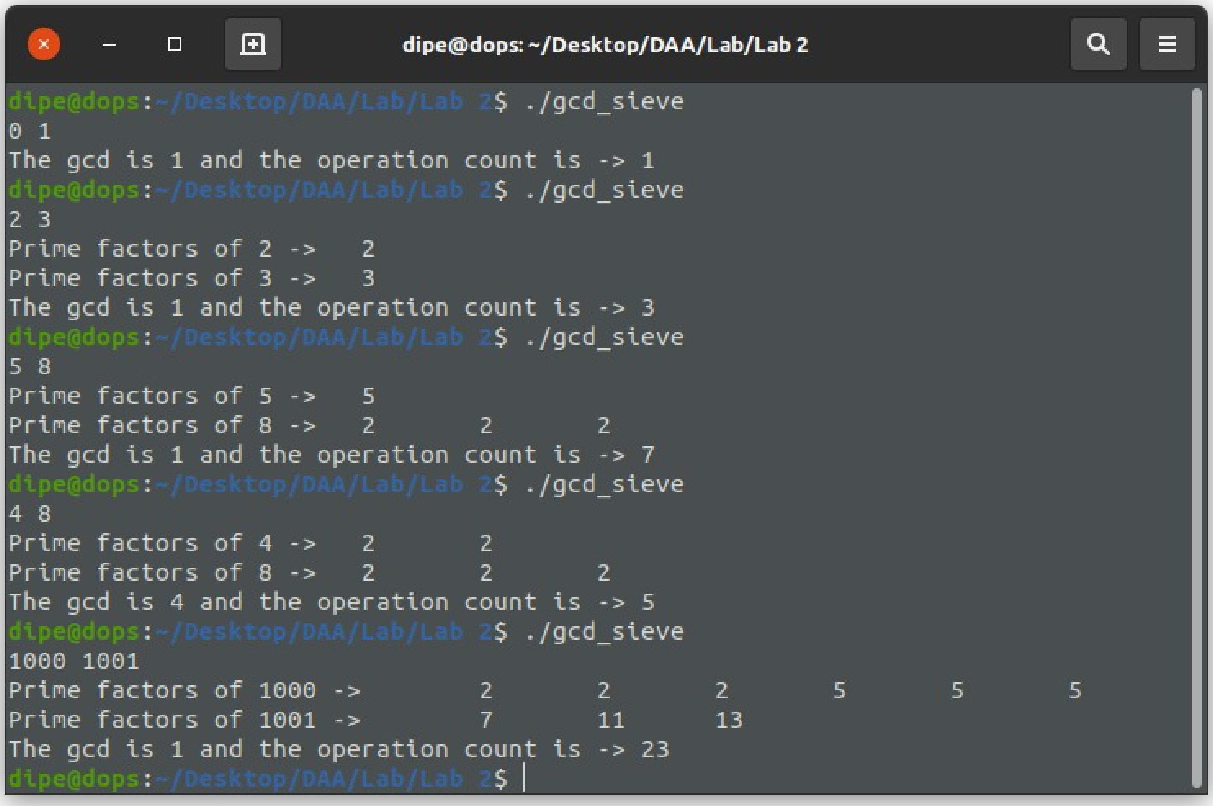
Write a program to find GCD using middle school method and analyze its time efficiency.

```

#include <stdio.h>
#include <stdlib.h>

void sieve(int m, int arr[])
{
for (int i = 2; i < m + 1; i++)
{
arr[i] = i;
}
int j;
```

```
for (int i = 2; i < m + 1; i++)
{
if (arr[i] != 0)
{
j = i * i;
while (j <= m)
{
arr[j] = 0;
j = j + i;
}
}
}
}

int pf(int n, int arr[], int *op)
{
int narr[n + 1];
sieve(n, narr);
int i = 2;
int cnt = 0;
while (i <= n)
{
(*op)++;
if (narr[i] != 0)
{
if (n % narr[i] == 0)
{
arr[cnt] = narr[i];
n = n / narr[i];
cnt++;
}
else
{
i++;
}
}
else
{
i++;
}
```

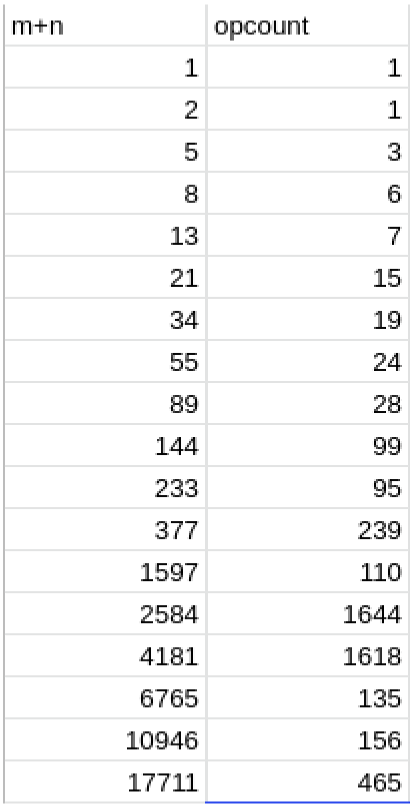
```
}
return cnt;
}

int gcd(int m, int n, int *opr)
{
if (m == 0 || n == 0)
{
*opr = 1;
return m == 0 ? n : m;
}
int marr[m], narr[n], op1 = 0, op2 = 0;
int a = pf(m, marr, &op1);
int b = pf(n, narr, &op2);
*opr = op1 + op2;
printf("Prime factors of %d -> \t", m);
for (int i = 0; i < a; i++)
{
printf("%d\t", marr[i]);
}
printf("\nPrime factors of %d -> \t", n);
for (int i = 0; i < b; i++)
{
printf("%d\t", narr[i]);
}
printf("\n");
int i = 0, j = 0;
int result = 1;
while (i < a && j < b)
{
if (marr[i] == narr[j])
{
result *= marr[i];
i++;
j++;
}
else if (marr[i] < narr[j])
{
i++;
}
```

```
else
{
j++;
}
}
return result;
}

int main()
{
int m, n;
scanf("%d %d", &m, &n);
int opr = 0;
int result = gcd(m, n, &opr);
printf("The gcd is %d and the operation count is -> %d\n", result, opr);
return 0;
}
```

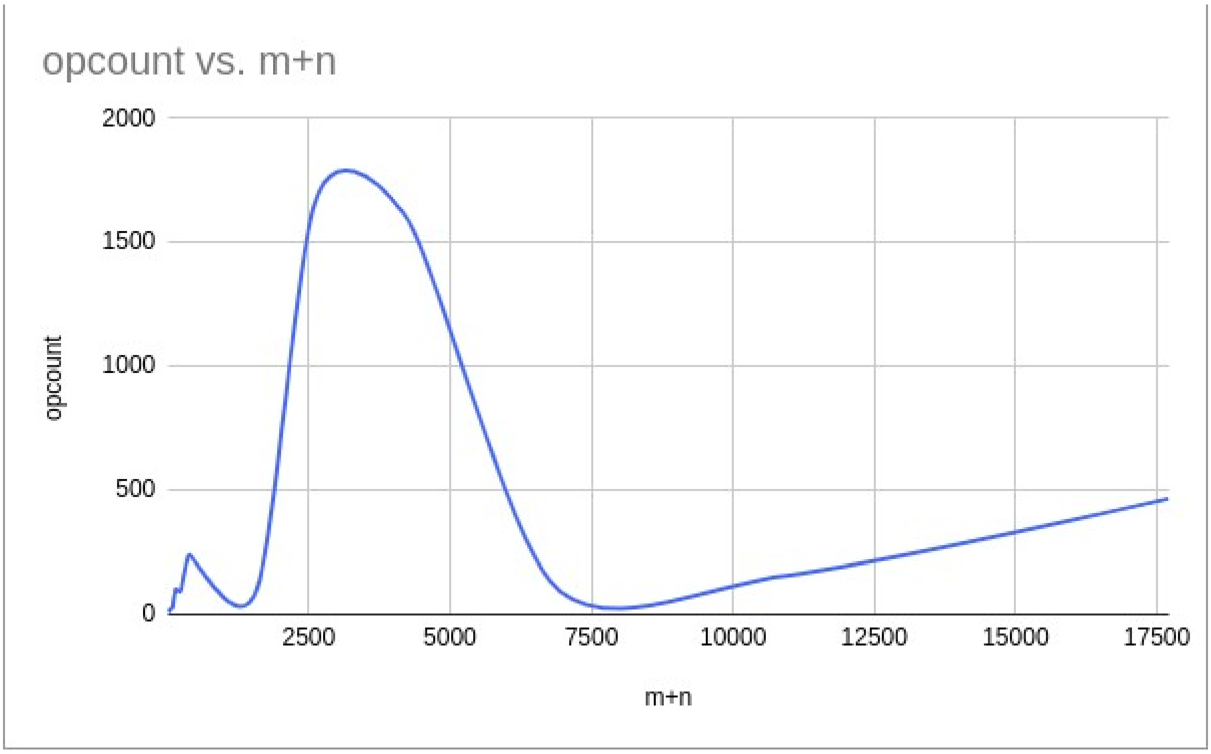


```
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./gcd_sieve
0 1
The gcd is 1 and the operation count is -> 1
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./gcd_sieve
2 3
Prime factors of 2 ->   2
Prime factors of 3 ->   3
The gcd is 1 and the operation count is -> 3
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./gcd_sieve
5 8
Prime factors of 5 ->   5
Prime factors of 8 ->   2       2       2
The gcd is 1 and the operation count is -> 7
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./gcd_sieve
4 8
Prime factors of 4 ->   2       2
Prime factors of 8 ->   2       2       2
The gcd is 4 and the operation count is -> 5
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ ./gcd_sieve
1000 1001
Prime factors of 1000 ->        2       2       2       5       5       5
Prime factors of 1001 ->        7       11      13
The gcd is 1 and the operation count is -> 23
dipe@dops:~/Desktop/DAA/Lab/Lab 2$ 
```

The basic operation is **comparison.**



m+n	opcount
1	1
2	1
5	3
8	6
13	7
21	15
34	19
55	24
89	28
144	99
233	95
377	239
1597	110
2584	1644
4181	1618
6765	135
10946	156
17711	465



As we can observe from the graph plot that the time complexity for the following algorithm is **O(n+m).**