NUMBER THEORY

LESSON – 1
GCD AND LCM

Topics to be covered

1. GCD & LCM
2. Basic Euclidean algorithm
3. Extended Euclidean algorithm
4. LCM of array
5. GCD of array
6. Linear Diophantine equation

1.GCD AND LCM

GCD

The HCF or GCD of two integers is the largest integer that can exactly divide both numbers (without a remainder).

Example :-

24 = 2 X 2 X 2 X 3

10 = 2 X 5

GCD = 1 X 2 = 2

LCM

The LCM is the smallest positive integer that is evenly divisible by both a and b.

Example :-

LCM(2,3) = 6

LCM(6,10) = 30.

Relation between GCD and LCM is : -

If the numbers are (a,b) then :-

LCM(A,B) x GCD(A,B) = (A*B)

Naïve Approach for gcd:

1. Traverse from 2 to n-1.

2. Check if a%i==0 && b%i==0

3. If the above condition satisfies than that no. can be the hcf but we have to continue this loop as there can be more such numbers or value of i as we need to find the highest value.

2. Basic Euclidean Algorithm for GCD

The algorithm is based on the below facts.

1. If we subtract a smaller number from a larger (we reduce a larger number), GCD doesn't change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.
2. Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find remainder 0.

Code:

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

// Function to return
// gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Driver Code
int main()
{
    int a = 10, b = 15;
    cout << "GCD(" << a << ", "
         << b << ") = " << gcd(a, b)
                        << endl;
    a = 35, b = 10;
    cout << "GCD(" << a << ", "
         << b << ") = " << gcd(a, b)
                        << endl;
    a = 31, b = 2;
    cout << "GCD(" << a << ", "
         << b << ") = " << gcd(a, b)
                        << endl;
    return 0;}
```

Time Complexity: O(Log min(a, b))

Q1:

Minimize cost to make X and Y equal by given increments

Given two integers X and Y, the task is to make both the integers equal by performing the following operations any number of times:

Increase X by M times. Cost = M – 1.

Increase Y by N times. Cost = N – 1.

Examples:

Input: X = 2, Y = 4

Output: 1

Explanation:

Increase X by 2 times. Therefore, X = 2 * 2 = 4. Cost = 1.

Clearly, X = Y. Therefore, total cost = 1.

Input: X = 4, Y = 6

Output: 3

Explanation:

Increase X by 3 times, X = 3 * 4 = 12. Cost = 2.

increase Y by 2 times, Y = 2 * 6 = 12. Cost = 1.

Clearly, X = Y. Therefore, total cost = 2 + 1 = 3

Native app:

```
1.  int minimumCost(int x, int y)
2.  {
3.          int costx = 0, dup_x = x;
4.          while (true) {
5.          int costy = 0, dup_y = y;
6.          // Check if it possible
7.          // to make x == y
8.          while (dup_y != dup_x
9.              && dup_y < dup_x) {
10.          dup_y += y;
11.          costy++;
12.         }
13.
14.         // Iif both are equal
15.         if (dup_x == dup_y)
16.          return costx + costy;
17.
18.         // Otherwise
19.         else {
20.             // Increment cost of x
21.          // by 1 and dup_x by x
22.             dup_x += x;
23.             costx++;
24.         }
25.         }
```

```
}

Efficient sol:
int gcd(int x, int y)
{
    if (y == 0)
      return x;
    return gcd(y, x % y);
}

// Function to find lcm of x and y
int lcm(int x, int y)
{
    return (x * y) / gcd(x, y);
}

// Function to find minimum Cost
int minimumCost(int x, int y)
{
    int lcm_ = lcm(x, y);
    // Subtracted initial cost of x
    int costx = (lcm_ - x) / x;
    // Subtracted initial cost of y
    int costy = (lcm_ - y) / y;
    return costx + costy;
}
```

3.Extended Euclidean Algorithm:

Extended Euclidean algorithm also finds integer coefficients x and y such that:

ax + by = gcd(a, b)

Examples:

Input: a = 30, b = 20

Output: gcd = 10

x = 1, y = -1

(Note that 30*1 + 20*(-1) = 10)

Input: a = 35, b = 15

Output: gcd = 5

x = 1, y = -2

(Note that 35*1 + 15*(-2) = 5)

The extended Euclidean algorithm updates results of gcd(a, b) using the results calculated by recursive call gcd(b%a, a). Let values of x and y calculated by the recursive call be x_1 and y_1. x and y are updated using the below expressions.

$$x = y_1 - \lfloor b/a \rfloor * x_1$$

$$y = x_1$$

Code:

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

// Function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
            *x = 0;
            *y = 1;
            return b;
     }

     int x1, y1; // To store results of recursive call
     int gcd = gcdExtended(b%a, a, &x1, &y1);

     // Update x and y using results of
     // recursive call
     *x = y1 - (b/a) * x1;
     *y = x1;

     return gcd;
}

// Driver Code
int main()
{
     int x, y, a = 35, b = 15;
     int g = gcdExtended(a, b, &x, &y);
     cout << "GCD(" << a << ", " << b
          << ") = " << g << endl;
     return 0;
}
```

Derivation:

(b % a)*X + a*Y=g --(1)

a*X + b*Y=g --(2)

we know: b%a=b- [b/a]*a --(3)

put (3) in (1)

(b- (b/a)*a)*X + a*Y =g

b*X1 + a*[Y1- (b/a)*X1] =g

Comparing this equation with (2)

Y=X1

X= Y1- (b/a)*X1

4.LCM OF AN ARRAY

Example : -

Input : {1, 2, 8, 3}

Output : 24

ALGORITHM :-

1. Initialize ans = arr[0].
2. Iterate over all the elements of the array i.e. from i = 1 to i = n-1

At the ith iteration ans = LCM(arr[0], arr[1], ……., arr[i-1]).
This can be done easily as **LCM(arr[0], arr[1], …., arr[i])** = **LCM(ans, arr[i])**. Thus at i'th iteration we just have to do **ans = LCM(ans, arr[i]) = ans x arr[i] / gcd(ans, arr[i])**

C++ CODE :-

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
ll findlcm(int arr[], int n)
{
    ll ans = arr[0];
     for (int i = 1; i < n; i++)
        ans = (((arr[i] * ans)) /
                (gcd(arr[i], ans)));
    return ans;
}

int main()
{
    int arr[] = { 2, 7, 3, 9, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("%lld", findlcm(arr, n));
    return 0;}

```

5.GCD OF ARRAY

The GCD of three or more numbers equals the product of the prime factors common to all the numbers, but it can also be calculated by repeatedly taking the GCDs of pairs of numbers.

gcd(a, b, c) = gcd(a, gcd(b, c)) = gcd(gcd(a, b), c) = gcd(gcd(a, c), b)

ALGORITHM :-

result = arr[0]

For i = 1 to n-1

result = GCD(result, arr[i])

C++ CODE :-

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. int gcd(int a, int b)
4. {
5.    if (a == 0)
6.       return b;
7.    return gcd(b % a, a);
8. }
9.
10.   int findGCD(int arr[], int n)
```

```
11.{
12.     int result = arr[0];
13.     for (int i = 1; i < n; i++)
14.     {
15.         result = gcd(arr[i], result);
16.
17.         if(result == 1)
18.         {
19.             return 1;
20.         }
21.     }
22.     return result;
23. }
```

Time Complexity: *O(N * log(M)), where M is the smallest element of the array*

6.Linear Diophantine Equation:

A Linear Diophantine Equation (in two variables) is an equation of the general form:

$$ax+by=c$$

1. find a solution:

To find one solution of the Diophantine equation with 2 unknowns, you can use the Extended Euclidean algorithm. First, assume that a and b are non-negative. When we apply Extended Euclidean algorithm for a and b, we can find their greatest common divisor g and 2 numbers X and Y such that:

$$aX+bY=c$$

If c is divisible by g= gcd(a,b) then the given Diophantine equation has a solution, otherwise it does not have any solution. The proof is straight-forward: a linear combination of two numbers is divisible by their common divisor.

Mul both sides by c/g.

a.X*c/g +bY*c/g =g*c/g

Therefore one of the solutions of the Diophantine equation is:

$$X0=X*c/g$$

$$Y0=Y*c/g$$

Code:

```
1.  #include <iostream>
2.  using namespace std;
3.  int extendedEuclid(int a,int b,int &x,int &y)
4.  {
5.          if(a==0){
6.          x=0;
7.          y=1;
8.          return b;
9.          }
10.         int x0, y0;
11.         int g=extendedEuclid(b%a,a,x0,y0);
12.         x=y0-(b/a)*x0;
13.         y=x0;
14.         return g;
15.     }
16.     bool find_any_sol(int a,int b,int c,int &x,int &y)
17.     {
18.         int x0,y0;
19.         int g=extendedEuclid(abs(a),abs(b),x0,y0);
20.       cout<<"x0:"<<x0<<" "<<"y0:"<<y0<<endl;
21.         if(c%g!=0)
22.         return 0;
23.
```

```
24.       x=(x0*c)/g;
25.       y=(y0*c)/g;
26.
27.       if(a<0)
28.       x=-x;
29.       if(b<0)
30.       y=-y;
31.       return 1;
32.
33.   }
34.
35.   //finding any sol for LDA.
36.   int main() {
37.    int t;
38.    cin>>t;
39.    while(t--){
40.       int a,b,c,x,y;
41.        cin>>a>>b>>c;
42.        if(!find_any_sol(a,b,c,x,y))
43.       cout<<"no sol..";
44.       else
45.          cout<<x<<" "<<y<<endl;
46.    }
47.   }
```

Question 2:

Minimum number to be added to minimize LCM of two given numbers

Given two numbers A and B, the task is to find the minimum number that needs to be added to A and B such that their LCM is minimized.

Examples:

Input: A = 6, B = 10

Output: 2

Explanation: On adding 2 to A and B, the value becomes 8 and 12 respectively. LCM of 8 and 12 is 24, which is the minimum LCM possible.

Input: A = 5, B = 10

Output: 0

Explanation:

10 is already the minimum LCM of both the given number.

Hence, the minimum number added is 0.

q3:

Find the pair (a, b) with minimum LCM such that their sum is equal to N

Given a number N, the task is to find two numbers a and b such that a + b = N and LCM(a, b) is minimum.

Examples:

Input: N = 15

Output: a = 5, b = 10

Explanation:

The pair 5, 10 has a sum of 15 and their LCM is 10 which is the minimum possible.

Input: N = 4

Output: a = 2, b = 2

Explanation:

The pair 2, 2 has a sum of 4 and their LCM is 2 which is the minimum possible

Approach: The idea is to use the concept of GCD and LCM. Below are the steps:

If N is a Prime Number then the answer is 1 and N – 1 because in any other cases either a + b > N or LCM(a, b) is > N – 1. This is because if N is prime then it implies that N is odd. So a and b, any one of them must be odd and other even. Therefore, LCM(a, b) must be greater than N (if not 1 and N – 1) as 2 will always be a factor.

If N is not a prime number then choose a, b such that their GCD is maximum, because of the formula LCM(a, b) = a*b / GCD (a, b). So, in order to minimize LCM(a, b) we must maximize GCD(a, b).

If x is a divisor of N, then by simple mathematics a and b can be represented as N / x and N / x*(x – 1) respectively. Now as a = N / x and b = N / x * (x – 1), so their GCD comes out as N / x. To maximize this GCD, take the smallest possible x or smallest possible divisor of N.

```
1.  void minDivisior(int n)
2. {
3.
4.        // Check if the number is prime
5.        if (prime(n)) {
6.        cout << 1 << " " << n - 1;
7.        }
8.
9.        // Now, if it is not prime then
10.       // find the least divisior
11.       else {
12.       for (int i = 2; i * i <= n; i++) {
13.
```

```
14.            // Check if divides n then
15.            // it is a factor
16.            if (n % i == 0) {
17.
18.               // Required output is
19.               // a = n/i & b = n/i*(n-1)
20.               cout << n / i << " "
21.                    << n / i * (i - 1);
22.               break;
23.            }
24.      }
25.      }
26.   }
```

q4:

Find M such that GCD of M and given number N is maximum

M<N

Given an integer N greater than 2, the task is to find an element M such that GCD(N, M) is maximum.

Examples:

Input: N = 10

Output: 5

Explanation:

gcd(1, 10), gcd(3, 10), gcd(7, 10), gcd(9, 10) is 1,

gcd(2, 10), gcd(4, 10), gcd(6, 10), gcd(8, 10) is 2,

gcd(5, 10) is 5 which is maximum.

Input: N = 21

Output: 7

Explanation:

gcd(7, 21) is maximum among all the integers from 1 to 21.

Efficient Approach: To optimize the above approach, we observe that the GCD of two numbers will be definitely one of its divisors in the range [1, N-1]. And, GCD will be maximum if the divisor is maximum.

```
1.  int findMaximumGcd(int n)
2.  {
3.          // Initialize a variable
4.          int max_gcd = 1;
5.
```

```
6.      // Find all the divisors of N and
7.      // return the maximum divisor
8.      for (int i = 1; i * i <= n; i++) {
9.
10.     // Check if i is divisible by N
11.     if (n % i == 0) {
12.
13.         // Update max_gcd
14.         if (i > max_gcd)
15.             max_gcd = i;
16.
17.         if ((n / i != i)
18.             && (n / i != n)
19.             && ((n / i) > max_gcd))
20.             max_gcd = n / i;
21.     }
22.     }
23.
24.     // Return the maximum value
25.     return max_gcd;
26. }
```

q5:

Minimum LCM of all pairs in a given array

Given an array arr[] of size N, the task is to find the minimum LCM (Least Common Multiple) of all unique pairs in the given array, where 1 <= N <= 105, 1 <= arr[i] <= 105.

Examples:

Input: arr[] = {2, 4, 3}

Output: 4

Explanation

LCM (2, 4) = 4

LCM (2, 3) = 6

LCM (4, 3) = 12

Minimum possible LCM is 4.

Input: arr [] ={1, 5, 2, 2, 6}

Output: 2

```
1.  int minLCM(int arr[], int n)
2.  {
3.          int mx = 0;
4.          for (int i = 0; i < n; i++) {
```

```
5.
6.        // find max element in the array as
7.        // the gcd of two elements from the
8.        // array can't greater than max element.
9.        mx = max(mx, arr[i]);
10.       }
11.
12.       // created a 2D array to store minimum
13.       // two multiple of any particular i.
14.      vector<vector<int> > mul(mx + 1);
15.
16.       for (int i = 0; i < n; i++) {
17.       if (mul[arr[i]].size() > 1) {
18.             // we already found two
19.             // smallest multiple
20.             continue;
21.       }
22.         mul[arr[i]].push_back(arr[i]);
23.       }
24.
25.       // iterating over all gcd
26.       for (int i = 1; i <= mx; i++) {
27.
28.       // iterating over its multiple
```

```
29.      for (int j = i + i; j <= mx; j += i) {
30.
31.          if (mul[i].size() > 1) {
32.
33.              // if we already found the
34.              // two smallest multiple of i
35.              break;
36.           }
37.          for (int k : mul[j]) {
38.              if (mul[i].size() > 1)
39.                  break;
40.              mul[i].push_back(k);
41.           }
42.      }
43.      }
44.
45.     int ans = INT_MAX;
46.      for (int i = 1; i <= mx; i++) {
47.
48.      if (mul[i].size() <= 1)
49.          continue;
50.
51.      // choosing smallest two multiple
52.      int a = mul[i][0], b = mul[i][1];
```

```
53.
54.        // calculating lcm
55.        int lcm = (a * b) / i;
56.
57.        ans = min(ans, lcm);
58.        }
59.
60.        // return final answer
61.        return ans;
62.    }
63.
```

Time Complexity: O((N + M) * log(M))

q6:

Sum of GCD of all numbers upto N with N itself

Given an integer N, the task is to find the sum of Greatest Common Divisors of all numbers up to N with N itself.

Examples:

Input: N = 12

Output: 40

Explanation:

GCD of [1, 12] = 1, [2, 12] = 2, [3, 12] = 3, [4, 12] = 4, [5, 12] = 1, [6, 12] = 6, [7, 12] = 1, [8, 12] = 4, [9, 12] = 3, [10, 12] = 2, [11, 12] = 1, [12, 12] = 12. The sum is (1 + 2 + 3 + 4 + 1 + 6 + 1 + 4 + 3 + 2 + 1 + 12) = 40.

Input: N = 2

Output: 3

Explanation:

GCD of [1, 2] = 1, [2, 2] = 2 and their sum is 3

Naive Approach: A simple solution is to iterate over all numbers from 1 to N and find their gcd with N itself and keep on adding them.

Time Complexity: O(N * log N)

Efficient Approach:

To optimize the above-mentioned approach, we need to observe that GCD(i, N) gives one of the divisors of N. So, instead of running a loop from 1 to N, we can check for each divisor of N that how many numbers are there with GCD(i, N) same as that divisor.

```
1. int getCount(int d, int n)
2. {
3.
```

```
4.      int no = n / d;
5.
6.      int result = no;
7.
8.      // Consider all prime factors
9.      // of no. and subtract their
10.     // multiples from result
11.     for (int p = 2; p * p <= no; ++p) {
12.
13.     // Check if p is a prime factor
14.     if (no % p == 0) {
15.
16.         // If yes, then update no
17.           // and result
18.         while (no % p == 0)
19.             no /= p;
20.         result -= result / p;
21.     }
22.     }
23.
24.     // If no has a prime factor greater
25.     // than sqrt(n) then at-most one such
26.     // prime factor exists
27.     if (no > 1)
```

```
28.         result -= result / no;
29.
30.       // Return the result
31.       return result;
32.   }
33.
34.   // Finding GCD of pairs
35.   int sumOfGCDofPairs(int n)
36.   {
37.       int res = 0;
38.
39.       for (int i = 1; i * i <= n; i++) {
40.       if (n % i == 0) {
41.             // Calculate the divisors
42.             int d1 = i;
43.             int d2 = n / i;
44.
45.             // Return count of numbers
46.             // from 1 to N with GCD d with N
47.             res += d1 * getCount(d1, n);
48.
49.             // Check if d1 and d2 are
50.             // equal then skip this
51.              if (d1 != d2)
```

```
52.                 res += d2 * getCount(d2, n);
53.         }
54.         }
55.
56.         return res;
57.     }
58.
59.
```

q7:

Minimize sum of K positive integers with given LCM

Given two positive integers K and X, the task is to find the minimum possible sum of K positive integers (repetitions allowed) having LCM X.

Examples:

Input: K = 2, X = 6

Output: 5

Explanation:

K(= 2) positive integers of minimum possible sum having LCM X(= 6) are { 2, 3 }.

Therefore, the minimum possible sum of K(= 2) positive integers = (2 + 3) = 5.

Therefore, the required output is 5.

Input: K = 3 X = 11

Output: 13

Explanation:

K(= 3) positive integers of minimum possible sum having LCM X(= 11) are { 1, 1, 11 }.

Therefore, the minimum possible sum of K(= 3) positive integers = (1 + 1 + 11) = 13.

Therefore, the required output is 13.

Approach: The problem can be solved using Greedy technique. The idea is to represent X in the form of a product of prime powers. Select K prime powers from all the prime powers of X in all possible ways and calculate their respective sums. Finally, print the minimum possible sum among all of them. Follow the steps below to solve the problem:

Initialize an array, say primePow[], to store all the prime powers of X.

If length of the array primePow[] is less than or equal to K, then include all the array elements of primePow[] in K positive integer and the remaining element Of K positive integers must be 1. Finally, print the sum of K positive integers

Illustration:

If K = 5, X = 240

primePow[] = { 24, 31, 51 } = { 16, 3, 5 } x ,ī

K positive integers will be { 16, 3, 5, 1, 1 }

Therefore, the sum of K positive integers will be 26

Otherwise, partition the primePow[] array into K groups in all possible ways and calculate their respective sums. Finally, print the minimum sum of the K positive integers.

If K = 3, X = 210

primePow[] = { 21, 31, 51, 51 } = { 2, 3, 5, 7 }

Partition primePow[] array into { { 2, 3 }, { 5 }, { 7 } }

210 = (2 * 3) * (5) * 7 = 6 * 5 * 7

K positive integers will be { 6, 5, 7 }

Therefore, the sum of K(= 3) positive integers will be 18