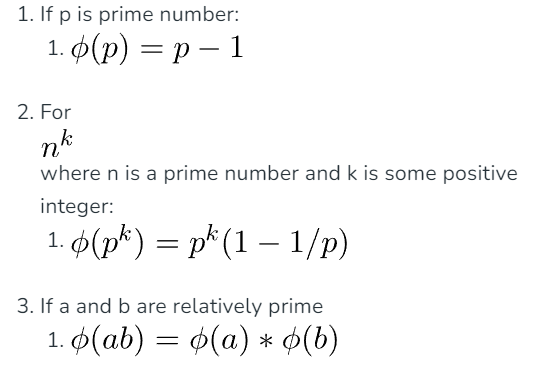
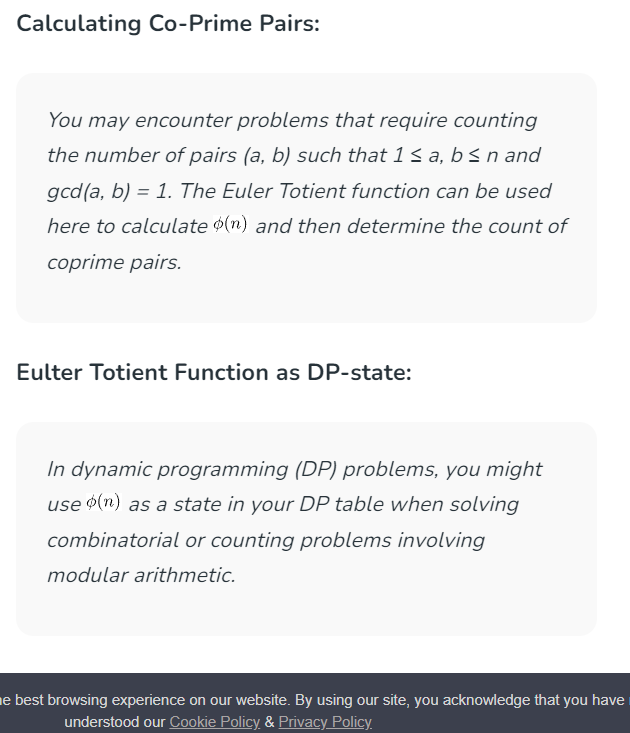
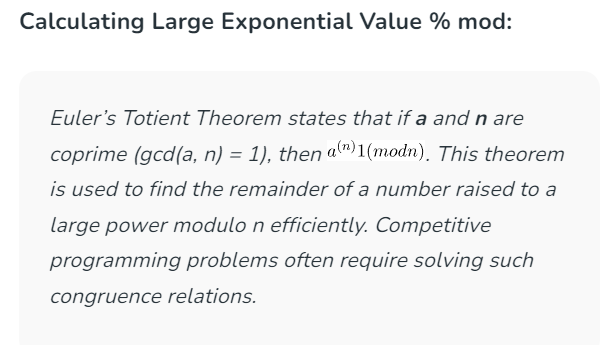
GCD and LCM

1. GCD of two number.
   1. The fastest way to find the GCD of two number is by using the “Euclidean algorithm” in time complexity O(log(min(a,b))).
   2. It uses recursion to repeatedly replace A with B and B with the remainder of A divided by B until B become zero. Returning A as the GCD
      1. **int** gcd (**int** a, **int** b) {
      2. **if** (b == 0)
      3. **return** a;
      4. **else**
      5. **return** gcd (b, a % b);
      6. }
   3. Problem identification that involve GCD

* Problems that require to determine **if one number is divisible by anothe**r may involve GCD, as GCD is related to the greatest common factor of two numbers.
* Many number theory problems which related to [**coprime numbers**](https://www.geeksforgeeks.org/check-two-numbers-co-prime-not/), **relatively prime numbers**, or**Euler’s totient function**, involve GCD calculation.
* Consider whether the problem hints about the need to factorize numbers into their prime factors. GCD problems sometimes involve [**prime factorization**](https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/) to find **common factors** efficiently.
* In graph theory, GCD can be used in problems related to **connected components**, in which a edge between the nodes will only exist if they are not **co-prime**.

1. Euclidean Algorithm
   1. The algorithm is based on below facts.
   2. If we subtract a smaller number from a larger one. GCD doesn’t change. So if we keep subtracting repeatedly the larger of two, we end up with the GCD.
   3. Now instead of subtracting, if we divide the smaller number, the algorithm stops when we find the remainder 0.
   4. Refer to the code at point 1.
   5. Time complexity – O(Log min(a, b ) )
   6. Auxiliary Space- O(Log min(a, b) )
2. Extended Euclidean Algorithm ( Have to check again)
   1. Extended Euclidean algorithm also find integer coefficients x and y such that : ax + by = gcd(a, b).
   2. The extended Euclidean algorithm is particularly useful when a and b are coprime (or gcd is 1). Since x is the modular multiplicative inverse of “a modulo b”, and y is the modular multiplicative inverse of “b modulo a”. In particular, the computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.
3. Euler Totient function(ETF)
   1. *Euler Totient Function or Phi-function for ‘****n’****, gives the count of integers in range ‘****1′****to ‘****n’****that are co-prime to ‘****n’****. It is denoted by .*
   2. *For example the below table shows the ETF value of first 20 +ve interger.*
   3. 
   4. 3 Important properties of Euler Totient Function:
   5. 
   6. Euler Totient Function in O(sqrt{n}) using prime factorization
      1. int ETF(int n) {
      2. int phi\_n = n;
      3. for (int i = 2; i \* i <= n; i++) {
      4. if (n % i == 0) {
      5. while (n % i == 0)
      6. n /= i;
      7. phi\_n -= phi\_n / i;
      8. }
      9. }
      10. if (n > 1)
      11. phi\_n -= phi\_n / n;
      12. return result;
      13. }
4. How to store ETF for each integer 1 to n optimally
   1. The above code shows that we can calculate the ETF value for ‘n’ in O(\sqrt{n} ), If we want to calculate ETF value for each integer from 1 to n then it will take us O(n\sqrt{n}) .
   2. Can we optimize this time complexity? YES we can, using precomputation similar to Sieve of Eratosthenes we can reduce the time complexity to O(n\*log(log n)) as shown in below code
      1. vector<int> ETF\_1\_to\_n(int n) {
      2. vector<int> phi(n + 1);
      3. for (int i = 0; i <= n; i++)
      4. phi[i] = i;
      5. for (int i = 2; i <= n; i++) {
      6. if (phi[i] == i) {
      7. for (int j = i; j <= n; j += i)
      8. phi[j] -= phi[j] / i;
      9. }
      10. }
      11. return phi
      12. }
   3. Key hints to identify a problem uses the knowledge of ETF.
5. GCD and LCM based problems
6. Use of Chinese Remainder Theorem
7. Use of Fermat’s Little Theorem
8. Problem requires calcuation of Large Exponentiations such as a^b^c
9. 
10. Stein’s Algorithm for finding GCD
    1. It use to compute the GCD of two non-negative integer. Stein’s algorithm replaces division with arithmetic shifts, comparisons and substraction.
    2. Algorithm: -
       1. If both a and b are 0, gcd is zero gcd(0, 0) = 0.
       2. gcd(a, 0) = a and gcd(0, b) = b because everything divides 0.
       3. If a and b are both even, gcd(a, b) = 2\*gcd(a/2, b/2) because 2 is a common divisor. Multiplication with 2 can be done with bitwise shift operator.
       4. If a is even and b is odd, gcd(a, b) = gcd(a/2, b). Similarly, if a is odd and b is even, then   
          gcd(a, b) = gcd(a, b/2). It is because 2 is not a common divisor.
       5. If both a and b are odd, then gcd(a, b) = gcd(|a-b|/2, b). Note that difference of two odd numbers is even
       6. Repeat steps 3–5 until a = b, or until a = 0. In either case, the GCD is power(2, k) \* b, where power(2, k) is 2 raise to the power of k and k is the number of common factors of 2 found in step 3.
       7. Time complexity : O(N\*N) . N is no of bits in number.
       8. Auxiliary space: iterative - O(1), recursive – O(N\*N).
       9. Stein’s algorithm is optimized version of Euclid’s GCD algorithm
       10. It is more efficient by using the bitwise shift operator.
11. LCM of given array
    1. LCM(a, b) = (a\*b) / gcd(a, b);
    2. Algorithm for LCM of an given array:-
       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])**
    3. Time Complexity: O(n\* log(min(a,b) ) ), where n represents the size of the given array.
    4. Auxiliary space: O(n\* log(min(a, b) ) ) due to recursive stack space
12. GCD, LCM and Distributive property
    1. GCD( LCM(x, y), LCM(x,z) ) = LCM(x, GCD(y, z) ).

Coding Problems

1. Check if two numbers are co-prime or not.
   * Two number A and B are said to be co-prime or mutually prime if the GCD of them is 1.
   * EX: Input: 2 3 Output : co-prime
   * EX: Input: 4 8 Output: Not Co-prime
2. GCD of an array.
   * Gcd(a, b, c) = gcd(a, gcd(b, c) )
3. Find the other number when the LCM and HCF and a number is given.
   * Given a number A and LCM and HCF. The task is to determine the other number B.
   * Formula :-
   * A\*B = LCM \* HCF
   * B = (LCM \* HCF) / A;
4. Minimum insertion to make a Co-prime array.
   * Given an array of N elements, find the minimum number of insertions to convert the given array into a co-prime array. Print the resultant array also.
   * **Co-prime array :** An array in which every pair of adjacent elements are co-prime.
   * https://www.geeksforgeeks.org/problems/make-coprime-array3058/1?itm\_source=geeksforgeeks&itm\_medium=article&itm\_campaign=bottom\_sticky\_on\_article
5. Find the minimum possible health of the wining player
   * Given an array of N elements, find the minimum number of insertions to convert the given array into a co-prime array. Print the resultant array also.
   * Given an array **health[]** where **health[i]** is the health of the **ith** player in a game, any player can attack any other player in the game. The health of the player being attacked will be reduced by the amount of health the attacking player has. The task is to find the minimum possible health of the winning player.
   * ***Input:****health[] = {4, 6, 8}****Output:****2   
     4 attacks 6, health[] = {4, 2, 8}   
     2 attacks 4 twice, health[] = {0, 2, 8}   
     2 attacks 8 four times, health[] = {0, 2, 0}****Input:****health[] = {4, 1, 5, 3}****Output:****1*
   * **Approach:** In order to minimize the health of the last player, only the player with the smaller health will attack a player with the larger health and by doing so if only two players are involved then the minimum health of the last player is nothing but the GCD of the initial health’s of the two players. So, the result will be the GCD of all the elements of the given array.
6. Minimum squares to evenly cut a rectangle
   * Given a rectangular sheet of length l and width w. we need to divide this sheet into square sheets such that the number of square sheets should be as minimum as possible.
   * *Input :l= 4 w=6   
     Output :6   
     We can form squares with side of 1 unit, But the number of squares will be 24, this is not minimum. If we make square with side of 2, then we have 6 squares. and this is our required answer.   
     And also we can’t make square with side 3, if we select 3 as square side, then whole sheet can’t be converted into squares of equal length.*
   * Optimal length of the side of a square is equal to [GCD](https://www.geeksforgeeks.org/basic-and-extended-euclidean-algorithms/) of two numbers
7. Minimum operation to make GCD of array a multiple of k
   * Given an array and k, we need to find the minimum operations needed to make GCD of the array equal or multiple of k. Here an operation means either increment or decrements an array element by 1.
   * ***Input :****a = { 4, 5, 6 }, k = 5****Output :****2****Explanation :****We can increase 4 by 1 so that it becomes 5 and decrease 6 by 1 so that it becomes 5. Hence minimum operation will be 2.*
   * ***Input :****a = { 4, 9, 6 }, k = 5****Output :****3****Explanation :****Just like the previous example we can increase and decrease 4 and 6 by 1 and increase 9 by 1 so that it becomes 10. Now each element has GCD 5. Hence minimum operation will be 3.*
   * Here we have to make the gcd of the array equal or multiple to k, which means there will be cases in which some elements are near k or to some of its multiple. So, to solve this we just have to make each array value equal to or multiple to K. By doing this we will achieve our solution as if every element is multiple of k then it’s GCD will be at least K. Now our next target is to convert the array elements in the minimum operation i.e. minimum number of increment and decrement. This minimum value of increment or decrement can be known only by taking the remainder of each number from K i.e. *either we have to take the remainder value or (k-remainder) value, whichever is minimum among them.*
8. GCD of two number.
9. GCD of two number.
10. GCD of two number.
11. GCD of two number.