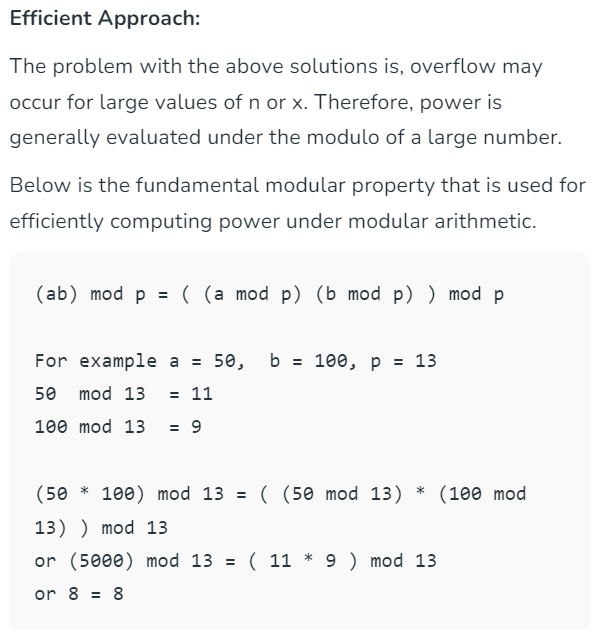
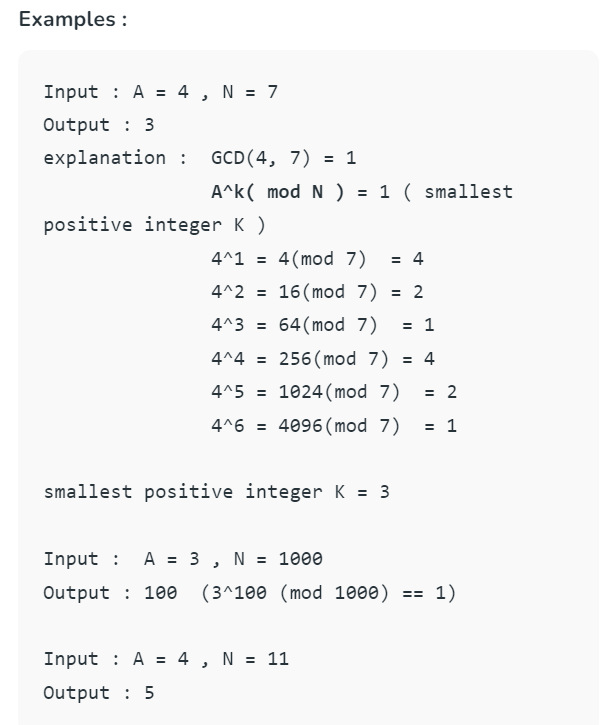
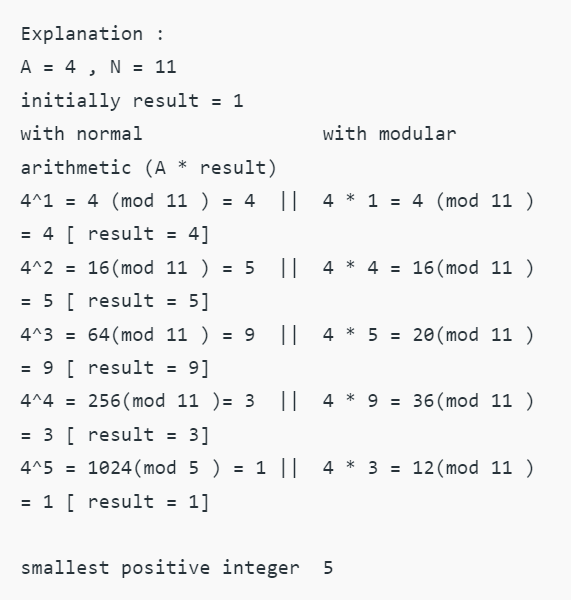
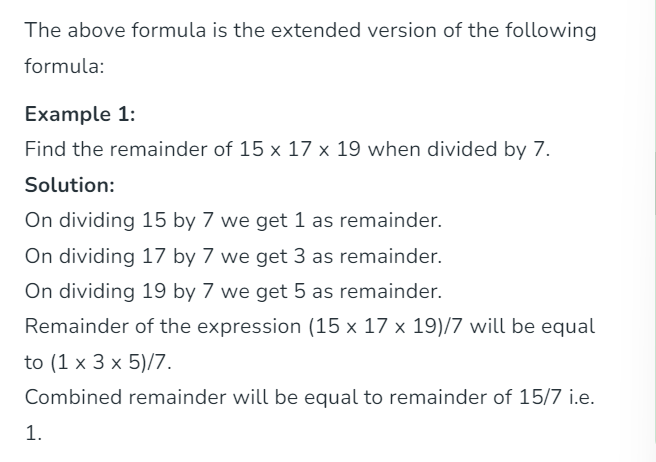
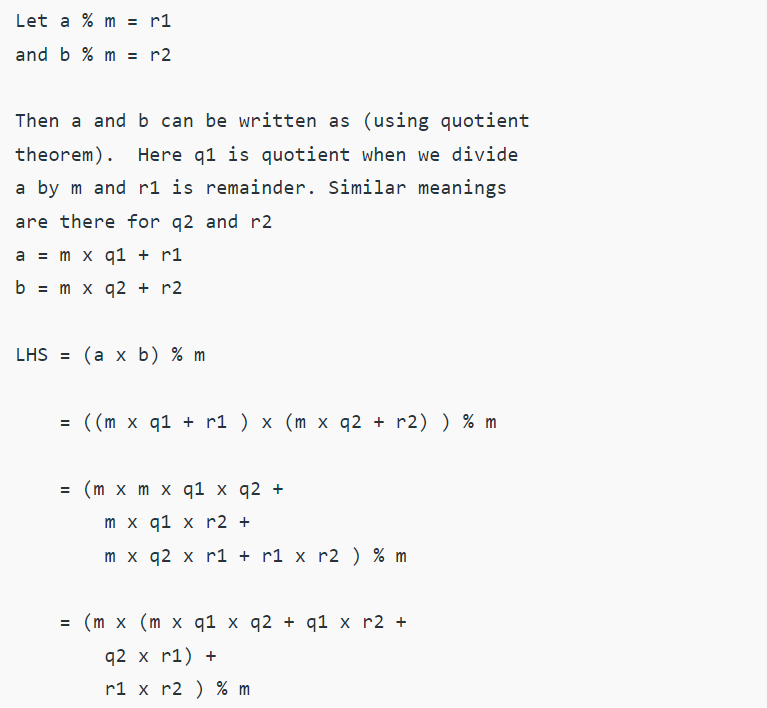
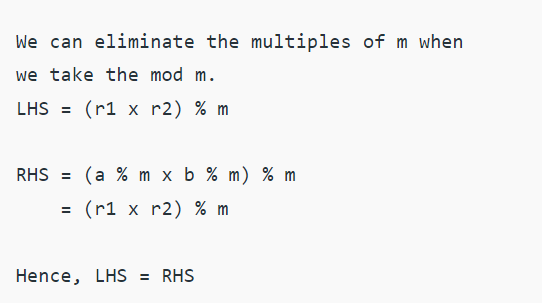
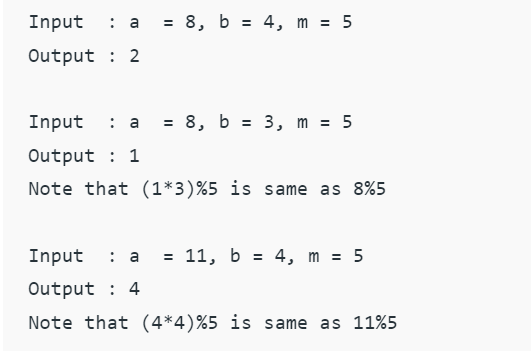
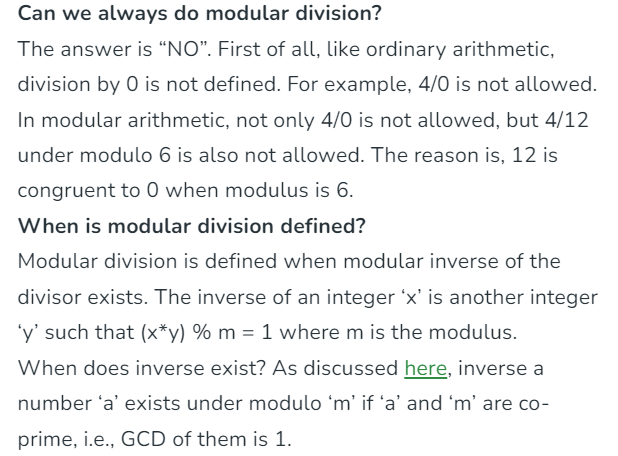
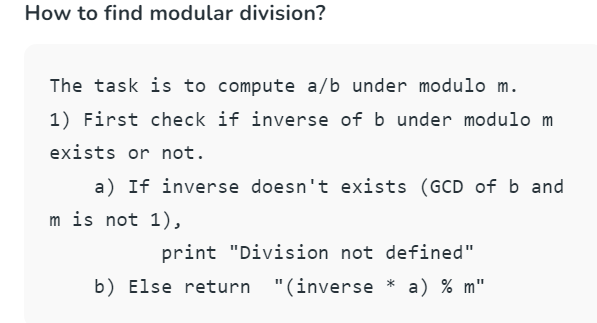
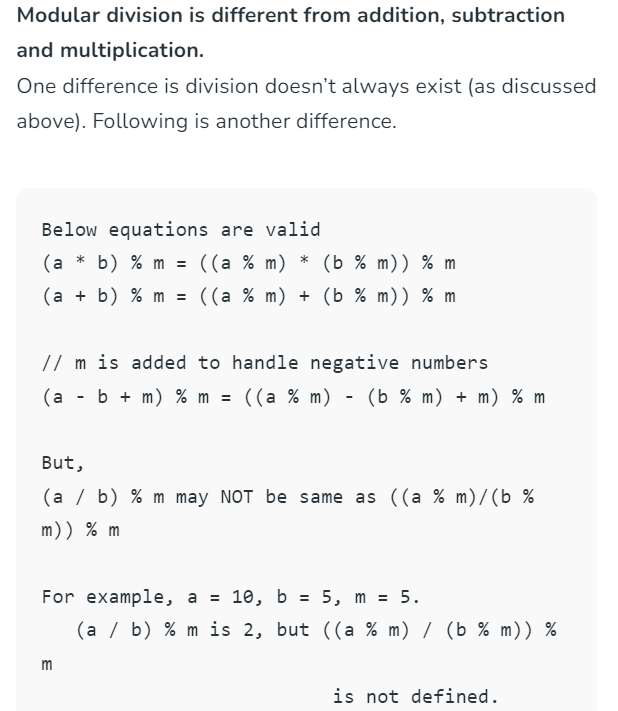
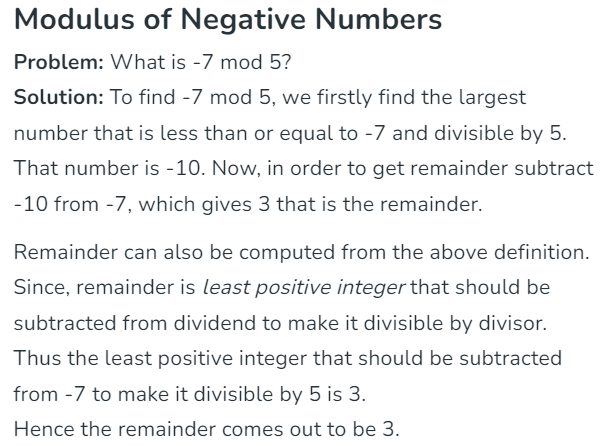
**Modular Exponentiation(Power in modular arithmetic)**

1. **Modular Exponentiation**
   1. Given three numbers x, y and p, compute (xy) % p.
   2. ****
   3. **Time** **Complexity:** O(Log y), where y represents the value of the given input.
   4. **Auxiliary Space:**O(1), as we are not using any extra space.
2. **Modular Multiplicative inverse**
   1. Given two integers **A** and **M**, find the modular multiplicative inverse of**A**under modulo **M**.  
      The modular multiplicative inverse is an integer **X**such that:
   2. *A X ≅ 1 (mod M)*
   3. **Note:** The value of**X** should be in the range {1, 2, … M-1}, i.e., in the range of integer modulo **M**. ( Note that **X** cannot be0as A\*0 mod M will never be 1). The multiplicative inverse of “A modulo M” exists if and only if A and M are relatively prime (i.e. if gcd(A, M) = 1)
3. **Multiplicative order**
   1. In number theory, given an integer A and a positive integer N with gcd( A , N) = 1, the multiplicative order of a modulo N is the smallest positive integer k with A^k( mod N ) = 1. ( 0 < K < N )
   2. ****
   3. If we take a close look then we observe that we do not need to calculate power every time. we can be obtaining next power by multiplying ‘A’ with the previous result of a module.
   4. ****
   5. Run a loop from 1 to N-1 and Return the smallest +ve power of A under modulo n which is equal to 1.
   6. **Time Complexity:** O(N)
   7. **Space Complexity:** O(1)
4. **Modular Multiplication**
   1. (a x b) mod m = ((a mod m) x (b mod m)) mod m
   2. (a x b x c) mod m = ((a mod m) x (b mod m) x (c mod m)) mod m
   3. The same property holds for more than three numbers.
   4. If we need to find remainder of multiplication of two large numbers, we can avoid doing the multiplication of large numbers, especially helpful in programming where multiplication of large numbers can cause overflow.
   5. ****
   6. ****
   7. ****
5. **Modular Division**
   1. Given three positive numbers a, b and m. Compute a/b under modulo m. The task is basically to find a number c such that (b \* c) % m = a % m.
   2. ****
   3. ****
   4. ****
   5. ****
6. **Modular on Negative Number**
   1. The modulus operator, denoted as %, returns the remainder when one number (the dividend) is divided by another number (the divisor).
   2. A remainder is**least positive integer** that should be subtracted from a to make it divisible by b (mathematically if, a = q\*b + r then 0 ≤ r < |b|), where a is dividend, b is divisor, q is quotient and r is remainder.
   3. ****
   4. Let’s see the result of -7 mod 5 in different programming languages:
   5. **Note:**The python program gives 3 as the remainder, meanwhile the other programming languages (C/C++) gives -2 as the remainder of -7 mod 5. The reason behind this is Python uses floored division to find modulus.
   6. As we know that **Remainder = Dividend – (Divisor \* Quotient)**and Quotient can be computed from Dividend and Divisor. To find the quotient there are two methods, which determine the sign of the remainder.
   7. **Floored division:**
      1. In this method, Quotient is determined by the floor division of Dividend and Divisor. It rounds off the quotient to the nearest smallest integer. So remainder can be computed from the following expression:
      2. r = a − b\* floor(a/b), where r, a, b are Remainder, Dividend and Divisor respectively.
      3. => -7 % 5 = -7 – 5 \*( floor(-7/5) )  
         =  -7 – 5\*( floor(-1.4) )  
         =  -7 – 5\*( -2)  
         = -7+10  
         = 3
      4. In Python, floor division is used to determine the remainder.
   8. Truncated Division:
      1. In this method, Quotient is determined by the truncate divison of Dividend and Divisor. It rounds off the quotient towards zero. So remainder can be computed from the following expression:
      2. r = a − b\* trunc(a/b), where r, a, b are Remainder, Dividend and Divisor respectively.
      3. => -7 % 5 = -7 – 5 \*( trunc(-7/5) )  
         =  -7 – 5\*( trunc(-1.4) )  
         =  -7 – 5\*( -1)  
         = -7+5  
         = -2
      4. In C/C++, **truncate division** is used to determine the remainder
   9. Thus, In programming languages which uses truncate divison to find the remainder, we always find remainder as **(a%b + b)%b** (add quotient to remainder and again take remainder) to avoid negative remainder and get the correct result.
7. **Modular Exponentiation**