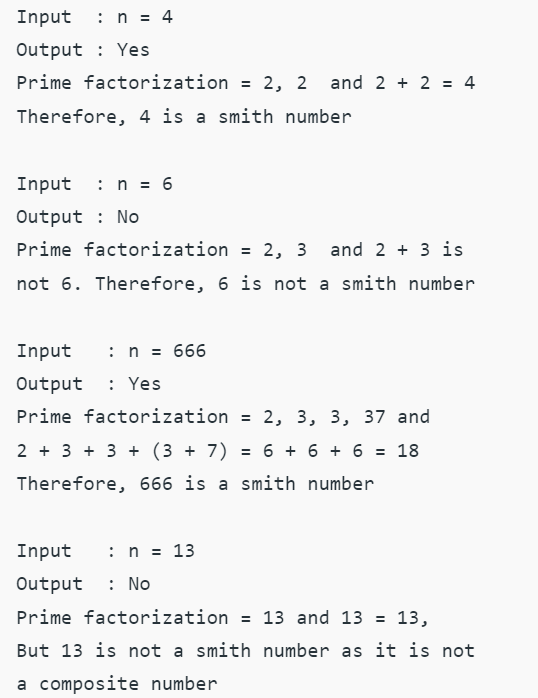
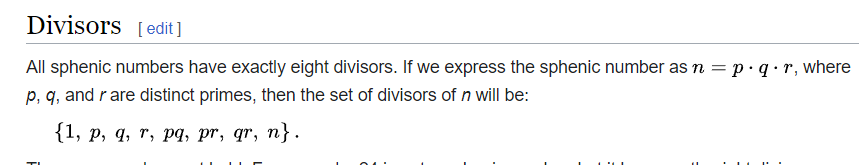
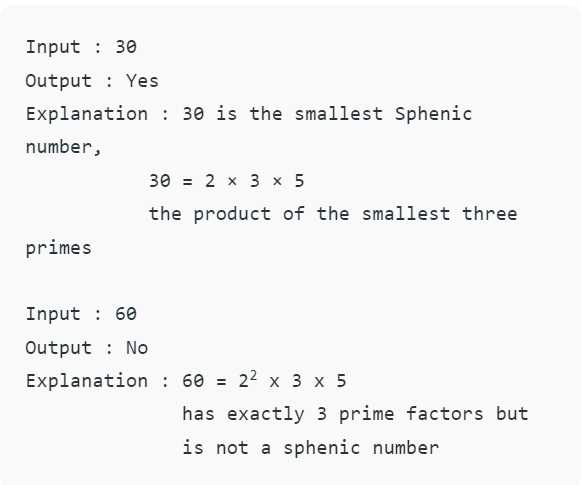
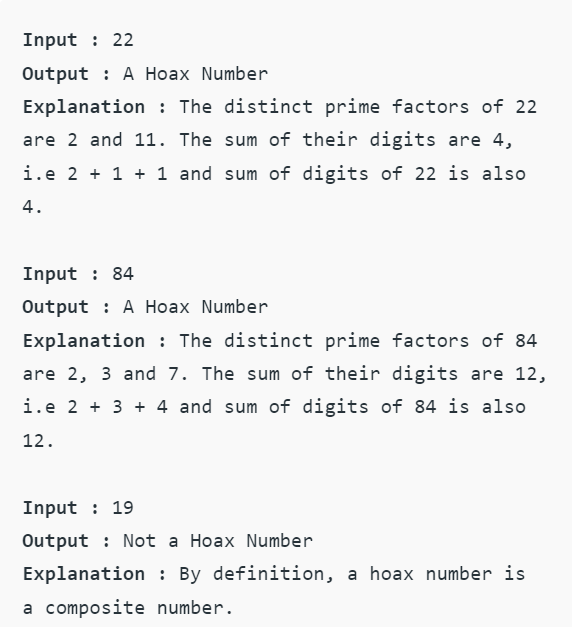
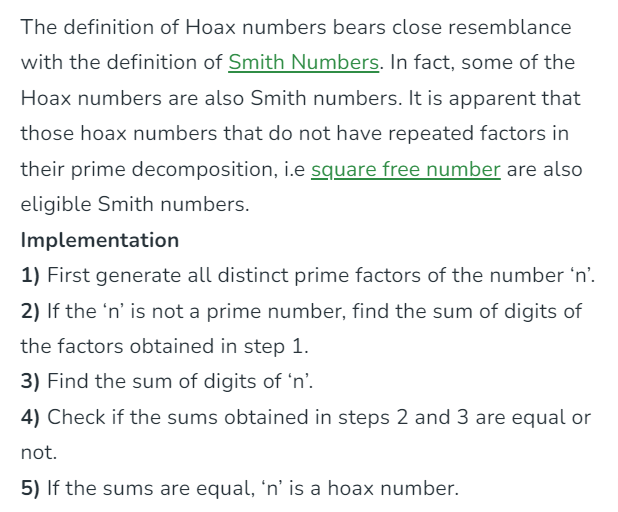
Prime Factorization & Divisors

1. Find a Prime Factor
   1. Given a number **n**, write an efficient function to print all [prime factors](https://www.geeksforgeeks.org/prime-factorization/#:~:text=the%20original%20number.-,What%20are%20Prime%20Factors%3F,-Prime%20factors%20are) of **n**. For example, if the input number is 12, then the output should be “2 2 3”. And if the input number is 315, then the output should be “3 3 5 7”.
   2. **First Approach:**
   3. Following are the steps to find all prime factors.  
      **1)** While n is divisible by 2, print 2 and divide n by 2.  
      **2)** After step 1, n must be odd. Now start a loop from i = 3 to the square root of n. While i divides n, print i, and divide n by i. After i fails to divide n, increment i by 2 and continue.  
      **3)** If n is a prime number and is greater than 2, then n will not become 1 by the above two steps. So print n if it is greater than 2.
   4. In the worst case ( when either n or sqrt(n) is prime, for example: take n=11 or n=121 for both the cases for loop runs sqrt(n) times), the for loop runs for sqrt(n) times. The more number of times the while loop iterates on a number it reduces the original n, which also reduces the value of sqrt(n). Although the best case time complexity is O(log(n)), when the prime factors of n is only 2 and 3 or n is of the form (2^x\*(3^y) where x>=0 and y>=0.
   5. **Time Complexity:**O(sqrt(n))
   6. **Auxiliary Space:** O(1)
   7. **How does this work?**  
      The steps 1 and 2 take care of composite numbers and step 3 takes care of prime numbers. To prove that the complete algorithm works, we need to prove that steps 1 and 2 actually take care of composite numbers. This is clear that step 1 takes care of even numbers. And after step 1, all remaining prime factors must be odd (difference of two prime factors must be at least 2), this explains why i is incremented by 2.
   8. Now the main part is, the loop runs till the square root of n not till n. To prove that this optimization works, let us consider the following property of composite numbers.
   9. *Every composite number has at least one prime factor less than or equal to*the *square root of itself.*  
      This property can be proved using a counter statement. Let a and b be two factors of n such that a\*b = n. If both are greater than √n, then a.b > √n, \* √n, which contradicts the expression “a \* b = n”.
   10. In step 2 of the above algorithm, we run a loop and do the following in loop   
       a) Find the least prime factor i (must be less than √n,)  
       b) Remove all occurrences i from n by repeatedly dividing n by i.  
       c) Repeat steps a and b for divided n and i = i + 2. The steps a and b are repeated till n becomes either 1 or a prime number.
2. **Smith Number**
   1. A smith number is a composite number whose sum of digits is equal to the sum of the digits in its prime factorization.
   2. ****
   3. The idea is first find all prime numbers below a limit using [Sieve of Sundaram](https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/) (This is especially useful when we want to check multiple numbers for Smith). Now for every input to be checked for Smith, we go through all prime factors of it and find sum of digits of every prime factor. We also find sum of digits in given number. Finally we compare two sums. If they are same, we return true.
   4. **Time Complexity:**O(n log n)   
      **Auxiliary Space:** O(n)
3. **Sphenic Number**
   1. A [Sphenic Number](https://en.wikipedia.org/wiki/Sphenic_number) is a positive integer **n** which is product of exactly three distinct primes. The first few sphenic numbers are 30, 42, 66, 70, 78, 102, 105, 110, 114, …   
      Given a number **n**, determine whether it is a Sphenic Number or not.
   2. ****
   3. ****
   4. Sphenic number can be checked by fact that every sphenic number will have exactly 8 divisor [SPHENIC NUMBER](https://en.wikipedia.org/wiki/Sphenic_number)   
      So first We will try to find if the number is having exactly 8 divisors if not then simply answer is no.If there are exactly 8 divisors then we will confirm whether the first 3 digits after 1 are prime or not.   
      Eg. 30 (sphenic number)   
      30=p\*q\*r(i.e p,q and r are three distinct prime no and their product are 30)   
      the set of divisor is (1,2,3,5,6,10,15,30).
   5. **Time Complexity:** O(?p log p)   
      **Auxiliary Space:** O(n)
4. **Hoax Number**
   1. A **Hoax Number** is defined as a composite number, whose sum of digits is equal to the sum of digits of its distinct prime factors. It may be noted here that, 1 is not considered a prime number, hence it is not included in the sum of digits of distinct prime factors.
   2. ****
   3. ****
5. **Kth prime factor of a given number**
   1. [**https://www.geeksforgeeks.org/problems/kth-prime-factor-of-a-number0132/1?itm\_source=geeksforgeeks&itm\_medium=article&itm\_campaign=bottom\_sticky\_on\_article**](https://www.geeksforgeeks.org/problems/kth-prime-factor-of-a-number0132/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=bottom_sticky_on_article)
   2. Given two numbers n and k, print k-th prime factor among all prime factors of n. For example, if the input number is 15 and k is 2, then output should be “5”. And if the k is 3, then output should be “-1” (there are less than k prime factors).
6. **Pollard’s Rho algo for prime factorization**
7. **Finding power of prime number p in n!**
8. **Find all distinct factors of a Natural number**
   1. If we look carefully, all the divisors are present in pairs. For example if n = 100, then the various pairs of divisors are: (1,100), (2,50), (4,25), (5,20), (10,10)  
      Using this fact we could speed up our program significantly.   
      We, however, have to be careful if there are two equal divisors as in the case of (10, 10). In such case, we’d print only one of them.
      1. The divisors of 100 are:
      2. 1 2 4 5 10 20 25 50 100
   2. Time Complexity: O(sqrt(n))   
      Auxiliary Space : O(1)
9. **Find number with n-divisors in a given range**