TOPICS WHICH YOU CAN'T SKIP | INTERVIEW PREPARATION | Part 1

leetcode.com/discuss/study-guide/3735417/topics-which-you-cant-skip-interview-preparation-part-1

The topic included in this article is very important. It can be used in many types of questions, so let's discuss how numbers are manipulated in questions and how we should think about them.

Let's start exploring number theory by understanding how to find factors of a number. **We can then gradually delve into various concepts and applicationsof number and number theory. how to deal with graph and how its mix with Dp and much much more...**

1.A naive approach is to iterates through all numbers from 1 to n and prints out any number that is a factor of n.
This function has a time complexity of O(n), since it has to iterate through all n numbers.

```
void finding_factor_of_number(int n) {
  for (int i = 1; i <= n; i++) {
    if (n % i == 0) {
      cout << i << " is a factor" << endl;
    }
  }
}
```

2.Lets try to find factor in sqrt(n) Time complexity
If a number n is not a prime, it can be factored into two factors a and b. n = a * b
Now a and b can't be both greater than the square root of n, since then the product a * b would be greater than sqrt(n) * sqrt(n) = n **So in any factorization of n, at least one of the factors must be less than or equal to the square root of n**

lets take example for 24 |

Factor	Product	Explanation
1	24	1 x 24 = 24, so 1 and 24 are factors
2	12	2 x 12 = 24, so 2 and 12 are factors
3	8	3 x 8 = 24, so 3 and 8 are factors
4	6	4 x 6 = 24, so 4 and 6 are factors

```
void find_factors(int number) {
  for (int factor = 1; factor * factor <= number; factor++) {
    if (number % factor == 0) {
      // Print the factor
      cout << factor << endl;

      // Check if factor and quotient are different (to avoid duplicates)
      if (number / factor != factor) {
        // Print the quotient (another factor)
        cout << number / factor << endl;
      }
    }
  }
}
```

3. So just using above approach we can check is number is prime or not in sqrt(n) Time complexity

If a number n is not a prime, it can be factored into two factors a and b . n = a * b
Now a and b can't be both greater than the square root of n, since then the product a * b would be greater than sqrt(n) * sqrt(n) = n. So in any factorization of n, at least one of the factors must be less than or equal to
the square root of n, and **if we can't find any factors less than or equal to the square root,**
n must be a prime.

This means that if we loop from 2 to sqrt(x) and don't find any factors, then x must be a prime number.

```
bool is_prime(int number) {
  // A prime number is a number that has only two factors: 1 and itself.

  // We can check if a number is prime by looping from 2 to the square root of the
number.
  // If we find any factor in this range, then the number is not prime.

  for (int factor = 2; factor * factor <= number; factor++) {
    if (number % factor == 0) {
      // The number is not prime because it has a factor greater than 1.
      return false;
    }
  }

  // If we reach the end of the loop without finding any factors, then the number
is prime.

  return true;
}
```

4-> Find all prime Divisor of number in root of n

The smallest factor of a number is always prime. This is because a prime number is a number that has only two factors: 1 and itself. **So, if we find the smallest factor of a**

number, we know that it must be prime.
We can use this property to find all the prime factors of a number.We can start by finding the smallest factor of the number. Once we have found the smallest factor, we can divide the number by the smallest factor to get a new number. We can then repeat this process until we reach a number that is 1. The prime factors of the original number will be the factors that we found along the way.

```
void Allprimefactor(int num)
{
  for(int primefac=2;primefac*primefac<=num;primefac++)
  {
        while(num%primefac==0)
        {
              cout << primefac << endl;
              num/=primefac;
        }
  }
  if(num>1) // this case will come number is prime
  cout << num<< endl;
}
```

Here's a table illustrating the steps to find the prime factors of the number 12 using the approach described:

Step	Number	Smallest Factor	New Number
1	12	2	6
2	6	2	3
3	3	3	1

5. There is another approach that is called as Sieve of Eratosthenes used to mark all prime

```
and used  for -> find all prime factor
              -> all factor of number
                      -> Mark lowest prime
                      -> Mark highest Prime
                      -> Printing all Prime factor
There is much much more things .. Lets discuss one by one
```

First say every number is prime then except 0 and 1, then we can say if 2 is prime all multiple of 2 is not prime then mark all multiple of 2 as false as its is not prime
eg->

Prime Numbers	Multiples
2	4, 6, 8, 10, 12, 14, ... mark false
3	6, 9, 12, 15, 18, ... mark false

Prime Numbers	Multiples
4	8 ,12 ,16 ... mark false
5	10, 15, 20, 25, 30, ... mark false

In this way we can find prime number easly

The following animation shows the marking for finding the primes within 121.

```
code :->
const int N = 1e6+ 10;
vector<bool> Prime(N, 1);
void Seive()
{
    Prime[0] = Prime[1] = false;
    for (int i = 2; i < N; i++)
    {
        if (Prime[i])
        for (int j = 2 * i; j < N; j += i)
        Prime[j] = 0;
    }
   // Time Complexity: O(n*log(log(n)))

}
```

As we have discussed :->The smallest factor of a number is always prime.so can we get all small factor of prime number using sieve and guess what answer is yes and also we can find highest prime multiple of any number

```
const int N = 1e6 + 10;
vector<bool> Prime(N, 1);
vector<int> lprime(N, 0), hprime(N, 0);

void Seive()
{
    Prime[0] = Prime[1] = false;
    for (int i = 2; i < N; i++)
    {
        if (Prime[i])
        {
            // Update smallest and highest prime factor as this is the first prime
            lprime[i] = hprime[i] = i;
            for (int j = 2 * i; j < N; j += i)
            {
                Prime[j] = 0; // Mark multiples as non-prime
                hprime[j] = i; // Update highest prime factor to the current prime

                // If smallest prime factor is not assigned yet, assign it
                if (lprime[j] == 0)
                {
                    lprime[j] = i;
                }
            }
        }
    }
}
```

Now we simple use these lowest prime and highest prime do more coolthings

1. Printing all prime factor of numbers ->Reason for this : All factors of a number are products of prime factors.

```
void Allprimefact(int num)
{
    map<int, int> primefac; // Here we have store prime factor and there
frequencies
    // we will use these frequencies to solve many questions
    while (num > 1)
    {
        int lowprime = lprime[num]; // you can use high prime also
        while (num % lowprime == 0)
        {
            primefac[lowprime]++;
            num /= lowprime;
        }
    }
    for (auto [num, count] : primefac)
        cout << num << " " << count << endl;
}
```

2. say If N=p^*aq^b*r^c where N is actually any number

```
a->Total Number of Factors for Number that is .(a+1) (b+1) (c+1)..
(where is a ,b,c is frequencie of prime)
b-> Product of factors of N = N^Total No. of Factors/2
c-> Sum of factors= (p^0 + p^1 + p^2 + ... + p^a) * (q^0 + q^1 + q^2 + ... + q^b)
* (r^0 + r^1 + r^2 + ... + r^c)
Note :  map<int, int> primefac:-> here we have find all primefact  and its freq
for a num
you can use this to implement above things
```

3.Another use of sieves:-> Find all factor of a number
Actually here we have coded like seives but actually its not sieves() You can name it anything method according to your convenience

```
code :
int const N=1e6+10;
vector<int>factor[N];

void allfactor()
{
  for(int i=2;i<N;i++)
  for (int j = i; j < N; j += i)
  factor[j].push_back(i);
  // you can also add 1 as factor
  Time comp: O(nlog(n))
}
```

4. Now lets discuss extended sieves :
The Sieve of Eratosthenes looks good, but consider the situation when n is large, the Simple Sieve faces the following issues.

An array of size Θ(n) may not fit in memory The simple Sieve is not cached friendly even for slightly bigger n.
The algorithm traverses the array without locality of reference can you write in more better way

Use a segmented sieve A segmented sieve is a variation of the Simple Sieve of Eratosthenes that divides the array into segments. This can help to reduce the amount of memory that is required.

The segmented sieve is useful when you need to find prime numbers within a given range, such as from 1 to 10^12 while considering additional constraints. One such constraint is when you want to find the number of primes or perform operations on primes within a subrange where the **difference between the left and right endpoints is less than or equal to 10^6.**

For example, if you have the range from 1 to 10^8 , the segmented sieve may not be applicable since the range is invalid. **The segmented sieve requires a valid range where the left and right endpoints have a difference less than or equal to 10^6**.

However, if you have a range such as 10^6 + 10005 to 10^6, the segmented sieve can be applied because the difference between the left and right endpoints is within the allowed constraint of 10^6.

**Note : If you are interested in algo of segmeneted sieve comment below i will also add that for now i
am just leaving here ;**

Note 2: Pollard Rho algorithm :-> is a probabilistic algorithm that can be used to factorize large numbers. It is a good choice for factorizing numbers that are of the order of 10^18 or larger.The Pollard Rho algorithm is a probabilistic algorithm, which means that it does not always work. However, it is very efficient and can be used to factorize very large numbers

Lets talk about gcd and lcm :
The GCD (Greatest Common Divisor) of two numbers is defined as the largest integers that divides both the numbers. For example, 2 is the GCD of 4 and 6.
From this concept, follows something called co-primes. Two numbers are said to be **co-primes** if their GCD is 1. For example, 3 and 5 are co-primes because their GCD is 1.

Coming to LCM (Least Common Multiple), it is defined as the smallest integer that is divisible by both the numbers. For example, 10 is the LCM of 2 and 5.
code

```
int calculateGCD(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return calculateGCD(b, a % b);
    }
}
```

we calculate the GCD of two numbers, we can calculate the LCM very easily

```
int calculateLCM(int a, int b) {
    int gcd = calculateGCD(a, b);
    int lcm = (a * b) / gcd;
    return lcm;
}
```

Examples on above discussed concepts

1. Count-primes

```
class Solution {
public:
    int countPrimes(int n)
    {
        vector<int>prime(n+10,true);
        prime[0]=prime[1]=false;
        int cnt=0;
        for(int i=2;i< n;i++)
        {
            if(prime[i])
            {
                cnt++;
                for(int j=2*i;j<n;j+=i)
                prime[j]=false;
            }
        }
        return cnt;
    }
};
```

2. Closest-prime-numbers-in-range

```
#define ll long long
const int N=1e6+20;
vector<int>prime(N,true);
class Solution {
public:
void seive()
{
       prime[0]=prime[1]=false;
       for(int i=2;i< N;i++)
       if(prime[i])
       for(int j=2*i;j< N;j+=i)
       prime[j]=false;
}
vector<int> closestPrimes(int left, int right)
{
        seive();
        vector<int>rangeprime;
        for(int i=left;i<=right;i++)
        if(prime[i])
        rangeprime.push_back(i);

        // Now simply iterate
        if(rangeprime.size()<2)
        return {-1,-1};
        int a=rangeprime[0],b=rangeprime[1],mn=b-a;
        for(int i=0 ;i<rangeprime.size()-1;i++)
        {
          int currmin=rangeprime[i+1]-rangeprime[i];
          if(currmin<mn)
          {
              a=rangeprime[i],b=rangeprime[i+1];
              mn=currmin;
          }
        }
        return {a,b};
}
};
```

3. Smallest-value-after-replacing-with-sum-of-prime-factors

```
class Solution {
public:
   int primesum(int n)
   {
        long sum=0;
        for(long long i=2;i*i<=n;i++)
        {
            while(n%i==0)
            {
                sum+=i;
                n/=i;
            }
        }
        if(n>1)
        return sum+n;
        return sum;
   }
    int smallestValue(int n)
    {
        // There is catch smallest is that value is actually prime
        // we can also use sieve to calculate all sum of prime at same time
        // will marking prime
       int ans=n;
       while(true)
       {
         int currN=primesum(n);
         if(currN==n)
         return n;
         n=currN;
       }
       return n;
    }
};
```

4. Distinct-prime-factors-of-product-of-array

```
Method 1
class Solution {
public:
    set<int>s ;
    void prime(int n)
    {
        for(int i=2;i*i<=n;i++)
        while(n%i==0)
        {
            s.insert(i);
            n/=i;
        }
        if(n>1)
        s.insert(n);
    }
    int distinctPrimeFactors(vector<int>& nums)
    {
        // simple approach is that we have 10^4 numbers and order of 10^3
        // so simply we  can find each prime and return ans using set or
        // unordered set
        for(auto num:nums)
        prime(num);
        return s.size();
    }
};

Method 2
class Solution {
public:
    vector<bool> seive(int n)
    {
        vector<bool>prime(n+10,true);
        prime[0]=prime[1]=false;
        for(int i=2;i<=n;i++)
        if(prime[i])
        for(int j=2*i;j<=n;j+=i)
        prime[j]=false;
        return prime;
    }
    int distinctPrimeFactors(vector<int>& nums)
    {
        vector<bool>mark=seive(1001);
        vector<int>prime;
        for(int i=0 ;i<=1000;i++)
        if(mark[i])prime.push_back(i);
        int cnt=0;
        for(auto check:prime)
        {
            for(auto num:nums)
            if(num%check==0)
            {
               cnt++;
               break;
            }
        }
        return cnt;
    }
};
```

5.Four-divisors

```
class Solution {
public:
    int cal(int n)
    {
        map<int,int>freq;
        for(int i=2;i*i<=n;i++)
        while(n%i==0)
        freq[i]++ ,n/=i;
        if(n>1)
        freq[n]++;
        int totaldiv=1,sum=1;
        for(auto [div,cnt]:freq)
        {
          totaldiv*=(cnt+1);
          if(cnt>4)return 0;
          sum*=((pow(div,cnt+1)-1)/(div-1));
        }
        if(totaldiv==4)
        return sum;
        return 0;

    }
    int sumFourDivisors(vector<int>& nums)
    {
        int ans=0;
        for(auto num:nums)
        ans+=cal(num);
        return ans;
    }
};
```

6 .Prime-arrangements

```
class Solution {
public:
   int mod=1e9+7;
    bool isprime(int n )
    {
        for(int i=2;i*i<=n;i++)
        if(n%i==0)return false;
        return true;
    }
    long long fact(int n)
    {
        long long fac=1;
        for(int i=1;i<=n;i++)
        fac=(fac%mod*i)%mod;
        return fac%mod;
    }
   int numPrimeArrangements(int n)
   {
       int cntprime=0;
       for(int i=2;i<=n;i++)
       {
           if(isprime(i))
           // cout << i << endl;
           cntprime+=isprime(i);
       }

      return fact(cntprime)*1LL*fact(n-cntprime)%mod;
   }
};
```

7 .largest-component-size-by-common-factor/

```
Disjoint Set Union + Sieve of Eratosthenes
We can see that if two numbers belong to the same group due to some common factor,
they should also have some common prime factor. This allows us to use slightly
different
approach where we find all prime factors of a number and union the elements based
on
the common prime factors amongst them.
class Solution {
public:
    const int N=1e5+7;
    vector<int>parent,Rank;
    int Find(int node)
    {
        if(parent[node]==-1)return node;
        return parent[node]=Find(parent[node]);
    }
    void Union(int x,int y)
    {
        int xPar=Find(x), yPar=Find(y);
        if(xPar==yPar)return ;
        if(Rank[xPar]> Rank[yPar])
        parent[yPar]=xPar;
        else  if(Rank[xPar]> Rank[yPar])
          parent[xPar]=yPar;
        else
        {
            parent[yPar]=xPar;
            Rank[xPar]++;
        }
    }
    int largestComponentSize(vector<int>& nums)
    {    parent.resize(N,-1);
         Rank.resize(N,0);
         int n = nums.size(),cnt=0;
         for (int i=0 ;i<n ;i++)
         for(int j=2; j*j<=nums[i];j++)
         if(nums[i]%j==0)
         Union(nums[i], j), Union(nums[i],nums[i]/j);
         unordered_map<int,int> map;
         for(int i=0 ;i<n ;i++)
         map[Find(nums[i])]++;
         for(auto val:map)cnt=max(cnt,val.second);
         return cnt;
    }
};
```

Questions For Practice:

You can click on each link to access the respective problem description on LeetCode.

Note: This article only covers the half part of number theory for LeetCode. If you are interested in learning more, I will post a second part that covers rest of topic of number theory. Please let me know in the comments if you would like me to do this.

My Other post:->
All-types-of-patterns-for-bits-manipulations-and-how-to-use-it
Binary-search-a-comprehensive-guide

If you found it helpful, please upvote. If you have any questions or comments, please feel free to leave them below. I'd love to hear your thoughts.

numbertheoryprimenumberseive-of-eratosthenesinterview guide

Read More

You can check out my **YouTube channel**., where I upload videos covering all the articles I've written,
along with new topics. **Cllick Here To Subscribe**.