Basic Maths for DSA

▼ Prime Number

204. Count Primes

Hint

Medium 7.2K 1.3K

Companies

Given an integer `n`, return *the number of prime numbers that are strictly less than* `n`.

Example 1:

```
Input: n = 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.
```

Example 2:

```
Input: n = 0
Output: 0
```

Example 3:

```
Input: n = 1
Output: 0
```

▼ Native Approach

▼ Logic

- If any number divides between 1 to N divides N fully i.e `N % i == 0`, So N is not a prime number

▼ Code

💡 Time Complexity → O(n^2)

```
class Solution {
public:
    bool isPrime(int N){
        if(N <= 1) return false;

        for(int i=2; i<N; i++){
            if(N % i == 0){
                // not prime
                return false;
            }
        }
        return true;
    }
    int countPrimes(int n) {
        int count = 0;
        for(int i=2; i<n; i++){
            if(isPrime(i)){
                count++;
            }
        }
        return count;
    }
};
```

Sqrt Approach

Logic

- Let N is non Prime i.e N = a x , then there exist at least one of the factor must be smaller then $Sqrt(n)$
- If we can't find any factor then it is not a prime number

Code

💡 Time Complexity → O$(n * sqrt(n))$

```
class Solution {
public:
    bool isPrime(int N){
        int sqrtN = sqrt(N);
        if(N <= 1) return false;

        for(int i=2; i<=sqrtN; i++){
            if(N % i == 0){
                // not prime
                return false;
            }
        }
        return true;

    }
    int countPrimes(int n) {
        int count = 0;
        for(int i=2; i<n; i++){
            if(isPrime(i)){
                count++;
            }
        }
        return count;
    }
};
```

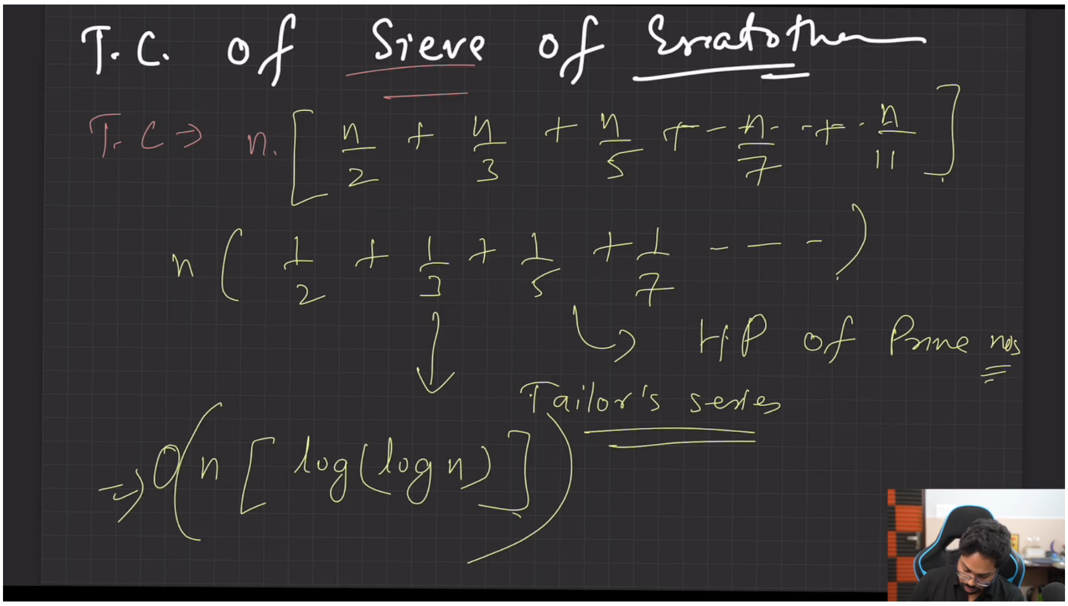
▼ Sieve of Eratostheness

▼ Logic

1. Create an array from 2→ n-1 and mark all of them as prime
2. Starting from 2 till n-1, check who is prime and mark all the number comes in the table of these as non prime
3. Rest elements who are marked as prime is the ans

▼ Time Complexity

💡 Time Complexity → O$(n * [log(logn)])$



▼ Code

```
class Solution {
public:
    int countPrimes(int n) {
       if(n == 0) return false;

        vector<bool> prime(n,true);

        //0 and 1 is already neither prime nor non pri
        prime[0] = prime[1] = false;

        //check from 2 to n-1
        int count = 0;
        for(int i=2; i<n; i++){
            if(prime[i])
                count ++;

            // multiple of prime numbers
            int j = 2*i;
            while(j<n){
                prime[j] = false;
```

```
                j+=i;
            }
        }
        return count;
    }
};
```

▼ Optimization 1 : Inner loop

Start marking from `int j=i*i` rather then `int j = 2*i` as others have been already marked by 2 to (i-1)

```
class Solution {
public:
    int countPrimes(int n) {
       if(n == 0) return false;

        vector<bool> prime(n,true);

        //0 and 1 is already neither prime nor non
        prime[0] = prime[1] = false;

        //check from 2 to n-1
        int count = 0;
        for(int i=2; i<n; i++){
            if(prime[i])
                count ++;

            // multiple of prime numbers
            int j=i*i;
            while(j<n){
                prime[j] = false;
                j+=i;
            }
        }
        return count;
    }
};
```

▼ Optimization 2 : Outer loop

We can start outer loop from $i = 2$ to $i = sqrt(n)$ as usse bada number inner loop me chlega hi nhi. Exmaple `int j = 6*6 ⇒ 36 > 25(n=25)`

```
class Solution {
public:
    int countPrimes(int n) {
       if(n == 0) return false;

        vector<bool> prime(n,true);

        //0 and 1 is already neither prime nor non
        prime[0] = prime[1] = false;

        //check from 2 to n-1
        int count = 0;
        for(int i=2; i*i<n; i++){
            if(prime[i])
                count ++;

            // multiple of prime numbers
            int j = i*i;
            while(j<n){
                prime[j] = false;
                j+=i;
            }
        }
        return count;
    }
};
```

▼ Segment sieve

💡 1. In any function array of `int,double,char` has max size of 10^6 and `bool` has max size 10^7.
2. In global array `int,double,char` has max size of 10^7 and `bool` has max size 10^8.

▼ Logic

1. Generate all the base primes responsible to mark segment sieve using normal sieve till sqrt(R)

2. Find the first index to start marking using the formula :

   $FM = (low/prime) * prime$ and `if(FM < low) => FM += prime`

▼ Code

```
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;

vector<bool> Sieve(int n)
{
    vector<bool> sieve(n + 1, true);
    sieve[0] = sieve[1] = false;

    // check from 2 to n-1
    for (int i = 2; i * i < n; i++)
    {
        if (sieve[i] == true)
        {
            int j = i * i;
            while (j <= n)
            {
                sieve[j] = false;
                j += i;
            }
        }
    }
```

```
    return sieve;
}

vector<bool> segSieve(int L,int R){
    vector<bool> sieve = Sieve(sqrt(R));
    vector<int> basePrimes;
    for(int i=0; i<sieve.size(); i++){
        if(sieve[i]){
            basePrimes.push_back(i);
        }
    }

    vector<bool> segmentSieve(R-L+1,true);
    if(L == 0 || L == 1){
        segmentSieve[L] = false;
    }

    for(auto prime : basePrimes){
        int firstMul = (L/prime) * prime;
        if(firstMul < L){
            firstMul += prime;
        }
        int j = max(firstMul,prime*prime);
        while(j <= R){
            // index 0 -> 110
            // index 1 -> 111
            segmentSieve[j-L] = false;
            j += prime;
        }
    }
    return segmentSieve;
}
int main()
{
    int L =110;
    int R =130;
    vector<bool> ss = segSieve(L,R);
    for(int i =0; i<ss.size(); i++){
```

```
            if(ss[i]){
                cout << i+ L << " ";
            }
        }

        return 0;
    }
```

▼ GCD(Greatest common factor) / HCF(Highest Common factor)

▼ Euclid’s Algorithm

▼ Logic

- It is defined as :

$$gcd(a,b) = gcd(a-b,b); a > b$$

$$gcd(a,b) = gcd(b-a,a); a < b$$

> 💡 Apply the above formula till one of parameter becomes zero

▼ Example

gcd(72,24) ⇒ gcd(48,24) ⇒ gcd(24,24) ⇒ gcd(0,24), so non-zero parameter is the answer

- Also defined as :
  gcd(a,b) = gcd(a % b , b) ; a > b
  gcd(a,b) = gcd(b % a , a) ; a < b

▼ Code

```
class Solution
{
    public:
    int gcd(int A, int B)
    {
        if(A == 0) return B;
```

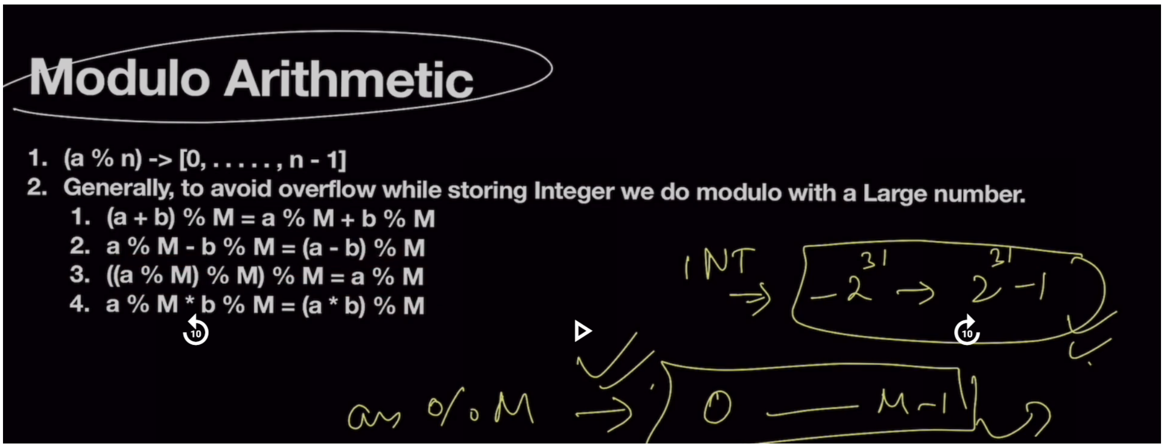
```
        if(B == 0) return A;

        while(A>0 && B>0){
            if(A > B){
                A = A-B;
            }
            else{
                B = B-A;
            }
        }
        return A == 0 ? B : A;
    }
};
```

▼ LCM

▼ Logic

- It is defined as : $LCM(a, b) = (a * b)/gcd(a, b)$

▼ Modulo Arithmetic



▼ Fast Exponentiation

▼ Slow Exponentiation

▼ Logic

- a^b ⇒ a * a * a * a …..b times
- loop → [0 → b-1] → ans = ans *a

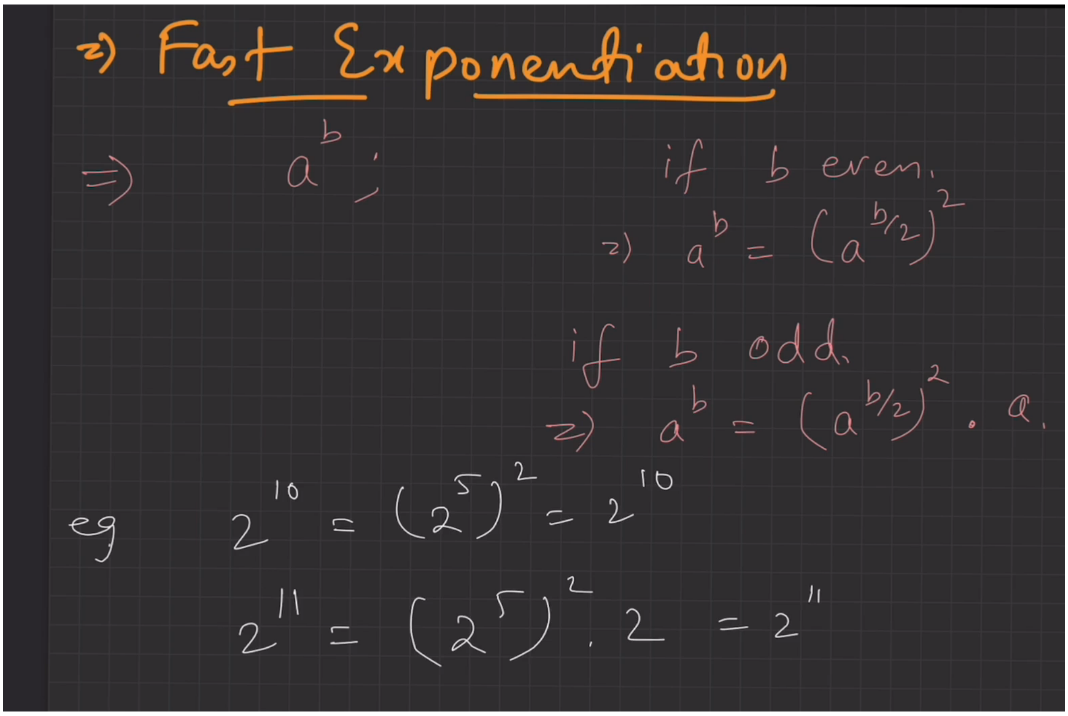
▼ Code

> 💡 Time complexity : O(b)

```
int slowExponentiation(int a,int b){
    int ans =1;
    for(int i=0; i<b; i++){
        ans += a;
    }
}
```

▼ Fast Exponentiation

▼ Logic



▼ Code

> 💡 Time complexity : O$(logb)$

```
int fastExponentiation(int a,int b){
    int ans =1;
    while(b>0){
        // checking if b is odd
        if(b & 1){
            ans = ans * a;
        }
        a = a * a;
        // b ko haar baar half krnna h
        b >>= 1;
    }
    return ans;
}
```