প্রাইমালিটি টেস্টিং

প্রাইম নাম্বার : ১ থেকে বড় কোন স্বাভাবিক সংখ্যা N, শুধু মাত্র ১ আর N দ্বারা নিঃশেষে বিভাজ্য হলে কিন্তু অন্য কোন সংখ্যা দ্বারা নিঃশেষে বিভাজ্য না হলে সেই N কে প্রাইম নাম্বার বা মৌলিক সংখ্যা বলা হয় ।

সোজা কথায় বলতে গেলে, কোন সংখ্যা N কে ১ আর N ছাড়া অন্য কোন সংখ্যা দ্বারা ভাগ দেওয়া না গেলে সেই সংখ্যাটি প্রাইম অথবা ১ আর N ছাড়া, N এর অন্য কোন মাল্টিপল না থাকলে N হবে প্রাইম নাম্বার। দুটো কথাই একি অর্থ বহন করে।

যেমন : ২ , ৩ , ৫ , ৭ , ১১ .....

কোন সংখ্যা N, অন্য কোন সংখ্যা M দ্বারা ভাগ করে ভাগশেষ শুন্য পাওয়া গেলে

বলা হয় N, M দ্বারা নিঃশেষে বিভাজ্য । অর্থাৎ N % M == ০ হলে বলা যায়, M, N কে ভাগ দিতে পারে, অথবা N এর একটি মাল্টিপল হল M ।

তাহলে কোন সংখ্যা প্রাইম কিনা যাচাই করতে হলে , আমাদের ২ থেকে N – ১ পর্যন্ত সবগুলো সংখ্যা দ্বারা N কে মড করে দেখতে হবে, যদি যে কোন একটা সংখ্যা দ্বারা মড শুন্য হয় তাহলে বুঝতে হবে N প্রাইম সংখ্যা নয় । আর যদি ২ থেকে N – ১ পর্যন্ত কোন সংখ্যাই না পারে N কে ভাগ করতে তাহলে বলা যেতে পারে N একটি প্রাইম নাম্বার । কমপ্লেক্সিটি O(N) ।

এই ফাংশনটিতে একটি নাম্বার দিলে সেই নাম্বারটি প্রাইম কি প্রাইম নয় সেটি বলে দেয় ।

```
1. // If this Function return 1 , it means N is prime
2. // Else if return 0 , then it means N is not prime
3. bool IsPrime(int n) {
4.     for(int i = 2 ; i <= n - 1 ; i++) {
5.         if(n % i == 0)
6.             return 0 ;
7.     }
8.     return 1 ;
9. }
```

একটু ভাল করে খেয়াল করলে আমরা দেখব , আমাদের ২ থেকে N – ১ পর্যন্ত ভাগশেষ চেক না করলেও হচ্ছে । আমরা যদি N / ২ পর্যন্ত চেক করি তাতেই হয়ে

যাচ্ছে । কারন কোন সংখ্যার অর্ধেকের থেকে বড় সংখ্যা সেই সংখ্যাকে কখনই ভাগ (নিঃশেষে বিভাজ্য) দিতে পারে না । তাই আমরা এখানে আমাদের লুপ অর্ধেকে নিয়ে আসতে পারি । যেমন : ১০ কে ভাগ করতে ৫ পর্যন্ত চেক করলেই হচ্ছে ।

```
1. bool IsPrime(int n) {
2.     for(int i = 2 ; i <= n / 2 ; i++) {
3.         if(n % i == 0)
4.             return 0 ;
5.     }
6.     return 1 ;
7. }
```

এখন আমরা একটু গভীরে চিন্তা করলে দেখব এর কমপ্লেক্সিটি আরো কমিয়ে নিয়ে আসা যায় । যেমন :

১০ = (১ , ১০) , (২ , ৫) ,

১২ = (১ , ১২) , (২ , ৬) , (৩ , ৪) ,

১৬ = (১ , ১৬) , (২ , ৮) ,(৪ , ৪)

এখানে দেখা যাচ্ছে কোন সংখ্যা i দ্বারা N কে ভাগ দেওয়া গেলে, N / i দ্বারাও N কে ভাগ দেওয়া যাচ্ছে , যেমন ১০ কে ২ ভাগ দিতে পারলে , ১০ / ২ = ৫ , এটিও ১০ কে ভাগ দিতে পারে । তাহলে আমরা একটা সিদ্ধান্তে আসতে পারি , উপরের পেয়ার গুলো থেকে প্রথম সংখ্যা i গুলো দ্বারা ভাগ করে চেক করলেই হচ্ছে । এখন তাহলে কথা হচ্ছে , এই i সর্বোচ্চ কত হতে পারে ! ১৬ এর দিকে যদি তাকাই , ৪ * ৪ = ১৬ , আর যদি ৫ * ৫ দিয়ে চেষ্টা করি তাহলে সেটা ১৬ এর চেয়ে বড় হয়ে যাচ্ছে । N > sqrt(N) * sqrt(N) এটি কখনই কোন সংখ্যার ক্ষেত্রে সম্ভব নয় , তাহলে বলা যায় N <= sqrt(N) * sqrt(N) । আর N / sqrt(N) = sqrt(N) , সুতরাং আমাদের sqrt(N) পর্যন্ত সর্বোচ্চ ভাগ দিয়ে চেক করলেই হচ্ছে । কমপ্লেক্সিটি O(sqrt(N)) ।

```
bool IsPrime(int n) {
    int sq = sqrt(n) ;
    for(int i = 2 ; i <= sq ; i++) {
        if(n % i == 0)
            return 0 ;
    }
    return 1 ;
}
```

আমরা চাইলে আরো ভাল করে এভাবে লিখতে পারি। এটা উপরের যে কোনটির চেয়ে ভালো কাজ করে। প্রাইমালিটি টেস্ট করতে এটাই ব্যবহার করা উত্তম। এটা নিজে চিন্তা করলেই বুঝতে পারা যাবে আশা করি।

```
bool IsPrime(int n) {
    if(n == 0 || n == 1) return 0 ;
    if(n == 2) return 1 ;
    if(n % 2 == 0) return 0 ;
    int sq = sqrt(n) ;
    for(int i = 3 ; i <= sq ; i += 2) {
        if(n % i == 0)
            return 0 ;
    }
    return 1 ;
}
```

প্র্যাকটিস প্রবলেম :

Uva : 10924 - Prime Words , 543 - Goldbach's Conjecture , 686 - Goldbach's Conjecture (II) , 10168 - Summation of Four Primes , 10948 - The primary problem

ডিভিজরস

ডিভিজর বা ভাজক এর মানে কি ? মনে করি কোন সংখ্যা N কে M দ্বারা ভাগ করলে ভাগশেষ শুন্য (অথবা নিঃশেষে বিভাজ্য হলে অথবা N%M==0) হলে আমরা M কে N এর ডিভিজর বা ভাজক বলি ।

যেমন : ১০ কে ২ দ্বারা ভাগ দিলে ভাগশেষ শুন্য অথবা 10 % 2 == 0 হয় , তাহলে বলা যায় ২ , ১০ এর একটি ডিভিজর, কিন্তু আবার ১০ % ৩ != ০ সুতরাং ৩ , ১০ এর ডিভিজর নয় ।

এখন যদি আমরা ১০ , ১২ আর ৯ এর সব গুলো ডিভিজর দেখতে চাই তাহলে ব্যপারটা অনেকটা এরকম হবে ,

১০ = ১ , ২ , ৫ , ১০ ।

১২ = ১ , ২ , ৩ , ৪ , ৬ , ১২ ।

৯ = ১ , ৩ , ৯ ।

এখন যদি কোন সংখ্যা N এর সবগুলো ডিভিজর দেখতে চাওয়া হয় তাহলে আমরা সেটা কিভাবে করতে পারি ? একটু চিন্তা করলে দেখব N কে যারা যারা ভাগ করলে ভাগশেষ ০ হয় অথবা মড করলে ০ হয় তারা সবাই N এর ডিভিজর ।

তাহলে আমরা N কে ১ থেকে কত পর্যন্ত মড করে চেক করতে পারি ? একটু খেয়াল করলে দেখব N এর চেয়ে বড় কোন সংখ্যাই N কে ভাগ করতে পারে না , অর্থাৎ N এর ডিভিজর N এর চেয়ে বড় হতে পারে না ।

তাহলে আমরা N এর সবগুলো ডিভিজর দেখতে চাইলে i = ১ থেকে N পর্যন্ত সবগুলো সংখ্যা i দিয়ে N কে মড করে দেখতে পারি , যদি N % i = ০ হয় তাহলে

বলব i , N এর একটি ডিভিজর । এভাবে সবগুলো i হবে N এর সব ডিভিজরস । আমরা সব গুলো ডিভিজরস কে একটা অ্যারেতে রাখব এরপর তাদের দেখব ।

কমপ্লেক্সিটি O(N)

নিচে এর ইমপ্লিমেন্টেশনটা দেখা যেতে পারে :

```
1.  int DivList[100] ;      // It's contains the divisors of N
2.  int sz ;                // This contains the number of divisor of N
3.  void Divisors(int n) {
4.      sz = 0 ;
5.      for(int i = 1 ; i <= n ; i++) {
6.          if(n % i == 0) {
7.              DivList[sz++] = i ;
8.          }
9.      }
10. }
11. int main() {
12.     printf("Enter the Number : ") ;
13.     int n ;
14.     scanf("%d" , &n) ;
15.     Divisors(n) ;
16.     printf("%d = ",n) ;
17.     for(int i = 0 ; i < sz ; i++)
18.         printf("%d ",DivList[i]) ;
19.     printf("\n") ;
20.     return 0 ;
21. }
```

আমরা যদি একটু ভাল করে লক্ষ্য করি , N এর সব ডিভিজরস গুলোর দিকে , তাহলে খুব সুন্দর একটা প্যাটার্ন দেখা যায় । আমরা চাইলে ডিভিজরস গুলোকে এভাবে সাজিয়ে লিখতে পারি :

১০ = ১ , ১০ , ২ , ৫ অথবা (১ , ১০) , (২ , ৫)

১২ = ১ , ১২ , ২ , ৬ , ৩ , ৪ অথবা (১ , ১২) , (২ , ৬) , (৩ , ৪)

১৫ = ১ , ১৫ , ৩ , ৫ অথবা (১ , ১৫) , (৩ , ৫)

৯ = ১ , ৯ , ৩ , ৩ অথবা (১ , ৯) , (৩ , ৩)

১৬ = ১ , ১৬ , ২ , ৮ , ৪ , ৪ অথবা (১ , ১৬) , (২ , ৮) , (৪ , ৪)

উপরে দেখতে পাচ্ছি একটু সাজিয়ে লিখলে ডিভিজরস সমূহ জোড়ায় জোড়ায় থাকে , ৪ , ৯ , ১৬ ,২৫ , ... বাদে । শুধু এই জোড়ার দুইজনই একই , কিন্তু

ডিভিজর তো একই নাম্বার আমরা দুইবার নিতে পারি না । এদের কে নিয়ে একটু পরে চিন্তা করি ।

১০ এর একটা ডিভিজর ১ , আবার ১০ / ১ = ১০ , এটিও ১০ এর একটা ডিভিজর ।

১০ এর একটা ডিভিজর ২ , আবার ১০/২ = ৫ , এটিও ১০ এর একটা ডিভিজর ।

১২ এর একটা ডিভিজর ১ , আবার ১২/১ = ১২ , এটিও ১২ এর একটা ডিভিজর ।

১২ এর একটা ডিভিজর ২ , আবার ১২/২ = ৬ , এটিও ১২ এর একটা ডিভিজর ।

১২ এর একটা ডিভিজর ৩ , আবার ১২/৩ = ৪ , এটিও ১২ এর একটা ডিভিজর ।

এখানে দেখা যাচ্ছে কোন সংখ্যা, i দ্বারা N কে ভাগ দেওয়া গেলে, N / i দ্বারাও N কে ভাগ দেওয়া যাচ্ছে , যেমন ১০ কে ২ ভাগ দিতে পারলে , ১০ / ২ = ৫ , এটিও ১০ কে ভাগ দিতে পারে । তাহলে আমরা একটা সিদ্ধান্তে আসতে পারি , উপরের পেয়ার গুলো থেকে প্রথম সংখ্যা i গুলো দ্বারা ভাগ করে চেক করলেই হচ্ছে । এখন তাহলে কথা হচ্ছে , এই i সর্বোচ্চ কত হতে পারে ! ১৬ এর দিকে যদি তাকাই , ৪ * ৪ = ১৬ , আর যদি ৫ * ৫ দিয়ে চেষ্টা করি তাহলে সেটা ১৬ এর চেয়ে বড় হয়ে যাচ্ছে । N > sqrt(N) * sqrt(N) এটি কখনই কোন সংখ্যার ক্ষেত্রে সম্ভব নয় , তাহলে বলা যায় N <= sqrt(N) * sqrt(N) । আর N / sqrt(N) = sqrt(N) , সুতরাং আমাদের sqrt(N) পর্যন্ত সর্বোচ্চ ভাগ দিয়ে চেক করলেই হচ্ছে ।

এখন খেয়াল করলে দেখব , ৪ , ৯ , ১৬ , ২৫ , ৩৬ , … এদের ক্ষেত্রে N / sqrt(N) == sqrt(N) , এরা হচ্ছে পারফেক্ট স্কয়ার নাম্বার । তাহলে আমরা যদি দেখি কোন নাম্বার পারফেক্ট স্কয়ার ,তাহলে আমরা সেখানে N / sqrt(N) == sqrt(N) ই পাব । তাহলে সেক্ষেত্রে আমরা একটা sqrt(N) নিব ।

তাহলে আমাদের এক ইটারেশনে ১ থেকে sqrt(N) পর্যন্ত চেক করলেই আমরা সব ডিভিজরস পেয়ে যাচ্ছি । এখত্রে কমপ্লেক্সিটি আমরা O(sqrt(N)) তে কমিয়ে এনেছি । ডিভিজরস গুলো এখানে পেয়ার অয়াইজ থাকে , আমরা যদি সরটেড অর্ডারে দেখতে চাই তাহলে লিস্ট টাকে সর্ট করে নিলেও হয় ।

ইমপ্লিমেন্টেশন :

```
int DivList[100] ;      // It's contains the divisors of N
int sz ;                // This contains the number of divisor of N
void Divisors(int n) {
    sz = 0 ;
    int sq = sqrt(n) ;
    for(int i = 1 ; i <= sq ; i++) {
        if(n % i == 0) {
            DivList[sz++] = i ;
            DivList[sz++] = n / i ;
        }
    }
    if(sq * sq == n) {     // if perfect square then we are taking this sq twice
        sz-- ;
    }
    sort(DivList , DivList + sz) ;   // if want to see the sorted List of Divisors
}
```

প্র্যাকটিস প্রবলেম :

Uva : 294 – Divisors

Codeforces : B. Duff in Love

প্রাইম জেনারেটিং (সিভ অব ইরাটোস্থিনিস)

প্রাইম জেনারেট বলতে বোঝায় , প্রাইম সংখ্যার একটা লিস্ট , যেই লিস্টের নাম্বার দিয়ে বিভিন্ন প্রবলেম সল্ভ করতে হতে পারে ।

প্রাইম লিস্ট : ২ , ৩ , ৫ , ৭ , ১১ , ...

আমাদের ১টা নাম্বার প্রাইম কি প্রাইম না এটি চেক করতে সময় লাগে O(sqrt(N)) , তাহলে আমাদের যদি N টা প্রাইম নাম্বারের একটা লিস্ট দরকার হয় তাহলে আমাদের প্রত্যেকটার জন্য sqrt(N) টাইম করে লাগবে । সুতরাং কমপ্লেক্সিটি দাঁড়ায় O(N * sqrt(N)) যা অনেক বেশি । নিচে বেসিক ইমপ্লিমেন্টেশন টা দেখা যাক :

```cpp
// Complexity: O(N * sqrt(N))
#define mxN 100
int prime[mxN + 10] ;       // This is going to store all prime numbers upto mxN
int sz = 0 ;                // This is the size of prime list..
// Primality Testing Function..
bool IsPrime(int n) {
    if(n == 0 || n == 1) return 0 ;
    if(n == 2) return 1 ;
    if(n % 2 == 0) return 0 ;
    int sq = sqrt(n) ;
    for(int i = 3 ; i <= sq ; i += 2) {
        if(n % i == 0)
            return 0 ;
    }
    return 1 ;
}
```

```
// This is going to generate all the primes Upto mxN
void GeneratePrimes() {
    for(int i = 1 ; i <= mxN ; i++) {
        if(IsPrime(i) == 1) {
            prime[sz++] = i ;
        }
    }
}
int main() {
    GeneratePrimes() ;
    printf("Total Number of Primes Upto %d is %d\n" , mxN , sz) ;
    printf("Here's them : \n") ;
    for(int i = 0 ; i < sz ; i++) {
        printf("%d\n",prime[i]) ;
    }
    return 0 ;
}
```

এটা খুব বেশি কার্যকর নয় , কারন আমাদের বড় নাম্বারের ক্ষেত্রে এটা অনেক বেশি টাইম নেবে । আমরা চাইলে এর কমপ্লেক্সিটি O(NlogN) এ নিয়ে আসতে পারি। আর এর উপায় টা হচ্ছে , সিভ অব ইরাটোস্থিনিস । এটা প্রাইম নাম্বার জেনারেট করার সবচেয়ে প্রাচিন এলগরিদম ।

সিভ অব ইরাটোস্থিনিস : ধরে নেই আমার ১ - ২০ টা নাম্বারের জন্য প্রাইম লিস্ট লাগবে । এই এলগরিদম অনুসারে , প্রথমেই সব সংখ্যাকে প্রাইম হিসেবে বিবেচনা করা হয় ।

আমরা ১ – ২০ এর জন্য প্রাইম বের করবো । তাহলে আমারা একটা অ্যারে বিবেচনা করি ২০+১ সাইজের , যাতে প্রাথমিক ভাবে সব ইন্ডেক্সে ০ রাখা হয়েছে । যদি কোন সময় আমরা যদি দেখি অ্যারের কোন ith ইন্ডেক্সে ১ আছে তাহলে আমরা ধরে নিব ওই i ইন্ডেক্স এর নাম্বার টা নন প্রাইম আর ০ হলে মনে করব প্রাইম । যেমন : isp[2] = 0 , isp[3] = 0 , isp[4] = 1 , isp[6] = 1 . মানে ২ , ৩ প্রাইম , কিন্তু ৪ , ৬ নন প্রাইম ।

তাহলে ইনিশিয়ালি আমাদের isp এর সব ইন্ডেক্সে ০ আছে । তার মানে আমরা ধরে নিচ্ছি সবাই প্রাইম ।

আমরা জানি , ০ এবং ১ নন প্রাইম । তাই প্রথমেই isp[০] = 1 , isp[1] = 1

করে দেই। ০ , ১ নাম্বার আমরা ম্যানুয়ালি চিহ্নিত করেছি নন প্রাইম হিসেবে , এরপরের সবার জন্য নিচের প্রসিডিউর ফলো করব।

এখন নেক্সট আমাদের হাতে ২ আছে, আর এখন isp[2] = 0 অর্থাৎ , ২ একটা প্রাইম। আমরা এখন ২ এর যতো মাল্টিপল আছে সবাইকে নন প্রাইম হিসেবে চিহ্নিত করে দিব অর্থাৎ ২ এর মাল্টিপলগুলোর ইন্ডেক্সে গিয়ে ১ রেখে আসব। তাহলে অ্যারে যেটা দাঁড়াবে সেটা ২ এর মাল্টিপলের ইন্ডেক্সে সব ১ হয়ে যাবে , যেমন : isp[4] = 1 , isp[6] = 1 , isp[8] = 1 , isp[10] = 1 , isp[12] = 1 , isp[14] = 1 , isp[16] = 1 , isp[18] = 1 , isp[20] = 1 .

এরপর আমাদের হাতে আছে ৩ এবং isp[3] = 0 , অর্থাৎ ৩ একটি প্রাইম। তাহলে ৩ এর যতো মাল্টিপল আছে সবার ইন্ডেক্সে ১ বসিয়ে আসতে হবে।

isp[6] = 1 , isp[9] = 1 , isp[12] = 1 , isp[15] = 1 , isp[18] = 1

এরপর আসে ৪ কিন্তু isp[4] = 1 , অর্থাৎ ৪ নন প্রাইম। ৪ কে নন প্রাইম চিহ্নিত করেছে ২ , তাহলে ৪ এর সব মাল্টিপল ২ আগেই কেটে দিয়েছে। তাই আমরা ৪ নিয়ে আগাব না , এভাবে আমরা যদি হাতে প্রাইম পাই , তাহলে সেই প্রাইমের সব মাল্টিপলের ইন্ডেক্সে ১ বসিয়ে দিয়ে আসব কিন্তু হাতে নন প্রাইম থাকলে কিছু করব না , পরের নাম্বারটা দেখব।

পরের হাতে আছে ৫ , এবং isp[5] = 0 অর্থাৎ ৫ প্রাইম। তাহলে এবার আমরা ৫ এর সব মাল্টিপলের ইন্ডেক্সেও ১ করে আসব।

এভাবে আমরা সেম প্রসিডিউর ১ থেকে ২০ এর জন্য করব। তাহলে সব প্রাইম এখানে isp অ্যারেতে চিহ্নিত হয়ে যাবে। এরপর আমরা ১ থেকে ২০ এর জন্য একটা লুপ চালাতে পারি আর প্রাইম কি প্রাইম না সেটা ইন্ডেক্স দেখে চেক দিয়ে N পর্যন্ত যে কয়টি প্রাইম আছে তার একটি লিস্ট প্রস্তুত করতে পারি।

একটু খাতায় ব্যপারটা সিমুলেট করলে একবারেই বুঝে ওঠা যাবে আশা করি।

এটা এভাবে O(NlogN) কেন? এর সুন্দর ও সহজ একটা প্রুফ আছে। আমরা একটা নাম্বার i প্রাইম হলে তার সব মাল্টিপলের ইন্ডেক্সে ১ বসাচ্ছি। এখন তাহলে কতবার লুপ ঘুরছে ? আমরা i এর মাল্টিপলের ইন্ডেক্সে যাচ্ছি প্রতিবার। এখন আমাদের তাহলে এটা বের করতে পারলেই হচ্ছে N পর্যন্ত i এর কয়টি মাল্টিপল আছে ! যতটা মাল্টিপল আছে ঠিক ততবার আমদের লুপটি ঘুরছে। আর এখানে প্রতিটা লুপ i এর ওপর ডিপেন্ড করছে। আমরা সহজেই দেখতে পারি একটা

নাম্বার i এর N পর্যন্ত ঠিক N / i টি মাল্টিপল আছে। যেমন : ১০ পর্যন্ত ২ এর মাল্টিপল আছে ১০ / ২ = ৫ টি , ২০ পর্যন্ত ৩ এর মাল্টিপল আছে ২০ / ৩ = ৬ টি । এখন , যদি i প্রাইম হয় তাহলে একটা লুপ N / i বার ঘুরবে। তাহলে যদি দেখি ব্যাপারটা এমন :

N / 2 + N / 3 + N / 5 + N / 7 + ... + N / (Pn – 1) + ১

এখন যদি আমরা এখান থেকে N কমন নেই তাহলে দাঁড়ায় :

N * (1 / 2 + 1 / 3 + 1 / 5 + 1 / 7 + ... + 1 / (Pn – 1) +1)

বা , N * (1 + 1 / 2 + 1 / 3 + 1 / 5 + 1 / 7 + ... + 1 / N)

1 + 1 / 2 + 1 / 3 + 1 / 5 + 1 / 7 + ... + 1 / N এই সিরিজটি 1 + 1 / 2 + 1 + 3 + 1 / 4 + ... + 1 / N এর সমতুল। এটা একটা পরিচিত সিরিজ ,

ইন্টারে আমরা দেখেছি। এটাকে হারমনিক সাম সিরিজ বলে। এবং এটা ln(N) এর এক্সপানশন ফর্ম। আর লন হচ্ছে লগারিদমিক। সুতরাং N * ln(N) ইজ সিমিলারটু

N * logN ।

```
1.  // Complexity: O(N * log(N))
2.  #define mxN 100000
3.  bool isp[mxN + 10] ;       // This is the sign array , if isp[i] == 0 then i is prime else non prime
4.  int prime[mxN + 10] ;      // This is going to store all prime numbers upto mxN
5.  int sz = 0 ;               // This is the size of prime list..
6.  void Sieve() {
7.      for(int i = 0 ; i <= mxN ; i++) isp[i] = 0 ; // initially all are 0
8.      isp[0] = isp[1] = 1 ;
9.      for(int i = 2 ; i <= mxN ; i++) {
10.         if(isp[i] == 0) {             // if is is prime , making all its multiple non prime
11.             for(int j = i + i ; j <= mxN ; j += i)
12.                 isp[j] = 1 ;
13.         }
14.     }
15. }
```

```cpp
// This is going to generate all the primes Upto mxN
void GeneratePrimes() {
    Sieve() ;
    for(int i = 1 ; i <= mxN ; i++) {
        if(isp[i] == 0) {
            prime[sz++] = i ;
        }
    }
}
int main() {
    GeneratePrimes() ;
    printf("Total Number of Primes Upto %d is %d\n" , mxN , sz) ;
    printf("Here's them : \n") ;
    for(int i = 0 ; i < sz ; i++) {
        printf("%d\n",prime[i]) ;
    }
    return 0 ;
}
```

প্রাক্টিস প্রবলেম :

UVA : 406 - Prime Cuts, 10140 - Prime Distance, 10394 - Twin Primes

সিভ অফ ডিভিজরস

সিভ অফ ডিভিজরস আসলে অনেক কঠিন শোনালেও এর কোড ২ লাইনের । সিভ অফ ডিভিজরস জিনিসটা আসলে কি?

একটা প্রবলেম চিন্তা করি , একটা নাম্বার N দেওয়া আছে , 1 থেকে N এর সবগুলো সংখ্যার ডিভিজরস গুলো বের করতে হবে ।

যেমন :

N = 6

১ : ১

২: ১ , ২

৩: ১ , ৩

৪: ১ , ২ , ৪

৫: ১ , ৫

৬: ১ , ২ , ৩ , ৬

আমরা যদি একদম সাধারন ভবে চিন্তা করি তাহলে এটা তেমন কঠিন কাজ না যদি কেউ কোন নাম্বার এর ডিভিজরস বের কতে পারে ।

নরমালি 1 থেকে N পর্যন্ত একটা লুপ চালাতে হবে আর প্রতিট i এর জন্য i এর ডিভিজরস গুলো বের করতে হবে । মনে করি , আমরা i এর জন্য i এর সব ডিভিজরস গুলো কে i তম ভেক্টরে রাখব ।

ইমপ্লিমেন্টেশন :

```cpp
#include <bits/stdc++.h>
using namespace std ;
#define mxN 100000
vector <int> divisor[mxN + 5] ;
void DivisorsUptoN(int n) {
   for(int i = 1 ; i <= n ; i++) {
        int sq = sqrt(i) ;
        for(int j = 1 ; j <= sq ; j++) {
            if(i % j == 0) {
                divisor[i].push_back(j) ;
                divisor[i].push_back(i / j) ;
            }
        }
        if(sq * sq == i)
            divisor[i].pop_back() ;
    }
}
int main() {
    int n ;
    printf("Enter the number : ") ;
    scanf("%d",&n) ;
    DivisorsUptoN(n) ;
    printf("Divisors List upto n : \n") ;
    for(int i = 1 ; i <= n ; i++) {
        printf("%d : " , i) ;
        for(int j = 0 ; j < divisor[i].size() ; j++)
            printf("%d ", divisor[i][j]);
        printf("\n");
    }
    return 0 ;
}
```

এখানে সমস্যা হচ্ছে কমপ্লেক্সিটি বেশি হয়ে যাচ্ছে। একটা সংখ্যার ডিভিজরস বের করতে সময় লাগে Sqrt(N) আর N টার জন্য লাগবে N * Sqrt(N) টাইম। যেটি অনেক বেশি। কিন্তু যদি আমরা সিভ অফ ডিভিজরস করি তাহলে সিভের কমপ্লেক্সিটি অর্থাৎ NlogN এ এই কমপ্লেক্সিটিতে নামিয়ে আনতে পারি।

আমরা আগেই দেখেছি সিভ অ্যালগরিদমে ১ থেকে N পর্যন্ত প্রাইম বের করি এবং কোন সংখ্যা প্রাইম হলে তার সবগুলো মাল্টিপল নন প্রাইম হিসেবে চিহ্নিত করে দেই। ব্যপারটা আমরা যদি এখন এভাবে একটু চিন্তা করি :

কোন একটা i এর সব মাল্টিপল যদি j হয় তাহলে কিন্তু বলা যায় ওই j এর একটা ডিভিজর i। যেমন : ২ এর মাল্টিপলগুলো ২ সহ , ২ , ৪ , ৬ , ৮ ,... । তাহলে আমরা বলতে পারি ২ , ৪ , ৬ , ৮ এর একটা ডিভিজর ২ ! আবার ৩ এর সব মাল্টিপল ৩ সহ , ৩ , ৬ , ৯ , ...। তাহলে আমরা বলতে পারি ৩ , ৬ , ৯ এর একটা ডিভিজর ৩ ! এভাবে আমরা i = ১ থেকে N পর্যন্ত লুপ চালাবো আর আরেকটা লুপ j = i থেকে j + i করে i এর মাল্টিপলদের ওপর N পর্যন্ত চালাবো। তাহলে আমরা একটা i এর জন্য প্রত্যেকটা j তে যাচ্ছি , যেখানে সেই j এর একটা ডিভিজর হচ্ছে i । আমরা এখানে তাহলে ভেক্টর ব্যবহার করতে পারি , যেহেতু j এর ডিভিজর i তাহলে j তম ভেক্টরে i পুশ করতে পারি। এভাবে সিভ মেথডে সিভের কমপ্লেক্সিটিতে এটি সব গুলো ডিভিজর বের করে দিচ্ছে।

ইমপ্লিমেন্টেশন :

```cpp
#include <bits/stdc++.h>
using namespace std ;
#define mxN 100000
vector <int> divisor[mxN + 5] ;
void SieveofDivisors(int n) {
    for(int i = 1 ; i <= n ; i++) {
        for(int j = i ; j <= n ; j += i) {
            divisor[j].push_back(i) ;
        }
    }
}
int main() {
    int n ;
    printf("Enter the number : ") ;
    scanf("%d",&n) ;
    SieveofDivisors(n) ;
    printf("Divisors List upto n : \n") ;
    for(int i = 1 ; i <= n ; i++) {
        printf("%d : " , i) ;
        for(int j = 0 ; j < divisor[i].size() ; j++)
            printf("%d ", divisor[i][j]);
        printf("\n");
    }
    return 0 ;
}
```

প্রাকটিস প্রব্লেম :

https://codeto.win/contest/31/problem/G

প্রাইম ফ্যাক্টরাইজেশন

প্রাইম ফ্যাক্টরাইজেশন শব্দটা অনেক কঠিন মনে হলেও এটি আমাদের সেই স্কুলের চিরচেনা কন্সেপ্ট । প্রাইম ফেক্টরাইজেশন আর বাংলায় মৌলিক উৎপাদকে বিশ্লেষণ, অর্থাৎ কোন একটা সংখ্যাকে এমন ভাবে ভেঙ্গে ফেলা যাতে তাদের আর ভাঙ্গা না যায় এবং ভাঙ্গা অংশগুলো আমাদের সেই সংখ্যাটির একটা রিপ্রেজেন্টেশন হয় । একটু খেয়াল করলে দেখব ভাঙ্গা অংশ গুলো মৌলিক সংখ্যা হলে আর তাদের ভাঙ্গা যায় না , কারন মৌলিক বা প্রাইম মানেই হচ্ছে তাকে আর ভাঙ্গা যাবে না ।

ফরমালি বলতে গেলে কোন সংখ্যা N কে প্রাইম সংখ্যা সমুহ p1 , p2 , p3 , p4 ,.... এর গুনফল আকারে প্রকাশ করা হলে সেই প্রাইম সংখ্যাগুলো হবে N এর প্রাইম ফ্যাক্টরস এবং এই প্রক্রিয়াটিই প্রাইম ফ্যাক্টরাইজেশন ।

যেমন:

৪ = ২ * ২

৫ = ৫

৬ = ২ * ৩

৮ = ২ * ২ * ২

১০ = ২ * ৫

১১ = ১১

১২ = ২ * ২ * ৩

N = p1 * p2 * p3 * .... * pk

তাহলে প্রাইম ফ্যাক্টরাইজেশন সম্পর্কে একটা ধারনা পাওয়া গেলো । এখন আমরা এটা লজিকালি কিভাবে বের করতে পারি? N এর প্রাইম ফ্যাক্টরাইজেশন করতে ২ থেকে N পর্যন্ত সবগুলো সংখ্যা i দিয়ে চেক করতে পারি , যদি N কে i দ্বারা ভাগ দেওয়া যায় বা n % i == 0 হয় তাহলে যতক্ষন i দ্বারা মড শুন্য হয় ততক্ষণ i দ্বারা ভাগ দিয়ে জেতেই পারি এবং সাথে আরেকটা অ্যারেতে প্রাইম ফেক্টরগুলোকে অর্থাৎ i কে করে সেভ রাখতে পারি ।

এখন একটু চিন্তা করি আমাদের আসলেই কি ২ থেকে N পর্যন্ত লুপ চালানোর দরকার আছে নাকি! না নেই। কারন আমরা ডিভিজর এর লজিক্টা এখানেও ব্যবহার করতে পারি। যেহেতু N কে ভাগ করা হচ্ছে, তাহলে আমরা জানি ভাগের ক্ষেত্রে sqrt(N) এর পরে না গেলেও চলে। আমরা sqrt(N) এর মধ্যেই আমাদের অপারেশন করতে পারি। এখন ২-৩ টা সংখ্যার কথা বিবেচনা করে দেখি।

৪ এর জন্য : sqrt(4) = 2 অর্থাৎ আমরা ৪ এর প্রাইম ফেক্টরস বের করার জন্য ২ থেকে ২ পর্যন্ত লুপ চালাব অর্থাৎ ১ বার এবং ৪ / ৪ = ২, আবার ২ / ২ = ১

৪ = ২ * ২

অর্থাৎ ২ বার ২ দ্বারা ভাগ দেওয়া হয়েছে কারন ১, ২ থেকে ছোট তাই আর দেওয়া যাচ্ছে না।

আবার ৫ এর জন্য : sqrt(5) = (int)2 তাহলে আমাদের ২ থেকে ২ পর্যন্ত অর্থাৎ ১ বার লুপ চালাব এবং আমরা দেখলাম ৫ % ২ != ০, তাহলে ব্যপারটা কি হল ? ৫ এর কোন প্রাইম ফেক্টর নেই ! আসলে ৫ নিজেই একটা প্রাইম, তাহলে ৫ এর প্রাইম ফেক্টর সে নিজেই।

তাহলে আমরা এক্ষেত্রে কি করতে পারি! যদি দেখি লুপ থেকে বের হবার পর আমদের n != 1 তার মানে n একটা প্রাইম নাম্বার যেটি আমাদের N এর প্রাইম ফ্যাক্টরস লিস্টে রেখে দিব।

৫ = ৫

১০ এর জন্য : sqrt(১০) = (int)৩ তাহলে আমরা ২ থেকে ৩ পর্যন্ত লুপ চালাব অর্থাৎ ২ বার, প্রথম ক্ষেত্রে ১০ % ২ == ০, তাহলে ১০ / ২ = ৫, আবার ৫ % ২ != ০, তাহলে আমরা এবার ৩ দিয়ে চেষ্টা করবো, ৫ % ৩ != ০।

সুতরাং আমারা শুধু ২ দিয়ে একবার ভাগ দিতে পেরেছি কিন্তু এখন n = ৫ , এটিও তাহলে লিস্টে এড করতে হবে যেহেতু n != 1।

১০ = ২ * ৫

এভাবে আমরা sqrt(N) এই প্রাইম ফেক্টরস বের করতে পারি। যেহেতু ভেতরের ডিভিশন অপারেশন সর্বোচ্চ logN টাইম হতে পারে তাহলে এর কমপ্লেক্সিটি sqrt(N) * logN হবার কথা নয় ? কিন্তু এখানে কিছু মাথম্যাটিকাল ব্যাপার আছে।

এর কমপ্লেক্সিটি sqrt(N) এই থাকে । এর পরুফ নাহয় গিগসফরগিক থেকে দেখে নেওয়া যায় !

ইমপ্লিমেন্টেশন :

```cpp
#include <bits/stdc++.h>
using namespace std ;
int fac[100] ;
int sz ;
void PrimeFactorisation(int n) {
    sz = 0 ;
    int sq = sqrt(n) ;
    for(int i = 2 ; i <= sq ; i++) {
        while(n % i == 0) {
            fac[sz++] = i ;
            n /= i ;
        }
    }
    if(n != 1)
        fac[sz++] = n ;
}
int main() {
    int n ;
    scanf("%d",&n) ;
    PrimeFactorisation(n) ;
    printf("Prime Factors of %d: ", n);
    for(int i = 0 ; i < sz ; i++)
        printf("%d ", fac[i]);
    printf("\n") ;
    return 0 ;
}
```

এখানে যদি আমরা একটা জিনিস লক্ষ্য করি তাহলে আমরা ২ , ৩ , ৪ , ৫ , ৬ , ৭ , ৮ , …এই সব গুলো নাম্বার দিয়েই ইটারেশন করছি কিন্তু আমাদের ৪ , ৬ , ৮ , ৯ … এদের তো নেওয়ার দরকারই নেই , কারন প্রাইম ফেক্টর তো শুধু প্রাইম সংখ্যাই হবে । তাহলে এক্ষেত্রে আমরা কিছু ইটারেশন বাড়তি করছি যেটি চাইলে আমরা বাদ দিতে পারি , আমরা শুধু প্রাইম নাম্বার দিয়েই ভাগ দিতে পারি । তাহলে আমরা আগে প্রাইম লিস্ট বানাবো সিভের মাধ্যমে এবং সেই লিস্টের নাম্বার দিয়ে

ভাগ দিয়ে প্রাইম ফেক্টর বের করব । এখত্রে যদি N <= ১০^৭ তাহলে প্রাইম ফেক্ট্র log(n) টাইমে বের করা সম্ভব ।

প্রাইম ফ্যাক্টরাইজেশন দিয়ে অনেক টাইপের প্রবলেম সল্ভ করা যায় । নাম্বার থিওরির একটা প্রিন্সিপল টপিক প্রাইম ফ্যাক্টরাইজেশনকে বলা যেতে পারে ।

কোড :

```cpp
#include <bits/stdc++.h>
using namespace std ;
#define mxN 10000
bool isp[mxN + 10] ;
int prime[mxN + 10] ;
int primesz ;
int fac[100] ;
int sz ;
void Sieve() {
    primesz = 0 ;
    for(int i = 0 ; i <= mxN ; i++) isp[i] = 0 ;
    isp[0] = isp[1] = 1 ;
    for(int i = 2 ; i <= mxN ; i++) {
        if(isp[i] == 0) {
            for(int j = i + i ; j <= mxN ; j += i) {
                isp[j] = 1 ;
            }
        }
    }
    for(int i = 1 ; i <= mxN ; i++) {
        if(isp[i] == 0) {
            prime[primesz++] = i ;
        }
    }
}
```

```
void PrimeFactorisation(int n) {
    int sq = sqrt(n) ;
    sz = 0 ;
    for(int i = 0 ; prime[i] <= sq && i < primesz ; i++) {
        while(n % prime[i] == 0) {
            fac[sz++] = prime[i] ;
            n /= prime[i] ;
        }
        if(isp[n] == 0) {
            fac[sz++] = n ;
            break ;
        }
    }
}
int main() {
    Sieve() ;
    int n ;
    scanf("%d",&n) ;
    PrimeFactorisation(n) ;
    printf("Prime Factors of %d: ", n);
    for(int i = 0 ; i < sz ; i++)
        printf("%d ", fac[i]);
    printf("\n") ;
    return 0 ;
}
```

প্রাকটিস প্রবলেম :

UVA : 583 - Prime Factors, 516 - Prime Land, 11466 - Largest Prime Divisor,