**// primality test in O(sqrt(n))**

**Bool isprime(int n){**

**if(n==1) return false;**

**for(int i=2;i\*i<=n;i++){**

**if(n%i==0) return false;**

**}**

**Return true;**

**}**

**// sieve of Eratosthenes**

**N is approx 10^6**

**Time complexity(O(log(log(n)))**

**Extra space O(n)**

**Int sieve[1000001]={0};**

**Void prime(){**

**sieve[0]=1;**

**sieve[1]=1;**

**for(int i=2;i<=1000000;i++){**

**if(sieve[i])**

**continue;**

**for(int u=2\*i;u<=1000000;u+=i){**

**sieve[u]=i;**

**}**

**}**

**}**

**// code n code sieve**

**// If array val ==1 it’s prime**

**Int is\_prime[1000001];**

**Void sieve(){**

**Int maxN=1000000;**

**for(int i=1;i<=maxN;i++) is\_prime[i]=1;**

**is\_prime[0]=is\_prime[1]=0;**

**for(int i=2;i\*i<=maxN;i++){**

**if(is\_prime[1]){**

**for(int j=i\*i;j<=maxN;j+=i)**

**is\_prime[j]=0;**

**}**

**}**

**}**

**// prime factorization**

**Brute force approach**

**Void primeFactors(int n){**

**for(int i=2;i<=n;i++){**

**if(n%i==0){**

**Int cnt=0;**

**while(n%i==0){**

**Cnt++;**

**n=n/i;**

**}**

**cout<<i<<” “ <<cnt<<endl;**

**}**

**}**

**}**

**Worst case O(n) for prime numbers**

**B. efficient approach for prime factorization**

**lemma::**

**If N is a composite number then at least 1 prime divisor of N is below sqrt(N);**

**Void primeFactors(int N){**

**for(int i=2;i\*i<=N;i++){**

**if(N%i==0){**

**Int cnt=0;**

**while(N%i==0){**

**Cnt++;**

**N=N/i;**

**}**

**cout<<i<<” ”<<cnt<<endl;**

**}**

**}**

**if(N>1)**

**cout<<N<<” ”<<1<<endl;**

**}**

**Mere pasand ka code**

**vector<int> factors;**

**Void prime\_factor(int N){**

**if(N%2==0){**

**factors.push\_back(2);**

**}**

**while(N%2==0){**

**N=N/2;**

**}**

**for(int i=3;i\*i<=N;i+=2){**

**if(N%i==0) v.push\_back(i);**

**Int cnt=0;**

**while(N%i==0){**

**N=N/i;  
cnt++;**

**}**

**}**

**}**

**if(N>1) v.push\_back(N);**

**// sqrt(N) time complexity**

**Be can also do O(LOgn) with the help of sieve**

**// BINARY EXPONENTIATION**

**Calculate A^n in O(logn ) time complexity**

**Int solve(int base , int n){**

**if(n==1) return base;**

**if(n==0 || base==1) return 1;**

**Int x= solve(base\*\*2, n/2);**

**if(n%2==0){**

**Return x**

**}**

**Else{**

**Return Base \*x;**

**}**

**}**

**Modular exponentiation**

**Int power(int a, int p){**

**Int res=1;**

**while(n){**

**if(n%2==0){**

**a=a\*a;**

**n=n/2;**

**}**

**Else{**

**res=res\*a;**

**n--**

**}**

**}**

**Return res;**

**}**

**// prime factorization using sieve**

**Const Int maxN=1000001;**

**Int prime[maxN];**

**Void sieveFactorisation(){**

**for(int i=0;i<maxN;i++){**

**prime[maxN]=-1;**

**}**

**for(int i=2;i<=maxN;i++){**

**if(prime[i]==-1){**

**for(int j=i;j<=maxN;j++){**

**if(prime[j]==-1)**

**prime[j]=i;**

**}**

**}**

**}**

**}**

**vector<int> v;**

**Int n;**

**while(n>1){**

**v.push\_back(prime[n]);**

**n=n/prime[n];**

**}**

**V contains all the prime factors..**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////////////////////////////////////////////////////////////////**

**MATRIX EXPONENTIATION**

**Given matrix A and an integer N , calculate A^N**

**Time complexity O(M^3 \* N) m is the rank of square matrix**

**Better time complexity : O(M^3\*(log N))**