1.

**//generate prime number up to 10^7..**

#include<bits/stdc++.h>

using namespace std;

#define mx 10000000

int isp[mx+5];

void sieve()

{

isp[0]=1;

isp[1]=1;

int sq= sqrt(mx);

for(int i=2; i<=sq; i++)

{

if(isp[i]==0)

{

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

isp[j]=1;

}

}

}

int main()

{

sieve();

int x;

while(cin>>x)

{

if(x==0)

break;

if(isp[x]==0)

cout<<"prime"<<endl;

else if(isp[x==1])

cout<<"Not prime"<<endl;

}

return 0;

}

**Prime factorizing:**

1.

/\* Read a number and generate prime factor of those number and print the prime factor \*/

// Complexity O(sqrt(N))

// it will work in upto 10^14

#include <bits/stdc++.h>

using namespace std ;

void PrimeFactorisation(long long x) {

vector <long long> pf ;

for(long long i = 2 ; i \* i <= x ; i++) {

if(x % i == 0) {

while(x % i == 0) {

x /= i ;

pf.push\_back(i) ;

}

}

}

if(x != 1)

pf.push\_back(x) ;

// printing the prime factors..

for(long long i = 0 ; i < pf.size() ; i++) {

cout << pf[i] << " " ;

}

}

int main() {

long long n ;

while(cin >> n) {

PrimeFactorisation(n) ; // getting the prime factors of n..

cout << "\n" ;

}

return 0 ;

}

2.

#include <bits/stdc++.h>

using namespace std ;

#define mxN 10000000

bool isp[mxN + 5] ;

vector <int> prime ;

// Complexity of this sieve O(NloglogN) which is faster than O(NlogN)

// Generating all prime numbers upto N and storing them in prime vector..

// this will only work in while mxN <= 10^7

void sieve() {

isp[0] = 1 ;

isp[1] = 1 ;

for(int i = 4 ; i <= mxN ; i += 2)

isp[i] = 1 ;

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

if(isp[i] == 0) {

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

isp[j] = 1 ;

}

}

}

prime.push\_back(2) ;

for(int i = 3 ; i <= mxN ; i += 2)

if(isp[i] == 0)

prime.push\_back(i) ;

}

// factorising the number x..

// Complexity O(logN) per call..

// this will only work in while mxN <= 10^7

void PrimeFactorisation\_logN(int x) {

vector <int> pf ;

for(int i = 0 ; i < prime.size() && prime[i] \* prime[i] <= x ; i++) {

if(isp[x] == 0)

break ;

if(x % prime[i] == 0) {

while(x % prime[i] == 0) {

x /= prime[i] ;

pf.push\_back(prime[i]) ; // storing in pf..

}

}

}

if(x != 1)

pf.push\_back(x) ;

// printing the prime factors..

for(int i = 0 ; i < pf.size() ; i++) {

cout << pf[i] << " " ;

}

}

int main() {

sieve() ; // calling the sieve..

int n ;

while(cin >> n) {

PrimeFactorisation\_logN(n) ; // getting the prime factors of n..

cout << "\n" ;

}

return 0 ;

}

3.

//generating prime upto 10^7 and generate prime factor upto 10^14

#include <bits/stdc++.h>

using namespace std ;

#define mxN 10000000

bool isp[mxN + 5] ;

vector <int> prime ;

// Complexity of this sieve O(NloglogN) which is faster than O(NlogN)

// Generating all prime numbers upto N and storing them in prime vector..

void sieve() {

isp[0] = 1 ;

isp[1] = 1 ;

for(int i = 4 ; i <= mxN ; i += 2)

isp[i] = 1 ;

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

if(isp[i] == 0) {

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

isp[j] = 1 ;

}

}

}

prime.push\_back(2) ;

for(int i = 3 ; i <= mxN ; i += 2)

if(isp[i] == 0)

prime.push\_back(i) ;

}

// factorising the number x..

// Complexity O(sqrtN)in worst case per call..but faster 50 times than without prime method..

// but actually in average and best its O(logN).

// this will work while N <= 10^14

void PrimeFactorisation\_sqrtN(long long x) {

vector <long long> pf ;

for(long long i = 0 ; i < prime.size() && prime[i] \* prime[i] <= x ; i++) {

if(x % prime[i] == 0) {

while(x % prime[i] == 0) {

x /= prime[i] ;

pf.push\_back(prime[i]) ; // storing in pf..

}

}

}

if(x != 1)

pf.push\_back(x) ;

// printing the prime factors..

for(long long i = 0 ; i < pf.size() ; i++) {

cout << pf[i] << " " ;

}

}

int main() {

sieve() ; // calling the sieve..

long long n ;

while(cin >> n) {

PrimeFactorisation\_sqrtN(n) ; // getting the prime factors of n..

cout << "\n" ;

}

return 0 ;

}

**Divisor Count:**

//read a number and count the divisor of those number perfectly

//but it's push same divisor twice if the number is perfect square

//it's work for N<=10^14

#include<bits/stdc++.h>

using namespace std;

void divisor(long long n)

{

vector<long long> div;

div.push\_back(n);//n is divisor of itself.so in the begining cnt=1

int cnt = 1;

int sq = sqrt(n);

for(int i = 2; i<=sq; i++)

{

if(n%i==0)

{

div.push\_back(i);

div.push\_back(n/i);

cnt = cnt+2;

}

}

if(sq\*sq==n)

cnt--;

//sort(div.begin(),div.end());

cout<<"Number of divisor: "<<cnt<<endl;

for(int i = 0; i<cnt; i++)

{

cout<<div[i]<<" ";

}

cout<<endl;

}

int main()

{

long long n;

cin>>n;

divisor(n);

return 0;

}

**String to Number:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

string str;

cin>> str;

int x = 0;

int len = str.size();

for(int i = 0; i< len; i++)

{

x = x\*10 + (str[i]-'0');

}

cout<< x<< endl;

return 0;

}

**Random:**

Size of array:

int arr[]={2,4,6,2,3,9};

int len = sizeof(arr) / sizeof(arr[0]);

sort(arr, arr + arr\_size) for sorting array

**Binary Search:**

// C++ program to implement recursive Binary Search

#include <bits/stdc++.h>

using namespace std;

// A recursive binary search function. It returns

// location of x in given array arr[l..r] is present,

// otherwise -1

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l)

{

int mid = l + (r - l) / 2;

// If the element is present at the middle

// itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x);

}

// We reach here when element is not

// present in array

return -1;

}

int main(void)

{

int x,n;

cout<<"Enter the number of elements:";

cin>>n;

int arr[n+5];

for(int i = 0; i < n; i++)

cin>>arr[i];

sort(arr,arr+n); //sort the array

cout<<"Enter the number that you want to search: ";

cin>>x;

int result = binarySearch(arr, 0, n - 1, x);

if(result == -1)

printf("Element is not present in array");

else

printf("Element is present at index %d", result);

return 0;

}