#include<bits/stdc++.h>

#include<ext/pb\_ds/assoc\_container.hpp>

#include<ext/pb\_ds/tree\_policy.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

#define ll long long

#define ull unsigned long long

#define pq priority\_queue

#define pb push\_back

typedef tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> pbds; // find\_by\_order, order\_of\_key --> these will return iterator

//--> to get value --> \*a.find\_by\_order(i); \*a.order\_of\_key(X);

//\*a.find\_by\_order(i); --> finding kth element

//\*a.order\_of\_key(i); --> finding number of elements smaller than X

//\*a.lower\_bound(X); --> lower bound -> Lower Bound of X = first element >= X in the set

//\*a.upper\_bound(X); --> Upper bound -> Upper Bound of X = first element > X in the set

//a.erase(X); --> Remove X from the ordered set

/\*

//typedef tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> pbds; //this will sort the set in ascending order

//typedef tree<int, null\_type, greater<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> pbds; //this will sort the set in descending order

//typedef tree<int, null\_type, less\_equal<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> pbds; //this will sort the set in ascending order + equal values will be included

\*/

/\*

# deque //both

# stack //LIFO

# queue //FIFO

\*/

//

/\*

# map2.insert(map1.begin(),map1.end()); //elements of map1 will be inserted to map2

# map <char,int>::iterator it;

# it = mp.find('b');

# a.erase(a.begin(), a.end())

# a.erase('c'); //erases element mapped at 'c'.

# if(mp.empty()) {cout<<"Map is empty"<<endl;} //returns true or false value

# for(it=a.begin(); it!=a.end(); it++) {cout<<it->first<<" "<<it->second<<endl;}

# map1.swap(map2); //swap contents of 2 maps namely map1 and map2.

\*/

//

using vb = vector<bool>;

using vvb = vector<vb>;

using vi = vector<int>;

using vvi = vector<vi>;

using vl = vector<ll>;

using vvl = vector<vl>;

using vc = vector<char>;

using vvc = vector<vc>;

using vs = vector<string>;

//

#define all(a) a.begin(), a.end()

#define vsort(a) sort(a.begin(), a.end())

#define srev(a) reverse(a.begin(), a.end())

#define ssort(a) sort(a.begin(), a.end())

#define grtsrt(v) sort(v.begin(), v.end(), greater<ll>())

#define vrot(a, rot\_by) rotate(a.begin(), a.end() - rot\_by, a.end())

//

#define vmin(a) \*min\_element(a.begin(), a.end())

#define vmax(a) \*max\_element(a.begin(), a.end())

#define smax(a) \*(a.rbegin()) //1st check this condition --> if (!a.empty())

#define smin(a) \*a.begin() //1st check this condition --> if (!a.empty())

//

#define vcount(v, a) count(v.begin(), v.end(), a)

#define scount(v, a) count(v.begin(), v.end(), a)

//

#define mp(a, b) make\_pair(a, b)

//

#define ub\_pos(a, x) upper\_bound(a.begin(), a.end(), x) - a.begin()

#define lb\_pos(a, x) lower\_bound(a.begin(), a.end(), x) - a.begin()

#define bins(a, n) binary\_search(a.begin(), a.end(), n) //returns true or false value

//

#define lower\_case(a) transform(a.begin(), a.end(), a.begin(), ::tolower)

#define upper\_case(a) transform(a.begin(), a.end(), a.begin(), ::toupper)

//

#define mem(a, x) memset(a, x, sizeof(a))

//

#define nxp(v) next\_permutation(v.begin(), v.end()); //generates all possible permutations

//

#define toint(a) atoi(a.c\_str())

#define total\_sum(a) accumulate(a.begin(), a.end(), 0) //total\_sum initializing with 0; //this fnc gives the total sum of all numbers in the array

//

#define check(n, pos) (n & (1<<(pos)))

#define biton(n, pos) (n | (1<<(pos)))

#define bitoff(n, pos) (n & ~(1<<(pos)))

#define get\_bit(n, pos) (bool)(n & (1 << pos)) //returns bool value --> 0 or 1;

//indexing starts from 0 --> 1st position bit is 0th bit, then 1st bit, 2nd bit and so on

#define set\_bit(n, pos) (n | (1 << pos))

//indexing starts form 0 --> meaning k=2; this 2nd bit is the (k-1)th; 1st position bit

#define clear\_bit(n, pos) (n & (~(1 << pos))) //~(1 << k) means the kth bit of ~(1 << k) is always 0

//clearing the kth bit --> ensuring that the kth bit is unset (0)

#define update\_bit(n, pos, value) ((clear\_bit(n, pos)) | (value << pos))

//value represents the bit we want to set at the kth position. To do so we need to clear the kth bit

//so we declared it first, then we are going to set the bit and the operation between them is or

const double pi= 3.141592653589793238462643383279502884197169399375105820974944;

const ll N=1e9+7;

const ll mod = 1e9 + 7, inf = 1e18;

/\*\*/

//if(s1.compare(s2) == 0) {cout << s << " is equal to " << s1 << endl;} --> here s1 and s2 are 2 strings and we are comparing these strings; if == 0 than they are same otherwise different

//string s = "dog:cat"; int pos = s.find(":"); string sub = s.substr(pos + 1); // Copy substring after pos; --> String is: cat

//string s1 = "Geeks"; string r = s1.substr(3, 2); // Copy two characters of s1 (starting from position 3); --> String is: ks

/\*\*/

//\_\_int128(n)\*mid\*mid\*4 <-- Here n\*mid\*mid\*4 is written using --> \_\_int128() which may not support in 64 bit computer

//but this is used for calculating large int value that can't be processed in i64 or long long

/\*\*/

//2^n is equal to (1 << n)

//xor of two same numbers is equal to 0

/\*

//a+b=a|b + a&b //a+b=a⊕b+2(a&b) here (a | b) equals to (a⊕b + a&b)

//a-b=a-(a&b)-x where x is basically (bitwise not of a) & b

//a^(a&b) = (a|b)^b

//b^(a&b) = (a|b)^a

Some properties of bitwise operations:

a|b = a⊕b + a&b

a⊕(a&b) = (a|b)⊕b

b⊕(a&b) = (a|b)⊕a

(a&b)⊕(a|b) = a⊕b

Addition:

a+b = a|b + a&b

a+b = a⊕b + 2(a&b)

Subtraction:

a-b = (a⊕(a&b))-((a|b)⊕a)

a-b = ((a|b)⊕b)-((a|b)⊕a)

a-b = (a⊕(a&b))-(b⊕(a&b))

a-b = ((a|b)⊕b)-(b⊕(a&b))

\*/

/\*

int rotate\_by=k%(n+1);

rotate(a.begin(), a.end() - rotate\_by, a.end());

\*/

/\*

# bitset<66> bits(x); //bitset operation --> converts any number to bits, can mention how many bits we want; here 66 bits that we want

//using the following operation we can convert the bits into string:

string res="";

res=bits.to\_string();

//using the following operation we can exclude the extra 0's that comes in front of the number:

reverse(res.begin(), res.end());

while(res.back()=='0')

{

res.pop\_back();

}

reverse(res.begin(), res.end());

here if 4 -> 000000100 is found using bitset; by using the above method we can get 100 the required bits only.

\*/

/\*dfs code: \*/

const int M = 1e5+10;

vector<int> g[M];

bool vis[M];

//4 sections in dfs to write code

//every section has a definition

void dfs(int vertex){

/\* Take action on vertex after entering the vertex \*/

//if(vis[vertex]) return; //this can be written if inside for we havent written vis child condn

cout<<vertex<<"\n";

vis[vertex] = true;

for(int child:g[vertex]){

cout<<"parent"<<vertex<<" ,child"<<child<<"\n";

if(vis[child]) continue;

/\* Take action on child before entering the child node \*/

dfs(child);

/\* Take action on child after existing the child node \*/

}

/\* Take action on child before existing the vertex \*/

}

int graph\_count\_connected\_components(int n)

{

int ctr=0;

for(int i=1; i<=n; i++)

{

if(vis[i]) continue;

dfs(i);

ctr++;

}

return ctr;

}

/\*

The numbers of triangles in complete graph with n nodes is: n\*(n−1)\*(n−2)/6

\*/

bool dfs\_loop\_exists(int vertex, int parent)

{

vis[vertex]= true;

bool isLoopExists= false;

for(int child: g[vertex])

{

if(vis[child] && child==parent) continue;

if(vis[child]) return true;

isLoopExists |= dfs\_loop\_exists(child, vertex); //or equal --> |=

}

return isLoopExists;

}

void graph\_is\_loop\_exists(int n)

{

bool isLoopExists = false; //initializing with false value

// if at least 1 cycle exists then

//replace with true value

for(int i=1; i<=n; i++)

{

if(vis[i]) continue;

if(dfs\_loop\_exists(i, 0))

{

isLoopExists = true;

break;

}

}

cout<<isLoopExists<<endl;

}

//have to call these vactors before the dfs fnc

vector<vector<int>> cc;

vector<int>current\_cc;

void graph\_list\_connected\_components(int n)

{

int ctr=0;

for(int i=1; i<=n; i++)

{

if(vis[i]) continue;

current\_cc.clear();

dfs(i);

cc.push\_back(current\_cc);

}

//printing size or number of connected components

cout<<"number of connected components: "<<cc.size()<<endl;

//printing the list of connected components

for(auto c\_cc : cc)

{

for(int vertex : c\_cc)

{

cout<<vertex<<' ';

}

cout<<endl;

}

}

const int M = 1e5+10;

vector<int> g[M];

int depth[M], height[M];

void dfs\_for\_trees(int vertex, int parent)

{

for(int child:g[vertex])

{

if(child==parent) continue;

depth[child]=depth[vertex] + 1;

dfs\_for\_trees(child, vertex);

height[vertex]=max(height[vertex], height[child] + 1);

}

}

void graph\_input()

{

int n;

cin>>n;

for(int i=1; i<n; i++)

{

int x, y;

cin>>x>>y;

g[x].push\_back(y);

g[y].push\_back(x);

}

dfs\_for\_trees(1, -1);

}

void answer()

{

graph\_input();

}

const int M = 1e5+10;

vector<int> g[M];

int depth[M], height[M];

void dfs\_for\_trees(int vertex, int parent)

{

for(int child:g[vertex])

{

if(child==parent) continue;

depth[child]=depth[vertex] + 1;

dfs\_for\_trees(child, vertex);

height[vertex]=max(height[vertex], height[child] + 1);

}

}

void graph\_input()

{

int n;

cin>>n;

for(int i=1; i<n; i++)

{

int x, y;

cin>>x>>y;

g[x].push\_back(y);

g[y].push\_back(x);

}

dfs\_for\_trees(1, -1);

}

void answer()

{

graph\_input();

}

const int segSize=1e5+5;

vector<int> a(segSize), seg(4 \* segSize);

void build(int index, int low, int high)

{

if(low == high)

{

seg[index] = a[low];

return;

}

int mid = (low + high) / 2;

build(2\*index+1, low, mid); //left side of the tree

build(2\*index+2, mid+1, high); //right side of the tree

//the below expression can be changed according to the ques:

seg[index] = max(seg[2\*index+1], seg[2\*index+2]); //--> here this expression gives the max element from index l to r

}

int query(int index, int low, int high, int l, int r)

{

if(low>=l && high<=r)

{

return seg[index];

}

if(high<l || low>r)

{

return INT\_MIN;

}

int mid = (low + high) / 2;

int left = query(2\*index+1, low, mid, l, r);

int right = query(2\*index+2, mid+1, high, l, r);

return max(left, right);

}

/\*\*/

void segment\_tree()

{

//taking input:

int n;

cin>>n;

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

{

cin>>a[i];

}

build(0, 0, n-1);

//taking query:

int q;

cin>>q;

while(q--)

{

int l, r;

cin>>l>>r;

cout<<query(0, 0, n-1, l, r)<<endl;

}

}

/\*\*/

const int MX = 1e9;

// === Segment Tree ===

struct Node;

struct Update;

template <typename T, typename Node = Node, typename Update = Update>

class SegmentTree {

private:

int size = 0;

vector<Node> seg;

void build (int start, int end, int ind, vector<T> &arr) {

if(start == end) {

seg[ind] = Node(arr[start]);

return;

}

int mid = (start + end) / 2;

int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;

build(start, mid, leftInd, arr);

build(mid + 1, end, rightInd, arr);

seg[ind].merge(seg[leftInd], seg[rightInd]);

}

Node query (int start, int end, int ind, int left, int right) {

if(start > right || end < left) {

return Node();

}

if(start >= left && end <= right) {

return seg[ind];

}

int mid = (start + end) / 2;

int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;

Node res;

Node leftItem = query(start, mid, leftInd, left, right);

Node rightItem = query(mid + 1, end, rightInd, left, right);

res.merge(leftItem, rightItem);

return res;

}

void update (int start, int end, int ind, int index, Update &u) {

if(start == end) {

u.apply(seg[ind]);

return;

}

int mid = (start + end) / 2;

int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;

if(index <= mid) update(start, mid, leftInd, index, u);

else update(mid + 1, end, rightInd, index, u);

seg[ind].merge(seg[leftInd], seg[rightInd]);

}

public:

SegmentTree () {

}

SegmentTree (int n) {

size = n;

seg.resize(4 \* size + 1);

}

SegmentTree (vector<T> &arr) {

size = arr.size();

seg.resize(4 \* size + 1);

build(arr);

}

void build (vector<T> &arr) {

build(0, size - 1, 0, arr);

}

Node query (int left, int right) {

return query(0, size - 1, 0, left, right);

}

void update (int index, int value) {

Update u = Update(value);

return update(0, size - 1, 0, index, u);

}

};

struct Node {

long long sum;

int mx, mn;

long long andd;

Node () : sum(0), mx(-MX), mn(MX), andd((1LL << 32) - 1) {}

Node (int val) : sum(val), mx(val), mn(val), andd(val) {}

void merge (Node &left, Node &right) {

sum = left.sum + right.sum;

mx = max(left.mx, right.mx);

mn = min(left.mn, right.mn);

andd = left.andd & right.andd;

}

};

struct Update {

int val;

Update () : val(0) {}

Update (int v) : val(v) {}

void apply (Node &node) {

node.sum = val;

node.mn = val;

node.mx = val;

node.andd = val;

}

};

ll binpow(ll a,ll b) {

ll ans = 1;

while(b > 0) {

if((b & 1) == 1) ans \*= a;

a \*= a;

b = b >> 1;

}

return ans;

}

ll binpowmod(ll a,ll b) {

ll ans = 1;

while(b > 0) {

if((b & 1) == 1) {

ans \*= a;

ans %= mod;

}

a \*= a;

a %= mod;

b = b >> 1;

}

return ans;

}

ll gcd(ll a,ll b) {

if(b == 0) return a;

return gcd(b, a % b);

}

ll lcm(ll a,ll b) {

return (a / gcd(a,b)) \* b;

}

const ll MAX = 2e5 + 7;

vector<ll> fact(MAX);

ll add(ll a, ll b) {

return (a + b) % mod;

}

ll sub(ll a, ll b) {

return ((a - b) % mod + mod) % mod;

}

ll mult(ll a, ll b) {

return ((a \* b) % mod);

}

ll inv(ll a) {

return binpowmod(a, mod - 2);

}

ll divide(ll a, ll b) {

return mult(a, inv(b));

}

ll nCr(ll n, ll r) {

if(n < r) return 0;

return divide(fact[n], mult(fact[r], fact[n - r]));

}

void preFactorial() {

fact[0] = 1;

for(ll i = 1; i < MAX; i++) fact[i] = mult(i, fact[i - 1]);

}

bool isVowel(char c) {

if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') return true;

if(c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') return true;

return false;

}

bool isSame(ll n, ll arr[]) {

for(ll i = 0; i < n; i++) {

if(arr[i] != arr[0]) return false;

}

return true;

}

bool isSame(ll n, vector<ll> &arr) {

for(ll i = 0; i < n; i++) {

if(arr[i] != arr[0]) return false;

}

return true;

}

bool isSorted(ll n, ll arr[]) {

for(ll i = 1; i < n; i++) {

if(arr[i] < arr[i - 1]) return false;

}

return true;

}

bool isSorted(ll n, vector<ll> &arr) {

for(ll i = 1; i < n; i++) {

if(arr[i] < arr[i - 1]) return false;

}

return true;

}

void inputArr(ll n, ll arr[]) {

for(ll i = 0; i < n; i++) cin >> arr[i];

}

void inputArr(ll n, vector<ll> &arr) {

ll x;

for(ll i = 0; i < n; i++) {

cin >> x;

arr.push\_back(x);

}

}

void printArr(ll n, ll arr[]) {

for(ll i = 0; i < n; i++) cout << arr[i] << " ";

cout << nl;

}

void printArr(ll n, vector<ll> &arr) {

for(ll i = 0; i < n; i++) cout << arr[i] << " ";

cout << nl;

}

ll sumOfArr(ll n, ll arr[]) {

ll ans = 0;

for(ll i = 0; i < n; i++) ans += arr[i];

return ans;

}

ll sumOfArr(ll n, vector<ll> &arr) {

ll ans = 0;

for(ll i = 0; i < n; i++) ans += arr[i];

return ans;

}

bool isPrime(ll n) {

if(n == 1) return false;

for(ll i = 2; i <= sqrt(n); i++) {

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

}

return true;

}

ll countSetBits(ll n) {

ll ans = 0;

while(n) {

ans++;

n = n & (n - 1);

}

return ans;

}

vector<ll> primeFactorization(ll n) {

vector<ll> factors;

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

ll cnt = 0;

while(n % i == 0) {

cnt++;

n /= i;

factors.push\_back(i);

}

}

if(n > 1) factors.push\_back(n);

return factors;

}

bool isPalindrome(string s) {

ll i = 0;

ll j = s.size() - 1;

while(i <= j) {

if(s[i] != s[j]) return false;

i++;

j--;

}

return true;

}

const int pre\_limit\_2d=1e3+10;

ll a[pre\_limit\_2d][pre\_limit\_2d];

ll pref[pre\_limit\_2d][pre\_limit\_2d];

void prefix\_sum\_2d\_array()

{

int n;

cin>>n;

//taking input for prefix sum for 2d array for n\*n square matrix; it can be rectangular as well.

for(int i=1; i<=n; i++) //in prefix sum we need to initialize i with 1; not 0.

{

for(int j=1; j<=n; j++)

{

cin>>a[i][j];

pref[i][j] = a[i][j] + pref[i-1][j] + pref[i][j-1] - pref[i-1][j-1];

}

}

//for q queries we will print the prefix sum form a point to another point (diagonal points will be given)

/\*

............

......sXXX..

......XXXX..

......XXXe..

here s and e are 2 points of the diagonal and we are going to calculate the sum of

are rectangle staring from s and ending at e.

\*/

int q;

cin>>q;

while(q--)

{

int x1, y1, x2, y2;

cin>>x1>>y1>>x2>>y2;

cout<<pref[x2][y2] - pref[x1-1][y2] - pref[x2][y1-1] + pref[x1-1][y1-1]<<endl;

}

}

const int prefsize = 2e5+7;

int prefsum[prefsize][32];

void prefix\_sum\_of\_bits(vl &a, int n) //Builds the prefix sums for each bit

{

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

{

for(int j=0; j<32; j++)

{

if((a[i] & (1<<j)))

{

prefsum[i+1][j] = prefsum[i][j]+1; //careful (i+1) is used before '='

}

else

{

prefsum[i+1][j] = prefsum[i][j];

}

}

}

}

const ll int pr=1e7+10;

vector<bool> isPrime(pr,1);

vector<ll int> lp(pr,0), hp(pr,0);

void prime\_array()

{

isPrime[0]= isPrime[1]= false;

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

{

if(isPrime[i]== true)

{

lp[i]=hp[i]=i;

for(int j=2\*i; j<pr; j+=i)

{

isPrime[j] = false;

hp[j]=i;

if(lp[j]==0)

{

lp[j]=i;

}

}

}

}

}

ll int num;

cin>>num;

map<ll int ,ll int> prime\_factors;

while(num > 1)

{

ll int prime\_factor =hp[num];

while(num % prime\_factor ==0)

{

num /= prime\_factor;

prime\_factors[prime\_factor]++;

}

}

//for a^b

ll binpow(ll x, ll y)

{

if(y==0)

return 1;

ll res= binpow(x, y/2);

if(y%2==0)

return (res \* res) % N;

else

return (((res \* res) % N) \* x) % N;

}

void solve()

{

}

int main()

{

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

// #ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

// #endif

int t=1;

cin>>t;

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

{

solve();

}

return 0;

}