const int inf = 1 << 30; //DIJKSTRAS

// given adjacency matrix adj, finds shortest path from A to B

lli dijk(lli A, lli B, vector< vector<lli> > adj) {

lli n = adj.size();

vector<lli> dist(n);

vector<bool> vis(n);

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

dist[i] = inf;

}

dist[A] = 0;

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

int cur = -1;

for(int j = 0; j < n; ++j) {

if (vis[j]) continue;

if (cur == -1 || dist[j] < dist[cur]) {

cur = j;

}

}

vis[cur] = true;

for(int j = 0; j < n; ++j) {

lli path = dist[cur] + adj[cur][j];

if (path < dist[j]) {

dist[j] = path;

}

}

}

return dist[B];

}-------------------------------------------------------------------

vector <lli> f,adj[100000],visited; //adj[][] is the adjacency list DFS

void dfs(lli p) {

visited[p] = 1;

for(lli i = 0;i < adj[p].size();i++)

if(visited[i])

dfs(p);

}------------------------------------------------------------------------------

vector <lli> adj[100000];

vector <bool> visited; //adj[][] is the adjacency list BFS

void bfs(lli p)

{

deque<lli> Q;

Q.clear();

visited[p]=1;

while (Q.empty()) {

lli p1 = Q.front();

Q.pop\_front();

for (int i=0; i<adj[p1].size(); i++) {

if( !visited[ adj[p1][i] ] )

{

Q.push\_back( adj[p1][i] );

visited[adj[p1][i]]=1;

}

}

}

}----------------------------------------------------------------------------------

void zalgo(string s, string s1, int \*z)

{

int R=0,L=0;

s1 += "$";

s1 += s;

s = s1;

long int n=s.length();

z[0]=0;

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

{

if (i > R)

{

L = R = i;

while (R < n && s[R-L] == s[R])

{

R++;

}

z[i] = R-L;

R--;

}

else

{

int k = i-L;

if (z[k] < R-i+1)

{

z[i] = z[k];

}

else

{

L = i;

while (R < n && s[R-L] == s[R])

{

R++;

}

z[i] = R-L;

R--;

}

}

}

}

int main()

{

string s1,s2;

cin>>s1>>s2;

int a[1000000];

zalgo(s1, s2, a);

// in a[], if s2.length = l, a[i]=l means that in s1, there is a substring s2 is starting at index i.

}-------------------------------------------------------------------------------------

#include <bits/stdc++.h> // single simple struct

#include <cmath>

#include <cstdio>

#include <vector>

#include <queue>

#include <iostream>

#include <algorithm>

#include <cstdlib>

#define REP(i,x,y) for(int i = x; i<=y; i++)

using namespace std;

//This is for problem of finding max sum of a subinterval in query range [l,r]

struct data {

int l\_sum = 0, r\_sum = 0, sum = 0, best\_sum = 0;

};

int a[100010]; // 1 indexed

data st[4\*100010]; // 1 indexed

data merge(data l\_data, data r\_data) {

// merge function to appropriately merge two segments

data ans;

ans.l\_sum = max(l\_data.l\_sum, l\_data.sum + r\_data.l\_sum);

ans.r\_sum = max(r\_data.r\_sum, r\_data.sum + l\_data.r\_sum);

ans.sum = l\_data.sum + r\_data.sum;

ans.best\_sum = max( max(l\_data.best\_sum, r\_data.best\_sum), l\_data.r\_sum+r\_data.l\_sum );

return ans;

}

data create(int val) {

data ans;

ans.sum = ans.l\_sum = ans.r\_sum = ans.best\_sum = val;

return ans;

}

void build\_st(int node, int start, int end) {

if(start == end) {

st[node] = create(a[start]);

}

else {

int mid = (start+end)/2;

build\_st(2\*node, start, mid);

build\_st(2\*node+1, mid+1, end);

st[node] = merge(st[2\*node], st[2\*node+1]);

}

}

void update\_st(int node, int start, int end, int l, int r) {

if(start == end) {

// make changes to leaves

}

else {

int mid = (start+end)/2;

if(r<=mid){

update\_st(2\*node, start, mid, l, r);

}

else if(l>mid) {

update\_st(2\*node+1, mid+1, end, l, r);

}

else {

update\_st(2\*node, start, mid, l, r);

update\_st(2\*node+1, mid+1, end, l, r);

}

st[node] = merge(st[2\*node], st[2\*node+1]);

}

}

data query\_st(int node, int start, int end, int l, int r) {

//cout << "query s,e: " << start << " " << end << endl;

if(start>r || end<l) {

// search interval completely outside [l,r]

return create(0);

}

else if(l<=start && end<=r) {

//search interval completely inside [l,r]

return st[node];

}

// search interval partially inside [l,r]

int mid = (start+end)/2;

if(r<=mid) {

return query\_st(2\*node, start, mid, l, r);

}

if(l>mid) {

return query\_st(2\*node+1, mid+1, end, l, r);

}

data m1 = query\_st(2\*node, start, mid, l, r);

data m2 = query\_st(2\*node+1, mid+1, end, l, r);

return merge(m1,m2);

}

int main() {

int t;

scanf("%d",&t);

while(t-->0) {

int n;

scanf("%d",&n);

REP(i,1,n) {

scanf("%d", &a[i]);

}

build\_st(1,1,n);

int q;

scanf("%d",&q);

while(q-->0) {

int x1, y1, x2, y2;

scanf("%d%d%d%d", &x1, &y1, &x2, &y2);

if(x2 > y1) {

int ans = query\_st(1,1,n,x1,y1).r\_sum + query\_st(1,1,n,y1+1,x2-1).sum + query\_st(1,1,n,x2,y2).l\_sum;

printf("%d\n", ans);

}

else {

int ans = max( query\_st(1,1,n,x1,x2-1).r\_sum + query\_st(1,1,n,x2,y2).l\_sum, max( query\_st(1,1,n,x1,y1).r\_sum + query\_st(1,1,n,y1+1,y2).l\_sum, query\_st(1,1,n,x2,y1).best\_sum ) );

printf("%d\n", ans);

}

}

}

return 0;

}-----------------------------------------------------------------------------------------------

//segtree simple RMQ

#include <bits/stdc++.h>

#include <cmath>

#include <cstdio>

#include <vector>

#include <queue>

#include <iostream>

#include <algorithm>

#include <cstdlib>

#define REP(i,x,y) for(int i = x; i<=y; i++)

using namespace std;

int a[1000009] = {0}; // stores input data. Starts from a[1].

int st[10000010]; // stores segtree, starts from st[1]. In general size of st = 4 \* size of input.

int maximum = 0; // problem specific!

//This build function works for finding min in [l,r].

void build\_st(int node, int start, int end) {

// takes O(n)

// node is current node in st. start, end of the corresponding interval in a[].

if(start == end) st[node] = a[start]; // reached leaf

else {

int mid = (start+end)/2;

build\_st(2\*node, start, mid);

build\_st(2\*node+1, mid+1, end);

//---------------CAN EDIT THIS PART ONLY---------------------

st[node] = min(st[2\*node],st[2\*node+1]); // this will depend on the problem

//-----------------------------------------------------------

}

}

//This query function works for finding sum of [l,r].

int query\_st(int node, int start, int end, int l, int r) {

// takes O(log(n))

// start, end for recursion. l, r are query inputs.

//---------------CAN EDIT RETURN VALUES ONLY---------------------

if(start>r || end<l) {

// search interval completely outside [l,r]

return maximum;

}

else if(l<=start && end<=r) {

//search interval completely inside [l,r]

return st[node];

}

// search interval partially inside [l,r]

int mid = (start+end)/2;

int m1 = query\_st(2\*node, start, mid, l, r);

int m2 = query\_st(2\*node+1, mid+1, end, l, r);

return min(m1,m2);

/\*if(l==start&&r==end) {

return a[start];

}

int mid = (start+end)/2;

if(r<=mid) {

return query\_st(2\*node, start, mid, l, r);

}

if(l>mid) {

return query\_st(2\*node+1, mid+1, end, l, r);

}

int m1 = query\_st(2\*node, start, mid, l, r);

int m2 = query\_st(2\*node+1, mid+1, end, l, r);

return min(m1,m2);\*/

//-----------------------------------------------------------

}

int main() {

int n; // size of input data array

cin >> n;

REP(i,1,n) {

cin >> a[i];

if(a[i]>maximum) maximum = a[i];

}

build\_st(1,1,n);

int q; // number of GET queries

cin >> q;

while(q-->0) {

int query\_l, query\_r; // stores left and right limits of query interval

cin >> query\_l >> query\_r;

cout << query\_st(1, 1, n, query\_l+1, query\_r+1) << endl; // MUST account for 1 indexed a[]

}

return 0;

}---------------------------------------------------------------------------------

//sparse table RMQ

#include <bits/stdc++.h>

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <cstdlib>

using namespace std;

int main() {

int n, q, x, y, k;

cin >> n;

//cout << ceil(log2(n));

vector<int> a(n, 0);

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

cin >> a[i];

}

cin >> q;

vector<vector<int>> m(n, vector<int>(ceil(log2(n)), 0));

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

m[i][0] = i;

}

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

for(int i = 0; i+(1<<j)-1 < n; i++) {

if( a[m[i][j-1]] < a[m[i+(1<<(j-1))][j-1]] )

m[i][j] = m[i][j-1];

else

m[i][j] = m[i+(1<<(j-1))][j-1];

}

}

while(q>0) {

cin >> x >> y;

k = floor(log2(y-x+1));

if(a[m[x][k]] < a[m[y-(1<<k)+1][k]])

cout << a[m[x][k]] << endl;

else

cout << a[m[y-(1<<k)+1][k]] << endl;

q--;

}

return 0;

}--------------------------------------------------------------------------

//merge sort

#include <bits/stdc++.h>

using namespace std;

void merge(int a[],int start,int mid,int end);//the array is zero indexed

void mergeSort(int a[],int start,int end);

int main()

{

int n;

cin>>n;

int a[n];

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

{

cin>>a[i];

}

mergeSort(a,0,n-1);

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

{

cout<<a[i]<<" ";

}

cout<<endl;

return 0;

}

void merge(int a[],int start,int mid,int end)

{

int p=start;int q=mid+1;

int arr[end-start+1];

int k=0;

for(int i=start;i<=end;i++)

{

if(p>mid)

{

arr[k++]=a[q++];

}

else if(q>end)

{

arr[k++]=a[p++];

}

else if(a[p]<a[q])

{

arr[k++]=a[p++];

}

else

{

arr[k++]=a[q++];

}

}

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

{

a[start++]=arr[i];

}

}

void mergeSort(int a[],int start,int end)

{

if(start<end)

{

int mid=(start+end)/2;

mergeSort(a,start,mid);

mergeSort(a,mid+1,end);

merge(a,start,mid,end);

}

}---------------------------------------------------------------------------

//quick sort

#include <bits/stdc++.h>

using namespace std;

int partition1(int a[], int start, int last);//the array is zero index

int check(int a[], int start, int last);

void quicksort(int a[], int start, int last);

int main()

{

int n;

cin>>n;

int a[n];

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

{

cin>>a[i];

}

quicksort(a, 0, n-1);

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

{

cout<<a[i]<<" ";

}

cout<<endl;

}

int partition1(int a[], int start, int last)

{

int piv=a[start];

int i=start+1;

for(int j=start+1;j<=last;j++)

{

if(a[j]<piv)

{

int temp=a[i];

a[i]=a[j];

a[j]=temp;

i+=1;

}

}

int t=a[start];

a[start]=a[i-1];

a[i-1]=t;

//cout<<i-1<<endl;

return i-1;

}

int check(int a[], int start, int last)

{

for(int i=start;i<last;i++)

{

if(a[i]>a[i+1])

{

return 0;

}

}

return 1;

}

void quicksort(int a[], int start, int last)

{

if(start<last)

{

int piv=partition1(a, start, last);

quicksort(a, start, piv-1);

quicksort(a, piv+1, last);

}

}-----------------------------------------------------------------

//binary search

#include <bits/stdc++.h>

using namespace std;

int binarySearch\_modified(int arr[], int low, int high, int num);//the array is zero index

int binarySearch(int arr[], int low, int high, int num);

int main()

{ //sorted array

int n,q;

cin>>n>>q;

long long int a[n];

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

cin>>a[i];

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

{

long long int x;

cin>>x;

cout<<binary\_search(a, 0, n-1, x)<<endl;

}//number of elements

//element to be searched

//result is -1 if element not present otherwise index of element

}

int binarySearch(int arr[], int low, int high, int num)

{

while (low <= high)

{

int mid = low + (high-low)/2;

// Check if x is present at mid

if (arr[mid] == num)

return mid;

// If x greater, ignore left half

if (arr[m]id < num)

low = mid + 1;

// If x is smaller, ignore right half

else

high = mid - 1;

}

// if we reach here, then element was not present

return -1;

}

int binarySearch\_modified(int arr[], int low, int high, int num)//returns least index in case of multiple occurrence

{

int ans=-1;

while (low <= high)

{

int mid = (low+high)/2;

// Check if x is present at mid

if (arr[mid] == num)

{

ans=mid;

high=mid-1;

}

// If x greater, ignore left half

else if (arr[mid] < num)

low = mid + 1;

// If x is smaller, ignore right half

else

high = mid - 1;

}

// if we reach here, then element was not present

return ans;

}----------------------------------------------------------------

//binary search struct

#include <bits/stdc++.h>

using namespace std;

int binarySearch\_modified(struct data arr[], int low, int high, int num);

struct data

{

int index;//variable w.r.t which search is conducted

string s;

};

int main()

{

int n;

cin>>n;

struct data d[n];

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

{

cin>>d[i].s;

d[i].index = i;

}

int q;

cin>>q;

cout<<d[binarySearch\_modified(d, 0, n-1, q)].s<<endl;

}

int binarySearch\_modified(struct data arr[], int low, int high, int num)//returns least index in case of multiple occurrence

{

int ans=-1;

while (low <= high)

{

int mid = (low+high)/2;

// Check if x is present at mid

if (arr[mid].index == num)

{

ans=mid;

high=mid-1;

}

// If x greater, ignore left half

else if (arr[mid].index < num)

low = mid + 1;

// If x is smaller, ignore right half

else

high = mid - 1;

}

// if we reach here, then element was not present

return ans;

}--------------------------------------------------------------

//fact wrt prime

#include<iostream>

#include<cmath>

using namespace std ;

// This function computes the highest power of prime p that divides n!

int fact\_wrt\_prime(int n, int p) {

int ans = 0 ;

for (int j = p; n/j != 0; j \*= p) {

ans += n/j ;

// cout << "j : " << j << " ans : " << ans << endl ;

}

return ans ;

}

int main () {

int number, prime, answer ;

cin >> number >> prime ;

answer = fact\_wrt\_prime(number, prime) ;

cout << answer ;

return 0 ;

}-------------------------------------------------

//exponent logn

long long int pow(long long int a, long long int b, long long int MOD)

{

long long int x=1,y=a;

while(b > 0)

{

if(b%2 == 1)

{

x=(x\*y);

if(x>MOD) x%=MOD;

}

y = (y\*y);

if(y>MOD) y%=MOD;

b /= 2;

}

return x;

}------------------------------------------------------------------------------

//sieve

#include<iostream>

#include<cmath>

#include<vector>

using namespace std ;

void sieve(int n, vector<int> &primes) {

for (int i = 2; i < n+1; i++) {

while (primes[i] == 0) i++ ;

for (int j = i+primes[i]; j < n+1; j += primes[i]) {

primes[j] = 0 ;

}

}

}

int main () {

int n ;

cin >> n ;

vector<int> primes(n+1, 0) ;

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

primes[i] = i ;

}

sieve(n, primes) ;

for (int i = 0; i < n+1; i++) {

cout << primes[i] << endl ;

}

return 0 ;

}----------------------------------------------------

//topo sort A C++ program to print topological sorting of a DAG

#include<iostream>

#include <list>

#include <stack>

using namespace std;

// Class to represent a graph

class Graph

{

    int V;    // No. of vertices'

    // Pointer to an array containing adjacency listsList

    list<int> \*adj;

    // A function used by topologicalSort

    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);

public:

    Graph(int V);   // Constructor

     // function to add an edge to graph

    void addEdge(int v, int w);

    // prints a Topological Sort of the complete graph

    void topologicalSort();

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

// A recursive function used by topologicalSort

void Graph::topologicalSortUtil(int v, bool visited[],

                                stack<int> &Stack)

{

    // Mark the current node as visited.

    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            topologicalSortUtil(\*i, visited, Stack);

    // Push current vertex to stack which stores result

    Stack.push(v);

}

// The function to do Topological Sort. It uses recursive

// topologicalSortUtil()

void Graph::topologicalSort()

{

    stack<int> Stack;

    // Mark all the vertices as not visited

    bool \*visited = new bool[V];

    for (int i = 0; i < V; i++)

        visited[i] = false;

    // Call the recursive helper function to store Topological

    // Sort starting from all vertices one by one

    for (int i = 0; i < V; i++)

      if (visited[i] == false)

        topologicalSortUtil(i, visited, Stack);

    // Print contents of stack

    while (Stack.empty() == false)

    {

        cout << Stack.top() << " ";

        Stack.pop();

    }

}

// Driver program to test above functions

int main()

{

    // Create a graph given in the above diagram

    Graph g(6);

    g.addEdge(5, 2);

    g.addEdge(5, 0);

    g.addEdge(4, 0);

    g.addEdge(4, 1);

    g.addEdge(2, 3);

    g.addEdge(3, 1);

    cout << "Following is a Topological Sort of the given graph \n";

    g.topologicalSort();

    return 0;

}-------------------------------------------------------------------------------------