Bugged Coders

1 templates

1.1 Template

```
#include <bits/stdc++.h>
using namespace std;
#define io ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define rep(i,a,b) for(int i=a;i<b;i++)
#define endl "\n"
#define pb push_back
#define each(i,x) for(auto &i:x)
#define deb(x) cout<<#x<<" "<<x<<endl;
#define ll long long
```

2 Data sctrucutres

2.1 Segment tree

```
int nums[]={1,3,4,5,7};
struct segmentTree{
    int l, r,sum;
    segmentTree *nodeLeft,*nodeRight;
    segmentTree(int a, int b){
        l=a;
        r=b;
        int m=(l+r)/2;
        if(l!=r){
            nodeLeft=new segmentTree(l,m);
            nodeRight=new segmentTree(m+1,r);
            sum=nodeLeft->sum+nodeRight->sum;
        }
        else sum=nums[l];
    }
    int query(int a, int b){
        if(b<l || a>r) return 0;
        if(a<=l && r<=b) return sum;
        return nodeLeft->query(a,b)+nodeRight->query(a,b);
    }
    void update(int pos, int v){
        if(l!=r){
            int m=(l+r)/2;
            if(pos<=m) nodeLeft->update(pos,v);
            else nodeRight->update(pos,v);
            sum=nodeLeft->sum+nodeRight->sum;
        }
        else sum=v;
    }
};
```

2.2 Segment tree iteractive

```
#include <bits/stdc++.h>
using namespace std;
#define io ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define rep(i,a,b) for(int i=a;i<b;i++)
#define endl "\n"
#define pb push_back
#define each(i,x) for(auto &i:x)
#define deb(x) cout<<#x<<" "<<x<<endl;
#define ll long long
```

2.3 Segment tree- Lazzy Propagation

```
int nums[]={1,3,5,7,9,11};
struct segmentTree{
    int l, r,sum,lazy;
    segmentTree *nodeLeft,*nodeRight;
    segmentTree(int a, int b){
        l=a;
        r=b;
        int m=(l+r)/2;
        lazy=0;
        if(l!=r){
            nodeLeft=new segmentTree(l,m);
            nodeRight=new segmentTree(m+1,r);
            sum=nodeLeft->sum+nodeRight->sum;
        }
        else sum=nums[l];
    }
    int query(int a, int b){
        if(nodeLeft!=nullptr && lazy!=0) nodeLeft->lazy=lazy;
        if(nodeRight!=nullptr && lazy!=0) nodeRight->lazy=lazy;
        sum+=(r-l+1)*lazy;lazy=0;
        if(b<l || a>r) return 0;
        if(a<=l && r<=b) return sum;
        return nodeLeft->query(a,b)+nodeRight->query(a,b);
    }
    int update(int a, int b, int v){
        int increment=0;
        if(b<l || a>r) return 0;
        if(a<=l && r<=b){
            if(nodeLeft!=nullptr) nodeLeft->lazy+=lazy;
            if(nodeRight!=nullptr) nodeRight->lazy+=lazy;
            increment=(r-l+1)*v;
            sum+=increment;
            return increment;
        }
        increment=nodeLeft->update(a,b,v)+nodeRight->update(a,b,v);
        sum+=increment;
        return increment;
    }
};
```

2.4 Segment tree Lazy Iteractive

```
#include <bits/stdc++.h>
using namespace std;
#define io ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl "\n"
#define pb push_back
#define each(i,x) for(auto &i:x)
#define all(x) x.begin(),x.end()
#define sz(x) (int)x.size()
#define rep(i,a,b) for(int i=a;i<b;i++)
#define ll long long

const int N=2e5+10;
ll st[4*N+10],lazy[4*N+10],arr[N];
void build(int l, int r, int i){
    lazy[i]=0;
    if(l==r){st[i]=arr[l];return;}
    int m=(l+r)>>1;
    build(l,m,2*i+1);
    build(m+1,r,2*i+2);
    st[i]=st[2*i+1]+st[2*i+2];
}
void push(int l, int r, int i){
    if(!lazy[i])return;
    st[i]+=(r-l+1)*lazy[i];
    if(l!=r){
        lazy[2*i+1]+=lazy[i];
        lazy[2*i+2]+=lazy[i];
    }
    lazy[i]=0;
}
void update(int l, int r, int a, int b, ll x, int i){
    push(l,r,i);
    if(a>r||b<l)return;
    if(a<=l&&r<=b){
        lazy[i]+=x;
        push(l,r,i);
        return;
    }
    int m=(l+r)>>1;
    update(l,m,a,b,x,2*i+1);update(m+1,r,a,b,x,2*i+2);
    st[i]=st[2*i+1]+st[2*i+2];
}
ll query(int l, int r, int a, int b, int i){
    if(a>r||b<l)return 0;
    push(l,r,i);
    if(a<=l&&r<=b) return st[i];
    int m=(l+r)>>1;
    return query(l,m,a,b,2*i+1)+query(m+1,r,a,b,2*i+2);
}
int main(){io
    ll n,q;cin>>n>>q;
    rep(i,0,n)cin>>arr[i];
    build(0,n-1,0);
    rep(i,0,q){
        int op;cin>>op;
        if(op==1){
            int a,b; ll x;
            cin>>a>>b>>x;a--;b--;
            update(0,n-1,a,b,x,0);
        }
        else{
            int k;cin>>k;k--;
            cout<<query(0,n-1,k,k,0)<<endl;
        }
    }
    return 0;
}
```

2.5 Disjoin Set

```
//Se usa para detectar cyclos en un grafo no dirigido convexo & en el algoritmo de Krustal.
vector<pair<int,int>>ds;
void init(int n){
    ds.assign(n+1,{-1,0});
}
int find(int x){
    if(-1==ds[x].first) return x;
    return ds[x].first=find(ds[x].first);
}
bool unionDs(int x, int y){
    int px=find(x),py=find(y);
    int &rx=ds[px].second,&ry=ds[py].second;
    if(px==py) return false;
    else{
        if(rx>ry) ds[py].first=px;
        else{
            ds[px].first=py;
            if(rx==ry) ry+=1;
        }
    }
    return true;
}
```

2.6 Sparce Table

```
//Se usa para RMQ porque se puede hacer en O(1), no acepta updates
vector<int>lg;
vector<vector<int>>st;
int *nums;
void init(int n){
    int logn=(int) log2(n)+1;
    lg.assign(n+1,0);
    st.assign(logn,vector<int>(n+1));
    for(int i=0;i<n;i++) st[0][i]=nums[i];
    lg[1]=0;
    for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
    for(int i=1;i<logn;i++)
        for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j+(1<<(i-1))]);
}
int query(int a,int b){
    int logn=lg[(b-a+1)];
    cout<<st[logn][a]<<endl;
    return min(st[logn][a],st[logn][b-(1<<logn)+1]);
}
```

2.7 Treap

```
#include <bits/stdc++.h>
using namespace std;
typedef struct Node *pitem;
struct Node{
    int value,key;
    pitem l,r;
    Node(int v) value(v),key(rand()),l(nullptr),r(nullptr);
};
struct treap
{
void split(pitem t, int value, pitem& left,pitem& right){
    if(!t) void(left=right=nullptr);
    if(t->value<=x) split(t->r,value,t->r,right),left=t;
    else split(t->l,value,left,t->l),right=t;
}
void marge(pitem t,pitem left,pitem right){
    if(!left || ! right){t=left?left:right;return;}
    if(left->key>right->key) marge(left->r,left->r,right), t=left;
    else marge(right->l,left,right->l), t=right;
}
void insert(pitem &t,pitem x){
    if(!t)t=x;
    else if(x->key>t->key){
        split(t,x->value,x->l,x->right), t=x;
    }
    else insert(x->value<t->value? t->l:t->r,x);
}
};
```

2.8 Trie

```
struct trie{
    bool isFinal;
    trie *children[26];
    trie(){
        isFinal=false;
        for(int i=0;i<26;i++)children[i]=nullptr;
    }
};

void inserString(string str,trie *root){
    trie *aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-’a’;
        if(aux->children[index]==nullptr){
            aux->children[index]=new trie();
        }
        aux=aux->children[index];
    }
    aux->isFinal=true;
}
bool existInTrie(string str,trie *root){
    trie *aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-’a’;
        if(aux->children[index]==nullptr) return false;
        aux=aux->children[index];
    }
    return aux->isFinal;
}
```

2.9 Cartesian Tree

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
struct node {
  int idx, val, par, ch[2];
  friend bool operator<(node a, node b) { return a.idx < b.idx; }
  void init(int _idx, int _val, int _par) {
    idx = _idx, val = _val, par = _par, ch[0] = ch[1] = 0;
  }
} tree[N];
int root, top, stk[N];
int cartesian_build(int n) {
  for (int i = 1; i <= n; i++) {
    int k = i - 1;
    while (tree[k].val > tree[i].val) k = tree[k].par;
    tree[i].ch[0] = tree[k].ch[1];
    tree[k].ch[1] = i;
    tree[i].par = k;
    tree[tree[i].ch[0]].par = i;
  }
  return tree[0].ch[1];
}
int dfs(int x) {
  if (!x) return 0;
  int sz = dfs(tree[x].ch[0]);
  sz += dfs(tree[x].ch[1]);
  ans = max(ans, (ll)(sz + 1) * tree[x].val);
  return sz + 1;
}
```

2.10 BIT

```
struct FenwickTree {
    vector<int> bit;  // binary indexed tree
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }

    FenwickTree(vector<int> const &a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

3 DP

3.1 Digit DP

```
ll dp[20][20][3];
ll n,k,d;
vector<int>num;
ll bk(int i, int len, int t){
    if(len>k) return 0;
    if(i==n){
        if(len==k) return 1;
        return 0;
    }
    ll &res=dp[i][len][t];
    if(res!=-1) return res;
    res=0;
    int tope;
    if(t==0) tope=num[i];
    else tope=9;
    for(int j=0;j<=tope;j++){
        int newt=t;
        int newlen=len;
        if(t==0 && j<tope) newt=1;
        if(d==j) newlen++;
        if(newlen<=k)res+=bk(i+1,newlen,newt);
    }
    return res;
}
ll rep(int a){
    num.clear();
    while(a>0){
        num.push_back(a%10);
        a/=10;
    }
    reverse(num.begin(),num.end());
    n=num.size();
    memset(dp,-1,sizeof(dp));
    return bk(0,0,0);
}
```

3.2 Prefix Sum 2D

```
const int MAX=50
ll prefix[MAX+4][MAX+4];
// x1-> left x2->right y1-> up  y2 ->down x1<=x2 && y1<=y2
ll query(int x1, int x2,int y1,int y2){
    return prefix[y2][x2]-prefix[y1-1][x2]-prefix[y2][x1-1]+prefix[y1-1][x1-1];
}
//Inizialisate prefix[i][j] with original values of the grid
void prefixSum(){
        for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++) prefix[i][j]+=prefix[i][j-1]+prefix[i-1][j]-prefix[i-1][j-1];
    }
}
```

4 Graph

4.1 Krustal

```
// Este algoritmo sirve para buscar MST de un grafo convexo no dirigido
vector<tuple<int,int,int>>edges;
int n;m;
//Insertar Disjoin set
int krustal(){
    sort(edges.begin(),edges.end());
    int res=0;
    for(int i=0;i<m;i++){
        int c,a,b;
        tie(c,a,b)=edges[i];
        // Si en el disjoin set estan conectados retorna false
        if(unionDs(a,b)==false) continue;
        else res+=c;
    }
    return res;
}
```

4.2 Kosaraju's (SCC)

```
//Sirve para encontrar los SCC
struct Kosaraju{
    int s;
    vector<vector<int>> g,gr;
    vector<int> visited,ids,topologic_sort;
    Kosaraju(int n){
        s=n;
        g.assign(n+1,vector<int>());
        gr.assign(n+1,vector<int>());
        visited.assign(n+1,0);
        ids.assign(n+1,0);
    }
    void addEdge(int a,int b){
        g[a].push_back(b);
        gr[b].push_back(a);
    }
    void dfs(int u){
        if(visited[u]!=0) return;
        visited[u]=1;
        for(int node:g[u])dfs(node);
        topologic_sort.push_back(u);
    }
    void dfsr(int u,int id){
        if(visited[u]!=0) return;
        visited[u]=1;
        ids[u]=id;
        for(int node:gr[u])dfsr(node,id);
    }
    void algo(){
        for(int i=1;i<=s;i++) if(visited[i]==0) dfs(i);
        fill(visited.begin(),visited.end(),0);
        reverse(topologic_sort.begin(),topologic_sort.end());
        int id=0;
        for(int i=0;i<topologic_sort.size();i++){
            if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id++);
        }
    }// Es el ago principal
    int search(int node){
        return ids[node];
    }// Retorana el componente que esta el nodo
};
```

4.3 2 Sat

```
//Se usa para los problams en los cuales tengamos dos dosible variables
struct twoSat{
    int s;
    vector<vector<int>> g,gr;
    vector<int> visited,ids,topologic_sort,val;
    twoSat(int n){
        s=n;
        g.assign(n*2+1,vector<int>());
        gr.assign(n*2+1,vector<int>());
        visited.assign(n*2+1,0);
        ids.assign(n*2+1,0);
        val.assign(n+1,0);
    }
    void addEdge(int a,int b){
        g[a].push_back(b);
        gr[b].push_back(a);
    }
    void addOr(int a,bool ba,int b,bool bb){
        addEdge(a+(ba?s:0),b+(bb?0:s));
        addEdge(b+(bb?s:0),a+(ba?0:s));
    }
    void addXor(int a,bool ba,int b,bool bb){
        addOr(a,ba,b,bb);
        addOr(a,!ba,b,!bb);
    }
    void addAnd(int a,bool ba,int b,bool bb){
        addXor(a,!ba,b,bb);
    }
    void dfs(int u){
        if(visited[u]!=0) return;
        visited[u]=1;
        for(int node:g[u])dfs(node);
        topologic_sort.push_back(u);
    }
    void dfsr(int u,int id){
        if(visited[u]!=0) return;
        visited[u]=1;
        ids[u]=id;
        for(int node:gr[u])dfsr(node,id);
    }
    bool algo(){
        for(int i=0;i<s*2;i++) if(visited[i]==0) dfs(i);
        fill(visited.begin(),visited.end(),0);
        reverse(topologic_sort.begin(),topologic_sort.end());
        int id=0;
        for(int i=0;i<topologic_sort.size();i++){
            if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id++);
        }
        for(int i=0;i<s;i++){
            if(ids[i]==ids[i+s]) return false;
            val[i]=(ids[i]>ids[i+s]?0:1);
        }
        return true;
    }
};
```

5 Strings

5.1 KMP

```
vector<int> kmp(string s){
    int n=s.size();
    vector<int>pi(n);
    for(int i=1;i<n;i++){
        int j=pi[i-1];
        while(j>0 && s[i]!=s[j])j=pi[j-1];
        if(s[i]==s[j]) j++;
        pi[i]=j;
    }
    return pi;
}
```

5.2 Hashing

```
struct Hash{
  const int mod=1e9+123;
  const int p=257;
  vector<int> prefix;
  static vector<int>pow;
  Hash(string str){
    int n=str.size();
    while(pow.size()<=n){
      pow.push_back(1LL*pow.back()*p%mod);
    }
    vector<int> aux(n+1);
    prefix=aux;
    for(int i=0;i<n;i++){
      prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
    }
  }
  inline int getHashInInerval(int i,int len,int MxPow){
    int hashing=prefix[i+len]-prefix[i];
    if(hashing<0) hashing+=mod;
    hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
    return hashing;
  }
};
vector<int> Hash::pow{1};
```

6 Math

6.1 Linear Sieve

```
//O(N) for find all the primes in the given range
bool is_compositive[10000000+1];
vector<int>primes;
void sieve(int n){
    primes.clear();
   fill(is_compositive,is_compositive+n,false);
   for(int i=2;i<=n;i++){
    if(!is_compositive[i]) primes.push_back(i);
    for(int j=0;j<primes.size() && primes[j]*i<=n;j++){
        is_compositive[i*primes[j]]=true;
        if(!(i%primes[j])) break;
    }
   }
}

    int n;cin>>n;
    sieve(n);
    cout<<primes.size()<<endl;
    for(int i=0;i<primes.size();i++){
        cout<<primes[i]<<endl;
    }
}
```

6.2 Euler Sieve

```
//this is a sieve for a euler funciton that given the number of coprime numbers of x but in a range
vector<int>sieve;
void eulerSieve(int n){
    sieve.clear();
    sieve.push_back(0);
    for(int i=1;i<=n;i++){
        sieve.push_back(i);
    }
    for(int i=2;i<=n;i++){
        if(sieve[i]==i)
            for(int j=i;j<=n;j+=i)sieve[j]-=(sieve[j]/i);
    }
}
```

6.3 Euler Sieve Gauss Reduction

```
// sum(pi(n)) of the divisors of n is equal to n
vector<int>sieve;
void eulerSieve(int n){
    sieve.clear();
    sieve.push_back(0);
    sieve.push_back(1);
    for(int i=2;i<=n;i++){
        sieve.push_back(i-1);
    }
    for(int i=2;i<=n;i++){
            for(int j=i*2;j<=n;j+=i)sieve[j]-=sieve[i];
    }
}
```

6.4 Mobius Sieve

```
/* f(x)=0 if has square prime factor
f(x)=1 if if is square-free and even
f(x)=-1 if is square-free and odd
properti the sum of function of divisors of x is equl to 0 if x>1*/
vector<int>sieve;
void ms(int n){
    sieve.assign(n+1,-1);
    sieve[1]=1;
    for(int i=2;i<=n;i++)
       for(int j=i*2;j<=n;j+=i)sieve[j]-=sieve[i];
}
```

6.5 Binary Exponentation

```
long long binPow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

7 Flows

7.1 Dinics

```
struct dinics{
    int m,n;
    ll mF=1e18;
    vector<tuple<int,ll,ll>>edge;
    vector<vector<int>>adj;
    vector<int>level,id;
    void init(int _n){
        m=0;
        n=_n;
        level.resize(n+1);
        id.resize(n+1);
        adj.resize(n+1);
    }
    void addEdge(int u,int v,ll f,bool directed=true){
        edge.push_back({v,f,0});
        adj[u].push_back(m);
        edge.push_back({u,(directed?0:f),0});
        adj[v].push_back(m+1);
        m+=2;
    }
    bool bfs(int s, int t){
        fill(level.begin(),level.end(),-1);
        queue<int>aux;
        aux.push(s);
        level[s]=0;
        while(!aux.empty()){
            int v=aux.front();aux.pop();
            for(auto idx:adj[v]){
                auto &[u,c,f]=edge[idx];
                if(c-f<1 || level[u]!=-1) continue;
                aux.push(u);
                level[u]=level[v]+1;
            }
        }
        return level[t]!=-1?1:0;
    }
    ll dfs(int u,int t, ll f){
        if(u==t || f==0) return f;
        for(auto &cdx=id[u];cdx<adj[u].size();cdx++){
            int idx=adj[u][cdx];
            auto &[v,c,fv]=edge[idx];
            if(level[v]!=level[u]+1 || c-fv<1) continue;
            ll res=dfs(v,t,min(f,c-fv));
            if(!(res)) continue;
            auto &fr=get<2>(edge[idx^1]);
            fv+=res;
            fr-=res;
            return res;
        }
        return 0;
    }
    ll maxFlow(int s,int t){
        ll mf=0;
        while(bfs(s,t)){
            fill(id.begin(),id.end(),0);
            while(ll f=dfs(s,t,mF)) mf+=f;
        }
        return mf;
    }
};
```

8 Tree

8.1 Binary-Lifting

```
//For get the k-th atecesor of a node in a tree 1 indexed
vector<int> *T;
vector<vector<int>>up;
vector<int>deep;
int lg;
void init(int n){
    lg=ceil(log2(n))+1;
    T=new vector<int>[n+1];
    up.assign(n+1,vector<int>(lg+1,1));
    deep.assign(n+1,0);
}
void dfs(int node){
    for(auto ch:T[node]){
        deep[ch]=deep[node]+1;
        up[ch][0]=node;
        for(int i=1;i<lg;i++){
            up[ch][i]=up[up[ch][i-1]][i-1];
        }
        dfs(ch);
    }
}
int getkthAtecesor(int node, int k){
    int res=node;
    for(int i=lg-1;i>=0;i--){
        if(k & (1<<i)) res=up[res][i];
    }
    return res;
}
```

8.2 Euler Tour

```
#include <bits/stdc++.h>
using namespace std;
const int MAX=2e5+300;
int S[MAX];
int F[MAX];
int FT[MAX];
vector<int>T[MAX];
//Inicalizar en 0 para 0 indexado y 1 par 1 indexado
int timer;
int n;
void dfs(int node,int par){
    S[node]=timer;
    FT[timer]=node;
    timer++;
    for(auto i:T[node])
       if(i!=par) dfs(i,node);
    F[node]=timer;
    FT[timer]=node;
    timer++;
}
```

9 Geometry

10 Others

10.1 Mo's algorithm

```
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

10.2 Matrix

```
const int N=100, MOD=1e9+7;
struct Matrix {
  ll a[N][N];
  Matrix() {memset(a,0,sizeof(a));}
  Matrix operator *(Matrix other) {  // Product of a matrix
    Matrix product=Matrix();
        rep(i,0,N) rep(j,0,N) rep(k,0,N) {
            product.a[i][k]+=a[i][j]*other.a[j][k];
            product.a[i][k]%=MOD;
        }
    return product;
  }
};
Matrix expo_power(Matrix a, ll n) {  // Matrix exponentiation
  Matrix res=Matrix();
    rep(i,0,N) res.a[i][i]=1;  // Matriz identidad
  while(n){
        if(n&1) res=res*a;
        n>>=1;
        a=a*a;
  }
  return res;
} // Ej. Matrix M=Matrix();  M.a[0][0]=1;  M=M*M;   Matrix res=
    expo_power(M,k);
```