*//\*Bismillahir Rahmanir Raheem*

*//! BlackBeard*

*#include* <bits/stdc++.h>

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

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

*#define* NFS ios::sync\_with\_stdio(false), cin.tie(0), cout.tie(0);

using namespace \_\_gnu\_pbds;

using namespace std;

template <typename DT>

using oset = tree<DT, null\_type, less<DT>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

template <typename DT, typename FUNC>

using o\_set = tree<DT, null\_type, FUNC, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

template <typename DT1, typename DT2>

using omap = tree<DT1, DT2, less<DT1>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

template <typename DT1, typename DT2, typename FUNC>

using o\_map = tree<DT1, DT2, FUNC, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

*/\*\**

*\* - less<data\_type>            --> Increasingly sorted set*

*\* - less\_equal<data\_type>      --> Increasingly sorted multiset*

*\* - greater<data\_type>         --> Decreasingly sorted set*

*\* - greated\_equal<data\_type>   --> Decreasingly sorted multiset*

*\*\*/*

*//!------*

*#define* YES cout << "YES" << endl

*#define* Yes cout << "Yes" << endl

*#define* yes cout << "yes" << endl

*#define* No cout << "No" << endl

*#define* NO cout << "NO" << endl

*#define* no cout << "no" << endl

*//!-----*

*#define* ll long long

*#define* int ll

*#define* lld long double

*#define* ull unsigned long long

*#define* pii pair<int, int>

*#define* pll pair<ll, ll>

*#define* vi vector<int>

*#define* vl vector<ll>

*#define* vii vector<pii>

*#define* vll vector<pll>

*#define* pb push\_back

*#define* ff first

*#define* ss second

*//!--------*

*#define* PQiimn priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>

*#define* PQllmn priority\_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<pair<ll, ll>>>

*#define* PQiimx priority\_queue<pair<int, int>>

*#define* PQllmx priority\_queue<pair<ll, ll>>

*#define* PQ priority\_queue<int>

*//!--------*

*#define* FILL(*x*, *y*) memset(x, y, sizeof(x))

*#define* clr(*x*, *y*) memset(x, 0, sizeof(x))

*#define* all(*x*) (x).begin(), (x).end()

*#define* rall(*x*) (x).rbegin(), (x).rend()

*#define* fore(*arr*) *for* (auto &x : (arr))

*#define* max3(*a*, *b*, *c*) max(a, max(b, c))

*#define* min3(*a*, *b*, *c*) min(a, min(b, c))

*#define* gcd(*x*, *y*) \_\_gcd(x, y)

*#define* lcm(*x*, *y*) x \*(y / gcd(x, y))

*#define* preci(*x*) fixed << setprecision(x)

*#define* PI (acos(-1.0))

*#define* SZ(*x*) (int)x.size()

*#define* EPS (1e-9)

*#define* bug cout << "\*\_\*\n"

*//!-------*

const ll INF = 1e18 + 5;

const ll MOD = 1e9 + 7;

const ll N = 100005;

*//!---------*

template <class T>

inline void print(vector<T> *v*) { fore(v) cout << x << ' '; }

template <class T>

inline void print(set<T> *v*) { fore(v) cout << x << ' '; }

template <class T>

inline void print(multiset<T> *v*) { fore(v) cout << x << ' '; }

template <class T, class U>

inline void print(map<T, U> *v*) { fore(v) cout << x.ff << ' ' << x.ss << endl; }

template <class T, class U>

inline void print(multimap<T, U> *v*) { fore(v) cout << x.ff << ' ' << x.ss << endl; }

template <class T>

inline void print(T *u*) { cout << '\*' << *u* << '\*' << endl; }

*//!---------*

int cs = 1;

inline void CASE() { cout << "Case " << cs++ << ":"; }

inline ll ciel(double *a*, double *b*) { *return* (*a* + (*b* - 1)) / *b*; }

inline int numOfDigit(ll *n*) { *return* log10(*n*) + 1; }

inline int bitsInBinary(ll *n*) { *return* log2(*n*) + 1; }

**Double Compare & Negative zero**

bool equalTo(double a, double b) { return ((fabs(a - b) <= EPS) ? true : false); }

bool notEqual(double a, double b) { return ((fabs(a - b) > EPS) ? true : false); }

bool lessThan(double a, double b) { return ((a + EPS < b) ? true : false); }

bool lessThanEqual(double a, double b) { return ((a < b + EPS) ? true : false); }

bool greaterThan(double a, double b) { return ((a > b + EPS) ? true : false); }

bool greaterThanEqual(double a, double b) { return ((a + EPS > b) ? true : false); }

bool negzero(double a) { return (a < EPS) ? true : false; }

**Ciel:**

inline ll ciel(double a, double b) { return (a + (b - 1)) / b; }

**Prime**:

bool is\_prime(ll n) //! O(sqrt(n))

{

    if (n <= 1) return false;

    for (ll i = 2; i \* i <= n; i++)

        if (n % i == 0) return false;

    return true;

}

//\* checking with all prime till sqrt(n)

bool ISPRIME(ll n) //! O(sqrt(n)/log(sqrt(n)))

{

    if (n <= 1) return 0;

    int idx = 0;

    int i = Primes[idx];

    while (i \* i <= n)

    {

        if (n % i == 0) return false;

        i = Primes[++idx];

    }

    return true;

}

**Seive Method**: Storing All the primes <= N

vector<int> Primes;

bool isPrime[10000001]; // default value 0

void sieve(int n)       //! O(n\*log(n))

{

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

        if (isPrime[i] == false)

            for (int j = i \* i; j <= n; j += i) isPrime[j] = true;

    Primes.pb(2);

    for (int i = 3; i <= n; i += 2)

        if (isPrime[i] == false)

            Primes.pb(i);

}

**High prime & Low Prime Using sieve method**:

vector<int> LowPrime(1000001, 0), HighPrime(1000001, 0);

void sieveLH(int n) //! High Prime Low Prime (n\*log(n))

{

    for (int i = 2; i \* i <= n; i++)

    {

        if (isPrime[i] == false)

        {

            LowPrime[i] = HighPrime[i] = i;

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

            {

                isPrime[j] = true;

                HighPrime[j] = i;

                if (LowPrime[j] == 0) LowPrime[j] = i;

            }

        }

    }

}

**Factorization**:

vector<int> factor(int n)

{

    vector<int> v;

    for (int i = 2; i <= n; i++)

    {

        while (n % i == 0)

            n /= i, v.pb(i);

    }

    if (n > 1) v.pb(n);

    return v;

}

vector<int> factor2(int n)

{

    vector<int> fact;

    for (int i = 0; Primes[i] \* Primes[i] <= n; i++)

    {

        while (n % Primes[i] == 0)

        {

            n /= Primes[i], fact.pb(Primes[i]);

        }

    }

    if (n > 1)fact.pb(n);

    return fact;

}

vector<int> factor(int n) //\* using HighPrime O(log(n))

{

    vector<int> fact;

    while (n > 1)

    {

        int f = HighPrime[n];

        while (n % f == 0)

        {

            fact.pb(f), n /= f;

        }

    }

    return fact;

}

**Divisors**:

vector<int> Divisors(int n) //sqrt(n)

{

    vector<int> v;

    for (int i = 1; i \* i <= n; i++)

    {

        if (n % i == 0)

        {

            v.pb(i);

            if (n / i != i) v.pb(n / i);

        }

    }

    return v;

}

**Storing divisors of n numbers using sieve**:

vector<int> divisors[1000001];

void DivisrosA(int n) //n\*log(n)

{

    for (int i = 2; i <= n; i++)

    {

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

            divisors[j].pb(i);

    }

}

**Binary exponential**:

ll bin\_expo(ll a, ll b) //O(log(b))

{

    ll s = 1;

    while (b > 0)

    {

        if (b % 2 == 1)

            s = ((s % MOD) \* (a % MOD)) % MOD;

        a = ((a % MOD) \* (a % MOD)) % MOD, b = b >> 1;

    }

    return s % MOD;

}

**(ab) compare (ba) => (a\*log b) compare (b\*log a)**

**Moduler Arithmetic**:

inline ll mod(ll a) { return (a % MOD + MOD) % MOD; }

inline ll modAdd(ll a, ll b) { return mod(mod(a) + mod(b)); }

inline ll modSub(ll a, ll b) { return mod(mod(a) - mod(b)); }

inline ll modMul(ll a, ll b) { return mod(mod(a) \* mod(b)); }

inline ll modInv(ll a) { return bin\_expo(a, MOD - 2); }

**Permutation & Combination**:

nPr = n!/(n-r)! [order fact] w/o repetation

nPr = nr [order fact] with repetation

nCr = n!/((n-r)!\*r!) [order doesn’t fact] w/o repetation

nCr = (n+r-1)Cr [order doesn’t fact] with repetation

nCr = nC(n-r)

nCr = n-1Cr + n-1Cr-1

ll fact[21];

void factorial()

{

    fact[0] = 1;

    for (int i = 1; i <= 20; i++)

        fact[i] = i \* fact[i - 1];

}

inline ll nPr(int n, int r) { return fact[n] / fact[n - r]; }               // max 20!

inline ll nCr(int n, int r) { return ((fact[n] / fact[n - r]) / fact[r]); } // max 20!

**Combination code for larger values**:

ll nCr2(int n, int r)// for larger values

{

    ll p = 1, k = 1;

    r = min(r, n - r);

    while (r)

    {

        p \*= n, k \*= r;

        ll m = gcd(p, k);

        p /= m, k /= m;

        n--, r--;

    }

    return p;

}

**nCr = ((max+1)\* (max+2)\* (max+3)\*… … …\*n ) / (2\*3\*… … …\*min) [ max=max(r,n-r) ; min=min(r,n-r) ]**

**nPr = (r+1)\* (r+2)\* (r+3)\*… … …\*n**

**sort compare**:

bool comp(pair<int, int> A, pair<int, int> B)

{

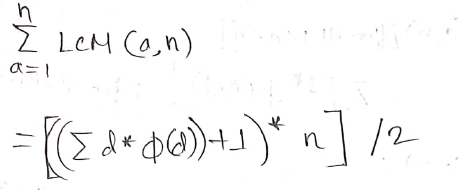
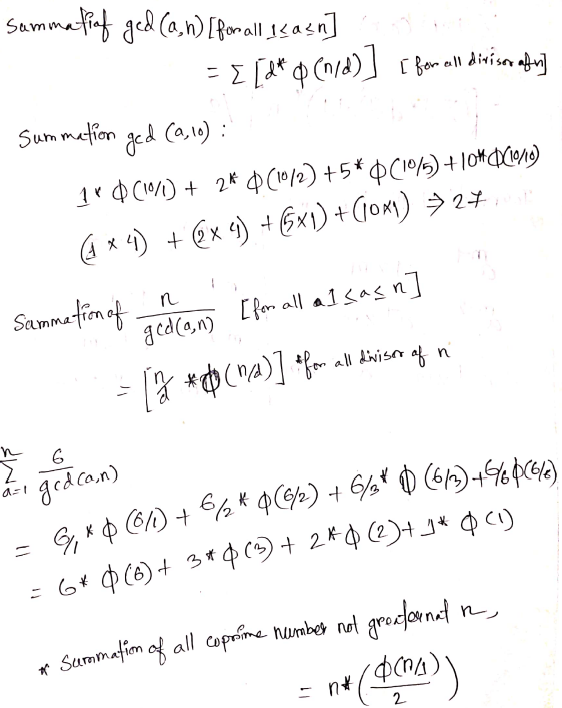
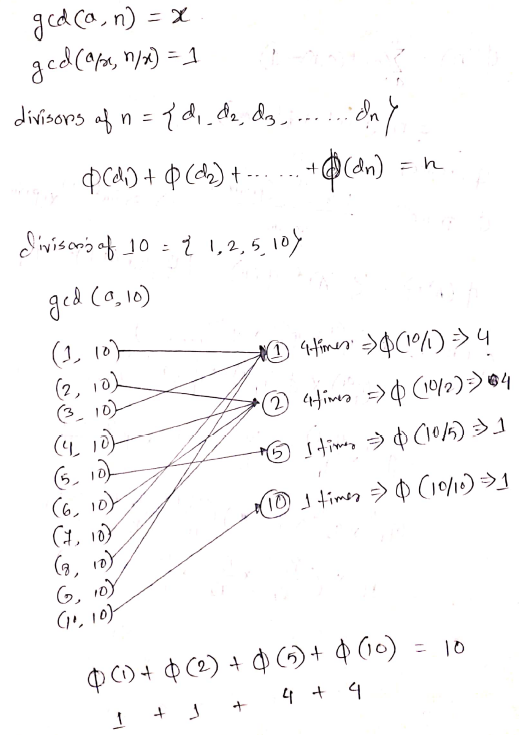
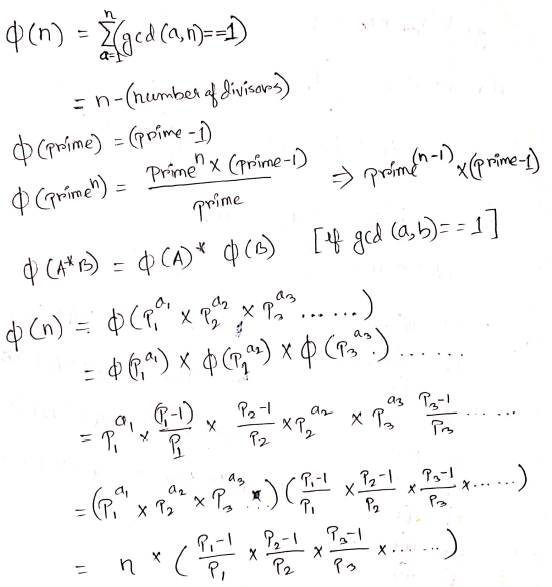
    if (A.first == B.first)

        return (A.second > B.second); //! if A.second is Larger ihen A will come before B

    return (A.first < B.first);       //! if A.first is smaller then A will come before B

}

**Euler Totient Function**:



ll ETF(int n) //! o(sqrt(n))

{

    ll total = n;

    for (int i = 2; i \* i <= n; i++)

    {

        if (n % i == 0)

        {

            total = (total / i) \* (i - 1);

            while (n % i == 0) n /= i;

        }

    }

    if (n > 1) total = (total / n) \* (n - 1);

    return total;

}

**EFT Seive:**

int phi[1000001];

bool visited[1000001];

void ETFsieve(int n) // O(n\*log(n))

{

    for (int i = 1; i <= n; i++)

        phi[i] = i;

    for (int i = 2; i <= n; i++)

    {

        if (!visited[i])

        {

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

            {

                visited[j] = true;

                phi[j] = (phi[j] / i) \* (i - 1);

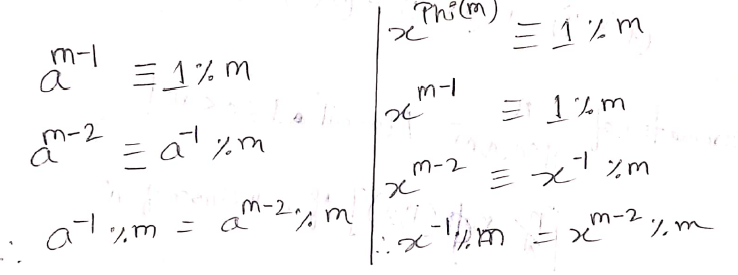
            }

        }

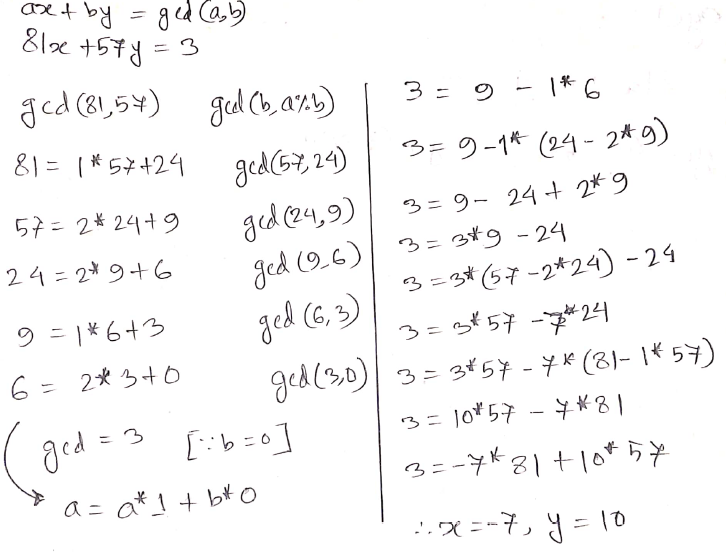
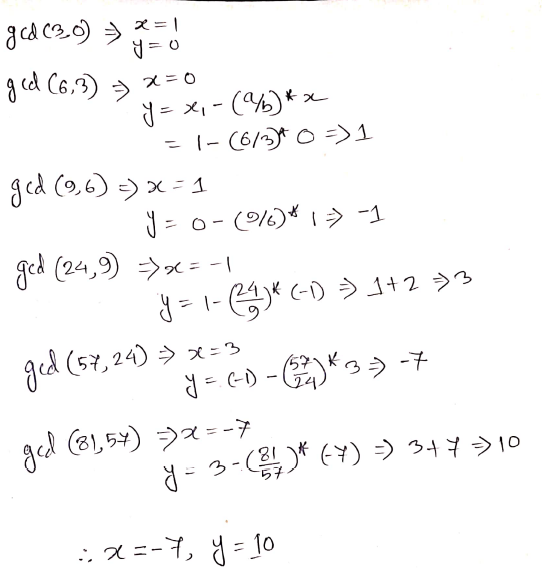
    }

}

**Fermat’s little Theorem**:



**Egcd**:

ll egcd(int a, int b, int &x, int &y)

{

    if (b == 0)

    {

        x = 1, y = 0;

        return a;

    }

    int x1, y1;

    int d = egcd(b, a % b, x1, y1);

    x = y1;

    y = x1 - (a / b) \* y1;

    return d;

}

**Kadanes Algorithm**: (max subarray)

// check for every index maxtillnow with current index ans compare it with max\_so\_far

int KadanesAlgo(vector<int> &v) // maxSubArray //O(n)

{

    int mxsofar = INT\_MIN, mxendthis\_index = 0;

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

    {

        mxendthis\_index += v[i];

        mxendthis\_index = max(mxendthis\_index, v[i]);

        mxsofar = max(mxsofar, mxendthis\_index);

    }

    return mxsofar;

}

**Sliding Window**: (give max sum of k consecutive elements)

int slidingWindow(vector<int> &v, int k) //O(N)

{

    int sum = 0;

    for (int i = 0; i < k; i++)

        sum += v[i];

    int ans = sum;

    for (int i = 0; i < (int)v.size() - k; i++)

    {

        sum -= v[i];

        sum += v[i + k];

        ans = max(ans, sum);

    }

    return ans;

}

**Two pointer**: check if sum of three element’s of Array equals to target

bool ThreeSum(vector<int> &v, int target)

{

    sort(all(v));

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

    {

        int lo = i + 1, hi = (int)v.size() - 1;

        while (lo < hi)

        {

            int cur = v[i] + v[lo] + v[hi];

            if (cur == target)

            {

                cout << v[i] << ' ' << v[lo] << ' ' << v[hi] << endl;

                return true;

            }

            else if (cur < target)

                lo++;

            else

                hi--;

        }

    }

    return false;

}

**Meet in the Middle**: **O(2n)**

You are given an array of n numbers. In how many ways can you choose a subset of the numbers with sum x?

1. get all subset sum of first half and second half separately (using bitmask)then sort the sum's of second half

2. now for each element a in first half sum's check how mane of (x-a)  is present in second half sum's

void sum(vector<int> &v, int l, int r, vector<int> &L, vector<int> &R)

{

    int n = r - l;

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

    {

        int s = 0;

        for (int j = 0; j < n; j++)

        {

            if (i & (1 << j))

                s += v[l + j];

        }

        if (l == 0 )L.pb(s);

        else R.pb(s);

    }

}

void solve()

{

    // CSES Meet in the Middle

    ll n, S;

    cin >> n >> S;

    vector<int> v(n), L, R;

    fore(v) cin >> x;

    sum(v, 0, n / 2, L, R);

    sum(v, n / 2, n, L, R);

    ll ans = 0;

    sort(all(R));

    if (R.empty())R.pb(0);

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

    {

        int ub = upper\_bound(all(R), S - L[i]) - R.begin();

        int lb = lower\_bound(all(R), S - L[i]) - R.begin();

        ans += (ub - lb);

    }

    cout << ans << endl;

}

**Bitmask**: getting all possible subset of k elemenst (k elements has 2^k subsets )

void bitmask(int k) //O(k\*(2^k))

{

    for (int j = 0; j < (1 << k); j++)

    {

        for (int i = 0; i < k; i++)

        {

            if (j & (1 << i))

            {

                //\* take the i-th element

            }

        }

    }

}

**Binary Search**: find any element in (log2n) complexity in a sorted array

int l = -1; //a[l]<=x

int r = n;  //a[r]>x

while (l + 1 < r)

{

    int m = (l + r) / 2;

    if (a[m] > x) r = m;

    else l = m;

}

if (a[l] == x) cout << "YES\n";

else cout << "NO\n";

**using recursion**

int binary(int arr[], int s, int l, int num)

{

    if (s <= l)

    {

        int mid = (s + l) / 2;

        if (arr[mid] == num) return mid;

        if (arr[mid] > num) return binary(arr, s, mid - 1, num);

        if (arr[mid] < num)  return binary(arr, mid + 1, l, num);

    }

    else return -1;

}

bool y = binary\_search(a.begin(), a.end(), x);

if (y)

cout << "YES\n";

else

    cout << "NO\n";

**Lower bound**: maximum index which in not greater than **X** / first jekhane **X** insert kora jabe

int l = -1; //a[l]<=x

    int r = n;  //a[r]>x

    while (l + 1 < r)

    {

        int m = (l + r) / 2;

        if (a[m] >= x) r = m;

        else l = m;

    }

    cout << l + 1 << endl;

    cout << lower\_bound(a.begin(), a.end(), x) - a.begin();

**Upper bound**: minimum index which in not less than **X** / last jekhane **X** insert kora jabe

    int l = -1; //a[l]<x

    int r = n;  //a[r]>=x

    while (l + 1 < r)

    {

        int m = (l + r) / 2;

        if (a[m] > x) r = m;

        else          l = m;

    }

    cout << r << endl;

    cout << upper\_bound(a.begin(), a.end(), x) - a.begin() << endl;

**Binary Search on Answer:**

1. Solution must be an monotonic function.
2. Used for getting maximum or minimum answer.

q. What is the minimum length of square that fill the area of H\*W rectengle with atmost n square??

    int l = -1, r = 1;

    while (!ok(r))

        r \*= 2;

    while (l + 1 < r)

    {

        int m = (l + r) / 2;

        if (ok(m))    r = m;

        else          l = m;

    }

    cout << r << endl;

bool ok(ll x

{

    return (x / h) \* (x / w) >= n;

}

**Binary Search on Answer (fraction):**

**q.** There are n ropes, you need to cut k pieces of the same length from them. Find the maximum length of pieces you can get.

    double l = 0, r = 1e7 + 10;

    for (int i = 1; i <= 100; i++)

    {

        double m = (l + r) / 2;

        if (ok(m))    l = m;

        else          r = m;

    }

    printf("%.6lf\n", l); /// can print l or r; /// both values are too close

bool ok(double x)

{

    int ans = 0;

    for (auto i : a)

        ans += (i / x);

    return ans >= k;

}

**Directional Array:**

int dx[] = {0, 0, -1, 1};

int dy[] = {1, -1, 0, 0};

int dxk[] = {-2, -2, -1, -1, 1, 2, 2, 1};

int dyk[] = {1, -1, 2, -2, 2, 1, -1, -2};

int dx1[] = {0, 1, 1, 1, 0, -1, -1, -1};

int dy1[] = {1, 1, 0, -1, -1, -1, 0, 1};

**BFS**: **O(V+E)**

**Main function**

void solve()

{

    cin >> n >> e;

    for (int i = 0; i < e; i++)

    {

        cin >> a >> b;

        v[a].pb(b), v[b].pb(a);

    }

    cout << "BFS: ";

    for (int i = 1; i <= n; i++)

    {

        if (!vis[i])  bfs(i);

    }

    cout << endl;

    cout << "Level Print: \n";

    for (int i = 1; i <= n; i++) cout << i << "->" << level[i] << endl;

    vector<int> p = path(1, 8);

    cout << "shortest path Print(1 to 8):\n";

    for (int i = 0; i < p.size(); i++) cout << p[i] << ' ';

    cout << endl;

}

**BFS function**

void bfs(int u)

{

    queue<int> q;

    q.push(u);

    vis[u] = 1,    par[u] = 0,    level[u] = 0;

    while (!q.empty())

    {

        int f = q.front();

        q.pop();

        cout << f << " ";

        for (int i = 0; i < v[f].size(); i++)

        {

            int x = v[f][i];

            if (!vis[x])

            {

                q.push(x);

                vis[x] = true;

                level[x] = level[f] + 1;

                par[x] = f;

            }

        }

    }

}

**path print**

vector<int> path(int s, int d)

{

    vector<int> p;

    while (par[d] != 0)

    {

        p.push\_back(d);

        d = par[d];

    }

    p.push\_back(d);

    return p;

}

**Multisource BFS on 2D**: push all source into the queue first & for 2D grid use **(O(n\*m))**

**Main function**

void solve()

{

    string st;

    cin >> n >> m;

    pii d;

    vector<pii> ss;

    for (int i = 0; i < n; i++)

    {

        cin >> grid[i];

        for (int j = 0; j < m; j++)

        {

            if (grid[i][j] == 'S')  ss.pb({i, j});

            if (grid[i][j] == 'G')  d.X = i, d.Y = j;

        }

    }

    bfs(ss);

    int dest = level[d];

    cout << dest << endl;

}

// S...#

// .#S..

// ....#

// #...D

**BFS function**

string grid[100];

map<pii, int> level;

int dx[] = {1, -1, 0, 0};

int dy[] = {0, 0, -1, 1};

bool valid(int x, int y)

{

    if (x < 0 || y < 0)

        return false;

    if (grid[x][y] != '#' && x >= 0 && x < n && y >= 0 && y < m) return 1;

    return 0;

}

void bfs(vector<pii> ss)

{

    queue<pii> q;

    fore(ss)

    {

        q.push(x);

        level[x] = 0;

    }

    while (!q.empty())

    {

        pii u = q.front();

        q.pop();

        for (int i = 0; i < 4; i++)

        {

            int xx = u.X + dx[i];

            int yy = u.Y + dy[i];

            if (valid(xx, yy) && level.find({xx, yy}) == level.end())

            {

                level[{xx, yy}] = level[{u.X, u.Y}] + 1;

                q.push({xx, yy});

            }

        }

    }

}

# 0/1 BFS: only 2 wght’s possible 0/1 (codechef Chef and Reversing) O(V+E)

# Main function

void solve()

{

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        cin >> x >> y;

        if (x == y)

            continue;

        g[x].pb({y, 0});

        g[y].pb({x, 1});

    }

    cout << bfs() << endl;

}

# BFS function

vector<pii> g[100006];

vector<int> lev(100006, MAX);

int n, m, x, y;

int bfs()

{

    deque<int> q;

    q.push\_back(1);

    lev[1] = 0;

    while (!q.empty())

    {

        int cur\_v = q.front();

        q.pop\_front();

        fore(g[cur\_v])

        {

            int child\_v = x.first;

            int wt = x.second;

            if (lev[cur\_v] + wt < lev[child\_v])

            {

                lev[child\_v] = lev[cur\_v] + wt;

                if (wt == 1)

                    q.push\_back(child\_v);

                else

                    q.push\_front(child\_v);

            }

        }

    }

    return lev[n] == MAX ? -1 : lev[n];

}

**DFS**: **O(V+E)**

int main()

{

    int n, m;

    cin >> n >> m;

    for (int i = 1; i <= m; i++)

    {

        int a, b;

        cin >> a >> b;

        adj[a].push\_back(b);

        adj[b].push\_back(a);

    }

    dfs(5);

}

**DFS function**

vector<int> adj[MAX];

bool vis[MAX];

/// PreOrder == DFS

void dfs(int v)

{

    vis[v] = 1;

    cout << v << ' ';

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

    {

        int x = adj[v][i];

        if (vis[x] == 1)

            continue;

        dfs(x);

    }

}

**DFS component count**:

    int count=0;

    for(int i=1; i<=n; i++)

    {

        if(vis[i]==0)

        {

            count++;

            dfs(i);

        }

    }

    cout<<count<<endl;

**DFS ParentPrint & PathPrint**:

**Main Fucntion**

void solve()

{

    cin >> n >> m;

    while (m--)

    {

        int u, v, c;

        cin >> u >> v >> c;

        edges[u].pb({v, c});

        edges[v].pb({u, c});

    }

    dfs(1);

    cout << endl;

    for (int i = 1; i <= n; i++)

    {

        cout << i << " : " << par[i] << endl;

    }

    path(5);

}

**DFS funtion**

int visited[10000];

int par[10000];

vector<pair<int, int>> edges[1000];

void dfs(int s)

{

    visited[s] = 1;

    cout << s << ' ';

    for (int i = 0; i < edges[s].size(); i++)

    {

        int v = edges[s][i].first;

        if (!visited[v])

        {

            par[v] = s;

            dfs(v);

        }

    }

}

**Path print**

void path(int d)

{

    if (d == 0)

        return;

    path(par[d]);

    cout << d << ' ';

}

**DFS Cycle Detection**:

**// White -> grey -> black method**

int visited[10000];

vector<pair<int, int>> edges[1000];

int cycle = 0;

void dfs(int s)

{

    visited[s] = 1; // grey

    for (int i = 0; i < edges[s].size(); i++)

    {

        int v = edges[s][i].first;

        if (visited[v] == 1)

            cycle = 1;

        if (visited[v] == 0)

            dfs(v);

    }

    visited[s] = 2;

}

**// Using Parent method**

bool dfs(int u, int par)

{

    vis[u] = 1;

    bool ans = 0;

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

    {

        int v = adj[u][i];

        if (v == par)continue;

        if (vis[v] == true)

        {

            ans = 1;

            continue;

        }

        bool temp = dfs(v, u);

        if (ans == true || temp == true)

            ans = true;

    }

    return ans;

}

**Dijkstra**: Single source Shortest Path **O((V+E)\*log(V))**

Drawback: **negative weighted edge**

vector<pair<int, int>> graph[501];

vector<int> visited(501, 1000000000);

    for (int i = 1; i <= m; i++)

    {

        cin >> u >> v >> w;

        graph[u].push\_back({w, v});

        graph[v].push\_back({w, u});

    }

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

    cin >> s;

    visited[s] = 0;

    pq.push({0, s});

    while (!pq.empty())

    {

        pair<int, int> f = pq.top();

        pq.pop();

        int p = f.second;

        for (int i = 0; i < graph[p].size(); i++)

        {

            int c = graph[p][i].second;

            int cw = graph[p][i].first;

            if (visited[c] > visited[p] + cw)

            {

                visited[c] = visited[p] + cw;

                pq.push({visited[c], c});

            }

        }

    }

    for (int i = 0; i < n; i++)

    {

        if (visited[i] == 1000000000)

            cout << "Impossible\n";

        else

            cout << visited[i] << endl;

    }

**Bellman Ford**: Single source Shortest Path **O(V\*E)**

// it works for negative edges and negative cycles

// it can detecte negative cycles as well

void solve()

{

    int n, m;

    cin >> n >> m;

    vector<vector<int>> edg;

    for (int i = 0; i < m; i++)

    {

        int u, v, w;

        cin >> u >> v >> w;

        edg.pb({u, v, w});

        // edg.pb({v, u, w});

    }

    vector<int> dis(n + 1, INF);

    int s;

    cin >> s;

    dis[s] = 0;

    bool cy = false;

    for (int i = 0; i < n - 1; i++)

    {

        cy = false;

        fore(edg)

        {

            int u = x[0];

            int v = x[1];

            int w = x[2];

            if (dis[v] > dis[u] + w)

                cy = true;

            dis[v] = min(dis[v], dis[u] + w);

        }

    }

    if (cy) cout << "Negative cycle detected." << endl;

    for (int i = 1; i <= n; i++)

        cout << dis[i] << ' ';

}

**Prims Algorithm**: Minimum Spanning Tree **O(2\*V)**

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

    pq.push({0, 0});

    while (!pq.empty())

    {

        int p = pq.top().second;

        int pw = pq.top().first;

        pq.pop();

        if (vis[p] != -1)

            continue;

        vis[p] = pw;

        for (int i = 0; i < graph[p].size(); i++)

        {

            int c = graph[p][i].second;

            int cw = graph[p][i].first;

            if (vis[c] == -1)

                pq.push({cw, c});

        }

    }

    for (int i = 0; i < n; i++)

    {

        sum += vis[i];

        cout << vis[i] << ' ';

    }

    cout << sum << endl;

**DSU**: Disjoint Set Union **[O(alpha(n))]**

int par[lim], \_size[lim];

void init()

{

    for (int i = 1; i <= n; i++)

        par[i] = i,  \_size[i] = 1;

}

int findP(int u)

{

    if (par[u] == u)  return u;

    return par[u] = findP(par[u]);

}

void union\_set(int u, int v)

{

    int pu = findP(u);

    int pv = findP(v);

    if (pu == pv) return;

    if (\_size[pv] > \_size[pu]) swap(pu, pv);

    par[pv] = pu;

    \_size[pu] += \_size[pv];

    \_size[pv] = 0;

}

void solve()

{

    cin >> n >> m;

    init();

    for (int i = 0; i < m; i++)

    {

        int u, v;

        cin >> u >> v;

        union\_set(u, v);

    }

    set<int> s;

    for (int i = 1; i <= n; i++) /// total number of components

    {

        s.insert(findP(i));

    }

    cout << s.size() << endl;

}

**DSU (cycle detection)**:

    for (int i = 0; i < egdes; i++)

    {

        int u = edges[i].first;

        int v = edges[i].second;

        if (findp(u) == findp(v))

        {

            cout << u << ' ' << v << endl;

            cout << "cycle detected\n";

            return;

        }

        else

            unionset(u, v);

    }

**Kruskal Algorithm**: **MST using DSU O(E log V)**

int kruskal()

{

    sort(edge.begin(), edge.end()); // mlogm

    int cost = 0, cnt = 0;

    for (int i = 0; i < m; i++) // mlogm

    {

        u = edge[i].second.first;

        v = edge[i].second.second;

        w = edge[i].first;

        u = findP(u);

        v = findP(v);

        if (v == u)  continue;

        cost += w;

        cnt++;

        par[v] = u;

        cout << u << ' ' << v << ' ' << w << endl;

        if (cnt == n - 1) break;

    }

    return cost;

}

void solve()

{

    // mlogm+mlogm

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

        par[i] = i;

    for (int i = 0; i < m; i++)

    {

        cin >> u >> v >> w;

        edge.push\_back({w, {u, v}});

    }

    int cost = kruskal();

    cout << cost << endl;

}

**Discovery time and finishing time using dfs**:

vector<int> v[100];

vector<int> f(100, 0);

vector<int> s(100, 0);

int t = 1;

void dfs(int u)

{

    s[u] = t++;

    for (int i = 0; i < v[u].size(); i++)

    {

        int x = v[u][i];

        if (f[x] == 0) dfs(x);

    }

    f[u] = t++;

}

void solve()

{

    int n, m, a, b;

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        cin >> a >> b;

        v[a].push\_back(b);

    }

    dfs(3);

    for (int i = 1; i <= n; i++)

        cout << i << '-' << s[i] << ' ' << f[i] << endl;

}

**Topological Sort (DFS)**:

stack<int> topoligicalysorted;

list<int> li;

vector<int> v[100];

vector<int> f(100, 0);

vector<int> s(100, 0);

int t = 1;

void dfs(int u)

{

    s[u] = t++;

    for (int i = 0; i < v[u].size(); i++)

    {

        int x = v[u][i];

        if (f[x] == 0)

            dfs(x);

    }

    f[u] = t++;

    topoligicalysorted.push(u);

    li.push\_front(u);

}

void solve()

{

    int n, m, a, b;

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        cin >> a >> b;

        v[a].pb(b);

    }

    dfs(3);

    for (auto it = li.begin(); it != li.end(); it++)

        cout << \*it << ' ';

    cout << endl;

}

**Topological Sort (BFS\_kahn’sAlgo)**: **O(V+E)**

vector<int> v[100];

int indegree[100];

int n, m, a, b;

void solve()

{

    cin >> n >> m;

    for (int i = 0; i < n; i++)

    {

        cin >> a >> b;

        v[a].push\_back(b);

        indegree[b]++;

    }

    queue<int> q;

    for (int i = 1; i <= n; i++)

    {

        if (!indegree[i])

            q.push(i);

    }

    vector<int> ans;

    while (!q.empty())

    {

        int f = q.front();

        q.pop();

        ans.push\_back(f);

        fore(v[f])

        {

            indegree[x]--;

            if (indegree[x] == 0)

                q.push(x);

        }

    }

    if (ans.size() == n)

    {

        print(ans);

        cout << endl;

    }

    else

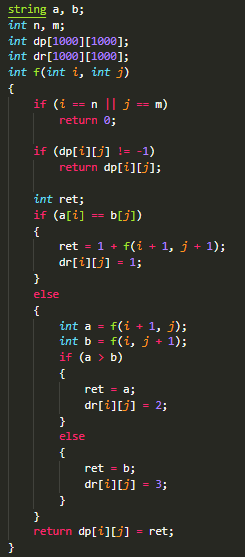
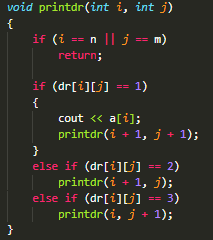
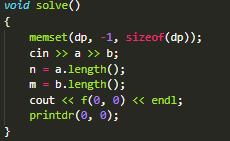
        cout << "IMPOSSIBLE\n";

}

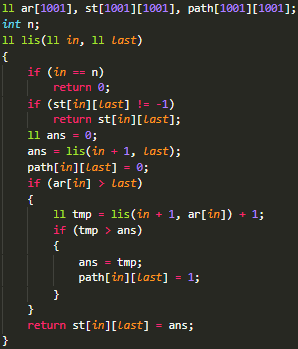
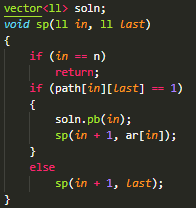
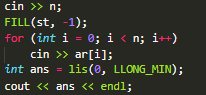
**Dynamic Programing**:

1. Array of state dimentions
2. Memset
3. if condition
4. value assign

**Longest common subsequence(pathPrint):**

**Longest increasing subsequence:**

**Iterative DP: which is known\_initialize it -> get ans from previous initialization for every transaction (uva-11715)**

**DigitDp:**

*//! given A ,B and X . find how many number*

*//! between A and B Has digit sum of X.*

string a, b;

int x;

int dp[11][2][92];

int f(int *pos*, bool *isSmall*, int *digitSum*, string &*s*)

{

*if* (*pos* == *s*.length())

*return* *digitSum* == x;

*if* (dp[*pos*][*isSmall*][*digitSum*] != -1)

*return* dp[*pos*][*isSmall*][*digitSum*];

    int hi = *s*[*pos*] - '0', ret = 0;

*if* (*isSmall*)

        hi = 9;

*for* (int i = 0; i <= hi; i++)

    {

        bool newSmall = *isSmall* || (i < hi);

        ret += f(*pos* + 1, newSmall, *digitSum* + i, *s*);

    }

*return* dp[*pos*][*isSmall*][*digitSum*] = ret;

}

void solve()

{

    cin >> a >> b >> x;

    FILL(dp, -1);

    int y = f(0, 0, 0, b);

    int A = stoi(a);

    A--;

    a = to\_string(A);

    FILL(dp, -1);

    int z = f(0, 0, 0, a);

    cout << y - z << endl;

}

**BitwiseDp:**

bool isSet(int *n*, int *k*)

{

*return* !(*n* & (1 << *k*));

}

int makeSet(int *n*, int *k*)

{

*n* = *n* | (1 << *k*);

*return* *n*;

}

int n, a;

vector<int> v[20];

int dp[20][70000];

int f(int *x*, int *w*)

{

*if* (*w* == (1 << n) - 1)

*return* 0;

    int ans = -INF;

*if* (dp[*x*][*w*] != -1)

*return* dp[*x*][*w*];

*for* (int i = 0; i < v[*x*].size(); i++)

    {

*if* (isSet(*w*, i))

        {

            ans = max(ans, f(*x* + 1, makeSet(*w*, i)) + v[*x*][i]);

        }

    }

*return* dp[*x*][*w*] = ans;

}

void solve()

{

    cin >> n;

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

        v[i].clear();

    FILL(dp, -1);

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

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

        {

            cin >> a;

            v[i].push\_back(a);

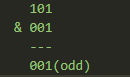
        }

    CASE();

    cout << f(0, 0) << endl;

}

**Bitwise operation**: **&(and), |(or), !(not), ^(xor if 2bits are same then 0)**



leftShift: a = a << 2; ///a<<n -> a\*2^n //multiply a with 2 n times

rightShift: b = b >> 2; ///a>>n -> a/a^n //divide b by 2 n times

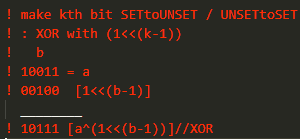
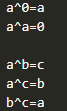
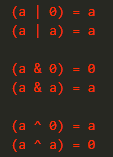
**Check if the kth bit is set or not?**

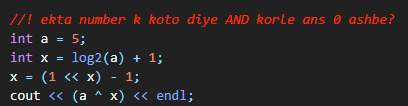
bool iskthbitSet(int a, int k)

{

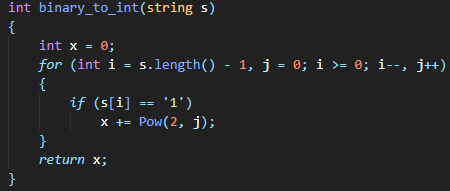
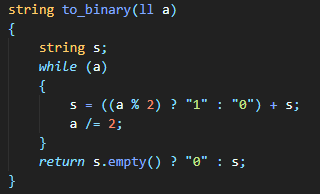
    return (a & (1 << k));

}



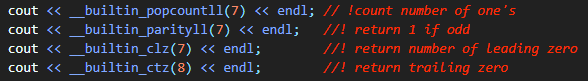
**Int to Binary**: **Binary to int**:



**unsetTOset: (a|(1<<i))**

**setTOunset: (a&(`(1<<i))**

**ToogleBit: (a^(1<<i))**



**Geometry**:

\*check if 3 point are collinear->

        if((y3-y2)\*(x2-x1)==(y2-y1)\*(x3-x2))YES;

   \*is point a,b in the same side of a line???

    point's of line (x1,y1),(x2,y2)

    (x2-x1)\*(y3-y1)-(y2-y1)\*(x3-x1);

    if above equation is positive/negetive for both a and b then they are in the same side

ll Area\_of\_threePoint(ll x1, ll y1, ll x2, ll y2, ll x3, ll y3)

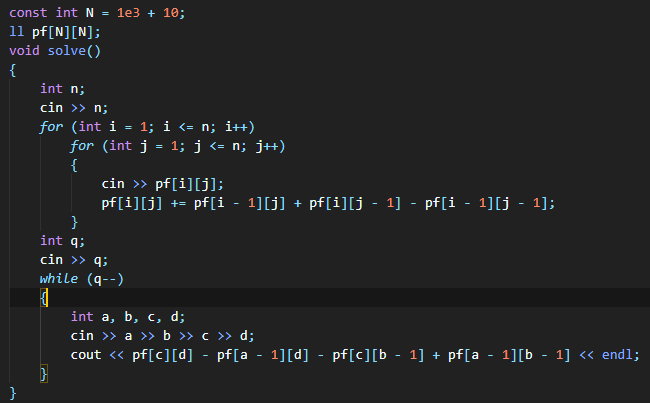
{

*return* abs(x1 \* (y2 - y3) + x2 \* (y3 - y1) + x3 \* (y1 - y2));

}

**Precalculate2D (prefix sum on 2D)**:

*//! given an 2d array we have to calculate the sum of point(a,b) to point(c,d)*

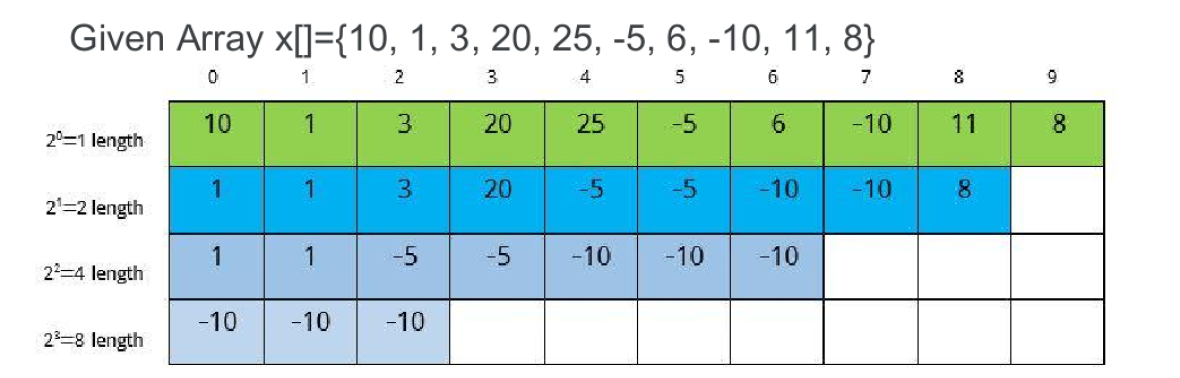
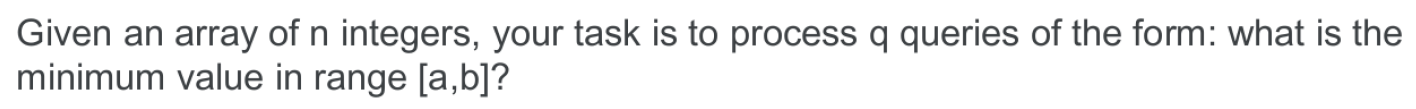


**Sparse Table**:

Preprocessing : nlog(n)

PerQuery : O(1) -> q query: O(q)

Total= O(nlogn+q)



Sparse[Power\_of\_two][staring\_index]=min(a[start],a[start+1],… … ,a[start+2^(power)-1])

**Built Formula:** **Sparse[k][i]=min(sparse[k-1][i],sparse[k-1][i+len])** here, Len= 2^(k-1)

Sparse[2][3]=min(a[3],a[4],a[5],a[6])

Sparse[2][3]=min(min(a[3],a[4]),min(a[5],a[6]))

Sparse[2][3]=min(sparse[1][3],sparse[1][3+2]); len=2^(2-1)=2

Sparse[2][3]=min(20,-5)=-5

Query(2,6) // 5 length

Min(a[2],a[3],a[4],a[5],a[6]) = min(min(a[2],a[3],a[4],a[5]),min(a[3],a[4],a[5],a[6]))

=min(sparse[2][2],sparse[2][3])

K= log(5)=2 ;len=2^k=4

**Query(left,right)=min(sparse[k][left],sparse[k][right-len+1]);**

template <typename V, typename T>

struct SparseTable

{

    static int highest\_bit(unsigned *x*) { *return* *x* == 0 ? -1 : 31 - \_\_builtin\_clz(*x*); }

    int n, level;

    vector<vector<T>> st;

*/\*OPRATION CHANGEABLE\*/*

    T f(T *a*, T *b*)

    {

*return* min(*a*, *b*);

    }

    void build(const vector<V> &*v* = {})

    {

        st[0] = *v*;

*for* (int k = 1; k <= level; k++)

        {

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

            {

                st[k][i] = f(st[k - 1][i], st[k - 1][i + (1 << (k - 1))]);

            }

        }

    }

    SparseTable(const vector<V> &*v*) : n((int)*v*.size()), level(highest\_bit(n))

    {

        st.resize(level + 1, vector<T>(n));

        build(*v*);

    }

    T qry(int *l*, int *r*)

    {

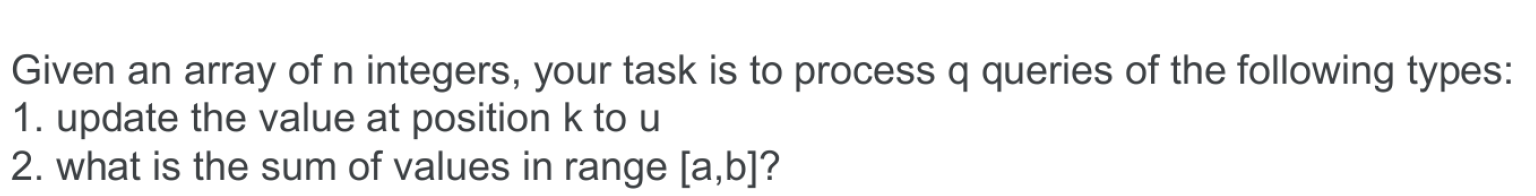
        int k = highest\_bit(*r* - *l* + 1);

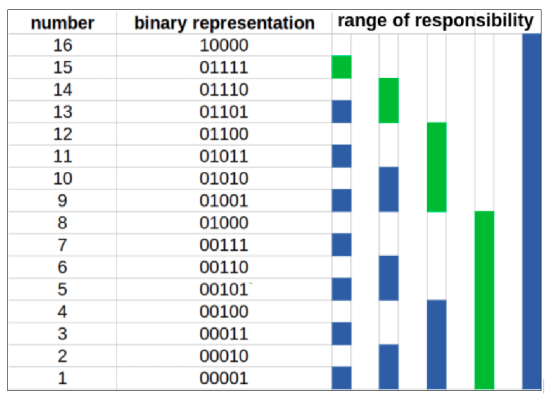
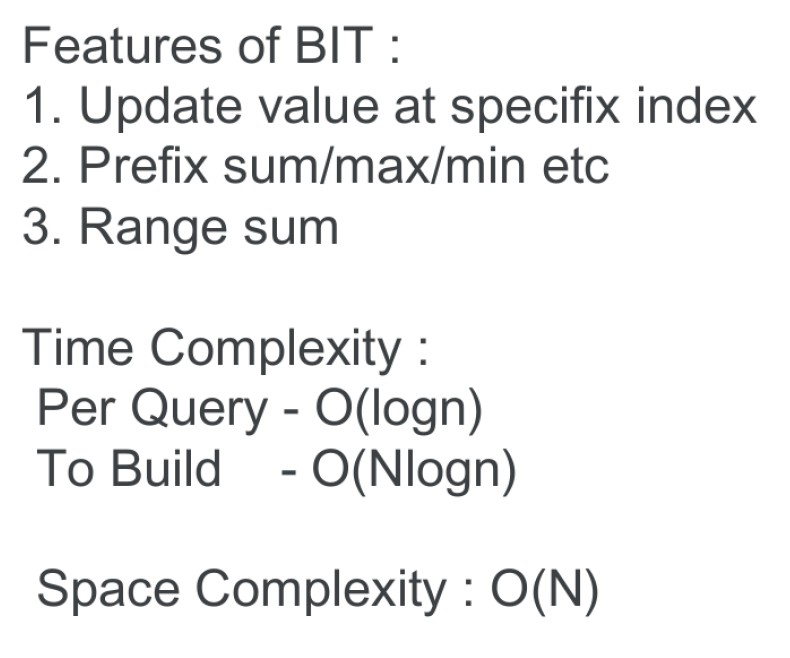
*return* f(st[k][*l*], st[k][*r* - (1 << k) + 1]);

    }

};

**BinaryIndexTree(BIT)/FenwickTree**:CSES-1648





int BIT[N]; *//! 1based*

void update(int *idx*, ll *val*) *//! set v[idx]+=val*

{

*while* (*idx* <= n)

    {

        BIT[*idx*] += *val*;

*idx* += (*idx* & -*idx*);

    }

}

int qry(int *idx*) *//! return the prefix sum form 0 to idx*

{

    ll ret = 0;

*while* (*idx* > 0)

    {

        ret += BIT[*idx*];

*idx* ^= (*idx* & -*idx*);

    }

*return* ret;

}

int buildBIT(int *n*) *//! nlog(n)*

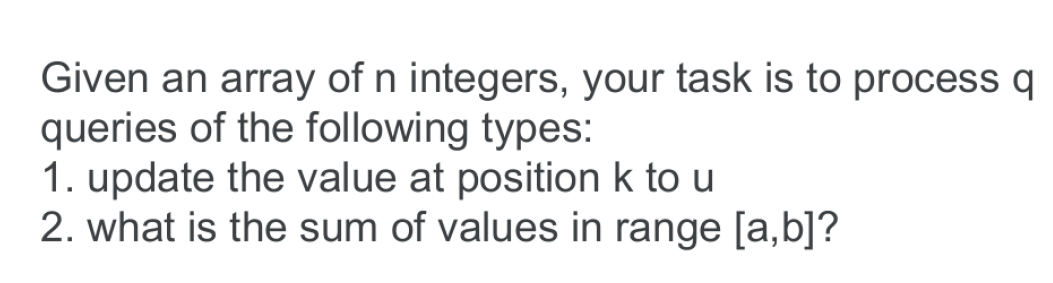
{

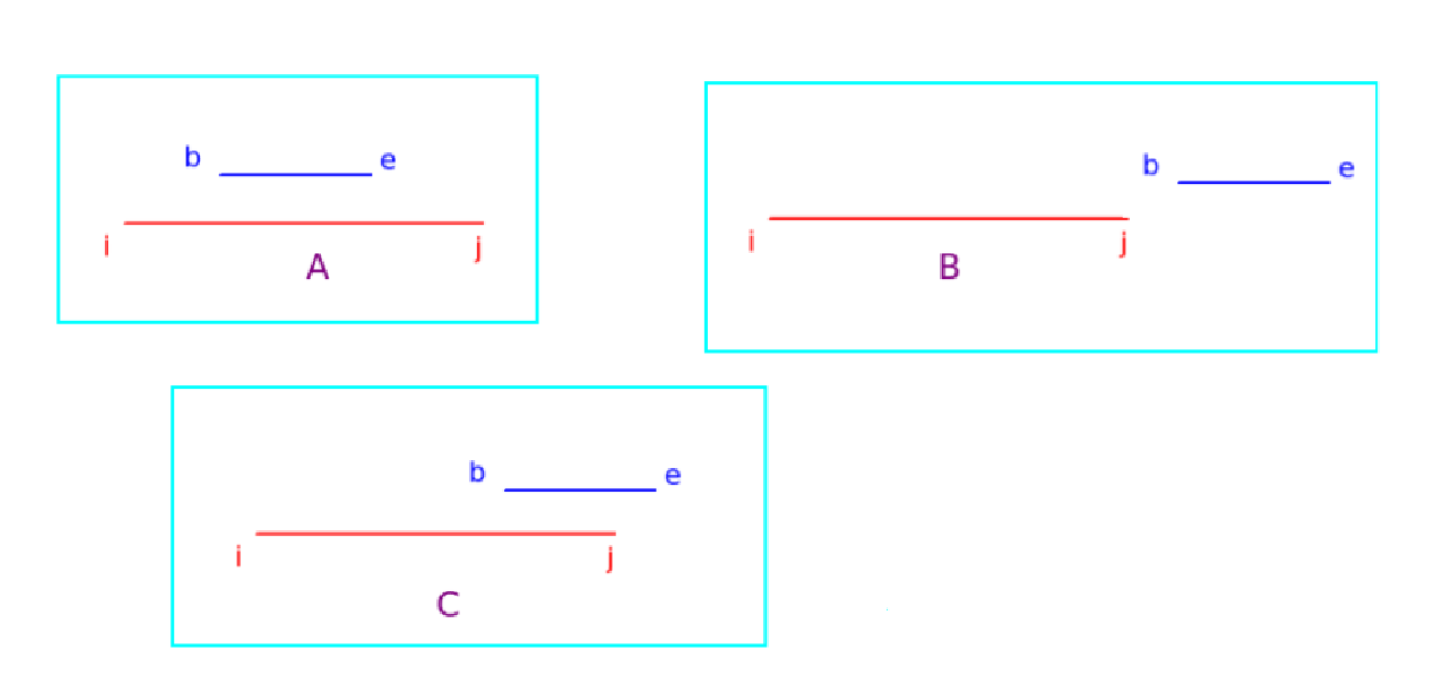
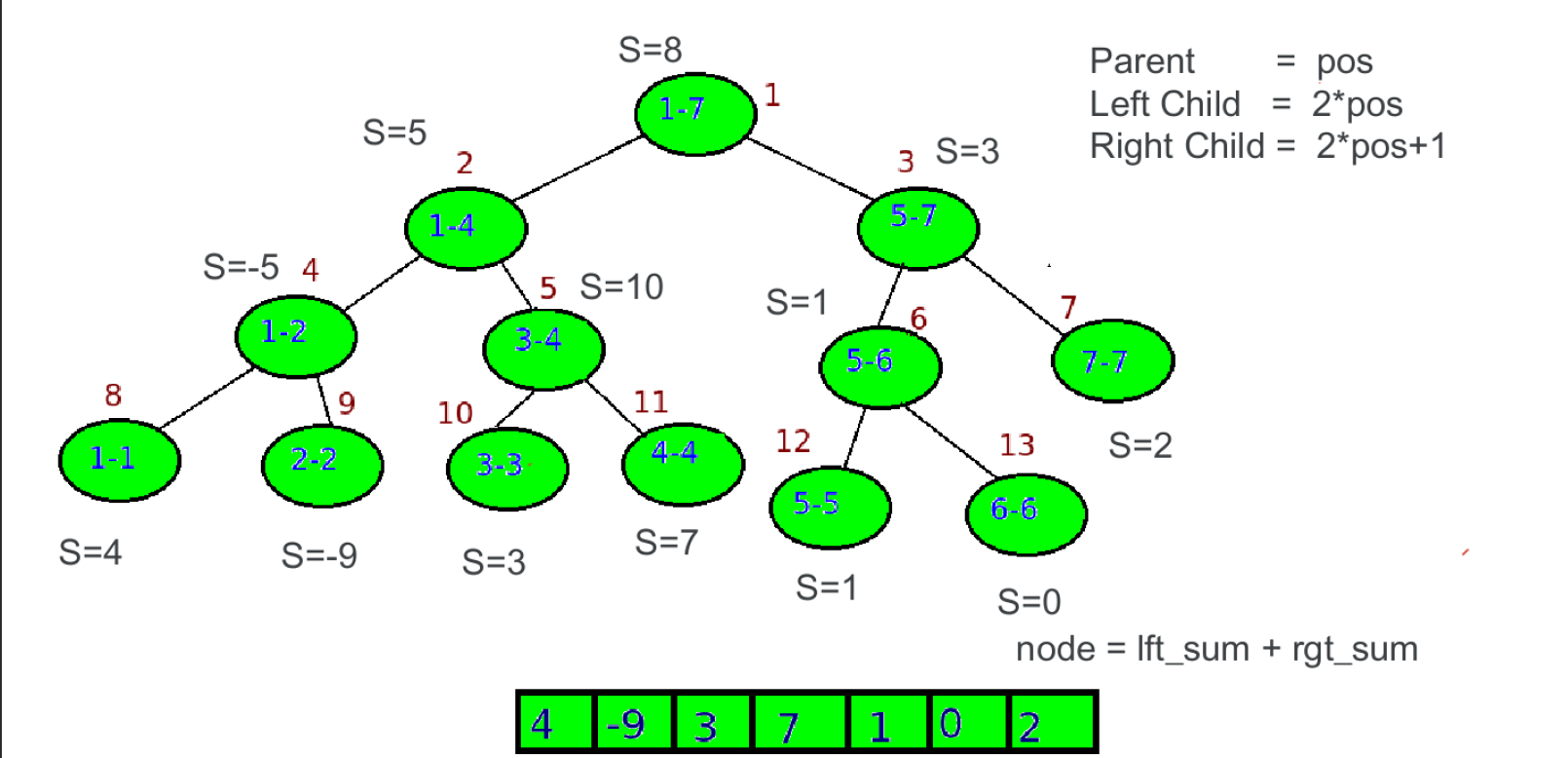
*for* (int i = 1; i <= *n*; i++)

        update(i, v[i]);

}

**SegmentTree**:





int arr[N];

int Tree[4 \* N];

void init(int *node*, int *b*, int *e*)

{

*if* (*b* == *e*)

    {

        Tree[*node*] = arr[*b*];

*return*;

    }

    int left = *node* \* 2;

    int right = *node* \* 2 + 1;

    int mid = (*b* + *e*) / 2;

    init(left, *b*, mid);

    init(right, mid + 1, *e*);

    Tree[*node*] = Tree[left] + Tree[right];

}

int query(int *node*, int *b*, int *e*, int *i*, int *j*)

{

*if* (*i* > *e* || *j* < *b*) *// out*

*return* 0;

*if* (*b* >= *i* && *e* <= *j*) *// relevent segment*

*return* Tree[*node*];

    int left = *node* \* 2; *// aro vangte hobe*

    int right = *node* \* 2 + 1;

    int mid = (*b* + *e*) / 2;

    int p1 = query(left, *b*, mid, *i*, *j*);

    int p2 = query(right, mid + 1, *e*, *i*, *j*);

*return* p1 + p2; *// left and right side r sum*

}

void update(int *node*, int *b*, int *e*, int *i*, int *newvalue*)

{

*if* (*i* > *e* || *i* < *b*) *// out*

*return*;

*if* (*b* >= *i* && *e* <= *i*) *// relevent segment*

    {

        Tree[*node*] = *newvalue*; *// assiging new value*

*return*;

    }

    int left = *node* \* 2;

    int right = *node* \* 2 + 1;

    int mid = (*b* + *e*) / 2;

    update(left, *b*, mid, *i*, *newvalue*);

    update(right, mid + 1, *e*, *i*, *newvalue*);

    Tree[*node*] = Tree[left] + Tree[right];

}

**Lazy Propagation**: for range update

const int maxN = 100001;

int ar[maxN];

int st[4 \* maxN], lazy[4 \* maxN];

void build(int *si*, int *ss*, int *se*)

{

*if* (*ss* == *se*)

    {

        st[*si*] = ar[*ss*];

*return*;

    }

    int mid = (*ss* + *se*) / 2;

    build(2 \* *si*, *ss*, mid);

    build(2 \* *si* + 1, mid + 1, *se*);

    st[*si*] = st[2 \* *si*] + st[2 \* *si* + 1];

}

int query(int *si*, int *ss*, int *se*, int *qs*, int *qe*)

{

*if* (lazy[*si*] != 0)

    {

        int dx = lazy[*si*];

        lazy[*si*] = 0;

        st[*si*] += dx \* (*se* - *ss* + 1);

*if* (*ss* != *se*)

            lazy[2 \* *si*] += dx, lazy[2 \* *si* + 1] += dx;

    }

*if* (*ss* > *qe* || *se* < *qs*)

*return* 0;

*if* (*ss* >= *qs* && *se* <= *qe*)

*return* st[*si*];

    int mid = (*ss* + *se*) / 2;

*return* query(2 \* *si*, *ss*, mid, *qs*, *qe*) + query(2 \* *si* + 1, mid + 1, *se*, *qs*, *qe*);

}

void update(int *si*, int *ss*, int *se*, int *qs*, int *qe*, int *val*)

{

*if* (lazy[*si*] != 0)

    {

        int dx = lazy[*si*];

        lazy[*si*] = 0;

        st[*si*] += dx \* (*se* - *ss* + 1);

*if* (*ss* != *se*)

            lazy[2 \* *si*] += dx, lazy[2 \* *si* + 1] += dx;

    }

*if* (*ss* > *qe* || *se* < *qs*)

*return*;

*if* (*ss* >= *qs* && *se* <= *qe*)

    {

        int dx = (*se* - *ss* + 1) \* *val*;

        st[*si*] += dx;

*if* (*ss* != *se*)

            lazy[2 \* *si*] += *val*, lazy[2 \* *si* + 1] += *val*;

*return*;

    }

    int mid = (*ss* + *se*) / 2;

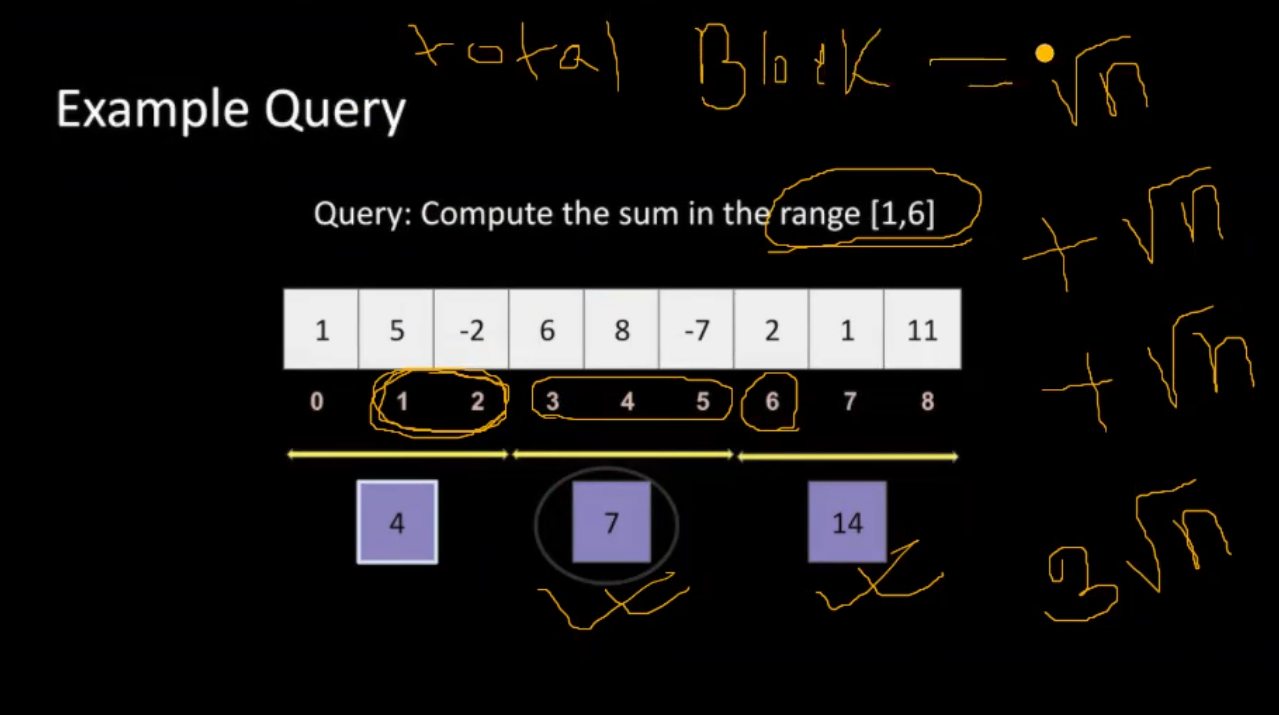
    update(2 \* *si*, *ss*, mid, *qs*, *qe*, *val*);

    update(2 \* *si* + 1, mid + 1, *se*, *qs*, *qe*, *val*);

    st[*si*] = st[2 \* *si*] + st[2 \* *si* + 1];

}

**Sqrt Decomposition**: O(3\*sqrt(n))



int Block[200005];

vector<int> v;

int preprocess(int *n*)

{

    int cur\_blk = -1, blk\_sz = sqrt(*n*);

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

    {

*if* (i % blk\_sz == 0)cur\_blk++; *// new block*

        Block[cur\_blk] += v[i];

    }

*return* blk\_sz;

}

int qry(int *blk\_sz*, int *l*, int *r*)

{

    int sum = 0;

*while* (*l* < *r* && *l* % *blk\_sz* != 0)

    {

        sum += v[*l*]; *l*++;

    }

*while* (*l* + *blk\_sz* <= *r*)

    {

        sum += Block[*l* / *blk\_sz*];  *l* += *blk\_sz*;

    }

*while* (*l* <= *r*)

    {

        sum += v[*l*];  *l*++;

    }

*return* sum;

}

void update(int *blk\_sz*, int *i*, int *val*)

{

    int blk\_no = *i* / *blk\_sz*;

    Block[blk\_no] -= v[*i*];

    Block[blk\_no] += *val*;

    v[*i*] = *val*;

}

void solve()

{

    int n, q, l, r;

    cin >> n;

    v.resize(n);

    fore(v) cin >> x;

    int blk\_size = preprocess(n);

    cin >> q;

*while* (q--)

    {

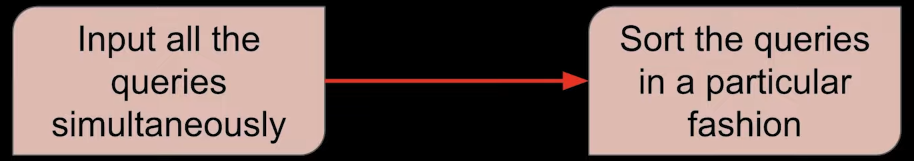
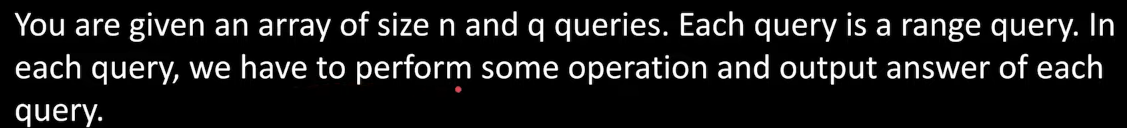
        cin >> l >> r;        l--, r--;

        cout << qry(blk\_size, l, r) << endl;

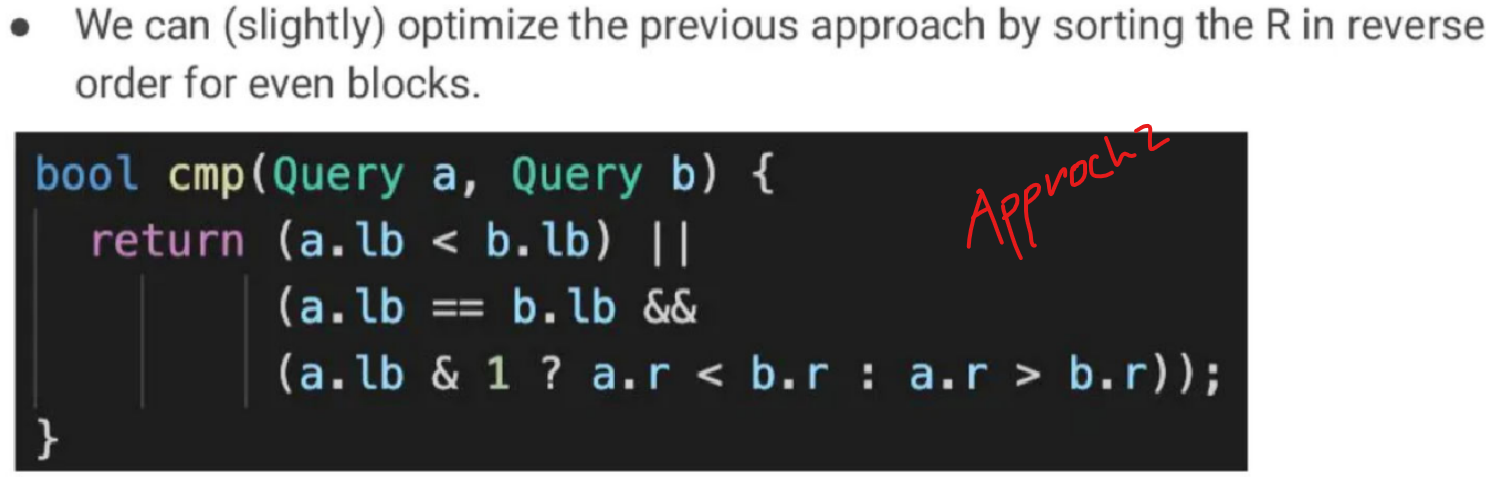
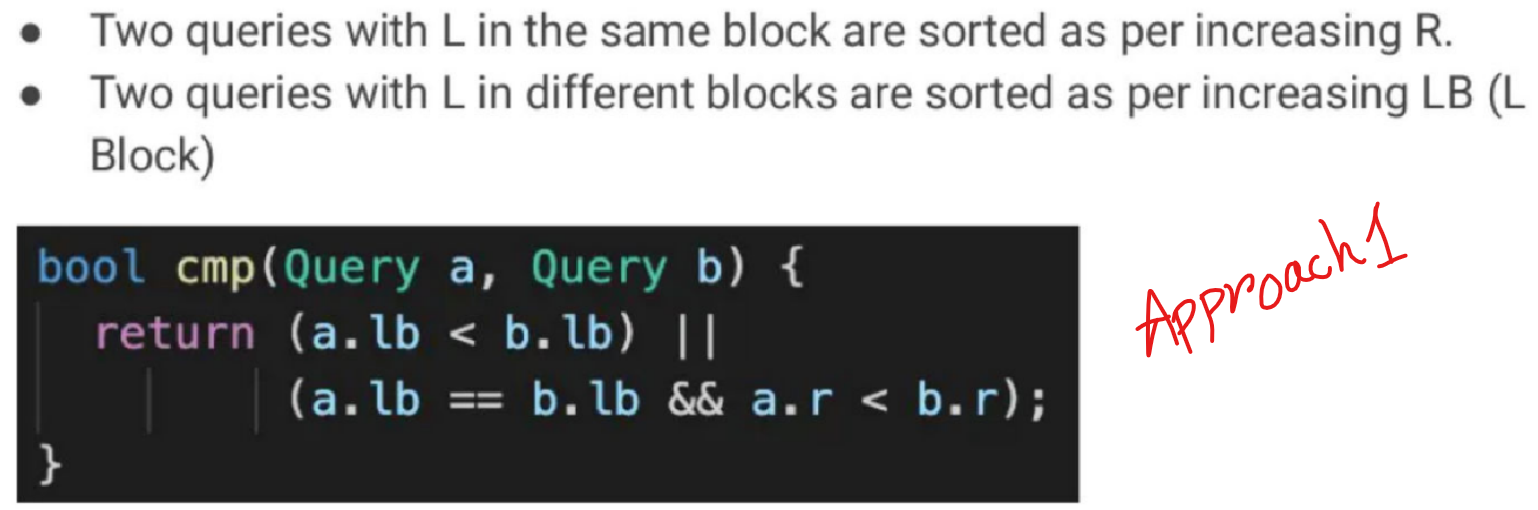
    }

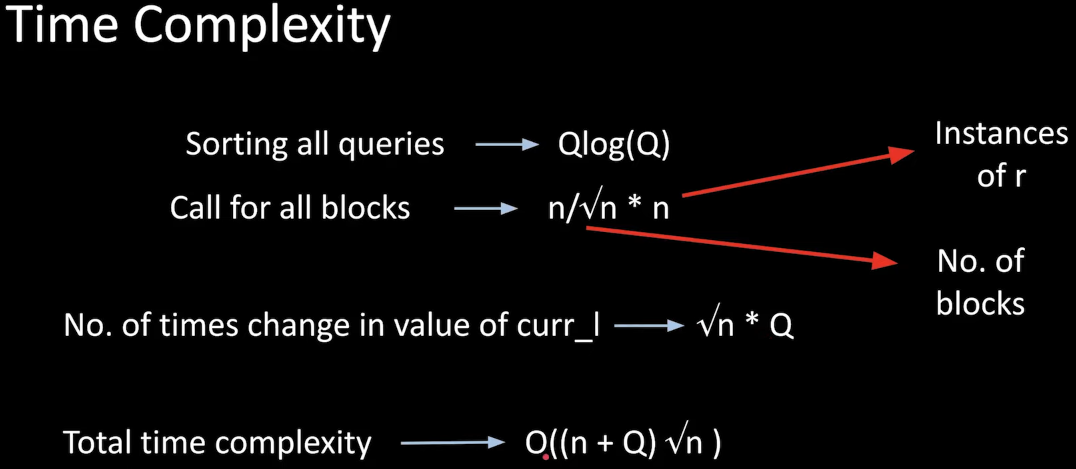
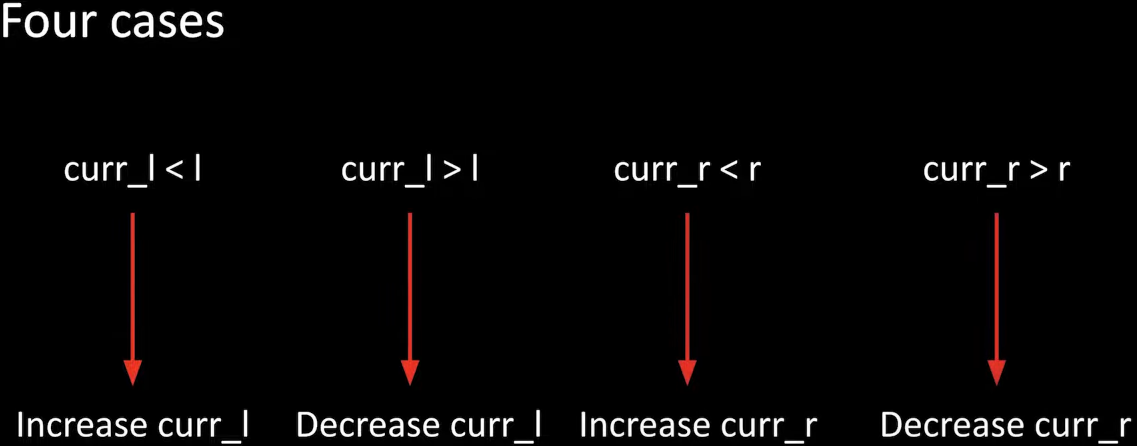
}

**MO’s Algorithm**:









int rootN;

struct Q

{

    int idx, l, r;

    bool operator<(const Q &*ob*) const

    {

*if* (l / rootN == *ob*.l / rootN)

*return* ((l / rootN) & 1) ? r > *ob*.r : r < *ob*.r;

*return* l < *ob*.l;

    }

};

Q q[100005];

void solve()

{

    int n, l, r;

    cin >> n;

    vector<int> a(n);

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

        cin >> a[i];

    rootN = sqrtl(n);

    int queries;

    cin >> queries;

*for* (int i = 0; i < queries; i++)

    {

        cin >> l >> r;

        q[i].l = l, q[i].r = r, q[i].idx = i;

    }

    sort(q, q + queries);

    vector<int> ans(queries);

    int curr\_l = 0, curr\_r = -1;

    int curr\_ans = 0;

*for* (int i = 0; i < queries; i++)

    {

        l = q[i].l, r = q[i].r;

        l--, r--; *//! since the indexing is 0 base and queries are one base*

*while* (curr\_r < r)

            curr\_r++, curr\_ans += a[curr\_r];

*while* (curr\_l > l)

            curr\_l--, curr\_ans += a[curr\_l];

*while* (curr\_l < l)

            curr\_ans -= a[curr\_l], curr\_l++;

*while* (curr\_r > r)

            curr\_ans -= a[curr\_r], curr\_r--;

        ans[q[i].idx] = curr\_ans;

    }

*for* (int i = 0; i < queries; i++)

        cout << ans[i] << endl;

}

**String Hashing**:

**H(s[0…n-1])= Σsi \* Pi**

**H(s[0…n-1]+x)= H(s)+ x\*Pn**

**H(x+ s[0…n-1])=x+ P\*H(s)**

**H(s[i…j])= H(s[0…j])-H(s[0…i-1])) \* p-i**

**STL:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Container** | **Find** | **Insert** | **erase** | **Clear** |
| Vector | O(n) | O(n) | O(n) | O(n) |
| Set/Map | Log(n) | Log(n) | Log(n) | O(n) |
| Unordered\_map/set | O(n) | O(n) | O(n) | O(n) |
| Deque | x | O(n) | O(n) | O(n) |

**Techniques**:

* Query L, R dile L++ & (R+1) – kore precalate korle hobe
* pair(i,j) condition [i<j] hole sort kore ans ber kore ans k 2 diye vag kore ans ber kora jabe.
* \* common number of divisors of some elements == divisors of gcd of those element**s**

Series:

Geometry Series: 1+r+r^2+r^3+… … +rn-1 = (1-r^n)/(1-r)