*Data Structure

Binary Indexed Tree:

```
void update(int idx,int x,int n){
    while(idx <= n){
        tree[idx] += x;
        idx += idx&(-idx);} }
int query(int idx) {
    int sum = 0;
    while(idx > 0){
        sum += tree[idx];
        idx -= idx&(-idx); }
    return sum; }
```

Merge Sort Tree:

```
ll cum[MAX];
vector<ll>tree[4*MAX];
void build(int l,int r,int node) {
    if(l == r) {

   tree[node].push_back(cum[l]);
        return; }
    int mid = (l+r)/2;
    int left = 2*node , right =
   2*node+1;
    build(l,mid,left);
    build(mid+1,r,right);

   merge(tree[left].begin(),tree[le
   ft].end(),tree[right].begin(),tr
   ee[right].end(),back_inserter(tr
   ee[node])); }
int query(int L,int R,int l,int
   r,int node,ll t) {
    if(l > R || r < L)
        return 0;
    else if(l>=L && r<=R)
        return
   lower_bound(tree[node].begin(),t
   ree[node].end(),t)-tree[node].be
   gin();
    int mid = (l+r)/2;
    return
   query(L,R,l,mid,2*node,t) +
   query(L,R,mid+1,r,2*node+1,t); }
```

Maximum Histogram:

```
ll maxHistogram(vector<ll>
   &hist,int n) {
    stack<int>st;
    ll mx = -1;
    int i = 0;
    while(i <= n) {
        ll h = (i == n) ? 0 :
   hist[i];
        if(st.empty() || hist[i] >=
   hist[st.top()])
            st.push(i++);
        else {
            int top = st.top();
            st.pop();
            mx = max(mx, hist[top]
   * (st.empty() ? i :
   i-1-st.top()));
        } }
    return mx; }
```

Policy Based Data Structure:

```
#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;
template <typename T> using Set =
   tree<T, null_type,less<T>,
```

```
rb_tree_tag,tree_order_statistic
s_node_update>;
Set <int> st;
int main() {
    st.insert(5); //Insert
    st.erase(5); //Delete
    st.insert(1);
    st.insert(2);
    st.insert(9);
    cout << *st.find_by_order(0) <<
endl; //Find value by rank
    cout << st.order_of_key(9) <<
endl; //Find value's rank
    /* For multiple same element,
use pair, store index in second
of pair */ }
```

Persistent Segment Tree:

```
struct Node {
    int left, right, val;
} tree[MAX*20];
int a[MAX], root[MAX], id;
void build(int pos,int l,int r) {
    if(l == r) {
        tree[pos].val = a[l];
        return; }
    int mid = (l+r)>>1;
    tree[pos].left = ++id,
tree[pos].right = ++id;
    build(tree[pos].left,l,mid);
    build(tree[pos].right,mid+1,r);
    tree[pos].val =
tree[tree[pos].left].val +
tree[tree[pos].right].val; }
int update(int pos,int l,int r,int
idx,int v) {
    if(idx > r || idx < l)
        return pos;
    else if(l == r) {
        tree[++id] = tree[pos];
        tree[id].val += v;
        return id; }
    int mid = (l+r)>>1;
    tree[++id] = tree[pos], pos =
id;
    tree[pos].left =
update(tree[pos].left,l,mid,idx,
v);
    tree[pos].right =
update(tree[pos].right,mid+1,r,i
dx,v);
    tree[pos].val =
tree[tree[pos].left].val +
tree[tree[pos].right].val;
    return pos; }
int query(int pos,int l,int r,int
L,int R) {
    if(l > R || r < L)
        return 0;
    else if(l >= L && r <= R)
        return tree[pos].val;
    int mid = (l+r)/2;
    int x =
query(tree[pos].left,l,mid,L,R);
    int y =
query(tree[pos].right,mid+1,r,L,
R);
    return x+y; }
int kthElement(int a,int b,int
l,int r,int k) {
    if(l == r)
        return l;
    int mid = (l+r)>>1;
    int cnt =
tree[tree[a].left].val -
tree[tree[b].left].val;
    if(cnt >= k)
```

```
        return
    kthElement(tree[a].left,tree[b].
    left,l,mid,k);
     else
        return
    kthElement(tree[a].right,tree[b]
    .right,mid+1,r,k-cnt); }
int lessCnt(int a,int b,int l,int
    r,int idx) {
     if(r <= idx)
        return tree[a].val -
    tree[b].val;
     int mid = (l+r)>>1;
     if(idx <= mid)
        return
    lessCnt(tree[a].left,tree[b].lef
    t,l,mid,idx);
     else
        return
    lessCnt(tree[a].left,tree[b].lef
    t,l,mid,idx) +
    lessCnt(tree[a].right,tree[b].ri
    ght,mid+1,r,idx); }
void init(int n,int m) {
     root[0] = tree[0].left =
    tree[0].right = tree[0].val = 0;
     for(int i=1;i<=n;i++)
        root[i] = update(root[i-1]
    , 1 , m , mp[a[i]]); }
```

Kth Element in Segment:

```
int findKth(int pos,int l,int r,int
    k) {
     if(l == r)
        return l;
     int mid = (l+r) >> 1;
     if(tree[pos*2] >= k)
        return
    findKth(pos*2,l,mid,k);
     else
        return
    findKth(pos*2+1,mid+1,r,
    k-tree[pos*2]); }
```

Sliding RMQ:

```
vector<int> slidingRMQ(int a[],int
    n,int k) {
     deque<int>d;
     vector<int>res;
     for(int i=0;i<n;i++) {
        while(!d.empty() &&
    d.front() >= a[i])
            d.pop_front();
        d.push_front(a[i]);
        if(i>=k && a[i-k] ==
    d.back())
            d.pop_back();
        if(i >= k-1)

    res.push_back(d.back()); }
     return res; }
```

Trie:

```
bool Check(ll n,ll pos) { return
    (n>>pos)&1; }
struct trieNode {
    trieNode *one,*zero;
    int cnt;
    trieNode() {
        one = zero = NULL;
        cnt = 0; }
};
trieNode *root;
void Insert(ll n) {
    trieNode *cur = root;
    for(ll i=50;i>=0;i--) {
        if(Check(n,i)) {
            if(!cur->one)
```

```
                cur->one = new
   trieNode();
            cur = cur->one; }
        else {
            if(!cur->zero)
                cur->zero = new
   trieNode();
            cur = cur->zero; }
        cur->cnt++; }
}
ll MaximumXor(ll n) {
    trieNode *cur = root;
    ll ret = 0;
    for(ll i=50;i>=0;i--) {
        if(Check(n,i)) {
            if(cur->zero &&
   cur->zero->cnt) {
                ret += (1LL<<i);
                cur = cur->zero; }
            else
                cur = cur->one; }
        else {
            if(cur->one &&
   cur->one->cnt) {
                ret += (1LL<<i);
                cur = cur->one; }
            else
                cur = cur->zero; }
    }
    return ret; }
```

Centroid Decomposition:

```
struct CentroidDecomposition{
    int path[MAX] , sub[MAX];
    bool vis[MAX];
    CentroidDecomposition(){
        memset(vis,0,sizeof vis);
        memset(path,0,sizeof path);
    }
    void subDFS(int src,int par){
        sub[src] = 1;
        for(auto i : adj[src]){
            if(i == par || vis[i])
                continue;
            subDFS(i,src);
            sub[src] += sub[i];
        }
    }
    int centroid(int src,int
par,int sz){
        for(auto i : adj[src])
        {
            if(i == par || vis[i])
                continue;
    else if(sub[i] > sz)
    return centroid(i,src,sz);
        }
        return src;
    }
    void decompose(int src,int
par){
        subDFS(src,-1);
        int c =
centroid(src,-1,sub[src]/2);
        vis[c] = 1;
        path[c] = par;
        for(auto i : adj[c]){
            if(!vis[i])
                decompose(i,c);
        }
    }
} tree;
bool color[MAX];
multiset<int>data[MAX];
struct QueryHandler {
    void update(int u) {
        color[u] ^= 1;
        int cur = u;
```

```
        while(cur != -1) {
            if(color[u])
    data[cur].insert(dist(u,cur));
            else
    data[cur].erase(data[cur].find(d
    ist(u,cur)));
            cur = tree.path[cur];
        }
     }
     int query(int u) {
        int cur = u , ret = 1e9;
        while(cur != -1) {
            if(data[cur].size())
                ret = min(ret ,
    *data[cur].begin() + dist(u,cur)
    );
            cur = tree.path[cur];
        }
        if(ret == 1e9)
            ret = -1;
        return ret;
     }
} ds;
```

Calls from main function :

```
DFS(1,1,0);
initLCA();
tree.decompose(1,-1);
ds.update(u);
ds.query(u)
```

Segment Tree Lazy

```
int arr[100005],lazy[262144],
segment_tree[262144];
void build(int low,int high,int
pos)  {
if(low==high)
{ segment_tree[pos]=arr[low];
        return ;   }
    int mid=(low+high)/2;
    build(low,mid,pos*2+1);
    build(mid+1,high,pos*2+2);
segment_tree[pos]=(segment_tree[2*p
   os+1]+segment_tree[2*pos+2]);
}
void lazypropagate(int low,int
high,int pos)   {
if(low!=high)
  {  lazy[pos*2+1]+=lazy[pos];
     lazy[pos*2+2]+=lazy[pos];
    }
segment_tree[pos]+=
        ((high-low)+1)*lazy[pos];
    lazy[pos]=0;
}
void update(int st,int en,int
   low,int high,int pos,int val)
{ if(lazy[pos]>0)
 lazypropagate(low,high,pos);
    if(st>high || en<low)
        return;
    if(st<=low && en>=high)
{     segment_tree[pos]=
                ((high-low)+1)*val;
      lazy[pos]+=val;
   if(low!=high)
   {  lazy[pos*2+1]+=lazy[pos];
      lazy[pos*2+2]+=lazy[pos];  }
      lazy[pos]=0;
      return ;   }
    int mid=(low+high)/2;
 update(st,en,low,mid,pos*2+1,val);
update(st,en,mid+1,high,pos*2+2,val
   );
segment_tree[pos]=(segment_tree[2*p
   os+1]+segment_tree[2*pos+2]);
}
```

```
int query(int st,int en,int low,int
high,int pos){
if(lazy[pos]>0)
   lazypropagate(low,high,pos);
 if(st>high || en<low)
   return 0;// no overlap
 if(st<=low && en>=high )
        return segment_tree[pos];
 int mid=(low+high)/2;
return
   query(st,en,low,mid,pos*2+1)+
query(st,en,mid+1,high,pos*2+2);
}
```

LCA:

```
vector<int>adj[MAX];
int dep[MAX] , T[MAX] , P[MAX][30];
void DFS(int src,int par,int lev){
    dep[src] = lev;
    T[src] = par;
    for(int i=0;
   i<adj[src].size();i++){
        int x = adj[src][i];
        if(x == par)
            continue;
        DFS(x,src,lev+1);}}
void initLCA(int n){
    memset(P,-1,sizeof P);
    for(int i=1;i<=n;i++)
        P[i][0] = T[i];
    for(int j=1; 1<<j <n;j++){
        for(int i=1;i<=n;i++){
            if(P[i][j-1] != -1)
                P[i][j] =
   P[P[i][j-1]][j-1];}}}
int query(int n,int u,int v){
    if(dep[u] < dep[v])
        swap(u,v);
    int log = 1;
    while(1){
        int next = log+1;
        if((1<<next) > dep[u])
            break;
        log++;}
    for(int i=log;i>=0;i--){
        if(dep[u]-(1<<i) >= dep[v])
            u = P[u][i];}
    if(u == v)
        return u;
    for(int i=log;i>=0;i--){
        if(P[u][i] != -1 && P[u][i]
   != P[v][i]){
            u = P[u][i];
            v = P[v][i];}}
    return T[u]; }
-> DFS(1,1,0);
-> initLCA(n);
```

HLD:

```
vector<int>adj[MAX];
int a[MAX];
int chainNo, ptr, chainHead[MAX],
   chainPos[MAX], chainIdx[MAX] ,
   sub[MAX] , maxSub[MAX];
int arr[MAX], tree[4*MAX];
int dep[MAX], T[MAX], P[MAX][20];
void init(const int& n){
    for(int i=1; i<=n; i++)
        adj[i].clear();
    chainNo = 0;
    ptr = 0;
    memset(chainHead,-1,sizeof
   chainHead);
    memset(tree,0,sizeof tree); }
void HLD(int cur,int par){
    if(chainHead[chainNo] == -1)
        chainHead[chainNo] = cur;
    chainIdx[cur] = chainNo;
```

```
    chainPos[cur] = ++ptr;
    arr[ptr] = a[cur];
    if(maxSub[cur] != -1)
        HLD(maxSub[cur],cur);
    for(int j=0; j<adj[cur].size();
   j++){
        int i = adj[cur][j];
        if(i != par && i !=
   maxSub[cur])
            chainNo++,
   HLD(i,cur);}}
int query_up(int u,int v,const int&
   n){
    int uchain , vchain =
   chainIdx[v] , ans = 0;
    while(chainIdx[u] != vchain){
        uchain = chainIdx[u];
        ans +=
   BIT_Query(chainPos[chainHead[uch
   ain]],chainPos[u],n);
        u = chainHead[uchain];
        u = P[u][0];}
    ans +=
   BIT_Query(chainPos[v],chainPos[u
   ],n);
    return ans;}
void ansUpdate(int i,int v,const
   int& n){

   BIT_Update(chainPos[i],-a[i],n);
    BIT_Update(chainPos[i],v,n);
    a[i] = v;}
int ansQuery(const int& n,int u,int
   v){
    int lca = LCA_query(n,u,v);
    int q1 = query_up(u,lca,n);
    int q2 = query_up(v,lca,n);
    return q1+q2-a[lca]; }
```

-> DFS(1,-1,0); -> BIT_UPDATE()
-> initLCA(n); -> ansUpdate();
-> HLD(1,-1) -> ansQuery();

MO's Algo:

```
struct data {
    int l,r,idx,k;
    bool operator<(const data &b)
   const {
        int x = l/BLOCK_SIZE, y =
   b.l/BLOCK_SIZE;
        if(x != y)
            return x < y;
        return r < b.r; } };
int BLOCK_SIZE;
ll cnt, ans[MAX];
ll n, q, a[MAX], freq[MAX];
data Q[MAX];
void add(ll x) {
    freq[x]++;
    if(freq[x] == 1)
        cnt++; }
void del(ll x) {
    freq[x]--;
    if(freq[x] == 0)
        cnt--; }
void MO() {
    BLOCK_SIZE = sqrt(n);
    sort(Q,Q+q,cmp);
    int st = 1, en = 0;
    for(int i=0; i<q; i++){
        int l = Q[i].l , r = Q[i].r
   , idx = Q[i].idx;
        while(en < r) { en++;
   add(a[en]); }
        while(en > r) { del(a[en]);
   en--; }
        while(st > l) { st--;
   add(a[st]); }
```

```
        while(st < l) { del(a[st]);
    st++; }
        ans[idx] = cnt; } }
```

---

string

KMP:

```
int lps[1000000],len1,len2;
vector<int>store;
void LPS_table(string pat)
{   int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < len2)
    { if (pat[i] == pat[len])
        {   len++;
            lps[i] = len;
            i++;
        }
        else
        {  if (len != 0)
                len = lps[len - 1];
            else
            {  lps[i] = 0;
                i++;
            }  }  }
}
void KMP_search(string s,string
   pat)
{   len1=s.length();
    len2=pat.length();
    LPS_table(pat);
    int i=0,j=0;
    while (i < len1)
    {  if (pat[j] == s[i])
        {   j++;
            I++;   }
       if (j == len2)
        {  store.push_back(i-j);
            j = lps[j - 1];  }
else if (i < len1 && pat[j] !=s[i])
        {  if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;  }  }
    }
```

Suffix Array:

```
const int LOGN = 20;
struct Info{
    int prev , now , pos; };
int sa[MAX] , P[LOGN][MAX] ,
   lcp[MAX] , logn , n;
Info L[MAX];
string s;
bool cmp(Info a,Info b) {
    if(a.prev == b.prev)
        return a.now < b.now;
    return a.prev < b.prev; }
bool cmp2(int i,int j) {
    return P[logn][i] < P[logn][j];
   }
void buildSuffixArray() {
    for(int i=0;i<n;i++)
        P[0][i] = s[i]-'a' , sa[i]
   = i;
    int gap = 1 , step = 1;
    while(gap < n) {
        for(int i=0;i<n;i++) {
            L[i].prev =
   P[step-1][i];
            L[i].now = (i+gap < n)
   ? P[step-1][i+gap] : -1;
            L[i].pos = i; }
        sort(L,L+n,cmp);
        for(int i=0;i<n;i++) {
```

```
            if(i && L[i].prev ==
   L[i-1].prev && L[i].now ==
   L[i-1].now)
                P[step][L[i].pos] =
   P[step][L[i-1].pos];
            else
                P[step][L[i].pos] =
   i; }
        step++ , gap *= 2; }
     logn = step-1;
     sort(sa,sa+n,cmp2); }
void buildLCP() {
    lcp[0] = 0;
    for(int i=1;i<n;i++) {
        int x = sa[i] , y =
   sa[i-1];
        lcp[i] = 0;
        for(int j=logn;j>=0 && x<n
   && y<n;j--) {
            if(P[j][x] == P[j][y])
   {
                lcp[i] += (1<<j);
                x += (1<<j);
                y += (1<<j); }
        } } }
-> n = s.size();
-> buildSuffixArray();
-> buildLCP();
```

Hashing:

```
const int base = 1337;
ll pw[300015],HASH[300015];
void pre_power()
{   pw[0] = 1;
    for(int i = 1; i < 300015; i++)
  pw[i] = (pw[i - 1] * base) % MOD;
}
ll get_hashval(string str)
{   int len=str.length();
    ll hash_val=0;
for(int i = 0; i < len; i++)
{hash_val=((hash_val*base)+str[i])%
   MOD;
 HASH[i+1]=hash_val; }
    return hash_val;
}
ll SubstringHash(int l, int r)
{  return (HASH[r]-(HASH[l-1]*
pw[r - l + 1]) % MOD + MOD) % MOD;
}
```

Z-Algorithm:

```
string S;
int z[MAX];
void zFunction() {
    int left , right;
    left = right = z[0] = 0;
    for(int i=1;i<S.size();i++) {
        if(i <= right)
            z[i] =
   min(z[i-left],right-i+1);
        while(i+z[i] < S.size() &&
   S[i+z[i]] == S[z[i]])
            z[i]++;
        if(i+z[i]-1 > right)
            left = i , right =
   i+z[i]-1; }
}
bool isSubstr(string t,string p) {
    S = p + "#" + t;
    zFunction();
    for(int
   i=p.size()+1;i<S.size();i++) {
        if(z[i] == p.size())
            return true; }
    return false; }
int countSubstr(string t,string p)
   {
```

```
    S = p + "#" + t;
    memset(z,0,sizeof z);
    zFunction();
    int cnt = 0;
    for(int
   i=p.size()+1;i<S.size();i++) {
        if(z[i] == p.size())
            cnt++; }
    return cnt; }
```

---

*GRAPHS

Max Flow:

```
struct Edge{
    int u , v , cap;
    Edge(){}
    Edge(int _u,int _v,int _cap){ u
   = _u , v = _v , cap = _cap; } };
vector<Edge>edges;
vector<int>adj[MAX];
bool vis[MAX];
int s,t;
void init(int _s,int _t){
    s = _s , t = _t;
    for(int i=0;i<MAX;i++)
        adj[i].clear();
    edges.clear(); }
void addEdge(int u,int v,int w){
    edges.push_back(Edge(u,v,w));
    edges.push_back(Edge(v,u,0));

   adj[u].push_back(edges.size()-2)
   ;

   adj[v].push_back(edges.size()-1)
   ;}
int pushFlow(int u,int flow = 1e9){
    vis[u] = 1;
    if(u == t)
        return flow;
    int ret = 0;
    for(int
   i=0;i<adj[u].size();i++){
        int idx = adj[u][i];
        Edge &e = edges[idx];
        if(!e.cap || vis[e.v])
            continue;
        ret =
   pushFlow(e.v,min(flow,e.cap));
        if(ret){
            Edge &rev =
   edges[idx^1];
            e.cap -= ret;
            rev.cap += ret;
            return ret;} }
    return ret; }
int maxFlow() {
    int ans = 0;
    while(1) {
        memset(vis,0,sizeof vis);
        int flow = pushFlow(s);
        if(flow == 0)
            break;
        ans += flow; }
    return ans; }
```

SCC:

```
vector < int > graph[MAX] ,
   reverseGraph[MAX] ,
   components[MAX];
bool vis[MAX];
int compCount;
stack<int>nodes;
void DFS(int src) {
    vis[src] = 1;
    for(auto i : graph[src]) {
        if(!vis[i])
```

```
            DFS(i); }
     nodes.push(src); }
void DFS2(int src) {
    vis[src] = 1;
    for(auto i : reverseGraph[src])
   {
        if(!vis[i])
            DFS2(i); }

   components[compCount].push_back(
   src); }
void init() {
    compCount = 1;
    for(int i=1;i<MAX;i++)
        graph[i].clear() ,
   reverseGraph[i].clear() ,
   components[i].clear(); }
void addEdge(int u,int v) {
    graph[u].push_back(v);
    reverseGraph[v].push_back(u); }
void kosaraju_SCC(int n) {
    memset(vis,0,sizeof vis);
    for(int i=1;i<=n;i++) {
        if(!vis[i])
            DFS(i); }
    memset(vis,0,sizeof vis);
    while(nodes.size()) {
        int top = nodes.top();
        nodes.pop();
        if(!vis[top]) {
            DFS2(top);
            compCount++; } } }
void print_SCCs() {
    for(int i=1;i<compCount;i++) {
        cout << "Component " << i
   << ":\n";
        for(auto j : components[i])
            cout << j << " -> ";
        cout << endl; } }
```

-> addEdge(u,v); -> print_SCCs();
-> kosaraju_SCC(n);

Longest path between two nodes in a graph:

```
int dfs(int src,int parent)
{   visit[src]=1;
    int path_cost=0;
for(int i=0; i<graph[src].size();
i++)
{int child=graph[src][i].first;
int child_cost=
graph[src][i].second;
 if(child!=parent)
   {   visit[child]=1;
int cur_cost=
dfs(child,src)+child_cost;
ans=max(ans,cur_cost+path_cost);
path_cost=max(path_cost,cur_cost);}
    }
    return path_cost;
}
```

Floyd Warshall:

```
int n,m;
int adj[MAX][MAX],path[MAX][MAX];
void floydWarsh() {
    for(int via=1; via<=n; via++) {
        for(int from=1; from<=n;
   from++) {
            for(int to=1; to<=n;
   to++) {

   if(adj[from][via]+adj[via][to] <
   adj[from][to]) {
                    adj[from][to] =
   adj[from][via]+adj[via][to];
                    path[from][to]
   = path[via][to];
```

```
                }
            } } }
}
```

BPM:

```
vector<int>graph[1000005];
bool visit[1000005];
int connection[1000005];
bool BPM(int node)
{  int sz=graph[node].size();
    for(int i=0; i<sz; i++)
    {  int child=graph[node][i];
        if(visit[child]==0)
  {   visit[child]=1;
  if(connection[child]<0 ||
   BPM(connection[child]))
  {  connection[child]=node;
     return true; } } }
    return false;
}
int maxBPM(int n)
{memset(connection,-1,sizeof(connec
   tion));
 int res=0;
for(int i=0; i<n; i++)
 { memset(visit,0,sizeof(visit));
       if(BPM(i))
           res++;  }
    return res;
}
```

Articulation point OR finding maximum number of connected components:

```
vector<int>graph[10050];
vector<int>entry_time,low_time;
int timer;
bool visited[10050];
set<int>s;
void dfs(int src,int parent)
{ visited[src]=1;
entry_time[src]=low_time[src]
=timer++;
 int children=0;
for(int i=0; i<graph[src].size();
   i++)
{   int child=graph[src][i];
    if(child==parent)
      continue;
   if(visited[child]==1)

   low_time[src]=min(low_time[src],
   entry_time[child]);
     else
     {dfs(child,src);
low_time[src]=min(low_time[src],low
   _time[child]);
if(low_time[child]>=entry_time[src]
   && parent!=-1)
  {s.insert(src);}
     children++;
        }
    }
if(parent == -1 && children > 1)
        s.insert(src);
}
void find_bridges(int n)
{   s.clear();
    fill(visited,visited+10020,0);
    entry_time.assign(n+2, -1);
    low_time.assign(n+2, -1);
    timer=0;
    for(int i=1; i<=n; i++)
        if(visited[i]==0)
            dfs(i,-1);
    for(int i=1; i<=n; i++)
        graph[i].clear();
}
```

Krushkal's algorithm:

```
pair<int,pair<int,int> >mst[1000];
int parent[1000];
int root(int child)
{while(parent[child]!=child)
{parent[child]=
          parent[parent[child]];
 child=parent[child];  }
    return child;
}
void node_union(int x,int y)
{   int a=root(x);
    int b=root(y);
parent[min(a,b)]=parent[max(a,b)];
}
int kruskal(int nodes,int edges)
{for(int i=0; i<=nodes; i++)
        parent[i]=i;
sort(mst,mst+edges);
int x,y,cost,min_cost=0,i;
for(i=0; i<edges; i++)
{  x=mst[i].second.first;
   y=mst[i].second.second;
   cost=mst[i].first;
if(root(x)!=root(y))
{ min_cost+=cost;
  node_union(x,y); }  }
    return min_cost;
}
```

*Number Theory

Euler Phi Sieve:

```
long long phi[10000000];
void computeTotient(int n)
{  for (int i=1; i<=n; i++)
        phi[i] = i;
   for (int j=2; j<=n; j++)
    {  if (phi[j] == j)
    {  phi[j] = j-1;
for (int i = 2*j; i<=n; i += j)
{phi[i] = (phi[i]/j) * (j-1);}
   }    }
}
```

Extended GCD:

```
ll extendedGCD(ll a,ll b,ll *x,ll
   *y) {
    if(a == 0) {
        *x = 0 , *y = 1;
        return b; }
    ll x1 , y1;
    ll gcd =
   extendedGCD(b%a,a,&x1,&y1);
    *x = y1 - (b/a)*x1;
    *y = x1;
    return gcd; }
ll modInverse(ll a,ll M) {
if(__gcd(a,M) > 1) return -1;
    ll x , y;
ll gcd = extendedGCD(a,M,&x,&y);
    return (x+M)%M; }
int main() {
    cout << modInverse(3,11) <<
   endl;
    /// ans = 4 , because (4*3)%11
   = 12%11 = 1 }
```

NCR:

```
ll fact[1000005],inverse[1000005];
ll Bigmod(int a,int b)
{if(b==0)
     return 1%MOD;
    ll x=Bigmod(a,b/2);
    x=(x*x)%MOD;
 if(b%2==1)
    x=(x*a)%MOD;
    return x;
```

```
}
ll nCr(int x, int y)
{  if(x<0 || y<0 || x<y)
        return 0;
    return fact[x] * (inverse[y] *
   inverse[x - y] % MOD) % MOD;
}
int main()
{  fact[0]=1;
  for(int i=1; i<=1000000; i++)
  fact[i]=fact[i-1]*1LL*i%MOD;
inverse[1000000]=Bigmod(fact[100000
   0],MOD-2);
for(int i=1000000; i>0; i--)
{inverse[i-1]=i* 1LL*inverse[i] %
   MOD;
 printf("%lld",nCr(45,2));
    return 0;
}
```

NCR with DP:

```
ll nCr(ll n, ll r)
{   ll C[r+1];
    memset(C, 0, sizeof(C));
    C[0] = 1;
for (ll i = 1; i <= n; i++){
for (ll j = min(i, r); j > 0; j--)
C[j] = (C[j] + C[j-1])%m; }
    return C[r];
}
```

Sum of divisors between a to b:

```
ll triangle(ll a, ll b) { return (a
   + b + 1) * (b - a) / 2 ; }
ll divSum(ll a, ll b) {
 ll n = sqrt(b);
 ll sum = 0;
 for (ll i = 1; i <= n; i++)
 sum += i*(b/i - a/i) +
   triangle(max(n,a/i),max(n,b/i));
 return sum; }
```

Sum of divisors from 1 To N:

```
long long int sod(long long int a)
{  long long int i=1,q,m,k,sum=0;
    q=sqrt(a);
    while(i<=q)
    {  sum+=(i*(a/i));
        i+=1;
    }
    i=1;
    while(i<=(a/(q+1)))
    {  m = a/i;
        k =a/(i+1);
        sum+=(i*(m*(m+1)- k*
   (k+1))/2);
        i += 1;
    }  return sum;
}///TC:O(sqrt(a))
```

Phi for a large number:

```
int phi(int n) {
    double res = n;
    for(int i=2;i*i<=n;i++) {
        if(n%i == 0) {
            while(n%i == 0)
                n /= i;
            res *= (1.0 - (1.0/i));
        } }
    if(n > 1)
        res *= (1.0 - (1.0/n));
    return (int)(res); }
```

Divisors of N!:

```
ll factorialDivisors(ll n) {
    ll res = 1;
    for(int i=0;primes[i]<=n;i++) {
```

```
        ll exp = 0;
        ll p = primes[i];
        while(p <= n) {
            exp += (n/p);
            p *= primes[i]; }
        res *= (exp+1);
    }
    return res; }
```

Trailing Zeroes of N!:

```
int trailingZeroes(int n) {
    int cnt = 0 , f = 5;
    while(f <= n) {
        cnt += n/f;
        f *= 5;
    }
    return cnt;}
```

Digits in N!:

```
int findDigits(int n) {
    if(n<=1)
        return n;
    double digits = 0;
    for(int i=2;i<=n;i++)
        digits += log10(i);
    return floor(digits)+1; }
```

N! Under Modulo P:

```
int largestPower(int n,int p) {
    int cnt = 0;
    while(n) {
        n /= p;
        cnt += n;
    }
    return cnt; }
int fact(int n,int p) {
    int res = 1;
    for(int i=0;primes[i]<=n;i++) {
        int k =
   largestPower(n,primes[i]);
        res =
   (res*bigMod(primes[i],k,p))%p;
    }
    return res; }
```

SOD of n^m:

```
ll primeFact(ll n,int m) {
    ll sum = 1;
    for(int i=0; i<primes.size() &&
   primes[i]<=n; i++) {
        ll cnt = 0, p = primes[i];
        if(n%p == 0) {
            while(n%p == 0)
                cnt++ , n /= p;
            cnt = cnt*m+1;
            ll calc =
   (bigMod(p,cnt,MOD)+MOD-1)%MOD;
            calc *=
   bigMod(p-1,MOD-2,MOD);
            calc %= MOD;
            sum = (sum*calc)%MOD; }
    }
    if(n > 1) {
        ll calc =
   (bigMod(n,1+m,MOD)+MOD-1)%MOD;
        calc *=
   bigMod(n-1,MOD-2,MOD);
        calc %= MOD;
        sum = (sum*calc)%MOD; }
    return sum; }
```

MatExpo:

```
struct Matrix{
    int n,m;
    vector< vector<int> > mat;
    Matrix() {}
    Matrix(int _n,int _m) {
```

```
        n = _n, m = _m;
        mat = vector< vector<int> >
   (n,vector<int>(m)); } };
Matrix multiply(Matrix a,Matrix
   b,int MOD) {
    Matrix c = Matrix(a.n,b.m);
    for(int i=0; i<a.n; i++) {
        for(int j=0; j<b.m; j++) {
            c.mat[i][j] = 0;
            for(int k=0; k<a.m;
   k++) {
                c.mat[i][j] += (1LL
   * a.mat[i][k] *
   b.mat[k][j])%MOD;
                if(c.mat[i][j] >=
   MOD)
                    c.mat[i][j] -=
   MOD; } } }
    return c; }
Matrix pow(Matrix a,ll p,int MOD) {
    if(p == 1)
        return a;
    Matrix x = pow(a,p/2,MOD);
    x = multiply(x,x,MOD);
    if(p&1)
        x = multiply(x,a,MOD);
    return x; }
Matrix createA() {
    Matrix a = Matrix(2,2);
    .........
    return a; }
Matrix createB(int a,int b) {
    Matrix bb = Matrix(2,1);
    .........
    return bb; }
void print(Matrix A) {
    for(int i=0; i<A.n; i++)
        for(int j=0; j<A.m; j++)
            cout << A.mat[i][j] <<
   " \n"[j ==A.m-1]; }
->Matrix M=pow(createA(),n-2,MOD);
->ans=multiply(M,createB(),MOD).mat
   [0][0];
```

---

*Dynamic programming

Longest Increasing Subsequence:

```
int lis[MAX],store_lis[MAX];
int LIS(int n)
{ for(int i=0; i<n; i++)
        lis[i]=INF;
    int pos, cnt = 0 ;
    lis[0] = -INF ;
    lis[n] = INF ;
for (int i = 0 ; i < n ; i++ )
{ pos = lower_bound( lis, lis+n+1,
             arr[i] ) - lis;
  lis[pos] = arr[i] ;
  cnt = max ( cnt, pos );
  store_lis[i] = cnt ;
    }
return cnt;
}
```

Longest Decreasing Subsequence:

```
int lds[MAX],store_lds[MAX];
int LDS(int n)
{ for(int i=0; i<n; i++)
        lds[i]=INF;
    int pos, cnt= 0 ;
    lds[0] = -INF ;
    lds[n] = INF;
 for(int i = n-1;i>=0;i--)
 {pos = lower_bound( lds, lds+n+1,
            arr[i] ) - lds;
  lds[pos] = arr[i] ;
  cnt = max ( cnt, pos );
```

```
  store_lds[i] = cnt ;
  }
return cnt;
}
```

Longest Non-Decreasing subsequence:

```
int lnds[MAX];
int LNDS(int n)
{  lnds[1]=arr[1]; //1 base index
    int len = 1 ;
for(int i = 2; i<=n;i++)
 { if(arr[i]>=lnds[len])
      lnds [++ len] = arr [i];
   else{
 int j=upper_bound(lnds+1,lnds+len+
                  1,arr[i])-lnds;
    lnds [j] = arr [i];  }   }
    return len;
}
```

Longest Non-Rising Subsequence:

```
->change in Lnds:
1.if(arr[i]<=lnds[len]
2.int j=upper_bound(lnds+1,lnds+len
   +1,arr[i],greater<int>())-lnds;
```

SOS DP:

```
for(int i = 0; i < (1 << N); ++i)
{   F[i] = A[i];  }
for(int i = 0; i < N; ++i)
{ for(int mask=0;mask<(1<<N);
   ++mask)
{ if(mask & (1 << i))
  {F[mask] += F[mask ^ (1 << i)]; }
    }
}
```

Meet in the middle:

```
ll X[2000005],Y[2000005];
void calcsubarray(ll a[], ll x[],
int n, int c)
{ for (int i=0; i<(1<<n); i++)
{  ll s = 0;
for (int j=0; j<n; j++)
   if (i & (1<<j))
    s += a[j+c];
        x[i] = s;  }   }
ll SSsum(ll a[],int n,ll S)
{  calcsubarray(a, X, n/2, 0);
   calcsubarray(a, Y, n-n/2, n/2);
   int size_X = 1<<(n/2);
   int size_Y = 1<<(n-n/2);
   sort(Y, Y+size_Y);
   ll mx = 0;
   for (int i=0; i<size_X; i++)
   { if (X[i] <= S)
{int p =lower_bound(Y,Y+size_Y,
        S-X[i])-Y;
if(p==size_Y || Y[p]!=(S-X[i]))
        p--;
 if ((Y[p]+X[i]) > mx)
   mx = Y[p]+X[i];  }  }
   return mx;
}
```

*Geometry

Triangle:

***To form a+b>c,b+c>a,a+c>b**

***Check if 3 points form triangle:**

```
|(x2-x1)(y3-y1)-(y2-y1)(x3-x1)| > 0
```

Perimeter: p = a+b+c

Area(A):(½) ab

A:abSinC/2

A:(Ax(By-Cy)+Bx(Cy-Ay)+Cx(Ay-By)/2

s = p/2

A:sqrt(s*(s-a)*(s-b)*(s-c))

A: bh/2
Area for Equi:s^2(sq(3)/4)
Pythagoras: a^2+b^2 = c^2
->sinC=h/2
Isosceles:
Base:2sq(L^2-A^2)
L=sq(A^2+(B/2)^2)
Area=BL/2
Base=L*sin(theta)
SineRule: a/SinA=b/Sinb=c/SinC
CosineRule: a^2=b^2+c^2-2bcCosA
Angle: cosA = (b^2+c^2-a^2)/2bc
Centre:
x=(x1+x2+x3)/3,y=(y1+y2+y3)/3
Median:AD=sq((2b^2+2c^2-a^2)/4)
Centroid:AG=sq(2b^2-2c^2-a^2)/3
Inradius:2A/P
Inradius for equi:S/2sq(3)
Circumradius->
Equilateral Tri:S/sq(3)
Radius:abc/
sq((a+b+c)(a+b-c)(a+c-b)(b+c-a))
Diameter:a/sinA (if side and opposite angle is known)

Circle:

Distance: sqrt((x2-x1)^2 + (y2-y1)^2)

Check if 3 points are in same line:
x1*(y2-y3)-x2(y1-y3)+x3(y1-y2) = 0
Find a circle that covers 2 given:
x3 = (x1+x2)/2 , y3 = (y1+y2)/2
r = dist(x1,y1,x2,y2)
Lattice Points:
1 + gcd(|x1-x2|,|y1-y2|)
Slope formed by 2 points:
(y2-y1) / (x2-x1)
Area of sector of circle: ½ r^2*θ
Arc Length: r*θ

Parallelogram:
Given 3 points find 4th point:
Dx = Ax + (Cx-Bx)
Dy = Ay + (Cy-By)
Area: |½((Ax*By+Bx*Cy+Cx*Dy+Dx*Ay)
-(Ay*Bx + By*Cx + Cx*Dx + Dy*Ax))|

Trapezium:
Area:(a+b)/(a-b) * sqrt((s-a)(s-b)
(s-b-c)(s-b-d))
-> s = (a+b+c+d)/2
-> a = long parallel side
-> b = short parallel side
-> c,d = non-parallel side
Area:h*((b1+b2)/2)
H:
sq(b^2-(b^2-d^2+(a-c)^2)/2(a-c))^2)

Right Circular Cone:
Volume:(pie*h/3)*(R^2+R*r+r^2)
Lateral surface Area:pie(r+R)*S
Area of the base:pie*r^2
Lateral area:½*c*L ..L=slant height
=pie*r*L
Total Surface A:pie*r^2+pie*r*s
Volume:⅓*pie*r^2*h
s=sq(r^2+h^2)
Polygon:
*After 4 sides the degree increases By 180 with each new side.
*Area of the largest square inside a pentagon->
s*(sin(108)/(sin(18)+sin(36)))
Area:(s^2*n)/4*tan(180/4)
Area:(r^2*n*sin(360/n))/2
Area:Apo^2*n*tan(180/n)
Area:¼*sq(5*(5+2sq(5)))*s^2

Area:(Apo*s)/2
Area:pr/2
Area:½*n*sin(360/n)*s^2
Area:n*apo^2*tan(180/n)
Area:(n*r*sin(2*(180/n)))/2
Area:(¼*n*s^2)/tan(180/n)
Perimeter:5*s
Diagonal:(s*(1+sq(5)))/2
Height:(s*sq(5+2sq(5)))/2
an:2*R*sin(180/n) ..here:an=side of regular inscribed polygon,R=radius of circumscribed circle.
Sum of interior angles of a Convex polygon:180(n-2)
Exterior taken one at each vertex:360
measurement of Exterior Ang:360/n
Measure Interior An:((n-2)*180)/n
No. Of Dia:(n*(n-3))/2
No. Of Tri:N-2
Side:2*R*sin(180/n)
Apo:R*cos(180/n)
Side:2*apo*tan(180/n)
Area of smallest tri:½*apo*(s/2)
½*Apo^2*tan(180/n)
Truncated Cone:
z=(H*r2^2)(r1^2-r2^2)
R=(½*r1^2(z+h))/(H+z)
Volume:⅓*pie*h*(R^2+(R*r2)+r2^2)
Volume of a cylinder: pi × r 2 × Height
Volume of a triangular prism: area of triangle × Height = (1/2 base × height) × Height
Combinatorics:
Summation of squares of n natural numbers: (n*(n+1)*(2n+1))/6
C(n,r): n! / (r! * (n-r)!)
C(n,r): (n*(n-1)*..*(n-r+1)) / r!
P(n,k): n! / (n-k)!
-> nCk = nCn-k
-> Ways to go from (0,0) to (r,c): (r+c)Cr or (r+c)Cc
-> Ways to go from (0,0,0) to (x,y,z): (x+y+z)Cx * (y+z)Cy
-> a1+a2+.+an = k , ai >= 0: C(k+n-1,n-1)
-> Catalan Numbers: C(n) = (2n)! / ((n+1)! * r!)

Others:
Number of digits: log10(n)+1
Depth of road water: (s^2-h^2)/2h

```
//sum of series n/1+n/2+n/3+....n/n
ll root=sqrt(n);
for(int i=1; i<=root; i++)
    sum+=n/i;
sum=(2*sum)-(root*root);
```

count the numbers that are divisible by given number in a certain range:a=2,b=3,c=7;

```
low=(a+b-1)/a;
high=c/a;
total=high-low+1;
```

Euler Constant:γ≈0.5772156649
#Number of squares in a n*n grid:

```
S=(n*(n+ 1)*(2*n + 1))/6;
```

#Number of rectangle in a n*n grid:

```
R=(n+1)*n/2*(n+1)*n/2 - S;
```

#Total number of rectangle and square in a n*n grid:

```
ans=[(n^2 + n)^2]/4
```

#Number of squares in a n*m grid exp:6*4

```
S=6*4+5*3+4*2+3*1=50
```

#Number of rectangles in n*m grid

```
R=m(m+1)n(n+1)/4
```

#Number of cubes in a n*n*n grid formula:n^k-(n-2)^k
C=n*(n+1)/2*n*(n+1)/2;
#Number of boxes in a n*n*n grid:
B=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2-C;
#Number of hypercube in a n^4grid:
start a loop from 1 to <=n;
HC=0;
for(i=1;i<=n;i++)
HC+=i*i*i*i;
#Number of hyper box in a n^4 grid:
HB=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2*(n+1)*n/2 - HC;
Templates:

```
struct Point
{  double x,y;
   Point() {}
Point(double x, double y):x(x),y(y) {}
Point(const Point &p):x(p.x),y(p.y) {}
void input()
 { scanf("%lf%lf",&x,&y);  }
Point operator + (const Point &p) const
{ return Point(x+p.x, y+p.y); }
Point operator - (const Point &p) const
{ return Point(x-p.x, y-p.y); }
Point operator * (double c) const
{ return Point(x*c, y*c); }
Point operator / (double c) const
{ return Point(x/c, y/c); }
};
vector<Point>polygon;
double getClockwiseAngle(Point p)
{ return -1*atan2(p.x, -1 * p.y);}
//compare function to compare clockwise
bool comparePoints(Point p1,Point p2)
{ return getClockwiseAngle(p1) < getClockwiseAngle(p2); }
// rotate 90 degree counter clockwise
Point RotateCCW90(Point p)
{ return Point(-p.y,p.x); }
// rotate 90 degree clockwise
Point RotateCW90(Point p)
{ return Point(p.y,-p.x);}
Point RotateCCW(Point p, double t)
{
return Point(p.x*cos(t)-p.y*sin(t),
p.x*sin(t)+p.y*cos(t)); }
Point RotateCW(Point p, double t)
{ returnPoint(p.x*cos(t)+p.y*
sin(t),-p.x*sin(t)+p.y*cos(t));}
double dot(Point A, Point B)
{ return A.x*B.x+A.y*B.y;}
double cross(Point A, Point B)
{ return A.x*B.y-A.y*B.x; }
double dist2(Point A, Point B)
{ return dot(A-B,A-B); }
// returns distance between two point
double dist(Point A, Point B)
{ return sqrt(dot(A-B,A-B)); }
// Distance between point A and B
double distBetweenPoint(Point A, Point B)
{ return sqrt(dot(A-B,A-B)); }
// project point c onto line AB (A!=B)
Point ProjectPointLine(Point A, Point B, Point C)
{  returnA+(B-A)*
dot(C-A,B-A)/dot(B-A,B-A);}
```

```
// Determine if Line AB and CD are
   parallel or collinear
bool LinesParallel(Point A, Point
   B, Point C, Point D)
{return fabs(cross(B-A,C-D))<EPS;}
// Determine if Line AB and CD are
   collinear
bool LinesCollinear(Point A, Point
   B, Point C, Point D)
{ return LinesParallel(A,B,C,D) &&
   fabs(cross(A-B,A-C))<EPS &&
   fabs(cross(C-D,C-A))<EPS;}


//checks if AB intersect with CD
bool SegmentIntersect(Point A,
   Point B, Point C, Point D)
{ if(LinesCollinear(A,B,C,D))
 { if(dist2(A,C)<EPS ||
   dist2(A,D)<EPS || dist2(B,C)<EPS
   || dist2(B,D)<EPS)
  return true;
if(dot(C-A,C-B) > 0 && dot(D-A,D-B)
   > 0 && dot(C-B,D-B) > 0)
  return false;
       return true;
 }if(cross(D-A,B-A) *
   cross(C-A,B-A) > 0)
       return false;
  if(cross(A-C,D-C) *
   cross(B-C,D-C) > 0)
       return false;
    return true;
}
// Compute the coordinates where AB
and CD intersect
Point ComputeLineIntersection(Point
   A, Point B, Point C, Point D)
{   double a1,b1,c1,a2,b2,c2;
    a1=A.y-B.y;
    b1=B.x-A.x;
    c1=cross(A,B);
    a2=C.y-D.y;
    b2=D.x-C.x;
    c2=cross(C,D);
    double Dist=a1*b2-a2*b1;
    return
   Point((b1*c2-b2*c1)/Dist,(c1*a2-
   c2*a1)/Dist);
}
//Project point C onto line segment
   AB -- return the Point from AB
   which is the closest to C --
Point ProjectPointSegment(Point A,
   Point B, Point C)
{ double r=dot(B-A,B-A);
    if(fabs(r)<EPS)
        return A;
    r=dot(C-A,B-A)/r;
    if(r<0)
        return A;
    if(r>1)
        return B;
    return A+(B-A)*r;
}
// return the minimum distance from
a point C to a line AB
double DistancePointSegment(Point
   A, Point B, Point C)
{  return distBetweenPoint
(C,ProjectPointSegment(A,B,C)); }
// return distance between P and a
point where p is perpendicular on
AB. AB er upore p jei point e lombo
shei point theke p er distance
double distToLine(Point p, Point a,
   Point b)
{  pair<double,double>c;
```

```
double scale=(double)
(dot(p-a,b-a))/(dot(b-a,b-a));
c.first=a.x+scale*(b.x-a.x);
c.second=a.y+scale*(b.y-a.y);
double dx=(double)p.x-c.first
double dy=(double)p.y-c.second;
   return sqrt(dx*dx+dy*dy); }
long long orientation(Point p,
   Point q, Point r)
{ long long val=(q.y-p.y)*(r.x-q.x)
-(q.x-p.x)*(r.y-q.y);
    if (val > 0) return 1;
    if (val < 0) return 2;
    else  return val; }
// Given three colinear points p,
q, r, the function checks if
// point q lies on line segment
'pr'
bool onSegment(Point p, Point q,
   Point r)
{if (q.x<=max(p.x,r.x) && q.x>=
   min(p.x, r.x) && q.y <= max(p.y,
   r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}
//checks if Point P is inside of
polygon or not
bool isInside(int n, Point p)
{ if (n < 3) return false;
 Point extreme = Point(INF, p.y);
   // here INF=1e4
int count = 0, i = 0;
  do
   {   int next = (i+1)%n;
if (SegmentIntersect(polygon[i],
   polygon[next], p, extreme))
     { if (orientation(polygon[i],
   p, polygon[next]) == 0)
return onSegment(polygon[i], p,
   polygon[next]);
          count++;
     }
        i = next;
   }
    while (i != 0);
    return count&1;
}
// returns the perimeter of a
polygon
double polygonPerimeter(int n)
{ double perimeter = 0.0;
for (int i = 0; i < n - 1; i++)
   //polygon vector holds the
   corner points of the given
   polygon
 perimeter += dist(polygon[i],
   polygon[i + 1]);
 perimeter += dist(polygon[0],
   polygon[n - 1]);
    return perimeter; }
//returns the area of a polygon
double polygonArea(int n)
{  double area = 0.0;
    int j = n - 1;
 for (int i = 0; i < n; i++)
{area +=(polygon[j].x+polygon[i].x)
*(polygon[j].y-polygon[i].y);
        j = i;   }
    return fabs(area)*0.5;  }
double getTriangleArea(Point a,
   Point b, Point c)
{return fabs(cross(b - a, c - a));}
bool compareConvex(Point X , Point
   Y)
{long long ret =
orientation(points[0],X,Y);
if(ret==0)
```

```
{ll dist11 = dist2(points[0],X);
ll dist22 = dist2(points[0],Y);
  return dist11 < dist22 ; }
  else if(ret==2) return true ;
  else return false ; }
Point nextToTop(stack<Point> &S)
{   Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// make a minimum area polygon
stack<Point> convexHull(int N)
{ int ymin=points[0].y,index = 0 ;
for(int i=1;i<N;i++)
{if(points[i].y<ymin||(points[i].y
==ymin&&points[i].x<points[index].x
   ))
      { ymin = points[i].y ;
      index = i ;}  }
 stack<Point>S;
 swap(points[0],points[index]);
sort(&points[1],&points[N],compareC
   onvex);
S.push(points[0]);
for(int i=1;i<N;i++)
{while(S.size()>1&&orientation(next
   ToTop(S),S.top(),points[i])!=2){
           S.pop(); }
        S.push(points[i]); }
    return S;  }
// Angle between Line AB and AC in
   degree
double angle(Point B, Point A,
   Point C)
{  double c=dist(A,B);
    double a=dist(B,C);
    double b=dist(A,C);
double ans=acos((b*b+c*c-a*a)
/(2*b*c));
return (ans*180)/acos(-1);  }
// returns number of vertices on
   boundary of a polygon
ll picks_theorem_boundary_count()
{  int sz=polygon.size(), i ;
    long long res=__gcd((long
   long)abs(polygon[0].x-polygon[sz
   -1].x),(long
   long)abs(polygon[0].y-polygon[sz
   -1].y));
 for ( i = 0; i < sz-1 ; i++ )
 { res += __gcd((long
long)abs(polygon[i].x-polygon[i+1].
   x),(long
   long)abs(polygon[i].y-polygon[i+
   1].y));
    }
    return res;}
// picks theorem
// Polygon area= inside points +
   boundary points/2 -1
// return inside points counts
ll lattice_points_inside_polygon()
{ ll ar=polygonArea(n);
  ll b=
picks_theorem_boundary_count();
    long long tot=ar+1-b/2;
    return tot; }
```

rectangle sum in ranges:

```
#define  ms0(s) memset(s,0,sizeof
   s)
ll bit[SZ][SZ],data[SZ][SZ],R, C;
void Update(ll r, ll c, ll val)
{for(ll i=r;i<=R;i+=i&-i)
for(ll j=c;j<=C;j+=j&-j)
   bit[i][j] += val;  }
ll Sum(ll r,ll c)
{  ll i,j,s = 0;
for (i = r; i > 0; i &= i - 1)
for (j = c; j > 0; j &= j - 1)
     s += bit[i][j];
    return s; }
int main()
{   R = C = n;
    ms0(bit);
    ms0(data);
if(!strcmp(s,"SET"))
 {   int r,c,val;
 scanf("%d %d %d",&r,&c,&val);
        r++,c++;
Update(r, c, -data[r][c] + val);
        data[r][c] = val;  }
else if(!strcmp(s,"SUM"))
{  int r1,c1,r2,c2;
   TAKEINPUT
  r1++,c1++,r2++,c2++;
  int res =0;
  res+=Sum(r2, c2) ;
  res-=Sum(r1 - 1, c2);
  res-=Sum(r2, c1 - 1);
  res+=Sum(r1 - 1, c1 - 1);
  printf("%d\n",res);}
```

Segmented Sieve:

```
vector < int > primes;
void sieveOfEratosthenes()
{  bool flag[mx+1];
for(int i=0 ; i<=mx ; i++)
        flag[i]=true;
    primes.push_back(2);
 flag[0]=flag[1]=false;
for(int i=4 ; i<=mx ; i+=2)
        flag[i]=false;
 for(int i=3 ; i<=mx ; i+=2)
 {if(flag[i]==true) /// i is prime
   {primes.push_back(i);
  for(int j=i+i ; j<=mx ; j=j+i)
 { flag[j]=false; ///j is not prime
           }  }  }  }
void segmentedSieve(ll L, ll R)
{  bool isPrime[R-L+1];
    for(int i=0 ; i<=R-L+1 ; i++)
        isPrime[i]=true;
if(L==1)
        isPrime[0]=false;
for(int i=0;primes[i]*primes[i]<=R;
   i++)
{ ll curPrime=primes[i];
  ll base=curPrime*curPrime;
if(base<L)
{base=((L+curPrime-1)/curPrime)*cur
   Prime;  }
for(ll j=base ; j<=R ; j+=curPrime)
        isPrime[j-L]=false;
 }
for(int i=0 ; i<=R-L ; i++)
{  if(isPrime[i]==true)
    cout<<L+i<<endl;  } }
```

Bridge Searching:

```
vector<int>graph[10050];
vector<int>entry_time,low_time;
int timer;
bool visited[10050];
vector<pair<int,int> >bridge;
void dfs(int src,int parent)
{visited[src]=1;
```

```
entry_time[src]=low_time[src]=timer
   ++;
for(int i=0;i<graph[src].size();
i++)
{int child=graph[src][i];
    if(child==parent)
            continue;
 if(visited[child]==1)
low_time[src]=min
(low_time[src],entry_time[child);
else{dfs(child,src);
low_time[src]=min
(low_time[src],low_time[child]);
if(low_time[child]>entry_time[src])
bridge.push_back(make_pair(src,chil
   d));
        }  }  }
void find_bridges(int n)
{bridge.clear();
    fill(visited,visited+10020,0);
    entry_time.assign(n, -1);
    low_time.assign(n, -1);
    timer=0;
    for(int i=0; i<n; i++)
        if(visited[i]==0)
            dfs(i,-1);
 sort(bridge.begin(),bridge.end());
    int sz=bridge.size();
    printf("%d critical
   links\n",sz);
    for(int i=0; i<sz; i++)
        printf("%d -
   %d\n",bridge[i].first,bridge[i].
   second);
    for(i=0;i<n;i++)
        graph[i].clear();
}
```

Fast I/O:

```
#define pc putchar
#define fastread() (ios_base::
   sync_with_stdio(false),cin.tie(N
   ULL));
void Cin(ll &num) {
    num = 0;
    char ch = gc();
    ll flag = 0;
    while(!((ch >= '0' & ch <= '9')
   || ch == '-')) {
        ch = gc();}
    if(ch == '-') {
        flag = 1;
        ch= gc();}
    while(ch >= '0' && ch <= '9') {
        num = (num << 1) + (num <<
   3) + ch - '0';
        ch = gc();}
    if(flag == 1) {
        num = 0 - num;}}
void Cout(ll n)
{ ll num=n,rev=n,cnt=0;
    char ch;
    if(n==0)
    {  pc('0');
        return ;}
    while(rev%10==0)
    { cnt++;rev/=10;
    }
    rev=0;
    while(num>0)
    { rev= (rev<<3) + (rev<<1) +
   num%10;
        num/=10;}
    while(rev>0)
    {ch=(rev%10)+'0';
        pc(ch);
        rev/=10;}
    while(cnt--)pc('0');}
```