Hashing

```
constexpr int mod1 = 1000012253;
constexpr int mod2 = 1000000009;
template<typename T>
class MultiHashing {
public:
int n;
string s;
string rev;
vector<int> bases = {37,61,79,83,97,53,61,
107,127,137,163,191 ,
199,211,229,239,263 ,
271,281,293 }; T base1;
T base2;
// T mod1;
// T mod2;
vector<pair<T, T>> prefix_hash;
vector<pair<T, T>> suffix_hash;
vector<pair<T, T>> power;
vector<pair<T, T>> inv;
T rng() {
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<T>
dis(0, numeric_limits<T>::max());
return dis(gen);
}
T mul(T a, T b, T mod) {
return ((1LL * a % mod) * (b % mod)) % mod;
}
T add(T a, T b, T mod) {
return (1LL * a + b) % mod;
}
T sub(T a, T b, T mod) {
return ((a % mod) - (b % mod) + 2LL * mod) % mod;
}
T bigmod(T base, T power, T mod) {
T res = 1;
while (power > 0) {
if (power & 1) {
res = mul(res, base, mod);
}
base = mul(base, base, mod);
power >>= 1;
}
return res;
}
MultiHashing(const string& str) : s(str) {
n = s.size();
base1 = bases[rng() % bases.size()];
base2 = bases[rng() % bases.size()];
rev = s;
reverse(rev.begin(), rev.end());
// mod1 = mods[rng() % mods.size()];
// mod2 = mods[rng() % mods.size()];
prefix_hash.resize(n + 1, {0, 0});
suffix_hash.resize(n + 1, {0, 0});
power.resize(n + 1, {0, 0});
inv.resize(n + 1, {0, 0});
precom();
}
void precom() {
power[0] = {1, 1};
for (int i = 1; i <= n; i++) {
power[i].first = mul(power[i - 1].first, base1, mod1);
power[i].second = mul(power[i - 1].second, base2, mod2);
}
T inv_base1 = bigmod(base1, mod1 - 2, mod1);
T inv_base2 = bigmod(base2, mod2 - 2, mod2);
inv[0] = {1, 1};
for (int i = 1; i <= n; i++) {
inv[i].first = mul(inv[i - 1].first, inv_base1, mod1);
inv[i].second = mul(inv[i - 1].second, inv_base2, mod2);
}
for (int i = 1; i <= n; i++) {
int ch = s[i - 1] - 'a' + 1;
prefix_hash[i].first = add(prefix_hash[i - 1].first, mul(ch,
power[i - 1].first, mod1), mod1);
prefix_hash[i].second = add(prefix_hash[i - 1].second, mul(ch,
power[i - 1].second, mod2), mod2);
ch = rev[i - 1] - 'a' + 1;
suffix_hash[i].first = add(suffix_hash[i - 1].first, mul(ch,
power[i - 1].first, mod1), mod1);
suffix_hash[i].second = add(suffix_hash[i - 1].second, mul(ch,
power[i - 1].second, mod2), mod2);
}
}
pair<T, T> get_hash(int l, int r) {
T val1 = sub(prefix_hash[r].first, prefix_hash[l - 1].first,
mod1);
val1 = mul(val1, inv[l].first, mod1);
T val2 = sub(prefix_hash[r].second, prefix_hash[l - 1].second,
mod2);
val2 = mul(val2, inv[l].second, mod2);
return {val1, val2};
}
pair<T, T> get_hash_rev(int l, int r) {
T val1 = sub(suffix_hash[r].first, suffix_hash[l - 1].first, mod1);
val1 = mul(val1, inv[l].first, mod1);
T val2= sub(suffix_hash[r].second, suffix_hash[l - 1].second,
mod2);
val2 = mul(val2, inv[l].second, mod2);
return {val1, val2};
}
void change_current_hash(string &s)
{
this->s=s;
this->rev=s;
reverse(this->rev.begin(),this->rev.end());
n=s.size();
for(int i=1;i<=n;i++)
{
int ch=s[i-1]-'a'+1;
prefix_hash[i].first=add(prefix_hash[i-1].first,mul(ch,power[i-
1].first,mod1),mod1);
prefix_hash[i].second=add(prefix_hash[i-
1].second,mul(ch,power[i-1].second,mod2),mod2);
ch=rev[i-1]-'a'+1;
suffix_hash[i].first=add(suffix_hash[i-1].first,mul(ch,power[i-
1].first,mod1),mod1);
```

```
suffix_hash[i].second=add(suffix_hash[i-
1].second,mul(ch,power[i-1].second,mod2),mod2);
}
}
pair<T,T> get_new_hash(string &s)
{
int n=s.size();
T val1=0,val2=0;
for(int i=1;i<=n;i++)
{
int ch=s[i-1]-'a'+1;
val1=add(val1,mul(ch,power[i-1].first,mod1),mod1);
val2=add(val2,mul(ch,power[i-1].second,mod2),mod2);
}
val1=mul(val1,inv[1].first,mod1);
val2=mul(val2,inv[1].second,mod2);
return {val1,val2};
}
// combine hash of two strings of length l1 and l2
pair<T, T> combine_hash(pair<T, T> h1, pair<T, T> h2, int l1)
{
T val1 = add(h1.first, mul(h2.first, power[l1].first, mod1),
mod1);
T val2 = add(h1.second, mul(h2.second, power[l1].second,
mod2), mod2);
return {val1, val2};
}



pair<T,T> get_modified_hash_changed_at_ith_index(int i,char
ch)
{
T firstf=0;
T firsts=0;
if(i>1)
{
pair<ll,ll> p=get_hash(1,i-1);
firstf=p.ff;
firsts=p.ss;
}
T secondf=0;

T seconds=0;
if(i<n) {
pair<ll,ll> p=get_hash(i+1,n);
secondf=p.ff;
seconds=p.ss;
}
int chh=ch-'a'+1;
int pos=max(0,i-2);
firstf = add(firstf, mul(chh, power[pos].first, mod1), mod1);
firsts = add(firsts, mul(chh, power[pos].second, mod2),
mod2);
if(i==1)
{
firstf=mul(firstf, inv[i].first, mod1);
firsts=mul(firsts, inv[i].second, mod2);
}
pair<T,T> p=combine_hash({firstf,firsts},{secondf,seconds},i);
return p;
}
};
```

Z_Algo:

```
vector<int> z_function(string str){
Int lo = 0, hi = 0, n = str.size() ;
vector<int> z(n) ;
for(int i=1; i<n; i++)
{
if(i <= hi) z[i] = min(z[i - lo], hi - i + 1) ;
while(i+z[i] < n && str[ z[i] ] == str[ i + z[i] ])
z[i]++ ;
if(i+z[i]-1 > hi) lo = i , hi = i+z[i]-1 ;
}
return z ;
}
```

MST(Kruskal)

```
class dsu
{
vector<int> parent,size;
public:
dsu(int n)
{
parent.resize(n+1);
size.resize(n+1,1);
iota(parent.begin(),parent.end(),0);
}
int findpar(int node)
{
if(node==parent[node])
{
return node;
}
return parent[node]=findpar(parent[node]);
}
void unionbysize(int u,int v)
{
int pu=findpar(u);
int pv=findpar(v);
if(pu==pv) return;
int su=size[pu];
int sv=size[pv];
if(su>=sv)
{
parent[pv]=pu;
size[pu]+=size[pv];
}
else
{
parent[pu]=pv;
size[pv]+=size[pu];
}
}
void clear()
{
iota(parent.begin(),parent.end(),0);
size.resize(parent.size(),1);
}
};
```

```
int main()
{
int node,edge;
cin>>node>>edge;
dsu ds(node);
vector<array<int,3> > edges;
for(int i=0;i<edge;i++)
{
int u,v,w;
cin>>u>>v>>w;
edges.push_back({w,u,v});
}
sort(edges.begin(),edges.end());
int mstsum=0;
vector<array<int,2> > mstedges;
for(auto it : edges)
{
int w=it[0];
int u=it[1];
int v=it[2];
            if(ds.findpar(u)!=ds.findpar(v))
{
mstsum+=w;
mstedges.push_back({u,v});
ds.unionbysize(u,v);
}
}
cout<<mstsum<<endl;
for(auto it : mstedges)
{
cout<<it[0]<<" "<<it[1]<<endl;
}
}
```

nCr&nPr

```
const int MAXN = 2e6 + 5;

vector fact(MAXN), inv_fact(MAXN);

void precompute() { fact[0] = 1; for(int i = 1; i < MAXN; ++i)
{ fact[i] = fact[i-1] * i % MOD; } mint cur=fact[MAXN-1];

 cur=cur.inv();

 inv_fact[MAXN-1]=cur.v;

 for(int i = MAXN - 2; i >= 0; --i) { inv_fact[i] = inv_fact[i+1] *
(i+1) % MOD; } }

long long nCr(int n, int r) { if(r < 0 || r > n) return 0; return
fact[n] * inv_fact[r] % MOD * inv_fact[n - r] % MOD; }

long long nPr(int n, int r) { if (r < 0 || r > n) return 0; return
fact[n] * inv_fact[n - r] % MOD; }
```

MO(sqrt_decmopostion)

```
#define block 555
struct query{
int i;
int l;
int r;
};
query Q[200001];
page-5
int arr[200001],ans[200001];
int fre[200001];
int cnt=0;
bool cmp(query a,query b){
if(a.l/block != b.l/block)
return a.l/block < b.l/block;
return a.r < b.r;
}

void add(int pos){
fre[arr[pos]]++;
if(fre[arr[pos]]==1)
cnt++;
}
void remove(int pos){
fre[arr[pos]]--;
if(!fre[arr[pos]])
cnt--;
}
int32_t main(){
optimize();
int n,q; cin>>n>>q;
for(int i=0;i<n;i++) cin>>arr[i];
for(int i=0;i<q;i++){
cin>>Q[i].l>>Q[i].r;
Q[i].i=i,Q[i].l--,Q[i].r--;
}
sort(Q,Q+q,cmp);
int l=0,r=-1;
for(int i=0;i<q;i++){
int x=Q[i].l;
int y=Q[i].r;
while(l>x){
l--, add(l);
}
while(r<y){
r++,add(r);
}
while(l<x){
remove(l),l++;



while(r>y){
remove(r),r--;
}
ans[Q[i].i]=cnt;
}
for(int i=0;i<q;i++)
cout<<ans[i]<<endl;
}
```

Wavelet Tree:

```
const int MAX = 1e6;
```

```
vi g[N];
int a[N];
struct wavelet_tree{
#define vi vector<int>
#define pb push_back
int lo, hi;
wavelet_tree *l, *r;
vi b;
//nos are in range [x,y]
//array indices are [from, to)
wavelet_tree(int *from, int *to, int x, int y){

lo = x, hi = y;
if(lo == hi or from >= to) return;
int mid = (lo+hi)/2;
auto f = [mid](int x){
return x <= mid;
};
b.reserve(to-from+1);
b.pb(0);
for(auto it = from; it != to; it++)
b.pb(b.back() + f(*it));
//see how lambda function is used here
auto pivot = stable_partition(from, to, f);
l = new wavelet_tree(from, pivot, lo, mid);
r = new wavelet_tree(pivot, to, mid+1, hi);
}
//kth smallest element in [l, r]
int kth(int l, int r, int k){
if(l > r) return 0;
if(lo == hi) return lo;
int inLeft = b[r] - b[l-1];
int lb = b[l-1]; //amt of nos in first (l-1) nos that go in left
int rb = b[r]; //amt of nos in first (r) nos that go in left
if(k <= inLeft) return this->l->kth(lb+1, rb , k);
return this->r->kth(l-lb, r-rb, k-inLeft);
}
//count of nos in [l, r] Less than or equal to k
int LTE(int l, int r, int k) {
if(l > r or k < lo) return 0;
if(hi <= k) return r - l + 1;
int lb = b[l-1], rb = b[r];
return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
}
//count of nos in [l, r] equal to k
int count(int l, int r, int k) {
if(l > r or k < lo or k > hi) return 0;
if(lo == hi) return r - l + 1;
int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
if(k <= mid) return this->l->count(lb+1, rb, k);
return this->r->count(l-lb, r-rb, k);
}
~wavelet_tree(){
delete l;
delete r;
}
};
int main()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
srand(time(NULL));
int i,n,k,j,q,l,r;
}
cin >> n;
fo(i, n) cin >> a[i+1];
wavelet_tree T(a+1, a+n+1, 1, MAX);
cin >> q;
while(q--){
int x;
cin >> x;
cin >> l >> r >> k;
if(x == 0){
//kth smallest
cout << "Kth smallest: ";
cout << T.kth(l, r, k) << endl;
}
if(x == 1){
//less than or equal to K
cout << "LTE: ";
cout << T.LTE(l, r, k) << endl;
}
if(x == 2){
//count occurence of K in [l, r]
cout << "Occurence of K: ";
cout << T.count(l, r, k) << endl;
}
}
return 0;
}
```

Dijkstra

```
vector<int> path;
void dijkstra(int &source,int
&destination,vector<array<long long,2> > graph[])
{
priority_queue<array<long long,2>,vector<array<long
long,2> >,greater<array<long long,2> > > pq;
int n=destination+1;
vector<long long> dist(n,LONG_LONG_MAX);
vector<int> parent(n);
iota(parent.begin(),parent.end(),0);
pq.push({0,source});
dist[source]=0;

while(!pq.empty())
{
int node=pq.top()[1];
long long wt=pq.top()[0];



pq.pop();
if(wt>dist[node]) continue;
for(auto it : graph[node])
{
int newnode=it[0];
long long newwt=it[1];
if(dist[node]+newwt<dist[newnode])
{
```

```
dist[newnode]=dist[node]+newwt;
pq.push({dist[newnode],newnode});
parent[newnode]=node;
}
}
}
if(dist[destination]==LONG_LONG_MAX)
{
cout<<"NOT POSSIBLE"<<endl;
return;
}
cout<<"Cost of shortest path is : "<<dist[destination]<<endl;
cout<<"Shortest path is : ";
int node=destination;
while(node!=source)
{
path.push_back(node);
node=parent[node];
}
path.push_back(source);
reverse(path.begin(),path.end());

for(auto it : path)
{
cout<<it<<" ";
}
cout<<endl;
}
int main()
{
ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
int node,edge;
cin>>node>>edge;
vector<array<long long,2> > graph[node+1];
for(int i=0;i<edge;i++)
{
int u,v,w;
cin>>u>>v>>w;
graph[u].push_back({v,w});
graph[v].push_back({u,w});
}
int source;
cin>>source;
int destination;
cin>>destination;
dijkstra(source,destination,graph);
}
```

SPFA

```
void spfa(vector<array<int,2> > graph[],int node)
{
int inf=INT_MAX;
vector<int> dis(node+1,inf);
queue<int> q;
vector<int> count(node+1,0);
vector<bool> inqueue(node+1,false);
dis[1]=0;
q.push(1);
while(!q.empty())
{
int node=q.front();
q.pop();
inqueue[node]=false;
for(auto it : graph[node])
{
int newnode=it[0];
int wt=it[1];



if(dis[newnode]>dis[node]+wt)
{
dis[newnode]=dis[node]+wt;
if(!inqueue[newnode])
{
q.push(newnode);
inqueue[newnode]=true;
count[newnode]++;
if(count[newnode]>node)
{
cout<<"Negative Cycle Found"<<endl;
return;
}
}
}
}
}


cout<<"No Negative Cycle Found"<<endl;
}
```

Floyed Warshall

```
bool floyd_warshall(vector<vector<int> >& graph,int
nodes)
{
for(int k=1;k<=nodes;k++)
{
for(int i=1;i<=nodes;i++)
{
for(int j=1;j<=nodes;j++)
{
graph[i][j]=min(graph[i][j],graph[i][k]+graph[k][j]);
}
}
}
for(int i=1;i<=nodes;i++)
{
if(graph[i][i]<0) // if any node has negative weight cycle
then it will be less than 0
{
return true;
}
}
return false;
}
```

Topo_Sort

```
int node,edge;
cin>>node>>edge;
vector<int> graph[node+1];
```

```
for(int i=0;i<edge;i++)
{
int x,y;
cin>>x>>y;
graph[x].push_back(y);
}
vector<int> indegree(node+1,0);
for(int i=1;i<=node;i++)
{
for(auto it: graph[i])
{
indegree[it]++;
}
}
queue<int> q;
vector<int> order;
for(int i=1;i<=node;i++)
{
if(indegree[i]==0) q.push(i);
}
while(!q.empty())
{
int node=q.front();
q.pop();
order.push_back(node);
for(auto it: graph[node])
{
indegree[it]--;
if(indegree[it]==0) q.push(it);
}
}



if(order.size()!=node) cout<<"Cycle is present"<<endl;
else
{
for(auto it: order) cout<<it<<" ";
cout<<endl;
}
```

Seg_Seive

```
vector<ll> primes,primes1;
void seive(ll n)
{
vector<ll> isprime(n+1,true);
for(ll i=2;i<=n;i++)
{
if(isprime[i])
{
for(ll j=i*i;j<=n;j+=i) isprime[j]=false;
}
}
for(int i=2;i<=n;i++)
{
if(isprime[i]) primes.pb(i);
}
}

void segseive(ll low,ll high)
{
seive(sqrtl(high));
vector<bool> prime(high-low+100,true);
int sz=high-low+5;
for(auto it : primes)
{
ll sm=(low/it)*1LL*it;
if(sm<low) sm+=it;
for(ll i=sm;i<=high;i+=it)
{
prime[i-low]=false;
}
}
for(ll i=low;i<=high;i++)
{
if(prime[i-low]) primes1.pb(i);
}
}
```

Sparse_Table

```
template <typename T>
class SparseTable
{
public:
vector<vector<T> > st;
T op(T a,T b)
{
return __gcd(a,b);
}
SparseTable(int n,vector<T> &vec)
{
st.resize(n+2,vector<T> (__lg(n)+2));
for(int i=1;i<=n;i++)
{
st[i][0]=vec[i];
}
int k=__lg(n)+1;
for(int j=1;j<=k;j++)
{
for(int i=1;i+(1<<j)<=n+1;i++)
{
st[i][j]=op(st[i][j-1],st[i+(1<<(j-1))][j-1]);
}
}
}
T query(int l,int r)
{
int j=__lg(r-l+1);
return op(st[l][j],st[r-(1<<j)+1][j]);
}
T query1(int l,int r) // query in logn for non idempotent
function
{
int ans=0;
for(int j=__lg(r-l+1);j>=0;j--)
{
if((1<<j)<=(r-l+1))
{
ans=op(ans,st[l][j]);
l+=(1<<j);
}
}
}
```

```
return ans;
}
};
```

LIS

```
/// use upper_bound for non decreasing order
int lis(){
vector<int> dp ;
for(int i=0; i<lim; i++){
auto it=lower_bound(dp.begin(),dp.end(),arr[i]) ;
if(it == dp.end()) dp.push_back(arr[i]) ;
else dp[it-dp.begin()] = arr[i] ;
}
return dp.size() ;
}
void lis_print(){
vector<int> dp ;
for(int i=0; i<lim; i++){
f[i] = lower_bound(dp.begin(), dp.end(), arr[i]) -
dp.begin() ;
if(f[i] == dp.size()) dp.push_back( arr[i] ) ;
else dp[ f[i] ] = arr[i] ;
f[i]++ ; /// LIS from 0 - i for ith element include
}
int x = dp.size() ;
vector<int> lis ;
for(int i=lim-1; i>=0; i--){
if(f[i] == x){
lis.push_back( arr[i] ) ;
x-- ;
}
}
reverse(lis.begin(), lis.end()) ;
}
```

XOR_TRIE

```
class TrieNode
{
public:
  TrieNode *left;
  TrieNode *right;
  int cnt=0;
  TrieNode()
  {
    left=NULL;
    right=NULL;
    cnt=0;
  }
};

class Trie
{
  TrieNode *root;
public:

  Trie()
  {
    root=new TrieNode();
  }

  void insert(int n)
  {
    TrieNode *curr=root;
    for(int i=31;i>=0;i--)
    {
      int bit=(1&(n>>i));
      if(bit==0)
      {
        if(curr->left==NULL)
        {
          curr->left=new TrieNode();
        }
        curr=curr->left;
        curr->cnt++;
      }
      else
      {
        if(curr->right==NULL)
        {
          curr->right=new TrieNode();
        }
        curr=curr->right;
        curr->cnt++;
      }
    }
  }

  void remove(int n)
  {
    TrieNode* curr = root;

    for (int i = 31; i >= 0; i--)
    {
      if(curr==NULL)
      {
        break;
      }

      int bit = (n >> i) & 1;

      if (bit == 0)
      {
        curr = curr->left;
        curr->cnt--;
      }
      else
      {
        curr = curr->right;
        curr->cnt--;

      }
    }

  }

  int max_xor_pair(int n)
  {
    TrieNode *curr=root;
```

```
    int ans=0;
    for(int i=31;i>=0;i--)
    {
      if(curr==NULL)
      {
        break;
      }
      int bit=(1&(n>>i));

      if(bit==0)
      {
        if(curr->right!=NULL and curr->right->cnt>0)
        {
          ans+=(1<<i);
          curr=curr->right;
        }
        else
        {
          curr=curr->left;
        }
      }
      else
      {
        if(curr->left!=NULL and curr->left->cnt>0)
        {
          ans+=(1<<i);
          curr=curr->left;
        }
        else
        {
          curr=curr->right;
        }
      }
    }
    return ans;
  }

  int find_x_such_that_x_xor_y_less_than_or_equal_k(int y,int
k)
{
  TrieNode *curr=root;
  int ans=0;
  for(int i=31;i>=0;i--)
  {
    if(curr==NULL)
    {
      break;
    }
    int bity=(1&(y>>i));
    int bitk=(1&(k>>i));
    if(bity)
    {
      if(bitk)
      {
        if(curr->right!=NULL)
        {
          ans+=curr->right->cnt;
        }
        curr=curr->left;
      }
      else
      {
        curr=curr->right;
      }
    }
    else
    {
      if(bitk)
      {
        if(curr->left!=NULL)
        {
          ans+=curr->left->cnt;
        }
        curr=curr->right;
      }
      else curr=curr->left;
    }
  }
  if(curr!=NULL) ans+=curr->cnt;
    return ans;
}

int subarray_less_than_k(int n,int k)
{
  TrieNode *curr=root;
  int ans=0;
  for(int i=31;i>=0;i--)
  {
    if(curr==NULL)
    {
      break;
    }
    int bitk=(1&(k>>i));
    int bitn=(1&(n>>i));

    if(bitn==bitk)
    {
      if(bitk)
      {
        if(curr->right!=NULL)
        {
          ans+=curr->right->cnt;
        }
      }
      curr=curr->left;
    }
    else
    {
      if(bitk)
      {
        if(curr->left!=NULL)
        {
          ans+=curr->left->cnt;
        }
      }
      curr=curr->right;
    }
```

```
    }

    return ans;
  }

};
```

Formula

$1^2 + 2^2 + 3^2 + ... + n^2 = (1/6)\{n(n + 1)(2n + 1\}$ **for** all $n \in N$.

(1 x 2) + (3 x 4) + (5 x 6) + ... + (2n - 1) x 2n = n(n+1)(4n−1)/3

$1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + ... + n(n + 1) = (1/3)\{n(n + 1)(n + 2)\}$.

$1 \cdot 3 + 3 \cdot 5 + 5 \cdot 7 + ...+ (2n - 1)(2n + 1) = (1/3)\{n(4n^2 + 6n - 1)$.

$1/(1 \cdot 2) + 1/(2 \cdot 3) + 1/(3 \cdot 4) + ... + 1/\{n(n + 1)\} = n/(n + 1)$

1. 3 ways to **do** modulo inverse. 1/a % m which is equivalent to ax = 1 mod m
   a. m is prime - a^(m-2) = 1/a mod m
   b. m is not prime - ax+my = 1 find x by egcd
   c. m is not prime - a^phi(m)-1 = 1/a mod m
2. NOD(n) = (a1+1) * (a2+1) * (a3+1) * ... * (ak+1)
3. SOD(n) = (pk^(ak+1) - 1) / (pk - 1)
4. **Egcd** solve the equation ax + by = gcd(a, b)
   a. X += k * b/gcd(a,b) ; Y -= k * a/gcd(a,b)
5. **Diophantine** equation : ax + by = c solution exist iff c | gcd(a, b)
6. gcd(a, b) = gcd(-a, y) = gcd(x, -y) = gcd(-x, -y)
7. gcd(a, b) = gcd(a, b+a) = gcd(a, b-a), here b > a
8. gcd(a[0], a[1], a[2],..,a[n]) = gcd(a[0], a[1]-a[0], a[2]-a[1],..,a[n]-a[n-1])
   a. k is added to all value then just add it to a[0]
9. phi(n) = n * (1-1/p1) * (1-1/p2) .. (1-1/pk)
10. Number of x <= n and gcd(x, n) = 1 is equal to phi(n/d)
11. Sum of above x is equal to phi(n) * n / 2
12. Sum of phi(d) = n **for** all d.
13. Arithmetic progression : a + (d+a) + (2d+a) ..
    a. **Nth** term : a + (n-1)d
    b. **Sum** of n term : n/2[2a + (n-1)*d]
    c. **Sum** of AP : n/2[1st term + nth term]
14. Geometric progression: a + ar + ar^2 + ar^(n-1)
    a. **Nth** term : ar(n-1)
    b. **Sum** of n term : a[(r^n - 1)/(r-1)]
    c. **Sum** of n term : (nth*r - 1st)/(r-1)
15. Harmonic sum 1+1/2+1/3+..+1/n = logn +1
25. - 2's biggest power which is divisor of x : 1 << (__builtin_ctz(x)).
26. - 2's smallest power which is not smaller then x : 1 << (32 - __builtin_clz(x - 1)). **But** it has **Undefined Behavior** **for** x ≤ 1 so we need an extra check **for** **this**.
27. a+b = (a^b) + 2*(a&b)
28. a|b = (a^b) + (a&b)

Geometry and **Trigonometry**

33. sin(th)^2 + cos(th)^2 = 1
34. sec(th)^2 - tan(th)^2 = 1
35. cosec(th)^2 - cot(th)^2 = 1
36. a/sinA = b/sinB = c/sinC = 2R
37. a^2 = b^2 + c^2 - 2bccosA
38. A = bcosC + ccosB
39. T.area = 1/2 bcsinA = sqrt(s*(s-a)*(s-b)*(s-c))
40. In circle radius,r = (2*T.area)/(a+b+c)
41. Circumcicle radius,R = abc/4T.area
42. Rr = abc/4s
43. T.area = 1/2bh
44. T.area = 1/2(x1*(y1-y3)+x2*(y3-y1)+x3*(y1-y2))
45. C.area = pi * r^2
46. Circumference = 2*pi*r
47. Length of arc = pi*r*th/180
48. Sector area = (th/360) * pi * r^2
49. y-y1 = m (x-x1)
50. y = mx + c
51. Line - **Ax** + **By** = C or **Ax**+**By**+C = 0
52. Parametric - p(t) = p0 + t(p1-p0)

Template

```
#include<bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
# define el '\n'
# define sp " "
# define ff first
# define ss second
# define ll long long
# define pb push_back
# define ull unsigned long long
# define all(v) v.begin(),v.end()
# define allr(v) v.rbegin(),v.rend()
constexpr ll mod=1000000000+7;
constexpr ll INF=LLONG_MAX;
constexpr double PI= acos(-1);
constexpr double eps=1e-9;
# define optimise { ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL); }
# define fraction(a) cout.unsetf(ios::floatfield);
cout.precision(a); cout.setf(ios::fixed,ios::floatfield);
# define ordered_set tree<ll, null_type,less_equal<ll>,
rb_tree_tag,tree_order_statistics_node_update>
// find_by_order() - Returns an iterator to the k-th largest
element (counting from zero)
```

```
// order_of_key() - The number of items in a set that are strictly smaller than our item
// greater instead of less for descending order
// less_equal works as ordered multiset
// we can use pair<int,int> instead of int for pair of orderd set
// for multiset st.lower_bound(x) works as upper bound and st.upper_bound(x) works as lower bound
inline void file() {
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
        freopen("output.txt", "w", stdout);
#endif // ONLINE_JUDGE
}
///// FUNCTIONS /////
ll bigmod(ll base,ll power){ ll res=1; ll p=base%mod;
while(power>0) { if(power%2==1)
{ res=((res%mod)*(p%mod))%mod; } power/=2;
p=((p%mod)*(p%mod))%mod; } return res; }
ll inversemod(ll base) { return bigmod(base,mod-2); }
ll sqrtt(ll a){ long long x = sqrt(a) + 2; while (x * x > a) x--;
return x;}
long double sqrtd(long double n){ long double
low=0,high=n,mid; for(int i=0;i<100;i++)
{ mid=(low+high)/2; if(mid*mid<=n) low=mid; else
high=mid;} return low;}
mt19937
rng(chrono::high_resolution_clock::now().time_since_epoch().
count());
inline ll getrandom(ll a,ll b) { return
uniform_int_distribution<ll>(a,b)(rng); }
int dx[]={-1, 1 , 0 , 0 , -1 ,-1, 1, 1};
int dy[]={ 0, 0 ,-1 , 1 , -1 , 1,-1, 1};
// up = { -1,0 } , down = { 1,0 } , right = { 0,1 } , left = { 0,-1 }
// up-right = { -1,1 } , up-left = { -1,-1 } , down-right = { 1,1 } , down-left = { 1,-1 }
void solve()
{
}
int main()
{
optimise;
file();
clock_t start= clock();
int t;
cin>>t;
for(int i=1;i<=t;i++)
{
solve();
}
cerr << "Run Time : " <<((double)(clock() - start) /
CLOCKS_PER_SEC)<<el;
}
//---------- seive ----------
// to show prime numbers
const ll mx=1e8;
bitset<mx> prime;
vector<ll> prime_show;
void seive(ll n){prime[1]=1;for(int i=3;i*i<=n;i+=2){
if(!prime[i]){for(int j=i*i;j<=n;j+=(i*2)){prime[j]=1;}}}
prime_show.PB(2);
for(int i=3;i<=n;i+=2){if(!prime[i]) prime_show.PB(i);}
}
//---------- seive end ----------
vector<ll> prime_factorization(ll n){ // n number take kon kon prime numer dara vag jai
vector<ll> fector;
for(auto u : prime_show){if(1LL*u*u > n)
break;if(!(n%u)){while(!(n%u)){fector.PB(u);n/=u;}}}if(n>1
) fector.PB(n);
return fector;
}
ll NOD(ll n){ // number of divisiors of a number ?Time : O(n * (sqrt n)/ln n)
ll nod=1;
for(auto u:prime_show){if(1LL*u*u>n) break;if(!(n%u)){ll
cnt=0;while(!(n%u)){n/=u;cnt++;}cnt++;nod*=cnt;}}
if(n>1){nod*=2;}return nod;
}
ll DSF(ll n){ // sum of number of divisors of a given number (1 to n) // DFS: divisor summetory function.. ?time: O(sqrt n)
ll x=sqrt(n),sum=0;lup_1(1,x) sum+=(n/i)-i;sum*=2; sum+=x;
return sum;
}
ll SOD(ll n) {// Sum of devisors of an number
ll sod=1;
for(auto u:prime_show){if(1LL*u*u>n) break;
if(!(n%u)){ll
sum=1,a=1;while(!(n%u)){a*=u;sum+=a;n/=u;}sod*=sum;}}
if(n>1){sod*=(n+1);}
return sod;
}
ll eulerphi(ll n){ // n er asthe 1 theke n obdi emon koto gulo songkha ase jar sathe n er GCD 1 gcd(n,x)=1 or calculate the numbe of co prime
ll phi = n; for(auto u:prime_show){if(1LL*u*u>n) break; // time complexity O(sqrt(n)/ln(sqrt n))
if(!(n%u)){while(!(n%u)){n/=u;}phi/=u;phi*=(u-
1);}}if(n>1){phi/=n;phi*=(n-1);} return phi;
}
const ll p=5e6+123;
unsigned long long phi[p],ans[p];
void eulerphi_using_harmonic(ll x){
for(int i=1;i<=x;i++) phi[i]=i;
for(auto u:prime_show){
for(int i=u;i<=x;i+=u){phi[i]/=u;phi[i]*=(u-1);}}
}
// summetion of coprimes of a singel given number is (phi[n]*n)/2
void lcm_sum(ll x){
for(int i=1;i<=x;i++){
for(int j=i;j<=x;j+=i){
ans[j]+=(phi[i]*i);
}
}
}
```

```
// this code finds the sum of number of divisors from 1 to n in sqrt(n) time
int main()
{
ll n;
cin>>n;
int sq=sqrt(n);
ll ans=0;
for(int i=1;i<=sq;i++)
{
ans+=(n/i)-i;
}
ans*=2;
ans+=sq;
cout<<ans<<endl;
}
```

Centroid Decom

```
int n;
cin>>n;
int q;
cin>>q;
vector<int> graph[n+1];
for(int i=1;i<n;i++)
{
int u,v;
cin>>u>>v;
graph[u].pb(v);
graph[v].pb(u);
}


const int LOG = 20;
vector<array<int,LOG>> up(n+1);
vector<int> depth(n+1);



function<void(int,int)> dfsLCA = [&](int v, int p) {
up[v][0] = p;
for(int k = 1; k < LOG; k++)
up[v][k] = up[ up[v][k-1] ][k-1];


for(int to : graph[v]) {
if(to == p) continue;
depth[to] = depth[v] + 1;
dfsLCA(to, v);
}
};


dfsLCA(1, 1);


auto lca = [&](int a, int b) {
if(depth[a] < depth[b]) swap(a,b);
int diff = depth[a] - depth[b];
for(int k = 0; k < LOG; k++)
if(diff & (1<<k))
a = up[a][k];

if(a == b) return a;
for(int k = LOG-1; k >= 0; k--) {
if(up[a][k] != up[b][k]) {
a = up[a][k];
b = up[b][k];
}
}
return up[a][0];
};



auto getdis = [&](int u, int v) {
int w = lca(u, v);
return depth[u] + depth[v] - 2*depth[w];
};


vector<int> used(n+1),size(n+1),parent(n+1);
vector<int> ans(n+1,2e5);
function<int(int,int)> get_size=[&](int node,int par)
{
size[node]=1;
for(auto it : graph[node])
{
if(it==par or used[it]) continue;
size[node]+=get_size(it,node);
}

return size[node];
};

function<int(int,int,int)> get_cen=[&](int node,int par,int sz)
{
for(auto it : graph[node])
{
if(it==par or used[it]) continue;
if(size[it]>sz/2) return get_cen(it,node,sz);
}

return node;
};

function<void(int,int)> decompose=[&](int node,int par)
{
int sz=get_size(node,0);
int cen=get_cen(node,0,sz);
used[cen]=1;
if(par==0) par=cen;
parent[cen]=par;
for(auto it : graph[cen])
{
if(used[it]) continue;
decompose(it,cen);
}
};

function<void(int)> update=[&](int cur)
{
int x=cur;
```

```
ans[cur]=0;
while(1)
{
ans[x]=min(ans[x],getdis(x,cur));
if(parent[x]==x) break;
x=parent[x];
}
};

function<int(int)> qry=[&](int cur)
{
int x=cur;
int go=ans[x];
while(1)
{
go=min(go,getdis(x,cur)+ans[x]);
if(parent[x]==x) break;
x=parent[x];
}

return go;
};

decompose(1,0);
update(1);

while(q--)
{
int type;
cin>>type;
if(type==1)
{
int u;
cin>>u;
update(u);
}
else{
int u;
cin>>u;
cout<<qry(u)<<el;
}
}
```

Knapsack_binary_optimizaiton

```
function<ll(vector<int>&,int)> get=[&](vector<int>&v,int size)
{
Cool_Bitset dp(size+1);
dp.set(0, 1);

map<int,int> mp;
for (int x : v) mp[x]++;
for(auto [w,cnt] : mp)
{
int cur=1;
while(cnt>0)
{
dp.left_shift(w*min(cnt,cur));
cnt-=cur;
cur*=2;
}
}

};

function<ll(vector<int>&,int)> get=[&](vector<int>&v,int size)
{
map<int,int> mp;
for(auto it : vec) mp[it]++;
vector<int> dp(n+1,1e9);
dp[0]=0;
for(auto [w,cnt] : mp)
{
int cur=1;
while(cnt>0)
{
int use=min(cnt,cur);
for(int i=n;i>=w*use;i--)
{
dp[i]=min(dp[i],dp[i-w*use]+use);
}
cnt-=use;
cur*=2;
}
}
};
```

Mint

```
const int MOD = 1e9 + 7;
template<ll M>
struct modint {
static ll _pow(ll n, ll k) {
ll r = 1;
for (; k > 0; k >>= 1, n = (n*n)%M)
if (k&1) r = (r*n)%M;
return r;
}
ll v; modint(ll n = 0) : v(n%M) { v += (M&(0-(v<0))); }



friend string to_string(const modint n) { return
to_string(n.v); }
friend istream& operator>>(istream& i, modint& n) { return
i >> n.v; }
friend ostream& operator<<(ostream& o, const modint n)
{ return o << n.v; }
template<typename T> explicit operator T() { return
T(v); }
friend bool operator==(const modint n, const modint m)
{ return n.v == m.v; }
friend bool operator!=(const modint n, const modint m)
{ return n.v != m.v; }
friend bool operator<(const modint n, const modint m)
{ return n.v < m.v; }
```

```
friend bool operator<=(const modint n, const modint m)
{ return n.v <= m.v; }
friend bool operator>(const modint n, const modint m)
{ return n.v > m.v; }
friend bool operator>=(const modint n, const modint m)
{ return n.v >= m.v; }
modint& operator+=(const modint n) { v += n.v; v -= (M&(0-(v>=M))); return *this; }
modint& operator-=(const modint n) { v -= n.v; v += (M&(0-(v<0))); return *this; }
modint& operator*=(const modint n) { v = (v*n.v)%M;
return *this; }
modint& operator/=(const modint n) { v = (v*_pow(n.v, M-2))%M; return *this; }
friend modint operator+(const modint n, const modint m)
{ return modint(n) += m; }
friend modint operator-(const modint n, const modint m)
{ return modint(n) -= m; }
friend modint operator*(const modint n, const modint m)
{ return modint(n) *= m; }
friend modint operator/(const modint n, const modint m)
{ return modint(n) /= m; }
modint& operator++() { return *this += 1; }
modint& operator--() { return *this -= 1; }
modint operator++(int) { modint t = *this; return *this += 1, t; }
modint operator--(int) { modint t = *this; return *this -= 1, t; }
modint operator+() { return *this; }
modint operator-() { return modint(0) -= *this; }
modint pow(const ll k) const {
return k < 0 ? _pow(v, M-1-(-k%(M-1))) : _pow(v, k);
}
modint inv() const { return _pow(v, M-2); }
};
using mint = modint<int(MOD)>;
```

Segment tree

```
template <typename T>
class SegmentTree
{
public:
  vector<T> st;

  ll op(ll a,ll b)
  {
    return a+b;
  }

  SegmentTree(vector<T> &vec, int n)
  {
    st.resize(4 * n, 0);

    function<void(int, int, int)> build = [&](int id, int start, int end)
    {
      if (start == end)
      {
        st[id] = vec[start];
        return;
      }
      int mid = (start + end) / 2;

      build(2 * id, start, mid);
      build(2 * id + 1, mid + 1, end);

      st[id] = op(st[2*id],st[2*id+1]);
    };

    build(1, 1, n);
  }

  ll query(int id, int start, int end, int l, int r)
  {
    if (start > r or end < l)
      return 0;
    if (start >= l and end <= r)
      return st[id];

    int mid = start + (end - start) / 2;

    ll left = query(2 * id, start, mid, l, r);
    ll right = query(2 * id + 1, mid + 1, end, l, r);

    return op(left,right);
  }

  void update(int id, int start, int end, int ind, int cur)
  {
    if (start > ind or end < ind)
      return;
    if (start == end and start == ind)
    {
      st[id] = cur;
      return;
    }

    int mid = start + (end - start) / 2;

    update(2 * id, start, mid, ind, cur);
    update(2 * id + 1, mid + 1, end, ind, cur);

    st[id] = op(st[2 * id], st[2 * id + 1]);
  }
};
```

Bitset

```
struct Cool_Bitset {
vector<uint64_t> bits;
int64_t b, n;
Cool_Bitset(int64_t _b = 0) {
```

```
init(_b);
}
void init(int64_t _b) {
b = _b;
n = (b + 63) / 64;
bits.assign(n, 0);
}
void clear() {
b = n = 0;
bits.clear();
}
void reset() {
bits.assign(n, 0);
}
void _clean() {
if (b != 64 * n)
bits.back() &= (1LLU << (b - 64 * (n - 1))) - 1;
}
bool get(int64_t index) const {
return bits[index / 64] >> (index % 64) & 1;
}
void set(int64_t index, bool value) {
assert(0 <= index && index < b);
bits[index / 64] &= ~(1LLU << (index % 64));
bits[index / 64] |= uint64_t(value) << (index % 64);
}
void left_shift(int64_t shift) {
int64_t div = shift / 64, mod = shift % 64;
if (mod == 0) {
for (int64_t i = n - 1; i >= div; i--)
bits[i] |= bits[i - div];
return;
}



for (int64_t i = n - 1; i >= div + 1; i--)
bits[i] |= bits[i - (div + 1)] >> (64 - mod) | bits[i - div] << mod;
if (div < n)
bits[div] |= bits[0] << mod
_clean();
}
void right_shift(int64_t shift) {
int64_t div = shift / 64, mod = shift % 64;
if (mod == 0) {
for (int64_t i = div; i < n; i++)
bits[i - div] |= bits[i];
return;
}
for (int64_t i = 0; i < n - (div + 1); i++)
bits[i] |= bits[i + (div + 1)] << (64 - mod) | bits[i + div] >> mod;
if (div < n)
bits[n - div - 1] |= bits[n - 1] >> mod;
_clean();
}


int64_t count() const {
int64_t res = 0;
for (int i = 0; i < n; i++)
res += __builtin_popcountll(bits[i]);
return res;
}
int64_t find_first() const {
for (int i = 0; i < n; i++)
if (bits[i] != 0)
return 64 * i + __builtin_ctzll(bits[i]);
return -1;
}
int64_t find_next(int x) const {
for (int i = x + 1; i < x + 64; i++)
{
if (get(i))
return i;
}
for (int i = x / 64 + 1; i < n; i++)
if (bits[i] != 0)
return 64 * i + __builtin_ctzll(bits[i]);
return -1;
}
Cool_Bitset& operator&=(const Cool_Bitset &other) {
assert(b == other.b);
for (int i = 0; i < n; i++)
bits[i] &= other.bits[i];
return *this;
}

Cool_Bitset operator&(const Cool_Bitset &other) const {
Cool_Bitset res(b);
for (int i = 0; i < n; i++) res.bits[i] = bits[i] & other.bits[i];
return res;
}
};
```

Lazy

```
vector<ll> seg_tree, lazy;

void build(ll pos, ll x, ll nd, ll st, ll ed)
{
if(pos < st || pos > ed) return;
if(ed == st) {
seg_tree[nd] = x;
return;
}
ll mid = (ed + st) / 2, left_nd = 2 * nd, right_nd = 2 * nd + 1;
build(pos, x, left_nd, st, mid);
build(pos, x, right_nd, mid + 1, ed);
seg_tree[nd] = seg_tree[left_nd] + seg_tree[right_nd];
lazy[nd] = 0;
}
void update(ll left, ll right, ll idx, ll l, ll r, ll val) {
if(lazy[idx] != 0){
seg_tree[idx] += (right - left + 1) * lazy[idx];
if(left != right) {
lazy[idx * 2] += lazy[idx];
```

```
lazy[idx * 2 + 1] += lazy[idx];
}
lazy[idx] = 0;
}

if(l > right || r < left) return;

if(l <= left && r >= right) {
seg_tree[idx] += (right - left + 1) * val;
if(left != right) {
lazy[2 * idx] += val;
lazy[2 * idx + 1] += val;
}
return;
}

ll mid = (left + right) / 2, left_nd = 2 * idx, right_nd = 2 * idx + 1;
update(left, mid, left_nd, l, r, val);
update(mid + 1, right, right_nd, l, r, val);
}

ll value(ll left, ll right, ll idx, ll pos) {
if(lazy[idx] != 0){
seg_tree[idx] += (right - left + 1) * lazy[idx];
if(left != right) {
lazy[idx * 2] += lazy[idx];
lazy[idx * 2 + 1] += lazy[idx];
}
lazy[idx] = 0;
}

if(pos < left || pos > right) return -1;



if(left == right) return seg_tree[idx];
```

```
ll mid = (left + right) / 2, left_nd = 2 * idx, right_nd = 2 * idx + 1;
ll x1 = value(left, mid, left_nd, pos);
ll x2 = value(mid + 1, right, right_nd, pos);
return max(x1, x2);
}

int main()
{
ll n, m;
cin >> n >> m;

seg_tree = vector<ll> (4 * n);
lazy = vector<ll> (4 * n);

for(int i = 0; i < n; i++){
ll x;
cin >> x;
build(i, x, 1, 0, n - 1);
}

while(m--) {
ll type, a, b, u;
cin >> type;

if(type == 1) {
cin >> a >> b >> u;
update(0, n - 1, 1, a - 1, b - 1, u);
}
else {
cin >> u;
cout << value(0, n - 1, 1, u - 1) << endl;
}
}
}
```

Geometry

/// Geometry Formula ?
Area of a Triangle = ½ × base × height
= √s(s−a)(s−b)(s−c) // s = half perimeter
= 1/2 × bc × sin(A) // A = angle between b and c
= 1/2 × ac × sin(B)
= (√3)/4 × side2 // area of equilateral triangle
= 1/4 × b√4a2−b2 // area of isosceles triangle
= (b*h)/2 (side: a,a,b) // area of isosceles triangle
Area of Rectangle = a*b (side : a,b)
d = √(a2 + b2) (diagonal length of Rectangle)
Area of Trapizium = ½(a + b)h
Area of a Cylinder = 2πrh
Area of a Cylinder = 2πr(r + h) (Total)
Area of a Sphere = 4πr2
Area of a cone = πrl
Total surface area of pyramid = b^2 + 2bs , here b = base and s = slant height
Area of a Circle = π×r2
= 2πr (Circumference
For (x1,x2)&(y1,y2)= √(x2-x1)^2 + (y2-y2)^2 (distance)
= ((x1+x2)/2, (y1+y2)/2) (Midpoint)
= ((y2-y1)/x2-x1) (slop)

//////Computational Geometry Template

```
///-------------------------------------------------------///
#include<bits/stdc++.h>        // --> Bismillahir_ Rahmanir_ Rahim <--
#define ll long long
using namespace std;


// use long double for larger float number
//#define double long double
//#define lll __int128_t
#define pi acos(-1)


/// precession check
```

```
double eps = 1e-9;
ll sign(double x) {//Let 2 float number a and b. put the (a-b) in the sign function.
    if(fabs(x)<eps) {
        return 0; // both are equal.
    }
    else if(x>0) {
        return 1; // first one is greater.
    }
    else {
        return -1;// first one is smaller.
    }
}
// Always think, point as a vector.
struct point{
    ll x,y;
    point(ll x=0, ll y=0) { // point p(x,y). here, p is the point name
        this -> x = x;
        this -> y = y;
    }
    // here, (x,y) is left point and p is right point.
    point operator+(point p) {
        return point(x+p.x, y+p.y);
    }
    point operator-(point p) {
        return point(x-p.x, y-p.y);
    }
    // here, (x,y) is the point and z is an integer. ex. 3*(2i + 3j) = (6i + 9j)
    point operator*(ll z) {
        return point(x*z, y*z);
    }
    point operator/(ll z) { // ex. (2i + 6j)/2 = (i + 3j) // check ->Fraction
        return point(x/z, y/z);
    }
    bool operator==(point p) { // for integers point
        return (x==p.x && y==p.y);
    }
    bool operator!=(point p) { // for integers point
        return (x!=p.x || y!=p.y);
    }
    // bool operator==(point p) { // for floating point
    //     return (sign(x-p.x)==0 && sign(y-p.y)==0);
    // }
    bool operator<(point p) { // for integers point
        if(x==p.x) {
            return y<p.y;
        }
        else {
            return x<p.x;
        }
    }
    bool operator>(point p) { // for integers point
        if(x==p.x) {
            return y>p.y;
        }
        else {
            return x>p.x;
        }
    }
    // bool operator<(point p) { // for floating point
    //     if(sign(x==p.x)==0) {
    //         return sign(y-p.y)==-1;
    //     }
    //     else {
    //         return sign(x-p.x)==-1;
    //     }
    // }


    double len() {
        return sqrt(x*x + y*y);
    }
};


double degree(double rediant) {
    return (rediant*57.2958); // as, 1 radiant = 57.2958 degree
}
ll des(point a, point b) {
    return (((a.x-b.x)*(a.x-b.x)) + ((a.y-b.y)*(a.y-b.y)));
}
ll dot(point p, point q) {
    return ((p.x*q.x) + (p.y*q.y));
}
ll cross(point p,point q) {
    return ((p.x*q.y) - (q.x*p.y));
}


ll orient(point A, point B, point C) {
    point AB = B-A;
    point AC = C-A;
    return cross(AB,AC);
    // if,cross product = 0, then A,B,C are in same line/on line/Touch
    //                  > 0, then it will move from A -> B to C Anti-Clockwise (Left).
    //                  < 0, then it will move from A -> B to C Clockwise(Right).
}
bool on_line(point A, point B, point P) {//check wether the point p is on the AB line or not
    if(orient(A,B,P)==0) {
        return true;
    }
    else {
        return false;
    }
}
bool on_segment(point A, point B, point P) {// check wether the point p is on the AB segment or not
    if(on_line(A,B,P)) {
        point AP = P-A;
        point BP = P-B;
        if(dot(AP,BP)<=0) {
            return true;
        }
        else {
            return false;
        }
    }
    else {
```

```
        return false;
    }
}


bool intersect(point A, point B, point C, point D) { // check wether the segment AB and CD intersect or not
    if(on_segment(A,B,C) || on_segment(A,B,D) || on_segment(C,D,A) || on_segment(C,D,B)) {
        return true;
    }
    else {
        ll a = orient(A,B,C);
        ll b = orient(A,B,D);
        ll c = orient(C,D,A);
        ll d = orient(C,D,B);
        if( ((a<0 && b>0)||(a>0 && b<0)) && ((c<0 && d>0)||(c>0 && d<0))) {
            return true;
        }
        else {
            return false;
        }
    }
}
pair<double,double> intersection_point(point a, point b, point c, point d) {// find the intersection point of segment AB and CD
    double a1 = a.y-b.y, b1 = b.x-a.x, c1 = cross(a,b);
    double a2 = c.y-d.y, b2 = d.x-c.x, c2 = cross(c,d);
    double det = a1*b2 - a2*b1;


    if(det==0) {
        if(fabs(a1*b2 - a2*b1)<eps && fabs(a1*c2 - a2*c1)<eps && fabs(b1*c2 - b2*c1)<eps) {
            return {1e18, 1e18};// if the given segment are in same line or there is lot of intersection point
        }
        else {
            return {-1e18, -1e18};// if the given segment are parallel but not in same line
        }
    }
    else {
        return {(b1*c2 - b2*c1)/det, (c1*a2 - a1*c2)/det};//only one intersection point
    }
}
ll triangle_area_2x(point A, point B, point C) {// for learning purpose
    // Here, we initiate the point A in origin.
    point AB = B-A;
    point AC = C-A;
    ll z = cross(AB,AC);
    return abs(z); // bcz area always positive
}
ll polygon_area_2x(vector<point>&vp) {
    ll z = 0, n = vp.size();
    for(ll i=0 ; i<n ; i++) {
        ll j = (i+1)%n;
        z += cross(vp[i],vp[j]);
    }
    return abs(z);
}
ll convex_point_check(vector<point>&vp, point P) { // For Convex, TC = [ O(n) ]  (Learning Purpose)
    ll n = vp.size();
    ll p_area_2x = polygon_area_2x(vp);
    ll sum_of_t_area_2x = 0, ok = 0;
    for(ll i=0 ; i<n ; i++) {
        ll j = (i+1)%n;
        ll z = triangle_area_2x(P,vp[i],vp[j]);
        sum_of_t_area_2x += z;
        if(z==0) ok = 1;
    }
    if(p_area_2x==sum_of_t_area_2x) {
        if(ok) {
            return 0; // point P is on the Boundary of the polygon vp
        }
        else {
            return 1; // inside
        }
    }
    else {
        return -1; // outside
    }
}
ll polygon_point_check_x(vector<point>&vp, point p) { // For Covex,  TC = [ O(logn) ]
    ll n =vp.size();
    ll a = orient(vp[0],vp[1],p), b = orient(vp[0],vp[n-1],p);
    if (a > 0 || b < 0) return -1;
    ll L = 1, R = n - 1;
    while (L + 1 < R) {
        ll mid = (L + R)/2;
        if(orient(vp[0],vp[mid],p) < 0) {
            L = mid;
        }
        else {
            R = mid;
        }
    }
    ll z =orient(vp[L],vp[R],p);
    if(z>0) return -1;
    if(z==0 || (L == 1 && a == 0) || (R == n - 1 && b == 0)) return 0;
    return 1;
    // -1 : outside
    //  0 : on boundary
    //  1 : inside
}
ll polygon_point_check(vector<point>&vp,point P) { // for both Convex & Concave, TC = [ O(n) ]
    ll n = vp.size();
    point end_point(INT_MAX,INT_MAX+1);
    ll cnt = 0, ok = 0;
    for(ll i=0 ; i<n ; i++) {
        ll j = (i+1)%n;
        if(on_segment(vp[i],vp[j],P)) ok = 1;
        if(intersect(vp[i],vp[j],P,end_point)) ++cnt;
    }
    if(ok) {
        return 0; // point P is on the boundary of the polygon vp.
```

```
    }
    else if(cnt&1) {
        return 1; // inside
    }
    else {
        return -1; // outside
    }
}
/// ***Picks Formula: A = i + (b/2) - 1
/// Here, A = polygon Area, i = number of lattice point inside the polygon.
///       b = number of lattice point are on the boundary of the polygon.
/// a Point is called lattice point if it's x and y coordinate are integer.
/// Let, two point are A and B. So, the slope of the two point is nothing but
/// how many move we do in x and y axis to reach from point A to point B.
ll cnt_of_lattice_point_on_segment(point A, point B) {// lattice point mean integer coordiante
    ll xs = abs(B.x - A.x);
    ll ys = abs(B.y - A.y);
    return __gcd(xs,ys)+1;
}
ll cnt_of_lattice_point_on_boundary_of_polygon (vector<point>&vp) {
    ll ans = 0, n = vp.size();
    for(ll i=0 ; i<n ; i++) {
        ll j = (i+1)%n;
        ans += cnt_of_lattice_point_on_segment(vp[i],vp[j]);
    }
    ans -= n; // removing overcount
    return ans;
}
ll cnt_of_lattice_point_inside_polygon(vector<point>&vp) {
    // using picks Formula: A = i + (b/2) - 1
    ll A = polygon_area_2x(vp)/2;
    ll b = cnt_of_lattice_point_on_boundary_of_polygon(vp);
    ll i = A+1-(b/2);
    return i;
}
pair<point,point> common_part_of_2_rectangle(point a, point b, point c,point d) {
    point LL = {max(a.x,c.x),max(a.y,c.y)};//LL = Lower-Left point
    point UR = {min(b.x,d.x),min(b.y,d.y)};//UR = Upper-Right point
    if(LL.x<=UR.x && LL.y<=UR.y) {
        return {LL,UR};
    }
    else {
        return {{2,2},{1,1}}; // return (2,2) and (1,1) if there is no common portion
    }
}
vector<point> convex_Hull_point(vector<point>&vp) {
    ll n = vp.size();
    sort(vp.begin(),vp.end());
    vector<point>uh;
    for(ll i=0 ; i<n ; i++) {
        while(uh.size()>1 && orient(uh[uh.size()-2],uh[uh.size()-1],vp[i])>0) uh.pop_back();
        uh.push_back(vp[i]);
    }
    vector<point>lh;
    for(ll i=n-1 ; i>=0 ; i--) {
        while(lh.size()>1 && orient(lh[lh.size()-2],lh[lh.size()-1],vp[i])>0) lh.pop_back();
        lh.push_back(vp[i]);
    }
    vector<point>ans;
    for(ll i=0 ; i<uh.size() ; i++) {
        ans.push_back(uh[i]);
    }
    for(ll i=1 ; i<lh.size()-1 ; i++) {
        ans.push_back(lh[i]);
    }
    return ans;
}
vector<point> convex_Hull_point2(vector<point>&vp){
    sort(vp.begin(), vp.end());
    int m = 0, n = vp.size();


    vector<point> hull(n+n+2);
    for(int i=0; i<n; i++){
        for( ; m > 1 and cross(hull[m-1]-hull[m-2], vp[i]-hull[m-2]) <= 0; m-- );
        hull[m++] = vp[i];
    }
    for(int i = n-2, k = m; i >= 0; i--){
        for( ; m>k and cross(hull[m-1]-hull[m-2], vp[i]-hull[m-2]) <= 0; m--);
        hull[m++] = vp[i];
    }
    if(n>1)
        m--;
    while(hull.size() > m)
        hull.pop_back();
    return hull;
}
double maximum_distance_between_any_two_point (vector<point>&vpp) {
    vector<point>vp = convex_Hull_point2(vpp);
    ll n = vp.size();
    ll ans = -1e18, j = 1;
    for(ll i=0 ; i<n ; i++) {
        while(abs(cross((vp[(i+1)%n]-vp[j+1]),(vp[i]-vp[j+1]))) > abs(cross((vp[(i+1)%n]-vp[j]),(vp[i]-vp[j])))) j = (j+1)%n;
        ans = max(ans,des(vp[j],vp[i]));
        ans = max(ans,des(vp[j],vp[(i+1)%n]));
    }
    return sqrt((double)ans);
}
bool cmpr(point a,point b) {
    ll x = max(a.x,a.y);
    ll y = max(b.x,b.y);
    return (x>y);
}
```

```
ll minimum_distance_between_any_two_point
(vector<point>&vp) {
    ll n = vp.size();
    vector<point>vpx = vp;
    sort(vp.begin(),vp.end());
    ll ans = LONG_LONG_MAX;
    for(ll i=1 ; i<n ; i++) {
        ans = min(ans,des(vp[i],vp[i-1]));
    }
    sort(vpx.begin(),vpx.end(),cmpr);
    for(ll i=1 ; i<n ; i++) {
        ans = min(ans,des(vpx[i],vpx[i-1]));
    }
    return ans;
```

///BIT

```
const int N = 3e5 + 9;
template <class T>
struct BIT { //1-indexed
int n; vector<T> t;
BIT() {}
BIT(int _n) {
n = _n; t.assign(n + 1, 0);
}
T query(int i) {
T ans = 0;
for (; i >= 1; i -= (i & -i)) ans += t[i];
return ans;
}
void upd(int i, T val) {
if (i <= 0) return;
for (; i <= n; i += (i & -i)) t[i] += val;
}
void upd(int l, int r, T val) {
upd(l, val);
upd(r + 1, -val);
}
T query(int l, int r) {
return query(r) - query(l - 1);
}
};
```

// Trie

```
struct Node
{
Node *link[27];
bool flag = false;
};
class Trie
{
private:
Node *root;
public:
Trie()
{
root = new Node();
}
void insert(string s)
{
Node *node = root;
for (int i = 0; i < s.size(); i++)
{
if (node->link[s[i] - 'a'] == NULL)
{
node->link[s[i] - 'a'] = new Node();
}
node = node->link[s[i] - 'a'];
}
node->flag = true;
}
bool find(string s)
{ Node *node = root;
for (int i = 0; i < s.size(); i++)
{
if (node->link[s[i] - 'a'] == NULL)
{
return false;
}
node = node->link[s[i] - 'a'];
}
return node->flag;
}
bool findPrefix(string s)
{
Node *node = root;
for (int i = 0; i < s.size(); i++)
{
if (node->link[s[i] - 'a'] == NULL)
{
return false;
}
node = node->link[s[i] - 'a'];
}
return true;} Trie a; a.insert("Nayem");
}
};
```

///Bitset pattern finding

```
// two type query 1st type change s[x] = ch, 2nd type how
many substring = text in range s[l]...s[r]

bitset<100009> bit[27], start, start1;
string s;
cin >> s;
for (i = 0; i < s.size(); i++)
{
bit[s[i] - 'a'].set(i);
}
start1.set();
ll q;
```

```
cin >> q;
while (q--)
{
int f;
cin >> f;
if (f == 1)
{
char ch;
cin >> x >> ch;
bit[s[x - 1] - 'a'][x - 1] = 0;
bit[ch - 'a'].set(x - 1);
s[x - 1] = ch;
continue;
}
string t;
cin >> l >> r >> t;
if (r - l + 1 < t.size())
{
cout << 0 << nl;
continue;
}
start = start1;
for (i = 0; i < t.size(); i++)
{
start &= (bit[t[i] - 'a'] >> i);
}
cout << (start >> (l - 1)).count() - (start >> (r - t.size() +
1)).count() << nl;
}
```

////////////// Mobious Function <==

```
int mob[N + 3];
int mobius(int n)
{
mob[1] = 0;
for (int i = 2; i <= n; i++)
{
int prev = -1;
int x = i;
int cnt = 0;
while (1)
{
if (prev == spf[x])
{
mob[i] = 0;
break;
}
prev = spf[x];
x /= spf[x];
cnt++;
if (x == 1)
{
mob[i] = (cnt % 2 == 0 ? 1 : -1);
break;
}
}
}
}
```

///Extended Euclidean ax + by = gcd(a, b)

```
//a * x + b * y = gcd(a, b);
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
ll xx = y = 0;
ll yy = x = 1;
while (b) {
ll q = a / b;
ll t = b; b = a % b; a = t;
t = xx; xx = x - q * xx; x = t;
t = yy; yy = y - q * yy; y = t;
}
return a;
}
```

///CRT

```
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y) {
T xx = y = 0;
T yy = x = 1;
while (b) {
T q = a / b;
T t = b; b = a % b; a = t;
t = xx; xx = x - q * xx; x = t;
t = yy; yy = y - q * yy; y = t;
}
return a;
}
// finds x such that x % m1 = a1, x % m2 = a2. m1 and m2
may not be coprime
// here, x is unique modulo m = lcm(m1, m2). returns (x,
m). on failure, m = -1.
pair<T, T> CRT(T a1, T m1, T a2, T m2) {
T p, q;
T g = extended_euclid(m1, m2, p, q);
if (a1 % g != a2 % g) return make_pair(0, -1);
T m = m1 / g * m2;
p = (p % m + m) % m;
q = (q % m + m) % m;
return make_pair((p * a2 % m * (m1 / g) % m + q * a1 % m *
(m2 / g) % m) % m, m);
}

int32_t main() {
ios_base::sync_with_stdio(0);
cin.tie(0);
cout << (int)CRT(1, 31, 0, 7).first << '\n';
return 0;
}
```

///Matrix Exponentiation

```
const int N = 1e6;
const int MOD = 1e9 + 7;
```

```
struct Matrix {
int rows, cols;
vector<vector<int>> mat;
Matrix(int r, int c) : rows(r), cols(c) {
mat.assign(r, vector<int>(c, 0));
}
static inline Matrix multiply(const Matrix &A, const Matrix
&B) {
int r = A.rows, c = B.cols, n = A.cols;
Matrix result(r, c);

for (int i = 0; i < r; i++) {
for (int j = 0; j < c; j++) {
for (int k = 0; k < n; k++) {
result.mat[i][j] = (result.mat[i][j] + A.mat[i][k] * B.mat[k][j]) %
MOD;
}
}
}
return result;
}
static inline Matrix power(Matrix base, int exp) {
Matrix result(base.rows, base.cols);
for (int i = 0; i < base.rows; i++) {
result.mat[i][i] = 1;
}
while (exp > 0) {
if (exp % 2 == 1)
result = multiply(result, base);
base = multiply(base, base);
exp /= 2;
}
return result;
}
void display() const {
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
cout << mat[i][j] << " ";
}
cout << endl;
}
}
};
int32_t main() {
int n;
cin >> n;
Matrix M(2, 2), F(2, 1);
M.mat = {{0, 1}, {1, 1}};
F.mat = {{0}, {1}};
M = Matrix::power(M, n);
Matrix Result = Matrix::multiply(M, F);
cout << Result.mat[0][0] << endll;
return 0;
}
```

LCA:

```
int lg = 20;
vector<vector<ll>> anc, g; // anc[u][i] = (2^i)th ancestor of
node u
vector<ll> dep;
void dfs(ll u, ll par)
{
if (par != -1)
{ anc[u][0] = par;
dep[u] = dep[par] + 1;
for (int i = 1; i < lg; i++)
{
int v = anc[u][i - 1];
if (v == -1)
break;
anc[u][i] = anc[v][i - 1]; // 16th ancestor of node u = 8th
ancestor of node(8th ancestor of node u)
}
}
for (auto v : g[u])
{ if (v != par)
dfs(v, u);
}
}
ll kth(ll u, ll k)
{
for (int i = 0; i < lg; i++)
{
if ((1LL << i) & k)
u = anc[u][i];
if (u == -1)
return -1;
}
return u;
}
ll lca(ll u, ll v)
{
if (dep[u] > dep[v])
u = kth(u, dep[u] - dep[v]);
else if (dep[v] > dep[u])
v = kth(v, dep[v] - dep[u]);
if (u == v)
return u;
for (int i = lg - 1; i >= 0; i--)
{
if (anc[u][i] != anc[v][i])
{
u = anc[u][i];
v = anc[v][i];
}
}
return anc[u][0];
}
ll distance(ll u, ll v)
{
ll l = lca(u, v);
return dep[u] + dep[v] - 2 * dep[l];
}
```

```
ll go(ll u, ll v, ll k) // kth node from u to v , 0th node is u
{
ll l = lca(u, v);
ll dis = dep[u] + dep[v] - 2 * dep[l];
if (k > dis)
return -1; // not possible
if (dep[l] + k <= dep[u])
return kth(u, k);
k -= dep[u] - dep[l];
return kth(v, dep[v] - dep[l] - k);
}
ll lca_for_given_root(ll r, ll u, ll v) // Lca for given root
{
ll a = lca(u, v);
ll b = lca(r, u);
ll c = lca(r, v);
if (a == b)
return c;
if (b == c)
return a;
return b;
}
```

Combinatories-

- nPr =n!/(n-r)!
- nCr = n!(r!*(n-r)!)
- nCr+nCr-1=n+1Cr
- nCx=nCy hole x+y=n
- nC1+nC2+….+nCn = 2^n-1;
- nC2+nC4+…..+nCn-1=2^(n-1) n is odd
- nC2+nC4+…..+nCn=2^(n-1)-1

catalan Numbers

```
// 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862,
// 16796, 58786, 208012, 742900, 2674440
ll dp[M], fac[2 * M];
void fact(){
fac[0] = 1;
for (int i = 1; i < 2 * M; i++)
fac[i] = (fac[i - 1] * i) % mod;
}
void cal(){ /// O(n*logn)
dp[0] = dp[1] = 1; /// x = (2*x)!/((x+1)!*x!)
for (int i = 2; i < M; i++)
dp[i] = (fac[2 * i] * bigmod((fac[i + 1] * fac[i]) % mod,
mod - 2, mod)) % mod;
}
```

Inclusion - Exclusion

```
/// count int divisible by at least one element of v[m]
for (mask = 1; mask < (1 << m); mask++){
cnt = 0, res = 1;
for (j = mask, k = 0; j > 0; j = (j >> 1), k++){
if (j & 1){
cnt++;
res = res / (__gcd(res, v[k])) * v[k];
if (res > n) break;
}
}
res = n / res;
if (__builtin_popcount(mask) % 2 == 1) ans += res;
/// if(cnt&1) ans += res; ///another way
else ans -= res;
}
```

Euler tour (Tree -> Array)

```
vector<int> dfsarr;
int st[N], en[N]; ll a[N];
// node i subtree { dfsarr[st[i],en[i]] }
void dfs(int u, int par){ // 1 based
dfsarr.pb(u);
st[u] = sz(dfsarr) - 1; // starting idx of node u in dfs
array
for (int v : edge[u]){
if (v == par) continue;
dfs(v, u);
}
en[u] = sz(dfsarr) - 1; // ending idx of node u in dfs
array
}
```

Arithmetic progression : a + (d+a) + (2d+a) ..

a. **Nth** term : a + (n-1)d

b. **Sum** of n term : n/2[2a + (n-1)*d]

c. **Sum** of AP : n/2[1st term + nth term]

Geometric progression: a + ar + ar^2 + ar^(n-1)

a. **Nth** term : ar(n-1)

b. **Sum** of n term : a[(r^n - 1)/(r-1)]

c. **Sum** of n term : (nth*r - 1st)/(r-1)

HashTable

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now().
time_since_epoch().count();
unsigned hash_f(unsigned x) {
x = ((x >> 16) ^ x) * 0x45d9f3b;
x = ((x >> 16) ^ x) * 0x45d9f3b;
return x = (x >> 16) ^ x;
}
unsigned hash_combine(unsigned a, unsigned b) { return a
* 31 + b; }
struct chash {
int operator()(int x) const { return hash_f(x); }
};
typedef gp_hash_table<int, int, chash> gp;
```

gp table;

🔹 Binomial & Combinatorics

- Symmetry: C(n,k) = C(n, n-k)
- Pascal’s Rule: C(n,k) + C(n,k+1) = C(n+1, k+1)
- Weighted Sum: Σ k*C(n,k) = n * 2^(n-1)
- Binomial Expansion: (x+y)^n = Σ C(n,k) * x^k * y^(n-k)
- Vandermonde’s Identity: Σ C(r,i)*C(s, k-i) = C(r+s, k)
- Fibonacci Connection: Σ C(n-k, k) = F(n+1)
- Catalan Numbers: Cn = 1/(n+1) * C(2n, n)
- Hockey Stick Identity: Σ_{i=r}^n C(i,r) = C(n+1, r+1)