**GP hash table:**

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

using namespace \_\_gnu\_pbds;

const int RANDOM = chrono::high\_resolution\_clock::now().time\_since\_epoch().count();

struct chash

{

int operator()(int x) const

{

return x ^ RANDOM;

}

};

gp\_hash\_table<int, int, chash> table;

**Hashing:**

**Hashing-with-segment-tree:\*\*\*\***

**const int** N = 2e5 + 9;  
ll modInverse(ll a, ll m) *// calculates inverse modulo*{  
 ll m0 = m;  
 ll y = 0, x = 1;  
 **if** (m == 1)  
 {  
 **return** 0;  
 }  
 **while** (a > 1)  
 {  
 ll q = a / m;  
 ll t = m;  
 m = a % m, a = t;  
 t = y;  
 y = x - q \* y;  
 x = t;  
 }  
 **if** (x < 0)  
 {  
 x += m0;  
 }  
 **return** x;  
}  
ll p\_pow[N]; *// p\_pow[i] = p^i % m*ll modIn[N]; *// inverse modulo for p\_pow[]*ll hash\_val[2][N]; *// prefix sum of hash values***const** ll p = 31; *// base***const** ll m = 1e9 + 7; *// mod***void** compute\_hash(string **const**& s, ll id) *// precomputes hash\_val*{  
 *// hash\_val[id][0] = s[0] - 'a' + 1;* **for** (ll i = 0; i < (ll)s.size(); i++)  
 {  
 *// hash\_val[id][i] = (hash\_val[id][i - 1] + (s[i] - 'a' + 1) \* p\_pow[i]) % m;* hash\_val[id][i] = ((s[i] - **'a'** + 1) \* p\_pow[i]) % m;  
 }  
}  
**void** powCalc(ll n) *// precomputes p\_pow[] and hash\_val[]*{  
 p\_pow[0] = 1;  
 **for** (ll i = 1; i <= n; i++)  
 {  
 p\_pow[i] = (p \* p\_pow[i - 1]) % m;  
 modIn[i] = modInverse(p\_pow[i], m);  
 }  
}  
ll ara[N],tree[2][4\*N];  
**void** build(ll node, ll b, ll e, ll id)  
{  
 **if**(b==e)  
 {  
 tree[id][node]=hash\_val[id][b];  
 **return**;  
 }  
 ll left=node\*2;  
 ll right=node\*2+1;  
 ll mid=(b+e)/2;  
 build(left,b,mid,id);  
 build(right,mid+1,e,id);  
 *// modify this* tree[id][node]=(tree[id][left]+tree[id][right])%m;  
}  
ll query(ll node, ll b, ll e, ll i, ll j, ll id)  
{  
 **if**(b>e || i>e || j<b)  
 {  
 **return** 0;  
 }  
 **if**(b>=i && e<=j)  
 {  
 **return** tree[id][node];  
 }  
 ll left=node\*2;  
 ll right=node\*2+1;  
 ll mid=(b+e)/2;  
 ll p1=query(left,b,mid,i,j,id);  
 ll p2=query(right,mid+1,e,i,j,id);  
 **return** (p1+p2)%m;  
}  
**void** update(ll node, ll b, ll e, ll i, ll value, ll id)  
{  
 **if**(b>e || i>e || i<b)  
 {  
 **return**;  
 }  
 **if**(b==i && e==i)  
 {  
 tree[id][node]=value;  
 **return**;  
 }  
 ll left=node\*2;  
 ll right=node\*2+1;  
 ll mid=(b+e)/2;  
 update(left,b,mid,i,value,id);  
 update(right,mid+1,e,i,value,id);  
 tree[id][node]=(tree[id][left]+tree[id][right])%m;  
}  
ll query\_hash(ll l, ll r, ll n, ll id) *// calculates hash value of substring in range [l, r]*{  
 **if** (r < l)  
 {  
 **return** 0;  
 }  
 **if** (l == 0)  
 {  
 **return** query(1,0,n-1,0,r,id);  
 }  
 **else** {  
 **return** ((((query(1,0,n-1,0,r,id) - query(1,0,n-1,0,l-1,id)) + m) % m) \* modIn[l]) % m;  
 }  
}  
ll getHasVal(ll i, **char** c)  
{  
 ll val=((c-**'a'**+1)\*p\_pow[i])%m;  
 **return** val;  
}  
**void** printTree(ll n)  
{  
 **for**(ll i=0; i<=4\*n; i++) cout<<tree[0][i]<<**", "**; cout<<endl;  
 **for**(ll i=0; i<=4\*n; i++) cout<<tree[1][i]<<**", "**; cout<<endl;  
}  
**void** test()  
{  
 ll n,m; cin>>n>>m;  
 string s0; cin>>s0;  
 string s1=s0;  
 reverse(all(s1));  
 powCalc(n);  
 compute\_hash(s0,0);  
 compute\_hash(s1,1);  
 build(1,0,n-1,0);  
 build(1,0,n-1,1);  
 *// for(ll i=0; i<n; i++) cout<<hash\_val[0][i]<<", "; cout<<endl;  
 // for(ll i=0; i<n; i++) cout<<hash\_val[1][i]<<", "; cout<<endl;  
 // printTree(n);  
 // ll val=query\_hash(0,1,n,0);  
 // debx(val);* **while**(m--)  
 {  
 ll type; cin>>type;  
 **if**(type==1)  
 {  
 ll index; cin>>index;  
 ll tmp=index;  
 index--;  
 **char** c; cin>>c;  
 ll val=getHasVal(index,c);  
 update(1,0,n-1,index,val,0);  
 index = n-tmp+1;  
 index--;  
 val=getHasVal(index,c);  
 update(1,0,n-1,index,val,1);  
 *// printTree(n);* }  
 **else** {  
 ll l,r; cin>>l>>r;  
 l--,r--;  
 ll mid=(l+r)/2;  
 ll bam=mid-l,dan=r-mid;  
 ll mid1;  
 **if**(bam==dan)  
 {  
 mid1 = n-mid-1;  
 }  
 **else** {  
 mid1 = n-mid-2;  
 }  
 ll l1=mid1-bam;  
 ll r1=mid1+dan;  
 *// debx(mid1);  
 // deb2(l1,r1);* ll val0 = query\_hash(l,r,n,0);  
 ll val1 = query\_hash(l1,r1,n,1);  
 *// deb2(val0,val1);* **if**(val0==val1)  
 {  
 cout<<**"YES"**<<endl;  
 }  
 **else** {  
 cout<<**"NO"**<<endl;  
 }  
 }  
 }  
}

**KMP:**

VI computeLPSArray(string pat, VI lps)

{

int len = 0;

lps[0] = len;

int i = 1;

while (i < pat.size())

{

if (pat[len] == pat[i])

++len, lps[i] = len, i++;

else

len != 0 ? (len = lps[len - 1]) : (lps[i] = len, i++);

}

return lps;

}

int KMPSearch(string txt, string pat)

{

int PAT\_SIZE = pat.size();

int TXT\_SIZE = txt.size();

VI lps(PAT\_SIZE, 0);

lps = computeLPSArray(pat, lps);

int count = 0;

int i = 0;

int j = 0;

while ((TXT\_SIZE - i) >= (PAT\_SIZE - j))

{

if (pat[j] == txt[i])

{

j++;

i++;

}

if (j == PAT\_SIZE)

{

count++;

j = lps[j - 1];

}

else if (i < TXT\_SIZE && pat[j] != txt[i])

{

if (j != 0)

j = lps[j - 1];

else

i++;

}

}

return count;

}

**Expected Value:**

*E*(*X*)=*μ*=∑*xP*(*x*)

**Grundy:\*\*\*\***

**const** ll mx=16,my=16;  
ll dx[]={-2,-2,1,-1};  
ll dy[]={1,-1,-2,-2};  
ll dp[mx][my];  
**bool** valid(ll x, ll y)  
{  
 **return** x>=1 && x<=15 && y>=1 && y<=15;  
}  
**bool** losingPos(ll x, ll y)  
{  
 **return** (x==1 && y==1) ||  
 (x==1 && y==2) ||  
 (x==2 && y==1) ||  
 (x==2 && y==2);  
}  
ll grundy(ll x, ll y)  
{  
 **if**(losingPos(x,y))  
 {  
 **return** 0;  
 }  
 **if**(dp[x][y]!=-1)  
 {  
 **return** dp[x][y];  
 }  
 set<ll> s;  
 **for**(ll i=0; i<4; i++)  
 {  
 ll xx=x+dx[i];  
 ll yy=y+dy[i];  
 **if**(valid(xx,yy))  
 {  
 s.insert(grundy(xx,yy));  
 }  
 }  
 ll mex=0;  
 **while**(s.count(mex))  
 {  
 mex++;  
 }  
 **return** dp[x][y]=mex;  
}  
**void** solve()  
{  
 ll n; cin>>n;  
 ll \_xor=0;  
 **for**(ll i=0; i<n; i++)  
 {  
 ll x,y; cin>>x>>y;  
 \_xor^=grundy(x,y);  
 }  
 **if**(\_xor) cout<<**"First"**<<endl;  
 **else** cout<<**"Second"**<<endl;  
}

**Unbounded Knapsack:**int unbknap(int W, int n)

{

int dp[W+1];

memset(dp,0,sizeof dp);

int i,j,ans=0;

for(i=0;i<=W;i++)

{

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

{

if (wt[j]<=i)

dp[i]=max(dp[i],dp[i-wt[j]]+val[j]);

}

}

return dp[W];

}

**No Of Substr div by d:**

**i**nt solve(string const& num,int d)

{

int i,j,ans=0,n=num.length();

vector<vector<int>>dp(n,vector<int>(d,0));

rep0(i,n)

{

int digit=num[i]-'0';

dp[i][digit%d]++;

if(i>0)

{

rep0(j,d)

{

int r=(j\*10+digit)% d;

dp[i][r]+=dp[i-1][j];

}

}

ans+=dp[i][0];

/\* if numbers with preceeding 0 aren't allowed except '0' itself

if(digit == 0)

dp[i][digit % d]--;

\*/

}

return ans;

}

**Probilistic DP:**

**double** dp[105][10005];  
**bool** vis[105][10005];  
**double** func(**int** pos,**int** m,**double** money[],**double** prob[])  
{  
 **if**(m<0)**return** 0.0;  
 **if**(pos==n+1)**return**(m==0?1.0:0.0);  
 **if**(vis[pos][m])**return** dp[pos][m];  
 vis[pos][m]=**true**;  
 **return** dp[pos][m]=max(func(pos+1,m,money,prob),func(pos+1,m-money[pos],money,prob)\*(1-prob[pos]));  
}

**LIS:**

**const** ll N=10005;  
ll ara[N];  
ll dp[N]; *// dp[i] = maximum length of LIS when the last element of LIS is ith element*ll n;  
ll F(ll i, ll j) *// ll i = current index, j = previous index*{  
 **if**(i>n)  
 **return** 0;  
 **if**(dp[j]!=-1)  
 **return** dp[j];  
 ll maxx=0;  
 **if**(ara[i]>ara[j])  
 {  
 maxx=max(maxx,F(i+1,i)+1);  
 }  
 maxx=max(maxx,F(i+1,j));  
 **return** dp[j]=maxx;  
}

**LCS:**

**const int** N=105;  
**int** dp[N][N];  
string a,b;  
**int** na,nb;  
**char** lcs[N];  
**int** cnt=0;  
set<string> st;  
**void** dfs(**int** i, **int** j, **int** k)  
{  
 **if**(i<=0 || j<=0)  
 {  
 string s;  
 **for**(**int** ii=k; ii>=1; ii--)  
 {  
 s+=lcs[ii];  
 }  
 st.insert(s);  
 **return**;  
 }  
 **if**(a[i]==b[j])  
 {  
 lcs[k+1]=a[i];  
 dfs(i-1,j-1,k+1);  
 }  
 **else if**(dp[i][j-1]>dp[i-1][j])  
 {  
 dfs(i,j-1,k);  
 }  
 **else if**(dp[i-1][j]>dp[i][j-1])  
 {  
 dfs(i-1,j,k);  
 }  
 **else** {  
 dfs(i-1,j,k);  
 dfs(i,j-1,k);  
 }  
}  
**void** solve()  
{  
 cin>>a>>b;  
 na=(**int**)a.size(),nb=(**int**)b.size();  
 a=**"."**+a,b=**"."**+b;  
 **for**(**int** i=1; i<=na; i++)  
 {  
 **for**(**int** j=1; j<=nb; j++)  
 {  
 **if**(a[i]==b[j])  
 {  
 dp[i][j]=dp[i-1][j-1]+1;  
 }  
 **else** {  
 dp[i][j]=max(dp[i-1][j],dp[i][j-1]);  
 }  
 }  
 }  
 cout<<dp[na][nb]<<endl;  
 dfs(na,nb,0);  
}

**Edit Distance:**

int dp[2010][2010];

int min3(int a,int b,int c)

{

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

}

int main()

{

string a,b;

cin>>a>>b;

int m=a.size(),n=b.size();

rep0(i,m)

dp[i][0]=i;

rep0(j,n)

dp[0][j]=j;

rep(i,m)

{

rep(j,n)

{

if(a[i-1]==b[j-1])

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

else

dp[i][j]=1+min3(dp[i-1][j],dp[i][j-1],dp[i-1][j-1]);

}

}

cout<<dp[m][n];

}

**LCIS:**

**int** arr1[N + 1], arr2[N + 1];  
**int** dp[N + 1];  
**int** LCIS(**int** n, **int** m)  
{  
 memset(dp, 0, **sizeof**(dp));  
 **for** (**int** i = 0; i < n; i++)  
 {  
 **int** mx = 0;  
 **for** (**int** j = 0; j < m; j++)  
 {  
 **if** (arr1[i] == arr2[j])  
 {  
 dp[j] = max(dp[j], mx + 1);  
 }  
 **else if** (arr1[i] > arr2[j])  
 {  
 mx = max(mx, dp[j]);  
 }  
 }  
 }  
 **int** res = 0;  
 **for** (**int** i = 0; i < m; i++)  
 {  
 res = max(res, dp[i]);  
 }  
 **return** res;  
}

**Formulas:**

sum(i^2) = (n(n+1)(2n+1))/6  
sum(i^3) = (n^2)((n+1)^2)/4  
a, a + d, a + 2d, ... , a + (n - 1) d = (n/2)[2a+(n−1)d]  
a, ar, ar2, ... , arn – 1 = a(1−r^n)/(1−r)

digits of n base k : floor(log2(n) / log2(k)) + 1

**Coin-1:**

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to produce a sum of money x **using** the available coins in such a way that the number of coins is minimal.  
**const** ll N=1000001;  
ll dp[N];  
ll ara[101];  
ll n;  
**void** solve()  
{  
 ll sum; cin>>n>>sum;  
 **for**(ll i=1; i<=n; i++)  
 {  
 cin>>ara[i];  
 dp[ara[i]]=1;  
 }  
 **for**(ll i=1; i<=sum; i++)  
 {  
 dp[i]=inf;  
 **for**(ll j=1; j<=n; j++)  
 {  
 **if**(i-ara[j]>=0)  
 {  
 dp[i] = min(dp[i],dp[i-ara[j]]+1);  
 }  
 }  
 }  
 **if**(dp[sum]==inf) dp[sum]=-1;  
 cout<<dp[sum]<<endl;  
}

**Coin-2:**

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ways you can produce a money sum x **using** the available coins.  
**const** ll N=1000001;  
ll dp[N];  
ll ara[101];  
ll n;  
**void** solve()  
{  
 ll sum; cin>>n>>sum;  
 **for**(ll i=1; i<=n; i++)  
 {  
 cin>>ara[i];  
 }  
 dp[0]=1;  
 **for**(ll i=1; i<=sum; i++)  
 {  
 dp[i]=0;  
 **for**(ll j=1; j<=n; j++)  
 {  
 **if**(i-ara[j]>=0)  
 {  
 dp[i]+=dp[i-ara[j]];  
 dp[i]%=MOD;  
 }  
 }  
 }  
 cout<<dp[sum]<<endl;  
}

**Coin-3:**

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ordered ways you can produce a money sum x **using** the available coins.  
**void** solve()  
{  
 **int** n,sum; cin>>n>>sum;  
 vector<**int**> ara(n+1);  
 **for**(ll i=1; i<=n; i++)  
 {  
 cin>>ara[i];  
 }  
 **int** dp[n+1][sum+1];  
 memo(dp,0);  
 dp[0][0]=1;  
 **for**(ll i=1; i<=n; i++)  
 {  
 **for**(ll j=0; j<=sum; j++)  
 {  
 dp[i][j]=dp[i-1][j];  
 **if**(j-ara[i]>=0)  
 {  
 dp[i][j]+=dp[i][j-ara[i]];  
 dp[i][j]%=MOD;  
 }  
 }  
 }  
 cout<<dp[n][sum]<<endl;  
}

**Unlucky Numbers DP:**

bool dp[20][10][10][2][2];

void ye(string str, int idx, int mxd, int mnd, bool f, bool ff)

{

if(idx>=ss.size())

{

if(mn>mxd-mnd)

{

mn=mxd-mnd;

ans\_str=str;

}

return;

}

bool &val=dp[idx][mxd][mnd][f][ff];

if(val)return;

val=true;

if(f&&ff)

{

for(int ind=0; ind<10; ind++)

{

str[idx]=ind+'0';

ye(str,idx+1,max(mxd,ind),min(mnd,ind),f,ff);

}

}

else if(f)

{

for(int ind=s[idx]-'0'; ind<=9; ind++)

{

str[idx]=ind+'0';

ye(str,idx+1,max(mxd,ind),min(mnd,ind),f,ind>(s[idx]-'0'));

}

}

else if(ff)

{

for(int ind=0; ind<=ss[idx]-'0'; ind++)

{

str[idx]=ind+'0';

ye(str,idx+1,max(mxd,ind),min(mnd,ind),ind<(ss[idx]-'0'),ff);

}

}

else

{

for(int ind=s[idx]-'0'; ind<=ss[idx]-'0'; ind++)

{

str[idx]=ind+'0';

ye(str,idx+1,max(mxd,ind),min(mnd,ind),ind<(ss[idx]-'0'),ind>(s[idx]-'0'));

}

}

}

**Rectengale-Cutting:**

**for**(**int** i=0; i<a; i++)  
{  
 val=min(val,ye(a-i,b)+ye(i,b)+1);  
}  
**for**(**int** i=0; i<b; i++)  
{  
 val=min(val,ye(a,b-i)+ye(a,i)+1);  
}

**0/1 KNAPSACK:**

int knapsack(int W,int wt[],int val[],int n)

{

int i,w,K[n+1][W+1];

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

{

for (w=0;w<=W;w++)

{

if (i==0 || w==0)

K[i][w]=0;

else if (wt[i-1]<=w)

K[i][w]=max(val[i-1]+K[i-1][w-wt[i-1]],K[i-1][w]);

else

K[i][w]=K[i-1][w];

}

}

return K[n][W];

}

**Digit DP:**

ll dp[11][2][92];

int a,b;

ll func(int pos, bool isSmall, int digitSum, string &str)

{

if(pos>=10)return digitSum==x;

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

int lo=0,hi=str[pos]-'0';

ll re=0;

if(isSmall)hi=9;

for(int idx=lo; idx<=hi; idx++)

{

ll val=func(pos+1,isSmall|(idx<hi), digitSum+idx,str);

re+=val;

}

return dp[pos][isSmall][digitSum]=re;

}

int main()

{

iamspeed

string str,str2;

cin>>a>>b>>x;

str2=to\_string(b);

while(str2.size()<10)str2='0'+str2;

str=to\_string(a-1);

while(str.size()<10)str='0'+str;

memset(dp,-1,sizeof(dp));

ans=func(0,0,0,str2);

memset(dp,-1,sizeof(dp));

sum=func(0,0,0,str);

cout<<ans-sum<<endl;

return 0;

}

**LIS nlogn:**

**Int** LongestIncreasingSubsequenceLength(std::vector<**int**>& v)  
{  
 **if** (v.size() == 0) *// boundary case* **return** 0;  
 std::vector<**int**> tail(v.size(), 0);  
 **int** length = 1; *// always points empty slot in tail* tail[0] =v[0];  
 **for** (**int** i = 1; i < v.size(); i++) {  
 **auto** b = tail.begin(), e = tail.begin() + length;  
 **auto** it = lower\_bound(b, e, v[i]);  
 **if** (it == tail.begin() + length&&v[i]==tail[length-1].first+1)  
 tail[length++] = v[i];  
 **else** \*it=v[i];  
 }  
 **return** length;  
}

**Binary-Decimal-Binary:**

string lltobinary(ll x) {std::string binary = std::bitset<64>(x).to\_string(); **return** binary;}  
ll toNumber(string s) {stringstream aa(s);ll mm;aa>>mm; **return** mm;}

**Sparse Table:**

int table[N][17];

int ara[N],k=17,n;

void sparse()

{

int i,j;

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

table[i][0]=ara[i];

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

{

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

//table[i][j]=max(table[i][j-1], table[i+(1<<(j-1))][j-1]);->max //table[i][j]=min(table[i][j-1], table[i+(1<<(j-1))][j-1]);->min

}

}

int query(int L,int R)

{

int j,ans=-1;

for(j=k;j>=0;j--)

{

if(L+(1<<j)-1<=R)

{

//ans=max(ans,table[L][j]);->max //ans=min(ans,table[L][j]);->min

L+=1<<j;

}

}

return ans;}

**Backward Hash:**

ll b\_hash\_val[N];  
**void** compute\_b\_hash(**const** string& s)  
{  
 ll n = sz(s);  
 b\_hash\_val[n-1] = s[n-1] - **'a'** + 1;  
 **for** (ll i = n-2, j = 1; i >= 0; i--, j++)  
 {  
 b\_hash\_val[i] = (b\_hash\_val[i + 1] + (s[i] - **'a'** + 1) \* p\_pow[j]) % m;  
 }  
}  
ll back\_query\_hash(ll l, ll r, ll n)  
{  
 **if** (r < l)  
 {  
 **return** 0;  
 }  
 **if** (r == n-1)  
 {  
 **return** b\_hash\_val[l];  
 }  
 **else** {  
 **return** ((((b\_hash\_val[l] - b\_hash\_val[r + 1]) + m) % m) \* modIn[n-1-r]) % m;  
 }  
}

**3 Points in same line:**

**bool** check(**const** pll& a, **const** pll& b, **const** pll& c)  
{  
 **return** (b.ff-a.ff)\*(c.ss-a.ss) - (b.ss-a.ss)\*(c.ff-a.ff) == 0;  
}

**SOS DP:**

**bool** vis[N][33];  
**int** freq[N],dp[N][33];  
**int** countBit(**int** x)  
{  
 **int** cnt=1;  
 **while**(x) cnt++, x/=2;  
 **return** cnt;  
}  
**bool** isSet(**int** mask, **int** curBit)  
{  
 **return** (**bool**)(mask&(1<<curBit));  
}  
**int** changeBit(**int** mask,**int** curBit)  
{  
 **return** (mask^(1<<curBit));  
}  
**int** func(**int** mask, **int** curBit)  
{  
 **if**(curBit==-1)**return** (freq[mask]==0?-1:mask);  
 **if**(vis[mask][curBit])**return** dp[mask][curBit];  
 vis[mask][curBit]=**true**;  
 **int** fir=func(mask,curBit-1);  
 **if**(isSet(mask,curBit))  
 {  
 **int** sec=func(changeBit(mask,curBit),curBit-1);  
 **return** dp[mask][curBit]=max(fir,sec);  
 }  
 **return** dp[mask][curBit]=fir;  
}  
**int** main()  
{  
 cin>>n; **int** a[n];  
 **for**(**int** i=0; i<n; i++)  
 {  
 cin>>a[i];  
 freq[a[i]]++;  
 }  
 **for**(**int** i=0; i<n; i++)  
 {  
 **int** revNum=((1<<countBit(a[i]))-1)^(a[i]);  
 **int** submask=func(revNum,countBit(a[i]));  
 cout<<a[i]<<**' '**<<submask<<endl;  
 }  
}

**MCM DP:**

**int** MatrixChainOrder(**int** p[],**int** i, **int** j)  
{  
 **if**(i==j)**return** dp[i][j]=0;  
 **if**(dp[i][j]!=-1)**return** dp[i][j];  
 **int** k,mn=INT\_MAX,cnt;  
 **for**(**int** k=i; k<j; k++)  
 {  
 dp[i][k]=MatrixChainOrder(p,i,k);  
 dp[k+1][j]=MatrixChainOrder(p,k+1,j);  
 cnt=dp[i][k]+dp[k+1][j]+p[i-1]\*p[k]\*p[j];  
 **if**(cnt<mn)  
 {  
 mn=cnt;  
 bracket[i][j]=k;  
 }  
 }  
 **return** dp[i][j]=mn;  
}

**Point inside quadrilateral:**

**struct** Point {  
 **double** x, y;  
};  
**bool** isPointInsideTriangle(Point A, Point B, Point C, Point P) {  
 **double** areaABC = abs((B.x - A.x) \* (C.y - A.y) - (C.x - A.x) \* (B.y - A.y));  
 **double** areaPBC = abs((B.x - P.x) \* (C.y - P.y) - (C.x - P.x) \* (B.y - P.y));  
 **double** areaPCA = abs((C.x - P.x) \* (A.y - P.y) - (A.x - P.x) \* (C.y - P.y));  
 **double** areaPAB = abs((A.x - P.x) \* (B.y - P.y) - (B.x - P.x) \* (A.y - P.y));  
 **return** areaPBC + areaPCA + areaPAB == areaABC;  
}  
**bool** isPointInsideQuadrilateral(Point A, Point B, Point C, Point D, Point P) {  
 **return** isPointInsideTriangle(A, B, C, P)  
 || isPointInsideTriangle(A, B, D, P)  
 || isPointInsideTriangle(A, C, D, P)  
 || isPointInsideTriangle(B, C, D, P);  
}

**Trapezium Area:**area = h\*(a+b)/2;

**Area of Polygon:**

Area = (number of sides × length of one side × apothem)/2

Apothem = [(length of one side)/{2 ×(tan(180/number of sides))}]

**Position of a point(c):**

inline bool isLeft(Point a, Point b, Point c)

{

return ((b.x - a.x)\*(c.y - a.y) - (b.y - a.y)\*(c.x - a.x)) > 0;

}

**Convex Hull:**

struct Point

{

llu x, y;

bool operator<(Point p)

{

return x < p.x || (x == p.x && y < p.y);

}

};

llu cross\_product(Point O, Point A, Point B)

{

return (A.x - O.x) \* (B.y - O.y)- (A.y - O.y) \* (B.x - O.x);}

vector<Point> convex\_hull(vector<Point> A)

{

int n = A.size(), k = 0;

if (n <= 3)return A;

vector<Point> ans(2 \* n);

sort(A.begin(), A.end());

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

{

while (k >= 2&& cross\_product(ans[k - 2], ans[k - 1], A[i])<= 0)k--;

ans[k++] = A[i];

}

for (size\_t i = n - 1, t = k + 1; i > 0; --i)

{

while (k >= t&& cross\_product(ans[k - 2], ans[k - 1], A[i - 1])<= 0)k--;

ans[k++] = A[i - 1];

}

ans.resize(k - 1);

return ans;

}

**RABIN KARP**

ll hashValue(char c)

{

return ((c - 'a' + 1) \* 7);

}

int RABIN\_KARP(string text, string pat){

ll desiredHash = 0, currentHash = 0;

int M = pat.size();

int N = text.size();

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

desiredHash += hashValue(pat[i]);

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

currentHash += hashValue(text[i]);

int i = 1;

int count = 0;

string ans = text.substr(i - 1, M);

while (i <= (N - M + 1))

{

if (currentHash == desiredHash && ans == pat)

count++;

currentHash -= hashValue(text[i - 1]);

currentHash += hashValue(text[i + M - 1]);

ans.erase(ans.begin());

ans += text[i + M - 1];

i++;

}

return count;

}

**PRIME\_FACTORIZATION**

int numberOfPrime(int n){

int primeCount = 0;

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

int prime = primes[i];

if (n % prime == 0){

primeCount++;

while (n % prime == 0){

n /= prime; }}

if (n <= 1)

break;}

return primeCount;

}

NORMAL\_SEGMENT\_TREE

// Here we are using 1 based indexing

void build(vector<int> arr, int current\_node, int left\_end, int right\_end)

{

// If the current node is a leaf node then store the value of the array at that index

if (left\_end == right\_end)

tree[current\_node] = arr[left\_end];

else

{

int mid = (left\_end + right\_end) / 2;

// THESE FORMULAS ARE FOR 1 BASED INDEXING

int left\_node = current\_node \* 2;

int right\_node = current\_node \* 2 + 1;

build(arr, left\_node, left\_end, mid); // Build the left subtree

build(arr, right\_node, mid + 1, right\_end); // Build the right subtree

tree[current\_node] = max(tree[left\_node], tree[right\_node]); // Update the current node

}

}

// Query the segment tree for the max elements in range [l, r]

int query(int v, int tl, int tr, int l, int r)

{

if (l > r) return -1e9;

if (l == tl && r == tr)

return tree[v];

int tm = (tl + tr) / 2;

return max(query(v \* 2, tl, tm, l, min(r, tm)), query(v \* 2 + 1, tm + 1, tr, max(l, tm + 1), r));

}

// Update the segment tree with a new value at position i

void update(ll v, ll tl, ll tr, ll i, ll val)

{

if (tl == tr)

{

tree[v] = val;

}

else

{

ll tm = (tl + tr) / 2;

if (i <= tm)

{ update(v \* 2, tl, tm, i, val); }

else

{

update(v \* 2 + 1, tm + 1, tr, i, val);

}

tree[v] = tree[v \* 2] + tree[v \* 2 + 1];

}

}

void solve()

{

ll n, q;

cin >> n >> q;

ll v[n];

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

cin >> v[i];}

ll root = 1;

build(v, root, 0, n - 1);

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

{

ll ch = 0, l = 0, r = 0;

cin >> ch >> l >> r;

if (ch == 1){

l--;

update(root, 0, n - 1, l, r);}

else{

l--; r--;

cout << query(root, 0, n - 1, l, r) << endl;} }}

**XOR FROM L TO R**

ll xor\_l\_to\_r(ll l, ll r){

return ((l > 0 ? xor0toN(l - 1) : 0) ^ xor0toN(r));}

**KADANE**

int maxSubArray(vector<int>& nums) {

int sum = 0;

int maxSum = INT\_MIN;

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

{

sum = max(sum + nums[i] , nums[i]);

maxSum = max(maxSum , sum);}

return maxSum;

}

**Count Divisor using sieve**

int divisors[10000005];

void countDivisorUsingSieve(int n){

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

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

divisors[j]++;}

**BITS**

#define leftShift(p) (p << 1)

#define rightShift(p) (p >> 1)

#define SetBit(x, k) (x |= (1LL << k))

#define ClearBit(x, k) (x &= ~(1LL << k))

#define ToggleBit(x, k) (x ^= (1LL << k))

#define CheckBit(x, k) ((x >> k) & 1)