namespace TwoSat{

vector<int> a[N];

void AddClause(int x,bool valX,int y,bool valY){ //x==valX||y==valY

x=x\*2+valX,y=y\*2+valY;

a[x^1].push\_back(y);

a[y^1].push\_back(x);

}

// $x=y$：`AddClause(0,1)`，`AddClause(1,0)`

// $x\not=y$：`AddClause(0,0)`，`AddClause(1,1)`

// $x=y=1$：`AddClause(0,1)`，`AddClause(1,0)`，`AddClause(1,1)`

// $x=y=0$：`AddClause(0,1)`，`AddClause(1,0)`，`AddClause(0,0)`

stack<int> s;bool mark[N];

bool DFS(int u){

if(mark[u^1])return 0;

if(mark[u])return 1;

s.push(u);mark[u]=1;

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

if(!DFS(a[u][i]))return 0;

return 1;

}

bool Solve(int n){

memset(mark,0,sizeof(mark));

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

if(!mark[i]&&!mark[i^1])

if(!DFS(i)){

while(s.top()!=i)mark[s.top()]=0,s.pop();

mark[i]=0,s.pop();

if(!DFS(i^1))return 0;

}

return 1;

}

};

namespace GaussElimination{

void GaussElimination(){

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

int cur=i;

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

if(fabs(a[j][i])>fabs(a[cur][i]))cur=j;

for(int j=0;j<=n;j++)swap(a[i][j],a[cur][j]);

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

for(int k=n;k>=i;k--)

a[j][k]-=a[i][k]\*a[j][i]/a[i][i];

}

for(int i=n-1;i>=0;i--){

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

a[i][n]-=a[j][n]\*a[i][j];

a[i][n]/=a[i][i];

}

}

};

namespace MobiusInversion{

void InitMu(){

static bool notPri[N];

static int pri[N];

mu[1]=1;

for(int i=2;i<N;i++){

if(!notPri[i])

pri[++pri[0]]=i, mu[i]=-1;

for(int j=1; j<=pri[0] && i\*pri[j]<N; j++){

int nxt=i\*pri[j]; notPri[nxt]=1;

if(i%pri[j])mu[nxt]=-mu[i];

else{

mu[nxt]=0; break;

}

}

}

};

namespace NetworkFlow{

struct Dinic{

const static int V=N\*2;

struct Edge{int v,res;};

vector<Edge> edg;

vector<int> a[V];

int st,ed;

void AddEdge(int u,int v,int cap){

edg.push\_back((Edge){v,cap});

edg.push\_back((Edge){u,0});

int siz=edg.size();

a[u].push\_back(siz-2);

a[v].push\_back(siz-1);

}

int dep[V];

bool BFS(){

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

dep[st]=0;

queue<int> q; q.push(st);

while(!q.empty()){

int u=q.front(); q.pop();

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

Edge& e=edg[a[u][i]];

if(dep[e.v]==-1&&e.res>0){

q.push(e.v),dep[e.v]=dep[u]+1;

}

}

}

return dep[ed]!=-1;

}

int cur[V];

int DFS(int u,int minF){

if(u==ed||minF==0)return minF;

int tmpF,sumF=0;

for(int& i=cur[u];i<a[u].size();i++){

Edge& e=edg[a[u][i]];

if( dep[e.v]==dep[u]+1 && (tmpF=DFS(e.v,min(e.res,minF)))>0 ){

e.res-=tmpF; edg[a[u][i]^1].res+=tmpF;

sumF+=tmpF; minF-=tmpF;

}

if(minF==0)break;

}

return sumF;

}

int MaxFlow(){

int ret=0;

while(BFS()){

memset(cur,0,sizeof(cur));

ret+=DFS(st,INF);

}

return ret;

}

};

};

namespace RabinMiller{

bool RabinMiller(ll n){

if(n==2)return 1;

if(n<=1||!(n&1))return 0;

int p=0;ll a;

while((n-1)%(1<<(p+1))==0)p++;

a=(n-1)/(1<<p);

for(int i=1;i<=UPP;i++){ //UPP为测试次数

int bas=rand()%(n-2)+2;

for(int j=p;j>=0;j--){

ll tmp=QPow(bas,a\*((ll)1<<j),n);

if(tmp==1)continue;

else if(tmp==n-1)break;

else return 0;

}

}

return 1;

}

};

namespace Eratothenes{

void Eratosthenes(){

int upp=sqrt(N);

for(int i=2;i<=upp;++i)

if(!notPri[i])

for(int j=i\*i;j<N;j+=i)

notPri[j]=1;

}

};

namespace Euler{

void Euler(){

static bool notPri[N];

static int pri[N];

for(int i=2;i<N;i++){

if(!notPri[i])pri[++pri[0]]=i;

for(int j=1; j<=pri[0] && i\*pri[j]<N; j++){

notPri[i\*pri[j]]=1;

if(i%pri[j]==0)break;

}

}

};

namespace Phi{

void InitPhi(){

for(int i=1;i<N;++i)phi[i]=i;

for(int i=2;i<N;++i)

if(phi[i]==i)

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

phi[j]-=phi[j]/i;

}

int Phi(int x){

int ret=x，upp=int(sqrt(x)+1);

for(int i=2;i<=upp;++i)

if(x%i==0){

ret-=ret/i;

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

}

if(x>1)ret-=ret/x;

return ret;

}

};

namespace BSGS{

int BSGS(ll a,ll b,ll p){

map<ll,int> s;

int m=(int)ceil(sqrt(p));

ll tmp=1;

for(int i=0;i<m;i++,tmp=tmp\*a%p)

if(!s.count(tmp))s[tmp]=i;

ll inv=Invert(tmp,p);tmp=b;

for(int i=0;i<m;i++,tmp=tmp\*inv%p)

if(s.count(tmp))return i\*m+s[tmp]+1;

return -1;

}

};

namespace ExGCD{

void ExtendGCD(int a,int b,int &x,int &y,int &g){

if(!b)x=1,y=0,g=a;

else ExtendGCD(b,a%b,y,x,g),y-=x\*(a/b);

}

//$x=x\_0+b',y=y\_0-a'$

//min $x\_+=(x\_0\bmod{b'}+b')\bmod{b'c'}$

};

namespace ExCRT{

ll ExtendCRT(){

ll a0,p0,a1,p1; bool flag=1;

cin>>p0>>a0;

for(int i=2;i<=n;i++){

ll x,y,g,c;

cin>>p1>>a1;

if(flag){

ExtendGCD(p0,p1,x,y,g);

c=a1-a0;

if(c%g){flag=0;continue;}

x=x\*(c/g)%(p1/g);

a0+=x\*p0;p0=p0\*p1/g;

a0%=p0;

}

}

if(flag)return (a0%p0+p0)%p0;

else return -1;

}

};

namespace QMul{

ll QMul(ll a,ll b){

if(a>b)swap(a,b);

ll ret=0;

for(;b;b>>=1,(a<<=1)%=p)

if(b&1)(ret+=a)%=p;

return ret;

}

};

namespace KMP{

int Init(int f[],string s){

f[0]=f[1]=0;

for(int i=1;i<s.length();i++) {

int j=f[i];

while(j&&s[i]!=s[j])j=f[j];

f[i+1] = (s[i]==s[j])?j+1:0;

}

}

int Query(string s,string t,int f[]){

int cnt=0,j=0;

for(int i=0;i<s.length();i++){

while(j&&s[i]!=t[j])j=f[j];

if(s[i]==t[j])j++;

if(j==t.length())cnt++;

}

return cnt;

}

};

namespace AhoCorasick{

int ch[N][C],f[N],pre[N];

int isEnd[N]; int idx;

void Init(){

memset(ch,0,sizeof(ch));

memset(f,0,sizeof(f));

memset(pre,0,sizeof(pre));

memset(isEnd,0,sizeof(isEnd));

idx=0;

}

void Build(char s[]){

int n=strlen(s),o=0;

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

int c=Idx(s[j]);

if(ch[o][c])o=ch[o][c];

else o=ch[o][c]=++idx;

}

isEnd[o]=i;

}

void GetFail(){

queue<int> q;

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

if(ch[0][i])q.push(ch[0][i]);

while(!q.empty()){

int h=q.front(); q.pop();

for(int i=0;i<C;i++){

int &u=ch[h][i], j=f[h];

if(!u){

u=ch[j][i];

continue;

}

q.push(u);

while(j&&!ch[j][i])j=f[j];

f[u]=ch[j][i];

pre[u] = isEnd[f[u]]?f[u]:pre[f[u]];

}

}

}

};

namespace SA{

//a \in [0,n)

//$a\_n$ = min(0)

//1 \leq a\_i< m

struct SuffixArray{

int sa[N],hei[N],rnk[N];

void Init(int \*a,int n){

InitSa(a,n);

InitHeight(a,n);

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

sa[i]=sa[i+1];

hei[i]=hei[i+1];

rnk[i]--;

}

}

inline bool Cmp(int \*a,int x,int y,int l){

return a[x]==a[y]&&a[x+l]==a[y+l];

}

void InitSa(int \*a,int n){

int m=26;

static int tmpX[N],tmpY[N],s[N];

int \*x=tmpX,\*y=tmpY;

a[n]=0;

for(int i=0;i<m;i++)s[i]=0;

for(int i=0;i<=n;i++)s[x[i]=a[i]]++;

for(int i=1;i<m;i++)s[i]+=s[i-1];

for(int i=n;i>=0;i--)sa[--s[x[i]]]=i;

for(int i=1,p=1;p<=n;i<<=1,m=p){

p=0;

for(int j=n-i+1;j<=n;j++)y[p++]=j;

for(int j=0;j<=n;j++)if(sa[j]>=i)y[p++]=sa[j]-i;

for(int j=0;j<m;j++)s[j]=0;

for(int j=0;j<=n;j++)s[x[y[j]]]++;

for(int j=1;j<m;j++)s[j]+=s[j-1];

for(int j=n;j>=0;j--)sa[--s[x[y[j]]]]=y[j];

swap(x,y);

p=1,x[sa[0]]=0;

for(int j=1;j<=n;j++)x[sa[j]]=Cmp(y,sa[j-1],sa[j],i)?p-1:p++;

}

}

void InitHeight(int \*a,int n){

for(int i=1;i<=n;i++)rnk[sa[i]]=i;

for(int i=0,j,k=0;i<n;hei[rnk[i++]]=k)

for(k?k--:0,j=sa[rnk[i]-1];a[i+k]==a[j+k];k++);

}

};

};

namespace SAM{

int ch[N][C],pa[N],len[N],siz[N];

int idx=1,pre=1;

void Insert(int x){

int p=pre,np=++idx;pre=np;

siz[np]=1; len[np]=len[p]+1;

for(;p&&ch[p][x]==0;p=pa[p])ch[p][x]=np;

if(p==0)pa[np]=1;

else{

int q=ch[p][x];

if(len[q]==len[p]+1)pa[np]=q;

else{

int nq=++idx; len[nq]=len[p]+1;

memcpy(ch[nq],ch[q],sizeof(ch[q]));

pa[nq]=pa[q]; pa[q]=pa[np]=nq;

for(;p&&ch[p][x]==q;p=pa[p])ch[p][x]=nq;

}

}

}

int tmp[N],topo[N];

void Build(){

for(int i=1;i<=idx;i++)tmp[len[i]]++;

for(int i=1;i<=idx;i++)tmp[i]+=tmp[i-1];

for(int i=1;i<=idx;i++)topo[tmp[len[i]]--]=i;

for(int i=idx;i>1;i--){

int v=topo[i];int u=pa[v];

siz[u]+=siz[v];

}

}

};

namespace Manacher{

ll Manacher(char t[],int n){

static char s[2\*N];

static int cnt[2\*N],f[2\*N];

for(int i=0;i<=n\*2;i++)

cnt[i]=f[i]=0;

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

s[i\*2-1]=t[i-1];

s[i\*2]=1;

}

s[0]=2,s[n\*2]=3;

int cur=f[0]=0,idx=0;

for(int i=1;i<2\*n;i++){

int& j=f[i]; j=0;

if(cur-i>=0&&2\*idx-i>=0)j=min(f[2\*idx-i],cur-i);

while(s[i-j-1]==s[i+j+1])j++;

if(i+j>cur)cur=i+j,idx=i;

//ans=max(ans,(j\*2+1)/2);

cnt[max(0,i-j)]++;

cnt[i+1]--;

}

ll ret=0;

for(int i=1;i<=2\*n;i++){

cnt[i]+=cnt[i-1];

if(i&1)ret+=cnt[i];

}

return ret;

}

};

namespace PAM{

int ch[N][C],pa[N]={1},len[N]={0,-1},siz[N];

int idx=1,pre=0;

void Insert(char s[],int pos){

int p=pre,x=s[pos]-'a';

for(;s[pos-len[p]-1]!=s[pos];)p=pa[p];

if(ch[p][x]==0){

int q=pa[p],np=++idx;

len[np]=len[p]+2;

for(;s[pos-len[q]-1]!=s[pos];)q=pa[q];

pa[np]=ch[q][x]; ch[p][x]=np;

}

pre=ch[p][x]; siz[pre]++;

}

ll Build(){

ll ans=0;

for(int i=idx;i>1;i--){

siz[pa[i]]+=siz[i];

ans=max(ans,1LL\*siz[i]\*len[i]);

}

return ans;

}

int Init(){

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

PAM::Insert(s,i);

printf("%lld",PAM::Build());

}

};

namespace HeavyLightDecomposition{

SegTree t;

int dep[N],siz[N],pa[N],son[N],top[N],idx[N];

int nIdx;

void Build(){

nIdx=dep[0]=siz[0]=son[0]=0;

DFS1(); DFS2();

}

void DFS1(int u=1,int pa=0){

dep[u]=dep[HLDcp::pa[u]=pa]+1;

siz[u]=1;son[u]=0;

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

int v=a[u][i];

if(v==pa)continue;

DFS1(v,u);

if(siz[v]>siz[son[u]])son[u]=v;

siz[u]+=siz[v];

}

}

void DFS2(int u=1,int pa=0){

idx[u]=++nIdx;top[u]=u;

if(son[pa]==u)top[u]=top[pa];

if(son[u])DFS2(son[u],u);

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

int v=a[u][i];

if(v==pa||v==son[u])continue;

DFS2(v,u);

}

}

};

namespace FHQTreap{

struct Node{

int v,w,siz,lazy; ll sum;

Node \*lch,\*rch;

Node(int \_v=0){

v=\_v, w=rand(), siz=1;

sum=v, lazy=0;

lch=rch=nullptr;

}

void Maintain(){

siz=1; sum=v;

if(lch!=nullptr)

siz+=lch->siz,sum+=lch->sum;

if(rch!=nullptr)

siz+=rch->siz,sum+=rch->sum;

}

void Pushdown(){

if((this==nullptr)||lazy==0)return;

if(lch!=nullptr)lch->lazy^=1;

if(rch!=nullptr)rch->lazy^=1;

swap(lch,rch); lazy=0;

}

};

typedef pair<Node\*,Node\*> pNode;

Node mp[N];

struct Treap{

Node \*rt,\*pit;

Treap(){

pit=mp; rt=nullptr;

}

Node\* NewNode(int v){

\*pit=Node(v);

return pit++;

}

void Insert(int v){

Node\* o=NewNode(v);

rt=Merge(rt,o);

}

pNode Split(Node\* o,int k){

pNode ret(nullptr,nullptr);

if(o==nullptr)return ret;

o->Pushdown();

int siz=(o->lch==nullptr)?0:o->lch->siz;

if(k<=siz){

ret=Split(o->lch,k);

o->lch=ret.second;

o->Maintain();

ret.second=o;

}else{

ret=Split(o->rch,k-siz-1);

o->rch=ret.first;

o->Maintain();

ret.first=o;

}

return ret;

}

Node\* Merge(Node\* a,Node\* b){

if(a==nullptr)return b;

if(b==nullptr)return a;

a->Pushdown(); b->Pushdown();

if(a->w < b->w){

a->rch=Merge(a->rch,b);

a->Maintain();

return a;

}else{

b->lch=Merge(a,b->lch);

b->Maintain();

return b;

}

}

void Print(Node\* o){

if(o==nullptr)return;

o->Pushdown();

Print(o->lch);

printf("%d ",o->v);

Print(o->rch);

}

ll Inverse(int L,int R){

pNode a=Split(rt,L-1);

pNode b=Split(a.second,R-L+1);

b.first->lazy^=1; //b一定非空

ll ret=b.first->sum;

rt=Merge(Merge(a.first,b.first),b.second);

return ret;

}

};

};

namespace FFT{

const double PI=acos(-1.0);

struct Complex{

double x,y;

Complex(double \_x=0,double \_y=0){

x=\_x;y=\_y;

}

Complex operator + (Complex a){

return Complex(x+a.x,y+a.y);

}

Complex operator - (Complex a){

return Complex(x-a.x,y-a.y);

}

Complex operator \* (Complex a){

return Complex(x\*a.x-y\*a.y,y\*a.x+x\*a.y);

}

};

void FFT(Complex \*w,int n,int op){

static int r[N];

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

r[i]=(r[i>>1]>>1)|((i&1)?n>>1:0);

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

if(i<r[i])swap(w[i],w[r[i]]);

for(int len=2;len<=n;len<<=1){

int sub=len>>1;

Complex det(cos(PI/sub),op\*sin(PI/sub));

for(int l=0;l<n;l+=len){

Complex rot(1,0);

for(int i=l;i<l+sub;i++){

Complex x=w[i],y=rot\*w[i+sub];

w[i]=x+y;w[i+sub]=x-y;

rot=rot\*det;

}

}

}

}

Complex a[N],b[N];

void Calc(){

int len=1;for(;len<=(n<<1);len<<=1);

FFT(a,len,1);

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

a[i]=a[i]\*a[i];

FFT(a,len,-1);

ans=(int)round(fabs(a[i].y/len/2));

}

};

namespace LinearBase{

const int N=61;

struct LBase{

ll b[N];

LBase(){memset(b,0,sizeof(b));};

void Insert(ll x){

for(int i=N-1;i>=0;i--)

if(x>>i){

if(b[i])x^=b[i];

else {b[i]=x;return;};

}

}

ll Max(){

ll ret=0;

for(int i=N-1;i>=0;i--)

if((ret^b[i])>ret)ret^=b[i];

return ret;

}

};

int main(){

LBase b;

int n;cin>>n;

while(n--){

ll x;cin>>x;

b.Insert(x);

}

cout<<b.Max();

return 0;

}

};