UIU_ Asteroids Template

```
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")

#define FOR(i,N) for(int i=0;i<N;i++)
#define ROF(i,N) for(int i=N-1;i>=0;i--)
#define ROF1(i,a,b) for(int i=(b);i>=(a);i--)
#define FOR1(i,a,b) for(int i=(a);i<=(b);i++)
#define FORi(i,a,b,c) for(int i=(a);i<=(b);i+=(c))

#define read freopen("IN.txt","r",stdin)
#define write freopen("OT.txt","w",stdout)
#define fastread {ios_base::sync_with_stdio(false);cin.tie(NULL);}
#define mem(a,b)          memset(a,b,sizeof(a))
#define EPS 1e-9
#define PI              acos(-1.0)
#define INF             2e16

#define ll              long long
#define ull             unsigned long long
#define ft              first
#define sd              second
#define bn              begin
#define en              end
#define pb              push_back
#define mp              make_pair
#define pii             pair<int,int>
#define pll             pair<LL,LL>
#define all(x) x.begin(), x.end()
#define gold ((1+sqrt(5))/2)
#define pi              acos(-1.0)
#define eps             1e-9
#define inf             2e18

#define dbg(args...) {cerr<<"--
>";debugger::call(#args,args);cerr<<"\n";}
struct debugger {
    static void call(const char* it) {}
    template<typename T, typename ... aT>
    static void call(const char* it, T a, aT... rest) {
        string b;
        for (; *it&&*it != ','; ++it) if (*it != ' ')b += *it;
        cerr << b << "=" << a << " ";
        call(++it, rest...);
    }
};

int main()
{
#ifdef UIU_Asteroids

    /// 1. Formulate Question
    /// 2. Gather Information
    /// 3. Apply The Information
    /// 4. Consider The Implication
    /// 5. Explore other points of view
    /// A. Always Think Easiest Way.
    /// B. Try To Guess Setter's View
    /// C. If got stuck or twisted skip and restart Later
    /// D. Use all known things and Find Out at least 2 or 3 way.
```

```
        double start_time = clock();

    #endif

    #ifndef ONLINE_JUDGE
        double end_time = clock();
        cout<<"Time : "<<(end_time - start_time)/ CLOCKS_PER_SEC;
    #endif

    return 0;
    }
    /****************** Functions ************************/

    /// scanf ("%[^\n]%*c", str); /// get string with space
    /// getline (cin, str); // gets string with space but has some issue
    with many test case

        /** convert string into number -->
             stringstream geek(token);
             int tm = -1;
             geek >> tm;
        **/
        /** takes string input except delimiter

        string delimiter1 = "{";
        size_t pos = 0;
        string token;
        while ((pos = str.find(delimiter1))!= string::npos)
            {
             token = str.substr(0, pos);
             str.erase(0,pos + 1);
             stringstream geek(token);
             int tm = -1;
             geek >> tm;
            if(tm!=-1){cout<<tm<<" ";}
            }
        **/

  string DeciToBin(LL q){

   /**ALGO:7=floor(log2(236)),Generally, p=int(logBV).

    (7) 236>128==1, number=236-128=108

    (6) 108>64==1, number=108-64=44

    (5) 44>32==1, number=44-32=12

    (4) 12>16==0

    (3) 12>8==1, number=12-8=4

    (2) 4>=4==1, number=4-4=0 number==0 ends calculations

    (1) ==0

    (0) ==0 So 236 is 11101100 in binary.**/

    string res="";LL p=floor(log2(q));

   while(p>=0){LL x=pow(2,p);if(q>=x){res+=to_string(1);q-=x;}

   else res+=to_string(0);p--;}return res;}

void bin(unsigned n){if (n > 1)bin(n>>1);printf("%d", n & 1);}
```

```
LL BinToDeci(string ans){LL dec = bitset<64>(ans).to_ullong();return dec}

LL BinToDeciA(LL ADD,LL sz){LL j=sz-1,ans=0;

for(LL i=0;i<sz;i++){ans+=(ADD[i]*pow(2,j));j--;}return ans;}

LL BinSum(LL ADD[],LL p,LL q){LL in=0,sz=0;

while(1){if(p==0){ADD[in]=p;reverse(ADD,ADD+sz);break;}ADD[in]=p%2;p=p/2;in++
;sz++;}return ADD;}

ULL getBITS(ULL q){ULL cnt=0,leng=1;

    while(q>0){cnt++;q=q>>1;}return cnt;}

int NODD(LL n)
{
    int cnt=0;
   for (int i = 1; i <= sqrt(n); i++)
    {
        if (n%i == 0)
        {
            if(n/i == i) cnt++;
            else cnt = cnt + 2;
        }
    }
    return cnt;
}
```

- **Saad Ibrahims BIG_INT:**

```
#include <iostream>
#include <sstream>
#include <string>
#include <algorithm>
using namespace std;

#define all(v)                ((v).begin()), ((v).end())
#define sz(v)                 ((int)((v).size()))
#define clr(v, d)       memset(v, d, sizeof(v))

const int MAX = 300;
typedef int big[MAX];

void big_zero (big num)             // b = 0
{
      int i;
      for (i=0 ; i<MAX ; i++)
            num[i] = 0;
}
void big_copy(big to, big from)     // to <-- from
{
      int i;
      for(i=0;i<MAX;i++)
            to[i] = from[i];
}
int big_len(big num)
{
      int j;
      for (j=MAX-1; j && !num[j]; j--);
      return j;
}
void big_print(big num)
{
      int i;
      for (i=big_len(num); i>=0; i--)
```

```
	{
		cout<<num[i];
		if (i && !(i % 3))
			cout<<',';
	}
	cout<<"\n";
}
void big_from_str(big num, string str)    // num = reverse(str)
{
	int i, len = str.size()-1;
	big_zero(num);

	for(i=len;i>-1;i--)num[len-i] = str[i]-'0';}//153-->153  00153-->153
string bit_to_str(big num)
{
	ostringstream oss;
	for (int i=big_len(num); i>=0; i--)
		oss<<num[i];
	return oss.str();
}
void big_from_int(big num, int x)
{
	int len = 0;
	big_zero(num);

	while(x)
	{
		num[len++] = x%10;
		x /= 10;
	}
}
int big_cmp (big a, big b)          // a <=> b return 1, 0, -1
{
	int i;
	for (i=MAX-1 ; i>=0 ; i--)
	{
		if (a[i] > b[i])
			return 1;
		else if (a[i] < b[i])
			return -1;
	}
	return 0;
}
void big_add(big ac, int n)   //add n over a
{
	int i, num, carry = n;
	for(i=0;i<MAX && carry;i++)
	{
		num  = ac[i] + carry;
		ac[i] = num%10;
		carry = num/10;
	}
}
void big_add (big a, big b, big result)   // result = a + b
{
	int i, carry=0;
	for (i=0 ; i<MAX ; i++)
	{
		result[i] = carry + a[i] + b[i];
		carry = result[i] / 10;
		result[i] %= 10;
	}
}
char big_sub (big a, big b, big result, char sign='+')
```

```
{
	if(big_cmp(a, b) < 0)
		return big_sub(b, a, result, '-');

	big_zero(result);
	int len_a = big_len(a);
	int len_b = big_len(b);

	int i, max = len_a > len_b ? len_a : len_b;

	for(i=0;i<=max;i++)
		result[i] = a[i] - b[i];

	for(i=0;i<=max;i++)
	{
		while(result[i] < 0)
		{
			result[i]   += 10;
			result[i+1] -= 1;
		}
	}
	return sign;
}
void big_product(big result, int factor)
{
    int carry = 0;
    for (int i=0; i<MAX; i++)
    {
        carry += result[i] * factor;
        result[i] = carry % 10;
        carry /= 10;
    }
}
//Work with leading zeros
void big_product(big a, big b, big result)
{
	int len_a = big_len(a);
	int len_b = big_len(b);

	if(len_a < len_b)
	{
		big_product(b, a, result);
		return;
	}

	big_zero(result);
	int i, j;

	int cursor = 0, pos = 0;

	for(i=0; i<=len_b; pos = 0, cursor++, i++)
	{
		for(j=0; j<=len_a; j++, pos++)
		{
			result[pos+cursor] += b[i]*a[j];

			if(result[pos+cursor] >= 10)
			{
				result[pos+cursor+1] += result[pos+cursor]/10;
				result[pos+cursor]   %= 10;
			}
		}
	}
}
```

```
int big_divide(big ac, int div)
{
    int i = big_len(ac);
    int reminder = 0;

    while (i >= 0)
    {
        reminder = reminder * 10;
            reminder = reminder + ac[i];
        ac[i] = 0;

        if (div <= reminder){
            ac[i] = reminder / div;reminder %= div;}i--;}return reminder;}
void big_divide(big ac, big div, big reminder)
{
    big cpy;
      big_zero(reminder);

      int i = big_len(ac);

    while (i >= 0)
    {
            big_product(reminder, 10);
        big_add(reminder, ac[i]);
        ac[i] = 0;

        while( big_cmp(reminder, div) >=0 ) //Insted of /, % is >= right or
only >
            {
                  big_sub(reminder, div, cpy);  //cpy = reminder-div
                  big_copy(reminder, cpy);            //reminder = cpy
                  ac[i]++;
            }
            i--;
    }
    big_print(reminder);
      big_print(ac);
}

void big_mod(big a, big div, big reminder)
{
      big cpy;
      big_zero(reminder);

      int i = big_len(a);

    while (i >= 0)
    {
            big_product(reminder, 10);
        big_add(reminder, a[i]);

        while( big_cmp(reminder, div) >=0 ) //Insted of /, %
            {
                  big_sub(reminder, div, cpy);  //cpy = reminder-div
                  big_copy(reminder, cpy);            //reminder = cpy
            }
            i--;
    }
}
void big_gcd(big ac, big bc)  //ac: means big a will be changed
{
      big t;

      while(!(bc[0] == 0 && big_len(bc) == 0))
```

```
	{
		big_mod(ac, bc, t);	//t = a % b
		big_copy(ac, bc);	//a = b
		big_copy(bc, t);	//b = t
	}	//a now represent the gcd
}
void factorial(int n, big fact)	/* Fast untill !1200 */
{
	int i, j, len;

	big_zero(fact);
	fact[0] = 1, len = 1;

	for(i=2; i<=n ; i++)
    {
		for(j=0; j<len; j++)
			fact[j] *= i;

		for(j=0; j<len; j++)
		{
			if(fact[j]>=10)
				fact[j+1] += fact[j]/10, fact[j] %= 10;
		}

		while(fact[len])
			if(fact[len++]>=10)
				fact[len] += fact[len-1]/10, fact[len-1] %= 10;
    }
}
big remain, odd, answer, temp;	//Set them out side to save memory
void big_squareRoot(string s) //Using Pell's equation // TLE
{
	long int group, count;

	big_zero(remain);
	big_zero(odd);
	big_zero(answer);

	if(s.size()%2 ==1)
		s = "0" + s;

	for(int i=0;i<s.size();i+=2)
	{
		group = (s[i]-'0') * 10 + s[i+1]-'0';

		big_copy(odd, answer);
		big_product(odd, 20);
		big_add(odd, 1);

		big_product(remain, 100);
		big_add(remain, group);

		count = 0;
		while( big_cmp(remain, odd) >= 0)
		{
			count++;
			big_sub(remain, odd, temp);
			big_copy(remain, temp);
			big_add(odd, 2);
		}
		big_product(answer, 10);
		big_add(answer, count);
	}
	big_print(answer);	//Contain only the int value //Give TLE
```

```
}

string big_to_str(big num) {
	ostringstream oss;
	int i;
	for (i = big_len(num); i >= 0; i--) {
		oss << num[i];
	}
	return oss.str();
}
string toBinay(string str) {
	big a;
	big_from_str(a, str);

	big zero;
	big_from_int(zero, 0);

	big two;
	big_from_int(two, 2);

	big mod;
	string ret = "";

	while (big_cmp(a, zero) != 0) {
		big_mod(a, two, mod);
		ret += big_to_str(mod);
		big_divide(a, 2);
	}

	reverse(all(ret));
	return ret;
}

string toDecimal(string binary) {
	big twoPow;
	big_from_int(twoPow, 1);

	big a;
	big_zero(a);

	big temp;
	for (int i = sz(binary) - 1; i >= 0; --i) {
		if (binary[i] == '1') {
			big_add(a, twoPow, temp);
			big_copy(a, temp);
		}

		big_product(twoPow, 2);
	}
	return big_to_str(a);
}

string fibonaccie(int n)
{
	int fib[3][MAX];

	big_from_int(fib[0], 1);
	big_from_int(fib[1], 1);
	big_from_int(fib[2], 1);

	for(int i=2;i<n;i++)
	{
		big_add(fib[0], fib[1], fib[2]);
		big_copy(fib[0], fib[1]);
```

```
            big_copy(fib[1], fib[2]);
      }
      return bit_to_str(fib[2]);
}

int main()
{
      string num1, num2;
      big a, b, c;

      while(cin>>num1>>num2)
      {
            big_from_str(a, num1);
            big_from_str(b, num2);

            big_product(a, b, c);
            big_print(c);
      }
      return 0;
}
```

❖ **Merge Sort Tree**

Solved:

1.SPOJ MKTHNUM (Fining Kth Smallest Element)

```
const int N = 100010;
vector<int>tree[5*N];
vector< pii > temp;

void build( int node , int s , int e ) /// O(NlogN)
{
     if( s==e  )
     {
           tree[node].pb(temp[s].sd);
            return ;
     }
     int left = node<<1, right = left|1, mid = (s+e)>>1;
     //  int left = node*2, right = (node*2)+1, mid = (s+e)/2;
     build(left , s , mid );
     build(right , mid+1 , e );
     merge( all(tree[left]), all(tree[right]) ,back_inserter(tree[node]));
}
int query(int node, int s, int e, int l, int r, int k) { /// O(Nlog^2(N))

      if(s==e )return tree[node][0];

      int left = node*2, right = (node*2)+1, mid = (s+e)/2;
      int leftR = lower_bound(all(tree[left]),l)-tree[left].bn();
      int rightR = upper_bound(all(tree[left]),r)-tree[left].bn();
      int smallestR = rightR-leftR;
      if(smallestR>=k)
            return query(left, s, mid, l, r, k);
      else
            return query(right, mid+1, e, l, r, (k-smallestR));

}
int query(int node, int s,int e,int l,int r,int k)  /// O(log^2(N))
{
    if(r<S || l>e)return 0;
    if(s>=l && e<=r)
    {
       return lower_bound(all(tree[node]),k)- tree[node].bn();// {elements
are less than k}
    }
    int left = node*2, right = (node*2)+1, mid = (s+e)/2;
```

```
    return query(left, s, mid, l, r, k) + query(right, mid+1, e, l, r, k);

}
```

❖ **Segment Tree**

```
void build(ll node, ll L, ll R)
{
    tree[node]=0;
    if(L == R)
    {
        tree[node]= arr[L];
        return;
    }
    ll mid = (L+R)>>1;
    ll Left = node<<1, Right = Left|1;
    build(Left, L, mid);
    build(Right, mid + 1, R);
    tree[node]= tree[Left] + tree[Right];
}
void PU(ll node, ll L, ll R, ll pos, ll val)
{
    if(L == R)
    {
        tree[node] += val;
        return;
    }
    ll mid = (L+R)>>1;
    ll Left = node<<1, Right = Left|1;
    if(pos <= mid) PU(Left, L, mid, pos, val);
    else PU(Right, mid + 1, R, pos, val);

    tree[node] = tree[Left] + tree[Right];
}
void RU(ll node, ll S, ll E, ll l, ll r , ll val)
{

    if(lazy[node] != 0) /// PROPAGATION
    {
        tree[node] += (E - S + 1) * lazy[node];

        if(S != E)
        {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }

   if(S > E || S > r || E < l)return;


    if(S>=l && E<=r)
    {
        tree[node] += (E - S + 1) * val;

         if(S != E)
        {
            lazy[node*2] += val;
            lazy[node*2+1] += val;
        }
        return ;
    }
    ll mid = (S+E)>>1;
```

```
    RU(node*2,S,mid,l,r,val);
    RU(node*2+1,mid+1,E,l,r,val);
    tree[node] = tree[node*2] + tree[node*2+1];
}
ll RQ(ll node, ll S, ll E, ll l, ll r)
{
    if(S > E || S > r || E < l)return 0;
    ll mid = (S+E)>>1;
    ll Left = node<<1, Right = Left|1;
     if(lazy[node] != 0)   /// PROPAGATION
    {
        tree[node] += (E - S + 1) * lazy[node];
        if(S != E)
        {
            lazy[Left] += lazy[node];
            lazy[Right] += lazy[node];
        }
        lazy[node] = 0;
    }

    if(S >= l && E <= r) return tree[node];

    ll x = RQ(Left, S, mid, l, r);
    ll y = RQ(Right, mid + 1, E, l, r);
    return x + y;
}
```

- **Treap**

```
struct item {
    int prior, value, cnt,res;
    bool rev;
    item *l, *r;
    item(){}
      item(int value) {
            cnt = 1;
            this->value = value;
            res = value;
            prior = rand();
            l = r = NULL;
            rev = false;
      }
}*t;

typedef struct item * pitem;

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}
void reset(pitem t){
   if(t) t->res = t->value;//lazy already propagated
}
void combine(pitem& t,pitem l,pitem r){//combine segtree ranges
    if(!l || !r)return void(t = l?l:r);
    t->res = max(l->res, r->res) ;
}
void operation(pitem t){//operation of segtree
```

```
    if(!t)return;
    reset(t);//node represents single element of array
    combine(t,t->l,t);combine(t,t,t->r);
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}
void merge (pitem & t, pitem l, pitem r) {

    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
    operation(t);
}
void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    int cur_key = add + cnt(t->l);
    if (key < cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
    operation(t);
}
void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}
void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}
int main()
{
            if(ch=='A')
            {
                pitem T;
                T = new item(x);
                pitem l,r,m;
                if(ok)t=T;
                else
                {
                    split(t, l, r, y-1);
                    merge(m, l, T);
```

```
                merge(t, m, r);
            }
            ok=0;
        }
        else
        {
            pitem L,mid,R;
            split(t,L,mid,x-1);
            split(mid,t,R,y-x+1);
            int ans = t->res;
            merge(mid,L,t);
            merge(t,mid,R);
            printf("%d\n",ans);
        }
}
```

Manacher's Algorithm - Finding all sub-palindromes in O(N)

The implementation of the trivial algorithm is:

```
vector<int> d1(n),  d2(n);
for (int i = 0; i < n; i++) {
    d1[i] = 1;
    while (0 <= i - d1[i] && i + d1[i] < n && s[i - d1[i]] == s[i
+ d1[i]]) {
        d1[i]++;
    }

    d2[i] = 0;
    while (0 <= i - d2[i] - 1 && i + d2[i] < n && s[i - d2[i] -
1] == s[i + d2[i]]) {
        d2[i]++;
    }
}
```

For calculating d1[]d1[], we get the following code:

```
vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
        k++;
    }
    d1[i] = k--;
    if (i + k > r) {
        l = i - k;
        r = i + k;
    }
}
```

For calculating $d2[]$d2[], the code looks similar, but with minor changes in arithmetical expressions:

```
vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i +
k]) {
        k++;
    }
    d2[i] = k--;
    if (i + k > r) {
        l = i - k - 1;
        r = i + k ;
    }
}
```

Binary Exponentiation $O(\log n)$

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Hashing :

```
long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
int count_unique_substrings(string const& s) {
    int n = s.size();
    const int p = 31;
    const int m = 1e9 + 9;
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
```

```
    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;

    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * p_pow[n-i-1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```

Search for duplicate strings in an array of strings:

```
vector<vector<int>> group_identical_strings(vector<string> const&
s) {
    int n = s.size();
    vector<pair<long long, int>> hashes(n);
    for (int i = 0; i < n; i++)
        hashes[i] = {compute_hash(s[i]), i};

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

    vector<vector<int>> groups;
    for (int i = 0; i < n; i++) {
        if (i == 0 || hashes[i].first != hashes[i-1].first)
            groups.emplace_back();
        groups.back().push_back(hashes[i].second);
    }
    return groups;
}
```

Divisibility Rules :

A positive integer N is divisible by

- **2** if the last digit of N is 2, 4, 6, 8, or 0;
- **3** if the sum of digits of N is a multiple of 3;
- **4** if the last 2 digits of N are a multiple of 4;
- **5** if the last digit of N is either 0 or 5;
- **6** if N is divisible by both 2 and 3;
- **7** if subtracting twice the last digit of N from the remaining digits gives a multiple of 7 (e.g. 658 is divisible by 7 because 65 - 2 x 8 = 49, which is a multiple of 7);
- **8** if the last 3 digits of N are a multiple of 8;
- **9** if the sum of digits of N is a multiple of 9;
- **10** if the last digit of N is 0;
- **11** if the difference of the alternating sum of digits of N is a multiple of 11 (e.g. 2343 is divisible by 11 because 2 - 3 + 4 - 3 = 0, which is a multiple of 11);
- **12** if N is divisible by both 3 and 4.

- **13** if 4 times the units digit of N plus the number obtained by removing the units digit of N is a multiple of 13;
- **17** if the units digit subtracted 5 times from the remaining number (excluding the units digit) results in a number that is divisible by 17;
- **19** if doubling the units digit and adding it to the number formed by removing the units digit in the original number is divisible by 19.

23 Add 7 times the last digit to the remaining truncated number. Repeat the step if necessary. If the result is divisible by 23, the original number is also divisible by 23.

- Check for 53935: $5393 + 7 \times 5 = 5428 \implies 542 + 7 \times 8 = 598 \implies 59 + 7 \times 8 = 115$, which is 5 times 23. Hence 53935 is divisible by 23.

29 Add 3 times the last digit to the remaining truncated number. Repeat the step if necessary. If the result is divisible by 29, the original number is also divisible by 29.

- Check for 12528: $1252 + 3 \times 8 = 1276 \implies 127 + 3 \times 6 = 145 \implies 14 + 3 \times 5 = 29$, which is divisible by 29. So 12528 is divisible by 29.

31 Subtract 3 times the last digit from the remaining truncated number. Repeat the step if necessary. If the result is divisible by 31, the original number is also divisible by 31.

- Check for 49507: $4950 - 3 \times 7 = 4929 \implies 492 - 3 \times 9 = 465 \implies 46 - 3 \times 5 = 31$. Hence 49507 is divisible by 31.

37 Subtract 11 times the last digit from the remaining truncated number. Repeat the step if necessary. If the result is divisible by 37, the original number is also divisible by 37.

- Check for 11026: We have $1102 - 11 \times 6 = 1036$. Since $103 - 11 \times 6 = 37$ is divisible by 37, 11026 is divisible by 37.

41 Subtract 4 times the last digit from the remaining truncated number. Repeat the step if necessary. If the result is divisible by 41, the original number is also divisible by 41.

- Check for 14145: We have $1414 - 4 \times 5 = 1394$. Since $139 - 4 \times 4 = 123$ is divisible by 41, 14145 is divisible by 41.

43 Add 13 times the last digit to the remaining truncated number. Repeat the step if necessary. If the result is divisible by 43, the original number is also divisible by 43. (This process becomes difficult for most of the people because of multiplication with 13.)

- Check for 11739: We have $1173 + 13 \times 9 = 1290$. Since 129 is divisible by 43, the 0 can be ignored. So, 11739 is divisible by 43.

47 Subtract 14 times the last digit from the remaining truncated number. Repeat the step if necessary. If the result is divisible by 47, the original number is also divisible by 47. (This too is difficult to operate for people who are not comfortable with table of 14.)

- Check for 45026: We have $4502 - 14 \times 6 = 4418$. Since $441 - 14 \times 8 = 329$, which is 7 times 47, 45026 is divisible by 47.

- **GCD:**

- **Naive** - *O(min(A, B)).*

```
int GCD(int a, int b)
{
        for (int i=min(a,b); i>=1; i--)
        if (a%i==0 && b%i==0)
        return i;
}
```

- **Euclid's algorithm** - *O(log min(a,b))*

```
int GCD(int a, int b)
{
        if (b==0) return a;
        return GCD(b,a%b);
}
```

Euclid's Algorithm is GCD(A, B) = GCD(B, A % B).Algorithm will recurse until A % B = 0.
GCD(16, 10) = GCD(10, 16 % 10) = GCD(10, 6)
GCD(10, 6) = GCD(6, 10 % 6) = GCD(6, 4)
GCD(6, 4) = GCD(4, 6 % 4) = GCD(4, 2)
GCD(4, 2) = GCD(2, 4 % 2) = GCD(2, 0)
Since B = 0 so GCD(2, 0) will return 2.

- **Extended Euclid Algorithm -** *O(log min(a,b))*

```
int d, x, y;
void extendedEuclid(int A, int B) {
   if(B == 0) {
      d = A;
      x = 1;
      y = 0;
   }
   else {
      extendedEuclid(B, A%B);
      int temp = x;
      x = y;
      y = temp - (A/B)*y;
   }
}

int main( ) {
extendedEuclid(16, 10);
cout << "The GCD of 16 and 10 is " << d << endl;
cout << "Coefficient x and y are: "<< x <<  "and  " << y << endl;
return 0;   }
```

This algorithm has a special property that it can always be represented in the form of an equation, i.e., Ax + By = GCD(A, B).This algorithm gives us the coefficients (x and y) of this equation which will be later useful in finding the ***Modular Multiplicative Inverse****.*

- *When is this algorithm used?*

This algorithm is used when A and B are co-prime. In such cases, x becomes the multiplicative modulo inverse of A under modulo B, and y becomes the multiplicative modulo inverse of B under modulo A. This has been explained in detail in the ***Modular multiplicative inverse*** section.

- **LCM:**

$$\mathrm{lcm}(a,b) = \frac{a \cdot b}{\gcd(a,b)}$$

```
int lcm (int a, int b) {
    return a / gcd(a, b) * b;
}
```

- **Number of divisors :**

  Equation for determining the number of divisors, or factors, in a number.
  The equation is : **d(n)=(a+1)(b+1)(c+1) ;**
  where **d(n)** is equal to the number of divisors in the number **n**, and **a**, **b**, and **c** are the exponents in the prime factorization equation for the number.

- Scan string with space -
  - scanf ("%[^\n]%*c", str);
  - getchar();
    getline (cin, str);

- **STL Set index Access : Slow - O(n)**

  - **Set pair:**

```
set<pii>set;
auto it=set.begin() ;
advance(it,5);
cout<<(it->sd)<<"\n";
```

  - **Set :**

```
set<int>set;
auto it=set.begin() ;
advance(it,5);
cout<<(*it)<<"\n";
```