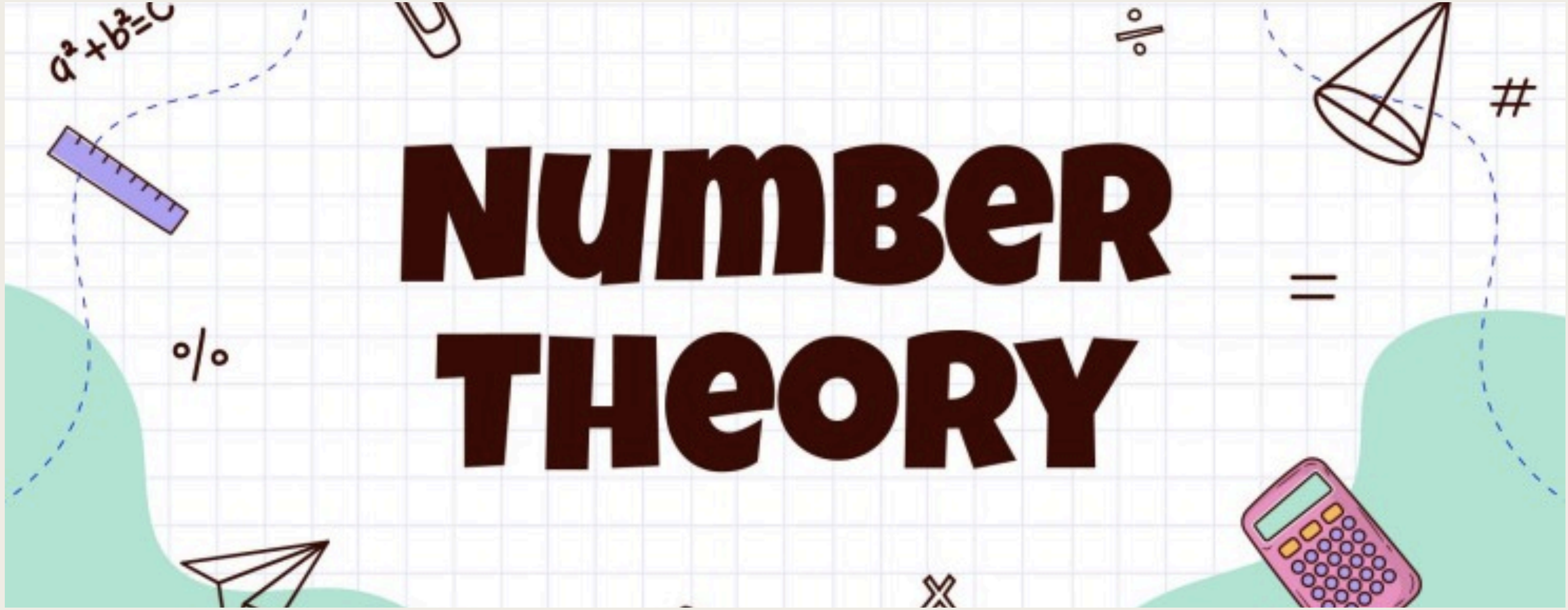

Important Question

Ajay Kumar

Codeforces

If M is 10^9+7

It serves mainly following purposes :-

1. Fits in the range of int in languages like C/C++.
2. One doesn't need to deal with overflows when doing addition/subtraction modulo M.
3. **M is a prime number.**
4. You can perform division modulo M just by calculating the **modular inverse of the divisor.**
5. Its square fits in a 64 bit integer.(This is important so that the formula **(a*b)%M=((a%M)*(b%M))%M** can be applied).
6. This allows you to apply Fermat's little theorem
7. It is easy to remember

Binary exponentiation

```
int binexp(int x, int y){
    int z = 1;
    while(y)
    {
        if(y & 1) z = (z*x)%mod;
        x = (x*x)%mod;
        y >>= 1;
    }
    return z;
}
```

```
int spf[5000001];
int pre[5000001];
// Calculating SPF (Smallest Prime Factor) for every
// number till MAXN.
// Time Complexity : O(nloglogn)

void sieve(){
   for(int i=0;i⩽sizee;i++)
      spf[i]=i;

   for(int i=2;i*i⩽sizee;i++){
      if(spf[i]==i){
         for(int j=i*i;j⩽sizee;j+=i){
            if(spf[j]==j)
               spf[j]=i;
         }
      }
   }
}
```

C. Koxia and Number Theory

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Joi has an array a of n **positive** integers. Koxia wants you to determine whether there exists a **positive** integer $x > 0$ such that $\gcd(a_i + x, a_j + x) = 1$ for all $1 \le i < j \le n$.

Here $\gcd(y, z)$ denotes the greatest common divisor (GCD) of integers y and z.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases. The description of test cases follows.

The first line of each test case contains an integer n ($2 \le n \le 100$) — the size of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^{18}$).

It is guaranteed that the sum of n over all test cases does not exceed 1000.

Output

For each test case, output "YES" (without quotes) if there exists a positive integer x such that $\gcd(a_i + x, a_j + x) = 1$ for all $1 \le i < j \le n$, and "NO" (without quotes) otherwise.

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

- For any i and j, if $a_i \equiv a_j (mod\ m)$ for some m>1, then x must meet the condition:

$x + a_i \not\equiv 0\ (mod\ m)$. Otherwise, the $gcd(a_i + x, a_j + x)$ will be a multiplication of m.

Generally, for any prime $p, x \not\equiv\ a_i\ (mod\ p)$ for any i and j such that $a_i\ \equiv a_j (mod\ p)$

Also , if there is two or more a_i such that $a_i\ \equiv r\ (mod\ p)$ for every $0 \leq r < p$,we can't find x since there will be a pair of a_i and a_j which are multiplication of p for any x.

And otherwise x always exists(can be shown by chinese remainder theoram)

So the solution is to check whether r exists such that there are at most one a_i which meets

$a_i\ \equiv r\ (mod\ p)$for any p,

Here we don't check for p> n since r will always exists such that no $a_i\ \equiv r\ (mod\ p)$

Example 3 6 4 7 8 11 [0 0 1 1 2 2] not possible

if p=5 [0-3 1-4 2-4 3-0 4-5]

$0 + x \equiv 3\ mod(5) => x \equiv 3\ mod(5)$

$1 + x \equiv 3\ mode(5) => x \equiv 2\ mod(5)$

$2 + x \equiv 3\ mod(5) => x \equiv 1\ mod(5)$

$4 + x \equiv 3\ mod(5) => x \equiv 4\ mod(5)$

- This idea can be extended to larger primes. For a given prime p, let cnt_j be the multiplicity of j in the multiset $[a_i \ mod \ p, a_2 mod \ p, a_3 \ mod \ p, \ldots a_n \ mod \ p \ ]$
- if $\min(cnt_0, cnt_1, cnt_2, \ldots cnt_{p-1}) \geq 2$, we output NO immediately.
- While there are many prime up to 10^{18},we only need to check for the primes upto n/2. This is because $\min(cnt_0, cnt_1, cnt_2, \ldots cnt_{p-1}) \geq 2$ is impossible for greater primes. According to Pigeonhole Principle .Since the number of primes up to is (n/2) is at most $o(\frac{n}{logn})$ .This problem can be solved in time $O(n^2/logn)$

Pigeon Hole Principle

Pigeonhole principle strong form –

Theorem: $q_1, q_2, q_3, q_4, q_5 \ldots q_n$ be positive integers.

If $q_1 + q_2 + q_3 + \ldots + q_n - n + 1$ objects are put into n boxes, then either the 1st box contains at least $q_1$objects, or the 2nd box contains at least q_2 objects, . . ., the nth box contains at least q_n objects.

Example – 1:

In a computer science department, a student club can be formed with either 10 members from first year or 8 members from second year or 6 from third year or 4 from final year. What is the minimum no. of students we have to choose randomly from department to ensure that a student club is formed?

Solution: we can directly apply from the above formula where,

q1 =10, q2 =8, q3 =6, q4 =4 and n=4

Therefore the minimum number of students required to ensure department club to be formed is

10 + 8 + 6 + 4 – 4 + 1 = 25

Introduction to Chinese remainder theoram

We are given two arrays num[0..k-1] and rem[0..k-1].
In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],
x % num[1] = rem[1],
……..
x % num[k-1] = rem[k-1]

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}
Output: 11
Explanation:
11 is the smallest number such that:
(1) When we divide it by 3, we get remainder 2.
(2) When we divide it by 4, we get remainder 3.
(3) When we divide it by 5, we get remainder 1.

Modulo inverse calculator $A\ X \cong 1 \pmod{M}$

Brute force Approch

```
int  modInverse(int  A, int  M){
    for (int   X = 1; X < M; X++)
        if (((A % M) * (X % M)) % M == 1)
            return X;
}
```

Time Complexity: O(M)

Here we can see the brute force taking from 1 to M so How we can optimise

1) We have to study the Euclid algorithms
2) Extended Euclid Algorithms

Modular multiplicative inverse when M and A are coprime or gcd(A, M)=1:

*The idea is to use **Extended Euclidean algorithms** that take two integers 'a' and 'b', then find their **gcd,** and also find **'x'** and **'y'** such that*

ax + by = gcd(a, b)

To find the multiplicative inverse of 'A' under 'M', we put b = M in the above formula. Since we know that A and M are relatively prime, we can put the value of gcd as 1.

Ax + My = 1

If we take modulo M on both sides, we get

Ax + My ≅ 1 (mod M)

We can remove the second term on left side as 'My (mod M)' would always be 0 for an integer y.

Ax ≅ 1 (mod M)

So the 'x' that we can find using Extended Euclid Algorithm is the multiplicative inverse of 'A'

- You can given an array A. For each K from 1 to N. compute the maximum GCD of a subarray of length K.

Explanantion:

- There are $O(N^2)$ subarray, and while going through them all is impossible. It's is actually possible to compute all their GCD's in a compressed form.
- The Main observation that allows this to happen is the fact that there are only $O(N \log \ max\ A)$ distinct subarray GCDs.
- For convenience, let $f(i,j)$= $gcd(A_i, A_{i+1}, \ldots, A_j)$.
- Lest fix and index R (1≤ R≤ N). and look at all subarrays of the form [i,R] for 1≤i≤R.
- Suppose you know $f(i,R)$. What's it's relation with $f(i-1,R)$
- we know that $f(i-1,R)$= $gcd(f(i,R), A_{i-1})$
- In particulat, $f(i-1,R)$ will always be a divisor of $f(i,R)$.
- So we either have f(i-1,R) = f(i,R) (which does'nt increase the number of distinct gcd's, or f(i-1,R) is strictly smaller factor of f(i,R), in which case $f(i-1,R) \leq f(i,R)/2$

- This halving can only happen $logA_R$ times before the GCD reaches and never changes again.
- So there are $logA_R$ distince values of f(i,R).
- Summing this across all R gives us an upper bound of ($N \log \max\ A$) distinct subarray GCDs
- Computing them all isn't too hard, in fact the proof above gives us a pretty reasonable ways to compute them all quickly

How ?

- Notice that the GCD's ending at a given index form continous segments, so it's enough to find the endpoints of these segments: that give us infomations about every subarray.
- Let's define $dp(i, x)$ be the length of the longest subarray ending at i with gcd x.
- We only need to care about those pairs(i,x) for which this value is non-zero, and the earlier discussion tells us that there are $\leq NlogmaxA$ such states.

- computing all this values isn’t too hard either : notice that a subarray ending at i can be obtained i-1, so:

$$dp(i,x) = 1 + \underbrace{max}_{\forall\ y \in gcd(A_i,y) = x} (dp(i-1), y)$$

- There are only O(log) non-zero value of y, so simply store them all and iterate across them,each time taking its GCD with x and updating the correct dp value.
- At any rate, now that we know all subarray GCDs, the problem is almost solved.

Contd..

- Let ans_i be the answer of the subarray of length i.
- Then, using what we computed earlier:
- Let L be the length of longest subarray ending at i with GCD g.
- The set $ans_L = max(ans_L, g)$
- Finally,set $ans_i = max(ans_i, ans_{i+1}, \ldots, ans_N)$; which can be done in O(N) by taking suffix maximums of the ans array.
- As an aside, we make $O(N \log maxA)$ GCD class, each of which is technically O(log max A)
- This gives us an upper bound of $O(Nlog^2(max\ A))$for the time complexity.

Cont…….

Code

```
int a[n];
     ...
    map<int, int> sub_gcd[n]; //declare an array of n maps
   /*
     Key is gcd,Value is the largest length such that gcd(a[i - len], ... , a[i]) equals to key.*/

    sub_gcd[0][a[0]] = 0;
    for(int i = 1; i < n; i++)
    {
        sub_gcd[i][a[i]] = 0;
        for(auto it: sub_gcd[i - 1])
        {
            int new_gcd = __gcd(it.first, a[i]);
            sub_gcd[i][new_gcd] = max(sub_gcd[i][new_gcd], it.second + 1);
        }
    }
```

Inclusion exclusion principle

Let's see an example problem Co-prime where this principle could be applied:

Problem: Given N,L,R you need to compute the number to x in interval [L,R] such is x is coprime with N, that is gcd(x,N)= 1,

There are 1≤ T≤ 100 testcases constraints: $1 \leq N \leq 10^9, 1 \leq L \leq R \leq 10^{15}$

Solution:

- If we can compute the number of integer x in the interval [1,X] such that x is coprime with N,denoted as f(x) then the answer is $f(R) - f(L-1)$.
- How we gonna compute f(x) instead of counting the number that are coprime with N,we can compare the numbers that aren't coprime with N, that is sharing at least one prime factor with. To do this, we can first sieve all primes not exceeding $O(\sqrt{n})$ then
- find all prime factors $p_1, p_2 \ldots . p_k$ of N.
- Let A_i be the set of numbers that are divisible by p_i then the answer is the $\cup_{i=1}^{n} A_i$,which may be hard to the compute directly .
- How ever using the restricted inclusion-exclusion principle .we can convert the problem into computing the size of intersection of sets. which is trivial .
- Time complexity is $O(\sqrt{(n)} + T2^K)$ k<10 equals to the number of distinct prime
- we are generating all the subset's of the using the concept of bitmask and finding the parity of the subset if the parity of the subset is 1 then we have to subtract the cnt from it otherwise adding into the the cnt.

```
vector<T> factor(T n) {
    n = abs(n);
    vector<T> res;
    for (T i = 2; i * i ⩽ n; i++) {
        if (n % i == 0) {
            res.emplace_back(i);
            while (n % i == 0) {
                n /= i;
            }
        }
    }
    if (n > 1) {
        res.emplace_back(n);
    }
    return res;
}
```

```
int fun(long long n, vector<long long> w) {
        long long res = 0;
        int sz = (int) w.size();
        for (int mask = 0; mask < (1 << sz); mask++) {
            long long c = 1;
            for (int i = 0; i < sz; i++) {
                if (mask & (1 << i)) {
                    c *= w[i];
                }
            }
            if (__builtin_parity(mask)) {
                res -= n / c;
            } else {
                res += n / c;
            }
        }
        return res;
    };
```

Another problem: This problem character encoding requires you to compute the number of solutions to the equation $x_1 + x_2 + x_3 + x_4 \ldots + x_n = k,$ satisfying that $0 \leq x_i < m$ modulo 998244353.
Constraints: $1 \leq n, m, k \leq 10^5, 1 \leq \sum n$, $\sum m, \sum k \leq 5 \times 10^6$

Hint The number of non-negative solutions to $x_1 + x_2 + x_3 + x_4 \ldots + x_n = k,$ is given by $\binom{n+k-1}{k}$

Solution:

The only thing we need to handle to get rid of that annoying constraints $x_i < m$.

To do that , we apply the inclusion-exclusion principle. Let $e_i = x_i \geq m$. Then $N = \phi$ is our desired answer .

Clearly this set of properties is homogeneous . Take $T = 1,2,\ldots . j(j \leq n)$. then $N \subseteq T$ is the number of solutions with

$$x_1 \geq m, x_2 \geq m \ldots, x_j \geq m \, .$$

Setting $y_i = x_i = m(i \leq j), y_i = x_i(i > j),$ and it's the same as the number of solutions of the system

$$y_1 + y_2 + \ldots + y_n = k - jm \, .$$

, thus is answer is therefore.

$$N_{=\phi} = N_{=0} = \sum_{j=0}^{n} (-1)^j \binom{n}{j} \binom{n + (k - jm) - 1}{n - 1}$$

The complexity of the solution is O(n), due to precomputing the factorials and the modular inverses of the factorial

- Firstly let's calculate the number of pairs (x,y) where 1≤ x≤ y≤ n with gcd(x,y)=k
- for each $k \in [1 \ldots n]$ in $O(nlog\ n)$ time . It can be solved in standard way using mobius function $\mu(x)$or using dyanamic programming.

- $dp[k] = \frac{1}{2} \lfloor \frac{n}{k} \rfloor \lfloor \frac{n}{k} - 1 \rfloor - dp[2k] - dp[3k] - dp[4k] \ldots dp[\lfloor \frac{n}{k} \rfloor * k]$

```
//to count the


vector<int>f(n+1,0),g(n+1,0);
for(i = n;i>0;i--){
    int cnt = n/i;
    g[i] = (cnt*(cnt-1))/2;
    f[i] = g[i];
    for(j = 2*i;j<=n;j+=i){
        f[i]-=f[j];
    }
}
```

To calculate the nCr mod m when m is not necessarily prime no

```
const int MAX=1002;
int dp[MAX][MAX]; // initialize dp with -1

int nCr(int n, int r, int m){
    if(r>n) return 0;
    if(r==0 || r==n){
        return 1;
    }
    if(dp[n][r]!=-1){
        return dp[n][r];
    }
    return  dp[n][r]=(nCr(n-1,r-1,m)+nCr(n-1,r,m))%m;
}
```

```
int fact[21];

void factPre(){
    fact[0]=1;
    for(int i=1;i<=20;i++){
        fact[i]=(fact[i-1]*i)%mod;
    }
}

int permutate(int n, int r){
    if(r>n) return 0;
    return (fact[n]*binexp(fact[n-r],mod-2))%mod;
}

int ways(int n, vector<int> &cnt){
    int tot=1;
    int done=0;
    for(int i=0;i<=n;i++){
        tot=(tot*permutate(i-done,cnt[i]))%mod;
        done+=cnt[i];
    }
    return tot;
}
```