# Some Useful Code:

```
Max priority_queue<ll>
Min priority_queue<ll,vector<ll>,greater<ll>>
#define pd(x,y)     fixed<<setprecision(y)<<x
#define dbg1(x)      cout<<"["<<#x<<":
"<<x<<"]"<<endl;
#define dbg2(x, y) cout<<"["<<#x<<":
"<<x<<"]"<<"  ["<<#y<<": "<<y<<"]"<<endl;
#define dbg3(x, y, z) cout<<"["<<#x<<":
"<<x<<"]"<<"  ["<<#y<<": "<<y<<"]"<<"
["<<#z<<": "<<z<<"]"<<endl;
#define dbg4(x, y, z, k) cout<<"["<<#x<<":
"<<x<<"]"<<"  ["<<#y<<": "<<y<<"]"<<"
["<<#z<<": "<<z<<"]"<<"  ["<<#k<<":
"<<k<<"]"<<endl;
#define endl            "\n"
#define FAST
ios_base::sync_with_stdio(0); cin.tie(0);
cout.tie(0);
#define pi         3.141592653
//  freopen("runway_input.txt", "r", stdin);
  //  freopen("output.txt", "w", stdout);
//cout<<"Case "<<++u<<": ";
```

# Number Theory

### Prime number under 100

```
// there are 25 numbers
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97
```

### Sieve:

```
const int maxN=1e7+7;
bool isPrime[maxN];
void sieve()
{
    isPrime[1] = true;
    for(int i=2; i<maxN; i++)
        isPrime[i]=true;

    for(int i=2; i*i<maxN; i++)
        if(isPrime[i])
            for(int j=i*i; j<maxN; j+=i)
                isPrime[j]=false;
}
```

### Sieve And Prime Factorization using Sieve O(logn) for multiple queries

```
const int N = 1e5 + 9;
int spf[N];
vector<int> primes;
void sieve() {
  for(int i = 2; i < N; i++) {
    if (spf[i] == 0) spf[i] = i,
primes.push_back(i);
    int sz = primes.size();
    for (int j = 0; j < sz && i * primes[j] <
N && primes[j] <= spf[i]; j++) {
      spf[i * primes[j]] = primes[j];
    }
  }
}

int main()
{
    int n=50;
  while(n>1)
  {
    cout<<spf[n]<<endl;
    n/=spf[n];
  }

  for(auto it: primes)
  cout<<it<<endl;
}
```

### Binary exponiential with MOD

```
long long binpow(long long a, long long b,
                   long long m) {
    a %= m;
    long long res = 1 % m;
    while (b > 0) {
        if (b & 1)
            res = (res * a) % m;
        a = (a * a) % m;
        b >>= 1;
    }
    return res;
}
```

### Binary exponiential

```
int binpow (int a, int n)
{
      int res = 1;
      while (n)
            if (n & 1)
            {
                  res *= a;
                  --n;
            }
            else
            {
                  a *= a;
                  n >>= 1;
            }
      return res;
}
```

### Greatest common divisor – GCD

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

### Least common multiple – LCM

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

### Leap year

```
bool isLeap(int n)
{
if (n%100==0)
if (n%400==0) return true;
else return false;
if (n%4==0) return true;
else return false;
}
```

### Prob: gcd(A^N+B^N,|A-B|)

Here |A-B| is small. So we can use the divisor of |A-B| and take Bin_exp with MOD (divisor) and if Bin_exp returns zero then it is a possible candidate for the ans.

### Modular Multiplication Inverse

```
long long inv_mod(long long a, long long m) {
    return binpow(a, m-2, m);
}
```

### Total Number of divisor:

If N=p^a*q^b*r^c . Then total number of divisors =**(a+1)*(b+1)*(c+1)**

### Sum of all divisors of a number(Sigma function):

N=P1^e1*P2*e2*P3*e3……Pk^ek
Sigma(n)**=** (1+p1+p1^2+p1^3+…p1^e1)*(1+p2+p2^2+p2^3+…p2^e2)* ………(1+pk+pk^2+pk^3+…pk^e1)
**=**((p1^(e1+1)-1)/(p1-1))*((p2^(e2+1)-1)/(p2-1))*……((pk^(ek+1)-1)/(pk-1))

```
int SOD( int n ) {
    int res = 1;
    int sqrtn = sqrt ( n );
    for ( int i = 0; i < prime.size() &&
prime[i] <= sqrtn; i++ ) {
        if ( n % prime[i] == 0 ) {
            int tempSum = 1; // Contains
value of (p^0+p^1+...p^a)
            int p = 1;
            while ( n % prime[i] == 0 ) {
                n /= prime[i];
                p *= prime[i];
                tempSum += p;
            }
            sqrtn = sqrt ( n );
            res *= tempSum;
        }
    }
    if ( n != 1 ) {
        res *= ( n + 1 ); // Need to multiply
(p^0+p^1)
    }
    return res;
}
```

**You have to use mod_inv to calculate division.**

```
/*cnt*=m;  cnt++;
ll x=(big_mod(prime[i],cnt,MOD)); x--;
if(x<0) x+=MOD;ans*=x;ans%=MOD;
x=inv_mod(prime[i]-1,MOD);
if(x<0) x+=MOD; ans*=x; ans%=MOD;*/
```

**//When we work with MOD we must check <0.**

### nCr % p using Fermat's little theorem

```
unsigned long long power(unsigned long long x,
                                  int y, int
p)
{
    unsigned long long res = 1; // Initialize
result

    x = x % p; // Update x if it is more than
or
    // equal to p

    while (y > 0)
    {

        // If y is odd, multiply x with
result
        if (y & 1)
            res = (res * x) % p;

        // y must be even now
        y = y >> 1; // y = y/2
        x = (x * x) % p;
    }
    return res;
}

// Returns n^(-1) mod p
unsigned long long modInverse(unsigned long
long n,
                                             i
nt p)
{
    return power(n, p - 2, p);
}

// Returns nCr % p using Fermat's little
// theorem.
unsigned long long nCrModPFermat(unsigned long
long n,
                                 int r, int p)
{
    // If n<r, then nCr should return 0
    if (n < r)
        return 0;
    // Base case
    if (r == 0)
        return 1;

    // Fill factorial array so that we
    // can find all factorial of r, n
    // and n-r
    unsigned long long fac[n + 1];
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = (fac[i - 1] * i) % p;

    return (fac[n] * modInverse(fac[r], p) %
p
                * modInverse(fac[n - r], p) % p)
               % p;
}
```

## Eular totient/Phi Function:

N=P1^x1*P2*x2*P3*x3……Pk^xk
Phi(N)=N*((P1-1)/P1)*((P2-1)/P2)*((P3-1)/P3)*((Pk-1)/Pk)

**You can implement this using Prime fact:**
x/=(prime[i]);x*=(prime[i]-1);

## Eular totient/Phi Extention:

Given a number $N$, let $d$ be a divisor of $N$. Then the number of pairs $a,N$, where $1≤a≤N$ and $gcd(a,N)=d$, is $\phi(N/d)$.

## Eular totient/Phi Function 1 to n (O(nlog(logn))

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; 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;
        }
    }
}
```

## Modular Inverse from 1 to N (O(N)):

```
int inv[SIZE];
inv[1] = 1;
for ( int i = 2; i <= n; i++ ) {
    inv[i] = (-(m/i) * inv[m%i] ) % m;
    inv[i] = inv[i] + m;
}
```

## Chinese Remainder Theorem:

```
long long inv(long long a, long long m)
{
    long long m0 = m, t, q;
    long long x0 = 0, x1 = 1;

    if (m == 1)
        return 0;

    // Apply extended Euclid Algorithm
    while (a > 1) {
        // q is quotient
        q = a / m;

        t = m;

        // m is remainder now, process same
as
        // euclid's algo
        m = a % m, a = t;

        t = x0;

        x0 = x1 - q * x0;

        x1 = t;
    }

    // Make x1 positive
    if (x1 < 0)
        x1 += m0;

    return x1;
}

long long findMinX(long long num[], long long
rem[], long long k)
{
    // Compute product of all numbers
    long long prod = 1;
    for (long long i = 0; i < k; i++)
        prod *= num[i];

    // Initialize result
    long long result = 0;

    // Apply above formula
    for (long long i = 0; i < k; i++) {
        long long pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) *
pp;
    }

    return result % prod;
}
```

## Or (log(m2+m3+…….mn)

```
ll x=a[0].ff,y=a[0].ss;  // x=rem and y=mod;
for(int i=1; i<n; i++)
{
    while(x%a[i].ss!=a[i].ff)
    {
        x+=y;
    }
    ll gcd=__gcd(a[i].ss,y);
    y=((y*a[i].ss)/gcd);
}
x%=y;
if(x<0)
    x+=y;
cout<<x<<endl;
```

## Segment Sieve

```
void SegmentSieve(int L,int R){
  if(L==1) L++;
  int maxN=R-L+1;
  int a[maxN]={0};
  for(auto p: prime)
  {
    if(p*p<=R)
    {
      int x=(L/p)*p;
      if(x<L) x+=p;
     for(int i=x;i<=R;i+=p)
     {
       if(i!=p)
       a[i-L]=1;
     }
    }
    else break;
  }
for(int i=0;i<maxN;i++)
    if(a[i]==0) cout<<i+L<<endl;
}
```

### Sum of Number of Divisors from 1 to N (SNOD)----(Sqrt(N)):

*N*=5,

*SNOD*(5)=*NOD*(1)+*NOD*(2)+*NOD*(3)+*NOD*(4)+*NOD*(5)=1+2+2

```
int SNOD( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}
```

### Sum of all Divisors from 1 to N---(Sqrt(N)):

```
int mod = 1000000007;
// Functions returns sum
// of numbers from 1 to n
int linearSum(int n)
{
    return (n * (n + 1) / 2) % mod;
}
// Functions returns sum
// of numbers from a+1 to b
int rangeSum(int b, int a)
{
    return (linearSum(b) -
            linearSum(a)) % mod;
}
// Function returns total
// sum of divisors
int totalSum(int n)
{
// Stores total sum
    int result = 0;
    int i = 1;

    // Finding numbers and
    //its occurrence
    while(true)
    {// Sum of product of each
        // number and its occurrence
        result += rangeSum(n / i, n / (i + 1)) * (i % mod) % mod;

        result %= mod;

        if (i == n)
            break;

        i = n / (n / (i + 1));
    }
    return result;
}
```

### Prob:for(1 to n)ans+=gcd(i,n);return ans;

1. GCD(i,N)=one of the dvisors of N
2. Instead of running loop from 1 to N,we can check for each divisor d of N how many numbers i are there with GCD (i,N)=d
   ```
   for(int i=1;i*i<=n;i++)
     if(n%i==0)
   int d1=i;
   int d2=N/i;
   res+=d1*getCount(d1,N);
   if(d1!=d2)
   res+=d2*getCount(d2,N);
   ```
3. Let x1,x2,x3…….xm are m different integers from 1 to N such that their GCD with N is d

   1<=xi<=N

   1<=xi/d<=N/d

   **So,#of integers having GCD d with N=#of integers Coprime with N/d**
   ```
   int getCount(int d,int N)
   return phi[N/d];
   ```

### Fermat primality test :

```
bool probablyPrimeFermat(int n, int iter=5) {
    if (n < 4)
        return n == 2 || n == 3;
for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (binpower(a, n - 1, n) != 1)
            return false;
    }
    return true;
}

using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
```

```
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}
```

**Miller Rabin primality test :**

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) { // returns true if n is probably prime, else returns false.
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}
```

**Prob:Given N, you have to answer Q quries, In each query you will be given a number K. You have to find count of common divisors of N and K:**

N=1800,k=200

1. Using each prime p in prime factorization of N,factorize K.
   1800={(2,3),(3,2),(5,2)}
   200=={(2,3),(3,0),(5,1)}
2. For each prime find minimum count and calculate total diviors.
   {(2,3),(3,0),(5,1)}
   So,Total number of divisors=(3+1)*(1)*(1+1)

Given K,find number of divisors of N which are multiple of K.

K=P1^a1*P2^a2……*Pm^am

1. Let d be a multiple of K and divides N,then all primes which exist in prime factorization of K must also exit in d and for each prime their power in d must be at least as much as in K.
   K=60={(2,2),(3,1),(5,1)}
   d=180={(2,2),(3,2),(5,1)}
2. d can not have any prime which is not present in N.
   d=2*3*3*5
   N=2*3*3*7 X
3. let prime P is present in d with count x and in N with count y the x<=y. So, y is upper bound.
   d=2*3*3*5
   N=2*3*3*5*5

Let N=2*3*3*3*5*5
 K=2*3

d=

| 2 | 3 | 5 |
|---|---|---|
| 1 (1-1) | 3 (1-3) | 3 (0-2) |

**Number of Digits of Factorial:**

```
int factorialDigitExtended ( int n, int base )
{
    double x = 0;
    for ( int i = 1; i <= n; i++ ) {
        x += log10 ( i ) / log10(base); // Base Conversion
    }
    int res = x + 1 + eps; // eps=10^-9
    return res;
}
```

**Prime factorization of factorial:**

**A given prime p,N! will have p^x as its prime factor where x=N/p + N/p^2 + N/p^3….. until it becomes 0.**

```
void factFactorize ( int n ) {
    for ( int i = 0; i < prime.size() && prime[i] <= n; i++ ) {
        int x = n;
        int freq = 0;

        while ( x / prime[i] ) {
                freq += x / prime[i];
                x = x / prime[i];
            }

            printf ( "%d^%d\n", prime[i], freq );
        }
    }
```

### Number of trailing Zeros of Factorial:

For 10! we have $x$=10/2+10/4+10/8=5+2+1=8 and $y$=10/5=2. Therefore number of trailing zero is $MIN(x,y)=MIN(8,2)=2$.

```
//calculating the count of x in the number n
int Num_of_trailing_Zeros_of_Factorial(int n, int x)
{
    int c = 0;
    while(n>0)
    {
        c+=n/x;
        n = n / x;
    }
    return c;
}
```

### Leading Digits of Factorial:

```
// Find the first K digits of N!
const double eps = 1e-9;
int leadingDigitFact ( int n, int k ) {
    double fact = 0;

    // Find log(N!)
    for ( int i = 1; i <= n; i++ ) {
        fact += log10 ( i );
    }

    // Find the value of q
    double q = fact - floor ( fact+eps );

    double B = pow ( 10, q );

    // Shift decimal point k-1 \times
    for ( int i = 0; i < k - 1; i++ ) {
        B *= 10;
    }

    // Don't forget to floor it
    return floor(B+eps);
}
```

### Trailing Zeros (I):

**you are given an integer. You can convert it to any base you would want to. But the condition is that if you convert it to any base then the number in that base should have at least one trailing zero, that means a zero at the end.**

we can see that we get 0 as remainder only when the number N is divided by the base. Only then we can get the first remainder as 0. So, we need to find out the number of divisors of N. N is always divided by 1. But we have to ignore it as the question demands us to find base from 2 to infinity. So, we have to reduce our answer by 1.

### Tailing Zeros (II):

**Ques:nCr * p^q**

**we can say that we will get (2 X 5) as many as the number of trailing zeros.**

If a number can be expressed as the product of x number of 2 and y number of 5 (other multiples may present), then there will be min(x, y) numbers of (2 X 5) unique pairs. This will be the number of trailing zeros in the number n.

### Tailing Zeros (III):

**we need to find the minimum natural number N such that, N! has exactly Q zeros on its trail (trailing zeros)**

Use Binary Search and use Num_of_trailing_Zeros_of_Factorial **Func**.

### Tailing Zeros (IV):

**n! (factorial n) has at least t trailing zeroes in b based number system. Given the value of n and t, what is the maximum possible value of b?**

**Sol:**Use **factFactorize**{

```
  if(freq>=t)
    {
      ans*=big_mod(prime[i],freq/t)%MOD;
      ans%=MOD;
    } }
```

### Exponentional(II): Calculate values a^b^c modulo 10^9+7:

First we have to calculate x=b^c MOD ((10^9+7)-1). Then calculate a^x MOD 10^9+7. Because x itself is a MOD reduced value.

```
  ll x=binpow(b,c,MOD-1);
  ll y=binpow(a,x,MOD);
```

### Common Divisors:

**You are given an array of *n* positive integers. Your task is to find two integers such that their greatest common divisor is as large as possible.**

```
int main()
{
    int n;
    cin>>n;
    vector<int> range(1e6+1,0);
    for(int i=0; i<n; i++)
    {
        int x;
        cin>>x;
        range[x]++;
    }
    for(int gcd=1e6; gcd >=1; gcd--)
    {
        int multiples=0;
        for(int pointer=gcd,pointer<=1e6;
pointer+=gcd)
        {
            multiples+=range[pointer];
        }
        if(multiples>1)
        {
            cout<<gcd<<endl;
            return 0;
        }
    }
}
```

## Divisor Analysis:

**There are *n* lines that describe the factorization. Each line has two numbers *x* and *k* where *x* is a prime and *k* is its power. Your task is to find the number, sum and product of its divisors**

```
ll gem(ll base, ll power)
{
 ll x= (binpow(base,power+1,MOD)-1+MOD)%MOD;

  ll y=binpow(base-1,MOD-2,MOD);
  x*=y;
  x%=MOD;
  // we have (a^b)^c which is equal to
a^(b*c) as you said. so in this problem, the
product b*c can be very large so we use the
theorem to calculate the product mod (p-1)
and without this we wont be able to calculate
it. and something slightly different was
going on with the other problem, we couldnt
calculate b^c so we needed to use the same
reduction and calculated b^c mod p-1 using
binary exponentiation.
  return x;
}
void solve()
{
  ll n;
  cin>>n;
  ll p[n],exp[n];
  for (int i=0; i<n; i++)
  {
    cin>>p[i]>>exp[i];
  }
  ll numOfdiv=1;
  for (int i=0; i<n; i++)
  {
    numOfdiv=(numOfdiv*(exp[i]+1))%MOD;
  }
  ll sumOfdiv=1;
  for (int i=0; i<n; i++)
  {

sumOfdiv=(sumOfdiv*(gem(p[i],exp[i])))%MOD;
  }

 ll proOfdiv=1,pos=-1;
 for (int i=0; i<n; i++)
 {
   if(exp[i]%2)
   pos=i;
 }
 if(pos!=-1)
 {
   ll outer=1;
   for (int i=0; i<n; i++)
   {
     if(i==pos)
     {
       outer=(outer*((exp[i]+1)/2))%(MOD-1);
     }
     else
     outer=(outer*((exp[i]+1)))%(MOD-1);
   }
   for (int i=0; i<n; i++)
   {

proOfdiv=(proOfdiv*(binpow(p[i],(exp[i]*outer
)%(MOD-1),MOD)))%MOD;
   }
 }
 else
 {
   ll outer=1;
   for (int i=0; i<n; i++)
   {


     outer=(outer*((exp[i]+1)))%(MOD-1);
   }
   for (int i=0; i<n; i++)
   {

proOfdiv=(proOfdiv*(binpow(p[i],((exp[i]/2)*o
uter)%(MOD-1),MOD)))%MOD;
   }
 }
  cout<<numOfdiv<<" "<<sumOfdiv<<"
"<<proOfdiv<<endl;
}
```

## Prime Multiples:

**Your task is to calculate how many of the first *n* positive integers are divisible by at least one of the given prime numbers**

```
if(k==1)
  {
    cout<<n/a[0]<<endl;
  }
  else
  {
    ll ans=0;
    for(ll mask=1;mask<(1<<k);mask++)
    {
      ll x=0,tmp=n;
      for(ll i=0;i<k;i++)
      {
        if((1<<i)&mask)
        {
          x++;
          tmp/=a[i];
        }
      }
      if(x%2==0)
      ans-=tmp;
      else
      ans+=tmp;
    }
    cout<<ans<<endl;
  }
```

### Odd numbers of Divisor Count:

The divisor count of Square number is always odd. We have to run a loop from 1 to sqrt(N). **Formula: (2*a+1)**. The we have to use upperbound and lowerbound function.

## Graph Theory

### Bipartite Graph Test:

```
bool dfs(int v, int c)
{
      vis[v]=1;
      col[v]=c;
      for(int child : ar[v]){
            if(vis[child]==0){
            if(dfs(child,c^1)==false)
                  return false;
            }
            else
                  if(col[v]==col[child])
                        return false;
      }
  return true;
}
```

### Cycle Detection:

**Returns if Graph has a cycle or not**

```
bool dfs(int node, int par)
{
      vis[node]=1;
      for(int child : ar[node]){
            if(vis[child]==0){
            if(dfs(child,node)==true)
                  return true;
            }
            else
                  if(child!=par)
                        return true;
      }
  return false;
}
```

### In Out Time of Nodes:

**Given 2 nodes, find whether one node lies in the subtree of another node.**

```
int timer=1;
bool dfs(int v)
{
      vis[v]=1;
      In[v]=timer++;
      for(int child : ar[v]){
            if(vis[child]==0){
                  dfs(child);
            }
      }
      Out[v]=timer++;
}
```

### Topological Sort:

```
void toposort()
{
 queue<int> q;
//priority_queue<int,vector<int>,greater<int>
> q; // for printing Toposort in
lexigraphically smallest order.
  for(int i=1;i<=n;i++)
  {
    if(in[i]==0)
    q.push(i);
  }
  while(!q.empty())
  {
    int cur=q.top();
    q.pop();
    res.push_back(cur);
    for(auto node : v[cur])
    {
      in[node]--;
      if(in[node]==0)
      q.push(node);
    }

  }
        cout<<"TopSort: ";
        for(int node: res)
         cout<<node<<" ";
}
```

### Disjoint Set Union:

```
#define ll    long long int
ll parent[200005],sz[200005];

    void makeSet(ll i)
{
    parent[i]=i;
    sz[i]=0;

}
ll findRepresentive(ll a)
{
    if(parent[a]==a) return a;
    ll r=findRepresentive(parent[a]);
    parent[a]=r;
    return r;

}
bool Union(ll a,ll b)
{
    ll x=findRepresentive(a);
    ll y=findRepresentive(b);
    if(x!=y)
    {

        if(sz[x]>sz[y])
        {
          sz[x]+=sz[y];
          parent[y]=x;


        }
        else
        {
          sz[y]+=sz[x];
          parent[x]=y;

        }
        return true;
    }
    return false;
}
```

### Minimum Spanning Tree (MST):

```
#include"DisjointSetUnion.h"
int n;
vector<pair<int,pair<int,int>>> v;
void MST()
{
    int mst=0;
    DisjointSetUnion d;
    for(int i=1;i<=n;i++)
    {
        d.makeSet(i);
    }
    for(auto edge: v)
    {

if(d.findRepresentive(edge.second.first)!=d.f
indRepresentive(edge.second.second))
        {

d.Union(edge.second.first,edge.second.second)
;
            mst+=edge.first;
        }
    }
    cout<<mst<<endl;

}
int main()
{
   int e;
    cin>>n>>e;
    for(int i=0;i<e;i++)
    {
        int x,y,z;
        cin>>x>>y>>z;
        v.push_back({z,{x,y}});

    }
    sort(v.begin(),v.end());
    MST();
}
```

### Dijsktra:

```
void dijsktra()
{

priority_queue<pair<int,int>,vector<pair<int,
int> >, greater<pair<int,int> > > pq;
    vector<int> dist(n+1,INF);
    pq.push({0,1});
    dist[1]=0;
    while(!pq.empty())
    {
        int curr=pq.top().second;
        int curr_d=pq.top().first;
        pq.pop();
        for(pair<int,int> edge : adj[curr])
        {
            if(curr_d+edge.second
<dist[edge.first])
            {

dist[edge.first]=curr_d+edge.second;

pq.push({dist[edge.first],edge.first});
            }
        }
    }
    for(int i=1; i<=n; i++)
        cout<<dist[i]<<" ";
}
```

**Prime Path:** Given two 4 digit prime numbers (A and B). Find minimum number of operations to convert A into B.

11,13,17,31

We create a path from one node to another for only one digit change. For example 11->13,11->17,11->31 etc. After that we use BFS for finding shortest distance which is considered minimum number of operations.

### Shortest Path in Directed Acyclic Graph (DAG):

```
#include<iostream>
#include<stack>
#define NODE 6
#define INF 9999

using namespace std;

int cost[NODE][NODE] = {
   {0, 5, 3, INF, INF, INF},
   {INF, 0, 2, 6, INF, INF},
   {INF, INF, 0, 7, 4, 2},
   {INF, INF, INF, 0, -1, 1},
   {INF, INF, INF, INF, 0, -2},
   {INF, INF, INF, INF, INF, 0}
};

void topoSort(int u, bool visited[],
stack<int>&stk) {
   visited[u] = true;       //set as the node
v is visited
   for(int v = 0; v<NODE; v++) {
      if(cost[u][v]) {       //for
allvertices v adjacent to u
         if(!visited[v])
            topoSort(v, visited, stk);
      }
   }

   stk.push(u);       //push starting vertex
into the stack
}

void shortestPath(int start) {
   stack<int> stk;
   int dist[NODE];

   bool vis[NODE];
   for(int i = 0; i<NODE;i++)
      vis[i] = false;          // make all
nodes as unvisited at first

   for(int i = 0; i<NODE; i++)     //perform
topological sort for vertices
      if(!vis[i])
         topoSort(i, vis, stk);

   for(int i = 0; i<NODE; i++)
      dist[i] = INF;       //initially all
distances are infinity
   dist[start] = 0;       //distance for
start vertex is 0

   while(!stk.empty()) {    //when stack
contains element, process in topological
order
      int nextVert = stk.top(); stk.pop();

      if(dist[nextVert] != INF) {
         for(int v = 0; v<NODE; v++) {
            if(cost[nextVert][v] &&
cost[nextVert][v] != INF){ if(dist[v] >
dist[nextVert] +cost[nextVert][v])dist[v] =
dist[nextVert] + cost[nextVert][v];
         }
      }
   }
   for(int i = 0; i<NODE; i++)
      (dist[i] == INF)?cout << "Infinity
":cout << dist[i]<<" ";
}

main() {
   int start = 1;
   cout << "Shortest Distance From Source
Vertex "<<start<<endl;
   shortestPath(start);
}
```

**Round Trip (II):**

Byteland has *n* cities and *m* flight connections. Your task is to design a round trip that begins in a city, goes through one or more other cities, and finally returns to the starting city. Every intermediate city on the route has to be distinct.

```
ll n,m,vis[N]={0},dis[N];
vector<ll> a[N],res;
stack<ll> rec;
bool is[N];
bool dfs(ll st)
{
  vis[st]=1;
  is[st]=true;
  rec.push(st);
  for(auto it: a[st])
  {
    if(vis[it]==0)
    {
      if(dfs(it))
      return true;
    }
    else
    {
      if(is[it])
      {
       // dbg1(it);
        rec.push(it);
        return true;
      }
    }
  }
  rec.pop();
  is[st]=false;
  return false;
}

void solve() {
  ll q;

cin>>n>>m;
loop(i,0,n+1)
{
  vis[i]=0;
  is[i]=false;
}
ll st=-1;
loop(i,0,m)
{
  ll x,y,z;
  cin>>x>>y;

  a[x].push_back(y);
   //a[y].push_back({x});
}
ll f=0;
loop(i,1,n+1)
{
  if(vis[i]==0)
```

```
 {
  if(dfs(i)){
   f=1;
  break;
  }
 }
}
if(f==0)
cout<<"IMPOSSIBLE"<<endl;
else
{
 ll tmp=rec.top();
 rec.pop();
 vector<ll> res;
 res.push_back(tmp);
 while(rec.size()>0 && rec.top()!=tmp)
 {
  res.push_back(rec.top());
  rec.pop();
 }
  res.push_back(tmp);
  reverse(all(res));
  cout<<res.size()<<endl;
  for(auto it : res)
  {
   cout<<it<<" ";
  }

}
}
```

# Segment Tree

### Build:

```
void init(int node, int b, int e)
{
  if (b == e) {
    tree[node] = arr[b];
    return;
  }
  int Left = node * 2;
  int Right = node * 2 + 1;
  int mid = (b + e) / 2;
  init(Left, b, mid);
  init(Right, mid + 1, e);
  tree[node] = tree[Left] + tree[Right];
}
```

### Query:

```
int query(int node,int b,int e,int i,int j)
{
  if (i > e || j < b)
    return 0;
  if (b >= i && e <= j)
    return tree[node];
  int Left = node * 2;
  int Right = node * 2 + 1;
  int mid = (b + e) / 2;
  int p1 = query(Left, b, mid, i, j);
  int p2 = query(Right,mid + 1, e, i, j);
  return p1 + p2;
}
```

### Update:

```
void update(int k,int i=0,int j=n-1,int ti=1)
{
  if(i==j) tree[ti]=a[k];
  int mid=(i+j)/2;
  if(k<=mid) update(k,i,mid,2*ti);
  else update(k,mid+1,j,2*ti+1);
  tree[ti]=tree[2*ti]+tree[2*ti+1];
}
```

### Distinct Numbers in a Range O((N + Q)sqrt(N)) or O((N + Q)lg N):

```
bool cmp(pair<pr,pr> a, pair<pr,pr> b)
{
 return a.ss.ff<b.ss.ff;
}
void solve()
{
 ll q;
  cin>>n>>q;
  memset(tree,0,sizeof(tree));
    memset(last,0,sizeof(last));

  loop(i,1,n+1)
  {
   cin>>arr[i];
  }
  vector<pair<pr,pr>> offline;
 loop(i,0,q)
 {
  ll x,y;
  cin>>x>>y;
  offline.push_back({{y,x},{i,0}});
 }
 sort(all(offline));
 ll j=0;
 loop(i,1,n+1)
 {
  if(last[arr[i]])
  {
   ll s=last[arr[i]];
   update(1,n,s,1);
  }
  last[arr[i]]=i;
  update(1,n,i,1);
  while(offline[j].ff.ff==i)
  {
   ll
ans=query(1,n,offline[j].ff.ss,offline[j].ff.
ff,1);
   offline[j].ss.ss=ans;
   j++;
  }

 }
 sort(all(offline),cmp);
 loop(i,0,q)
 {
  cout<<offline[i].ss.ss<<endl;
```

```
   }


}
```

### Maximum Subarray Sum in a given Range:

```
struct Node
{
int suffix,prefix,best_sum,sum;
};
int n;
Node tree[4*mx];
int arr[mx];
void combine(Node &lf, Node &rg , Node &n)
{
n.best_sum=max(lf.best_sum,rg.best_sum);
n.best_sum=max(n.best_sum,lf.suffix+rg.prefix
);
n.suffix=max(rg.suffix,rg.sum+lf.suffix);
n.prefix=max(lf.prefix,lf.sum+rg.prefix);
  n.sum=lf.sum+rg.sum;
}
void inti(int st, int end, int index)
{
    if(st==end)
    {
        tree[index].best_sum=arr[st];
         tree[index].suffix=arr[st];
          tree[index].prefix=arr[st];
           tree[index].sum=arr[st];
        return ;
    }
    int mid=(st+end)/2;
     int left=2*index,right=2*index+1;
    inti(st,mid,left);
     inti(mid+1,end,right);
combine(tree[left],tree[right],tree[index]);
}

Node query(int st,int end, int i, int j,int
index)
{Node res;
    if(i>end || j<st)
    {
      res.best_sum=INT32_MIN;
        res.suffix=INT32_MIN;
         res.prefix=INT32_MIN;
          res.sum=INT32_MIN;
           return res;
    }
    else if(i<=st && j>=end) return
tree[index];
     int mid=(st+end)/2;
     int left=2*index,right=2*index+1;
    Node p1 = query( st, mid, i, j, left);
    Node p2 = query(mid + 1, end, i,
j,right);
    combine(p1,p2,res);
    return res;
}
void update(int st,int end, int i,int index)
{   int j=i;
    if(i>end || j<st) return;
    else if(i<=st && j>=end)
    {
         tree[index].best_sum=arr[i];

         tree[index].suffix=arr[i];
          tree[index].prefix=arr[i];
           tree[index].sum=arr[i];
        return ;
    }
    int mid=(st+end)/2;
     int left=2*index,right=2*index+1;
       update( st, mid, i, left);
   update(mid + 1, end, i,right);
combine(tree[left],tree[right],tree[index]);
}
Node ans=query(0,n-1,x,y,1);
      cout<<ans.best_sum<<endl;
```

### (Merge Sort Tree)The number of elements greater than k in the subsequence from L to R:

```
int n;
vector<int> tree[4*mx];
int arr[mx];

void inti(int st, int end, int index)
{
    if(st==end)
    {
        tree[index].push_back(arr[st]);
        return ;
    }
    int mid=(st+end)/2;
     int left=2*index,right=2*index+1;
    inti(st,mid,left);
     inti(mid+1,end,right);
     int i=0,j=0;
     while(i<tree[left].size() &&
j<tree[right].size())
     {
if(tree[left][i]<=tree[right][j])
         {
tree[index].push_back(tree[left][i]);
             i++;
         }
         else
         {
tree[index].push_back(tree[right][j]);
              j++;
         }
     }
    while(i<tree[left].size())
    {
tree[index].push_back(tree[left][i]);
        i++;
    }
    while(j<tree[right].size())
    {
tree[index].push_back(tree[right][j]);
```

```
        j++;
    }
}
int query(int st,int end, int i, int j,int
index, int k)
{
    if(i>end || j<st) return 0;
    else if(i<=st && j>=end)
    {int
x=upper_bound(tree[index].begin(),tree[index]
.end(),k)-tree[index].begin();
        return (tree[index].size()-x);
    }
     int mid=(st+end)/2;
     int left=2*index,right=2*index+1;
         int p1 = query( st, mid, i, j,
left,k);
    int p2 = query(mid + 1, end, i,
j,right,k);
    return p1+p2;
}
```

## K'th smallest number(Persistent Seg Tree)

### Solution-1(Persistent Segment Tree):

```
#define MAXN 100005

struct node
{
    int val;
    node* left, *right;
    node() {}
    node(node* l, node* r, int v)
    {
        left = l;
        right = r;
        val = v;
    }
};

node* version[MAXN];
void build(node* n,int low,int high)
{
    if (low==high)
    {
        n->val = 0;
        return;
    }
    int mid = (low+high) / 2;
    n->left = new node(NULL, NULL, 0);
    n->right = new node(NULL, NULL, 0);
    build(n->left, low, mid);
    build(n->right, mid+1, high);
    n->val = n->left->val + n->right->val;
}

void upgrade(node* prev, node* cur, int low,
int high,
             int idx, int value)
{
    if (idx > high or idx < low or low >
high)
        return;

    if (low == high)
    {
        cur->val = value;
        return;
    }
    int mid = (low+high) / 2;
    if (idx <= mid)
    {
        cur->right = prev->right;
        cur->left = new node(NULL, NULL, 0);
        upgrade(prev->left,cur->left, low,
mid, idx, value);
    }
    else
    {

        cur->left = prev->left;
        cur->right = new node(NULL, NULL, 0);
        upgrade(prev->right, cur->right,
mid+1, high, idx, value);
    }

    cur->val = cur->left->val +
cur->right->val;
}

int query(node* past, node *pres, int l, int
r, int k)
{
    if(l==r)
    {
        return l;
    }
    else
    {
        int mysegC=
pres->left->val-past->left->val;
        int mid=(l+r)/2;
        if(k<=mysegC)
        {
            return
query(past->left,pres->left,l,mid,k);
        }
        else
        {
            return
query(past->right,pres->right,mid+1,r,k-myseg
C);
        }
    }
}

int query1(node * cur, int lo, int hi, int i,
int j)
{
    if (hi < i || lo > j)
        return 0;
    if (i <= lo && hi <= j)
    {
        return cur->val;
    }
    int mid = lo + hi >> 1;
    int lf = query1(cur->left,lo, mid, i, j);
    int rt= query1(cur->right,mid + 1, hi, i,
j);
    int val = lf+rt;
    return val;
}

int main()
{
    fast_io;
    int n,q;
    cin>>n>>q;
    vector<int> a(n),indexTree(n);
    vector<pair<int,int>> sorted;
    for(int i=0; i<n; i++)
    {
        cin>>a[i];
        sorted.push_back({a[i],i});
    }
    sort(sorted.begin(),sorted.end());
    for(int i=0; i<n; i++)
    {
        indexTree[sorted[i].second]=i;
    }
    node* root = new node(NULL, NULL, 0);
    build(root, 0, n-1);
    version[0]=root;
    for(int i=0; i<n; i++)
    {
        version[i+1]= new node(NULL, NULL,
0);

upgrade(version[i],version[i+1],0,n-1,indexTr
ee[i],1);
    }
    while(q--)
    {
        int l,r,k;
        cin>>l>>r>>k;
        l--,r--;
        int
ans=query(version[l],version[r+1],0,n-1,k);
        cout<<sorted[ans].first<<'\n';
    }

    return 0;
}
```

**Solution-2(Merge Sort Tree + Binary Search):**

```
const int N = 4e5+5;
int n,q;
int a[N];
vector<int>v;
vector<int> seg[N];
void build(int n,int b,int e)
{
    if(b==e){
        seg[n].pb(a[b]);
        return;
    }
    int mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e);
    merge(seg[lc].begin(),seg[lc].end(),
seg[rc].begin(),seg[rc].end(),
back_inserter(seg[n]));
}


int query(int n,int b,int e,int i,int j,int v)
{
    if(b>j || e< i) return 0;
    if(b>=i && e<= j) {
        int k = upper_bound(all(seg[n]), v )
- seg[n].begin();
        return k;
    }
    int mid = (b+e)/2;
    return query(lc,b,mid,i,j,v) +
query(rc,mid+1,e,i,j,v);
}



int main()
{
    //freopen("in.txt","r",stdin);
    scanf("%d %d",&n,&q);
    for(int i = 1; i <= n; i ++ ) {
        scanf("%d",&a[i]);
        v.pb(a[i]);
    }
    sort(all(v));
    for(int i = 1;i <= n; i ++ ) {
        a[i] = lower_bound(all(v),a[i]) -
v.begin() + 1;
    }

    build(1,1,n);

    while(q--) {
        int l,r,x;
        scanf("%d %d %d",&l,&r,&x);
        int low = 1, high = n , mid, ans ;
        while(low <= high) {
            mid = low + high >> 1;
            int k = query(1,1,n,l,r,mid);

            if(k >= x) {
                ans = mid;
                high = mid-1;
            }
            else low = mid+1;
        }
        printf("%d\n",v[ans-1]);
    }
    return 0;
}
```

### All Possible increasing Subsequence:

**An increasing subsequence from a sequence $\{A_1, A_2 \ldots A_n\}$ is defined by $\{A_{i1}, A_{i2} \ldots A_{ik}\}$, where the following properties hold:**
**$i_1 < i_2 < i_3 < \ldots < i_k$, and**
**$A_{i1} < A_{i2} < A_{i3} < \ldots < A_{ik}$.**
**Now you are given a sequence, you have to find the number of all possible increasing subsequences.**

**Sol:**

```
ll tree[4*N];
ll lazy[4*N];
ll arr[N];
 ll height;
 ll num=1;
ll query(ll st,ll end, ll i, ll j,ll index)
{
    if(i>end || j<st) return 0;
    else if(i<=st && j>=end) {
     // dbg3(i,j,tree[index])
    return tree[index];
    }
     ll mid=(st+end);
     mid=(mid>>1LL);
     ll left=index << 1LL; ll right=left|
1LL;
         ll p1 = query( st, mid, i, j, left);
    ll p2 = query(mid + 1, end, i, j,right);
    return (p1+p2)%MOD;
}
void update(ll st,ll end, ll i,ll val,ll
index)
{
    ll j=i;
    if(i>end || j<st) return;
    else if(i<=st && j>=end)
    {
         tree[index]+=val;
         //dbg2(i,val)
        if(tree[index]>=MOD)
tree[index]-=MOD;
         return;
    }
     ll mid=(st+end);
     mid=(mid>>1LL);
     ll left=index << 1LL; ll right=left|
1LL;
       update( st, mid, i,val, left);
   update(mid + 1, end, i,val,right);
    tree[index]=(tree[left]+tree[right])%MOD;
}
map<ll,ll> mp;
set<ll> st;
ll n,q;

void solve()
{
  memset(tree,0,sizeof(tree));
  mp.clear();
  st.clear();
    cin>>n;
    vector<ll> a(n);

    loop(i,0,n)
    {
      cin>>a[i];
      st.insert(a[i]);
    }
    ll num=1;
    for(auto it : st)
    {
      mp[it]=num;
    //  dbg2(it,num)
      num++;
    }
   // dbg1(num)
    ll ans=0;
    loop(i,0,n)
    {
      ll x=query(0,num,0,mp[a[i]]-1,1);
      update(0,num,mp[a[i]],x+1,1);
     // dbg2(x,mp[a[i]])
      ans+=x+1;
      ans%=MOD;

    }
    cout<<ans<<endl;
}
```

### Strongest Communitity:

In a strange city, houses are built in a straight line oneafter another. There are several communities in the city. Each communityconsists of some **consecutive** houses such that every house belongs to **exactly** one community. The houses are numbered from **1** to **n**, and thecommunities are numbered from **1** to **c**.
Now some inspectors want to find the strongest communityconsidering all houses from **i** to **j**. A community is strongest ifmaximum houses in the range belong to this community. So, there can be morethan one strongest community in the range. So, they want to know the number ofhouses that belong to the strongest community. That's why they are seeking yourhelp.

**Sol:**

```
struct Node {
  ll mx_cnt;
ll value;
};
```

```
ll tree[4*N] ;


ll arr[N],a[N];

void build(ll st, ll end, ll index)
{
  if(st==end)
  {

          tree[index]=a[st];
    return;
  }
  ll mid = (st+end)/2;
  ll left=2*index;
  ll right=left+1;
  build(st,mid,left);
  build(mid+1,end,right);
  tree[index]=max(tree[left],tree[right]);

}
ll query(ll st,ll end, ll i, ll j,ll index)
{


  if(i>end || j<st) return 0;
  if(i<=st && j>=end)
  {
    return tree[index];
  }

  ll mid = (st+end)/2;
  ll left=2*index;
  ll right=left+1;
 ll ans1= query(st,mid,i,j,left);
  ll ans2=query(mid+1,end,i,j,right);


   return max(ans1,ans2);

}

ll n,q;
void solve()
{
  ll c;
  cin>>n>>c>>q;


  map<ll,ll> last,mp;
  loop(i,0,n)
  {
    ll x;
    cin>>x;
    mp[x]++;
    arr[i]=x;
    last[x]=i;
    a[i]=mp[x];
  }
  build(0,n-1,1);
  while(q--)
  {
    ll x,y;
    cin>>x>>y;
    x--,y--;
    ll v=arr[x];
    ll rng=last[v]-x+1;
    ll p=x;
    x=last[v]+1;
    if(x>y)
    {
      cout<<y-p+1<<endl;
    }
    else
    {
      ll qq=query(0,n-1,x,y,1);
      qq=max(qq,rng);
      cout<<qq<<endl;
    }

  }
}
```

## Hotel Quries:

**There are *n* hotels on a street. For each hotel you know the number of free rooms. Your task is to assign hotel rooms for groups of tourists. All members of a group want to stay in the same hotel.**

**The groups will come to you one after another, and you know for each group the number of rooms it requires. You always assign a group to the first hotel having enough rooms. After this, the number of free rooms in the hotel decreases.**

```
struct tt{
  ll val,ind;
};
tt tree[4*N];
ll arr[N];

void build(ll st, ll end, ll index)
{
    if(st==end)
    {
        tree[index].val=arr[st];
        tree[index].ind=st;
        return ;
    }
    ll mid=(st+end)/2;
     ll left=2*index,right=2*index+1;
    build(st,mid,left);
     build(mid+1,end,right);
     if(tree[left].val>=tree[right].val)
     {
       tree[index]=tree[left];
     }
     else
     tree[index]=tree[right];

    // cout<<index<<" "<<tree[index]<<endl;
}
tt query(ll st,ll end, ll k,ll index)
{
    if(st==end)
    {
      if(tree[index].val>=k)
      return tree[index];
      else
      {
        tt p={INT64_MIN/100,-1};
        return p;
      }

    }
     ll mid=(st+end)/2;
     ll left=2*index,right=2*index+1;
     if(tree[left].val>=k)
        return query( st, mid, k, left);
        else if(tree[right].val>=k)
    return query(mid + 1, end, k,right);
    tt p={INT64_MIN/100,-1};
     return p;
}
void update(ll st,ll end, ll i,ll index)
{
    ll j=i;
    if(i>end || j<st) return;
    else if(i<=st && j>=end)
    {
          tree[index].val=arr[st];
         tree[index].ind=st;
         return;
    }
    ll mid=(st+end)/2;
     ll left=2*index,right=2*index+1;
       update( st, mid, i, left);
   update(mid + 1, end, i,right);
   if(tree[left].val>=tree[right].val)
     {
       tree[index]=tree[left];
     }
     else
     tree[index]=tree[right];
}
ll n,q;
void solve()
{

    cin >> n>>q;
    for(ll i=0;i<n;i++)
            cin>> arr[i];
  build(0,n-1,1);
    while (q--)
    {
        ll x,y,k;
        cin>>k;
      tt p=query(0,n-1,k,1);
      if(p.ind==-1)
      {
          cout<<0<<" ";
          continue;
        }
        arr[p.ind]-=k;
       // dbg2(arr[p.ind],p.ind);
        update(0,n-1,p.ind,1);
        cout<<p.ind+1<<" ";
    }

}
```

### Segment Tree Lazy Propagation:

```
struct info {
    i64 prop, sum;
} tree[mx * 3]; //sum ছাড়াও নিচে অতিরিক্ত কত যোগ
হচ্ছে সেটা রাখবো prop এ
void update(int node, int b, int e, int i,
int j, i64 x)
{
    if (i > e || j < b)
        return;
    if (b >= i && e <= j) //নোডের রেঞ্জ আপডেটের
রেঞ্জের ভিতরে
    {
        tree[node].sum += ((e - b + 1) * x);
//নিচে নোড আছে e-b+1 টি, তাই e-b+1 বার x যোগ হবে
এই রেঞ্জে
        tree[node].prop += x; //নিচের নোডগুলোর
সাথে x যোগ হবে
        return;
    }
    int Left = node * 2;
    int Right = (node * 2) + 1;
    int mid = (b + e) / 2;
    update(Left, b, mid, i, j, x);
    update(Right, mid + 1, e, i, j, x);
    tree[node].sum = tree[Left].sum +
tree[Right].sum + (e - b + 1) *
tree[node].prop;
    //উপরে উঠার সময় পথের নোডগুলো আপডেট হবে
    //বাম আর ডান পাশের সাম ছাড়াও যোগ হবে নিচে
অতিরিক্ত যোগ হওয়া মান
}
int query(int node, int b, int e, int i, int
j, int carry = 0)
{
    if (i > e || j < b)
        return 0;

    if (b >= i and e <= j)
        return tree[node].sum + carry * (e -
b + 1); //সাম এর সাথে যোগ হবে সেই রেঞ্জের সাথে
অতিরিক্ত যত যোগ করতে বলেছে সেটা

    int Left = node << 1;
    int Right = (node << 1) + 1;
    int mid = (b + e) >> 1;
    int p1 = query(Left, b, mid, i, j, carry
+ tree[node].prop); //প্রপাগেট ভ্যালু বয়ে নিয়ে যাচ্ছে
carry ভ্যারিয়েবল
    int p2 = query(Right, mid + 1, e, i, j,
carry + tree[node].prop);

    return p1 + p2;
}
```

## Horriable Queries:

You are given an array of **n** elements, which are initially all **0**. After that you will be given **q** commands. They are:

0 x y v - you have to add **v** to all numbers in the range of **x** to **y**

1 x y - print a line containing a single integer which is the sum of all the array elements between **x** and **y** (inclusive).

The array is indexed from **0** to **n - 1**.

**Sol:**

```
ll tree[4*N],lazy[4*N],arr[N];
void build(ll st, ll end, ll index)
{
  if(st==end)
  {
    tree[index]=0;
    lazy[index]=0;
    return;
  }
  ll mid = (st+end)/2;
  ll left=2*index;
  ll right=left+1;
  build(st,mid,left);
  build(mid+1,end,right);
  tree[index]=tree[left]+tree[right];
}
ll query(ll st,ll end, ll i, ll j,ll index)
{
  if(lazy[index]!=0)
  {
    ll add=lazy[index];
    lazy[index]=0;
    ll left=2*index;
  ll right=left+1;
    if(st!=end)
    {
      lazy[left]+=add;
      lazy[right]+=add;
    }
    tree[index]+=(end-st+1)*add;
  }
  if(i>end || j<st) return 0;
  if(i<=st && j>=end) return tree[index];
  ll mid = (st+end)/2;
  ll left=2*index;
  ll right=left+1;
 ll ans1= query(st,mid,i,j,left);
  ll ans2=query(mid+1,end,i,j,right);
  return ans1+ans2;
}
void update(ll st,ll end, ll i,ll j,ll index
, ll value)
{
  if(lazy[index]!=0)
  {
    ll add=lazy[index];
    lazy[index]=0;
    ll left=2*index;
  ll right=left+1;
    if(st!=end)
    {
      lazy[left]+=add;
      lazy[right]+=add;
    }
    tree[index]+=(end-st+1)*add;
  }
  if(i>end || j<st) return;
  if(i<=st && j>=end)
  {
   tree[index]+=(end-st+1)*value;
      ll left=2*index;
  ll right=left+1;
    if(st!=end)
    {
      lazy[left]+=value;
      lazy[right]+=value;
    }

    return;
  }
  ll mid = (st+end)/2;
  ll left=2*index;
  ll right=left+1;
 update(st,mid,i,j,left,value);
  update(mid+1,end,i,j,right,value);
    tree[index]=tree[left]+tree[right];
}

ll n,q;
void solve()
{
  cin>>n>>q;
  memset(lazy,0,sizeof(lazy));
   memset(tree,0,sizeof(tree));
  while (q--)
  {
    ll k,x,y,val;
    cin>>k;
    if(k==0)
    {
      cin>>x>>y>>val;
      update(0,n-1,x,y,1,val);
    }
    else
    {
      cin>>x>>y;
      ll ans=query(0,n-1,x,y,1);
      cout<<ans<<endl;
    }
  }
}
```

## Sum of Squares with Segment Tree:

```
2 --- return the sum of the squares of the
numbers [l,r]
1--- x -- add "x" to all numbers with indices
0 --- x -- set all numbers with indices [l,r]
ll arr[MX];
struct Node
{
    ll sum,sqrsum,lazy,upd;
} tree[3*MX];

void init(int node, int b, int e)
{
    if(b==e)
    {
        tree[node].sum=arr[b];
        tree[node].sqrsum=sqr(arr[b]);
        tree[node].lazy=0;
        tree[node].upd=0;
        return;
    }

    ll left=node*2;
    ll right=(node*2)+1;
    ll mid=(b+e)/2;

    init(left,b,mid);
    init(right,mid+1,e);


tree[node].sum=tree[left].sum+tree[right].sum
;

tree[node].sqrsum=tree[left].sqrsum+tree[righ
t].sqrsum;
    tree[node].lazy=0;
    tree[node].upd=0;
}

void push_down(int node, int b, int e)
{
    ll left=node*2;
    ll right=(node*2)+1;
    ll mid=(b+e)/2;

    if(tree[node].upd) // set value;
    {
        tree[left].lazy=0;
        tree[right].lazy=0;


tree[left].sqrsum=(mid-b+1)*sqr(tree[node].up
d);

tree[right].sqrsum=(e-mid)*sqr(tree[node].upd
);

tree[left].sum=(mid-b+1)*tree[node].upd;

tree[right].sum=(e-mid)*tree[node].upd;

        tree[left].upd=tree[node].upd;
        tree[right].upd=tree[node].upd;
        tree[node].upd=0;
    }
    if(tree[node].lazy) // add value;
    {

tree[left].sqrsum+=(tree[left].sum*(2*tree[no
de].lazy))+(mid-b+1)*sqr(tree[node].lazy);

tree[right].sqrsum+=(tree[right].sum*(2*tree[
node].lazy))+(e-mid)*sqr(tree[node].lazy);


tree[left].sum+=(mid-b+1)*tree[node].lazy;

tree[right].sum+=(e-mid)*tree[node].lazy;

        tree[left].lazy+=tree[node].lazy;
        tree[right].lazy+=tree[node].lazy;
        tree[node].lazy=0;
    }
}

void update(int node, int b, int e, int i,
int j, ll val, int type)
{
    if(b>j || e<i)
        return;
    if(b>=i && e<=j)
    {
        if(type==0)
        {
            tree[node].sum=(e-b+1)*val;

tree[node].sqrsum=(e-b+1)*sqr(val);
            tree[node].lazy=0;
            tree[node].upd=val;
        }
        else
        {

tree[node].sqrsum+=(tree[node].sum*2*val)+((e
-b+1)*sqr(val));
            tree[node].sum+=(e-b+1)*val;
            tree[node].lazy+=val;
        }
        return;
    }

    push_down(node,b,e);

    ll left=node*2;
    ll right=(node*2)+1;
    ll mid=(b+e)/2;
    update(left,b,mid,i,j,val,type);
    update(right,mid+1,e,i,j,val,type);


tree[node].sum=tree[left].sum+tree[right].sum
;

tree[node].sqrsum=tree[left].sqrsum+tree[righ
t].sqrsum;
}

ll query(int node, int b, int e, int i, int
j)
{
    if(b>j || e<i)
        return 0;
    if(b>=i && e<=j)
        return tree[node].sqrsum;

    push_down(node,b,e);

    ll left=node*2;
    ll right=(node*2)+1;
    ll mid=(b+e)/2;

    ll p=query(left,b,mid,i,j);
    ll q=query(right,mid+1,e,i,j);
    return p+q;
}

int main()
{
    ios::sync_with_stdio(0);  //cin.tie(0);

    ll t;
    cin>>t;
    for(int tc=1; tc<=t; tc++)
    {
        ll n,q;
        cin>>n>>q;
        for(int i=1; i<=n; i++)
            cin>>arr[i];

        init(1,1,n);

        cout<<"Case "<<tc<<":\n";
        while(q--)
        {
            ll type;
            cin>>type;
            if(type==0)
            {
                ll l,r,x;
                cin>>l>>r>>x;
                update(1,1,n,l,r,x,0);
            }
            else if(type==1)
            {
                ll l,r,x;
                cin>>l>>r>>x;
                update(1,1,n,l,r,x,1);
            }
            else if(type==2)
            {
                ll l,r;
                cin>>l>>r;
                cout<<query(1,1,n,l,r)<<endl;
            }
        }

    }

    return 0;
}
```

## String Algorithm

**Hashing:**

```
#define N 1
#define MAX 100000
long long base[N],mod[N],power[N][MAX+10];
int totalHash;
void init()
{
    totalHash=2;
    base[0]=3407;
    base[1]=4721;
    mod[0]=1000003999;
    mod[1]=1000000861;
    for(int i=0;i<totalHash;i++)
    {
        power[i][0]=1;
        for(int j=1;j<=MAX;j++)
        {

power[i][j]=((power[i][j-1]*base[i])%mod[i]);
        }
    }
}
struct HashData{
     long long ara[N][MAX+10],Hash[N];
   // char str[MAX+10];
   string str;
    int len;
    void init(string s)
    {
        str=s;
        len=s.size();
        for(int i=0;i<totalHash;i++)
        {
            ara[i][0]=str[0];
            for(int j=1;j<len;j++)
            {

ara[i][j]=(ara[i][j-1]*base[i])%mod[i];
                ara[i][j]+=str[j];
                if(ara[i][j]>=mod[i])
                ara[i][j]-=mod[i];
            }
            Hash[i]=ara[i][len-1];
        }
    }
    inline pair<int,int> query (int st,int
ed)
    {
        int ret[2];
        for(int i=0;i<totalHash;i++)
        {
            long long nw=ara[i][ed];
            if(st>0)
            {

nw-=(ara[i][st-1]*power[i][ed-st+1])%mod[i];
                if(nw<0)
                nw+=mod[i];
            }
            ret[i]=nw;
        }
        return {ret[0],ret[1]};
    }
    inline void append(char c)
    {len++;
        for(int i=0;i<totalHash;i++)
        {
            if(len>1)

ara[i][len-1]=(ara[i][len-2]*base[i])%mod[i];
            else
            ara[i][len-1]=0;
            ara[i][len-1]+=(c);
            if(ara[i][len-1]>=mod[i])
            ara[i][len-1]-=mod[i];
            Hash[i]=ara[i][len-1];
        }
        str[len-1]=c;
        str[len]=0;
    }
    inline bool isEqual (const HashData &b)
    {
        for(int i=0;i<totalHash;i++)
        {
            if(Hash[i]!=b.Hash[i])
            return false;
        }
        return true;
    }
    inline void update (int idx,char c)
    {for(int i=0;i<totalHash;i++)
        {

Hash[i]-=(power[i][len-idx-1]*str[idx])%mod[i
];
            if(Hash[i]<0)
            Hash[i]+=mod[i];

Hash[i]+=(power[i][len-idx-1]*c)%mod[i];
            if(Hash[i]>=mod[i])
            Hash[i]-=mod[i];
        }
        str[idx]=c;
    }
};
```

### Longest Common Substring(1 ≤ N ≤100000):

```
void solve()
{

    long long n,k;
    cin>>n;
    string s,r;
    cin>>s;
    cin>>r;
    HashData hd1,hd2;
    hd1.init(s);
    hd2.init(r);

long long l=0,h=n,mid,ans=-1,mx=-1,ans1=-1;
    while(l<=h)
    {
        mid=(l+h)/2;
        set<pair<long long,long long> > st;
        long long f=0;
        for(int i=0; i<s.size(); i++)
        {
            long long L=i,R=i+mid-1;
            if(i+mid-1>=s.size())
                break;
            st.insert(hd1.query(L,R));
        }
        for(int i=0; i<r.size(); i++)
        {
            long long L=i,R=i+mid-1;
            if(i+mid-1>=s.size())
                break;
            if(st.count(hd2.query(L,R)))
            {
                long long x=R-L+1;
                if(mx<x)
                {
                    mx=x;
                    ans=i;
                    ans1=R;
                }
                f=1;
            }
        }
        if(f)
            l=mid+1;
        else
            h=mid-1;
    }
    for(int i=ans; i<=ans1; i++)
        cout<<r[i];
    cout<<endl;
}
```

**Ques:The shortest substring that happens only once in the input string. If there are multiple shortest substrings (with the same length), output the one that occurs first.**

```
HashData hd1;
 ll n,k;
string s;
 bool chk(ll x)
 {
     map<pr,ll> mp;
     loop(i,0,n)
     {
         if(i+x-1<n)
         {
            pr p= hd1.query(i,i+x-1);
            mp[p]++;
         }
     }
     for(auto it: mp)
     {
         if(it.ss==1) return 1;
     }
     return 0;
 }

void solve()
{
    cin>>s;
    n=s.size();


    hd1.init(s);

    ll l=1,h=n,mid,ans=1,mx=-1,ans1=-1;
    while(l<=h)
    {
        mid=(l+h)/2;

        if(chk(mid))
        {
            ans=mid;
            h=mid-1;
        }
        else
        l=mid+1;
    }
ll x=ans;
     map<pr,ll> mp;
     loop(i,0,n)
     {
         if(i+x-1<n)
         {
            pr p= hd1.query(i,i+x-1);
            mp[p]++;
         }
     }
      loop(i,0,n)
     {
         if(i+x-1<n)
         {

            pr p= hd1.query(i,i+x-1);
            if(mp[p]==1)
            {
               for(int j=i;j<=i+x-1;j++)
               cout<<s[j];
               cout<<endl;
               break;
            }

         }
     }
}
```

**KMP:**

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

**Trie:**

```
struct Node
{
    int count;
    Node *children[26];
};

Node *root;

Node *createNode()
{
    Node *n = (Node*)malloc(sizeof(Node));
    //Node *n=new Node;
    n->count = 0;
    for(int i=0; i<26; i++)
        n->children[i]=NULL;
    return n;
}

void createEdge(Node *u, Node *v, char c)
{
    int i = (int)c - 65;
    u->children[i]=v;
}

void init()
{
    root = createNode();
}

void insert(string s)
{
    Node *u = root;
    int len = s.size();

    for(int i=0; i<len; i++)
    {
        char c = (int)s[i];
        int relPos = (int)c - 65;
        Node *v = createNode();
        if(u->children[relPos]==NULL)
            createEdge(u, v, c);
        u = u->children[relPos];
    }

    u->count++;
}
void del(Node* node)
{
    for(int i=0; i<5; i++)
    {
        if(node->ch[i]!=NULL)
            del(node->ch[i]);
    }
    delete(node);
}
```

**DNA Prefix:** Given a set of **n** DNA samples, where each sample is a string containing characters from **{A, C, G, T}**, we are trying to find a subset of samples in the set, where the length of the longest common prefix multiplied by the number of samples in that subset is maximum.

To be specific, let the samples be:

ACGT
ACGTGCGT
ACCGTGC
ACGCCGT

If we take the subset **{ACGT}** then the result is **4 (4 * 1)**, if we take **{ACGT, ACGTGCGT, ACGCCGT}** then the result is **3 * 3 = 9** (since ACG is the common prefix), if we take **{ACGT, ACGTGCGT, ACCGTGC, ACGCCGT}** then the result is **2 * 4 = 8**.

Now your task is to report the maximum result we can get from the samples.

```
struct Node
{
  int count;
  bool Eow;
  Node * ch[5];
};
Node *root=NULL;
Node* getNode()
{
    Node* n=new Node;
    for(int i=0;i<5;i++)
    n->ch[i]=NULL;
    n->Eow=false;
    n->count=0;
    return n;
}
int getpos(char c)
{
  if(c=='A') return 0;
  else if(c=='C') return 1;
  else if(c=='G') return 2;
  else return 3;
}
void insert(string x)
{
    Node * temp=root;
    for(int i=0;i<x.size();i++)
    {
      int index=getpos(x[i]);
      //dbg1(index);
        if(temp->ch[index]==NULL)
        {
            temp->ch[index]=getNode();

        }

         temp=temp->ch[index];
         temp->count++;


    }
    temp->Eow=true;



}

int search(Node *n , int length )
{
    ll ans=0;
    loop(i,0,5)
    {
      if(n->ch[i]!=NULL)
      {
        //dbg2(n->ch[i]->count*length,i)

ans=max(ans,n->ch[i]->count*length);

ans=max(ans,search(n->ch[i],length+1));
      }
    }
    return ans;


}
void del(Node* node)
{
    for(int i=0; i<5; i++)
    {
        if(node->ch[i]!=NULL)
            del(node->ch[i]);
    }
    delete(node);
}
void solve()
{
  root=getNode();
  ll n,k;
  cin>>n;
  ll mx=-1;
  loop(i,0,n)
  {
  string s;
  cin>>s;
  insert(s);
  mx=max(mx,(ll)s.size());
  }
  ll ans=search(root,1);
  ans=max(ans,mx);
  cout<<ans<<endl;
  del(root);


}
```

**Bitwise Trie:** **Given an array of positive integers you have to print the number of subarrays whose XOR is less than** K. **Subarrays are defined as a sequence of continuous elements** $A_i$, $A_{i+1}$, ..., $A_j$ . **XOR of a subarray is defined as** $A_i$ ^ $A_{i+1}$ ^ ... ^ $A_j$. **Symbol ^ is Exclusive Or.**

```
#define ll long long int

struct Node{
    ll lC,rC;
    Node *right, *left;
};

Node *getNode(){
    Node *temp = (Node*)malloc(sizeof(Node));
    temp->lC = temp->rC = 0;
    temp->left = temp->right = NULL;
    return temp;
}

Node* insert(Node *root, ll n, ll level){
    if(level == -1){return root;}

    if(n&(1<<level)){
        root->rC += 1;
        if(!root->right){
            root->right = getNode();
        }
        root->right = insert(root->right, n,
level-1);
    }
    else{
        root->lC += 1;
        if(!root->left){
            root->left = getNode();
        }
        root->left = insert(root->left, n,
level-1);
    }

    return root;
}
```

```
ll query(Node *root, ll n, ll level, ll k){
    if(!root || level == -1){return 0;}

    bool p = (n&(1<<level));
    bool q = (k&(1<<level));

    if(q){
        if(!p){return
root->lC+query(root->right, n, level-1, k);}
        else{return
root->rC+query(root->left, n, level-1, k);}
    }
    else{
        if(!p){return
query(root->left,n,level-1,k);}
        else{return
query(root->right,n,level-1,k);}
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);cout.tie(NULL);

    ll t, n, k, temp, ans, x;
    cin>>t;
    while(t--){
        ans = temp = x = 0;
        cin>>n>>k;
        Node *root = getNode();
        insert(root, x, 20);
        for(ll i = 0; i < n; i++){
            cin>>temp;
            x ^= temp;
            ans += query(root, x, 20, k);
            insert(root, x, 20);
        }
        cout<<ans<<endl;
    }
      return 0;
}
```

# Dynamic Programming (DP)

## 0-1 Knapsack:

**Recursive:**

```
int knapsack (int wt[],int val[],int w,int n)
{
    if(n==0 || w==0)
        return 0;
    if(t[n][w]!=-1)
        return t[n][w];
    if(wt[n-1]<=w)
        return
t[n][w]=max(val[n-1]+knapsack(wt,val,w-wt[n-1
],n-1),knapsack(wt,val,w,n-1));
    else
        rturn t[n][w]=knapsack(wt,val,w,n-1);
}
```

**Iterative:**

```
for(int i=1; i<n+1; i++)
    for(int j=1; j<w+1; j++)
    {
        if(wt[i-1]<=j)

t[i][j]=max(val[i-1]+t[i-1][j-wt[i-1]],t[i-1]
[j]);
        else
            t[i][j]=t[i-1][j];
    }
```

Longest Common Subsequence (LCS):

**Recursive:**

```
int LCS(string x,string y,int m,int n)
{
    if(n==0 || m==0)
        return 0;
    if(x[m-1]==y[n-1])
        return 1+LCS(x,y,m-1,n-1);
    else
        return max
(LCS(x,y,m,n-1),LCS(x,y,m-1,n));
}
```

**Iterative:**

```
int t[m+1][n+1];
for(int i=0; i<m+1; i++)
    for(int j=0; j<n+1; j++)
        if(i==0 || j==0)
            t[i][j]=0;
for(int i=1; i<m+1; i++)
    for(int j=1; j<n+1; j++)
    {
        if(x[i-1]==y[j-1])
            t[i][j]= 1+t[i-1,j-1];
        else
            return max (t[i,j-1],t[i-1,j]);
    }
```

## From Recusive DP to Iterative DP:

```
Int DP(int i, int j, int k)
{
      if(k == 0)
            Return edge_weight[i][j];
      Int &ret = dp[i][j][k];
      if(vis[i][j][k] == 1) return ret;

      //Ret = DP(i,j,k-1); // where k is not
used as an intermediate node.
      Ret = min(DP(i,j,k-1), DP(i,k,k-1) +
DP(k, j, k-1));

      Return ret;
}
```

1st and 2nd index sometimes increases, sometimes decreases while calling, But 3rd index always decreases while calling.

**Which index value is fixed (always decreasing or always increasing) we have to put that index most outer loop with appropriate order (from 0 to n or n to 0 or something else)**

**Right:**

```
for(int k = 0; k<=n; k++)
{
      for(int i = n; i>=1; i--)
      {
            for(int j = 1; j<=n; j++)
            {
                  if(k == 0)
                        DP[i][j][k] =
edge_weight[i][j];
                  Else
DP[i][j][k] = min(DP[i][j][k-1],
DP[i][k][k-1] + DP[k][j][k-1]);
            }
}
}
```

**Wrong:**

```
for(int i = 1; i<=m; i++)
{
      for(int j = 1; j<=n; j++)
      {
            for(int k  = 1; k<=n; k++)
                  DP[i][j][k] =
min(DP[i][j][k-1], DP[i][k][k-1] +
DP[k][j][k-1]);
/// i = 10, j = 20, k = 50, n = 100
// DP[i][j][k] uses DP[i][k][k-1]
// DP[10][20][50] uses DP[10][50][49]
}
}
```

## Path Printing:

**There are *n* booking requests received by now. Each request is characterized by two numbers: $c_i$ and $p_i$ — the size of the group of visitors who will come via this request and the total sum of money they will spend in the restaurant, correspondingly.**

**We know that for each request, all $c_i$ people want to sit at the same table and are going to spend the whole evening in the restaurant, from the opening moment at 18:00 to the closing moment.**

**Unfortunately, there only are *k* tables in the restaurant. For each table, we know $r_i$ — the maximum number of people who can sit at it. A table can have only people from the same group sitting at it. If you cannot find a large enough table for the whole group, then all visitors leave and naturally, pay nothing.**

**Your task is: given the tables and the requests, decide which requests to accept and which requests to decline so that the money paid by the happy and full visitors was maximum.**

```
ll n,k;
vector<pair<pair<ll,ll>,ll>> a;
```

```
vector<pr> v,res,b;

ll dp[1001][1001],path[1001][1001];
ll uttor=0;
ll rec(ll idx1,ll idx2  )
{
    if(idx2>=k || idx1>=n)
    {
        return 0;
    };
    if(dp[idx1][idx2]!=-1)
        return  dp[idx1][idx2];

    ll ans=INT32_MIN/100;
    if(a[idx1].ff.ff<=b[idx2].ff)
    {

        ll
x=rec(idx1+1,idx2+1)+a[idx1].ff.ss;
        if(x>ans)
        {
            ans=x;
            path[idx1][idx2]=0;
        }


    }
    ll x=rec(idx1+1,idx2);
    if(x>ans)
    {
        ans=x;
        path[idx1][idx2]=1;
    }

    x=rec(idx1,idx2+1);
    if(x>ans)
    {
        ans=x;
        path[idx1][idx2]=2;
    }

    return dp[idx1][idx2]=ans;
}

void solve()
{
    cin>>n;
    a.resize(n);
    memset(dp,-1,sizeof(dp));
    loop(i,0,n)
    {
        cin>>a[i].ff.ff>>a[i].ff.ss;
        a[i].ss=i+1;
    }
    cin>>k;
    b.resize(k);
    loop(i,0,k)
    {
        cin>>b[i].ff;
        b[i].ss=i+1;
    }
    sort(all(a));
    sort(all(b));
    // dbg1(a[0].ff.ff)

    ll ans=rec(0,0);
    //dbg1(ans);
    ll i=0,j=0;
    while (i<n  && j<k)
    {
        if(path[i][j]==0)
        {
            v.push_back({a[i].ss,b[j].ss});
            i++,j++;
        }
        else if(path[i][j]==1)
        {
            i++;
        }
        else
            j++;
    }
    // sort(all(v));
    cout<<v.size()<<" "<<ans<<endl;
    for(auto it : v)
    {
        cout<<it.ff<<" "<<it.ss<<endl;
    }
}
```

## Tree DP

**Carry_non-rooted: Given a non-rooted tree, for each node find the sum of distances of every other node from it.**

```
const int N = 1e5 + 5;
int n;
vector<int> e[N];
int sub[N], dp1[N], dp2[N];

void dfs1(int at, int parent) {
  dp1[at] = 0;
  sub[at] = 1;
  for (int child : e[at]) {
    if (child != parent) {
      dfs1(child, at);
      sub[at] += sub[child];
      dp1[at] += dp1[child] + sub[child];
    }
  }
}

void dfs2(int at, int parent, int carry) {
  dp2[at] = dp1[at] + carry;
  for (int child : e[at]) {
    if (child != parent) {
      int parent_dp = dp2[at];
      parent_dp -= dp1[child] + sub[child];
      int parent_sub = (n - sub[child]);
      int new_carry = parent_dp + parent_sub;
      dfs2(child, at, new_carry);
    }
  }
}

int main() {
  cin >> n;
  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    u--, v--;
    e[u].push_back(v);
    e[v].push_back(u);
  }
  dfs1(0, -1);
  dfs2(0, -1, 0);
  for (int i = 0; i < n; i++) {
    cout << i + 1 << " " << dp2[i] << "\n";
  }
}
```

**Merge: Given a non-rooted tree and node weights, find the sum of path weights over all possible paths.**

```
const int N = 1e5 + 5;

int n;
vector<int> e[N];
int a[N];
int sub[N], dp[N];

void dfs(int at, int parent) {
  dp[at] = 1;
  sub[at] = 1;

  vector<int> children_subs;

  for (int child : e[at]) {
    if (child != parent) {
      dfs(child, at);
      sub[at] += sub[child];
      children_subs.push_back(sub[child]);
    }
  }

  int parent_sub = n - sub[at];
  children_subs.push_back(parent_sub);

  int prefix = 1;
  for (int child_sub : children_subs) {
    dp[at] += prefix * child_sub;
    prefix += child_sub;
  }
}

int main() {
  cin >> n;

  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    u--, v--;
    e[u].push_back(v);
    e[v].push_back(u);
  }

  dfs(0, -1);

  int ans = 0;
  for (int i = 0; i < n; i++) {
    ans += a[i] * dp[i];
  }

  cout << ans << "\n";
}
```

## Game Theory

**Nim Game:**

The current player has a winning strategy if and only if the xor-sum of the pile sizes is non-zero.

**Miser Nim:**

-Last player to remove stones loses.
-Winning state if xor-sum of pile sizes is non-zero.
-Exception: Each pile has one stone only.
-Winning strategy: If there is only one pile of size greater than one,take all or all but one from that pile leaving an odd number one-size piles. Otherwise, same as normal nim.

**Grundy's Number (Partitioning Game):**

**Alice and Bob are playing a strange game. The rules of the game are:**
**1.Initially there are n piles.**
**2.A pile is formed by some cells.**
**3.Alice starts the game and they alternate turns.**
**4.In each turn, a player can pick any pile and divide it into two unequal piles.**
**5.If a player cannot do so, he/she loses the game.**

```
ll const N=1000006;
#define mxp 1000007

using namespace std;
bool t[N];
ll mex(const vector<ll> &grd)
{
    for(auto it : grd)
    {
        t[it]=true;
    }
    ll res=0;
    while(t[res]) res++;
    for(auto it : grd)
```

```
        {
                t[it]=false;
        }
        return res;

}
ll dp[N];
ll g(ll n)
{
        if(n==0) return 0;
        ll &ret=dp[n];
        if(ret!=-1) return ret;
        vector<ll> grd;
        for(int i=1;i<n-i;i++)
        {
                ll x=g(i)^g(n-i);
                grd.push_back(x);
        }
        ll ans=mex(grd);
        return ret=ans;
}
void solve()
{
 ll n;

 cin>>n;
 ll ans=0;
 loop(i,0,n)
 {
         ll x;
         cin>>x;
         ans^=g(x);
 }
 if(ans)
 {
         cout<<"Alice"<<endl;
 }
 else
 cout<<"Bob"<<endl;
}
signed main()
{
   FAST;
         ll t=1,u=0;
         cin>>t;
    memset(dp,-1,sizeof(dp));

         while(t--)
         {
           cout<<"Case "<<++u<<": ";
             solve();
         }
}
```

## Again Stone Game:

**Alice and Bob are playing a stone game. Initially there are n piles of stones and each pile contains some stone. Alice stars the game and they alternate moves. In each move, a player has to select any pile and should remove at least one and no more than half stones from that pile. So, for example if a pile contains 10 stones, then a player can take at least 1 and at most 5 stones from that pile. If a pile contains 7 stones; at most 3 stones from that pile can be removed.**

```
bool t[N];
ll mex(const vector<ll> &grd)
{
        for(auto it : grd)
        {
                t[it]=true;
        }
        ll res=0;
        while(t[res]) res++;
        for(auto it : grd)
        {
                t[it]=false;
        }
        return res;

}
ll dp[N];
ll g(ll n)
{
        if(n<=1) return 0;
        ll &ret=dp[n];
       if(ret!=-1) return ret;
        vector<ll> grd;

        for(int i=1;i<=n/2;i++)
        {
                ll x=g(n-i);
               // dbg3(i,n-i,x);
                grd.push_back(x);

        }
        ll ans=mex(grd);
        return ret=ans;
}
ll get_g(ll n)
{
        if(n<2) return 0;
        if(n%2==0) return n/2;
        return get_g(n/2);
}
void solve()
{
 ll n;

 cin>>n;
 ll ans=0;
 loop(i,0,n)
 {
         ll x;
         cin>>x;
        ll p=get_g(x);
        ans^=p;
       // dbg1(p)
 }
 if(ans)
 {
         cout<<"Alice"<<endl;
 }
 else
 cout<<"Bob"<<endl;
}
```

## Bit Manipulation

### Counting Number of Set Bits:

```
int cnt=0;
while(n>0)
{
      cnt++;
      n=n&(n-1);
}
```

### Find XOR from 1 to N:

```
int computeXOR(int n)
{
  if (n % 4 == 0)
    return n;
  if (n % 4 == 1)
    return 1;
  if (n % 4 == 2)
    return n + 1;
  return 0;
}
```

### Given x,y. Find whether x is a submask (subset) of y:

If there is 1 in the bit representation of x, there is 1 same position of y.

**Checking**: if(x&y==x) return true;

### Given any number n, iterate through all the submasks of n. (decreasing order)

```
for(int x=n; x>0 ; x=(x-1)&x)
{     cout<<x<<endl;
}
      cout<<0<<endl;
```

**Time Complexity**: O(2^ set_bits)

### Given any set of n elements, iterate through all the subsets (submasks) of all the subsets of size n.

```
for(int i=0; i<(1<<n); i++)
{
     for(int x=i; x>0 ; x=(x-1)&x)
     {
        cout<<x<<endl;
     }
     cout<<0<<endl;

}
```

**Time Complexity**: O(2^n)

## Combinatorics

### Stars and Bars:

**X1+x2+x3……xr=n; where xi>=0**

Ans: n+r-1Cr-1

**If(xi>0)**

z1=1+y1,x2=1+y2,x3=1+y3………

So,   n-1Cr-1

**If(xi>=b)**

x1=b+y1,x2=b+y2,x3=b+y3…….

So, n-r*b+r-1Cr-1

### Catalan Number:

**How many ways are there to arrange n open brackets and n close brackets to form a balanced bracket sequence?**

**Recusive solution**:long long ans=0;

```
for(int k=1; k<=n; k++)
       Cn= Ck-1 * Cn-k
```

**2nd Approach**:

Ans= (1/n+1) * (2nCn)

**Count the number of ways to partition n labelled objects in K(possible empty) labelled sets. / Put n different balls in K different boxes.**

Ans: k^n

**What if objects are unlabeled, but sets are labelled? / Put n identical balls in K different boxes.**

Ans: start and bars theorem: n+k-1Ck-1

**What if objects are labeled, but sets are unlabeled? / Put n different balls in k identical boxes.**

**Recusive solution**:long long ans=0;

```
for(int i=1; i<=k; i++)
       solve(n,i);
solve(n,k)=solve(n-1,k)*k+solve(n-1,k-1)
```

### Derangement:

**There are n persons in a room and each of them has a hat. How many ways can the persons wear a hat so that none of them wears his own hat.**

Ans: Dp(n)=Dp(n-1)*(n-1)+Dp(n-2)*(n-1)

**Given N distinct integers from 1 to N, find the number of ways the N integers can be rearranged in M empty slots such that, no integer matches with its slot index. Slots-> 1 to M [M>=N]**

Ans:
Dp(n)=Dp(n-1)*(n-1)+Dp(n-2)*(n-1)+(m-n)*dp(n-1)

**What if(M<N)**

Ans: same problem if we swap M,N.

### Exclusion DP:

**Given an array of numbers, find out number of coprime pairs. Consider pairs are ordered, so (a,b) and (b,a) are different pairs.**

```
int f[n]; // Number of tuples with GCD as multiple of i
int g[n]; // Number of tuples with GCD as i.
for(int i=maxn; i>=1; i--)
{
    g[i]=f[i];
    for(int j=2*i; j<=maxn; j+=i)
    {
        g[i]-=g[j];
    }
}
cout<<g[1]<<endl;
```

**Same problem but pairs are unordered. So (a,b), (b,a) are same.**

1. Repeative: **n=d(i)=number of multiple of (i)** // See it next ques.
   Ans: nC2+ n = (n*(n+1))/2
2. Non-repeative : Ans: nc2= (n*(n-1))/2

**Same problem, but instead of pairs find k-tuples (i1,i2,i3….ik) with gcd(a_i1,a_i2..a_i3)=1**

1. If ordered : f[i]=d(i)^k
2. If unoredered,
   - Repetive: f[i]=(d(i)+ k-1)Ck
   - Non-repetive : f[i]=d(i)Ck

**Given an array of numbers,find the subset of array gcd 1 (find the number of its coprime subsequences) Note that two subsequences are considered different if chosen indices are different. For example, in the array [1,1] there are 3 different subsequences: [1], [1] and [1,1].**

```
void solve()
{
  ll n;
  cin>>n;
  loop(i,0,n)
  {
    ll x;
    cin>>x;
    cnt[x]++;
  }
  for(ll i=1;i<=mx;i++)
  {
    ll c=0;// d(i)
    for(ll j=i;j<=mx;j+=i)
    {
      c+=cnt[j];
    }
    g[i]=binpow(2,c,MOD)-1;
    g[i]%=MOD;

  }
  for(ll i=mx;i>=1;i--)
  {
    f[i]=g[i];
      for(ll j=2*i;j<=mx;j+=i)
      {
        f[i]-=f[j];
        f[i]%=MOD;
        f[i]+=MOD;
        f[i]%=MOD;

      }
  }
  cout<<f[1]<<endl;

}
```

**Given an array of n numbers, find out sum of all pairs gcd.**

Find **Given an array of numbers, find out number of coprime pairs. Consider pairs are ordered, so (a,b) and (b,a) are different pairs.**

Then, for( i=0 to maxn) g[i]*i;

**Given a list of *n* positive integers, your task is to count the number of pairs of integers that are coprime**

```
long long const N=1000006;
long long f[N],g[N];
long long mp[N]= {0};
void solve()
{
    long long n,a,b;
    cin>>n;
    vector<long long> v(n);
    long long mx=-1;

    for(int i=0; i<n; i++)
    {
        cin>>v[i];
        mx=max(mx,v[i]);
        mp[v[i]]++;
    }
    for(int i=1; i<=mx; i++)
    {
        long long cnt=0;
        for(int j=i; j<=mx; j+=i)
        {

            cnt+=mp[j];
        }
        f[i]=(cnt*(cnt-1))/2;
    }
    for(int i=mx; i>=1; i--)
    {
        g[i]=f[i];
        for(int j=2*i; j<=mx; j+=i)
        {
            g[i]-=g[j];
        }

    }
    cout<<g[1]<<endl;
}
```

**nCr:**

```
ll power(ll x, int y, int p)
{
    ll res = 1;
    x = x % p;
    while (y > 0)
    {
        if (y & 1)
            res = (res * x) % p;
        y = y >> 1;
        x = (x * x) % p;

    }
    return res;
}

ll modInverse(ll n, int p)
{
    return power(n, p - 2, p);
}
ll fact[2*N+10];
ll ncr(ll n , ll r, ll M)
{
  if(r==0) return 1;
    ll ans=fact[n];
    ans*=modInverse(fact[r],MOD);
    ans%=M;
     ans*=modInverse(fact[n-r],M);
    ans%=M;
    return ans;
}
void inti()
{
  fact[0]=1;
  for(ll i=1;i<2*N;i++)
  {
    fact[i]=fact[i-1]*i;
    fact[i]%=MOD;
  }

}
```

## Extra:

### Ordered Set:

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>,
rb_tree_tag,
tree_order_statistics_node_update> pbds; //
find_by_order, order_of_key
// finding kth element - 4th query
*A.find_by_order(0)--- index 0 er value
// finding number of elements smaller than X
A.order_of_key(6) --- 6 er smaller kotogulo
elements
```

### Multiply Large Numbers represented as Strings

```
#include<bits/stdc++.h>
using namespace std;
// Multiplies str1 and str2, and prints
result.
string multiply(string num1, string num2)
{
    int len1 = num1.size();
    int len2 = num2.size();
    if (len1 == 0 || len2 == 0)
    return "0";

    // will keep the result number in vector
    // in reverse order
    vector<int> result(len1 + len2, 0);

    // Below two indexes are used to find
positions
    // in result.
    int i_n1 = 0;
    int i_n2 = 0;

    // Go from right to left in num1
    for (int i=len1-1; i>=0; i--)
    {
        int carry = 0;
        int n1 = num1[i] - '0';

        // To shift position to left after
every
        // multiplication of a digit in num2
        i_n2 = 0;

        // Go from right to left in
num2
        for (int j=len2-1; j>=0; j--)
        {
            // Take current digit of second
number
            int n2 = num2[j] - '0';

            // Multiply with current digit of
first number
            // and add result to previously
stored result
            // at current position.
            int sum = n1*n2 + result[i_n1 +
i_n2] + carry;

            // Carry for next iteration
            carry = sum/10;

            // Store result
            result[i_n1 + i_n2] = sum % 10;

            i_n2++;
        }

        // store carry in next cell
        if (carry > 0)
```

```
            result[i_n1 + i_n2] += carry;

        // To shift position to left after
every
        // multiplication of a digit in num1.
        i_n1++;
    }

    // ignore '0's from the right
    int i = result.size() - 1;
    while (i>=0 && result[i] == 0)
    i--;

    // If all were '0's - means either both
or
    // one of num1 or num2 were '0'
    if (i == -1)
    return "0";

    // generate the result string
    string s = "";


    while (i >= 0)
        s += std::to_string(result[i--]);

    return s;
}

// Driver code
int main()
{
    string str1 =
"1235421415454545454545454544";
    string str2 =
"1714546546546545454544548544544545";


    if((str1.at(0) == '-' || str2.at(0) ==
'-') &&
        (str1.at(0) != '-' || str2.at(0) !=
'-' ))
        cout<<"-";


    if(str1.at(0) == '-')
        str1 = str1.substr(1);

    if(str2.at(0) == '-')
        str2 = str2.substr(1);

    cout << multiply(str1, str2);
    return 0;
}
```

**Or:**

```
/ Include header file
#include <bits/stdc++.h>
using namespace std;


int main()
{
    string num1 =
"1235421415454545454545454544";
    string tempnum1 = num1;
    string num2 =
"1714546546546545454544548544544545";
    string tempnum2 = num2;
    // Check condition if one string is
negative
    if (num1[0] == '-' && num2[0] != '-') {
        num1 = num1.substr(1);
    }
    else if (num1[0] != '-' && num2[0] == '-')
{
        num2 = num2.substr(1);
    }
    else if (num1[0] == '-' && num2[0] == '-')
{
        num1 = num1.substr(1);
        num2 = num2.substr(1);
    }
    string s1 = num1;
    string s2 = num2;
    reverse(s1.begin(), s1.end());
    reverse(s2.begin(), s2.end());

    vector<int> m(s1.length() + s2.length());
    // Go from right to left in num1
    for (int i = 0; i < s1.length(); i++) {
        // Go from right to left in num2
        for (int j = 0; j < s2.length(); j++)
{
            m[i + j]
                = m[i + j] + (s1[i] - '0') *
(s2[j] - '0');
        }
    }
    string product = "";
    // Multiply with current digit of first
number
    // and add result to previously stored
product
    // at current position.
    for (int i = 0; i < m.size(); i++) {
        int digit = m[i] % 10;
        int carry = m[i] / 10;
        if (i + 1 < m.size()) {
            m[i + 1] = m[i + 1] + carry;
        }
        product = to_string(digit) + product;
    }
    // ignore '0's from the right
    while (product.length() > 1 && product[0]
== '0') {
        product = product.substr(1);
    }
    // Check condition if one string is
negative
    if (tempnum1[0] == '-' && tempnum2[0] !=
'-') {
        product = "-" + product;
    }
    else if (tempnum1[0] != '-' && tempnum2[0]
== '-') {
        product = "-" + product;
    }

    cout << "Product of the two numbers is :"
         << "\n"
         << product << endl;
}
```

## Sum of two large numbers:

```
#include<bits/stdc++.h>
using namespace std;

// Function for finding sum of larger numbers
string findSum(string str1, string str2)
{
    // Before proceeding further, make sure
length
    // of str2 is larger.
    if (str1.length() > str2.length())
        swap(str1, str2);

    // Take an empty string for storing
result
    string str = "";

    // Calculate length of both string
    int n1 = str1.length(), n2 =
str2.length();
    int diff = n2 - n1;

    // Initially take carry zero
    int carry = 0;

    // Traverse from end of both strings
    for (int i=n1-1; i>=0; i--)
    {
        // Do school mathematics, compute sum
of
        // current digits and carry
        int sum = ((str1[i]-'0') +
                   (str2[i+diff]-'0') +
                   carry);
        str.push_back(sum%10 + '0');
        carry = sum/10;
    }

    // Add remaining digits of str2[]
    for (int i=n2-n1-1; i>=0; i--)
    {
        int sum = ((str2[i]-'0')+carry);
        str.push_back(sum%10 + '0');
        carry = sum/10;
    }

    // Add remaining carry
    if (carry)
        str.push_back(carry+'0');

    // reverse resultant string
    reverse(str.begin(), str.end());

    return str;
}

// Driver code
int main()
{
    string str1 = "12";
    string str2 = "198111";
    cout << findSum(str1, str2);
    return 0;
}
```

## Divide large number represented as string:

```
#include <bits/stdc++.h>
using namespace std;

// A function to perform division of large
numbers
string longDivision(string number, int
divisor)
{
    // As result can be very large store it
in string
    string ans;

    // Find prefix of number that is larger
    // than divisor.
    int idx = 0;
    int temp = number[idx] - '0';
    while (temp < divisor)
        temp = temp * 10 + (number[++idx] -
'0');

    // Repeatedly divide divisor with temp.
After
    // every division, update temp to include
one
    // more digit.
    while (number.size() > idx) {
        // Store result in answer i.e. temp /
divisor
        ans += (temp / divisor) + '0';

        // Take next digit of number
        temp = (temp % divisor) * 10 +
number[++idx] - '0';
    }

    // If divisor is greater than number
    if (ans.length() == 0)
        return "0";

    // else return ans
    return ans;
}

// Driver program to test longDivision()
int main()
{
    string number = "1248163264128256512";
    int divisor = 125;
    cout << longDivision(number, divisor);
    return 0;
}
```

**Subtraction of two large numbers:**

```
#include <bits/stdc++.h>
using namespace std;

// Returns true if str1 is smaller than str2,
// else false.
bool isSmaller(string str1, string str2)
{
    // Calculate lengths of both string
    int n1 = str1.length(), n2 =
str2.length();

    if (n1 < n2)
        return true;
    if (n2 < n1)
        return false;

    for (int i = 0; i < n1; i++) {
        if (str1[i] < str2[i])
            return true;
        else if (str1[i] > str2[i])
            return false;
    }
    return false;
}

// Function for finding difference of larger
numbers
string findDiff(string str1, string str2)
{
    // Before proceeding further, make sure
str1
    // is not smaller
    if (isSmaller(str1, str2))
        swap(str1, str2);

    // Take an empty string for storing
result
    string str = "";

    // Calculate lengths of both string
    int n1 = str1.length(), n2 =
str2.length();
    int diff = n1 - n2;

    // Initially take carry zero
    int carry = 0;

    // Traverse from end of both strings
    for (int i = n2 - 1; i >= 0; i--) {
        // Do school mathematics, compute
difference of
        // current digits and carry
        int sub = ((str1[i + diff] - '0') -
(str2[i] - '0')
                   - carry);
        if (sub < 0) {
            sub = sub + 10;
            carry = 1;
        }
        else
            carry = 0;

        str.push_back(sub + '0');
    }

    // subtract remaining digits of str1[]
    for (int i = n1 - n2 - 1; i >= 0; i--) {
        if (str1[i] == '0' && carry) {
            str.push_back('9');
            continue;
        }
        int sub = ((str1[i] - '0') - carry);
        if (i > 0 || sub > 0) // remove
preceding 0's
            str.push_back(sub + '0');
        carry = 0;
    }

    // reverse resultant string
    reverse(str.begin(), str.end());

    return str;
}

// Driver code
int main()
{
    string str1 = "88";
    string str2 = "1079";

    // Function call
    cout << findDiff(str1, str2);
    return 0;
}
```

### Sqrt-root Decomposition:

```
int n;
int a[N];
int block_size;
int block_ans[N];

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }

  block_size = sqrt(n);

  for (int i = 0; i < n; i++) {
    int block_id = i / block_size;
    block_ans[block_id] += a[i];
  }

  int q;
  cin >> q;

  while (q--) {
    int l, r;
    cin >> l >> r;
    l--, r--;

    int ans = 0;

    for (int i = l; i <= r; ) {
      if (i % block_size == 0 and i +
block_size <= r + 1) {
        ans += block_ans[i / block_size];
        i += block_size;
      }
      else {
        ans += a[i];
        i++;
      }
    }

    cout << ans << "\n";
  }
  return 0;
}
```

### Mo’s Algorithm:

```
const int N = 2e5;
int n;
int a[N];
int q;
int block_size;
int range_ans;

struct Query {
  int l, r;
  int ans;
  Query() {}
  Query(int _l, int _r) {
    l = _l, r = _r;
  }
  bool operator<(Query &other) const {
    int curr_block = l / block_size;
    int other_block = other.l / block_size;
    if (curr_block != other_block) {
      return curr_block < other_block;
    }
    else {
      return r < other.r;
    }
  }
} query[N];

void include(int id) {
  range_ans += a[id];
}

void remove(int id) {
  range_ans -= a[id];
}

int get_ans() {
  return range_ans;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }

  block_size = sqrt(n);

  cin >> q;
  for (int i = 0; i < q; i++) {
    int l, r;
    cin >> l >> r;
    l--, r--;
    query[i] = Query(l, r);
  }
  sort(query, query + q);

  int L = 0, R = -1;
  for (int i = 0; i < q; i++) {
    int ql = query[i].l;
    int qr = query[i].r;

    while (R < qr) include(++R);
    while (L > ql) include(--L);
    while (L < ql) remove(L++);
    while (R > qr) remove(R--);

    query[i].ans = get_ans();
  }

  for (int i = 0; i < q; i++) {
    cout << query[i].ans << "\n";
  }

  return 0;
}
```