前置知识：

二项式定理 $(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$

推论式 $\sum_{i=0}^{n} (-1)^i \binom{n}{i} = [n=0]$

组合数公式

通项式 $\binom{n}{m} = \frac{n!}{(n-m)!\,m!}$

递推式 $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$

分离式 $\binom{n}{k}\binom{k}{m} = \binom{n}{m}\binom{n-m}{k-m}, \quad n \geq k \geq m$

$$\begin{aligned} \text{左边} &= \frac{n!}{k!\,(n-k)!}\frac{k!}{m!\,(k-m)!} \\ &= \frac{n!}{m!\,(n-m)!}\frac{(n-m)!}{(k-m)!\,(n-k)!} = \text{右边} \end{aligned}$$

- 对第一个推论式的说明：

  由二项式定理: $(a+b)^n = \sum_{i=0}^{n} C_n^i a^{n-i} b^i$ 当 `a = 1 , b = -1 的时候有`：

  右边等于$\sum_{i=0}^{n} C_n^i (-1)^i = [n=0]$, $[n=0]$ 的意思是当 n = 0 的时候，$\sum_{i=0}^{n} C_n^i (-1)^i = 1$;;; 如果 n != 0 则 $\sum_{i=0}^{n} C_n^i (-1)^i = 0$

- 对第二个推论式的说明：
  - 左边的实际意义：从n个数中选出k个，再从k个数当中选出m个的方案数。
  - 右边的实际含义：从n中选出m个，再从剩余的n - m 个中选出 k - m 个。

二项式反演

二项式定理 $(a+b)^n=\sum_{i=0}^{n}\binom{n}{i}a^{n-i}b^i$

推论式 $\sum_{i=0}^{n}(-1)^i\binom{n}{i}=[n=0]$

组合数公式

通项式 $\binom{n}{m}=\frac{n!}{(n-m)!m!}$

递推式 $\binom{n}{m}=\binom{n-1}{m-1}+\binom{n-1}{m}$

分离式 $\binom{n}{k}\binom{k}{m}=\binom{n}{m}\binom{n-m}{k-m}$，$n\geq k\geq m$

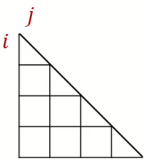
$$\text{左边}=\frac{n!}{k!\,(n-k)!}\frac{k!}{m!\,(k-m)!}=\frac{n!}{m!\,(n-m)!}\frac{(n-m)!}{(k-m)!\,(n-k)!}=\text{右边}$$

二项式反演

$$f(n)=\sum_{i=0}^{n}\binom{n}{i}g(i)\iff g(n)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(i)$$

证：

$$\begin{aligned}&\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}\sum_{j=0}^{i}\binom{i}{j}g(j)\\&=\sum_{j=0}^{n}\sum_{i=j}^{n}(-1)^{n-i}\binom{n}{i}\binom{i}{j}g(j)\\&=\sum_{j=0}^{n}\sum_{i=j}^{n}(-1)^{n-i}\binom{n}{j}\binom{n-j}{i-j}g(j)\\&=\sum_{j=0}^{n}\binom{n}{j}g(j)\sum_{i=j}^{n}(-1)^{n-i}\binom{n-j}{i-j}\\&=\sum_{j=0}^{n}\binom{n}{j}g(j)\sum_{i=0}^{n-j}(-1)^{n-j-i}\binom{n-j}{i}\\&=\sum_{j=0}^{n}\binom{n}{j}g(j)[n=j]=g(n)\end{aligned}$$



二项式反演证明的说明：

设“证：”所在行为第一行：

第一行到第二行：我们将i看成行，j看成列，那么，第一行相当于是按照行枚举，第二行相当于按照列枚举，所以两者等价，

第二行到第三行：根据左边的分离式，我们可以得到第三行。

第三行到第四行：分离变量，将与i无关的变量向左移动，

第四行到第五行：对于第四行中的关于i的求和式，将其中的i用 `i + j` 进行替换，也就是令 `i = i + j` ，得到第五行。

第五行到第六行：根据左边二项式定理的推论得到第六行。

得证。

今天终于学明白了！

二项式反演用于间接求方案数

$g(k)$：恰好 k 个条件满足，方案数

$f(k)$：钦定 k 个条件满足，方案数

如何计算 $f(k)$？

(选定k个条件) × (这k个条件一定成立) × (剩下$n-k$个条件随机)

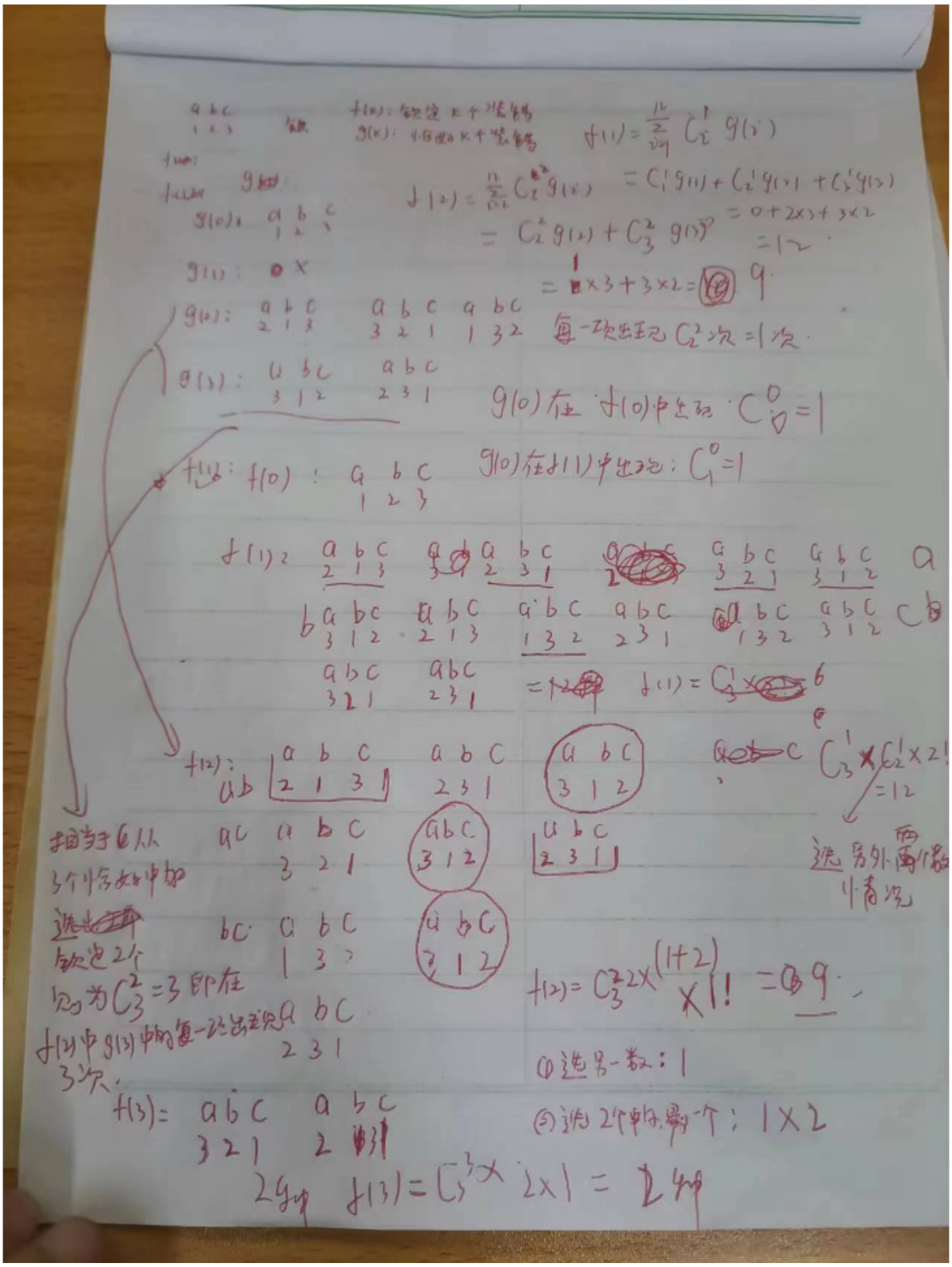
在 $f(k)$ 中，$g(i)$ 中的每个方案都出现了 $\binom{k}{i}$ 次　这里应该是从i个中选出k个

$$f(k)=\sum_{i=k}^{n}\binom{i}{k}g(i)\iff g(k)=\sum_{i=k}^{n}(-1)^{i-k}\binom{i}{k}f(i)$$

- 如何计算f(k)

f(k) 代表的是我们钦定k个满足条件的方案数 ， 设总的有n个，对于其中的某一个钦定 ，当前钦定的k个数据一定满足条件，但另外n - k 个无论是否满足不满足条件 都算一组方案数，也就是说，f(k) = F(n , k) * (n - k)! ，其中F(n , k) 需要我们利用动态规划去求，F(n , k) 代表的是前n组数据，选出k个满足条件的方案数，对于F(n , k) 来说 ， 我们并不关心剩余的 n - k 对，F(n , k) 分组的依据: 任意两组的这k个元素不完全相同 ；如果对于f(k) 中的某一组钦定，F(n , k) 中必定有一个方案与其相同 ， 但这一组钦定所能贡献的方案数是 (n - k) ！ ，综上 f(k) = F(n , k)*(n - k)!

对上图中的一些解释。



- 一点小疑惑：
  - 为什么$f_k = \sum_{i=k}^{n} C_i^k g_i$ 这个等式为什么可以成立？？？

> 因为在f(k) 中g(i) 中的每个方案都出现了C(i , k) 次 .
> 解释：
>
> 对于 f(k) 中g(i) 中的每个方案都出现了C(i , k) 次 的解释：
> f(k) 表示钦定了k个，但其余的n - k 个是随意排列的，也就是说 f(k) 中一定存在方案中满足条件的个数为 k
> 满足条件的个数为 k + 1 ..... 满足条件的个数为 n :
>
> g(i) 表示恰好满足条件为 i 个的方案数 ， 其余的 n - i 个一定不满足。
>
> i = k : 对于g(k) 中的每一个在f(k) 中都出现了 C(k , k) = 1 次
> i = k + 1 : 对于g(k + 1) 中的每一项都在 f(k) 中出现了 C(k , k + 1) 次，
>
> 拿 i = k + 1 这一项进行解释， f(k) 说明我们钦定了k组，在g(k + 1) 这一集合中任意一个方案t，只有 k + 1 个是满足的
> 条件的如果想要钦定元素个数为k个，则需要从这 k + 1 个当中进行选择，这k + 1 个元组成 C(k , k + 1)个 包含k元素个钦定 ，对于当前方案 t ，在C(k , k + 1) 种钦定方式所代表的方案数的集合中，t都会出现一次，所以t在f(k) 中一定出现了C(k , k + 1) 次，进而得出g(k + 1) 中的每一项在f(k) 中都出现了C(k , k + 1) 次 ；
> 进而得出: f(k) 中g(i) 中的每个方案都出现了C(i , k) 次。
> .....
>
> 对于g(i) 中的每一项都在f(k) 中出现了C(k , i) 次 ；

- P4859

> 抽象出来的题意：
> 给定两个长度为n的序列a和序列 b ，令其两两配对，问满足 a[i] > b[i] 的对数 比 a[i] < b[i] 的对数多 k 组的情况下的有多少种，对 1e9 + 9 取模。 0 <= k <= n <= 2000 ，保证两个序列中没有相同的元素。

思路分析：

按照上面的分析来：

- 设g_m表示恰好有m对满足 a > b 的方案数
- 设f_m 表示钦定m对满足 a > b 的方案数

> 设恰好有x对的a > b 则 ， x - (n - x) = k 得到 x = (n + k) / 2 ，我们求得f((n + k) / 2) 即可 。

则 $f_m = \sum_{i=m}^{n} C_i^m g_i$ 根据二项式反演得到 $f_m = \sum_{i=m}^{n} (-1)^{i-m} C_i^m f_i$

$$f_i = F(n, i) * (n - i)!$$

F(n , i) 表示从前 n 项中选择出来 i 组满足 a > b 的方案数,利用动态规划来求解。

AC代码

```
#include <bits/stdc++.h>
using namespace std ;
using LL =  long long ;
const int N = 2010 , mod = 1e9 + 9 ;
int a[N] , b[N] ;
int n , k ;
LL f[N][N] ;  // 这个数组最好开的是long long 防止我们的溢出
LL fact[N] ;  // 这个数组也最好开long long
LL infact[N] ;  // 这个数组也最好开 long long
LL qw(LL a , LL b ) // 快速幂求逆元。
{
    LL res = 1 ;
    while(b)
    {
        if(b & 1LL ) res = res * a % mod ;
        b >>= 1LL ;
        a = a % mod * a % mod ;
    }
    return res ;
}
void init()
{
    fact[0] = infact[0] = 1 ;
    for(int i = 1 ; i <= n ; i ++ )
    {
         // 时刻进行mod防止溢出
        fact[i] = fact[i - 1] * i % mod ;
        infact[i] = infact[i - 1] * qw(i , mod - 2 ) % mod ;
    }
}
LL C(LL m , LL n )
{
    // 时刻进行mod防止溢出
    return fact[n] * infact[m] % mod * infact[n - m] % mod ;
}
int main()
{
    cin >> n >> k ;
    int m = n + k >> 1LL ;
    init() ;
    for(int i = 1 ; i <= n ; i ++ ) cin >> a[i] ;
    for(int i = 1 ; i <= n ; i ++ ) cin >> b[i] ;
    sort(a + 1 , a + 1 + n ) ;
    sort(b + 1 , b + 1 + n ) ;
    // dp初始化
    for(int i = 0 ; i <= n ; i ++ ) f[i][0] = 1 ;
    // dp 预处理
    LL p = 0 ; // p代表的是b数组中比当前a[i] 小的元素的个数 。
    for(int i = 1 ; i <= n ; i ++ )
    {
        while(p + 1 <= n && b[p + 1] < a[i] ) p ++ ;
        for(int j = 1 ; j <= i ; j ++ )
        {
```

```
            f[i][j] = ( f[i - 1][j] + f[i - 1][j - 1] * ( p - j + 1 ) % mod
) % mod  ;
        }
    }
    LL ans = 0 ;
    int sign = -1 ;
    for(int i = m ; i <= n ; i ++ )
    {
        sign = -sign ;
         // 注意这里要时刻记得mod 因为是在mod 的意义下，
        ans = ( ans + sign * C(m , i ) * f[n][i] % mod * fact[n - i] % mod )
% mod ;
    }
    // 并且我们的答案需要减去一些，ans可能为负，所以我们需要 取正 。
    cout << ( ans + mod ) % mod  ;

    return 0 ;
}
```

需要注意的是，当一个int类型和一个long long 类型在一起运算的时候，int 类型会进行一个隐式提升，成为long long 的类型。