**广义容斥原理**

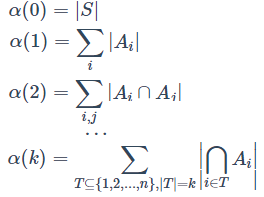
**基础概念**

用语言描述，容斥原理求的是不满足任何性质的方案数，我们通过计算所有至少满足k个性质的方案数之和来计算。

同样的，我们可以通过计算所有至少满足k个性质的方案数之和来计算恰好满足k个性质的方案数。这样的容斥方法我们称之为广义容斥原理。

**容斥方法**

首先，我们设α(k)代表所有至少满足k的性质的方案数之和。

也就是说：

我们发现α(k)将具有p(p≥k)个性质的元素计算了次。

假设β(k)代表恰好具有k个性质的方案数，则有递推公式如下：



组合意义就是我们把多算的那些方案数都减掉就可以了。

但是这样计算我们需要求出每一个β的值，时间复杂度是O(n2))的。

我们有一个更好的计算式：



样就可以O(n)计算单个β的值了，我们现在来证明这个公式的正确性。

我们考虑具有t个的性质的元素被计算了几次：

1. 如果t<k，则该元素一次都没有被计算过。

2. 如果t=k，则该元素恰好被计算了一次。

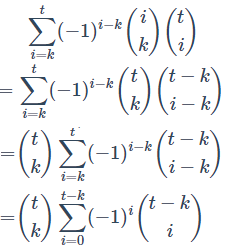
3. 如果t>k，............真复杂

那么现在我们要证明，当t>k时，具有t个性质的元素没有被计算过。

根据上面提到的α中多算的次数，我们容易得知具有t个性质的元素被计算了这么多次：



利用三项式系数恒等式，我们对此进行化简：



根据二项式定理，我们构造：



令x=1，我们得知：



所以，当t>k时，具有t个性质的元素没有被计算过。那么我们就可以愉快地容斥了。

**集合计数**

**Description**

一个有N个元素的集合有2 ^ N个不同子集（包含空集），现在要在这2^N个集合中取出若干集合（至少一个），使得它们的交集的元素个数为K，求取法的方案数，答案模1000000007。（是质数喔~）

**Input Format**

一行两个整数N，K。

**Output Format**

一行为答案。

**Sample Input**

Copy

3 2

**Sample Output**

Copy

6

**解析**

当这些集合具有一个元素的交集的时候，我们就认为这种方案具有一个性质，那么我们要求的就是具有k个性质的元素个数，是广义容斥原理计算的对象。

那么我们现在只需考虑如何计算至少具有k个性质的元素个数即可。

首先，我们只要强制选k个元素，使他们成为交集的一部分，然后剩下的随便选，这样的方案就一定具有k个元素以上的交集。

那么就可以这样计算了：



组合意义：首先我们选k个元素有种方案，剩下的元素可选可不选，可以组成2n−k个子集，每个子集可选可不选，就有种方案了。

直接使用公式计算β(k)的值即可。

Code:

include <bits/stdc++.h>

using namespace std;

const int N = 1e6+20 , Mod = 1e9+7;

int n,k,alpha[N],fac[N],inv[N],Pow[N],ans;

inline int add(int a,int b) { return a + b >= Mod ? a + b - Mod : a + b; }

inline int mul(int a,int b) { return 1LL \* a \* b % Mod; };

inline int sub(int a,int b) { return a - b < 0 ? a - b + Mod : a - b; }

inline void Add(int &a,int b) { a = add( a , b ); }

inline void Mul(int &a,int b) { a = mul( a , b ); }

inline void Sub(int &a,int b) { a = sub( a , b ); }

inline int quickpow(int a,int b) { int res = 1; for (;b;Mul(a,a),b>>=1) if ( 1 & b ) Mul(res,a); return res; }

inline void init(void)

{

fac[0] = inv[0] = Pow[0] = 1;

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

fac[i] = mul( fac[i-1] , i ) , Pow[i] = Pow[i-1] \* 2LL % (Mod-1);

inv[n] = quickpow( fac[n] , Mod-2 );

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

inv[i] = mul( inv[i+1] , i+1 );

}

inline int C(int n,int m) { return mul( fac[n] , mul( inv[m] , inv[n-m] ) ); }

inline void solve(void)

{

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

alpha[i] = mul( C(n,i) , quickpow( 2 , Pow[n-i] ) );

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

if ( ( i - k ) & 1 ) Sub( ans , mul( C(i,k) , alpha[i] ) );

else Add( ans , mul( C(i,k) , alpha[i] ) );

}

int main(void)

{

scanf("%d%d",&n,&k);

init();

solve();

printf("%d\n",ans);

return 0;

}

**已经没有什么好害怕的了**

**Description**

有糖果和药片各n个。糖果i有能量ai，药片i有能量bi。

你需要将药片和糖果两两配对，求有多少种方案满足糖果比药片能量大的组数减去药片比糖果能量大的组数恰好为k。

保证所有的能量两两不同，答案对109+9取模。

**Input Format**

第一行两个整数n，k。

第二行n个整数，表示糖果的能量。

第三行n个整数，表示药片的能量。

**Output Format**

输出一行一个整数，表示方案数。

**Sample Input**

Copy

4 2

5 35 15 45

40 20 10 30

**Sample Output**

Copy

4

**解析**

首先，我们设糖果能量大于药片能量有x组，药片能量大于糖果能量有y组，那么符合题意的方案应该满足：



很容易得知，n+k为奇数的时候无解，反之x=(n+k)/2。

那么我们假设存在一组糖果能量大于药片能量就具有一个性质，原问题就是要求具有x=(n+k)/2个性质的方案数。

套用广义容斥原理的方法，我们需要计算具有至少k个性质的方案数。

我们需要把糖果盒药片先排一下序，然后考虑dp计算方案。设f[i][j]代表前i个糖果，有j个糖果的能量大于药片的方案数，容易写出状态转移方程：

f[i][j]=f[i−1][j]+f[i−1][j−1]×(cnt[i]−(j−1))

其中cnt[i]代表比第i个糖果能量小的药片的数量。

容易得知：α(k)=f[n][k]×(n−k)!，套用广义容斥原理计算β即可。

Code:Code:

Copy

include <bits/stdc++.h>

using namespace std;

const int N = 2020 , Mod = 1e9+9;

int n,k,a[N],b[N],C[N][N],fac[N];

int alpha[N],f[N][N],cnt[N],ans;

inline int add(int a,int b) { return a + b >= Mod ? a + b - Mod : a + b; }

inline int mul(int a,int b) { return 1LL \* a \* b % Mod; };

inline int sub(int a,int b) { return a - b < 0 ? a - b + Mod : a - b; }

inline void Add(int &a,int b) { a = add( a , b ); }

inline void Mul(int &a,int b) { a = mul( a , b ); }

inline void Sub(int &a,int b) { a = sub( a , b ); }

inline int quickpow(int a,int b) { int res = 1; for (;b;Mul(a,a),b>>=1) if ( 1 & b ) Mul(res,a); return res; }

inline void input(void)

{

scanf("%d%d",&n,&k);

for (int i=1;i<=n;i++) scanf("%d",&a[i]);

for (int i=1;i<=n;i++) scanf("%d",&b[i]);

}

inline void init(void)

{

int p = 0; fac[0] = 1;

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

{

fac[i] = mul( fac[i-1] , i );

while ( p < n && b[p+1] < a[i] ) p++;

cnt[i] = p;

}

C[0][0] = C[1][0] = C[1][1] = 1;

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

{

C[i][0] = C[i][i] = 1;

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

C[i][j] = add( C[i-1][j-1] , C[i-1][j] );

}

}

inline void DynamicProgram(void)

{

for (int i=0;i<=n;i++) f[i][0] = 1;

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

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

f[i][j] = add( f[i-1][j] , mul( f[i-1][j-1] , max( cnt[i]-j+1 , 0 ) ) );

}

inline void solve(void)

{

k = ( n + k ) / 2;

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

alpha[i] = mul( f[n][i] , fac[n-i] );

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

if ( ( i - k ) & 1 ) Sub( ans , mul( C[i][k] , alpha[i] ) );

else Add( ans , mul( C[i][k] , alpha[i] ) );

}

int main(void){

input();

if ((n+k) &1 )

return puts("0"),0;

sort( a+1 , a+n+1 );

sort( b+1 , b+n+1 );

init();

DynamicProgram();

solve();

printf("%d\n",ans);

return 0;

}