Derangement Numbers

In combinatorial mathematics, a derangement is a permutation of the elements of a set such that none of the elements appear in their original position.

Derangement number is very important and used in solving different combinatorial problem. It is normally denoted by **!n** (subfactorial n).

Formula:

!n=(n-1)*(!(n-1)+!(n-1))

Base case:- !1=0 and !0=1

Sequence :

1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932……

List of Solved Problems:

1. SPOJ_9097 Derangements HARD
2. Topcoder SRM 176 Division-1 Level-2
3. UVA 10497 - Sweet Child Makes Trouble
4. Lightoj 1095 - Arrange the Numbers

Description of the problems:

SPOJ_9097 Derangements HARD

A derangement of *n* numbers is a permutation of those numbers in which none of the numbers appears in its original place. For example, the numbers {1,2,3} can be deranged into {2,3,1} and {3,1,2}. We can modify this slightly for *n* numbers that are not necessarily distinct by saying that no number in the derangement

can be in the place that a number of the same value was in in the original ordering. So the numbers {1,1,2,2,3} could be deranged into {2,2,1,3,1}, {2,2,3,1,1}, {2,3,1,1,2}, and {3,2,1,1,2}.

Input

First line contains T(1<=T<=100) the number of test cases. Each test case contains two lines. First line contains an integer N(1<=N<=15) denoting total number of elements in the array. Second line contains a space separated list of N integers Ai such that 0<=Ai<N.

Output

For each test case output an integer, the total number of derangements of the array.

Example

Input:

```
2
5
1 1 2 2 3
6
0 0 0 1 1 1
```

Output:

```
4
1
```

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.h>
```

```
using namespace std;
#define print1(a)    cout<<a<<endl
#define print2(a,b) cout<<a<<" "<<b<<endl
#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl
#define SZ(s)      ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)
LL dp[16][1<<16];
int arr[20];
int n;

LL dprec(int in,int mask)
{
    //print2(in,mask);
    LL &ret=dp[in][mask];
    if(ret!=-1) return ret;
    if(in==n) return ret=1;

    ret=0;
    int i;
    for(i=0;i<n;i++)
    if((mask&(1<<i))==0)
    {
        if(arr[i]==arr[in])
            continue;
        ret+=dprec(in+1,mask|(1<<i));
    }
    return ret;
}
int col[20];


int main()
{
   int t;
   cin>>t;
```

```
  LL fact[20];
  fact[0]=1;
  int i;
  for(i=1;i<18;i++)
     fact[i]=fact[i-1]*i;
  while(t--)
  {
     cin>>n;
     mem(col,0);
     for(i=n-1;i>=0;i--)
     {
        cin>>arr[i];
        col[arr[i]]++;
     }
     mem(dp,-1);
     LL ans=dprec(0,0);
     for(i=0;i<=n;i++)
         ans/=fact[col[i]];
     print1(ans);
  }
  return 0;
}
```

Analysis:

The main concept of the problems solution comes from the derangement number. In derangement number every number is occurred just once. But here there may be multiple occurrence. As the constraint is low (n<=15) we can easily do a straight forward 2 dimensional dp. First dimension means the current index we are handling and the second dimension means which place is left to place the numbers. To avoid repetition we have divided the ans with corresponding factorials.

Topcoder SRM 176 Division-1 Level-2

Problem Statement

A derangement of *n* numbers is a permutation of those numbers in which none of the numbers appears in its original place. For example, the numbers {1,2,3} can be deranged into {2,3,1} and {3,1,2}. We can modify this slightly for *n* numbers that are not necessarily distinct by saying that no number in the derangement can be in the place that a number of the same value was in in the original ordering. So the numbers {1,1,2,2,3} could be deranged into {2,2,1,3,1}, {2,2,3,1,1}, {2,3,1,1,2}, and {3,2,1,1,2}. Create a class Deranged with a function numDerangements that takes a int[] **nums** and return the number of derangements of **nums**.

Definition

Class:	Deranged
Method:	numDerangements
Parameters:	int[]
Returns:	long
Method signature:	long numDerangements(int[] nums)
(be sure your method is public)	

Notes

- The return value will fit in a 64-bit unsigned integer.

Constraints

- **nums** will contain between 1 and 15 elements, inclusive.
- **nums** will only contain values between 0 and the number of elements in **nums** - 1, inclusive.

Examples

0)

```
{1,1,2,2,3}
Returns: 4
```

The example from above.

1)

```
{0,0,0,1,1,1}
Returns: 1
```

The only derangement is {1,1,1,0,0,0}.

2)

```
{0,0,0,1,1,1,1}
Returns: 0
```

There is no way to arrange the numbers such that no 1 is where a 1 was originally.

3)

```
{0,0,0,0,0,0,0,1,1,1,1,1,1,1,2}
Returns: 14
```

4)

```
{0,5,4,2,3,6,6}
Returns: 640
```

5)

```
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14}
Returns: 481066515734
```

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.h>
```

```
using namespace std;
#define print1(a)    cout<<a<<endl
#define print2(a,b) cout<<a<<" "<<b<<endl
#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl
#define SZ(s)       ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)
LL dp[16][1<<16];
int arr[20];
int n;

LL dprec(int in,int mask)
{
    //print2(in,mask);
    LL &ret=dp[in][mask];
    if(ret!=-1) return ret;
    if(in==n) return ret=1;

    ret=0;
    int i;
    for(i=0;i<n;i++)
    if((mask&(1<<i))==0)
    {
        if(arr[i]==arr[in])
            continue;
        ret+=dprec(in+1,mask|(1<<i));
    }

    return ret;

}

int col[20];



class Deranged
{
public:
    long long numDerangements(vector <int> nums)
```

```
    {
        LL fact[20];
        fact[0]=1;
        int i;
        for(i=1;i<18;i++)
        fact[i]=fact[i-1]*i;
        mem(col,0);
        n=SZ(nums);
        for(i=n-1;i>=0;i--)
        {
           arr[i]=nums[i];
           col[arr[i]]++;
        }
        mem(dp,-1);
        LL ans=dprec(0,0);
        for(i=0;i<=n;i++)
            ans/=fact[col[i]];
        return ans;
    }
};
```

Analysis:

This is exactly the same problem as previously stated. So the solution is the same.

UVA 10497 - Sweet Child Makes Trouble

Problem F

Sweet Child Makes Trouble

Input: standard input
Output: standard output
Time Limit: 8 seconds

Children are always sweet but they can sometimes make you feel bitter. In this problem, you will see how Tintin, a five year's old boy, creates trouble for his parents.

Tintin is a joyful boy and is always busy in doing something. But what he does is not always pleasant for his parents. He likes most to play with household things like his father's wristwatch or his mother's comb. After his playing he places it in some other place. Tintin is very intelligent and a boy with a very sharp memory. To make things worse for his parents, he never returns the things he has taken for playing to their original places.

Think about a morning when Tintin has managed to 'steal' three household objects. Now, in how many ways he can place those things such that nothing is placed in their original place. Tintin does not like to give his parents that much trouble. So, he does not leave anything in a completely new place; he merely permutes the objects.

Input

There will be several test cases. Each will have a positive integer less than or equal to 800 indicating the number of things Tintin has taken for playing. Each integer will be in a line by itself. The input is terminated by a –1 (minus one) in a single line, which should not be processed.

Output

For each test case print an integer indicating in how many ways Tintin can rearrange the things he has taken.

Sample Input

```
2
3
4
-1
```

Sample Output

```
1
2
9
```

Mycode:

```
#include <iostream>
#include <string>
#include <map>
#include <queue>
#include <stack>
#include <algorithm>
#include <list>
#include <set>
```

```
#include <cmath>
#include <cstring>
#include <stdio.h>
#include <string.h>
#include <sstream>
#include <stdlib.h>
#include <vector>

using namespace std;

#define INF 1<<28
#define SIZE 1000

#define REP(i,n) for (i=0;i<n;i++)
#define REV(i,n) for (i=n;i>=0;i--)
#define FOREACH(it,x) for(__typeof((x).begin()) it=(x.begin());
it!=(x).end(); ++it)
#define FOR(i,p,k) for (i=p; i<k;i++)
#define Reverse(x) reverse(x.begin(),x.end())

#define bug(x)    cout<< "->" <<#x<<": "<<x<<endl   //debug(x)
variable
#define Sort(x) sort(x.begin(),x.end())
#define MP(a,b) make_pair(a,b)
#define CLEAR(x,with) memset(x,with,sizeof(x))
#define COPY(c,r)   memcpy(c,r,sizeof(r))
#define SZ(x) (int)x.size()
#define PB push_back
#define popcount(i) __builtin_popcount(i)
#define gcd(a,b)    __gcd(a,b)
#define fs first
#define sc second

//structure model
struct pq
{
    int cost,node;
    bool operator<(const pq &b)const
    {
        return cost>b.cost;    // Min Priority Queue
    }
};

//Comp Sort
bool comp(pq a,pq b)
{
    return a.cost > b.cost;
}
```

```
string Addition(string a,string b); //Addition any two string
string Multiplication(string a,string b); //Multiplication between a and b
string Multiplication(string a,int k);  //Multiplication between a and int k
string Subtraction(string a,string b);  // Subtraction from a to b(a always >=b)
string Division(string a,string b); //Division return a/b
string Division(string a,int k);    //Division return a/k
string Div_mod(string a,string b);  //Modulus of Division a%b
int    Div_mod(string a,int k);     //Modulus of Division a%k
string cut_leading_zero(string a);  //leading zero cut 001 -> 1
int    compare(string a,string b);  //(1 means a>b) (-1 means a<b) (0 means a=b)
string ans[1000];


int main()
{
   ans[0]="1";
   ans[1]="0";
   int i;
   for(i=2;i<=810;i++)
      ans[i]=Multiplication(Addition(ans[i-1],ans[i-2]),i-1);
   int n;
   while(cin>>n&&n!=-1)
   {
      cout<<ans[n]<<endl;
   }
   return 0;
}
 string Multiplication(string a,int k)
{
   string ans;
   int i,sum,carry=0;

   REV(i,SZ(a)-1)
   {
      sum=(a[i]-'0')*k+carry;
      carry=sum/10;
      ans+=(sum%10)+'0';
   }
   while(carry)    {ans+=(carry%10)+'0';carry/=10;}
   Reverse(ans);
   ans=cut_leading_zero(ans);
   return ans;
}
```

```
string Addition(string a,string b)
{
   int carry=0,i;
   string ans;

   if(SZ(a)>SZ(b)) b=string(SZ(a)-SZ(b),'0')+b;
   if(SZ(b)>SZ(a)) a=string(SZ(b)-SZ(a),'0')+a;
   ans.resize(SZ(a));
   REV(i,SZ(a)-1)
   {
      int sum=carry+a[i]+b[i]-96;
      ans[i]=(char)(sum%10+'0');
      carry=sum/10;
   }
   if(carry)   ans.insert(0,string(1,carry+'0'));
   ans=cut_leading_zero(ans);
   return ans;
}

string cut_leading_zero(string a)
{
   string s;
   int i;
   if(a[0]!='0')   return a;
   REP(i,SZ(a)-1)  if(a[i]!='0')   break;
   FOR(i,i,SZ(a))  s+=a[i];
   return s;
}
```

Analysis:

This is a straight forward derangement number problem. The only problem is the ans is huge when **n** increases. So string operations are needed.

Lightoj 1095 - Arrange the Numbers

Consider this sequence **{1, 2, 3 ... N}**, as an initial sequence of first **N** natural numbers. You can rearrange this sequence in many ways. There will be a total of **N!** arrangements. You have to calculate the number of arrangement of first **N** natural numbers, where in first **M** positions; exactly **K** numbers are in their initial position.

For Example, **N = 5, M = 3, K = 2**

You should count this arrangement **{1, 4, 3, 2, 5}**, here in first 3 positions 1 is in **1st** position and 3 in **3rd** position. So exactly 2 of its first 3 are in there initial position.

But you should not count **{1, 2, 3, 4, 5}**.

Input

Input starts with an integer **T (≤ 1000)**, denoting the number of test cases.

Each case contains three integers **N (1 ≤ N ≤ 1000), M (M ≤ N), K (0 < K ≤ M)**.

Output

For each case, print the case number and the total number of possible arrangements modulo **1000000007**.

Sample Input	Output for Sample Input
2 5 3 2 10 6 3	Case 1: 12 Case 2: 64320

PROBLEM SETTER: MD. ARIFUZZAMAN ARIF
SPECIAL THANKS: JANE ALAM JAN (SOLUTION, DATASET)

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.h>
using namespace std;
#define print1(a)    cout<<a<<endl
#define print2(a,b) cout<<a<<" "<<b<<endl
#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl
#define SZ(s)      ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)
#define mod         1000000007
#define mo 1010
long long comb[mo][mo];
void combination()
{
    LL i,j;
    for(i=0;i<mo;i++)
        comb[i][0]=1;
    comb[1][1]=1;
    for(i=2;i<mo;i++)
        for(j=1;j<mo;j++)
            comb[i][j]=(comb[i-1][j]+comb[i-1][j-
1])%mod;
}
LL dp[mo][mo],fact[mo];

LL dprec(LL m,LL k)
{
    LL &ret=dp[m][k];
    if(ret!=-1) return ret;
```

```
    if(k==0)  ret=fact[m];
    else if(k==1) ret=((m-1)*dprec(m-1,k-1))%mod;
    else ret=(((m-1)*dprec(m-1,k-1))%mod+((k-1)*dprec(m-
2,k-2))%mod)%mod;
    return ret;
}


int main()
{
    combination();
    LL i;
    fact[0]=1;
    fr(i,1,1000) fact[i]=(i*fact[i-1])%mod;
    mem(dp,-1);

    int t;
    int cas=0;
    cin>>t;
    while(t--)
    {
        LL n,m,k;
        cin>>n>>m>>k;
        LL ans=(comb[m][k]*dprec(n-k,m-k))%mod;
        printf("Case %d: ",++cas);
        print1(ans);
    }
    return 0;
}
```

Analysis:

This is a good implementation of derangement number. Before we solve the problem we divide the problem into two sub-problems.

1. Exactly **K** element from the first **M** element are in their initial position. So there are **C(M,K)** ways to select which numbers will be in their own position..
2. Now for every **C(M,K)** ways there are **N-K** elements remaining. There are at least **M-K** elements which will not be in there own position. So the sub-problem is to calculate the number of ways we can arrange **X** numbers so that at least **Y** numbers are not in their own position. The recursive relation is based on exactly the same reasoning as of derangement number. Except for the base case which is factorial of the remaining **X**.