度度熊想去商场买一顶帽子，商场里有 N 顶帽子，有些帽子的价格可能相同。度度熊想买一顶价格第三便宜的帽子，问第三便宜的帽子价格是多少？

```
#include<iostream>
usingnamespacestd;
intmain(){
intn,t=0,syn=0;
intprice[1000]={0};
cin>>n;
while(n--){
cin>>t;
price[t]=1;
}
t=0;
for(inti=0;i<1000;i++){
if(price[t]&&syn<3)
syn++;
if(syn==3)
break;
t++;
}
syn==3?cout<<t:cout<<-1;
}
```

一个数轴上共有 N 个点，第一个点的坐标是度度熊现在位置，第 N-1 个点是度度熊的家。现在他需要依次的从 0 号坐标走到 N-1 号坐标。
但是除了 0 号坐标和 N-1 号坐标，他可以在其余的 N-2 个坐标中选出一个点，并直接将这个点忽略掉，问度度熊回家至少走多少距离？

```
importjava.util.Scanner;

publicclassMain{
publicstaticvoidmain(String[]args){
Scannerscanner=newScanner(System.in);
intlength=scanner.nextInt();
int[]arrays=newint[length];
for(inti=0;i<length;i++){
arrays[i]=scanner.nextInt();
}
/**
*sum 总距离
*repetition 三个节点中被重复计算的总距离
*max 所有三个节点中节点两两相加距离最长的
*add 三个结尾距离为 max 中头尾节点的距离
```

```
*last 最后三个节点中尾距离没有被计算两次需要加上
*/

intsum=0,repetition=0,max=0,add=0,last=0;
for(inti=0;i<=arrays.length-3;i++){
intbegin=arrays[i];
intmid=arrays[i+1];
intend=arrays[i+2];
//三个点之间的距离
intthreePointDistance=Math.abs(mid-begin)+Math.abs(end-mid);
//两个点之间的距离即被多次计算待会需要减掉的距离
inttwoPointDistance=Math.abs(end-mid);
intcontrast=threePointDistance;

repetition+=twoPointDistance;
sum+=threePointDistance;
last=twoPointDistance;

/**
*因为头尾节点无法去除所以头尾节点在判断是否为最长节点的时候需要特殊处理
*例如 132100 按照之前的思虑会去掉 2 但是这却不是正确答案
*因为 100 太大直接导致末尾三个节点距离最长反之头节点也存在相同问题
*所以头尾三个节点距离不能直接和其它依次获取节点相比较
*而应该往前取一个节点让 13100 距离以及 32100 相比较
*/
if(i==0&&arrays.length>4){
intx=Math.abs(arrays[i+3]-begin)+Math.abs(end-begin);
if(threePointDistance<x)
contrast-=Math.abs(mid-begin);
}elseif(i==arrays.length-3&&arrays.length>4){
intx=Math.abs(begin-arrays[i-1])+Math.abs(end-begin);
if(threePointDistance<x)
contrast-=Math.abs(end-mid);
}

if(contrast>max){
max=threePointDistance;
add=Math.abs(end-begin);
}

}
System.out.println(sum-max+last-repetition+add);

}
```

```
}
```

三维空间中有 N 个点，每个点可能是三种颜色的其中之一，三种颜色分别是红绿蓝，分别用'R','G','B'表示。
现在要找出三个点，并组成一个三角形，使得这个三角形的面积最大。
但是三角形必须满足：三个点的颜色要么全部相同，要么全部不同。

```
#include<iostream>
#include<string>
#include<cmath>
#include<cstdio>

usingnamespacestd;

structPoints{//定义结构体 Points
charc;
intx,y,z;
};

doubleCountTriangleArea(PointsA,PointsB,PointsC){//根据三个点计算三角形面积
doublea=sqrt(pow(A.x-B.x,2)+pow(A.y-B.y,2)+pow(A.z-B.z,2));
doubleb=sqrt(pow(A.x-C.x,2)+pow(A.y-C.y,2)+pow(A.z-C.z,2));
doublec=sqrt(pow(C.x-B.x,2)+pow(C.y-B.y,2)+pow(C.z-B.z,2));//计算三边长度 a,b,c
if(a+b<=c||a+c<=b||b+c<=a)//排除掉不符合的情形
return-1;
doublep=(a+b+c)/2;
returnsqrt(p*(p-a)*(p-b)*(p-c));
}

intmain(){
intN;
Pointsp[N];
doubleMaxArea=0;
for(inti=0;i<N;cin>>p[i++].c>>p[i-1].x>>p[i-1].y>>p[i-1].z);
for(inti=0;i<N;++i)
for(intj=0;j<N;++j)
if(i!=j)
for(intk=0;k<N;++k)
if(k!=i&&k!=j&&((p[i].c!=p[j].c&&p[i].c!=p[k].c
&&p[j].c!=p[k].c)||(p[i].c==p[j].c&&p[i].c==p[k].c)))
if(CountTriangleArea(p[i],p[j],p[k])>MaxArea)
MaxArea=CountTriangleArea(p[i],p[j],p[k]);
printf("%.5f",MaxArea);
```

}

度度熊有一个 N 个数的数组，他想将数组从小到大排好序，但是萌萌的度度熊只会下面这个操作：
任取数组中的一个数然后将它放置在数组的最后一个位置。
问最少操作多少次可以使得数组从小到大有序？

```
importjava.util.Scanner;

publicclassMain{

//思路是这样的，这串数字可以分为两类：一类是不动数另一类是动数。动数的个数就是算法要移动的次数。

//不动数满足两个条件：1.后面的数都大于该数。并且,2.前面最小动数大于该数。

//动数和不动数刚好相反：1.后面的数至少有一个数大于该数。或者 2.前面最小动数小于该数。

//不是不动数的数就是动数。只需要找到所有动数个数，就能得到算法移动次数。

publicstaticvoidmain(Stringargs[]){

Scannersc=newScanner(System.in);

intpoint_num=sc.nextInt();

int[]list=newint[point_num];

for(inti=0;i<list.length;i++){

list[i]=sc.nextInt();

}



intmove_count=0;//记录动数的个数。

intmin_move_value=1001;//初始化最小动数的值。

for(inti=0;i<list.length;i++){
```

```
if(min_move_value<list[i]){
move_count++;//满足动数条件 2
continue;
}

for(intj=i+1;j<list.length;j++){
if(list[i]>list[j]){
move_count++;//满足动数条件 1
min_move_value=list[i];//该动数与最小动数值比较
break;
}
}
}

System.out.println(move_count);
}
}
```

度度熊最近对全排列特别感兴趣,对于 1 到 n 的一个排列,度度熊发现可以在中间根据大小关系插入合适的大于和小于符号(即'>'和'<')使其成为一个合法的不等式数列。但是现在度度熊手中只有 k 个小于符号即('<")和 n-k-1 个大于符号(即'>'),度度熊想知道对于 1 至 n 任意的排列中有多少个排列可以使用这些符号使其为合法的不等式数列。

//根据最高赞答案，写的 java 代码，感谢！

```
importjava.util.Scanner;
```

```
publicclassMain{
staticintdp[][];

publicstaticvoidmain(String[]args){
Scannerin=newScanner(System.in);
intn=in.nextInt();
intk=in.nextInt();
dp=newint[n][k+1];

for(inti=1;i<n;i++){
dp[i][0]=1;
}

System.out.print(permutationAmount(n,k));
}

publicstaticintpermutationAmount(intn,intk){
if(n==2&&k==0){
dp[n-1][k]=1;
returndp[n-1][k];
}
if(n==2&&k==1){
dp[n-1][k]=1;
returndp[n-1][k];
}

if(n<=k)
return0;
if(dp[n-1][k]!=0)
returndp[n-1][k];

dp[n-1][k]=(permutationAmount(n-1,k-1)*(n-k)+permutationAmount(n-1,k)
*(k+1))%2017;

returndp[n-1][k];
}
}
```