АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ПЕСНИ

Наивен алгоритъм 1 (Сложност $O(n^2)$)

В двумерен масив mas, който в началото е запълнен с нули, записваме стойности 1 в клетки с номер на ред *a[i]* и номер на стълб *a[j]* , където *a[i]* и *a[j]* са всички възможни двойки елементи от първия масив. Аналогично – записваме стойности 2 в клетки с номер на ред *b[i]* и номер на стълб *b[j],* където *b[i]* и *b[j]* са всички възможни двойки от втория масив, но само ако преди това в тази клетка не е записана стойност 1-ца. Търсения брой на двойките песни, за които препочитанията на Иванчо и на журито не съвпадат ще намерим като обходим масива и намерим броя на елементите със стойност 2. Алгоритъмът решава задачата за предвиденото време за 70% от тестовите примери.

```
#include <iostream>
using namespace std;
const int n_max = 5005;
int mas[n_max][n_max];
int main()
{int n,br=0;
 int a[n_max];
 int b[n_max];
 cin>>n;
 for(int i=1;i<=n;i++)
 cin>>a[i];
 for(int i=1;i<=n;i++)
 cin>>b[i];

 for(int i=1;i<n;i++)
  for(int j=i+1;j<=n;j++)
   mas[a[i]][a[j]]=1;

 for(int i=1;i<n;i++)
  for(int j=i+1;j<=n;j++)
   if(mas[b[i]][b[j]]!=1)
   mas[b[i]][b[j]]=2;

 for(int i=1;i<=n;i++)
  for(int j=1;j<=n;j++)
    if(mas[i][j]==2) br++;
    cout<<br<<endl;
}
```

Наивен алгоритъм 2 (Сложност $O(n^3)$)

Намират се номерата *pa1, pa2, pb1, pb2* на различните двойки числа от 1 до *n* в двата масива от входа. Ако се случи *pa1<pa2 и pb1>pb2* или *pa1>pa2 и pb1<pb2,* тогава броя на двойките песни, за които препочитанията на Иванчо и на журито не съвпадат се

увеличава с единица. Този алгоритъм решава достатъчно бързо задачата за 40% от тестовите примери.

```
#include<iostream>
using namespace std;

const int n_max=50005;
int n;
int a[n_max], b[n_max];

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

  int v=0;

  for(int i1=1;i1<=n;i1++)
  for(int i2=i1+1;i2<=n;i2++)
  {
    int pa1=0;while(a[pa1]!=i1)pa1++;
    int pa2=0;while(a[pa2]!=i2)pa2++;
    int pb1=0;while(b[pb1]!=i1)pb1++;
    int pb2=0;while(b[pb2]!=i2)pb2++;

    if(((pa1<pa2) && (pb1>pb2))||((pa1>pa2) &&(pb1<pb2))) v++;

  }

  cout << v << endl;

}
```

Ефективен алгоритъм(Сложност (O(nlogn)):

Даниите прочитаме в масивите *a[]* и *b[]*.

С помощта на спомагателният масив *c[]* пренареждаме елементите на масива *b[],* така че задачата се свежда до намиране на броя на инверсиите в масива *b*, т.е. броя на двойките елементи *b[i], b[j],* за които *i<j* и *b[i]>b[j]*.

Тази задача решаваме с подхода „разделяй и владей", като получената програма е модификация на известната реализация за сортиране чрез смесване (Merge Sort).

Чрез рекурсивно слизане масивът *b[]* се разделя последователно на по две приблизително равни части, докато се получат части от един елемент. След това при излизане от всяко рекурсивно извикване се реализира смесване на два сортирани масив в нов сортиран масив, като при показаната по-долу реализиция се извършва и броене на инверсиите. В случая това броене се улеснява, понеже двата масива са сортирани.

```
#include<iostream>
using namespace std;
```

```
const int n_max=50005;
int n;
int a[n_max], b[n_max], c[n_max];

int merge(int a[],int beg, int mid, int end)
{
int v=0;
int n = end - beg + 1;
int b[n];
int i1 = beg;
int i2 = mid + 1;
int j = 0;
while ((i1 <= mid) && (i2 <= end))
{ if (a[i1] < a[i2]) {b[j]=a[i1]; i1++;}
else {b[j]=a[i2]; i2++; v += (mid-i1+1);}
j++;
}
while(i1 <= mid) {b[j]=a[i1]; i1++; j++;}
while(i2 <= end) {b[j]=a[i2]; i2++; j++;}
for(j=0; j<n; j++) a[beg+j] = b[j];

return v;
}

int merge_sort(int a[],int beg, int end)
{
if (beg == end) return 0;
int mid = (beg + end) / 2;
int v=0;
v += merge_sort(a, beg, mid);
v += merge_sort(a, mid + 1, end);
v += merge(a, beg, mid, end);
return v;
}

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

  for(int i=0;i<n;i++) c[a[i]]=i+1;
  for(int i=0;i<n;i++) b[i]=c[b[i]];

  int vb=merge_sort(b,0,n-1);

  cout << vb << endl;
}
```

Автор: Зорница Дженкова