

## D. 隱藏的排列 2

Problem ID: permutation2

Time Limit: 1.0s

Memory Limit: 512MiB

本題為**互動題**

### — 問題描述 —

Alice 和 Bob 正在遊玩猜謎遊戲，由 Alice 負責出題目、Bob 負責猜謎。

遊戲過程如下：

- Alice 會先在心中想好一個  $1 \sim n$  的排列。也就是說，Alice 心中已經有一個隱藏的序列  $p_1, p_2, \dots, p_n$ ，滿足這些數字都介在  $1 \sim n$  之間，且每個數字出現恰好一次。
- 接著，Bob 可以詢問  $q$  個問題，每個問題都會是以「請問有多少數對  $(i, j)$  滿足  $l \leq i < j \leq r$ ，且  $p_i > p_j$ 」這種形式呈現。Alice 在收到問題後，必須如實回答。
- 在 Bob 問完所有問題後，Alice 會問 Bob  $k$  個問題，每個問題都會是以「請問  $p_i$  是多少？」這種形式呈現。Bob 在收到問題後，必須給出回答。

這個遊戲的目的就是要讓 Bob 詢問的問題數量  $q$  儘量小來使得 Bob 能正確回答出 Alice 的所有詢問。請協助 Bob，在  $q$  儘量小的情況下，正確回答所有 Alice 的  $k$  個問題。

## — 實作細節 —

你需要實作兩個函式 `bob_init()` 與 `query_from_alice()`：

```
void bob_init(int n);
```

- 對於每一筆測試資料，正式評分程式會呼叫你實作的 `bob_init()` 函式恰好 1 次。
- $n$  代表 Alice 心中想著的排列的長度

```
int query_from_alice(int a);
```

- $a$  為 1 到  $n$  之間的整數
- 對於每一筆測試資料，正式評分程式會呼叫你實作的 `query_from_alice()` 函式恰好  $k$  次。
- 保證在呼叫完 `bob_init()` 後才會呼叫此函式。
- `query_from_alice()` 需要回傳一個整數  $x$ ，代表  $p_a$  的實際數值。

此外，在實作 `bob_init` 時可以呼叫 `compare_numbers()` 這個函式。

```
int compare_numbers(int l, int r);
```

- $l, r$  是於  $1 \sim n$  的整數
- $l \leq r$
- 此函式會回傳有多少數對  $(i, j)$  滿足  $l \leq i < j \leq r$ ，且  $p_i > p_j$
- **範例評分程式**內的 `compare_numbers()` 實作與**實際評分程式**內的實作完全相同

### — 範例程式碼 —

以下是一個可以編譯但保證不會獲得任何分數的範例程式碼：

```
#include<vector>
using namespace std;
int compare_numbers(int l, int r);
int v[1010];
void bob_init(int n){
    v[1] = compare_numbers(1, n);
    v[2] = compare_numbers(2, n);
}
int query_from_alice(int a){
    return v[a];
}
```

### — 互動範例 —

若 Alice 所想的序列為  $\{2, 3, 1\}$ ，一個可能被評為 Accepted 的互動例子顯示如下：

評分程式端	參賽者端
呼叫 bob_init(3)	
	呼叫 compare_numbers(1, 3)
回傳 2	
	呼叫 compare_numbers(2, 3)
回傳 1	
	回傳 void()
呼叫 query_from_alice(1)	
	回傳 2
呼叫 query_from_alice(2)	
	回傳 3

### — 測資限制 —

- $3 \leq n \leq 1000$
- $1 \leq k \leq 1000$
- $1 \leq p_i \leq n$
- $p_i$  兩兩相異

### — 評分說明 —

對於每一筆測試資料，若你的程式在函式 `bob_init()` 中呼叫 `compare_numbers` 的次數為  $x$ ，則定義  $Q$  為：

$$Q = \left\lfloor \frac{x}{n} \right\rfloor$$

若你正確回答了所有 Alice 的詢問，根據  $Q$ ，你將得到分數比重  $W$ ：

$$W = \begin{cases} 1 & \text{if } Q = 0 \\ 1 - \frac{\sqrt{Q}}{50} & \text{if } 0 \leq Q \leq 500 \\ 0 & \text{if } Q > 500 \end{cases}$$

本題共有兩組子任務，條件限制如下所示。每一組可有一或多筆測試資料，你在該子任務的得分為所有測試資料中分數比重  $W$  的最小值，乘以該子任務的總分。

### — 子任務 —

編號	分數	額外限制
1	0	範例互動
2	10	$n = 3$
3	90	無額外限制

## — 範例評分程式 —

範例評分程式採用以下格式輸入：

$$\begin{array}{c} n \ k \\ p_1 \ p_2 \ \dots \ p_n \\ a_1 \ a_2 \ \dots \ a_k \end{array}$$

請注意，正式的評分程式一定不會採用以上格式輸入。**請不要自行處理輸入輸出。**

範例評分程式首先呼叫 `bob_init( $n$ )`，接著範例評分程式會呼叫  $k$  次 `query_from_alice( $a_i$ )`。接著，若範例評分程式偵測到從 `bob_init` 對 `compare_numbers` 的呼叫有任何不合法、或在 `query_from_alice` 的期間有對 `compare_numbers` 的呼叫，此程式將輸出

Wrong Answer : msg

後並終止程式執行，其中 `msg` 為下列其中之一錯誤訊息：

- Invalid position: l r: 你的程式傳入 `compare_numbers` 的集合中有不介在  $1 \sim n$  之間的數字，或是你傳入的  $l > r$ 。
- Invalid call: 你的程式嘗試在 `query_from_alice` 的期間呼叫 `compare_numbers`。

否則，範例評分程式將會以下列格式印在標準輸出中：

$$\begin{array}{c} b_1 \ b_2 \ \dots \ b_k \\ Q \end{array}$$

其中，

- $b_i$  為第  $i$  次呼叫 `query_from_alice()` 時你的回傳值。
- $Q$  為根據你的程式呼叫 `compare_numbers` 的次數得來的數值，詳細定義請見評分說明欄位。
- 請注意，範例評分程式並不會幫助你檢查你回傳的數值是否正確。

下方程式碼可在 pA 下方壓縮檔 permutation2.zip 裡的 grader.cpp 中獲得  
請注意，上傳程式碼時請勿直接上傳此範例評分程式，上傳格式請參考上述範例程式碼。

```
#include <cstdlib>
#include <iostream>
using namespace std;
// Functions to be implemented in the solution.
void bob_init(int n);
int query_from_alice(int a);
// Functions to be implemented in the solution.
namespace{
    int N, K, Query_count = 0, P[1005], Ans[1005], Inv[1005][1005];
    bool EndInit = false;
    void WA(const string msg) {
        cout << "Wrong Answer: " << msg << endl;
        exit(0);
    }
}
int compare_numbers(int l, int r){
    if(EndInit) WA("Invalid call");
    if(l <= 0 || l > N || r <= 0 || r > N || l > r)
        WA("Invalid position: " + to_string(l) + " " + to_string(r));
    Query_count++;
    return Inv[l][r];
}
int main() {
    cin >> N >> K;
    for(int i = 1; i <= N; ++i) cin >> P[i];
    for(int i = 1; i <= N; ++i) for(int j = i + 1; j <= N; ++j)
        if(P[i] > P[j]) Inv[i][j]++;
    for(int i = 1; i <= N; ++i) for(int j = 1; j <= N; ++j)
        Inv[i][j] += Inv[i][j - 1];
    for(int j = 1; j <= N; ++j) for(int i = j - 1; i > 0; --i)
        Inv[i][j] += Inv[i + 1][j];
    bob_init(N);
    EndInit = true;
    for(int i = 1, x; i <= K; ++i)
        cin >> x, Ans[i] = query_from_alice(x);
    for(int i = 1; i <= K; ++i) cout << Ans[i] << " \n"[i == K];
    cout << Query_count / N << "\n";
}
```