合唱队形 解题报告

<问题描述>

N位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的K位同学排成合唱队形。

合唱队形是指这样的一种队形：设K位同学从左到右依次编号为1，2…，K，他们的身高分别为T1，T2，…，TK， 则他们的身高满足T1<...<Ti>Ti+1>…>TK(1<=i<=K)。

你的任务是，已知所有N位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

- 输入文件

输入文件chorus.in的第一行是一个整数N(2<=N<=100)，表示同学的总数。第一行有n个整数，用空格分隔，第i个整数Ti(130<=Ti<=230)是第i位同学的身高(厘米)。

- 输出文件

输出文件chorus.out包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

- 样例输入

```
8
186 186 150 200 160 130 197 220
```

- 样例输出

```
4
```

- 数据规模

对于50％的数据，保证有n<=20；
对于全部的数据，保证有n<=100。

<算法分析>

动态规划。最基本的想法是：枚举中间最高的一个人，接着对它的左边求最长上升序列（注意序列中最高的同学不应高过基准），对右边求最长下降序列（同样的，序列中最高的同学不应高过基准）。时间复杂度为O(n^3)，算法实现起来也很简单。

接着对这个算法进行分析，我们不难发现，假如还是基于枚举一个同学的话，设Incsq[i]表示了1 - i的最长上升序列，Decsq[i]表示了i - n的最长下降序列，那么，

Current[i] = Incsq[i] + Decsq[i] - 1（两个数组中i被重复计算了）

那么，我们只需要先求好最长上升和下降序列，然后枚举中间最高的同学就可以了。

<算法优化>

求最长上升序列的经典状态转移方程为：

opt[i] = max{opt[j]+1, 其中 i<j<=n, 且 list[j]>list[i]}

我们对状态转移方程稍微做一些修改：

opt[i] = max{opt[i+1], min{j, rec[j]>=list[i]}}

rec[j] = list[i]

很明显可以看出，在 opt[i]的寻找 j 的过程当中，查询序列是单调的，于是可以用二分法，就十分巧妙地在 logn 的时间内找到指定的 j，而问题的总体复杂度为 O(nlogn)。这样，这个问题的算法效率就得到了大幅度的提升，即便 n 是 10^6，也可以轻松应对。

<代码清单>

```
#include <fstream>
#include <cstring>
using namespace std;

ifstream fin("chorus.in");
ofstream fout("chorus.out");

const int maxn = 100;

int n, a[maxn];
int incsq[maxn], decsq[maxn];

void init() {
    fin >> n;
    for (int i = 0; i < n; i ++)
        fin >> a[i];
}

void LIncSeq()
{
    int i, low, high, mid, ans = 0;
    int sol[maxn];

    for (i = 0; i < n; i ++) {
        low = 1; high = ans;
        while (low <= high) {
            mid = (low + high) >> 1;
            if (sol[mid] < a[i]) low = mid + 1;
            else high = mid - 1;
        }
        if (low > ans) ans ++;
        sol[low] = a[i];
        incsq[i] = ans;
    }
}

void LDecSeq()
{
    int i, low, high, mid, ans = 0;
    int sol[maxn];

    for (i = 0; i < n; i ++) {
        low = 1; high = ans;
```

```
        while (low <= high) {
            mid = (low + high) >> 1;
            if (sol[mid] > a[i]) low = mid + 1;
            else high = mid - 1;
        }
        if (low > ans) ans ++;
        sol[low] = a[i];
        decsq[i] = ans;
    }
}

void work() {
    int i, max = 0;

    LIncSeq();
    LDecSeq();

    for (i = 0; i < n; i ++)
        if (incsq[i] + decsq[i] - 1 > max)
            max = incsq[i] + decsq[i] - 1;

    fout << n - max << endl;
}

int main() {
    init();
    work();
    return 0;
}
```

<小结>

问题虽然简单，仍然不能放过思考的余地。O(n^3)的算法是可以通过所有测试数据的，但是nlogn的算法里，不但体现了二分法的思想，而且也体现了多次动态规划的思想，这个思想在解决很多问题的时候，都有很大的作用。