F7

published by lx

Analysis

主要的方法是每一次对一个选手进行判断，判断他是否能够在最优情况下得冠军。为了方便我们处理，我们先将选手按照他们之前的分数排序，使Score[1] ≥ Score[2] ≥ ... ≥ Score[n]

先考虑之前分数最低的n号选手，贪心策略便是在这次比赛中，1号选手得1分，二号选手得2分，…，n号选手得n分，这样显然是所有得分方案中对n号选手最有利的方案。那么我们只要判断在这种方案下n号选手是否有可能的冠军即可。

如果$\max_{1\le i<n}$\{Score[i]+i\}≤Score[n]+n，那么n号选手就有获得冠军的可能。我们暴力求最大值Max=$\max_{1\le i<n}$\{Score[i]+i\}并判断。

接着考虑第n-1号选手，根据贪心策略，1号选手得1分，二号选手得2分，…，n-2号选手得n-2分，n-1号选手得n分，n号选手得n-1分。那么我们需要判断max(Score[n]+n−1,$\max_{1\le i<n-1}$\{Score[i]+i\})≤Score[n]+n是否成立，其实，仔细思考可以发现Score[n]<Score[n−1], n−1<n，所以Score[n]+n−1<Score[n−1]+n一定成立，我们就可以不考虑Score[n]造成的影响，判断$\max_{1\le i<n-1}$\{Score[i]+i\}≤Score[n−1]+n是否成立即可，而$\max_{1\le i<n-1}$\{Score[i]+i\}取自\{$\max_{1\le i<n-2}$\{Score[i]+i\}，Score[n−1]+n−1\}。

考虑到这里，其实我们可以直接取我们之前算过的Max，因为max里的取值在第二种情况时一定小于Score[n−1]+n，而取第一种情况时可以用Max代替，所以这次找$\max_{1\le i<n-1}$\{Score[i]+i\}时直接用之前的Max判断，不用再找了。

是不是很奇妙，其实仔细讨论可以发现，每一次都可以用Max来判断，那么Max就成了我们预处理的对象，再O(n)判断每一个人就可以了，这就是本题所有的贪心及优化策略。

Code

```
#include <stdio.h>
#include <stdlib.h>
int cmp(const void *x, const void *y)
{
	return *(int *)x < *(int *)y ? 1: -1;
}

int main()
{
	int n, score[300005], i;
	scanf("%d", &n);
	for (i = 0; i < n; ++i)
		scanf("%d", score+i);
	qsort(score, n, sizeof(score[0]), cmp);
	int ans = 0;
	int Max = 0;
	for (i = 0; i < n; ++i) {
		ans += (score[i] + n >= Max);
		if (Max < score[i]+i+1) Max = score[i]+i+1;
	}
	printf("%d\n", ans);
	return 0;
}
```