# G. Hiring

## PROBLEM DESCRIPTION

給 n 天，每天共可上班$t_i$ 小時；

有 m 個應試者，每個人在每天上班時必須先準備$d_j$小時，總共需要$r_j$小時才能完成任務。

問每個應試者可以在哪一天結束完成任務(或是無法 => 輸出 0)。

## SOLUTION TECHINQUES

Sweeping line / Heap / Fenwick Tree

## SOLUTION SKETCHES

核心想法是掃描線，

我們對於應試者先用 d(準備時數)做排序，接著從小到大開始做；

我們必須在過程中維護第 1~某天共有多少時數、有多少天時數比當前的 d 還大；

對於每個應試者 j，我們先排除比 $d_j$ 還小的工作天(更新時數總和、以及天數的 fenwick tree)，然後由於完成天有單調性(若某天可以，那在之後的天也一定可以)，我們可以二分搜尋在哪一天可以完成任務，檢查到該天的總工作時數夠不夠$r_j$ (時數總和 $-$ 天數 $*$ $d_j \geq r_j$ )，二分搜後即可得到應試者 j 的答案。

## TIME COMPLEXITY

$O(m * \log m * \log[\max(n, m)]$

## SOLUTION PROGRAM FOR REFERENCE

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cctype>
#include <cstdlib>
#include <vector>
#include <queue>

using namespace std;
typedef long long ll;

const int N = 2 * 1e5 + 2;
const int M = N;

int n, m;
ll fen[M];
int vld[M];
int ans[N];

struct worker
{
        int prep, tot, pos;
        bool operator <(const worker &rhs)const
        {
                return prep < rhs.prep;
        }
} a[N];

struct work
{
        int days, pos;
        bool operator <(const work &rhs)const
        {
                return days > rhs.days;
        }
};
priority_queue<work> pq;

void upd(int pos, ll add)
{
        while (pos <= m)
        {
                fen[pos] += add;
                pos += pos & (-pos);
        }
}

void upd2(int pos, int add)
{
        while (pos <= m)
        {
                vld[pos] += add;
                pos += pos & (-pos);
```

```
        }
}

ll que(int pos)
{
        ll ret = 0;
        while (pos >= 1)
        {
                ret += fen[pos];
                pos -= pos & (-pos);
        }
        return ret;
}

int que2(int pos)
{
        int ret = 0;
        while (pos >= 1)
        {
                ret += vld[pos];
                pos -= pos & (-pos);
        }
        return ret;
}

int main()
{
        int i, j, k;
        int wi;
        int lo, mi, hi;
        scanf("%d%d", &n, &m);
        for (i = 1; i <= m; i++)
        {
                scanf("%d", &wi);
                upd(i, wi);
                upd2(i, 1);
                pq.push( (work){wi, i} );
        }
        for (i = 0; i < n; a[i].pos = i, i++)
                scanf("%d%d", &a[i].prep, &a[i].tot);
        sort(a, a + n);
        for (i = 0; i < n; i++)
        {
                while (!pq.empty() && pq.top().days < a[i].prep)
                {
                        work w = pq.top(); pq.pop();
                        upd(w.pos, -w.days);
                        upd2(w.pos, -1);
                }
                if (que(m) - (ll)a[i].prep * que2(m) < a[i].tot)
                        ans[a[i].pos] = 0;
                else
                {
                        lo = 1; hi = m;
                        while (lo < hi)
                        {
```

```
                                mi = (lo + hi) >> 1;
                                if (que(mi) - (ll)a[i].prep * que2(mi) < a[i].tot)
                                        lo = mi + 1;
                                else
                                        hi = mi;
                        }
                        ans[a[i].pos] = lo;
                }
        }
        for (i = 0; i < n; i++)
                printf("%d ", ans[i]);
        return 0;
}
```