USACO OPEN12 Problem 'running' Analysis

by Brian Dean

Let us define L(i) as the number of laps cow i performs until the race ends. For simplicity, we will think of L(i) as a real number, although in the implementation below we can manage to do all of our math in integers (always a good idea, to avoid round-off issues). If L(i) > L(j), then the number of times cow i crosses cow j is given by the floor of L(i)-L(j). Our goal is therefore to sum up floor(L(i)-L(j)) over all i>j (assuming the cows are ordered in increasing order of L(i)).

If all we had to do was sum up L(i)-L(j) over all i>j, this would be easy: we would first precompute the prefix sums P(j)=L(1)+...+L(j), and then we can write the sum of L(i)-L(j) over all i>j as the sum of jL(i)-P(i) over all i; this can be therefore computed in linear time. The floor function is really the tricky aspect of this problem. To deal with this properly, we start by setting each L(i) to its floor, and by computing prefix sums as before. We then sum up jL(i)-P(i) over all i, but in increasing order of the fractional part of L(i). As we proceed, we add +1 to each L(i) we encounter (and adjust the prefix sums accordingly, using an appropriate data structure like a binary index tree). Travis' code below shows how to implement this idea.

```
#include <cstdio>
#include <algorithm>
using namespace std;
#define nmax 100005

int bit[nmax];
int bitlen;

inline void bit_init(int n) {
        for(int i = 1; i <= n; i++) {
                bit[i] = 0;
        }
        bitlen = n;
}

inline int bit_prefix_sum(int i) {
        int sum = 0;
        for(int j = i; j > 0; j -= (j & (-j))) {
                sum += bit[j];
        }
        return sum;
}

inline void bit_inc(int i) {
        for(int j = i; j <= bitlen; j += (j & (-j))) {
                bit[j]++;
        }
}

struct cow {
        long long speed;
        long long modulus;
        int rank;
};
cow cows[nmax];
long long max_speed = 0;
long long n, l, c;

inline bool sort_cow_by_modulus(cow const& a, cow const& b) {
        return a.modulus < b.modulus;
}

inline bool sort_cow_by_speed(cow const& a, cow const& b) {
        return a.speed < b.speed;
}

int main() {
        freopen("running.in","r",stdin);
        freopen("running.out","w",stdout);
        scanf("%lld", &n);
        scanf("%lld", &l);
        scanf("%lld", &c);

        for(int i = 0; i < n; i++) {
                scanf("%lld", &cows[i].speed);
                if(cows[i].speed > max_speed)
                        max_speed = cows[i].speed;
        }

        for(int i = 0; i < n; i++)
                cows[i].modulus = (l*c*cows[i].speed) % (c * max_speed);
        sort(cows, cows + n, sort_cow_by_modulus);
        int a = 0;
        int rank = 1;
        while(a < n) {
                int b = a+1;
                while(b < n && cows[a].modulus == cows[b].modulus) b++;
                for(int i = a; i < b; i++)
                        cows[i].rank = rank;
                a = b;
                rank++;
        }

        sort(cows, cows + n, sort_cow_by_speed);
        bit_init(n);

        long long total = 0;
        long long sum_of_floors = 0;
        for(int i = 0; i < n; i++) {
                long long floor = (l*cows[i].speed) / (max_speed);
                long long addition = floor*i - sum_of_floors - (long long)i +
                        (long long)bit_prefix_sum(cows[i].rank);

                total += addition;
                sum_of_floors += floor;
                bit_inc(cows[i].rank);
        }
        printf("%lld\n", total);
}
```