Blitz 4 Editorial

June 22nd, 2020

1 Omkar and Washing

Iterate through the input and keep a counter for every time a number does not exceed X, and is less than K away from X.
Time Complexity: $O(N)$

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, x, k, ans = 0; cin >> n >> x >> k;
    while(n--) {
        int a; cin >> a;
        if(x-a >= 0 and x-a <= k) ans++;
    }
    cout << ans << endl;
}
```

2 Omkar and Number

This question is an apparent simple implementation to check how many factors a number has. However, if we desire a prime number of factors, the smallest number which fits the criteria is far too large, even considering the relatively small bounds. To overcome this, make the observation that the lowest number with N factors (where N is prime) is in the form of $2^{(N-1)}$.

Time Complexity: $O(N^2 \cdot log(N))$

C++ Code

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll raise(ll a, ll b) { return b ? raise(a*a,b/2)*(b%2?a:1) : 1; }

ll incr(ll n){
    ll out = 1;
    for(ll i = 2; i*i <= n; i++){
      ll t = 1;
      while(!(n%i)) { t++; n /= i; }
      out *= t;
    }
    return n > 1 ? out*2 : out;
}

bool isprime(ll n) {
    for(ll i = 2; i*i <= n; i++) if(!(n%i)) return false;
    return true;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    ll n; cin >> n;
    if(isprime(n)) cout << raise(2, n-1);
    else {
        ll ans = 1;
        while(incr(ans) != n) ans++;
        cout << ans << endl;
    }
}
```

3 Omkar and Politics

Because the seating is circular, we can greedily insert Omkaryotes by alternating between the smallest and largest political belief values available.
If the question involved a line instead of a circle, this algorithm would not work!
Time Complexity: $O(N)$

C++ Code

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int n; cin >> n;
    vector<int> nums(n), ans; for(auto &e : nums) cin >> e;
    sort(nums.begin(), nums.end());
    int l = 0, r = n-1;
    while(l < r){
      ans.push_back(nums[l++]);
      ans.push_back(nums[r--]);
    }
    if(l == r) ans.push_back(nums[l]);
    ll tot = abs(ans[n-1]-ans[0]);
    for(int i = 1; i < n; i++) tot += abs(ans[i]-ans[i-1]);
    cout << tot << endl;
}
```

4 Omkar and Squad

To answer each query, we can do a binary search on the skill level of the k'th Omkaryote. Since the number of groups in each squad is small, we can binary search to find how many Omkaryotes are below/above a certain threshold in every member group for each binary pass. When this number reaches K, we have found the desired skill level.

Time Complexity: $O(Q \cdot log(N))$

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int n, q, t, l; cin >> n >> q;
    vector<vector<int>> sq(n);
    for(int i = 0; i < n; i++){
        int l; cin >> l;
        for(int j = 0; j < l; j ++) { cin >> t; sq[i].push_back(t); }
        sort(sq[i].begin(), sq[i].end());
    }
    int m, k;
    while(q--){
        cin >> m >> k;
        vector<int> ind(m);
        for(auto& e : ind) { cin >> i; --i; }
        int l = 1, r = 100000;
        while(l < r){
            int m = (l+r)/2;
            int ma = 0;
            for(auto& i : ind){
                ma +=
(upper_bound(sq[i].begin(),sq[i].end(),m)-sq[i].begin());
            }
            if(ma < k) l = m+1;
            else r = m;
        }
        cout << l << endl;
    }
}
```

5 Omkar and Laps

Without going into too much depth on the mathematics involved, by writing out the first couple terms of when Lord Omkar looks up (1, 3, 6, 4...), we can notice a pattern. Clearly, both Omkaryotes will only be back at the starting line when $t = x \cdot LCM(A, B)$, where x is some arbitrary coefficient. By taking the mod of the list of times when Lord Omkar looks up, you can see that a pattern repeats every $2N$ times (can you think why?). We see that the mod equals zero twice in this recurrence (and twice as many for odd numbers). Using this knowledge, we can do some simple math to find how many cycles will be within Lord Omkar's viewing.

Time Complexity: $O(LCM(A, B))$

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int main(){
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  ll a, b, l; cin >> a >> b >> l;
  ll t = 2*a*b/__gcd(a,b);
  ll ans = -1;
  for(ll i = 0; i < t && i <= l; i++) if(i*(i+1)%t == 0) ans += 1+(l-i)/t;
  cout << ans << endl;
}
```