Editorial: Monsters - 1849B

Avighna

June 2024

1 Initial (Naïve) Approach

The most obvious initial approach would be to simply implement what the question is asking. Keep track of the number of killed monsters, keep finding the greatest health monster, subtracting k from it and updating the value of the counter variable when necessary.

The code for this (which will be too slow) is as follows:

```
#include <bits/stdc++.h>

using namespace std;

#define ll long long

void solve() {
  ll n, damage;
  cin >> n >> damage;
  vector<ll> m(n);
  for (ll i = 0; i < n; ++i) {
    cin >> m[i];
  }

  ll killed = 0;
  while (killed != n) {
    ll mind = max_element(m.begin(), m.end()) - m.begin();
    if (m[mind] - damage <= 0) {
      cout << mind + 1 << " ";
      killed++;
    }
    m[mind] -= damage;
  }
  cout << "\n";
}

int main() {
  ll t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

2 Slightly Improved Naïve Approach

At this point, one may notice that instead of finding the largest monster in $O(n)$, one can instead use a `std::priority_queue` to do the same in $O(\log n)$, but this is still too slow to pass.

There are various more optimizations that can be further made to the naïve approach, but I highly doubt anything will get it fast enough to be accepted. That being said, let's now move on to a better method.

3 Optimal Solution

Key Idea: If the initial health of the monsters is given as $[a_1, a_2, a_3, ..., nk, a_m, a_{m+1}, a_{m+2}, ...]$, and $a_x \mod k \neq 0$, then the monster nk will be the first one to die (i.e. get reduced to 0).

Here is the proof for this.

Note that if $nk > a_x$ at every stage, then this is trivially true (since we use the special attack on the largest element at every stage).

If not, then $a_x < nk$ (equality is impossible due to our indivisibility assumption). Since all monster healths are integers, $n \in Z^+$. If $n = 0$, then nk has already been reduced to 0 (which is what we wanted to prove). Otherwise, $a_x > nk$ and thus $a_x > k$ (n has now been proven to be a natural number) and $a_x - nk > 0$. This tells us that at every single stage, the *reduced value* of any monster whose health is not divisible by k can never go under 1.

More specifically, after complete reduction, the health of any monster not divisible by k will fall in the range of $[1, k)$ (if this were not true, then nk would be reduced to $(n-1)k$ and then k would be subtracted from a_x again).

Thus, we can conclude that all monsters divisible by k will be reduced first. Try to see why they will be equal to k at the same time as well. Of course, since we choose the highest index first, we will go from left to right while eliminating them.

4 Building on this idea

We have previously concluded that all monsters divisible by k will be killed first. In this process, all other monsters will also be reduced to the range $[1, k)$.

Then, since we choose monsters with the maximum health first, we'll kill the ones which are the greatest in the range $[1, k)$. These numbers have been reduced to this range as, previously, k has been repeatedly subtracted from them. In other words (I'm sure some of you see what has to be done already): we kill the ones whose modulus with k is the greatest first (here I'm referring to the original healths).

a_x will be killed before a_y if $m_x \mod k > m_y \mod k$.

I'd recommend that you try coding the solution yourself now. If you didn't completely understand any part, let your brain passively think about this idea for a few minutes and re-read this. With that being said, here's the solution code:

```
#include <bits/stdc++.h>

using namespace std;

#define ll long long

void solve() {
  ll n, k;
  cin >> n >> k;
  vector<pair<ll, ll>> m(n);
  for (ll i = 0; i < n; ++i) {
      cin >> m[i].first;
      m[i].first %= k;
      if (m[i].first == 0) {
        m[i].first = k;
      }
      m[i].second = -1 * i;
  }

  sort(m.begin(), m.end(), greater<pair<ll,ll>>());
  for (ll i = 0; i < n; ++i) {
    cout << -1*m[i].second+1<<" ";
  }

  cout<<"\n";
}

int main() {
  ll t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```