Sorting and Set Questions

Raveesh Gupta 03/05/2020

Problem 1

(Codeforces 339A) Xenia the beginner mathematician is a third year student at elementary school. She is now learning the addition operation.

The teacher has written down the sum of multiple numbers. Pupils should calculate the sum. To make the calculation easier, the sum only contains numbers 1, 2 and 3. Still, that isn't enough for Xenia. She is only beginning to count, so she can calculate a sum only if the summands follow in non-decreasing order. For example, she can't calculate sum 1+3+2+1 but she can calculate sums 1+1+2 and 3+3.

You've got the sum that was written on the board. Rearrange the summans and print the sum in such a way that Xenia can calculate the sum.

Solution

Summation will not differ on re-arrangement of summans because addition is a commutative binary operation and by definition order of summans is not important.

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
int parse(char c){
    if(c == '1') return 1;
    if(c == '2') return 2;
    if(c == '3') return 3;
}
int main(){
    vector<int> a;
    string s;
    cin>>s;
    for(char c: s){
        if (c == '+') continue;
        a.push_back(parse(c));
    }
    sort(a.begin(), a.end());
    cout<<a[0];
    for(int i = 1; i < a.length(); i++)
        cout<<'+'<<a[i];
    return 0;
}
```

Problem 2

(Codeforces 405A) Little Chris is bored during his physics lessons (too easy), so he has built a toy box to keep himself occupied. The box is special, since it has the ability to change gravity.

There are n columns of toy cubes in the box arranged in a line. The i_{th} column contains a i cubes. At first, the gravity in the box is pulling the cubes downwards. When Chris switches the gravity, it begins to pull all the cubes to the right side of the box. The figure shows the initial and final configurations of the cubes in the box: the cubes that have changed their position are highlighted with orange.

Given the initial configuration of the toy cubes in the box, find the amounts of cubes in each of the n columns after the gravity switch!

Solution

Transformation : Every block on (x, y) repeatedly moved to $(x + 1, y)$ until another block encountered. Assume that y coordinate of a block has to be positive.
Assume if a block is not present on $(x, y = 1)$ position after flipping the gravity then it implies that block is not present on $(x - 1, y = 1)$.
Lemma: if there exists a block on $(x+1, y)$ and $(x-1, y)$ in final configuration after gravity flips then there exists a block on (x, y).

Assume: if there exists a block on $(x + 1, y)$ and $(x - 1, y)$ in final configuration after gravity flips then there does not exists a block on (x, y).
$\implies$ block on $(x - 1, y)$ co-ordinate must move to (x, y) on flipping of gravity by definition of transformation.
$\implies$ That's a contradiction.

Corollary: Blocks form a monotonically increasing sequence of heights.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    int n;
    cin>>n;
    vector<int> a;
    for(int i = 0;i<n;i++){
        int b;
        cin >> b;
        a.push_back(b);
    }
    sort(a.begin(), a.end());
    for(int i = 0; i < n; i++) cout<<a[i]<<" ";
    return 0;
}
```

Problem 3

(Codeforces 160A) Imagine that you have a twin brother or sister. Having another person that looks exactly like you seems very unusual. It's hard to say if having something of an alter ego is good or bad. And if you do have a twin, then you very well know what it's like.

Now let's imagine a typical morning in your family. You haven't woken up yet, and Mom is already going to work. She has been so hasty that she has nearly forgotten to leave the two of her darling children some money to buy lunches in the school cafeteria. She fished in the purse and found some number of coins, or to be exact, n coins of arbitrary values $a_1, a_2, ..., a_n$. But as Mom was running out of time, she didn't split the coins for you two. So she scribbled a note asking you to split the money equally.

As you woke up, you found Mom's coins and read her note. "But why split the money equally?" — you thought. After all, your twin is sleeping and he won't know anything. So you decided to act like that: pick for yourself some subset of coins so that the sum of values of your coins is strictly larger than the sum of values of the remaining coins that your twin will have. However, you correctly thought that if you take too many coins, the twin will suspect the deception. So, you've decided to stick to the following strategy to avoid suspicions: you take the minimum number of coins, whose sum of values is strictly more than the sum of values of the remaining coins. On this basis, determine what minimum number of coins you need to take to divide them in the described manner.

Solution

Lemma: Largest sum of subset of size k among all the subsets is the suffix of size k of the array in sorted order.

Assume Largest sum of subset of size k among all the subsets is not the suffix of size k of the array in sorted order.

$\implies$ Exchanging an element from the prefix of size $n - k$ will make the suffix larger. Let the prefix element be p and suffix element be s

$\implies$ Since the sequence is in sorted order. For every p in prefix $p < s \implies$ exchanging the elements between suffix and prefix will decrease the suffix sum for every element in prefix.

$\implies$ suffix is the largest subset of size k. Contradiction.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main(){
    int n;
    cin>>n;
    vector<int> a;
    long long N = 0;
    for(int i = 0; i < n; i++){
        int x;
        cin>>x;
        a.push_back(x);
        N += x;
    }
    sort(a.begin(), a.end());
    long long sm = 0;
    for(int i = n -1; i>= 0; i--){
        sm += a[i];
        if( 2 * sm > N ){
            cout<< n - i;
            break;
        }
    }
    return 0;
}
```

Problem 4

(cses.fi $\#4$) You are given an array of n integers, and your task is to find two values (at distinct positions) whose sum is x.

The first input line has two integers n and x: the array size and the target sum.

The second line has n integers $a_1, a_2, ..., a_n$: the array values.

Print two integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.

Solution

To find u in $(u, v) \mid u + v = x$ by iteration of $\{a_i\}$ and for each $a_i = u$, v is searched in the rest using binary search of value $key = x - u$ in the sorted $\{a_i\}$.

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


struct pt{
    int val, pos;
};
bool comp(struct pt lt, struct pt rt){
    return ( lt.val < rt.val ||( ( lt.val == rt.val) && (lt.pos < rt.pos) ↩
       ) );
}
vector<struct pt> a;
int n;

int _search(struct pt ele, int x){
    int key = ele.val, in = ele.pos;
    int lo = 0, hi = (int) n - 1;
    while( lo + 1 < hi ){
        int mi = lo + ( (hi - lo) >> 1);
        if(a[mi].val + key < x) lo = mi;
        else hi = mi;
    }
    for(int i = lo; i <= hi; i++){
        if(in == a[i].pos) continue;
        if(a[i].val + key == x) return a[i].pos;
    }
    return -1;
}


int main(){
    int x;
    scanf("%d %d", &n, &x);
    for( int i = 0; i < n; i++){
        int v;
        struct pt tmp;

        scanf("%d", &v);
        tmp.val = v, tmp.pos = i;
        a.emplace_back( tmp );
    }
    sort(a.begin(), a.end(), comp);

    for( int i = 0 ; i < n; i++) {
        struct pt a_i = a[i];
        int in = _search(a_i, x);
        if( in == -1) continue;
        int k = a_i.pos;
        printf("%d %d\n", in+1, k+1);
        return 0;
    }
    printf("IMPOSSIBLE\n");
    return 0;
}
```

Problem 5

(cses.fi $\#5$)Given an array of n integers, your task is to find the maximum sum of values in a contiguous, nonempty subarray.

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, ..., x_n$: the array values.

Solution

The common simplification is divide $maxsuf(i)$ into finding the maximum sum_i among all $suffixes$ of $prefix_i$ where $prefix_i$ contains elements $x_0, x_1, ..., x_i$.

Lemma : $maxsuf(i+1)$ is $maxsuf(i) + x_i$ or x_i.

$$maxsuf(i+1) = \begin{cases} maxsuf(i) + x_i \\ x_i \end{cases}$$

Proof : Assume the lemma is true for $maxsuf(0...i)$.

Base Case: x_i is the max $suffix$ of size 1 for $prefix_i$.

The element x_{i+1} has two options
$\{x_i\}$ remain alone.

If $suffix$ is of size 1 then x_{i+1} is the only choice else for size > 1 it is $x_{i+1} + max(suffixes)$ of $prefix_i \implies x_{i+1} + maxsuf(i)$ which is true by our assumption.

Since there are only 2 options taking max of them is the $maxsuf(i+1)$. $\square$

The maximum subarray is the maximum among all all the $MAX(maxsuf(i) \mid i \in [1, n])$

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

int n;

long long kadane(vector<long long> &a){
    long long max_l = -1e18, max_g = -1e18;
    for(long long a_i : a){
        max_l = max(max_l + a_i * 1LL, a_i * 1LL);
        max_g = max(max_g, max_l);
    }
    return max_g;
}

int main(){
    scanf("%d", &n);
    vector<long long> a(n);
    for(int i = 0;i < n; i++)
        scanf("%lld", &a[i]);
    long long sm = kadane(a);
    printf("%lld\n", sm);
    return 0;
}
```

Problem 6

(cses.fi $\#6$)You are given a playlist of a radio station since its establishment. The playlist has a total of n songs.

What is the longest sequence of successive songs where each song is unique?

The first input line contains an integer n : the number of songs.

The next line has n integers $k_1, k_2, ..., k_n$: the id number of each song.

Print the length of the longest sequence of unique songs.

Solution

Sliding window approach to find the longest continuous segment of distinct elements $\{k_i\}$.

Question is to find the the longest continuous sub-array of distinct elements.

```
#include <bits/stdc++.h>
#define ll long long int
using namespace std;
int n
vector<ll> a, fq;
void normalize(vector<ll> b){
    sort(b.begin(), b.end());
    for(int i =0; i < n; i++){
        int ind = distance(b.begin(), lower_bound(b.begin(), b.end(), a[i↩
           ]));
        a[i] = ind;
    }
}
int main(){
    scanf("%d", &n);
    a.resize(n);
    for(int i= 0; i < n; i++)
        scanf("%lld", &a[i]);
    normalize(a);
    fq.resize(n, -1);
    int lt = 0;
    int ans = -1;
    fq[a[0]] = 1;
    for(int rt = 1;rt < n;rt++){
        if(fq[a[rt]] != -1){
            while(lt < fq[a[rt]] ) fq[a[lt++]] = -1;
        }
        fq[ a[rt] ] = rt + 1;
        ans = max(rt - lt + 1, ans);
    }
    printf("%d", ans);
    return 0;
}
```

Problem 7

(Codeforces 1134B) Many years ago Berland was a small country where only n people lived. Each person had some savings: the $i-th$ one had a_i burles.

The government considered a person as wealthy if he had at least n burles. To increase the number of wealthy people Berland decided to carry out several reforms. Each reform looked like that:

the government chooses some subset of people (maybe all of them)
the government takes all savings from the chosen people and redistributes the savings among the chosen people equally.
For example, consider the savings as list [5,1,2,1]: if the government chose the 1-st and the 3-rd persons then it, at first, will take all 5+2=7 burles and after that will return 3.5 burles to the chosen people. As a result, the savings will become [3.5,1,3.5,1].

A lot of data was lost from that time, so we don't know how many reforms were implemented and to whom. All we can do is ask you to calculate the maximum possible number of wealthy people after several (maybe zero) reforms.

Solution

Definition: Mean x of n items is a value such that sum of distances $\Sigma(a_i - x) \mid a_i > x$ greater than mean is equal to sum of distances $\Sigma(x - a_i) \mid a_i < x$ smaller than the mean.

Lemma 1: Mean of all people with wealth $\geq x$ and some people $< x$ wealth should be at least $x \implies$ after one reform all selected people wealth $\geq x$

Proof Assume taking mean of all(k) people with wealth $> x$ would increase the number of wealthy people.
$x < a_i \forall i$
$\implies kx < \Sigma(a_i)$
$\implies x < \Sigma(a_i)/k$
After one reform all the people with wealth$> x$ would be still greater than x hence no change in the number of wealthy people. Contradiction !

Observation: smaller the sum of people with wealth $< x$ the better it is. Let's assume sum of wealth of people (> x) is S. $\implies$ and total distance from mean is $S - kx$

Lemma 2: Optimal selection of q people with wealth $< x$ is taking q people closest to x.
Sum of difference is $D_q = qx - S_q$ where S_q is sum of q closest people to x whose wealth $< x$

Assume taking mean of more people than q say $q + \delta$ and then performing a reform by taking mean of those people and selecting q people from mean position will result in smaller difference sum $D_{q+\delta}$ from x than D_q from x.

Let A be additional sum of wealth of δ people

new mean $= \frac{S_q+A}{\delta+q}$
Taking q people will result in sum $= \frac{q(S_q+A)}{q+\delta}$
$D_{\delta+q} = qx - \frac{q(S_q+A)}{q+\delta}$
$D_{\delta+q} < D_q$
$\implies qx - \frac{q(S_q+A)}{q+\delta} < qx - S_q$
$\implies q(S_q + A) > S_q$
$\implies qS_q + qA > qS_q + S_q$
$\implies qA > S_q$
$\implies \frac{A}{\delta} > \frac{S_q}{q}$ Contradiction !!

```
#include <iostream>
#include <algorithm>
using namespace std;
```

```
int a[100003];
bool check(long long sm, long long m, long long x){
    if(m*x > sm) return false;
    return true;
}
int main(){
    int t;
    scanf("%d", &t);
    int n, x;
    for(int i = 0; i<t; i++){
        scanf("%lld %lld", &n, &x );
        for(int j = 0; j < n; j++)
            scanf("%d", &a[j]);
        sort(a, a+n);
        long long sm = 0;
        int ans = 0;
        for(int j = n - 1; j > - 1; j--){
            sm += a[j];
            if(check(sm, n - j, x)) ans = n - j;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

Problem 8

(Codeforces 1339B) You have array of n numbers $a_1, a_2, a_3, ..., a_n$

Rearrange these numbers to satisfy $|a_1 a_2| \geq |a_2 a_3| \geq ...|a_{n-1} - a_n|$, where $|x|$ denotes absolute value of x. It's always possible to find such rearrangement.

Note that all numbers in $\{a_i\}$ are not necessarily different. In other words, some numbers of $\{a_i\}$ may be same.

You have to answer independent t test cases.

Solution

Lemma 1 : Largest difference between 2 elements in sorted a is $|a_1 - a_n|$

Lemma 2 : It's always possible to decrease the distance by either increasing $l \rightarrow l + 1$ or decreasing $r \rightarrow r - 1$

$d(l,r)$is defined as $a_r - a_l$

$$d(1,n) \geq d(2,n) \geq d(2,n-1)...$$

To Prove: $I(l,r) = d(l,r) \geq d(l+1,r) \geq I(l+1,r-1)$ where $l < r$

Base: $d(i,i)$ or $d(i,i+1) = a_{i+1} - a_i$ which is positive as a is sorted order.

Assume : Inequality $I(l+1,r-1)$ is true

Inductive Step: $I(l,r) = d(l,r) \geq d(l+1,r) \geq I(l+1,r-1)$
$\implies d(l,r) \geq d(l+1,r$
$\implies a_r - a_l \geq a_r - a_{l+1}$
$\implies a_{l+1} \geq a_l$ True as a is sorted
and $d(l+1,r) \geq I(l+1,r-1)$
$\implies d(l+1,r) \geq d(l+1,r-1)$
$\implies a_r - a_{l+1} \geq a_{r-1} - a_{l+1}$
$\implies a_r \geq a_{r-1}$ True as a is sorted

QED by Induction.

```cpp
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
    int a[100005];
    int t;
    vector<int> b;
    while(t--){
        int n;
        scanf("%d", &n);
        for(int i= 0; i < n; i++){
            scanf("%d", &a[i]);
        }
        sort(a, a+n);

        int left = 0, right = n -1;
        bool flag = true;
        while(left <= right){
            if(!flag){
                b.push_back(a[left]);
                left++;
              }
            else{
                b.push_back(a[right]);
                right--;
            }
            flag = !flag;
        }
        reverse(b.begin(), b.end());
        b.clear();
    }
    return 0;
}
```

Problem 9

(Codeforces 1324D) The next lecture in a high school requires two topics to be discussed. The $i-th$ topic is interesting by $a_i$units for the teacher and by b_i units for the students.

The pair of topics i and j $(i < j)$ is called good if $a_i + a_j > b_i + b_j$ (i.e. it is more interesting for the teacher).

Your task is to find the number of good pairs of topics.

Solution

lemma: If $S$is all the pairs $(i, j) \mid i \neq j$ such that $c_i + c_j > 0$ where $c_k \forall k \in [0, n-1] = a_k - b_k$ then number of pairs in S such that $i > j$ is $\frac{\#S}{2}$.

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    int n, a[200005], b[200005];
    vector<int> c;
    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    for(int i =0;i< n; i++){
        scanf("%d ", &b[i]);
    }
    for(int i = 0;i < n;i++)
```

```cpp
        c.push_back(a[i] - b[i]);
    sort(c.begin(), c.end());
    long long m =0;
    for(int i = 0; i < n;i++){
        m +=  c.end() - upper_bound(c.begin(), c.end(), -c[i]);
        if(c[i] > 0) m--;
    }
    m /= 2;
    cout<<m;
    return 0;
}
```

Problem 10

(Codeforces 1335C) You have n students under your control and you have to compose exactly two teams consisting of some subset of your students. Each student had his own skill, the $i-th$ student skill is denoted by an integer a_i (different students can have the same skills).

So, about the teams. Firstly, these two teams should have the same size. Two more constraints:
The first team should consist of students with distinct skills (i.e. all skills in the first team are unique).

The second team should consist of students with the same skills (i.e. all skills in the second team are equal).

Note that it is permissible that some student of the first team has the same skill as a student of the second team.

Your task is to find the maximum possible size x for which it is possible to compose a valid pair of teams, where each team size is x (skills in the first team needed to be unique, skills in the second team should be the same between them). A student cannot be part of more than one team.

You have to answer t independent test cases.

Solution

Let k_i be the number of elements in A that have the same value of a_i

Let T be the size of team of distinct elements in A and $team_{2i}$ be the size of team with elements having same value a_i

Lemma: if $n > 1$ then for every a_i there exists $team_1$ and$team_{2i}$ teams where $team_1 = team_{2i}$ and $team_1$ is a set having different elements.

if $k_i > T$ then $team_1 = min(k_i - 1, T)$

if $T = k_i$ then $team_1 = min(T - 1, k_i - 1)$

if $T > k_i$ then $team_1 = min(T - 1, k_i)$

```cpp
#include<iostream>
#include<vector>
#include<set>
#include<algorithm>
using namespace std;
vector<int> a, k;
set<int> s;

int test(){
    int n;
    scanf("%d", &n);

    for(int i = 0; i< n;i++){
        int x;
        scanf("%d", &x);
        a.push_back(x);
        s.insert(x);
    }
    if(n == 1){
        return 0;
    }
    sort(a.begin(), a.end());
    for(auto a_i: s){
    k.push_back(upper_bound(a.begin(), a.end(), a_i) - lower_bound(a.begin↩
       (), a.end(), a_i));
    }
    int T = (int)k.size();
    int MAX  = -1;
    for(int i = 0; i<T; i++) {
        MAX = max(MAX, max(min(k[i] - 1, T), min(T - 1, k[i])));
    }
    return MAX;
}
int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        a.clear();
        s.clear();
        k.clear();
    printf("%d\n", test());
    }
    return 0;
}
```

Problem 11

(Codeforces 1312B) You are given an array $a_1, a_2, ..., a_n$. Array is good if for each pair of indexes $i < j$ the condition $j - a_j \neq i - a_i$ holds. Can you shuffle this array so that it becomes good? To shuffle an array means to reorder its elements arbitrarily (leaving the initial order is also an option).

It's guaranteed that it's always possible to shuffle an array to meet this condition.

Solution

Lemma : if minimum index is subtracted with the maximum a and maximum index is subtracted with minimum a then $i_{min} - a_{max}$ and $i_{max} - a_{min}$ are not equal to any $i_j - a_k$.

Assume $a_y = a_x + \delta$ and $j = i + \Delta$ where j and i are 2 unique indices.

Let's say the 2 values are equal

$i - a_y = j - a_x$
$\implies i - a_x - \delta = i + \Delta - a_x$
$\implies -\delta = \Delta$ where $\delta \geq 0, \Delta > 0$
Contradiction!!
QED

```
#include<iostream>
```

```
#include<vector>
#include<set>
#include<algorithm>
using namespace std;
int main(){
 int a[200005], n;
 int t;
 scanf("%d", &t);
 while(t--){
    scanf("%d", &n);
    for(int i = 0; i < n;i++)
        scanf("%d", &a[i]);
    sort(a, a+n);
    reverse(a, a+n);
    for(int i =0;i< n;i++)
        printf("%d ", a[i] );
    printf("\n");
 }
  return 0;
}
```

Problem 12

(Codeforces 1320A) Tanya wants to go on a journey across the cities of Berland. There are n cities situated along the main railroad line of Berland, and these cities are numbered from 1 to n.

Tanya plans her journey as follows. First of all, she will choose some city c_1 to start her journey. She will visit it, and after that go to some other city $c_2 > c_1$, then to some other city $c_3 > c_2$, and so on, until she chooses to end her journey in some city $c_k > c_{k-1}$. So, the sequence of visited cities $[c_1, c_2, ..., c_k]$ should be strictly increasing.

There are some additional constraints on the sequence of cities Tanya visits. Each city i has a beauty value b_i associated with it. If there is only one city in Tanya's journey, these beauty values imply no additional constraints. But if there are multiple cities in the sequence, then for any pair of adjacent cities c_i and c_{i+1}, the condition $c_{i+1}c_i = b_{c_{i+1}}b_{c_i}$ must hold.

Tanya wants her journey to be as beautiful as possible. The beauty value of the whole journey is the sum of beauty values over all visited cities. Can you help her to choose the optimal plan, that is, to maximize the beauty value of the journey?

Solution

If some sequence is strictly increasing consider them indices, hence the condition is $y - x = b_y - b_x$ where y and x are adjacent cities.

$y - x = b_y - b_x$ where $y = c_{k+1}$ and $x = c_k$
$\implies y - b_y = x - b_x$
$\implies d_y = d_x$ where $d_i = i - b_i$

Sort the array d_i then calculate the maximum sum of beauty among the continuous sequence of d_i that has the same value and choose the maximum sum of b_i.

```cpp
#include <iostream>
#include <algorithm>
#include <utility>
#include <vector>
using namespace std;

int main(){
    int n, b[200005];
    vector<pair<long long, int> > di;
    scanf("%d", &n);
    for(int i = 0; i< n; i++) scanf("%d", &b[i]);
    for(int it = 0;it< n;it++)
        di.emplace_back(make_pair((long long)(it - b[it]), b[it]));
    sort(di.begin(), di.end());
    int i = 0;
    long long val = di[0].first;
    long long MAX = -1e12 - 5;
    for(;i<n;){
        long long mx = 0;
        val = di[i].first;
        for(;i < n && di[i].first == val;i++) mx += di[i].second;
        MAX = max(MAX, mx);
    }
    printf("%lld", MAX);
    return 0;
}
```

Problem 13

(Codeforces 1296D) There are n monsters standing in a row numbered from 1 to n. The $i-th$ monster has h_i health points (hp). You have your attack power equal to a hp and your opponent has his attack power equal to b hp.

You and your opponent are fighting these monsters. Firstly, you and your opponent go to the first monster and fight it till his death, then you and your opponent go the second monster and fight it till his death, and so on. A monster is considered dead if its hp is less than or equal to 0.

The fight with a monster happens in turns.

You hit the monster by a hp. If it is dead after your hit, you gain one point and you both proceed to the next monster.
Your opponent hits the monster by b hp. If it is dead after his hit, nobody gains a point and you both proceed to the next monster.

You have some secret technique to force your opponent to skip his turn. You can use this technique at most k times in total.

Your task is to determine the maximum number of points you can gain if you use the secret technique optimally.

In each fight the first turn is always yours (it does not depend on who killed the last monster)

Solution

Lemma: if health of the monster $> b$ then attack by the opponent will not change the maximum number of points.

Calculate the amount of skips required to defeat $i-th$ monster and gain a point.
$c_i = ceil(\frac{max(rem'(hp_i,a+b)-a,0)}{a})$ where $rem'(x,y) = x \pmod y$ if $x \pmod y \neq 0$ else $rem'(x,y) = y$

Question reduces to largest subset of vector $\{c_i\}$. as done before.
Sort the monsters by c_i and calculate the largest prefix such that $\Sigma c_i \leq k$ the length of the prefix is the maximum number of points.

```
#include <iostream>
#include <algorithm>
#include <utility>
```

```cpp
#include <vector>
using namespace std;
int n, k, a, b, hp[200005];
int rem(int a, int b){
    if(a % b) return a%b;
    else return b;
}
int cal_c(int h){
    return (max(rem(h, a+b) - a, 0) + a - 1)/a;
}
bool cmp(int left, int right){
    return cal_c(left) < cal_c(right);
}
int main(){
    scanf("%d %d %d %d", &n, &a, &b, &k);
    for(int i = 0; i< n;i++) scanf("%d", &hp[i]);
    sort(hp, hp+n, cmp);
    long long sm= 0;
    long long ans = 0;
    for(int i = 0; i < n; i++){
        sm+= cal_c(hp[i]);
        if(sm <= k) ans = i+1;
    }
    printf("%lld\n", ans);
    return 0;
}
```

Problem 14

(Codeforces 1334C) You are playing another computer game, and now you have to slay n monsters. These monsters are standing in a circle, numbered clockwise from 1 to n. Initially, the $i-th$ monster has a_i health.

You may shoot the monsters to kill them. Each shot requires exactly one bullet and decreases the health of the targeted monster by 1 (deals 1 damage to it). Furthermore, when the health of some monster i becomes 0 or less than 0, it dies and explodes, dealing b_i damage to the next monster (monster $i+1$, if $i < n$, or monster 1, if $i = n$). If the next monster is already dead, then nothing happens. If the explosion kills the next monster, it explodes too, damaging the monster after it and possibly triggering another explosion, and so on.

You have to calculate the minimum number of bullets you have to fire to kill all n monsters in the circle.

Solution

Lemma: if the attack is done in order of indices then minimum number of bullets can be achieved

Proof:

Assume: if the attack is not done in order of indices then minimum number of bullets can be achieved.

$\implies$ There exist an order in which $i+1-th$ monster is defeated before $i-th$ monster.

$\implies$ In that order the b_i damage done to $i+1-th$ monster is absent and extra bullets $= min(b_i, a_i) > 0$ than the bullets required in order of indices.

$\implies$ Contradiction to assumption. QED

If the attack starts from $i-th$ monster.

Total bullets are

$D_i = a_i + max(a_{i+1} - b_i, 0) + ... + max(a_{i-1} - b_{i-2}, 0)$

$\implies D_i = a_i - max(a_i - b_{i-1}, 0) + C$

where $C = max(a_0 - b_{n-1}, 0) + max(a_1 - b_0, 0) + max(a_2 - b_1, 0) + ... + max(a_{n-1} - b_{n-2}, 0)$

$MIN(D_i) = C + min(a_i - max(a_i - b_{i-1}, 0))$

```cpp
#include <iostream>
#include <algorithm>
#include <utility>
#include <vector>
using namespace std;

int main(){
    long long int a[300004], b[300004],c[300004], n;
    int t;
    scanf("%d", &t);
    while(t--){
        scanf("%lld", &n);
        for(int i = 0; i<n; i++)
            scanf("%lld %lld", &a[i], &b[i]);
        for(int i = 0; i < n; i++)
            c[i] = a[i] - max(a[i] - b[i - 1 < 0 ? n - 1:i - 1], 0LL);
        long long int MIN = 1e12 + 9;
        for(int i = 0; i < n; i++)
            MIN = min(MIN, c[i]);
        long long int C =0;
        for(int i =0; i < n; i++)
            C += max(a[i] - b[i - 1 < 0 ? n - 1: i - 1], 0LL);
        printf("%lld\n", C + MIN);
    }
    return 0;
}
```

Problem 15

(Codeforces 1282B2) Vasya came to the store to buy goods for his friends for the New Year. It turned out that he was very lucky — today the offer "k of goods for the price of one" is held in store.

Using this offer, Vasya can buy exactly k of any goods, paying only for the most expensive of them. Vasya decided to take this opportunity and buy as many goods as possible for his friends with the money he has.

More formally, for each good, its price is determined by a_i — the number of coins it costs. Initially, Vasya has p coins. He wants to buy the maximum number of goods. Vasya can perform one of the following operations as many times as necessary:

Vasya can buy one good with the index i if he currently has enough coins (i.e $p \geq a_i$). After buying this good, the number of Vasya's coins will decrease by a_i , (i.e it becomes $p = p - a_i$).

Vasya can buy a good with the index i, and also choose exactly $k - 1$ goods, the price of which does not exceed a_i, if he currently has enough coins (i.e $p \geq a_i$). Thus, he buys all these k goods, and his number of coins decreases by a_i (i.e it becomes $p = p - a_i$).

Please note that each good can be bought no more than once.

Solution

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int n, k, t;
    long long p, a[200005];
    scanf("%d", &t);
    while(t--){
        scanf("%d %lld %d", &n, &p, &k);
        for(int i = 0; i < n; i++) scanf("%lld", &a[i]);
        sort(a, a+n);
        for(int i = 1; i < n;i++){
            if(i == k - 1) continue;
            if(i >= k)
                a[i] += a[i - k];
            else
```

```
                a[i] += a[i - 1];
        }
        int ans = 0;//extreme case(degenerate)
        for(int i = 0; i < n;i++) if(a[i] <= p) ans = i + 1;
        printf("%d\n", ans);
    }
    return 0;
}
```

Problem 16

(Codeforces 1348C) Phoenix has a string s consisting of lowercase Latin letters. He wants to distribute all the letters of his string into k non-empty strings $a_1, a_2, ..., a_n$ such that every letter of s goes to exactly one of the strings a_i. The strings a_i do not need to be substrings of s. Phoenix can distribute letters of s and rearrange the letters within each string a_i however he wants.

Phoenix wants to distribute the letters of his string s into a_i strings $a_1, a_2, ..., a_n$ to minimize the lexicographically maximum string among them, i. e. minimize $max(a_1, a_2, ..., a_n)$. Help him find the optimal distribution and print the minimal possible value of $max(a_1, a_2, ..., a_n)$.

Solution

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
char s[100005];
int n, k, t;

bool allsame(char str[], int n){
    char val = str[0];
    for(int i =0; i < n; i++){
        if(str[i] != val) return false;
    }
    return true;
}
void solve(){
        scanf("%d %d", &n, &k);
        scanf("%s", s);
        sort(s, s+n);
        if(k == 1){
            printf("%s\n", s);
            return;
        }
        if(allsame(s, n)){
            for(int i = 0; i < (n+k-1)/k;i++) cout<<s[0];
            cout<<endl;
            return;
        }
        if(!allsame(s, k)){
            printf("%c\n", s[k - 1]);
            return;
        }

        if(allsame(s + k, n - k)){
            cout<<s[k - 1];
            for(int i = 0; i < (n -1)/k;i++) cout<<s[k];
        }
        else{
            for(int i = k - 1; i < n;i++) cout<<s[i];
        }
        cout<<endl;
}
int main() {

    scanf("%d", &t);
    while(t--){
        solve();
    }
    return 0;
}
```

Problem 17

(Codeforces 1316B) Vasya has a string s of length n. He decides to make the following modification to the string:

Pick an integer k, $(1 \leq k \leq n)$. For i from 1 to $n-k+1$, reverse the substring $s[i, i+k-1]$ of s.

Vasya wants to choose a k such that the string obtained after the above-mentioned modification is lexicographically smallest possible among all choices of k. Among all such k, he wants to choose the smallest one. Since he is busy attending Felicity 2020, he asks for your help.

Solution

```cpp
#include <iostream>
#include <string>

using namespace std;

string rev(string t, int index){
    string a;
    for(int i = index; i> -1; i--) a.push_back(t[i]);
    return a;
}
int main() {
    string s;
    int n;
    int t;
    cin>>t;
    while(t--){
        cin>>n; cin>>s;
        string MIN = s;
        int min_arg = 1;
        for(int k = 2; k <= n;k++){
            string f = s.substr(k - 1, n - k + 1);
            if((n - k + 1) % 2== 1)
                f += rev(s, k-2);
            else
                f += s.substr(0, k - 1);

            if(f < MIN){
                MIN = f;
                min_arg = k;
            }
        }
        s = rev(s, n -1);
        if(s < MIN){
                MIN = s;
                min_arg = n;
            }
        cout <<MIN << endl << min_arg<<endl;
    }
    return 0;
}
```

Problem 18

(Codeforces 1284B) A sequence $a = \{a_1, a_2, ..., a_l\}$of length l has an ascent if there exists a pair of indices (i, j) such that $1 \leq i < j \leq n$ and $a_i < a_j$. For example, the sequence [0,2,0,2,0] has an ascent because of the pair (1,4), but the sequence [4,3,3,3,1] doesn't have an ascent.

Let's call a concatenation of sequences p and q the sequence that is obtained by writing down sequences p and q one right after another without changing the order. For example, the concatenation of the [0,2,0,2,0] and [4,3,3,3,1] is the sequence [0,2,0,2,0,4,3,3,3,1]. The concatenation of sequences p and q is denoted as $p + q$.

Gyeonggeun thinks that sequences with ascents bring luck. Therefore, he wants to make many such sequences for the new year. Gyeonggeun has n sequences $s_1, s_2, ..., s_n$ which may have different lengths.

Gyeonggeun will consider all n^2 pairs of sequences s_x and s_y $1 \leq x < y \leq n$, and will check if its concatenation $s_x + s_y$ has an ascent. Note that he may select the same sequence twice, and the order of selection matters.

Please count the number of pairs (x, y) of sequences $s_1, s_2, ..., s_n$ whose concatenation $s_x + s_y$ contains an ascent.

Solution

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
bool good_check(vector<int> &s){
    for(int i = 0;i < s.size() - 1; i++)
        if(s[i] < s[i + 1]) return true;
    return false;
}
int GET_MIN(vector<int> &s){
    int MIN = 1e9;
    for(int i =0;i< s.size();i++)
        MIN = min(MIN, s[i]);
    return MIN;
}
int GET_MAX(vector<int> &s){
    int MAX = -1e9;
    for(int i =0;i< s.size();i++)
        MAX = max(MAX, s[i]);
    return MAX;
}
int main(){
     int n;
     scanf("%d", &n);
     vector<vector<int> > S(n), G, B;
     for(int i =0;i < n;i++){
        int m;
        scanf("%d", &m);
        for(int ii = 0; ii< m;ii++){
            int x;
            scanf("%d", &x);
            S[i].push_back(x);
        }
        if(good_check(S[i]))
            G.push_back(S[i]);
        else
            B.push_back(S[i]);
     }

     vector<int> MIN, MAX;
     for(int i = 0;i < B.size();i++){
        MIN.push_back(GET_MIN(B[i]));
        MAX.push_back(GET_MAX(B[i]));
     }
     sort(MAX.begin(), MAX.end());
     long long cnt = 0;
     for(int i = 0; i < B.size(); i++)
        cnt += MAX.end() - upper_bound(MAX.begin(), MAX.end(), MIN[i]);

     printf("%lld", G.size() * G.size() + 2*G.size()*B.size() + cnt);
     return 0;
}
```

Problem 19

(Codeforces 1213D2) You are given an array a_i consisting of n integers. In one move you can choose any a_i and divide it by 2 rounding down

You can perform such an operation any (possibly, zero) number of times with any a_i.

Your task is to calculate the minimum possible number of operations required to obtain at least k equal numbers in the array.

Don't forget that it is possible to have $a_i = 0$ after some operations, thus the answer always exists.

Solution

```cpp
#include <iostream>
#include <algorithm>
#include <numeric>
using namespace std;

int main(){
    int a[200005], n, k;
    vector<int> cost[200005];
    scanf("%d %d", &n, &k);
    for(int i =0;i < n;i++){
        scanf("%d", &a[i]);
    }
    int MAX = -1;
    for(int i = 0; i < n;i++){
        int cur = 0;
        int x = a[i];
        MAX = max(MAX, a[i]);
        while(x >= 0){
            cost[x].push_back(cur);
            cur++;
            if(x==0) break;
            x /= 2;
        }
    }
    int MIN = 1e9;
    for(int x = 0; x < MAX+1;x++){
        sort(cost[x].begin(), cost[x].end());
        MIN = min(MIN, accumulate(cost[x].begin(), cost[x].begin() + k));
    }
    printf("%d", MIN);
    return 0;
}
```

Problem 20

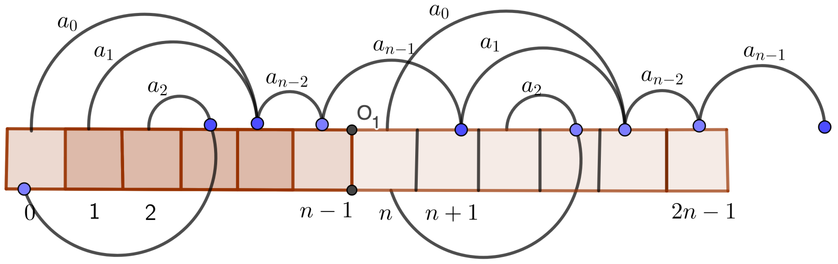
(Codeforces 1344A) Hilbert's Hotel is a very unusual hotel since the number of rooms is infinite! In fact, there is exactly one room for every integer, including zero and negative integers. Even stranger, the hotel is currently at full capacity, meaning there is exactly one guest in every room.

The hotel's manager, David Hilbert himself, decides he wants to shuffle the guests around because he thinks this will create a vacancy (a room without a guest).

Then the shuffling works as follows. There is an array of n integers $a_0, a_1, ..., a_{n-1}$. Then for each integer k, the guest in room k is moved to room number $k + a_k \pmod{n}$.

After this shuffling process, determine if there is still exactly one guest assigned to each room. That is, there are no vacancies or rooms with multiple guests.

Solution



```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
bool collide[200005];
int n, a[200005];
int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        memset(collide, false, 200005 * sizeof collide[0]);
        scanf("%d", &n);
        for(int i =0;i <n;i++){
            scanf("%d", &a[i]);
        }
        for(int i =0; i < n;i++){
            a[i] = (n + ( a[i] + i )%n)%n;
            collide[a[i]] = true;
        }
        bool flag = true;
        for(int i = 0;i < n && flag;i++)
            if(!collide[i]) {
                printf("NO\n");
                flag = false;
            }
        if(flag)
            printf("YES\n");
        }
    return 0;
}
```

Problem 21

(Codeforces 1348B) Phoenix loves beautiful arrays. An array is beautiful if all its subarrays of length k have the same sum. A subarray of an array is any sequence of consecutive elements.

Phoenix currently has an array a_i of length n. He wants to insert some number of integers, possibly zero, into his array such that it becomes beautiful. The inserted integers must be between 1 and n inclusive. Integers may be inserted anywhere (even before the first or after the last element), and he is not trying to minimize the number of inserted integers.

Solution

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

void solve(){
  int N,K;
  cin>>N>>K;
  set<int>s;
  for (int i=0;i<N;i++){
    int a;
    cin>>a;
    s.insert(a);
  }
  //if more than K distinct numbers, print -1
  if (s.size()>K){
    cout<<-1<<endl;
    return;
  }
  cout<<N*K<<endl;
  for (int i=0;i<N;i++){
    //print the distinct numbers
    for (int b:s)
      cout<<b<<' ';
    //print the extra 1s
    for (int j=0;j<K-(int)s.size();j++)
      cout<<1<<' ';
  }
  cout<<endl;
}

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