A ExAб И нОд

$a=1$

$b=x-1$

Решение C++

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
      int x;
      scanf("%d",&x);
      printf("1 %d\n",x-1);
    }
}
```

Решение Python

```
for _ in range(int(input())):
    print(1,int(input())-1)
```

B Плотный массив

Заметим, что добавление элементов между позициями i $(1 \le i \le n-1)$ и $i+1$ не изменит отношение между соседними элементами, кроме только что добавленных. Поэтому для каждой пары соседних чисел задачу можно решать независимо.

Решим задачу для соседней пары чисел a_i и a_{i+1} для которых не выполнено неравенство из условия. Предположим, что $2a_i \le a_{i+1}$ (если не так, то поменяем их местами). Тогда между a_i и a_{i+1} необходимо вставить $\left\lceil log_2\left(\frac{a_{i+1}}{a_i}\right) - 1 \right\rceil$ элементов вида:

$$2a_i, 4a_i, \ldots, 2^{\left\lceil log_2\left(\frac{a_{i+1}}{a_i}\right)-1 \right\rceil}a_i$$

Для подсчёта ответа лучше не использовать явную формулу, а воспользоваться следующим циклом:

```
while a[i] * 2 < a[i + 1]:
  a[i] *= 2
  ans += 1
```

Решение C++

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

using namespace std;
using ll = long long;

void solve() {
```

```
  int n;
  cin >> n;
  int last;
  cin >> last;
  int ans = 0;
  for (int i = 1; i < n; i++) {
    int nw;
    cin >> nw;
    int a = min(last, nw), b = max(last, nw);
    while (a * 2 < b) {
      ans++;
      a *= 2;
    }
    last = nw;
  }
  cout << ans << "\n";
}

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

Решение Python

```
t=int(input())
for l in range(0,t):
    n=int(input())
    s=list(map(int,input().split()))
    sl=0
    for i in range(0,n-1):
        a=min(s[i],s[i+1])
        b=max(s[i],s[i+1])
        while(a*2<b):
            sl+=1
            a*=2
    print(sl)
```

C Сумма кубов

В этой задаче требовалось найти такие a и b, что $x = a^3 + b^3$ и $a \geq 1, b \geq 1$.

Так как a и b положительные, то a^3 и b^3 тоже положительные. Следовательно $a^3 \leq a^3 + b^3 \leq x$. Поэтому число a можно перебрать от 1 до $\sqrt[3]{x}$. Так как во всех тестах $x \leq 10^{12}$, то $a \leq 10^4$.

Для каждого a можно однозначно найти b по формуле $b = \sqrt[3]{x - a^3}$. Это положительное число. Осталось проверить, что b целое.

Решение C++

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

using namespace std;
using ll = long long;
using ld = long double;
```

```
const ll N = 1'000'000'000'000L;

unordered_set<ll> cubes;

void precalc() {
  for (ll i = 1; i * i * i <= N; i++) {
    cubes.insert(i * i * i);
  }
}

void solve() {
  ll x;
  cin >> x;
  for (ll i = 1; i * i * i <= x; i++) {
    if (cubes.count(x - i * i * i)) {
      cout << "YES\n";
      return;
    }
  }
  cout << "NO\n";
}

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

Решение Python

```
for t in range(int(input())):
      n=int(input())
      i=1
      while i**3<n:
            j=int((n-i**3)**(1/3)+0.5)
            if i**3+j**3==n:
                  print('YES')
                  break
            i+=1
      else:
            print('NO')
```

D Настройка сети

В этой задаче нужно было отсортировать массив по невозрастанию и вывести *k*-ый элемент. Также ограничения позволяли решать следующим образом. Переберем *ans* — величину ответа и посчитаем, сколько компьютеров уже имеют скорость не меньшую *ans*. Если их не меньше *k*, то такой ответ нам подходит. Среди всех подходящих ответов выберем максимум, это и будет искомой величиной.

Решение C++

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

using namespace std;

int main(){
```

```
    int n,k;
    cin >> n >> k;
    int s[n];
    for(int j = 0;j<n;j++)cin >> s[j];
    sort(s,s+n);
    cout << s[n-k] << "\n";
    return 0;
}
```

Решение Python

```
n,k=map(int,input().split())
l=list(map(int,input().split()))
l.sort()
print(l[n-k])
```

E Трансформация перестановки

Будем строить искомое дерево рекурсивно. Опишем состояние построения дерева тремя величинами (l, r, d), где $[l, r]$ — отрезок перестановки, а d — текущая глубина. Тогда можно описать следующие переходы:

- найдём позицию m максимального элемента на отрезке $[l, r]$, то есть $a_m = \max\limits_{i=l}^{r} a_i$;
- глубина вершины a_m равна d;
- если $l < m$, тогда сделаем переход в состояние $(l, m-1, d+1)$;
- если $m < r$, тогда сделаем переход в состояние $(m+1, r, d+1)$;

Тогда, чтобы построить искомое дерево, необходимо принять за исходное состояние $(1, n, 0)$.

Решение C++

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

using namespace std;

void build(int l, int r, vector<int> const &a, vector<int> &d, int curD = 0)
{
  if (r < l) {
    return;
  }
  if (l == r) {
    d[l] = curD;
    return;
  }
  int m = l;
  for (int i = l + 1; i <= r; i++) {
    if (a[m] < a[i]) {
      m = i;
    }
  }
  d[m] = curD;
  build(l, m - 1, a, d, curD + 1);
  build(m + 1, r, a, d, curD + 1);
}

void solve() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int &x : a) {
```

```
    cin >> x;
  }
  vector<int> d(n);
  build(0, n - 1, a, d);
  for (int x :d) {
    cout << x << " ";
  }
  cout << endl;
}

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

Решение Python

```
def f(l,r,d):
    if l>r:return
    b[a.index(max(a[l:r+1]))]=d
    f(l,a.index(max(a[l:r+1]))-1,d+1)
    f(a.index(max(a[l:r+1]))+1,r,d+1)

for _ in range(int(input())):
    n = int(input())
    a  = list(map(int, input().split()))
    b = [0]*n
    f(0,n-1,0)
    print(*b)
```

F Случайная победа

Как можно для конкретного игрока проверить, может ли он выиграть чемпионат? Очевидно, что он должен поучаствовать во всех играх (иначе мы увеличим количество фишек у противников). Так что можно остортировать всех людей и жадно играть с самыми слабыми. Такая проверка будет работать за линейное время после сортировки, так что мы получили решение за $\mathcal{O}(n^2)$.

Самое простое решение этой задачи — бинарный поиск по ответу. Остортируем всех игроков по количеству имеющихся у них фишек. Давайте докажем, что если игрок i может выиграть, то игрок $i+1$ также может выиграть (номера раздаются после сортировки). Если игрок i смог выиграть, значит исходя из стратегии выше он смог победить всех игроков на префиксе $[0 \dots i+1]$. Игрок $i+1$ также может победить всех этих игроков, так как у него не меньше фишек. Теперь обоим игрокам осталось победить всех противников с номерами $[i+2 \dots n]$ и количество фишек у обоих игроков равно сумме первых $i+1$ чисел в массиве. Значит если у игрока i есть стратегия, то игрок $i+1$ может воспользоваться той же самой стратегией.

Следовательно ответ на задачу — отсортированный суффикс входного массива. Найти этот суффикс можно с помощью бинарного поиска и проверки за линейное время.

Бонус: у этой задачи также есть полностью линейное (после сортировки) решение.

Решение C++

```
#include<bits/stdc++.h>
#define ll long long int
#define M 1000000007
using namespace std;
ll n,t,x,y,m;
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    cin>>t;
```

```
    while(t--){
      cin>>n;
      pair<ll,ll>p[n];
      ll sum=0;
      vector<ll> ans;
      for(int i=0;i<n;i++){
            cin>>p[i].first;
            p[i].second=i+1;
      }
      sort(p,p+n);
      for(int i=0;i<n;i++){
            if(sum<p[i].first) ans.clear();
            ans.push_back(p[i].second);
            sum+=p[i].first;
      }
      sort(ans.begin(), ans.end());
      cout<<ans.size()<<endl;
      for(int i=0;i<ans.size();i++){
            cout<<ans[i]<<" ";
      }
      cout<<endl;
    }
}
```

Решение Python

```
def win(pos : int, a : list):
    power = a[pos]
    for i in range(len(a)):
        if i == pos:
            continue
        if power < a[i]:
            return False
        power += a[i]
    return True

t = int(input())

while t > 0:
    t -= 1
    n = int(input())
    a = list(map(int, input().split(' ')))
    b = a.copy()
    a.sort()

    l = -1
    r = n - 1
    while r - l > 1:
        m = (l + r) // 2
        if (win(m, a)):
            r = m
        else:
            l = m

    winIds = []
    for i in range(n):
        if b[i] >= a[r]:
            winIds.append(i + 1)

    print(len(winIds))
    print(*winIds)
```

G Уравняй массив

Посчитаем величину cnt_x — сколько раз число x встречается в массиве a. Будем перебирать значение величины C и искать, какое минимальное количество ходов необходимо сделать, чтобы каждое число встречалось в массиве a либо 0 раз, либо C раз. Заметим, что если не существует такого числа y, что $cnt_y = C$, то такое значение C не будет давать минимальный ответ (потому что мы удалили лишние элементы).

Тогда для конкретного C ответ вычисляется следующим образом:

$$\sum_{cnt_x<C} cnt_x + \sum_{cnt_x \geq C} (cnt_x - C)$$

Так как количество кандидатов на значение C не больше, чем n, то данный метод работает за $\mathcal{O}(n^2)$.

Далее есть два пути оптимизации:

- можно рассмотреть только уникальные величины C (таких не больше, чем $\mathcal{O}(\sqrt{n})$), и получить решение за $\mathcal{O}(n\sqrt{n})$;
- можно отсортировать величины cnt_x и воспользоваться префиксными суммами, такое решение работает за $\mathcal{O}(n \log n)$ или за $\mathcal{O}(n)$ (если воспользоваться сортировкой подсчётом).

Решение C++

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

void solve() {
  int n;
  cin >> n;
  map<int, int> cnt;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x]++;
  }
  map<int, int> groupedByCnt;
  for (auto[x, y] : cnt) {
    groupedByCnt[y]++;
  }

  int res = n;
  int left = 0, right = n, rightCnt = (int) cnt.size();
  for (auto[x, y] : groupedByCnt) {
    res = min(res, left + right - rightCnt * x);
    left += x * y;
    right -= x * y;
    rightCnt -= y;
  }
  cout << res << endl;
}

int main() {
  int tests;
  cin >> tests;
  while (tests-- > 0) {
    solve();
  }
  return 0;
}
```

Решение Python

```python
import sys
input = sys.stdin.readline
from collections import Counter
```

```python
for _ in range(int(input())):
    n = int(input())
    A = list(map(int, input().split()))
    cnt = Counter(A)
    B = sorted(cnt.values(), reverse=True)
    ans = 0
    for i, b in enumerate(B):
        ans = max(ans, b * (i + 1))
    print(n - ans)
```

H Старый дисковод

Обозначим за S сумму всех элементов массива, а за $pref$ массив его префиксных сумм.

Если дисковод работает t секунд, то сумма равна $\left\lfloor \frac{t}{s} \right\rfloor \cdot S + pref[t \bmod n]$.

Из этой формулы сразу видно, что если $\max\limits_{i=0}^{n-1} pref[i] < x$ и $S \leq 0$, то диск будет работать бесконечно. В противном случае ответ существует.

Диск не может сделать меньше чем, $\left\lceil \frac{x - \max\limits_{i=0}^{n-1} pref[i]}{S} \right\rceil$ полных оборотов, иначе нужная сумма просто не наберется. Больше оборотов диск сделать тоже не может, так как дойдя до позиции максимальной префиксной суммы, x уже будет набран. Так что мы умеем определять количество полных оборотов диска. Сделаем эти обороты:

$$x := x - \left\lceil \frac{x - \max\limits_{i=0}^{n-1} pref[i]}{S} \right\rceil \cdot S$$

.

Теперь мы получили новую задачу, задан x найти первую позицию i в массиве, такую что $pref[i] \geq x$. Эту задачу можно решать с помощью бинарного поиска. Если в массив $pref$ не остортирован, то есть существует j, такое что $pref[j-1] > pref[j]$, то $pref[j]$ можно просто выкинуть из массива (на нем никогда не будет достигнут ответ).

Решение C++

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

using namespace std;
using ll = long long;

void solve() {
  int n, m;
  cin >> n >> m;
  vector<ll> a(n);
  ll allSum = 0;
  vector<ll> pref;
  vector<int> ind;
  int curInd = 0;
  for (ll &e : a) {
    cin >> e;
    allSum += e;
    if (pref.empty() || allSum > pref.back()) {
      pref.push_back(allSum);
      ind.push_back(curInd);
    }
    curInd++;
  }
  for (int q = 0; q < m; q++) {
```

```
    ll x;
    cin >> x;
    if (pref.back() < x && allSum <= 0) {
      cout << -1 << " ";
      continue;
    }
    ll needTimes = 0;
    if (pref.back() < x) {
      needTimes = (x - pref.back() + allSum - 1) / allSum;
    }
    x -= needTimes * allSum;
    cout << needTimes * n + ind[lower_bound(pref.begin(), pref.end(), x) -
pref.begin()] << " ";
  }
  cout << "\n";
}

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

Решение Python

```
from bisect import bisect_left
for i in range(int(input())):
      n, m = map(int,input().split())
      a = list(map(int,input().split()))
      p = [0]*(n+1)
      M = [0]*(n+1)
      for i in range(n):
            p[i+1] = p[i] + a[i]
            M[i+1] = max(M[i], p[i+1])
      s = p[-1]
      ans = []
      for x in map(int,input().split()):
            r = 0
            if s > 0:
                  t = max((x-M[-1]+s-1)//s,0)
                  r += t*n
                  x -= t*s
            if x > M[-1]:
                  ans.append('-1')
            else:
                  pos = bisect_left(M,x)
                  ans.append(str(r + pos - 1))
      print(' '.join(ans))
```