C. Pinkie Pie Eats Patty-cakes

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Pinkie Pie has bought a bag of patty-cakes with different fillings! But it appeared that not all patty-cakes differ from one another with filling. In other words, the bag contains some patty-cakes with the same filling.

Pinkie Pie eats the patty-cakes one-by-one. She likes having fun so she decided not to simply eat the patty-cakes but to try not to eat the patty-cakes with the same filling way too often. To achieve this she wants the minimum distance between the eaten with the same filling to be the largest possible. Herein Pinkie Pie called the distance between two patty-cakes the number of eaten patty-cakes strictly between them.

Pinkie Pie can eat the patty-cakes in any order. She is impatient about eating all the patty-cakes up so she asks you to help her to count the greatest minimum distance between the eaten patty-cakes with the same filling amongst all possible orders of eating!

Pinkie Pie is going to buy more bags of patty-cakes so she asks you to solve this problem for several bags!

**Input**

The first line contains a single integer TT (1≤T≤1001≤T≤100): the number of bags for which you need to solve the problem.

The first line of each bag description contains a single integer nn (2≤n≤1052≤n≤105): the number of patty-cakes in it. The second line of the bag description contains nn integers a1,a2,…,ana1,a2,…,an (1≤ai≤n1≤ai≤n): the information of patty-cakes' fillings: same fillings are defined as same integers, different fillings are defined as different integers. It is guaranteed that each bag contains at least two patty-cakes with the same filling.

It is guaranteed that the sum of nn over all bags does not exceed 105105.

**Output**

For each bag print in separate line one single integer: the largest minimum distance between the eaten patty-cakes with the same filling amongst all possible orders of eating for that bag.

**Example**

**input**

**Copy**

4

7

1 7 1 6 4 4 6

8

1 1 4 6 4 6 4 7

3

3 3 3

6

2 5 2 3 1 4

**output**

**Copy**

3

2

0

4

**Note**

For the first bag Pinkie Pie can eat the patty-cakes in the following order (by fillings): 11, 66, 44, 77, 11, 66, 44 (in this way, the minimum distance is equal to 33).

For the second bag Pinkie Pie can eat the patty-cakes in the following order (by fillings): 11, 44, 66, 77, 44, 11, 66, 44 (in this way, the minimum distance is equal to 22).

[1393C - Pinkie Pie Eats Patty-cakes](https://codeforces.com/contest/1393/problem/C)

Let's note that if you can find the arrangement with the maximum distance ≥X≥X, then you can also find the arrangement with the maximum distance ≥X−1≥X−1. It allows you to use the binary search on the answer.

To check that the answer is at least XX, we can use the greedy algorithm. Each time let's use the element that we can use (we didn't use it on the last x−1x−1 steps) and the number of the remaining elements equal to it is as large as possible. You can store the set of elements that you can use in the sorted by the number of appearances in std::setstd::set. This sol works O(nlog2n)O(nlog2n).

Also there is a solution in O(n)O(n).

Solution (AlFlen)

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

**using** **namespace** std;

**const** **int** MAXN = 100009;

**int** cnt[MAXN];

vector**<int>** a;

**int** n;

**bool** check(**int** x) {

**for** (**int** i = 1; i <= n; i ++) cnt[i] = 0;

**for** (**int** i = 0; i < n; i ++) cnt[a[i]]++;

**set**<pair<**int**, **int**>, greater<pair<**int**, **int**>>> ss; **//use greater comparator to sort set in descending order**

**for** (**int** i = 1; i <= n; i ++) {

**if** (cnt[i] > 0) ss.insert({cnt[i], i});

}

vector**<int>** b;

**for** (**int** i = 0; i < n; i ++) {

**if** (i >= x && cnt[b[i - x]]) {

ss.insert({cnt[b[i - x]], b[i - x]});

}

**if** (ss.empty()) **return** 0;

b.push\_back(ss.**begin**()->second);

ss.erase(ss.**begin**());

cnt[b.back()]--;

}

**return** 1;

}

**signed** main() {

ios :: sync\_with\_stdio(0);

cin.tie(0);

**int** ttt;

cin >> ttt;

**while** (ttt--) {

cin >> n;

a.resize(n);

**for** (**int** i = 0; i < n; i ++) {

cin >> a[i];

}

**int** l = 0, r = n;

**while** (r - l > 1) {

**int** m = (r + l) / 2;

**if** (check(m)) {

l = m;

}

**else** {

r = m;

}

}

cout << l - 1 << **"\n"**;

}

**return** 0;

}