Samajh to liye hai bas implement nhi kiye hai

Date

B. Andrey and Problem

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

Andrey needs one more problem to conduct a programming contest. He has n friends who are always willing to help. He can ask some of them to come up with a contest problem. Andrey knows one value for each of his fiends — the probability that this friend will come up with a problem if Andrey asks him.

Help Andrey choose people to ask. As he needs only one problem, Andrey is going to be really upset if no one comes up with a problem or if he gets more than one problem from his friends. You need to choose such a set of people that maximizes the chances of Andrey not getting upset.

Input

The first line contains a single integer n $(1 \leq n \leq 100)$ — the number of Andrey's friends. The second line contains n real numbers p_i $(0.0 \leq p_i \leq 1.0)$ — the probability that the i-th friend can come up with a problem. The probabilities are given with at most 6 digits after decimal point.

Output

Print a single real number — the probability that Andrey won't get upset at the optimal choice of friends. The answer will be considered valid if it differs from the correct one by at most 10^{-9}.

Let's sort all friends in such a way that $p_i \leq p_j$ iff $i \leq j$. If there is $p_i = 1$ Andrey should ask only this friend. Now we can assume that all probabilities are less then 1. What should we maximize?

$$\sum_{i=0}^{n-1} p_i \prod_{j \neq i}(1 - p_j) = \sum_{i=0}^{n-1} \frac{p_i}{1-p_i} \prod(1 - p_j) = \prod(1 - p_i) \cdot \sum \frac{p_i}{1-p_i}$$

Let $P = \prod(1 - p_i)$, $S = \sum \frac{p_i}{1-p_i}$. Assume we already have some group of people we would ask a help. Let's look what will happen with the probability of success if we add a friend with probability p_i to this group:

$$\Delta = -P \cdot S + P(1 - p_i) \cdot (S + \frac{p_i}{1-p_i}) = P \cdot p_i(1 - S)$$

It means adding a new people to group will increase a probability of success only if $S < 1$. Now let's look at another question. We have some group of people with $S < 1$. And we want to add only one friend to this group. Which one is better? Let the probability of the first friend is p_i and the second friend is p_j. It's better to add first one if

$\Delta_i - \Delta_j = P \cdot p_i \cdot (1 - S) - P \cdot p_j \cdot (1 - S) = P \cdot (1 - S) \cdot (p_i - p_j) > 0$. As $S < 1$ we get $p_i > p_j$.

But it's only a local criteria of optimality. But, we can prove that globally you should use only a group of people with the biggest probabilities. We can use proof by contradiction. Let's look at the optimal answer with biggest used suffix (in the begining of editorial we sort all friends). Of all such answers we use one with minimum number of people in it. Where are two friends i and j ($p_i < p_j$) and i-th friend is in answer and j-th isn't. Let's look at the answer if we exclude i-th friend. It should be smaller because we used optimal answer with minimum numer of people in it. So adding a new people to this group will increase success probability. But we know that adding j-th is better than i-th. So we have found a better answer.

So we have a very easy solution of this problem. After sorting probabilities we should you some suffix of it. Because of sorting time complexity is $O(nlogn)$.

J - Sushi

Editorial

Time Limit: 2 sec / Memory Limit: 1024 MB

Score : 100 points

Problem Statement

There are N dishes, numbered $1, 2, \ldots, N$. Initially, for each i ($1 \leq i \leq N$), Dish i has a_i ($1 \leq a_i \leq 3$) pieces of sushi on it.

Taro will perform the following operation repeatedly until all the pieces of sushi are eaten:

- Roll a die that shows the numbers $1, 2, \ldots, N$ with equal probabilities, and let i be the outcome. If there are some pieces of sushi on Dish i, eat one of them; if there is none, do nothing.

Find the expected number of times the operation is performed before all the pieces of sushi are eaten.

Constraints

- All values in input are integers.
- $1 \leq N \leq 300$
- $1 \leq a_i \leq 3$

// one of the big observation is you don’t have to mind in which outcome out of which 1..to ..N is coming instead you can mark you can divide in such a way a group to 0’s 1’s 2’s and 3’s.

Here dp[x][y][z] will represent the state where this state depends upon x no’s of 1’s y no of 2’s, z no of 3’s

Here p1=(x)/n; corresponding to number of which has exactly no of 1’s shushi’s are in plate
Here p2=(y)/n; corresponding to number of which has exactly no of 2’s shushi’s are in plate
Here p3=(z)/n; similarly

To calculate the no of zeros are (n-(x+y+z))
p0 = (n-(x+y+z))/n

So transition between dp state are as follow’s
dp[i][j][k]= p1* dp[i+1][j][k] + p2* dp[i-1][j+1][k] + p3* dp[i][j-1][k+1] + **p0*dp[i][j][k];**

since the dp state depends upon own state so we can’t do the same
(1-p0)dp[i][j][k]= p1* dp[i+1][j][k] + p2* dp[i-1][j+1][k] + p3* dp[i][j-1][k+1]

or , dp[i][j][k]= (p1* dp[i+1][j][k] + p2* dp[i-1][j+1][k] + p3* dp[i][j-1][k+1])/(1-p0);

```
ll dp[301][301][301];
ll solve(int one, int two, int three, int& n){

    if(one < 0 || two < 0 || three < 0)
        return 0;
    if(three == 0 && two == 0 && one == 0)
        return 0;

    if(dp[one][two][three] > 0)
        return dp[one][two][three];

    int remaining = one + two + three;
    ll exp_val = n+one*solve(one-1,two,three,n) +   two*solve(one+1,two-1,three,n) +
                    three*solve(one,two+1,three-1,n);

    return dp[one][two][three] = exp_val/remaining;

}
```

```
const int MX=305;
double dp[MX][MX][MX];

int main() {
  int n;
  scanf("%d",&n);
  vi a(3);
  rep(i,n) {
    int x;
    scanf("%d",&x);
    a[x-1]++;
  }
  double p = 1.0/n;
  for(int k=0;k<n+1;k++)
      for(int j=0;j<n+1;j++)
        for(int i=0;i<n+1;i++) {
             int z = n-i-j-k;
             if (z < 0) break;
             if (z == n) continue;
             double x = 1-z*p;
             if (i) dp[i][j][k] += dp[i-1][j][k]*i*p;
             if (j) dp[i][j][k] += dp[i+1][j-1][k]*j*p;
             if (k) dp[i][j][k] += dp[i][j+1][k-1]*k*p;
             dp[i][j][k] += 1;
             dp[i][j][k] /= x;
  }
  printf("%.10f\n",dp[a[0]][a[1]][a[2]]);
  return 0;
}
```