ADS期末辅学题目集——程序填空题与编程题

程序填空题

5-1 The function `BinQueue_Merge` is to merge two binomial queues `H1` and `H2`, and return `H1` as the resulting queue.

```
BinQueue BinQueue_Merge( BinQueue H1, BinQueue H2 ) {
    BinTree T1, T2, Carry = NULL;
    int i, j;
    H1->CurrentSize += H2-> CurrentSize;
    for ( i=0, j=1; j<= H1->CurrentSize; i++, j*=2 ) {
        T1 = H1->TheTrees[i]; T2 = H2->TheTrees[i];
        switch( __________(3分) ) {
        case 0:
        case 4: break;
        case 3:
        case 7:
                __________(3分);
                H2->TheTrees[i] = NULL; break;
        case 1:
                H1->TheTrees[i] = Carry; Carry = NULL; break;
        case 2:
                H1->TheTrees[i] = T2; H2->TheTrees[i] = NULL; break;
        case 5:
                Carry = CombineTrees( T1, Carry );
                H1->TheTrees[i] = NULL; break;
        case 6:
                Carry = CombineTrees( T1, T2 );
                H1->TheTrees[i] = H2->TheTrees[i] = NULL; break;
        } /* end switch */
    } /* end for-loop */
    return H1;
}
```

5-2 Suppose we are given n points $p_1, p_2, ... p_n$ located on the x-axis. x_i is the x-coordinate of p_i. Let us further assume that $x_1 = 0$, and the points are given from left to right. These n points determine $\frac{n(n-1)}{2}$ (not-necessarily unique) distances $d_1, d_2, ... d_{n(n-1)/2}$ between every pair of points of the form $|x_i - x_j|$ $(i \neq j)$.

The *Turnpike reconstruction problem* is to reconstruct a point set from the distances.

This algorithm is to read the number n and $\frac{n(n-1)}{2}$ distances d_i, then print one valid sequence of points p_i. Please complete the following program.

```
#include <algorithm>
#include <cstdio>
const int MAXN = 1000, MAXD = MAXN * (MAXN - 1) / 2;
int p[MAXN], d[MAXD], n, m;
int id[MAXD];
bool used[MAXD];
int binary_search(int x, int m) {
    int l = 0, r = m;
    while (l < r) {
        int mid = (l + r) / 2;
        if (d[mid] < x || (d[mid] == x && used[mid]))
            __________(2分);
        else
            r = mid;
    }
    return l;
}
bool recursive(int now, int top, int m) {
    int i;
    for (i = 0; i < now; i++) {
        id[top + i] = binary_search(abs(p[i] - p[now]), m);
        if(__________(2分) && !used[id[top + i]])
            used[id[top + i]] = true;
        else break;
    }
    if (i == now) {
        if (now == n - 1)
            return true;
        while (used[m - 1])
            m--;
        p[now + 1] = d[m - 1];
        if (recursive(now + 1, top + now, m))
            return true;
        if (now <= 1)
            return false;
        p[now + 1] = __________(2分);
        if (recursive(now + 1, top + now, m))
            return true;
    }
    for(int j = 0; j < i; j++)
        __________(2分);
    return false;
}
int main()
{
    scanf("%d", &n);
    m = n * (n - 1) / 2;
    for (int i = 0; i < m; i++)
        scanf("%d", &d[i]);
    std::sort(d, d + m);
    p[0] = 0;
    if (!recursive(__________(2分))) {
        puts("NO ANSWER");
        return 0;
    }
    std::sort(p, p + n);
    for (int i = 0; i < n; i++)
        printf("%d\n", p[i]);
    return 0;
}
```

5-3 An n-digit number that is the sum of the n-th powers of its digits is called an n-narcissistic number. For example, 153 is a 3-narcissistic number since $153 = 1^3 + 5^3 + 3^3$, and 1634 is a 4-narcissistic number since $1634 = 1^4 + 6^4 + 3^4 + 4^4$.

Please complete the following program that prints the sum of all n-narcissistic numbers.

```
#include <cstdio>
long long cost[10], ans;
int cnt[10], t[10], n;
void dfs(int rest, int now, long long current) {
    if (rest == 0) {
        long long temp = current;
        for (int i = 0; i < 10; i++)
            t[i] = 0;
        while (temp > 0) {
            ++t[temp % 10];
            temp /= 10;
        }
        bool flag = 1;
        for (int i = 0; i < 10; i++)
            if (cnt[i] != t[i]) {
                flag = 0;
                break;
            }
        if (flag) {
            __________(2分);
        }
        return;
    }
    if (__________(2分)) {
        return;
    }
    for (cnt[now] = 0; cnt[now] <= rest; cnt[now]++)
        dfs(rest - cnt[now], now + 1, __________(2分));
    __________(2分) = 0;
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < 10; i++) {
        cost[i] = 1;
        for (int j = 0; j < n; j++)
            cost[i] *= i;
    }
    dfs(__________(2分));
    printf("%lld\n",ans);
    return 0;
}
```

5-4 In a permutation A, if there exists a pair of numbers A_i and A_j satisfied $i < j$ and $A_i > A_j$, then A_i and A_j are called an inverted pair.

Giving a permutation of N, please find the number of the inverted pairs in it.

Hint: The function $work(l, r)$ returns the number of inverted pairs in $A_l A_{l+1} \dots A_r$ and makes $A_l A_{l+1} \dots A_r$ being sorted in increasing order.

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int tmp[N], a[N];
long long work(int l, int r) {
    if (l == r)
        return 0;
    int mid = (l + r) >> 1;
    long long res = 0;
    res += work(l, mid);
    res += __________(2分);
    int t1 = l, t2 = mid + 1, tt = l;
    while __________(2分){
        if (a[t1] < a[t2])
            tmp[tt++] = a[t1++];
        else {
            res += __________(3分);
            tmp[tt++] = a[t2++];
        }
    }
    while (t1 <= mid)
        tmp[tt++] = a[t1++];
    while (t2 <= r)
        tmp[tt++] = a[t2++];
    for (int i = l; i <= r; i++)
        __________(2分);
    return res;
}
int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    printf("%lld\n", __________(1分));
    return 0;
}
```

5-5 Giving an array of N integers, please calculate the maximum subsegment sum.

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int a[N];
long long work(int l, int r) {
    if (l == r)
        return __________(2分);
    int mid = (l + r) >> 1;
    long long res = 0;
    res = max(res, work(l, mid));
    res = __________(2分);
    long long mxl = 0, suml = 0, mxr = 0, sumr = 0;
    for (int i = mid; i >= l; i--) {
        suml += a[i];
        mxl = max(mxl, suml);
    }
    for (int i = mid + 1; i <= r; i++) {
        __________(2分);
        __________(2分);
    }
    res = __________(2分);
    return res;
}
int main()
{
    int T;
    scanf("%d",&T);
    while (T--) {
        int n;
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        printf("%lld\n", work(1, n));
    }
    return 0;
}
```

编程题

7-1 **Hand-made Cream** (8分)

Suppose you are a baker planning to bake some hand-made cream breads.

To bake a cream bread, we need to use one slice of bread and one kind of cream. Each hand-made cream bread has a taste score to describe how delicious it is, which is obtained by multiplying the taste score for bread and the taste score for cream. (The taste scores could be negative, howerver, two negative tast scores can still produce delicious cream bread.)

The bread and cream are stored separately.

The constraint is that you need to examine the breads in a given order, and for each piece of bread, you have to decide immediately (without looking at the remaining breads) whether you would like to take it.

After you are finished with breads, you will take the same amount of cream in the same manner. The breads and creams you take will be paired in the same order as you take them.

Given N taste scores for bread and M taste scores for cream, you are supposed to calculate the maximum total taste scores for all hand-made cream bread.

Input Specification:

Each input file contains one test case. For each case, the first line contains two integers N and M ($1 \le N, M \le 10^3$), which are the numbers of bread and cream, respectively.

The second line gives N taste scores for bread.

The third line gives M taste scores for cream.

The taste scores are integers in $[-10^3, 10^3]$.

All the numbers in a line are separated by a space.

Output Specification:

Print in a line the maximum total taste score.

Sample Input:

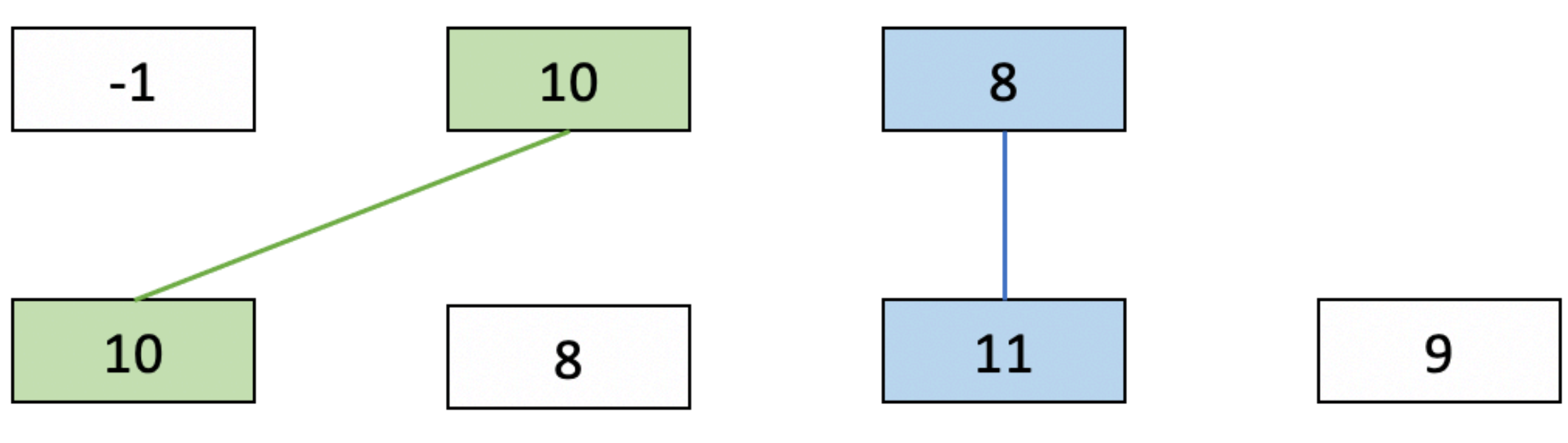
```
3 4
-1 10 8
10 8 11 9
```

Sample Output:

```
188
```

Hint:

The maximum total taste score for the sample case is $10 \times 10 + 8 \times 11 = 188$.



7-2 **Bonus(I). Binary String** (50分)

Bonus(I). Binary String

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a string s consisting of characters `0` and/or `1`.

You have to remove several (possibly zero) characters from the beginning of the string, and then several (possibly zero) characters from the end of the string. **The string may become empty after the removals.** The cost of the removal is the **maximum** of the following two values:

- the number of characters 0 left in the string;
- the number of characters 1 removed from the string.

What is the **minimum** cost of removal you can achieve?

Input

The first line contains one integer t ($1 \le t \le 10^4$) --- the number of test cases.

Each test case consists of one line containing the string s ($1 \le |s| \le 2 \cdot 10^5$), consisting of characters `0` and/or `1`.

The total length of strings `s` in all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print one integer --- the minimum cost of removal you can achieve.

Example

input

```
5
101110110
1001001001001
0000111111
00000
1111
```

output

```
1
3
0
0
0
```

Note

Consider the test cases of the example:

1. in the first test case, it's possible to remove two characters from the beginning and one character from the end. Only one `1` is deleted, only one `0` remains, so the cost is 1;
2. in the second test case, it's possible to remove three characters from the beginning and six characters from the end. Two characters `0` remain, three characters `1` are deleted, so the cost is 3;
3. in the third test case, it's optimal to remove four characters from the beginning;
4. in the fourth test case, it's optimal to remove the whole string;
5. in the fifth test case, it's optimal to leave the string as it is.

Clarifications

- You only have to minimize the cost. You don't have to make the string empty (but you are allowed to do it), you don't have to make the string consist of only equal characters (but you are allowed to do it), et cetera.
- The cost is the maximum of the number of 0s remaining in the string and the number of 1s removed from the string.
- If you remove no characters (leave the string as it is), the cost is NOT necessarily zero.

7-3 **H1. Maximum Crossings (Easy Version)** (50分)

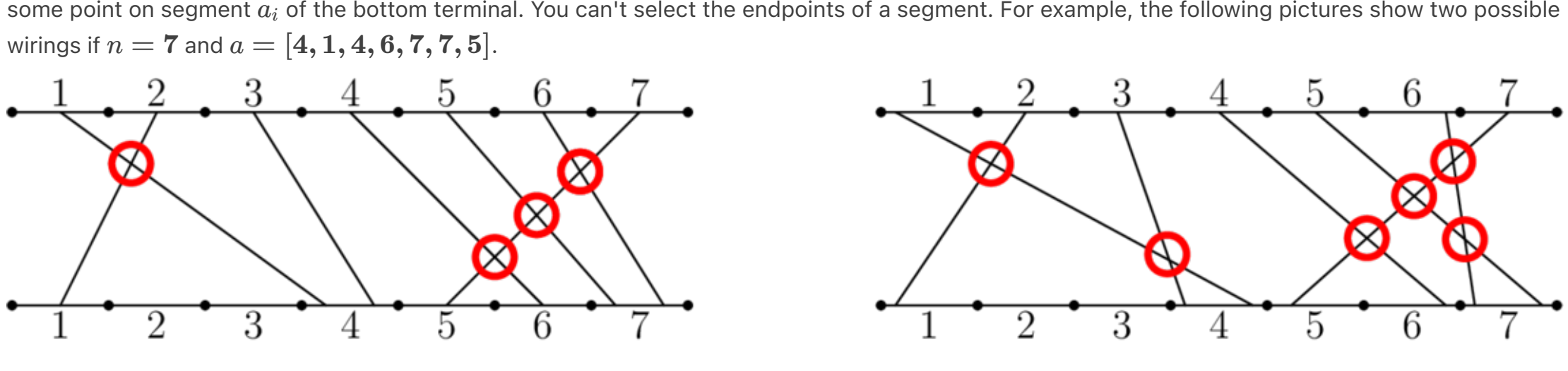
H1. Maximum Crossings (Easy Version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The only difference between the two versions is that in this version $n \le 1000$ and the sum of n over all test cases does not exceed 1000.

A *terminal* is a row of n equal segments numbered 1 to n in order. There are two terminals, one above the other.

You are given an array a of length n. For all $i = 1, 2, \dots, n$, there should be a straight wire from some point on segment i of the top terminal to some point on segment a_i of the bottom terminal. You can't select the endpoints of a segment. For example, the following pictures show two possible wirings if $n = 7$ and $a = [4, 1, 4, 6, 7, 7, 5]$.



A *crossing* occurs when two wires share a point in common. In the picture above, crossings are circled in red.

What is the **maximum** number of crossings there can be if you place the wires optimally?

Input

The first line contains an integer t ($1 \le t \le 1000$) --- the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 1000$) --- the length of the array.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) --- the elements of the array.

The sum of n across all test cases does not exceed 1000.

Output

For each test case, output a single integer --- the **maximum** number of crossings there can be if you place the wires optimally.

Example

input

```
4
7
4 1 4 6 7 7 5
2
2 1
1
1
3
2 2 2
```

output

```
6
1
0
3
```

Note

The first test case is shown in the second picture in the statement.

In the second test case, the only wiring possible has the two wires cross, so the answer is 1.

In the third test case, the only wiring possible has one wire, so the answer is 0.

7-4 **H2. Maximum Crossings (Hard Version)** (80分)

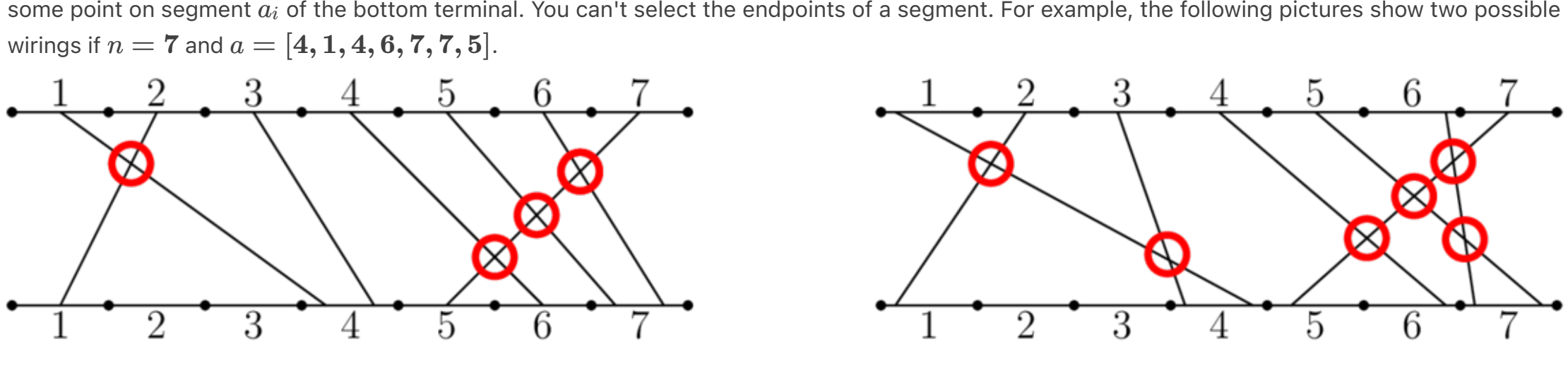
H2. Maximum Crossings (Hard Version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The only difference between the two versions is that in this version $n \le 2 \cdot 10^5$ and the sum of n over all test cases does not exceed $2 \cdot 10^5$.

A *terminal* is a row of n equal segments numbered 1 to n in order. There are two terminals, one above the other.

You are given an array a of length n. For all $i = 1, 2, \dots, n$, there should be a straight wire from some point on segment i of the top terminal to some point on segment a_i of the bottom terminal. You can't select the endpoints of a segment. For example, the following pictures show two possible wirings if $n = 7$ and $a = [4, 1, 4, 6, 7, 7, 5]$.



A *crossing* occurs when two wires share a point in common. In the picture above, crossings are circled in red.

What is the **maximum** number of crossings there can be if you place the wires optimally?

Input

The first line contains an integer t ($1 \le t \le 1000$) --- the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) --- the length of the array.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) --- the elements of the array.

The sum of n across all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer --- the **maximum** number of crossings there can be if you place the wires optimally.

Example

input

```
4
7
4 1 4 6 7 7 5
2
2 1
1
1
3
2 2 2
```

output

```
6
1
0
3
```

Note

The first test case is shown in the second picture in the statement.

In the second test case, the only wiring possible has the two wires cross, so the answer is 1.

In the third test case, the only wiring possible has one wire, so the answer is 0.