Mid-Sem Lab Exam
- Question 1 - “Voting”
  - Problem Description
  - Input constraints
  - Input format
  - Output Format
  - Sample input and output
  - Solution
- Question 2 - “Jump Back”
  - Problem Description
  - Input constraints
  - Input format
  - Output Format
  - Sample input and output
  - Solution
- Question 3 - “Max AND”
  - Problem Description
  - Input constraints
  - Input format
  - Output Format

Sample input and output

Solution

Mid-Sem Lab Exam

Question 1 - “Voting”

Problem Description

The Elections for the new academic year are in full swing!

There are N students in the college and two candidates for the current elections (where N is guaranteed to be an odd number). The students cast their votes in the form of $0s$ or $1s$ where the i^{th} vote ($1 <= i <= N$) being 1 represents a vote to candidate A and it being 0 represents a vote for candidate B.

A candidate wins the elections if they have the strict majority. That is, if the number of votes cast for them is strictly greater than the number of votes cast for the other candidate.

Output `Yes` if candidate A wins the elections and `No` otherwise.

Link to problem on OJ

Input constraints

- $1 \leq N \leq 99$
- $N \text{ is an odd number}$
- $V_i = \{0, 1\}$

Input format

The first line of input contains N , the number of students. Then, N line follows.

The i^{th} line contains a single integer $V_i = \{0, 1\}$ where 1 represents a vote for candidate A and 0 represents a vote for candidate B.

Output Format

Output `Yes` if candidate A wins the elections and `No` otherwise.

Sample input and output

Sample Input	Sample Output
3 1 0 1	Yes

Explanation: There are 2 votes for candidate A and 1 vote for candidate B. Therefore, candidate A has the majority and wins the elections.

Sample Input	Sample Output
5 0 0 1 0 1	No

Explanation: There are 2 votes for candidate A and 3 votes for candidate B. Therefore, candidate B has the majority and wins the elections.

Solution

```c
#include <stdio.h>

int main() {
    int n; scanf("%d", &n);
    int a = 0, b = 0;
    for (int i = 0; i < n; i++) {
        int x; scanf("%d", &x);
        if (x) a++;
        else b++;
    }
    if (a > b) printf("Yes\n");
    else printf("No\n");
    return 0;
}
```

Question 2 - “Jump Back”

Problem Description

There are N trees in the forest, numbered Tree 1, Tree 2,..., Tree N .

Tree 1, is a special tree which is home to the entire forest. Each tree i ($2 \leq i \leq N$), has a parent tree T_i from which it grew. It is guaranteed that $T_i < i$.

You are currently on tree N and want to reach tree 1 . However, from a given tree, you can only jump to its parent tree (unless you are already on tree 1 , in which case you have already reached the destination).

For example, consider the input $[1124]$ which means, $T_2 = 1, T_3 = 1, T_4 = 2, T_5 = 4$ (note that tree 1 has no parent tree). Starting from tree 5 , the only option is to jump to tree 4, its parent. Similarly, from tree 4, you jump to tree 2. Finally, from tree 2 you jump to tree 1, taking a total of 3 jumps to go from tree 5 to tree 1.

Find the number of jumps that you need to make in order to reach tree 1 starting from tree N . It can be shown that you can reach it in a finite number of jumps.

Link to problem on OJ

Input constraints

- $2 \leq N \leq 50$
- $1 \leq T_i < i \, (\, 2 \leq i \leq N \,)$

Input format

The first line of input contains a single integer N , the number of trees.

The following line contains $N - 1$ space separated integers, $T_2, T_3, \ldots, T_N$ where T_i denotes the parent of the i^{th} tree.

Output Format

Output a single integer denoting the number of jumps required to go from tree N to tree 1 .

Sample input and output

Sample Input	Sample Output
3 1 2	2

Sample Input	Sample Output
5 1 1 2 4	3

Solution

```c
#include <stdio.h>

int main() {
    int n; scanf("%d", &n);
    int a[n];
    a[0] = -1;
    for (int i = 1; i < n; i++) {
        scanf("%d", &a[i]);
        a[i]--;
    }
    int curr = n - 1, ans = 0;
    while (curr != -1) {
        ans++;
        curr = a[curr];
    }
    printf("%d\n", ans - 1);
    return 0;
}
```

Question 3 - “Max AND”

Problem Description

In this problem, we use the symbol $\wedge$ to denote the **bitwise AND** operation, and the symbol $\vee$ to denote the **bitwise OR** operation.

Given an array of integers A of length N, $A_1, A_2, \ldots, A_N$. Define a "set operation" as the following:

- Select any index i where $1 \leq i \leq N$ and set the j^{th} bit of A_i where j is any integer between 0 and 30 inclusive. In other words, replace A_i with $A_i \vee 2^j$.

You are also given a non-negative integer K. You can perform **at most** K operations on the given array. After doing so, output the maximum possible value of $A_1 \wedge A_2 \wedge \ldots \wedge A_N$.

Link to problem on OJ

Input constraints

- $1 \leq N \leq 2 \times 10^5$
- $0 \leq K \leq 10^9$
- $0 \leq A_i < 2^{31}$

Input format

The first line of input contains two space separated integers N and K, denoting the length of the array and the maximum number of operations that can be performed respectively.

The second line contains N space separated integers $A_1, A_2, \ldots, A_N$ that denote the initial array.

Output Format

Output the maximum possible value of the bitwise AND of the entire array after performing at most K operations.

Sample input and output

Sample Input	Sample Output
3 2 2 1 1	2

Explanation: We set the $j = 1$ bit (that is, the bit corresponding to 2^1) for A_2 and A_3 . This requires $2 \leq K$ operations. After doing so, the array now looks like $[2, 3, 3]$, giving us a bitwise AND value of 2 which can be shown is the maximum possible value for the given input.

Sample Input	Sample Output
7 0	
4 6 6 28 6 6 12	4

Explanation: Here, $K = 0$. So, we cannot perform any operations. Hence, the bitwise AND of the initial array, 4 is our final answer.

Solution

```c
#include <stdio.h>

int main() {
    int n, k; scanf("%d %d", &n, &k);
    int cnt[31] = {0};
    for (int i = 0; i < n; i++) {
        int x; scanf("%d", &x);
        for (int j = 0; j < 31; j++, x /= 2) {
            cnt[j] += (x & 1);
        }
```

```
    }
    int ans = 0;
    for (int i = 30; i >= 0; i--) {
        if (k >= n - cnt[i]) {
            ans |= (1 << i);
            k -= (n - cnt[i]);
        }
    }
    printf("%d\n", ans);
    return 0;
}
```