Firework Show

```
#include<stdio.h>
#include<string.h>

int all_combinations[400][21];

/*
Input

The first line contains an integer T, representing the number of testcases.

Each testcase contains two lines :

The first line contains two integer n, m, representing the number of fireworks and the supreme
leader's request.

The second line contains n integer ai, representing the beauty grade of each firework.

It is guaranteed that :

1 ≤ T ≤ 10
1 ≤ n ≤ 20
1 ≤ m ≤ 50000
1 ≤ ai ≤ 2000

Output

For each testcase, please output a line contains one integer represents the number of different
kinds of shows that meet the supreme leader’s demanding request.


Sample Input

3
3 5
1 2 3
6 3
1 1 1 1 1 1
6 6
1 2 3 1 2 3

Sample Output

1
1
3
*/

void dump(int data[], int size) {
    printf("DEBUG: ");
    for (int i=0; i<size; i++) printf("%d ", data[i]);
    printf("\n");
}

void sort(int data[], int size) {
    for (int i=0; i<size-1; i++) {
        for (int j=0; j<size-1; j++) {
            if (data[j] > data[j+1]) {
                int tmp = data[j+1];
                data[j+1] = data[j];
                data[j] = tmp;
            }
        }
    }
}

int update_all_combinations(int data[], int size, int total) {
    int updated;

    sort(data, size);

    if (total == 0) {
        all_combinations[0][0] = size;
        for (int k=0; k<size; k++) all_combinations[0][k+1] = data[k];
        updated = 1;
        return updated;
    }
    else {
        for (int i=0; i<total; i++) {
            if (all_combinations[i][0] == size) {
                int duplicate = 1;
                for (int j=0; j<size; j++) {
                    if (all_combinations[i][j+1] == data[j]) continue;
                    else duplicate = 0;
                }

                if (duplicate) {
                    updated = 0;
                    return updated;
                }
                else {
                    // DO NOTHING
                }
            }
            else {
                // DO NOTHING
            }
        }

        // found new combination
        //printf("DEBUG: found new combination\n");
        //dump(data, size);
        all_combinations[total][0] = size;
        for (int j=0; j<size; j++) {
            all_combinations[total][j+1] = data[j];
        }
        updated = 1;
        return updated;
    }

    return updated;
}

// =============================================
// Source: https://ide.geeksforgeeks.org/index.php

void combinationUtil(int arr[], int data[], int start, int end, int index, int r, int m, int* result);

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r, int m, int* result)
{
    // A temporary array to store all combination one by one
    int data[r];

    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, data, 0, n-1, 0, r, m, result);
}

/* arr[] ---> Input Array
data[] ---> Temporary array to store current combination
start & end ---> Staring and Ending indexes in arr[]
index ---> Current index in data[]
r ---> Size of a combination to be printed */
void combinationUtil(int arr[], int data[], int start, int end, int index, int r, int m, int* result)
{
    // Current combination is ready to be printed, print it
    if (index == r)
    {
        int sum = 0;
        for (int j=0; j<r; j++) {
            sum += data[j];
        }
        //printf("DEBUG: sum = %d\n", sum);

        if ( sum == m ) {
            //printf("DEBUG: matched\n");
            //dump(data, r);
            //printf("DEBUG: old result = %d\n", *result);
            if (update_all_combinations(data, r, *result)) {
                *result = *result + 1;
                //printf("DEBUG: new result = %d\n", *result);
            }
        }
        return;
    }

    // replace index with all possible elements. The condition
    // "end-i+1 >= r-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r, m, result);
    }
}

// ================================================

int fab(int n) {
    if (n == 0) return 1;
    else if (n == 1) return 1;
    else return n*fab(n-1);
}

int c_n_r(int n, int r) {
    int c = fab(n)/(fab(n-r)*fab(r));
    //printf("DEBUG: c = %d, n = %d, r = %d\n", c, n ,r);
    return c;
}

void run_testcase(int n, int m, int data[]) {
    //printf("DEBUG: m = %d\n", *m);

    int result = 0;

    for (int r=1; r<=n; r++) {
        //int c = c_n_r(n, r);
        //int all_combinations[c][r];
        printCombination(data, n, r, m, &result);
    }

    printf("%d\n", result);
}

void load_testdata(int *T, int *n, int *m, int data[]) {
    scanf("%d", T);
    //printf("DEBUG: T = %d\n", *T);

    for (int t=0; t<*T; t++) {
        //printf("DEBUG: case %d\n", t+1);

        scanf("%d", n);
        //printf("DEBUG: n = %d\n", *n);

        scanf("%d", m);
        //printf("DEBUG: m = %d\n", *m);

        for (int i=0; i<*n; i++) {
            scanf("%d", &data[i]);
            //printf("DEBUG: i = %d, a[i] = %d\n", i, data[i]);
        }

        run_testcase(*n, *m, data);
    }
}

int main(){
    int T_MIN = 1;
    int T_MAX = 10;
    int n_MIN = 1;
    int n_MAX = 20;
    int m_MIN = 1;
    int m_MAX = 50000;
    int ai_MIN = 1;
    int ai_MAX = 2000;

    int T;
    int n;
    int m;
    int data[n_MAX];

    load_testdata(&T, &n, &m, data);
    return 0;
}
```

Flattening the tree

```
#include <stdio.h>
#include <string.h>
int n;
int a[2000];
int b[2000];

void InorderTraversal(int index) {

  if (index <= n) {
    int leftNode = index*2;
    int rightNode = index*2+1;

    InorderTraversal(leftNode);
    if (index == n){
      printf("%d\n", a[index]);
    }
    else{
      printf("%d ", a[index]);
    }
    InorderTraversal(rightNode);
  }
}

int main()
{
  scanf("%d", &n);
  for(int i=1; i<=n; i++)
  {
    scanf("%d", &a[i]);
  }
  int index = 1;
  InorderTraversal(index);

  // for(int i=0; i<n; i++)
  // {
  //   printf("%d ", a[i]);
  // }
}
```

Big Mod

```
#include <stdio.h>
int T;
int a, n, p;
unsigned long int result;

int multiply(unsigned long int result, int n)
{
  if(n>1)
  {
    result = result * multiply(result, n-1);
  }
  return result;
}

int main()
{
  scanf("%d", &T);
  for(int i=0; i<T; i++)
  {
    scanf("%d %d %d", &a, &n, &p);
    //result = a*a*a*a*a*a*a*a*a*a = a^n
    result = a;
    result = multiply(result, n);
    //printf("%d\n", result);
    result = result % p;
    printf("%lu\n", result);
  }
  return 0;
}
```