Problem Statement 1

Reena is studying number theory and is fascinated by the concept of the Least Common Multiple (LCM) of three numbers. She wants to write a program to find the LCM of three numbers using **recursion**.

Reena seeks your assistance to guide the development of the code.

Function Specifications: int find_lcm(int a, int b, int c);

Input format :

The input consists of three integers n1, n2, and n3, separated by a space.

Output format :

The output displays a single integer, which represents the LCM (Least Common Multiple) of the given input.

Refer to the sample output for the formatting specifications.

Code constraints :

In the given scenario, the test cases will fall under the following constraints:

1 <= n1, n2, n3 <= 50

Sample test cases :

Input 1 :

10 15 20

Output 1 :

60

Input 2 :

5 7 3

Output 2 :

105

Input 3 :

48 50 49

Output 3 :

58800

```
#include<iostream>
using namespace std;

int find_gcd(int a, int b){
  if(b == 0){
    return a;
  }

  return find_gcd(b, a%b);
}

int find_lcm_of_two(int a, int b){
  return (a*b)/ find_gcd(a,b);
}

int find_lcm(int a, int b, int c){
  int lcm_ab = find_lcm_of_two(a,b);

  return find_lcm_of_two(lcm_ab, c);
}

int main(){
  int a, b, c;
  cin>>a>>b>>c;

  cout<<find_lcm(a, b, c)<<endl;

  return 0;
}
```

Problem Statement 2

You are given T test cases, each consisting of two integers, 'a' and 'b'. Your task is to calculate 'a' raised to the power of 'b' and output the result for each test case.

You need to implement the **power()** function that efficiently calculates the modular exponentiation of 'a' raised to the power of 'b', using recursion and modular arithmetic.

Note: This kind of question will be helpful in clearing Infosys recruitment.

Input format :

The first line of input consists of an integer **T**, representing the number of test cases.

Each of the following **T** lines contains two space-separated integers **a** and **b**, where **a** is the base, and **b** is the exponent.

Output format :

For each test case, the output prints a single integer on a new line, representing the result of 'a' raised to the power of 'b'.

Code constraints :

$1 \leq T \leq 10$

$0 \leq a \leq 200$

$0 \leq b \leq 10$

Sample test cases :

Input 1 :

2

15 5

100 5

Output 1 :

759375

10000000000

Input 2 :

1

7 5

Output 2 :

16807

```
#include<iostream>
using namespace std;

int calculate_Power(int a, int b){
   if (b == 0)
      return 1;
   return a * calculate_Power(a, b - 1);
}

int main(){
   int n;
   cin >> n;

   int a, b;

   for(int i = 0; i < n; i++){
      cin >> a >> b;
      cout << calculate_Power(a, b) << endl;
   }

   return 0;
}
#include<iostream>
using namespace std;

long long calculate_Power(long long a, long long b){
  if( b == 0){
   return 1;
  }
```

```
 long long half = calculate_Power(a, b/2);

 if(b %2 == 0){
  return half * half;
 }else{
  return a * half * half;
 }
}

int main(){
 int n;
 cin>>n;

 long long a,b;

 for(int i=0; i<n; i++){
  cin>>a>>b;
  cout << calculate_Power(a, b) << endl;
 }

 return 0;
}
```

Problem Statement 3

You work in a large warehouse where products are numbered from 0 to N-1. Your task is to identify any products that appear twice in the inventory list and report them in ascending order. This helps ensure accurate inventory management.

For instance, if you have N = 4 products with IDs {0, 1, 3, 2}, and no duplicates exist, you report -1. However, if you have N = 5 products with IDs {1, 1, 2, 2, 3}, you must report the duplicates in

ascending order: 1 2. This ensures your warehouse maintains efficient inventory control. The input products must be listed in ascending order.

Now design a program using recursion to solve the given problem statement.

Input format :

The first line contains an integer N, the size of the array.

The second line contains N space-separated integers a[0], a[1], ..., a[N-1], representing the elements in the array in ascending order.

Output format :

The output should be a space-separated list of integers.

If there are elements that occur twice in the given array, list them in ascending order.

If no such elements are found, the output should be "-1" to indicate that there are no duplicates.

Refer to the sample output for exact format specifications.

Code constraints :

1 <= N <= 25

The input should be in ascending order.

Sample test cases :

Input 1 :

4

0 1 2 3

Output 1 :

-1

Input 2 :

5

1 1 2 2 3

Output 2 :

1 2

```
#include<iostream>

#include<vector>

using namespace std;
```

```
void find_duplicates(int arr[], int n, int i, vector<int>& duplicates){

 if( i >= n-1){
  return;
 }

 if(arr[i] == arr[i+1]){
  if(duplicates.empty() || duplicates.back() != arr[i]){
   duplicates.push_back(arr[i]);
  }
 }

 find_duplicates(arr, n, i+1, duplicates);
}

int main(){
 int n;
 cin>>n;

 int arr[n];
 for(int i=0; i<n; i++){
  cin>>arr[i];
 }

 vector<int> duplicates;
 find_duplicates(arr, n, 0, duplicates);

 if(duplicates.empty()){
  cout<<"-1";
 }else{
  for(int x : duplicates){
```

```
    cout<<x<<" ";
  }
 }

 return 0;

}
```

Problem Statement 4

In this mystical kingdom, there were three enchanted pegs: the Source Peg, the Destination Peg and the Helper Peg. Each peg had unique magical properties essential for saving the kingdom from an evil sorceress's destructive curse.

To counter this spell and protect the kingdom, they had to transfer magical disks of varying sizes, represented by positive integers, from the Source Peg to the Destination Peg using the Helper Peg. Notably, each disk was smaller than the one beneath it.

According to the ancient legends, the Sorceress' curse could be broken only if the disks were transferred from the Source Peg to the Destination Peg following these rules:

1. Only one disk could be moved at a time.
2. A larger disk could never be placed on top of a smaller disk.
3. The Helper Peg could be used temporarily for disk movements.

The kingdom's wise sage had devised a solution using the Tower of Hanoi algorithm, a mysterious ritual that required the minimum number of moves to transfer all the magical disks from the Source Peg to the Destination Peg.

Write a program to calculate the minimum number of moves required to solve the Tower of Hanoi problem for **n** magical disks, thus breaking the Sorceress' curse and saving the kingdom from impending doom.

Note: This kind of question will be helpful in clearing Capgemini recruitment.

Input format :

The input consists of an integer denoting the number of disks **n**.

Output format :

The output displays the minimum number of moves required to solve the Tower of Hanoi problem for **n** disks.

Code constraints :

$0 < n \leq 8$

Sample test cases :

Input 1 :

2

Output 1 :

3

Input 2 :

4

Output 2 :

15

```
#include<iostream>
using namespace std;

void towerOfHanoi(int n, char source, char helper, char destination, int&count){
 if(n == 1){
  count++;
  return;
 }
 towerOfHanoi(n-1, source, destination, helper, count);
 count++;
 towerOfHanoi(n-1, helper, source, destination, count);
}

int main(){
 int n;
 cin>>n;
```

```
 int count =0;
 towerOfHanoi(n, 'A', 'B', 'C', count);
 cout<<count;
 return 0;
}
```

Problem Statement 5

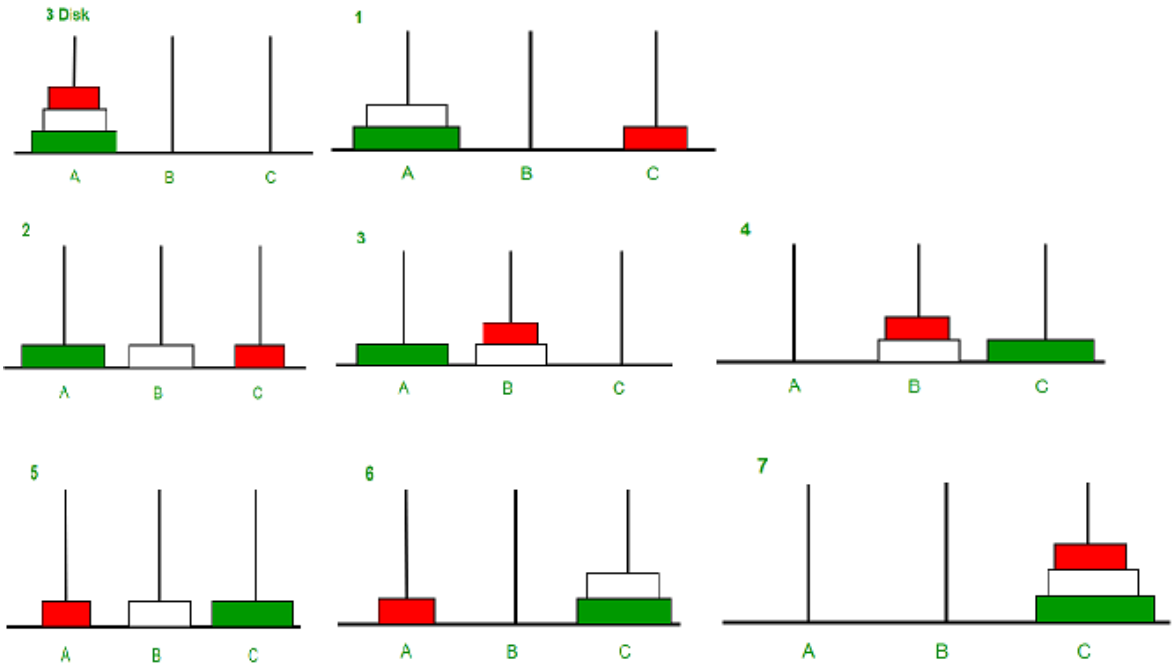
You are tasked with organizing a set of numbered disks (**1** to **n**) on three pegs labelled **A**, **B**, and **C**. The disks are initially stacked in ascending order of size on peg A. Your goal is to move all the disks from peg A to peg C using the Towers of Hanoi game rules.

Each move involves transferring one disk at a time, and you must follow the rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the top disk from one of the pegs and placing it on top of another peg.
3. A larger disk cannot be placed on top of a smaller disk.

You need to write a recursive function, **countMoves(n)**, that takes the number of disks **n** as input and returns the total number of moves required to solve the Towers of Hanoi problem.

For example, if the number of disks is 3, the disks can be transferred as follows: The total number of moves made is 7.



Note: This question helps in clearing Capgemini recruitment.

Input format :

The input consists of an integer denoting the number of disks.

Output format :

The output displays the sequence of moves required to solve the Tower of Hanoi puzzle, along with the total number of moves.

Refer to the sample output for formatting specifications.

Code constraints :

$0 < n \leq 8$

Sample test cases :

Input 1 :

3

Output 1 :

Move disk 1 from A to C

Move disk 2 from A to B

Move disk 1 from C to B

Move disk 3 from A to C

Move disk 1 from B to A

Move disk 2 from B to C

Move disk 1 from A to C

Total number of moves: 7

Input 2 :

2

Output 2 :

Move disk 1 from A to B

Move disk 2 from A to C

Move disk 1 from B to C

Total number of moves: 3

```
#include<iostream>
using namespace std;

int count = 0;

void towerOfHanoi(int n, char source, char helper, char destination){
 if(n == 1){
  count++;
  cout<<"Moving disc "<<n<<" from "<<source<<" to "<<destination<<endl;
  return;
 }
 towerOfHanoi(n-1, source, destination, helper);
 count++;
 cout<<"Moving disc "<<n<<" from "<<source<<" to "<<destination<<endl;
 towerOfHanoi(n-1, helper, source, destination);
}

int main(){
 int n;
 cin>>n;
```

```
 towerOfHanoi(n, 'A', 'B', 'C');
 cout<<"Total number of moves:"<<count<<endl;
 return 0;
}
```

Problem Statement 6

During an archaeological expedition, a team discovers the "Disk of Light," an artefact consisting of concentric rings with unique symbols. To unlock its powers, they must perform the "Tower of Wisdom" ritual.

In this ritual:

1. There are three pedestals: Source (A), Auxiliary (B), and Illuminated (C).
2. Disks are stacked on the Source pedestal, smaller ones on top of larger ones.
3. Only one disk can be moved at a time.
4. A disk can only be placed on top of a larger disk or an empty pedestal.

Write a program to assist the archaeological team in calculating the total number of times each ring of the Disk of Light is moved during the Tower of Wisdom ritual.

Note: This kind of question will be helpful in clearing TCS recruitment.

Input format :

The input consists of an integer n denoting the number of disks.

Output format :

The output displays the sequence of moves required to solve the puzzle.

The last line of output displays the total number of times the disk is moved.

Refer to the sample output for formatting specifications.

Code constraints :

$0 < n \leq 8$

Sample test cases :

Input 1 :

3

Output 1 :

Move disk 1 from A to C

Move disk 2 from A to B

Move disk 1 from C to B

Move disk 3 from A to C

Move disk 1 from B to A

Move disk 2 from B to C

Move disk 1 from A to C

Total number of times the disk is moved: 7

Input 2 :

4

Output 2 :

Move disk 1 from A to B

Move disk 2 from A to C

Move disk 1 from B to C

Move disk 3 from A to B

Move disk 1 from C to A

Move disk 2 from C to B

Move disk 1 from A to B

Move disk 4 from A to C

Move disk 1 from B to C

Move disk 2 from B to A

Move disk 1 from C to A

Move disk 3 from B to C

Move disk 1 from A to B

Move disk 2 from A to C

Move disk 1 from B to C

Total number of times the disk is moved: 15

```
#include<iostream>
using namespace std;

int count = 0;

void towerOfHanoi(int n, char source, char helper, char destination){
 if(n == 1){
  count++;
  cout<<"Move disc "<<n<<" from "<<source<<" to "<<destination<<endl;
  return;
 }
 towerOfHanoi(n-1, source, destination, helper);
 count++;
 cout<<"Move disc "<<n<<" from "<<source<<" to "<<destination<<endl;
 towerOfHanoi(n-1, helper, source, destination);
}

int main(){
 int n;
 cin>>n;
 towerOfHanoi(n, 'A', 'B', 'C');
 cout<<"Total number of times the disk is moved: "<<count<<endl;
 return 0;
}
```