Question 1

Problem Description

In a given town of `n` people, each person is identified with a unique label ranging from 1 to `n`. Among them, a rumor suggests that one person might be the town judge.

Now, there are certain properties that the town judge follows:

1. The town judge doesn't trust anyone.
2. Except for the town judge, everyone else trusts the town judge.
3. Only one person can have these two properties combined.

You will be given an array called `trust`, where every element is a pair, `[ai, bi]`, indicating that person `ai` trusts person `bi`. If a trust relationship isn't present in the trust array, it means that relationship doesn't exist in the town.

Your task is to find the label of the town judge, if they exist and can be identified. If not, return -1.

Class Definition

Class: `Question1`

Method: `findJudge`

The class `Question1` includes the method `findJudge` which accepts four `long` parameters and returns a `String`.

```
// java
public int findJudge(int n, int[][] trust)

//cpp
```

```
public  int  findJudge(int n, vector<vector<int>>& trust)
```

```
#python
# python
def findJudge(n: int, trust: List[List[int]]) -> int:
```

Constraints

- The number of people, `n`, lies between 1 and 1000, inclusive.
- The length of `trust` array is between 0 and 10^4.
- Every trust pair, `trust[i]`, has a size of 2.
- All pairs in `trust` are unique.
- For any trust pair, `ai` is not equal to `bi`.
- Both `ai` and `bi` lie in the range from 1 to `n`.

Example Scenarios

Example 1:

```
int n = 2;
int[][] trust = {{1, 2}};
System.out.println(findJudge(n, trust));  // Output: 2
In this case, person 1 trusts person 2, which makes person 2 a potential judge
```

Example 2:

```
int n = 3;
int[][] trust = {{1, 3}, {2, 3}};
System.out.println(findJudge(n, trust));  // Output: 3
Persons 1 and 2 trust person 3, making person 3 the town judge.
```

Example 3:

```
int n = 3;
int[][] trust = {{1, 3}, {2, 3}, {3, 1}};
System.out.println(findJudge(n, trust));  // Output: -1
Although persons 1 and 2 trust person 3, person 3 trusts person 1, which means
```

Test Cases

Input	Expected Output
2, [[1,2]]	2
3, [[1,3],[2,3]]	3
3, [[1,3],[2,3],[3,1]]	-1
4, [[1,3],[2,3],[3,4]]	-1
5, [[1,5],[2,5],[3,5],[4,5]]	5
5, [[1,5],[2,5],[3,5],[4,5],[5,1]]	-1
1, []	1

Note: Expected outputs are provided for all test cases for clarity.
Note2: There will be some hidden testcases that aren't listed here

Question 2

Problem Statement

Xenia and Yvona have obtained a new toy - a collection of wooden cubes where each cube has a unique, positive integer side length. They possess an infinite supply of cubes of every possible size.

Both girls aim to build a tower using exactly `N` cubes, ensuring that the total height of the tower sums up to exactly `H`. Your task is to compute and return the largest possible positive difference between the volumes of the two towers they construct.

A few important rules regarding the placement of the cubes:

- No unconventional cube placements are allowed. The base cube of the tower must

be positioned directly on the ground (ensuring its bottom face remains horizontal). Every subsequent cube is then placed directly on the top face of the preceding cube such that their touching faces partially overlap.

- Make sure to handle potential integer overflows, as the correct return value might sometimes exceed the capacity of a 32-bit integer.
- The smallest cube has a side length of 1. Using cubes with side = 0 is not permitted.

Class Definition

Class: Question2

Method: difference

The class `Question2` comprises a method `difference`.

Java Signature:

```
public long difference(int H, int N)
```

```
public:
long long difference(int H, int N);
```

```
Python Signature:
def difference(H: int, N: int) -> int:
```

Constraints

- `H` lies between 1 and 10^6, inclusive.
- `N` ranges between 1 and `H`, inclusive.

Example Scenarios

For instance, if Xenia chooses cubes with sizes 1, 2, and 3, and Yvona chooses cubes with sizes 1, 1, and 4 for a target height `H` of 6 and `N` of 3:

```
CubeTower tower = new CubeTower();
System.out.println(tower.difference(6, 3));  // Expected output: 42
```

The difference in volumes would be (1^3 + 1^3 + 4^3) - (2^3 + 2^3 + 2^3) = 42.

Test Cases

Input	Expected Output
4, 2	12
5, 3	12
3, 3	0
12, 6	300
1000000, 123456	
1000000, 200000	
1000000, 7	
999999, 2	
100971, 1919	
726412, 81201	
945884, 2595	
884564, 13885	
988440, 955075	
961968, 960070	
588195, 584701	

Note: The expected outputs for some test cases are not provided for clarity.