Beautiful Arrangement

Suppose you have **N** integers from 1 to N. We define a beautiful arrangement as an array that is constructed by these **N** numbers successfully if one of the following is true for the i_{th} position ($1 \le i \le N$) in this array:

1. The number at the i_{th} position is divisible by **i**.
2. **i** is divisible by the number at the i_{th} position.

Now given N, how many beautiful arrangements can you construct?

Example 1:

```
Input: 2
Output: 2
Explanation: 

The first beautiful arrangement is [1, 2]:

Number at the 1st position (i=1) is 1, and 1 is divisible by i (i=1).

Number at the 2nd position (i=2) is 2, and 2 is divisible by i (i=2).

The second beautiful arrangement is [2, 1]:

Number at the 1st position (i=1) is 2, and 2 is divisible by i (i=1).

Number at the 2nd position (i=2) is 1, and i (i=2) is divisible by 1.
```

Note:

1. **N** is a positive integer and will not exceed 15.

Solution 1

Just try every possible number at each position...

```
public class Solution {
    int count = 0;

    public int countArrangement(int N) {
        if (N == 0) return 0;
        helper(N, 1, new int[N + 1]);
        return count;
    }

    private void helper(int N, int pos, int[] used) {
        if (pos > N) {
            count++;
            return;
        }

        for (int i = 1; i <= N; i++) {
            if (used[i] == 0 && (i % pos == 0 || pos % i == 0)) {
                used[i] = 1;
                helper(N, pos + 1, used);
                used[i] = 0;
            }
        }
    }
}
```

written by shawngao original link here

Solution 2

```
// By lovellp
// Time: 6ms
class Solution {
public:
    int countArrangement(int N) {
        vector<int> vs;
        for (int i=0; i<N; ++i) vs.push_back(i+1);
        return counts(N, vs);
    }
    int counts(int n, vector<int>& vs) {
        if (n <= 0) return 1;
        int ans = 0;
        for (int i=0; i<n; ++i) {
            if (vs[i]%n==0 || n%vs[i]==0) {
                swap(vs[i], vs[n-1]);
                ans += counts(n-1, vs);
                swap(vs[i], vs[n-1]);
            }
        }
        return ans;
    }
};
```

written by love_FDU_llp original link here

Solution 3

The trick is: Arrange the values starting from the end of the array.

```
public class Solution {
    public int countArrangement(int N) {
        dfs(N, N, new boolean[N + 1]);
        return count;
    }

    int count = 0;

    void dfs(int N, int k, boolean[] visited) {
        if (k == 0) {
            count++;
            return;
        }
        for (int i = 1; i <= N; i++) {
            if (visited[i] || k % i != 0 && i % k != 0) {
                continue;
            }
            visited[i] = true;
            dfs(N, k - 1, visited);
            visited[i] = false;
        }
    }
}
```

written by ibmtp380 original link here