MICROSOFT

#2: Combination Sum III

Medium

Find all valid combinations of `k` numbers that sum up to `n` such that the following conditions are true:

- Only numbers `1` through `9` are used.
- Each number is used **at most once**.

Return *a list of all possible valid combinations*. The list must not contain the same

combination twice, and the combinations may be returned in any order.

Example:

```
Input: k = 3, n = 9
Output: [[1,2,6],[1,3,5],[2,3,4]]
Explanation:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
There are no other valid combinations.
```

Constraints:

- `2 <= k <= 9`
- `1 <= n <= 60`

```
vector<vector<int>>res;
    vector<int>temp;
    void comb(vector<int>nums,int n,int sum,int k)
    {
        if(sum==0 && k==0)
        {
            res.push_back(temp);
            return;
        }
        if(sum<0 || n<0)
            return;
        temp.push_back(nums[n]);
        comb(nums,n-1,sum-nums[n],k-1);
        temp.pop_back();
        comb(nums,n-1,sum,k);
    }
    vector<vector<int>> combinationSum3(int k, int n) {
        vector<int>nums;
        for(int i=1;i<=9;i++)
            nums.push_back(i);
        comb(nums,8,n,k);
        return res;
    }
```

Microsoft:

#2 Explanation:

This is simple backtracking problem. First we push the numbers from 1 to 9 into a vector then we'll pass k and n as target. Then we'll backtrack with two choices either we can take the element from the array in that case we'll subtract the element from sum (sum-element) and push that element into temp vector and and second choice we don't pick the element and if sum and k becomes 0 we'll push the temp vector into resultant vector

Complexity

- Time complexity:O(9^k)
- Space complexity: `O(K)`