2044. Count Number of Maximum Bitwise-OR Subsets

Given an integer array `nums`, find the **maximum** possible **bitwise OR** of a subset of `nums` and return *the **number of different non-empty subsets** with the maximum bitwise OR*.

An array `a` is a **subset** of an array `b` if `a` can be obtained from `b` by deleting some (possibly zero) elements of `b`. Two subsets are considered **different** if the indices of the elements chosen are different.

The bitwise OR of an array `a` is equal to `a[0] OR a[1] OR ... OR a[a.length - 1]` (**0-indexed**).

Example 1:

```
Input: nums = [3,1]
Output: 2
Explanation: The maximum possible bitwise OR of a subset is 3. There are 2 subsets with a bitwise OR of 3:
- [3]
- [3,1]
```

Example 2:

```
Input: nums = [2,2,2]
Output: 7
Explanation: All non-empty subsets of [2,2,2] have a bitwise OR of 2. There are 23 - 1 = 7 total subsets.
```

Example 3:

```
Input: nums = [3,2,1,5]
Output: 6
Explanation: The maximum possible bitwise OR of a subset is 7. There are 6 subsets with a bitwise OR of 7:
- [3,5]
- [3,1,5]
- [3,2,5]
- [3,2,1,5]
- [2,5]
- [2,1,5]
```

Constraints:

- `1 <= nums.length <= 16`
- `1 <= nums[i] <= 10^5`

2044. Count Number of Maximum Bitwise-OR Subsets

```
/*
   Bit manipulation
   Time complexity: O(2^n n)
   Spave complexity: O(1)
*/
class Solution {
public:
   int countMaxOrSubsets(vector<int>& nums) {
      int mx=0;
      for(int e: nums) mx|=e;

      int n=nums.size();
      int m=1<<n;
      int ans=0;
      for(int i=0;i<m;++i){
         int x=0;
         for(int j=0;j<n;++j){
            if( (i&(1<<j))!=0 ) x|=nums[j];
         }
         if(x==mx) ans++;
      }

      return ans;
   }
};
```

2044. Count Number of Maximum Bitwise-OR Subsets


```
/*
   Recursion
   Time complexity: O(n+2^n)=O(2^n)
   Spave complexity: O(n)
*/
class Solution {
   public:
      int countMaxOrSubsets(vector<int>& nums) {
         int n=nums.size();
         int mx=0;
         for(int e: nums) mx|=e;

         auto solve=[&](int index,int cur,auto& self)->int{
            if(index>=n) return cur==mx?1:0;

            int exclude=self(index+1,cur,self);
            int include=self(index+1,cur|nums[index],self);

            return include+exclude;
         };

         return solve(0,0,solve);
      }
};
```

2044. Count Number of Maximum Bitwise-OR Subsets


```
/*
   Memoization
   Time complexity: O(n+n.mx)=O(n.mx)
   Space complexity: O(n+n.mx)=O(n.mx)
*/
class Solution{
   public:
      int countMaxOrSubsets(std::vector<int>& nums){
         int mx=0;
         for(int e: nums) mx|=e;

         int n=nums.size();
         std::vector<std::vector<int>> memo(n,std::vector<int>(mx+1,-1));

         auto solve=[&](int index,int cur,auto& self)->int{
            if(index>=n) return cur==mx?1:0;

            if(memo[index][cur]!=-1) return memo[index][cur];
            int exclude=self(index+1,cur,self);
            int include=self(index+1,cur|nums[index],self);

            return memo[index][cur]=include+exclude;
         };

         return solve(0,0,solve);
      }
};
```

2044. Count Number of Maximum Bitwise-OR Subsets

```
/*
   0/1 Knapsack
   Time complexity:O(n mx)
   Spave complexity: O(n mx)
*/
class Solution {
   public:
      int countMaxOrSubsets(vector<int>& nums) {
         int n=nums.size();
         int mx=0;
         for(int e: nums) mx|=e;

         std::vector<std::vector<int>> dp(n+1,std::vector<int>(mx+1,0));

         dp[0][0]=1;

         for(int i=1;i<=n;++i){
            for(int j=0;j<=mx;++j){
               // Exclude nums[i-1]
               dp[i][j]+=dp[i-1][j];

               // Include nums[i-1]
               int or_val=j|nums[i-1];
               if(or_val<=mx) dp[i][or_val]+=dp[i-1][j];
            }
         }

         return dp[n][mx];
      }
};
```