```
/*
两个整数的 汉明距离 指的是这两个数字的二进制数对应位不同的数量。

计算一个数组中，任意两个数之间汉明距离的总和。

示例:

输入: 4, 14, 2

输出: 6

解释: 在二进制表示中，4表示为0100，14表示为1110，2表示为0010。（这样表示是为了体现后四位之间关系）
所以答案为：
HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

注意:

    数组中元素的范围为从 0到 10^9。
    数组的长度不超过 10^4。

在真实的面试中遇到过这道题?

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/total-hamming-distance
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析：

- 常规方法：遍历所有的可能组合$O(N^2)$，分别求汉明距离再累加起来，方法一果然超时了
- 纵向比较方法 顺便致谢：
    - 用32个长度的数组统计各位上1的个数
    - 在某位上，若1的个数为n,则0的个数为$N-n$，1和1的距离是0，1和0的距离是n,n个1到$N-n$个0的距离是$n(N-n)$
    - 时间复杂度$O(N)$

方法一:C_组合穷举法

```
int HammingDistance(int a , int b)
{
    int c       = a^b   ;
    int count   = 0     ;

    while(c!=0)
    {
        count ++;
        c = c & (c-1);
```

```
    }
    return count;
}

int totalHammingDistance(int* nums, int numsSize)
{
    int i = 0;
    int j = 0;
    int ret_val = 0;

    for(i = 0; i < numsSize ; i++)
    {
        for(j = i+1; j < numsSize ; j++)
        {
            ret_val += HammingDistance(nums[i],nums[j]);
        }
    }
    return ret_val;
}

/*
执行结果：
超出时间限制
显示详情
最后执行的输入：
[411914430,351525660,364254723,700735511,254391807,716137650,627125762,45921036,
641201357,391245031,853287481,306898501,934869860,12870499,566111993,84688719,16
7599428,138020110,421650010,123166617,473053367,416456553,529059276,109931730,22
5123199,926903226,409122622,674248838,976498694,736498262,23526704,69301658,6098
07910,42558464,754007153,90198102,629024110,93706236,253668710,654969675,7415320
3,752367561,657884191,813783381,726915302,231012931,131381188,96394756,342467763
,599518781,118880543,865494491,459169106,139399449,567837622,22564264,73441754,1
18419609,354006272,253711314,16510034,458750975,761344095,202635839,584460509,43
7573385,311874367,457882420,195925739,624443100,55577389,726769056,68023703,2515
49626,546746886,885366067,433298006,322539865,678786027,923622925,318699959,3589
73873,785297666,856356578,349604252,283405172,71996758,16218445,645960524,128691
283,191339430,851759029,180988079,822778179,578629998,21401348,501160296,5702313
38,806064852,998100866,879466723,8602497

*/
```

方法二：C_纵向比较方法

```
int totalHammingDistance(int* nums, int numsSize)
{
    int ret_val             =   0       ;
    int bit1_nums           =   {0,}    ;
    int i                   =   0       ;
    int j                   =   0       ;
    int bit_arr[32]         =   {
        0x00000001,0x00000002,0x00000004,0x00000008,
        0x00000010,0x00000020,0x00000040,0x00000080,
        0x00000100,0x00000200,0x00000400,0x00000800,
        0x00001000,0x00002000,0x00004000,0x00008000,
        0x00010000,0x00020000,0x00040000,0x00080000,
```

```
        0x00100000,0x00200000,0x00400000,0x00800000,
        0x01000000,0x02000000,0x04000000,0x08000000,
        0x10000000,0x20000000,0x40000000,0x80000000,
    };


    for(i = 0; i < 32 ; i++)
    {
        bit1_nums  = 0;
        for(j=0;j<numsSize;j++)
        {
            if(bit_arr[i]&nums[j])
            {
               bit1_nums++;
            }
        }
        ret_val += bit1_nums*(numsSize-bit1_nums);
    }
    return ret_val;
}

/*
执行结果:
通过
显示详情
执行用时 :112 ms, 在所有 C 提交中击败了6.90% 的用户
内存消耗 :7.8 MB, 在所有 C 提交中击败了100.00%的用户
*/
```

方法三：C_纵向法(优化逻辑)

```
int totalHammingDistance(int* nums, int numsSize)
{
    int             ret_val             =   0       ;
    int             bit1_nums           =   0       ;
    int             i                   =   0       ;
    int             j                   =   0       ;
    unsigned int    temp                =   0x01    ;

    for(i = 0; i < 32 ; i++)
    {
        bit1_nums  = 0;

        for(j=0;j<numsSize;j++)
        {
            if(temp&nums[j])
            {
               bit1_nums++;
            }
        }
        temp <<= 1;
        ret_val += bit1_nums*(numsSize-bit1_nums);
    }
```

```
    return ret_val;
}
/*
执行结果:
通过
显示详情
执行用时 :92 ms, 在所有 C 提交中击败了24.14% 的用户
内存消耗 :7.9 MB, 在所有 C 提交中击败了80.00%的用户
*/
```

AlimyBreak
2019.08.16