# Bit operation

1. a << b:
A = 1100 B = 2;
A << B = 110000

2. a >> b ; a >>> b;
A = 11111111111111111111111111110001 (32 bit)
B = 2
A >> B = 11111111111111111111111111111100
A >>> B = 00111111111111111111111111111100

3. tips:
    x & (x - 1)：消除x最后一位的1
    x = 1100
    x - 1 = 1011
    x & (x - 1) = 1000

example 1: detect whether n is power of 2 by using O(1) time
    solution: if n is power of 2, then: n > 0 and n only has one 1
```
bool checkPowerOf2(int n){
    return n > 0 && (n & (n - 1)) == 0;
}
```
example 2: calculate the number of 1s in a 32 bit integer n
```
public int countOnes(int n){
    int count = 0;
    while(n != 0){
        n = n & ( n - 1);
        count ++;
    }
    return count;
}
```
example 3: if change A to B, how many bit should be changed?
```
public int countOnes(int n){
    int count = 0;
    while(n != 0){
        n = n & ( n - 1);
        count ++;
    }
    return count;
}
public int bitSwapRequired(int a, int b){
    return countOnes(a ^ b);
}
```
example 4: give the sub sets of a list;
    solution: 使用一个正整数二进制表示的第i位是1还是0，代表集合的第i个数取或者不取，所以从0到2^n-1总共2^n个整数，正好对应集合的2^n个子集。

```
public ArrayList<ArrayList<Integer>> subsets(int[] nums){
    ArrayList<ArrayList<Integer>> res = new ArrayList<>();
    int n = nums.length;
    Arrays.sort(nums);

    //1 << n is 2 ^ n
    //each subset equals to an binary integer between 0 … 2 ^ n - 1
    //0 -> 000 - > []
    //1 -> 001 -> [1]
    //2 -> 010 -> [2]
    //7 -> 111 -> [1,2,3]

    for(int i = 0; i < (1 << n); i ++){
        ArrayList<Integer> subset = new ArrayList<>();
        for(int j = 0; j < n; j ++){
            if((i & (1 << j)) != 0){
                subset.add(nums[j]);
            }
        }
        res.add(subset);
    }
    return res;
}
```

tips 2: a ^ b ^ b = a

example: in a list, only one number appears once, others appears twice, find the one.

```
public int singleNumber(int[] A){
    if(A == null || A.length == 0){
        return -1;
    }
    int res = 0;
    for(int i = 0; i < A.length; i ++){
        res ^= A[i];
    }
    return res;
}
```

example: in a list, only one number appears once, others appears three times, find the single one.

solution： 因为数是出现三次的，也就是说，对于每一个二进制位，如果只出现一次的数在该二进制位为1，那么这个二进制位在全部数字中出现次数无法被3整除。膜3运算只有三种状态：00，01，10.因此我们可以使用两个位来表示当前位%3，对于每一个位，我们让Two, One表示当前位的状态，列出真值表查找状态后可得出。

One += (One ^ B) & (~Two)

Two += (~One +) & (Two ^ B)

```
public singleNumber(int A[], int n){
    int one = 0;
    int two = 0;
    int i, j, k;
```

```
        for(i = 0; i < n; i ++){
            two = two | (one & A[i]);
            one = one ^ A[i];
            int three = two & one;
            two = two ^ three;
            one = one ^ three;
        }
        return one | two;
    }
```

example: in a list, only two numbers appears once, others appears twice, find these two numbers.

将数组分成两个部分，每个部分里只有一个元素出现一次，其余元素都出现两次。那么使用这种方法就可以找出这两种元素了。不妨假设出现一个的两个元素是x，y，那么最终所有的元素亦或的结果就是res＝x^y.and res != 0。那么我们可以找出res二进制表示中的某一位是1.对于原来的数组，我们可以根据这个位置是不是1就可以将数组分成两个部分。x，y在不同的两个子数组中。而却对于其它成对出现的元素，要么在x所在的那个数组，要么在y所在的那个数组。

```
    public int[] singleNumber(int[] nums){
        int diff = 0;
        for(int i  =0; i < nums.length; i ++){
            diff ^= nums[i];
        }

        //取最后一位1
        //源码：二进制表示（有一位符号位），反码：正数的反码就是源码，负数的反码是符号位不变，其余位取反
        //补码：正数的补码就是源码，负数的补码是反码 + 1
        //diff & (-diff)就是取diff的最后一位1的位置
        diff &= (-diff);
        int[] res = {0, 0};
        for(int i = 0; i < nums.length; i ++){
            if((nums[i] & diff) == 0){
                res[0] ^= nums[i];
            }
            else{
                res[1] ^= nums[i];
            }
        }
        return res;
    }
```