Approach 1: List operation

**Algorithm**

1. Iterate over all the elements in \text{nums}nums
2. If some number in \text{nums}nums is new to array, append it
3. If some number is already in the array, remove it

**Complexity Analysis**

* Time complexity : O(n^2)*O*(*n*2). We iterate through \text{nums}nums, taking O(n)*O*(*n*) time. We search the whole list to find whether there is duplicate number, taking O(n)*O*(*n*) time. Because search is in the for loop, so we have to multiply both time complexities which is O(n^2)*O*(*n*2).
* Space complexity : O(n)*O*(*n*). We need a list of size n*n* to contain elements in \text{nums}nums.

Approach 2: Hash Table

**Algorithm**

We use hash table to avoid the O(n)*O*(*n*) time required for searching the elements.

1. Iterate through all elements in nums and set up key/value pair.
2. Return the element which appeared only once.

**Complexity Analysis**

* Time complexity : O(n \cdot 1) = O(n)*O*(*n*⋅1)=*O*(*n*). Time complexity of for loop is O(n)*O*(*n*). Time complexity of hash table(dictionary in python) operation pop is O(1)*O*(1).
* Space complexity : O(n)*O*(*n*). The space required by hash\\_table*hash*\_*table* is equal to the number of elements in \text{nums}nums.

#### Approach 3: Math

**Concept**

2 \* (a + b + c) - (a + a + b + b + c) = c2∗(*a*+*b*+*c*)−(*a*+*a*+*b*+*b*+*c*)=*c*

**Complexity Analysis**

* Time complexity : O(n + n) = O(n)*O*(*n*+*n*)=*O*(*n*). sum will call next to iterate through \text{nums}nums. We can see it as sum(list(i, for i in nums)) which means the time complexity is O(n)*O*(*n*) because of the number of elements(n*n*) in \text{nums}nums.
* Space complexity : O(n + n) = O(n)*O*(*n*+*n*)=*O*(*n*). set needs space for the elements in nums

Approach 4: Bit Manipulation

**Concept**

* If we take XOR of zero and some bit, it will return that bit
  + a \oplus 0 = a*a*⊕0=*a*
* If we take XOR of two same bits, it will return 0
  + a \oplus a = 0*a*⊕*a*=0
* a \oplus b \oplus a = (a \oplus a) \oplus b = 0 \oplus b = b*a*⊕*b*⊕*a*=(*a*⊕*a*)⊕*b*=0⊕*b*=*b*

So we can XOR all bits together to find the unique number.

**Complexity Analysis**

* Time complexity : O(n)*O*(*n*). We only iterate through \text{nums}nums, so the time complexity is the number of elements in \text{nums}nums.
* Space complexity : O(1)*O*(1).