Find the duplicate Number

BRUTE FORCE :

Sort the array and compare adjacent elements if matches return that element because the duplicate will occur side by side.

- Time complexity : O(NlogN) + O(N)
- SPace Complexity : O(1)

CODE :

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        for(int i=0;i<nums.size();i++)
        {
            cout<<nums[i]<<" ";
        }
        for(int i=1;i<nums.size();i++)
        {
            if(nums[i-1]==nums[i])
                return nums[i];
        }
        return 0;
    }
};
```

Approach 1:

Create a hash array as the elements are from 1 to n it is easy to maintain their frequency. Wherever frequency is greater than 1 return that element

- Time Complexity : O(N)
- Space Complexity : O(N)but in ques it is given not to use extra space.

```
class Solution {
public:
```

```
    int findDuplicate(vector<int>& nums) {
        int n=nums.size();
        vector<int> freq(n,0);
        for(int i=0;i<n;i++)
        {
            if(freq[nums[i]]==0)
            {
                freq[nums[i]]++;
            }
            else
            {
                return nums[i];
            }
        }
        return 0;
    }
};
```

Approach 3 : Linkedlist Cycle.

We keep two pointers one fast and other slow. We follow linkedlist cycle approach where the elements are linked to each other through index as they are in range 1 to n each element will be found at an index and as duplicate element occurs twice there will be a cycle formedas fast pointe travels 2 pos ahead and slow by 1 position ahead there will exist a point in cycle where both pointer meets this is the first collision point where slow has travelled a distance a and fast has travelled a distance 2a. Now getting fast pointer out of the cycle and moving both pointers by only 1 pos, the difference in distance is 2a-a=a ehich needs to be a multiple of length of cycle for slow and fast pointer to meet again and now they meet at the duplicate number point.

For 2nd collision idea is to keep one pointer inside the cycle and other outside the cycle and get the starting point of the cycle by moving both pointers by only 1 position.

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int slow=nums[0];
        int fast=nums[0];
        do
        {
            slow=nums[slow];
            fast=nums[nums[fast]];
        }while(slow!=fast);
        fast=nums[0];
```

```
        while(slow!=fast)
        {
            slow=nums[slow];
            fast=nums[fast];

        }
        return slow;
    }
};
```

- Time Complexity : O(N)
- Space Complexity : O(1)