Finding first duplicate in an array

Simple approach Approach with a constraint that number of elements is limited to range 1-100

```
int arr[10];
// Using array as a hashmap
class Solution {
public:
    int containsDuplicate(vector<int>& nums) {

        int visited[101];
        // O(1)
        for (int i = 0; i < 101; i++) {
            visited[i] = 0;
        }

        //
        for (int i = 0; i < nums.size(); i ++) { // O(N)
            int num = nums[i];

            if (visited[num] != 0) { // O(1)
                return num;
            } else {
                visited[num] = 1; // O(1)
            }

        }

        return -1;
    }
};

O(N)

// Using hashmap
class Solution {
public:
    int containsDuplicate(vector<int>& nums) {

        std::unordered_map<int, int> visited;

        for (int i = 0; i < nums.size(); i ++) { // O(N)
            int num = nums[i];

            if (visited.count(num) == 0) { // visited.find(num) == visited.end()  // O(1)
                visited[num] = 1 // O(1)
            } else {
                return num;
            }

        }

        return -1;
    }
};

// TC: O(N)
// SC: O(N)

[1,2,3,5,3]
 0 1 2 3 4
i           0    1           2                3                   4
visited {} {1:1} {1:1, 2:1}  {1:1, 2:1, 3:1}  {1:1, 2:1, 3:1, 5:1}
[1,1,2,4,5,6]
```

In [ ]:

Hashmap

C++: map (O(log n)), unordered_map (O(1))
Java: HashMap, LinkedHashMap
Python: dict
Javascript: map

- set value for a key: O(1)
- get value by key O(1)
- check if a key is present O(1)
- delete a key and it's value O(1)

- find value O(N)

In [ ]:

In [ ]:

Counting frequency

In [5]:
```
"abcdefab"

def most_frequent(s):
    freq = {} # hashmap

    for c in s:
        if c in freq:
            freq[c] += 1
        else:
            freq[c] = 1

    res = ""
    max_freq = 0
    for k,v in freq.items():
        if v > max_freq:
            res = k
            max_freq = v
        elif v == max_freq:
            res  += k

    return res

print("Result", most_frequent("xababcdccd"))
# "xababcdccd"


# {x:1, a:2, b:2, c:3, d: 2}

# max_freq = 0  1    2     2   3    3
# res = ""      x    a     ab  c    c
#             x:1   a:2  b:2  c:3  d:2
```

```
Result c
```

In [ ]:
```
def most_frequent(s):
    freq = {} # hashmap

    max_freq  = 0
    res = ""
    for c in s:
        if c in freq:
            freq[c] += 1
        else:
            freq[c] = 1

        f = freq[c]
        if f > max_freq:
            res = c
            max_freq = f
        elif f == max_freq:
            res  += c

    return res

# "ababccdc"
#              a      b          a         b           c               c              c               d
#          {}  {a:1} {a:1, b:1} {a:2, b:1} {a:2, b:2}  {a:2, b:2, c:1} {a:2, b:2, c:2} {a:2, b:2, c:3} {a:2, b:2, c:3, d:1}
# max_freq 0    1    1           2          2          2               2              3               3
# res      ''   a    ab          a          ab         ab              abc            c               c
```

In [ ]:
```

```

In [6]:
```
## this only works for lower case alphabets
def most_frequent(s):
    freq = [0]*26 # hashmap

    for c in s:
        c = ord(c) - 97
        freq[c] += 1

    res = ""
    max_freq = 0
    for i in range(26):
        if freq[i] > max_freq:
            max_freq = freq[i]
            res = chr(i+97)
        elif freq[i] == max_freq:
            res += chr(i+97)
    return res

print("Result", most_frequent("xababcdccd"))
# [0 0 0 0 0 ... 0]
#  0 1 2 3       25
#  a b c d.......z
```

```
Result c
```

In [ ]:
```

```

Introducing Hashmaps

In [ ]:
```

```

In [ ]:
```

```

https://leetcode.com/problems/single-number/ (https://leetcode.com/problems/single-number/)

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:

        # # O(N^2)
        # for num in nums: # O(N)
        #     if nums.count(num) == 1: # O(N)
        #         return num

        # TC: O(N)
        # SC: O(N)
        # freq = {}
        # for num in nums:
        #     # if present, increase frequency by +1
        #     if num in freq:
        #         freq[num] += 1
        #     else:
        #         freq[num] = 1

        # for key, value in freq.items():
        #     if value == 1:
        #         return key

        # TC: O(N)
        # SC: O(1)

        # res = 0
        # for el in nums:
        #     res = res ^ el
        # return res

        return reduce(lambda x,y: x^y, nums)
```

```java
public int missingNumber(int[] nums) {
        Arrays.sort(nums); // O(n log N)
        int max = nums[nums.length - 1];

        if(max != nums.length){
            return nums.length;
        }
        for(int i = nums.length - 2 ; i >= 0 ; i--){ // O(N)
            if(nums[i] == max - 1){
                max--;
            } else {
                return (max -1);
            }
        }
        return 0;
    }
```

TC: O(n log n)
SC: O(1)

In [ ]:

https://leetcode.com/problems/missing-number/ (https://leetcode.com/problems/missing-number/)

```cpp
class Solution {
public:
    int missingNumber(vector<int>& nums) {

        // TC: O(N^2)
        // SC: O(1)
        // for(int i = 0; i <= nums.size(); i++) { // O(N)
        //     if (std::find(nums.begin(), nums.end(), i) == nums.end()) { // O(N)
        //         return i;
        //     }
        // }
        // return 0;

        // TC: O(N)
        // SC: O(N)
        std::unordered_map<int, int> visited;

        // // O(N)
        // for (auto num:nums) {
        //     visited[num] = 1;
        // }

        // // O(N)
        // for (int i = 0; i <= nums.size(); i++) {
        //     if (visited.count(i) == 0) { // O(1)
        //         return i;
        //     }
        // }

        // [3,0,1]
        //  {0:1, 3:1, 1:1}
        //
        //

        // O(N)
        // SC: O(1)
        int sum = 0;
        // O(N)
        for (auto num: nums) {
            sum += num;
        }

        // O(1)
        int n = nums.size();
        int expectedSum  = (n*(n+1))/2;

        return expectedSum - sum;
    }
};
```