WORKSHEET-4

Student Name: Nikhil Kumar **UID: 20BCS1817**

Branch: BE-CSE **Section/Group: 20BCS-DM-716/B**

Semester: 6 **Subject Code: 20CSP-351**

Subject Name: CC-II **Date of Performance: 23-03-2023**

Aim:

To demonstrate the concept of Hashing.

Question 1

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

Example 1:

Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Solution:

Approach 1

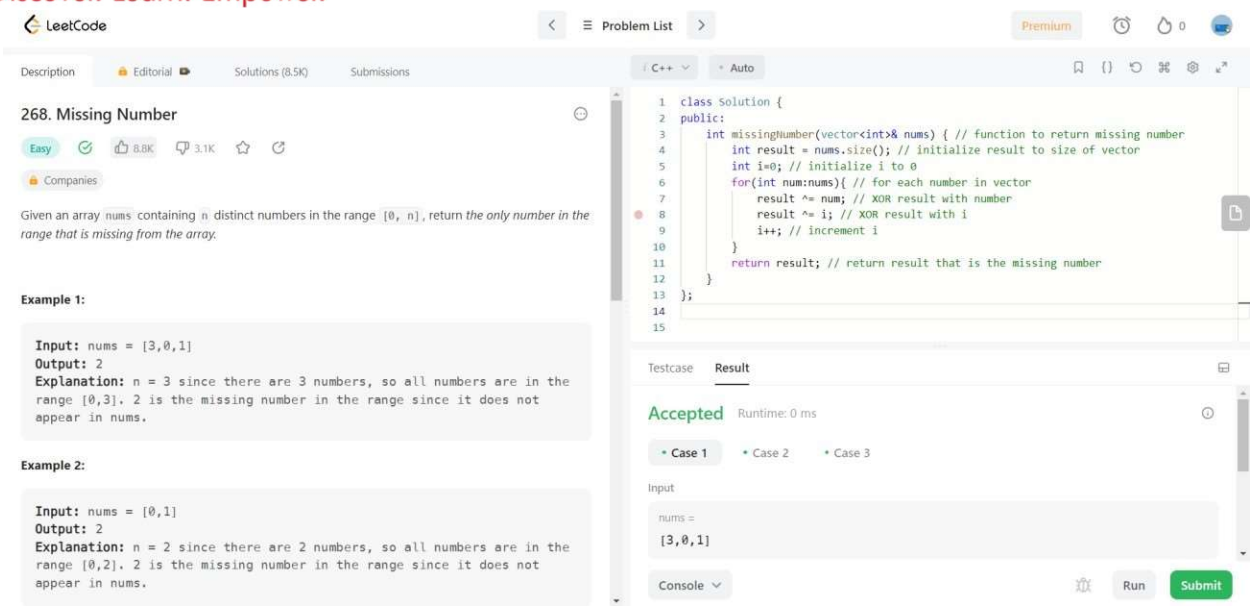
```
class Solution { public:
    int missingNumber(vector<int>& nums) { // function to return missing number
        int result = nums.size(); // initialize result to size of vector int i=0; //
        initialize i to 0
        for(int num:nums){ // for each number in vector
            result ^= num; // XOR result with number
            result ^= i; // XOR result with i
            i++; // increment i
        }
        return result; // return result that is the missing number
    }
};
```

Approach 2

```
class Solution { public:
    int missingNumber(vector<int>& nums) {
        int n=nums.size(); int
        sum=(n*(n+1))/2; for(int
        i=0;i<nums.size();i++)
        { sum-=nums[i];
        } return sum;


    }
};
```

Output



Question 2

Longest duplicate substring

Given a string s, consider all *duplicated substrings* : (contiguous) substrings of s that occur 2 or more times. The occurrences overlap.

Return **any** substring, the ring that has the long s does not have a duplicated answer is "".

Example 1:

Input: s = "banana"
Output: "a na"

Example 2:

Input: s = abcd"
Output: ""

Answer 2

Approach 1 class Solution { public:

```
string longestDupSubstring(string S) {
int n = S.length();

    string res;
    unordered_set<string_view>set;
    int l = 1, r = n;

    while (l <= r) {
      int m = l + (r-l)/2; bool found
      = false; for (int i = 0; i+m <=
      n; i++) {
        auto [it, inserted] = set.emplace(S.data()+i, m); if
        (!inserted) {

          found = true;
          res = move(*it);
          break;
        }
      }

      if (found)
        l = m+1;
      else
        r = m-1;
    }

    return res;
  }
};
```

Approach 2

```
class Solution { public: string
longestDupSubstring(string s) { int N =
s.length();
    unordered_map<char,vector<int>>
    indexes; int max_len=0, max_ind=0; string
    out;
    for(int i=0; i<N; i++){
```

```
            // breaking if the length of rest of the string is not greater than max_len
            if(max_len>= N-i+1) break; auto it = indexes.find(s[i]);
            if(it==indexes.end()){ indexes[s[i]] = {i}; continue;
            }
            for(int ind:it->second){
               // thorough check int
               len=1;
               while(i+len <N && s[i+len]==s[ind+len])len++; if(max_len<len)
               {
                  max_len = len; max_ind
                  = i;
               }
            }
            it->second.push_back(i);
         }

         return s.substr(max_ind,max_len);
      }
};
```

Output