



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## WORKSHEET-4

**Student Name:** Nikhil Kumar

**Branch:** BE-CSE

**Semester:** 6

**Subject Name:** CC-II

**UID:** 20BCS1817

**Section/Group:** 20BCS-DM-716/B

**Subject Code:** 20CSP-351

**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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

268. Missing Number

Easy

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`.

```
1 class Solution {
2 public:
3     int missingNumber(vector<int>& nums) { // function to return missing number
4         int result = nums.size(); // initialize result to size of vector
5         int i=0; // initialize i to 0
6         for(int num:nums){ // for each number in vector
7             result ^= num; // XOR result with number
8             result ^= i; // XOR result with i
9             i++; // increment i
10        }
11        return result; // return result that is the missing number
12    }
13 };
14
15
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

`nums =`  
`[3,0,1]`

Console Run Submit

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 `longest` that has the longest length. If `s` does not have a duplicated substring, the answer is `""`.

**Example 1:**

**Input:** `s = "banana"`

**Output:** `"abab"`

**Example 2:**

**Input:** `s = "abcd"`

**Output:** `""`

Answer 2



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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++){



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// 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

LeetCode

1044. Longest Duplicate Substring

Hard

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

Return **any** duplicated substring that has the longest possible length. If *s* does not have a duplicated substring, the answer is "".

**Example 1:**

Input: *s* = "banana"  
Output: "ana"

**Example 2:**

Input: *s* = "abcd"  
Output: ""

```
1 class Solution {
2 public:
3     string longestDupSubstring(string s) {
4         int N = s.length();
5         unordered_map<char, vector<int>> indexes;
6         int max_len=0, max_ind=0;
7         string out;
8         for(int i=0; i<N; i++){
9             // breaking if the length of rest of the string is not greater than max_len
10            if(max_len >= N-i+1) break;
11            auto it = indexes.find(s[i]);
12            if(it==indexes.end()){
13                indexes[s[i]] = {i};
14                continue;
15            }
16        }
17    }
```

Testcase Result

Accepted Runtime: 5 ms

Case 1 Case 2

Input

s = "banana"

Console Run Submit