Experiment-1.4

Student Name: Nabha Varshney **UID: 20BCS4995**

Branch: CSE **Section/Group: 20BCS-DM-704 (A)**

Semester: 6th **Date of Performance:01st Mar 2023**

Subject Name: Competitive Coding II **Subject Code: 20CSP- 351**

Aim – To demonstrate the concept of Hashing

Objective-

- The objective is to build problem solving capability and to learn the basic concepts of data structures.
- The implementation of Last Stone Weight which shows and brushes up the concept of Heap and can be solved through various approaches.
- The implementation of Cheapest flights with shops which is heap by default in C++.

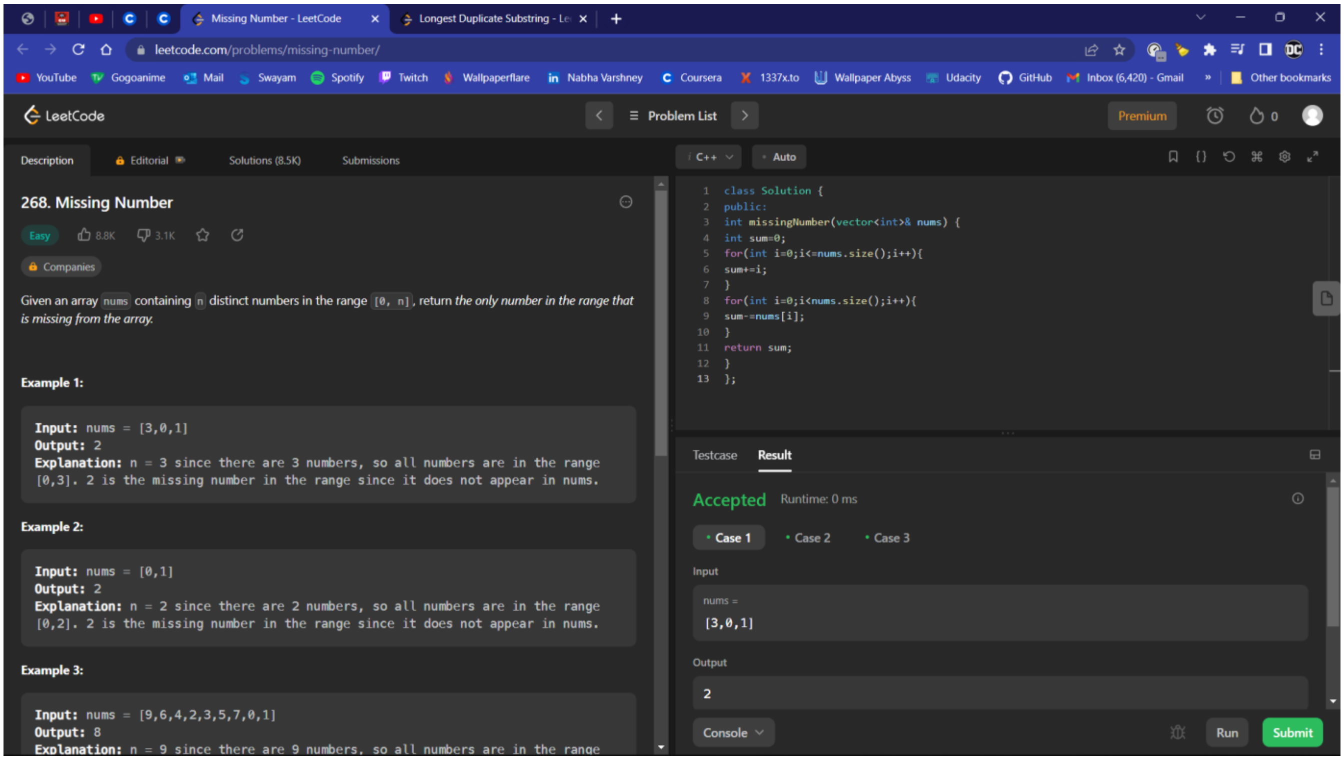
1) Missing Number
https://leetcode.com/problems/missing-number/

Code –

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

```
    }
    };
```

Output -



268. Missing Number

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.
```

Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range
```

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

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

Output

```
2
```

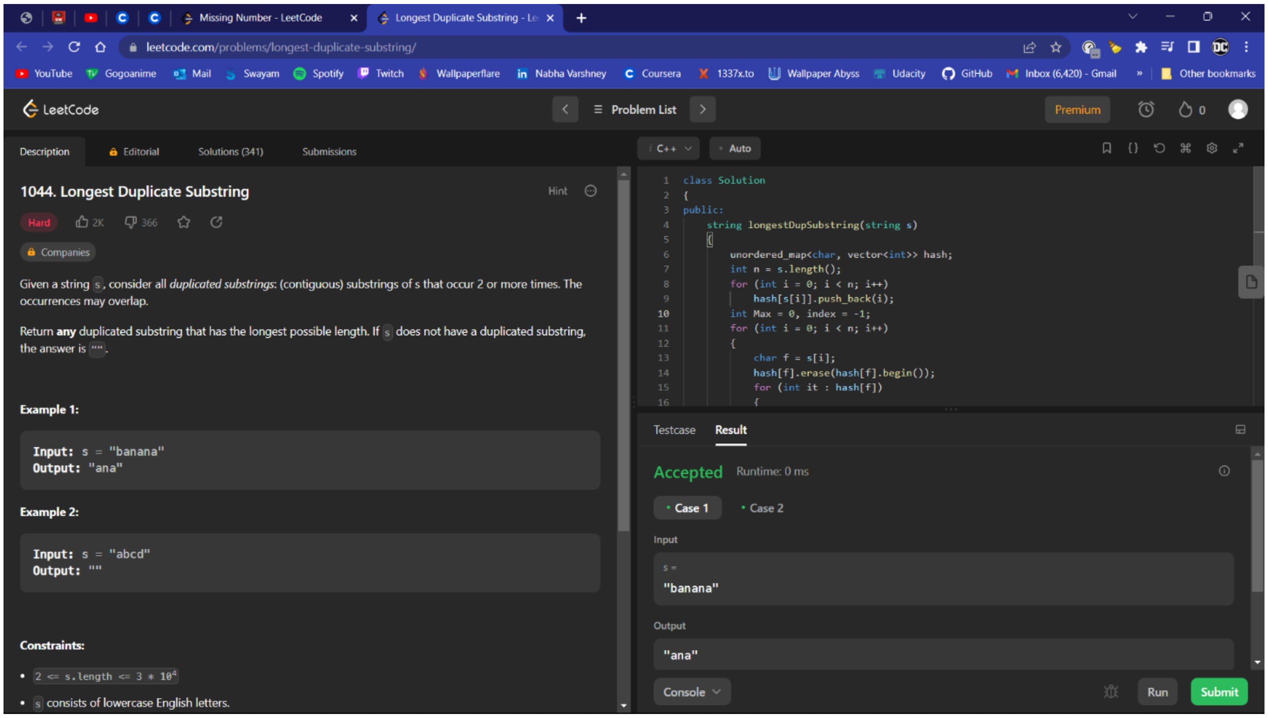
2) Longest Duplicate Substring

https://leetcode.com/problems/longest-duplicate-substring/

Code -

```
class Solution
{
public:
    string longestDupSubstring(string s)
    {
        unordered_map<char, vector<int>> hash;
        int n = s.length();
        for (int i = 0; i < n; i++)
            hash[s[i]].push_back(i);
        int Max = 0, index = -1;
        for (int i = 0; i < n; i++)
        {
            char f = s[i];
            hash[f].erase(hash[f].begin());
            for (int it : hash[f])
            {
                int j = 0;
                while (i + j < n and it + j < n and s[i + j] == s[it + j])
                    j++;
                if (j > Max)
                {
                    Max = j;
                    index = i;
                }
                if (Max == n - i - 1)
                    return s.substr(index, Max);
            }
        }
        if (Max == 0)
            return "";
        else
            return s.substr(index, Max);
}};
```

Output –



1044. Longest Duplicate Substring

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: ""
```

Constraints:

- `2 <= s.length <= 3 * 10^4`
- s consists of lowercase English letters.

```
class Solution
{
public:
    string longestDupSubstring(string s)
    {
        unordered_map<char, vector<int>> hash;
        int n = s.length();
        for (int i = 0; i < n; i++)
            hash[s[i]].push_back(i);
        int Max = 0, index = -1;
        for (int i = 0; i < n; i++)
        {
            char f = s[i];
            hash[f].erase(hash[f].begin());
            for (int it : hash[f])
            {
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

s =
"banana"

Output

"ana"