Experiment – 1.4

Student Name: Avinash Jena **UID: 20BCS2690**

Branch: CSE **Section/Group: 707/B**

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

Aim: Missing Number

Objective:

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.

Code:

```
class Solution {
    public int missingNumber(int[] nums) {
        int sum = 0;
        int numsSize = nums.length;

        for(int num : nums) {
            sum += num;
        }

        return (numsSize * (numsSize + 1)) / 2 - sum;
    }
}
```

Output:

Java Auto

```java
class Solution {
    public int missingNumber(int[] nums) {
        int sum = 0;
        int numsSize = nums.length;

        for(int num : nums) {
            sum += num;
        }

        return (numsSize * (numsSize + 1)) / 2 - sum;
    }
}
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =
[3,0,1]

Output

2

Expected

2

Console Run Submit

Aim: Longest Duplicate Substring

Objective:

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 * 104
s consists of lowercase English letters.

Code:

```
class Solution {
    private static final long q = (1 << 31) - 1;
    private static final long R = 256;

    public String longestDupSubstring(String S) {
        int left = 2;
        int right = S.length() - 1;
        int start = 0;
        int maxLen = 0;

        while (left <= right) {
            int len = left + (right - left) / 2;
            boolean found = false;

            Map<Long, List<Integer>> map = new HashMap<>();
            long hash = hash(S, len);
            map.put(hash, new ArrayList<>());
            map.get(hash).add(0);
            long RM = 1l;
            for (int i = 1; i < len; i++) RM = (R * RM) % q;

            loop:
            for (int i = 1; i + len <= S.length(); i++) {
                hash = (hash + q - RM * S.charAt(i - 1) % q) % q;
                hash = (hash * R + S.charAt(i + len - 1)) % q;
                if (!map.containsKey(hash)) {
                    map.put(hash, new ArrayList<>());
                } else {
                    for (int j : map.get(hash)) {
                        if (compare(S, i, j, len)) {
                            found = true;
                            start = i;
                            maxLen = len;
                            break loop;
                        }
                    }
                }
```

```
                map.get(hash).add(i);
            }
            if (found) left = len + 1;
            else right = len - 1;
        }

        return S.substring(start, start + maxLen);
    }

    private long hash(String S, int len) {
        long h = 0;
        for (int j = 0; j < len; j++) h = (R * h + S.charAt(j)) % q;
        return h;
    }

    private boolean compare(String S, int i, int j, int len) {
        for (int count = 0; count < len; count++) {
            if (S.charAt(i++) != S.charAt(j++)) return false ;
        }
        return true ;
    }
}
```

Output:

Java Auto

```java
class Solution {
    private static final long q = (1 << 31) - 1;
    private static final long R = 256;

    public String longestDupSubstring(String S) {
        int left = 2;
        int right = S.length() - 1;
        int start = 0;
        int maxLen = 0;

        while (left <= right) {
            int len = left + (right - left) / 2;
            boolean found = false;

            Map<Long, List<Integer>> map = new HashMap<>();
            long hash = hash(S, len);
            map.put(hash, new ArrayList<>());
            map.get(hash).add(0);
            long RM = 1l;
            for (int i = 1; i < len; i++) RM = (R * RM) % q;

            loop:
            for (int i = 1; i + len <= S.length(); i++) {
                hash = (hash + q - RM * S.charAt(i - 1) % q) % q;
                hash = (hash * R + S.charAt(i + len - 1)) % q;
                if (!map.containsKey(hash)) {
                    map.put(hash, new ArrayList<>());
                } else {
                    for (int j : map.get(hash)) {
                        if (compare(S, i, j, len)) {
                            found = true;
                            start = i;
                            maxLen = len;
                            break loop;
                        }
                    }
                }
                map.get(hash).add(i);
            }
            if (found) left = len + 1;
            else right = len - 1;
        }
```

Testcase Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

s =
"banana"

Output

"ana"

Expected

"ana"

Console Run Submit