Experiment 1.1

Student Name: HARSHIT RAJ	**UID:** 20BC9266
Branch: CSE	**Section/Group:** 608-A
Semester: 6th	**Date of Performance:** 20/02/23
Subject Name: Competitive Coding-II	**Subject Code:** 20CSP-351

Aim: Implementing the concepts of Arrays, Stacks, Queues linked list

Question 1

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0]. Each Element nums[i] represents the maximum length of a forward index i, in other words, if you are at nums[i], you can jump to any nums[i+j] where:

0 <= j <= nums[i] and

i + j < n

Return the minimum number of jumps to reach nums[n-1]. The test cases are generated such that you can reach nums[n-1]

Intuition

Advice: Frist ,You need to analyse the question to which data structure to apply for this solution,It will come automatically,when you solve more problems.

Approach

My first Approach was Dynammic programming,but solution beats 50%.
I moved to greedy method and Beats 95%.

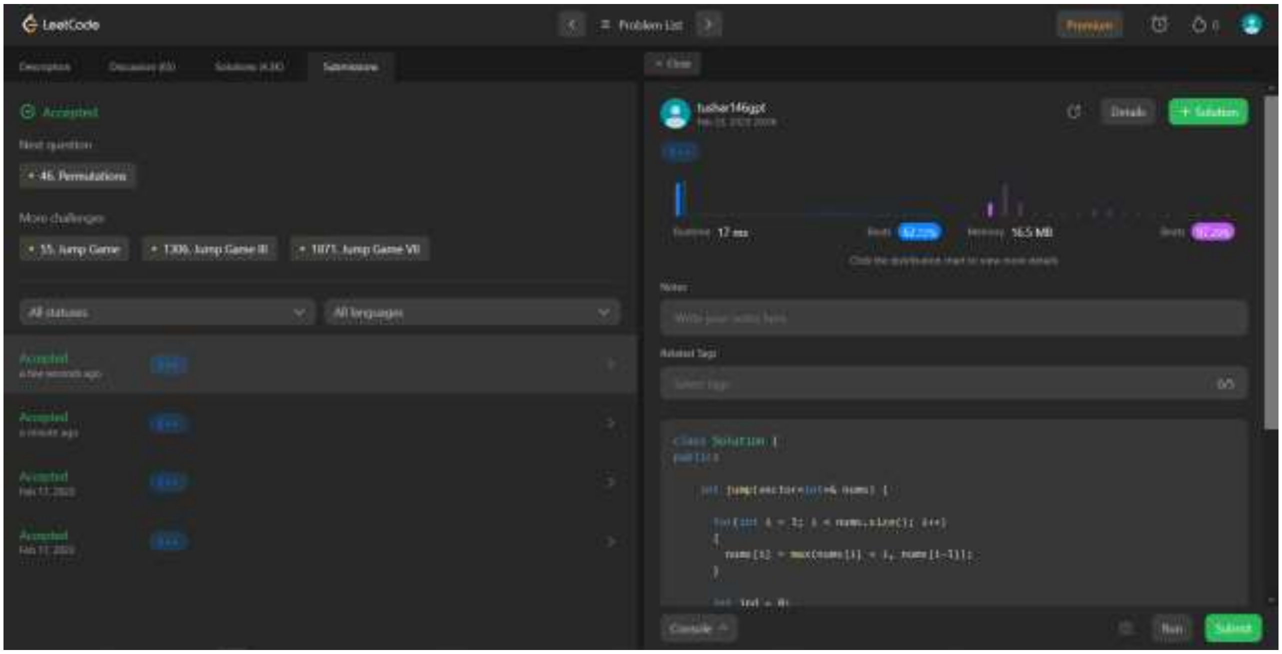
Complexity

- Time complexity:O(N)
- Space complexity:O(1)

Code and output:

```
class Solution {
public:
     int solve(vector<int>& nums, int index, vector<int>& dp){
        int n = nums.size();
        if(index == n-1) return 0;
        if(nums[index] == 0) return INT_MAX;
        if(dp[index] != -1) return dp[index];

        int minJumps = INT_MAX;
        for(int i = index+1; i <= min(nums[index]+index, n-1); i++){
            int jump = solve(nums, i, dp);
            if(jump != INT_MAX){
                minJumps = min(minJumps, jump+1);
            }
        }

        return dp[index] = minJumps;
    }

    int solveTab(vector<int>& nums){
        int n = nums.size();
        vector<int> jumps(n, 0);

        for (int i = 1; i < n; i++) {
            jumps[i] = INT_MAX;
            for (int j = 0; j < i; j++) {
                if (i <= j + nums[j] && jumps[j] != INT_MAX) {
                    jumps[i] = min(jumps[i], jumps[j] + 1);
                    break;
                }
            }
        }

        return jumps[n-1];
    }

    int jump(vector<int>& nums) {
        vector<int> dp(nums.size(), -1);
```

```
            return solve(nums, 0, dp);
        }
    };
```



```
class Solution {
public:

    int jump(vector<int>& nums) {

      for(int i = 1; i < nums.size(); i++)
      {
        nums[i] = max(nums[i] + i, nums[i-1]);
      }

      int ind = 0;
```

Question 2

Given the head of the sorted linked list, delete all the duplicates such that eachelement appears only once. Return the sorted linked list as well.

Intuition

We can solve this question using Linklist + Two Pointer.

Approach

We can easily understand the approache by seeing the code which is easy to understand with comments.

Complexity

- Time complexity:

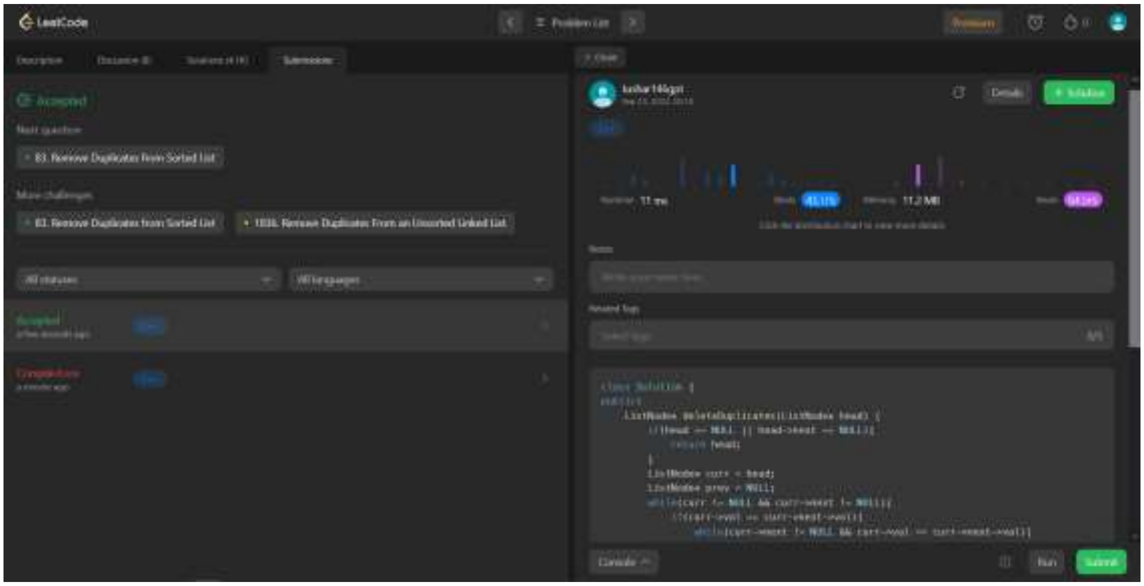
Time Complexity :O(N), because we are traversing over the elements exactly one. Thus the time complexity is linear.

- Space complexity:

Space Complexity : O(1), because we have used constant elements. Thus the space complexity is constant.

Code and Output:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head == NULL) return head;
        if(head->next == NULL) return head;
        ListNode* p = head;
        int temp = head->val;
        while(p->next != NULL){
            if(p->val == p->next->val){
                p->next = p->next->next;


            }
            else{
                p = p->next;
            }
        }
        return head;

    }
};
```



```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head == NULL || head->next == NULL){
            return head;
        }
        ListNode* curr = head;
        ListNode* prev = NULL;
        while(curr != NULL && curr->next != NULL){
            if(curr->val == curr->next->val){
                while(curr->next != NULL && curr->val == curr->next->val){
```