

---

## Experiment No. - 1

**Student Name: Vivek Kumar**

**Branch: BE-CSE(LEET)**

**Semester: 6<sup>th</sup>**

**Subject Name: Competitive coding - II**

**UID: 21BCS8129**

**Section/Group:20BCS-ST-801/B**

**Date of Performance: 14/02/2023**

**Subject Code: 20CSP-351**

**1. Aim/Overview of the practical:**

**Q1. Jump Game II**

<https://leetcode.com/problems/jump-game-ii/>

**2. Apparatus / Simulator Used:**

1. Windows 7 or above
2. Google Chrome

**3. Objective:**

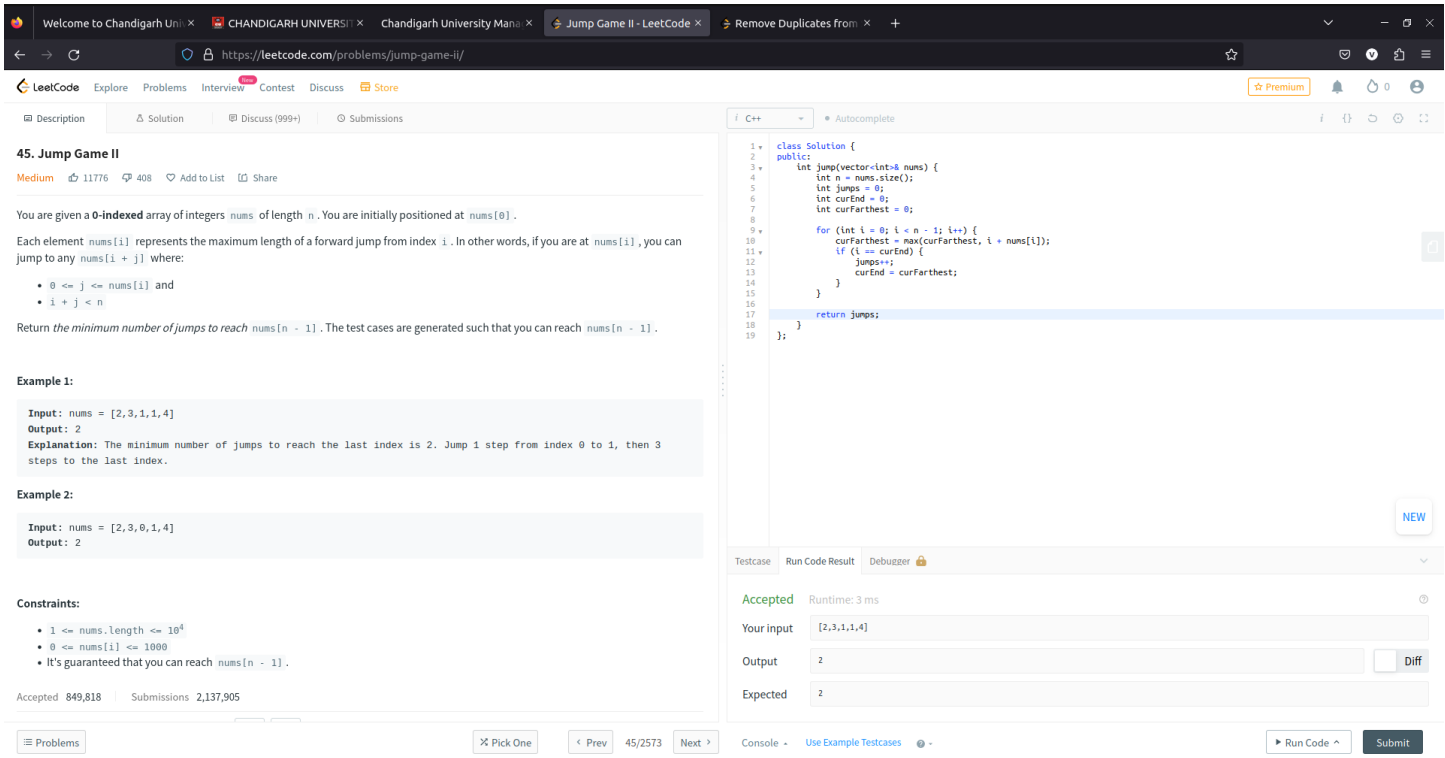
- a. To understand the concept of Array and Jump Concept
- b. To implement the concept of Array Implementation.

**4. Code:**

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int n = nums.size();
        int jumps = 0;
        int curEnd = 0;
        int curFarthest = 0;

        for (int i = 0; i < n - 1; i++) {
            curFarthest = max(curFarthest, i + nums[i]);
            if (i == curEnd) {
                jumps++;
                curEnd = curFarthest;
            }
        }
        return jumps;
    }
};
```

## 5. Result/Output/Writing Summary:



**45. Jump Game II**  
Medium 11776 408 Add to List Share

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 jump from index `i`. In other words, if you are at `nums[i]`, you can jump to any `nums[i + j]` where:

- $0 \leq j \leq \text{nums}[i]$  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]`.

**Example 1:**

Input: `nums = [2,3,1,1,4]`  
Output: 2  
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Example 2:**

Input: `nums = [2,3,0,1,4]`  
Output: 2

**Constraints:**

- $1 \leq \text{nums.length} \leq 10^4$
- $0 \leq \text{nums}[i] \leq 1000$
- It's guaranteed that you can reach `nums[n - 1]`.

Accepted 849,818 Submissions 2,137,905

```

1 class Solution {
2 public:
3     int jump(vector<int>& nums) {
4         int n = nums.size();
5         int jumps = 0;
6         int curEnd = 0;
7         int curFarthest = 0;
8
9         for (int i = 0; i < n - 1; i++) {
10             curFarthest = max(curFarthest, i + nums[i]);
11             if (i == curEnd) {
12                 jumps++;
13                 curEnd = curFarthest;
14             }
15         }
16         return jumps;
17     }
18 };

```

Testcase Run Code Result Debuzzer

Accepted Runtime: 3 ms

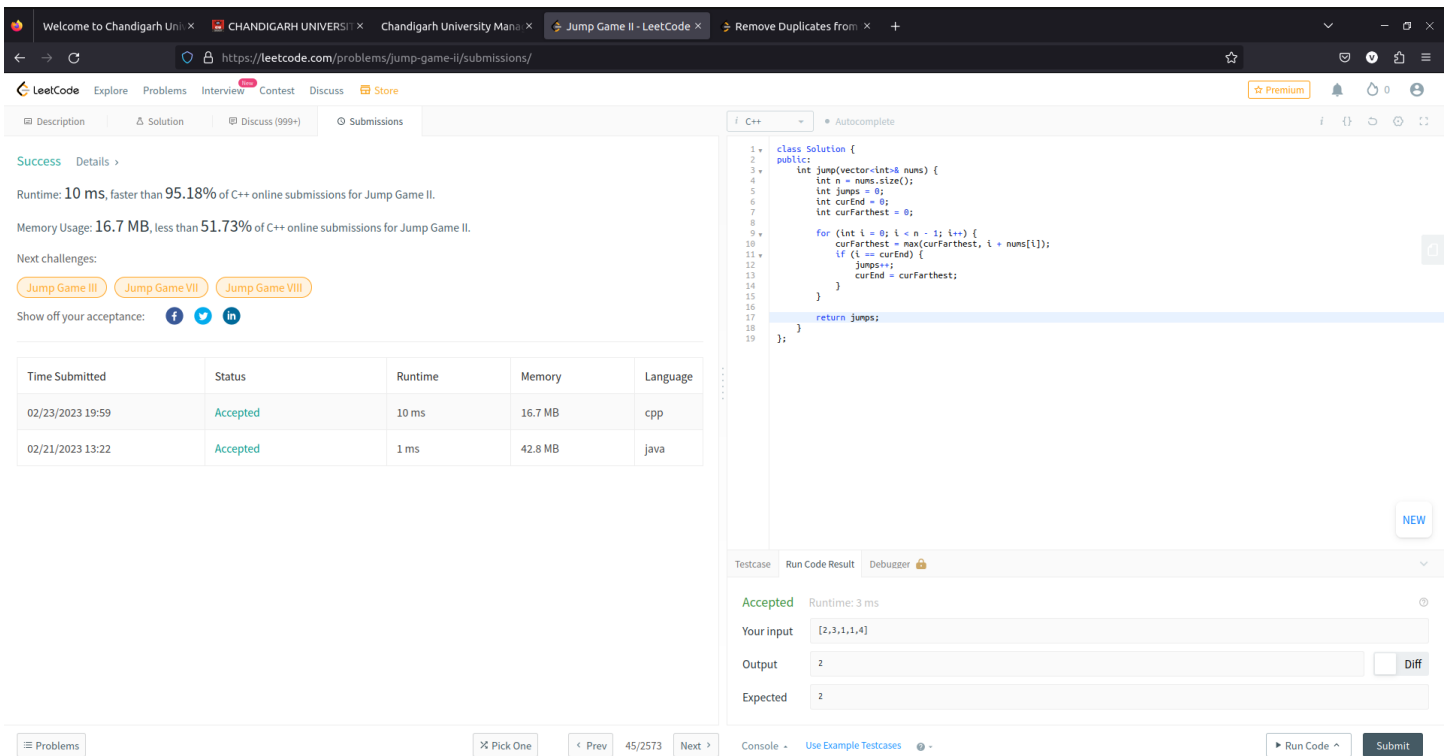
Your input [2,3,1,1,4]

Output 2

Expected 2

Diff

Run Code Submit



**Success** Details

Runtime: 10 ms, faster than 95.18% of C++ online submissions for Jump Game II.

Memory Usage: 16.7 MB, less than 51.73% of C++ online submissions for Jump Game II.

Next challenges: [Jump Game III](#) [Jump Game VII](#) [Jump Game VIII](#)

Show off your acceptance: [Facebook](#) [Twitter](#) [LinkedIn](#)

Time Submitted	Status	Runtime	Memory	Language
02/23/2023 19:59	Accepted	10 ms	16.7 MB	cpp
02/21/2023 13:22	Accepted	1 ms	42.8 MB	java

```

1 class Solution {
2 public:
3     int jump(vector<int>& nums) {
4         int n = nums.size();
5         int jumps = 0;
6         int curEnd = 0;
7         int curFarthest = 0;
8
9         for (int i = 0; i < n - 1; i++) {
10             curFarthest = max(curFarthest, i + nums[i]);
11             if (i == curEnd) {
12                 jumps++;
13                 curEnd = curFarthest;
14             }
15         }
16         return jumps;
17     }
18 };

```

Testcase Run Code Result Debuzzer

Accepted Runtime: 3 ms

Your input [2,3,1,1,4]

Output 2

Expected 2

Diff

Run Code Submit

1. **Aim/Overview of the practical:**

**Q2. Merge Two Sorted List**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/>

2. **Apparatus / Simulator Used:**

1. Windows 7 or above
2. Google Chrome

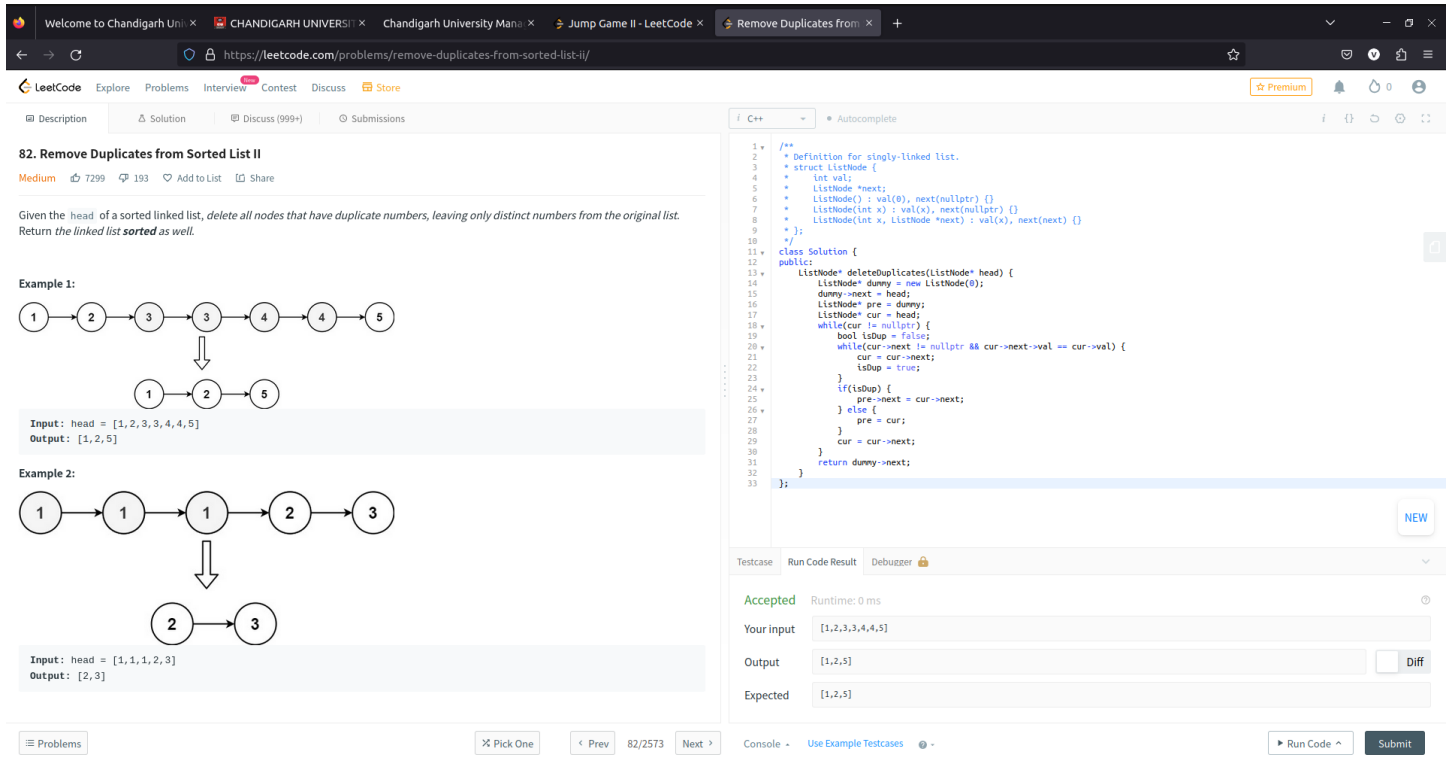
3. **Objective:**

- To understand the concept of List and Node
- To implement the concept of Remove duplicates from the list

4. **Code:**

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* pre = dummy;
        ListNode* cur = head;
        while(cur != nullptr) {
            bool isDup = false;
            while(cur->next != nullptr && cur->next->val == cur->val) {
                cur = cur->next;
                isDup = true;
            }
            if(isDup) {
                pre->next = cur->next;
            } else {
                pre = cur;
            }
            cur = cur->next;
        }
        return dummy->next;
    }
};
```

## 5. Result/Output/Writing Summary:



**82. Remove Duplicates from Sorted List II**

Medium 7299 193 Add to List Share

Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

**Example 1:**

Input: head = [1,2,3,3,4,4,5]  
Output: [1,2,5]

**Example 2:**

Input: head = [1,1,1,2,3]  
Output: [2,3]

```

1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         ListNode* dummy = new ListNode(0);
15         dummy->next = head;
16         ListNode* pre = dummy;
17         ListNode* cur = head;
18         while(cur != nullptr) {
19             bool isDup = false;
20             while(cur->next != nullptr && cur->next->val == cur->val) {
21                 cur = cur->next;
22                 isDup = true;
23             }
24             if(!isDup) {
25                 pre->next = cur->next;
26             } else {
27                 pre = cur;
28             }
29             cur = cur->next;
30         }
31         return dummy->next;
32     }
33 };

```

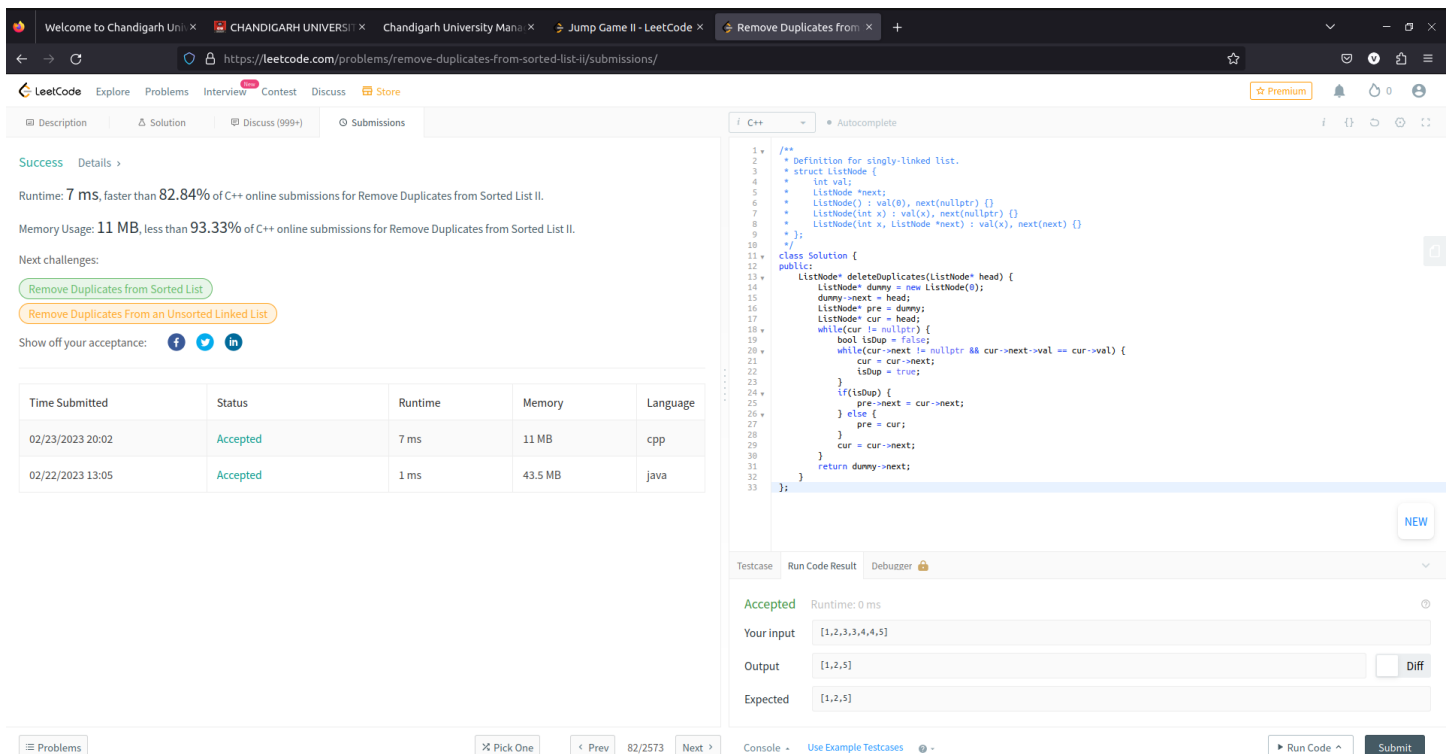
Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [1,2,3,3,4,4,5]

Output [1,2,5]

Expected [1,2,5]



Success Details

Runtime: 7 ms, faster than 82.84% of C++ online submissions for Remove Duplicates from Sorted List II.

Memory Usage: 11 MB, less than 93.33% of C++ online submissions for Remove Duplicates from Sorted List II.

Next challenges:

- Remove Duplicates from Sorted List
- Remove Duplicates From an Unsorted Linked List

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
02/23/2023 20:02	Accepted	7 ms	11 MB	cpp
02/22/2023 13:05	Accepted	1 ms	43.5 MB	java

```

1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         ListNode* dummy = new ListNode(0);
15         dummy->next = head;
16         ListNode* pre = dummy;
17         ListNode* cur = head;
18         while(cur != nullptr) {
19             bool isDup = false;
20             while(cur->next != nullptr && cur->next->val == cur->val) {
21                 cur = cur->next;
22                 isDup = true;
23             }
24             if(!isDup) {
25                 pre->next = cur->next;
26             } else {
27                 pre = cur;
28             }
29             cur = cur->next;
30         }
31         return dummy->next;
32     }
33 };

```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [1,2,3,3,4,4,5]

Output [1,2,5]

Expected [1,2,5]

### Learning outcomes (What I have learnt):

- Learned the concept of LinkedList.
- Learnt about Removing Duplicates from the List.

### Evaluation Grid (To be created per the faculty's SOP and Assessment guidelines):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Worksheet completion including writing learning objectives/Outcomes. (To be submitted at the end of the day).		
2.	Post-Lab Quiz Result.		
3.	Student Engagement in Simulation/Demonstration/Performance and Controls/Pre-Lab Questions.		
	Signature of Faculty (with Date):	Total Marks Obtained:	