
Experiment No. - 1

Student Name: Vivek Kumar
Branch: BE-CSE(LEET)
Semester: 6th
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

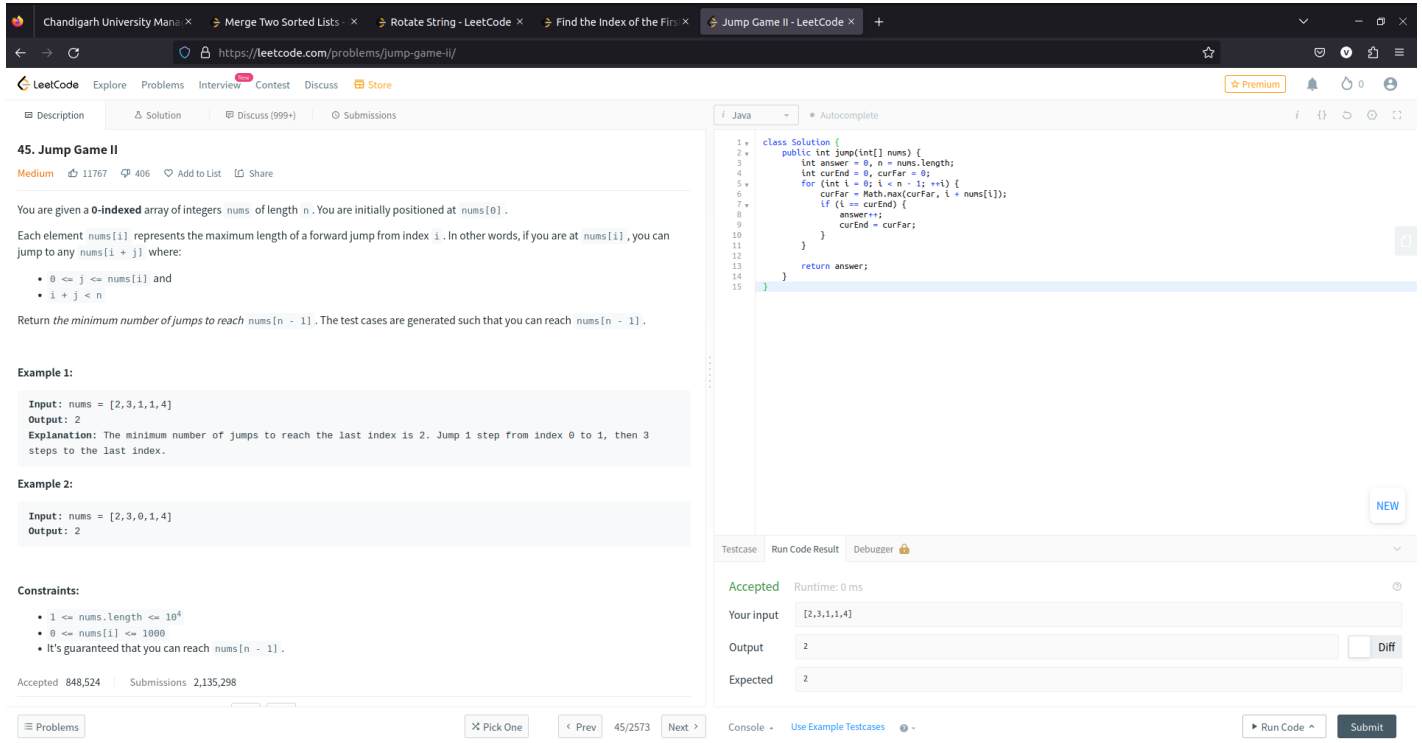
3. Objective:

- 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(int[] nums) {  
        int answer = 0, n = nums.length;  
        int curEnd = 0, curFar = 0;  
        for (int i = 0; i < n - 1; ++i) {  
            curFar = Math.max(curFar, i + nums[i]);  
            if (i == curEnd) {  
                answer++;  
                curEnd = curFar;  
            }  
        }  
        return answer;  
    }  
}
```

5. Result/Output/Writing Summary:



45. Jump Game II
Medium 11767 406 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 848,524 Submissions 2,135,298

```

1  class Solution {
2      public int jump(int[] nums) {
3          int answer = 0, n = nums.length;
4          int curEnd = 0, curFar = 0;
5          for (int i = 0; i < n - 1; ++i) {
6              curFar = Math.max(curFar, i + nums[i]);
7              if (i == curEnd) {
8                  answer++;
9                  curEnd = curFar;
10             }
11         }
12         return answer;
13     }
14 }

```

Testcase Run Code Result Debuzzer

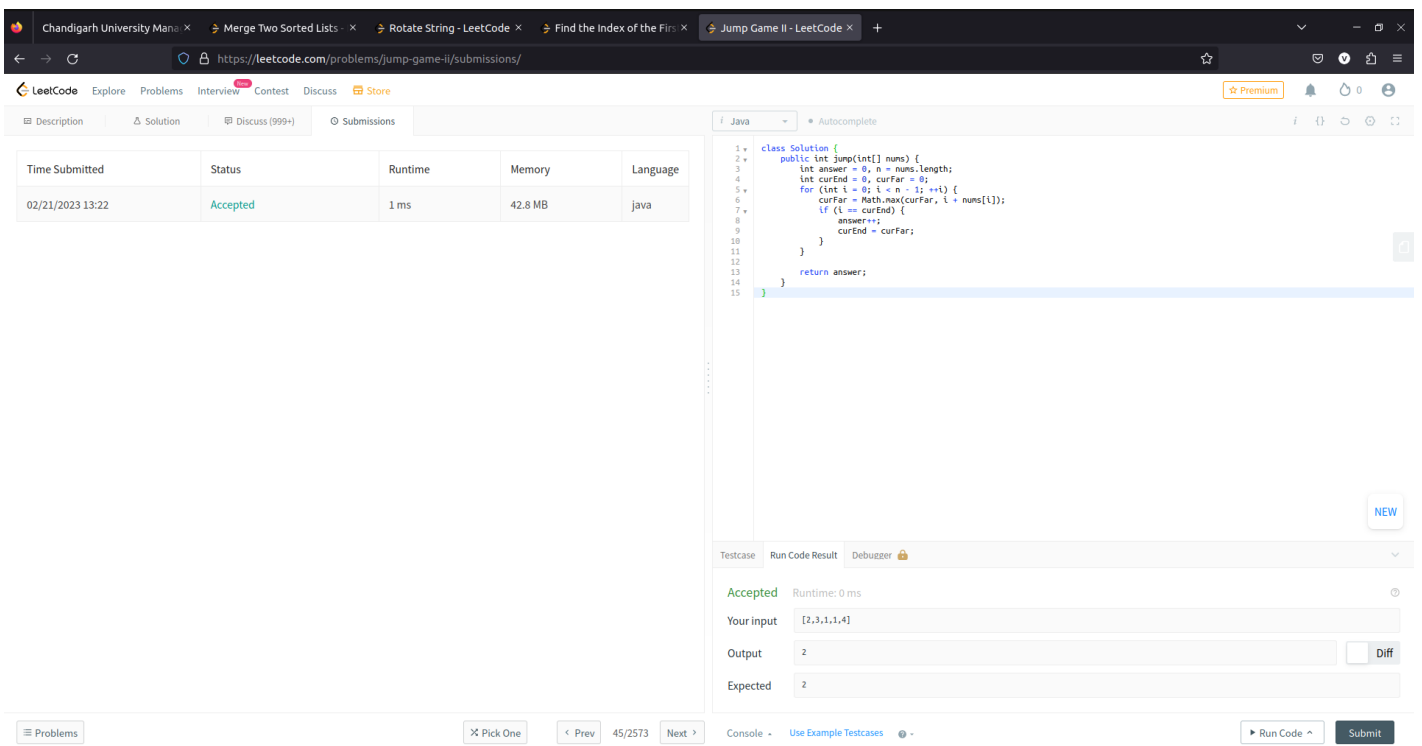
Accepted Runtime: 0 ms

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

Output `2` Diff

Expected `2`

Console Use Example Testcases Run Code Submit



45. Jump Game II
Medium 11767 406 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 848,524 Submissions 2,135,298

Time Submitted	Status	Runtime	Memory	Language
02/21/2023 13:22	Accepted	1 ms	42.8 MB	java

```

1  class Solution {
2      public int jump(int[] nums) {
3          int answer = 0, n = nums.length;
4          int curEnd = 0, curFar = 0;
5          for (int i = 0; i < n - 1; ++i) {
6              curFar = Math.max(curFar, i + nums[i]);
7              if (i == curEnd) {
8                  answer++;
9                  curEnd = curFar;
10             }
11         }
12         return answer;
13     }
14 }

```

Testcase Run Code Result Debuzzer

Accepted Runtime: 0 ms

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

Output `2` Diff

Expected `2`

Console Use Example Testcases 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 sentinel = new ListNode(0, head);
        ListNode pred = sentinel;

        while (head != null) {
            if (head.next != null && head.val == head.next.val) {
                while (head.next != null && head.val == head.next.val) {
                    head = head.next;
                }
                pred.next = head.next;
            } else {
                pred = pred.next;
            }

            head = head.next;
        }
        return sentinel.next;
    }
}
```

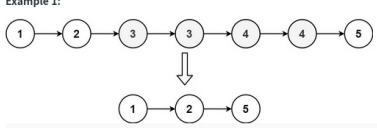
5. Result/Output/Writing Summary:

Remove Duplicates from Sorted List II

Medium 7297 193 Add to List Share

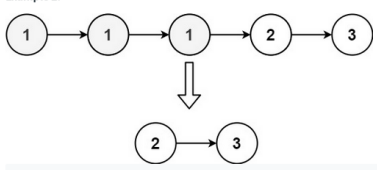
Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

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  * public class ListNode {
4  *     int val;
5  *     ListNode next;
6  *     ListNode() {}
7  *     ListNode(int val) { this.val = val; }
8  *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9  * }
10 */
11 class Solution {
12     public ListNode deleteDuplicates(ListNode head) {
13         ListNode sentinel = new ListNode(0, head);
14         ListNode pred = sentinel;
15
16         while (head != null) {
17             if (head.next != null && head.val == head.next.val) {
18                 while (head.next != null && head.val == head.next.val) {
19                     head = head.next;
20                 }
21                 pred.next = head.next;
22             } else {
23                 pred = pred.next;
24             }
25             head = head.next;
26         }
27         return sentinel.next;
28     }
29 }

```

Accepted Runtime: 0 ms

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

Output [1,2,5]

Expected [1,2,5]

Remove Duplicates from Sorted List II

Success Details

Runtime: 1 ms, faster than 25.82% of Java online submissions for Remove Duplicates from Sorted List II.

Memory Usage: 43.5 MB, less than 11.37% of Java 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
02/22/2023 13:05	Accepted	1 ms	43.5 MB	java

```

1  /**
2  * Definition for singly-linked list.
3  * public class ListNode {
4  *     int val;
5  *     ListNode next;
6  *     ListNode() {}
7  *     ListNode(int val) { this.val = val; }
8  *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9  * }
10 */
11 class Solution {
12     public ListNode deleteDuplicates(ListNode head) {
13         ListNode sentinel = new ListNode(0, head);
14         ListNode pred = sentinel;
15
16         while (head != null) {
17             if (head.next != null && head.val == head.next.val) {
18                 while (head.next != null && head.val == head.next.val) {
19                     head = head.next;
20                 }
21                 pred.next = head.next;
22             } else {
23                 pred = pred.next;
24             }
25             head = head.next;
26         }
27         return sentinel.next;
28     }
29 }

```

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.

Submitted By: Vivek Kumar

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:	