

Experiment1.1

Student Name: Anshuman Singh UID: 20BCS2665

Explanation: The only possible triplet does not sum up to 0.

Branch: CSE Section/Group: 902/A

Semester: 6th Date of Performance: 08-02-2023

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

Aim: https://leetcode.com/problems/3sum/ Description Discussion (136) Solutions (6.5K) Submissions 15. 3Sum Medium △ 24.1K ♀ 2.2K ☆ ♂ Companies Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets. Example 1: **Input:** nums = [-1,0,1,2,-1,-4]Output: [[-1,-1,2],[-1,0,1]] Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter. Example 2: **Input:** nums = [0,1,1]Output: []

Discover. Learn. Empower.

```
Example 3:
  Input: nums = [0,0,0]
  Output: [[0,0,0]]
  Explanation: The only possible triplet sums up to 0.
Constraints:
• 3 <= nums.length <= 3000
• -10^5 \le nums[i] \le 10^5
```

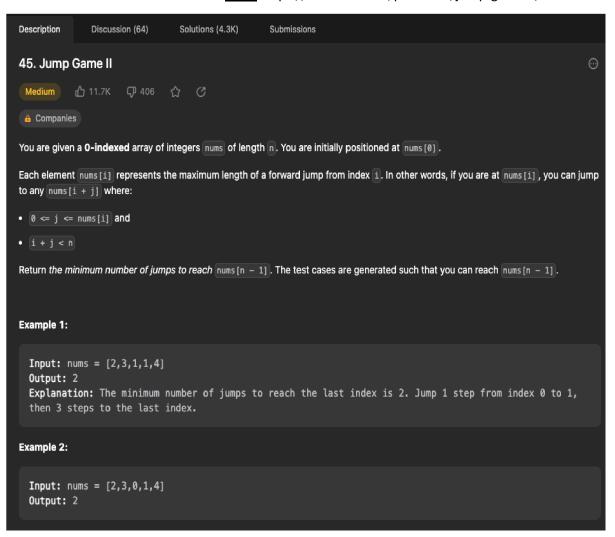
Code:-

```
class Solution {
  public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    Arrays.sort(nums);
    for (int i = 0; i + 2 < nums.length; i++) {
       if (i > 0 \&\& nums[i] == nums[i - 1]) {
         continue;
       int j = i + 1, k = nums.length - 1;
       int target = -nums[i];
       while (j < k) {
         if (nums[j] + nums[k] == target) {
           res.add(Arrays.asList(nums[i], nums[j], nums[k]));
           j++;
           k--;
           while (j < k \&\& nums[j] == nums[j - 1]) j++;
            while (j < k \&\& nums[k] == nums[k + 1]) k--;
         } else if (nums[j] + nums[k] > target) {
            k--;
         } else {
           j++;
      }
    return res;
```

Output:-



Aim: https://leetcode.com/problems/jump-game-ii/



Code:-

class Solution {
public int jump(int[] nums) {
int n=nums.length;
int[] dp=new int[n];
Arrays.fill(dp,Integer.MAX_VALUE);
dp[n-1]=0;//from last position to last position you need 0 jumps
for(int i=n-2;i>-1;i)
{
int min=Integer.MAX_VALUE;
for(int j=i+1;j<=Math.min(n-1,i+nums[i]);j++)//from that index
till the maximum index it can jump,check which will take minimum steps to
reach the end
/*we are taking Math.min(n-1,i+nums[i])
reason:2 3 5 1 1
here if i+nums[i]=2+5=7 you can jump till 7th position
but size of array is only 5 so we can just check jumps till n-1 thus we are
taking Math.min(n-1,i+nums[i])*/
{
min=Math.min(min,dp[j]);
}
if(min!=Integer.MAX_VALUE)
dp[i]=min+1;
}
return dp[0];
$ \mathbf{P} $

Output:-

