**Array**

**Easy**

***Problem 1*** *:- 1 Two Sum*

*Link:* <https://leetcode.com/problems/two-sum/>

Statement :

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to *target*.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

Solution 1:

Sort the array and declare two pointers on front and back apply two pointer algorithm TC: O(nLogn) SC: O(1) considering

TC is O(nlogn) because sorting best case is O(nlogn) and traversing using two pointers O(n) so O(nlogn) + O(n) = O(nlogn) SC is O(1) if we don’t use any extra space for sorting the array but if we use merge sort or quick sort we may end up using O(n) auxillary space because of recursion

Solution 2:

Declare a hashmap and traverse each element and check if target – currentelement is present in the hashmap or not

If present store their indexes in an result array/set/list and break there

Else add the element as key and index as value in the hashmap

*Code:* [*https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/\_1TwoSum.java*](https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_1TwoSum.java)

*TC:O(N) SC: O(N)*

Considering hashmap has no collisions so time complexity would be O(1\*n) which is O(n) and space complexity would be O(n) because we may end up storing all elements in the hashmap

TC is time complexity and SC is space complexity

***Problem 2*** *:- 121 Best time to buy and sell stock*

*Link:* <https://leetcode.com/problems/best-time-to-buy-and-sell-stock/>

Statement:

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Solution:

Declare minUntilNow as first element of the array and maxProfit as 0

Traverse everyelement and first check if currProfit which is currentElement – minUntilNow is greater than maxProfit then assign maxProfit as currProfit

Then check if current element is smaller than minUntilNow then assign minUntilNow as current element

After all the elements are traversed then return the maxProfit

*Code:* [*https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/\_2BestTimeToBuyAndSellStock.java*](https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_2BestTimeToBuyAndSellStock.java)

TC: O(n) and SC:O(1)

***Problem 3****: 88 Merge Sorted Array*

Link: <https://leetcode.com/problems/merge-sorted-array/>

Statement:

You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

**Merge** nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array*nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

**Example 1:**

**Input:** nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

**Output:** [1,2,2,3,5,6]

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Solution:

Declare two pointers m and n such that m is the last non zero index in the array 1 and n is the last element in the array2 and end as last element of array1

Run a while loop m >= 0 && n >= 0 such that if array1[m] > array2[n] then array1[end] is assigned as array1[m] and m,end are decremented

Or else array1[end] is assigned as array2[n] and n,end are decremented

In the end if any of the array2 elements left then array1[end] = array2[n] and end,n are decremented this is done because it fills the remaining elements

*Code:* [*https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/\_3MergeSortedArray.java*](https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_3MergeSortedArray.java)

TC: O(len1) because we traverse every element of both the arrays so O(len1+len2) becomes O(len1) because len1 is larger and SC: O(1)

***Problem 4****: 283 Move Zeroes*

Link: <https://leetcode.com/problems/move-zeroes/>

Statement:

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

**Note** that you must do this in-place without making a copy of the array.

**Example 1:**

**Input:** nums = [0,1,0,3,12]

**Output:** [1,3,12,0,0]

Solution:

Declare a pointer ind as 0

Traverse the array and if you found non zero element assign array[ind] = array[current index in loop] and then ind is incremented

After the end of the loop run a while loop until the ind is less than length of array

Set array[ind] as 0

*Code:* [*https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/\_4MoveZeroes.java*](https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_4MoveZeroes.java)

TC: O(n) because we traverse the array once for every element and SC: O(1)

***Problem 5****: 122 Best Time to Buy and Sell Stock II*

*Link:* [*https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/*](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/)

Statement:

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return the ***maximum*** profit you can achieve.

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

Solution 1:

If current element lesser than previous element we can add the difference of those both elements to the total profit

Code : <https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_5BestTimeToBuyAndSellStock.java>

***Problem 6****: 1480 Running Sum*

Easy Not required to solve

***Problem 7****: 724 Find Pivot Index*

Easy Not required to solve

***Problem 8****: 169 Majority element*

Solution: Traverse all elements in an array while adding them and their counts to a map and if any element count is greater than array length/2 then return it

After the loop if no majority element is returned then return -1 which indicates no majority element

***Problem 9****: 509 Fibonacci number*

Easy Not required to solve

**Problem 10**: 977 Squares of a sorted array

Link : <https://leetcode.com/problems/squares-of-a-sorted-array/>

Solution: take two pointers on left and right side and if absolute of one is greater then square that and add it to the end of result array and modify pointers

**Problem 11**: 118 Pascal’s Triangle

Link: <https://leetcode.com/problems/pascals-triangle/>

Solution: Easy one

**Problem 12**: 26 Remove duplicates from sorted array

Link: <https://leetcode.com/problems/remove-duplicates-from-sorted-array/>

Solution:

Declare index as 0 which is start of the array

Iterate the array from second element using j

If you find an element at j which is not equal to element at index then increment the index and assign the jth element to index place in the end return index+1 because no of elements is index+1 which are unique

Code: <https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_12RemoveDuplicatesFromSortedArray.java>

**Medium**

***Problem 13****: 56 Merge Intervals*

*Link:* [*https://leetcode.com/problems/merge-intervals/*](https://leetcode.com/problems/merge-intervals/)

*Description:*

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

**Example 1:**

**Input:** intervals = [[1,3],[2,6],[8,10],[15,18]]

**Output:** [[1,6],[8,10],[15,18]]

**Explanation:** Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

**Example 2:**

**Input:** intervals = [[1,4],[4,5]]

**Output:** [[1,5]]

**Explanation:** Intervals [1,4] and [4,5] are considered overlapping.

Solution:

Sort the array and the compare first two elements if they can be merged

Then add the new range from start1,end2 and then compare this to others and merge if possible and so on until all the non mergeable ranges are added in the result

Code: <https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_13MergeIntervals.java>

***Problem 14****: 15 3Sum*

*Link:* [*https://leetcode.com/problems/3sum/*](https://leetcode.com/problems/3sum/)

*Description:*

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.

Solution:

Sort the array

Run a for loop take the first element and then run a two sum logic for others

Take the first element as the negation of itself and call it target

If other 2 numbers sum is equal to target then we found the triplet we add that to the result list and then we skip the 2 numbers and then go to next unique numbers to find the triplet

*Code:* <https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_14_3Sum.java>

TC : O(nlogn) for sorting and O(n\*n) for iterating 2 sum for every element so O(nlogn) + O(n\*n) becomes O(n\*n)

SC: O(1) because we are not using any extra space to solve the result but we may use O(n) to store the result

***Problem 15****: 238 Product of Array Except Self*

*Link:* [*https://leetcode.com/problems/product-of-array-except-self/*](https://leetcode.com/problems/product-of-array-except-self/)

*Description:*

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

**Example 1:**

**Input:** nums = [1,2,3,4]

**Output:** [24,12,8,6]

Solution:

First we go from beginning and find the prefix product

And then come from backside to product the postfix to get the desired result

Code: <https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_15_ProductOfArrayExceptSelf.java>

TC: O(n)

SC: O(n) for result else O(1) for solving the problem

***Problem 16:*** *380 Insert Delete GetRandom O(1)*

*Link:* [*https://leetcode.com/problems/insert-delete-getrandom-o1/*](https://leetcode.com/problems/insert-delete-getrandom-o1/)

*Description:*

Implement the RandomizedSet class:

* RandomizedSet() Initializes the RandomizedSet object.
* bool insert(int val) Inserts an item val into the set if not present. Returns true if the item was not present, false otherwise.
* bool remove(int val) Removes an item val from the set if present. Returns true if the item was present, false otherwise.
* int getRandom() Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the **same probability** of being returned.

You must implement the functions of the class such that each function works in **average** O(1) time complexity.

**Example 1:**

**Input**

["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"]

[[], [1], [2], [2], [], [1], [2], []]

**Output**

[null, true, false, true, 2, true, false, 2]

**Explanation**

RandomizedSet randomizedSet = new RandomizedSet();

randomizedSet.insert(1); // Inserts 1 to the set. Returns true as 1 was inserted successfully.

randomizedSet.remove(2); // Returns false as 2 does not exist in the set.

randomizedSet.insert(2); // Inserts 2 to the set, returns true. Set now contains [1,2].

randomizedSet.getRandom(); // getRandom() should return either 1 or 2 randomly.

randomizedSet.remove(1); // Removes 1 from the set, returns true. Set now contains [2].

randomizedSet.insert(2); // 2 was already in the set, so return false.

randomizedSet.getRandom(); // Since 2 is the only number in the set, getRandom() will always return 2.

**Constraints:**

* -231 <= val <= 231 - 1
* At most 2 \* 105 calls will be made to insert, remove, and getRandom.
* There will be **at least one** element in the data structure when getRandom is called.

Solution:

Declare a arraylist,hashmap and a random object

Insert:

If data already exists in the hashmap we can return false else

Insert into hashmap the number as key and current arraylist size as value

And add the number to the last of the arraylist

This is to ensure that every number which gets added to the list is added into hashmap using its index which it is currently present in the list

Remove:

If data doesn’t exist return false else

First fetch the index of the value of the data from the hashmap

And then check If it’s the last index of the arraylist if yes then remove data from both arraylsit and hashmap else

Swap last element with the index of the data from the hashmap and add the last element as the key index of the data element as value in the hashmap and then delete from both list and map

Random : using random object in java utility new Random().nextInt( arraylist size-1)

Code:

<https://github.com/Rishi143/DSASheet/blob/master/src/main/java/com/dsa/overall/_16_InsertDeleteGetRandom.java>