Session 7 - Leetcode Problem "Two Sum"

==========

in python we do not have Array..

instead we have a list

[1,2,3,4,5] - Array

[1,'sumit',3.5] - List

from python 3.5 version onwards we can give hints while creating lists..

Python 3 is the langauge in which we will code

https://leetcode.com/ is the platform from where we will pick the questions.

you do not need a premium account.

1st test case

nums = [2,7,11,15]

target = 9

output = [0,1]

2nd test case

nums = [3,2,4], target = 6

output = [1,2]

3rd test case

Input: nums = [3,3], target = 6

Output: [0,1]

To practise coding in python 3 we can use google colab

https://colab.research.google.com

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
```

input

[1,5,8,4,9,3,18,2]

outer loop - 1
inner loop - 5,8,4,9,3,18,2

outer loop - 5
inner loop - 8,4,9,3,18,2

outer loop - 8
inner loop - 4,9,3,18,2

outer loop - 4
inner loop - 9,3,18,2

outer loop - 9
inner loop - 3,18,2

outer loop - 3
inner loop - 18,2

outer loop - 18
inner loop - 2

.
target
27

output
[4,6]

1st approach - brute force..

O(n2)

code :

```
from typing import List
class Solution:
   def twoSum(self, nums: list, target: int) -> list:
     for x in range(0,len(nums)-1):
       for y in range(x+1,len(nums)):
         if (nums[x] + nums[y] == target):
           return [x,y]

s1 = Solution()
s1.twoSum([1,5,8,4,9,3,18,2],20)
```

! python --version

Python 3.8.10

2nd Approach -

[1,5,8,4,9,3,18,2]
(1,4)

lets sort the array - O(nlogn)

[1,2,3,4,5,8,9,18] - target 14
(4,6)

14 is sum of (5,9)
. .

we would scan each element at the max once to get the answer

if the sum of numbers at 2 pointers is more than the target than we shift the 2nd pointer one step backwards

if the sum of numbers at 2 pointers is less than the target than we shift the 1st pointer one step forward.

O(nlogn) + O(n) ~ O(nlogn)

[1,5,8,4,9,3,18,2]

correct answer should be (0,6)

[1,2,3,4,5,8,9,18] - target 19

if I consider just sorted array (0,7) which is not the correct answer.

```
if(new_nums[i] + new_nums[j] == target):
    return [nums.index(new_nums[i]),nums.index(new_nums[j])]
```

nums [3,3]

target 6

[3 ,0, 5, 2 ,3, 10] target 6

[0,2,3,3,5,10]

1st element is 3

2nd element is 3

code:

```
from typing import List
class Solution:
   def twoSum(self, nums: List[int], target: int) -> List[int]:
     new_nums = nums.copy()
     new_nums.sort()
     i = 0
     j = len(nums) - 1

     while (i < j):
       if(new_nums[i] + new_nums[j] == target):
         if (new_nums[i] == new_nums[j]):
           return [nums.index(new_nums[i]),nums.index(new_nums[j],nums.index(new_nums[i])+1)]
         else:
           return [nums.index(new_nums[i]),nums.index(new_nums[j])]
       elif (new_nums[i] + new_nums[j] < target):
         i = i + 1
       else:
         j = j - 1

s1 = Solution()
s1.twoSum([1,5,8,4,9,3,18,2],27)
```

3rd Approach -

dictionary

in java we have a hash map

in python we have dictionary

it stores key and value

```
employee_list = []

dict = {'sumit' : 9900100901, 'kapil' : 99001000902}

dict['satish'] = 9090909090

print(dict)
```

[1,5,8,4,9,3,18,2] 27 is my target element

27-18 = 9

dict =

1->0

5->1
8->2
4->3
9->4
3->5

(4,6)

space complexity = O(n)
time complexity = O(n)

code:

```
class Solution:
   def twoSum(self, nums: List[int], target: int) -> List[int]:
     dict = {}
     for i in range(0,len(nums)):
       complement = target - nums[i]
       if complement in dict:
         return [i,dict[complement]]
       dict[nums[i]] = i
```

Leetcode - Single Number
==========

input
[5,6,7,5,7]

output
6

solution - 1

brute force solution

O(n2) it involves 2 for loops

time complexity was O(n2)
space complexity was O(1)

solution - 2

what is a hashmap

key -> value

[5,6,7,5,7]

dict[5]

key value
5 -> 2
6 -> 1
7 -> 2

in case of python (dictionary)

key -> value

extra space or auxillary space

O(n)

time complexity O(n)

dict

[5,6,7,5,7]

hash_table[5] = hash_table[5] + 1

5 -> 2
6 -> 1
7 -> 2

```
class Solution:
   def singleNumber(self, nums: List[int]) -> int:
      hash_table = defaultdict(int)
      for num in nums:
         hash_table[num] = hash_table[num] + 1

      for num in nums:
         if hash_table[num] == 1:
            return num
```

time complexity = O(n)
space complexity = O(n)

solution 3
==========

[4,1,2,1,2]

xor - when we xor a number even number of times it returns 0

and when we xor a number odd number of time it returns the num itself

result = 0

result = result ^ 4 ^ 1 ^ 2 ^ 1 ^ 2

```
class Solution:
   def singleNumber(self, nums: List[int]) -> int:
      result = 0
      for num in nums:
         result = result ^ num
      return result
```

time complexity = O(n)
space complexity = O(1)

Majority Element - Leetcode
==========

Brute force - O(n2)

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

[1,1,1,2,2,2,2] O(nlogn)

O(1)

O(nlogn) + O(1) ~ O(nlogn)

9/2 = 4.5

floor(4.5) = 4

```
class Solution:
   def majorityElement(self, nums: List[int]) -> int:
      nums.sort()
      return nums[math.floor(len(nums)/2)]
```

Time complexity = O(nlogn)
space complexity = O(1)

Boyer-Moore voting algorithm

Time complexity = O(n)

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

majority element = 2
count = 1

majority elememt = 2

count = 2

majority elememt = 2
count = 1

majority elememt = 2
count = 0

majority elememt = 1
count = 1

majority elememt = 1
count = 0

majority elememt = 2
count = 1

majority_element = 1
count = 1

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

[3]

majority element = 3
count = 1

[2,2,3,3 | 4,4,2]

majority element = 4
count = 1

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

```
class Solution:
   def majorityElement(self, nums: List[int]) -> int:
      count = 0
      majority = None
      for num in nums:
         if count == 0:
            majority = num
         if majority != num:
            count = count - 1
         else:
            count = count + 1
      return majority
```

[2,2,1,1,1,2,2]
majority = 2

count = 1

In a single pass we are able to handle this..

O(n)

no extra space required...

Best Time to Buy and Sell Stock
==========

[5,8,9,4,3,10]

min_num = 3
profit = 7
max_profit = 7

[7,1,5,3,6,4]

profit = current_num - min_num 5
max_profit = 5
min_num = 1

```
class Solution:
   def maxProfit(self, prices: List[int]) -> int:
      max_profit  = 0
      min_num = prices[0]
      for i in range (1,len(prices)):
         if prices[i] < min_num :
            min_num = prices[i]
         profit = prices[i] - min_num
         if profit > max_profit :
            max_profit = profit
      return max_profit
```

Best Time to Buy and Sell Stock II
==========

[7,1,5,3,6,4]

5-1 = 4
6-3 = 3

4+3 = 7

[1,2,3,4,5]

total_profit = 0

```
for i in range (1,len(prices)):
            if prices[i] > prices[i-1]:
              total_profit = total_profit + prices[i] - prices[i-1]

return total_profit
```

buy at 1

sell at 5

5-1 = 4

buy - 1
sell - 2

buy - 2
sell - 3

buy - 3
sell - 4

buy - 4
sell - 5

total_profit = 7

[7,1,5,3,6,4]
        .

```
class Solution:
   def maxProfit(self, prices: List[int]) -> int:
      total_profit = 0
      for i in range (1,len(prices)):
         if prices[i] > prices[i-1]:
            total_profit = total_profit + prices[i] - prices[i-1]
      return total_profit
```

Remove Duplicates from Sorted Array
==========

input array
[1,1,1,3,3,5,5,5,5,8,8,8,9]

[1,3,5,8,9,5,5,5,5,8,8,8,9]

```
class Solution:
   def removeDuplicates(self, nums: List[int]) -> int:
      size = len(nums)
      insertIndex = 1
      for i in range(1,size):
```

```
        if nums[i-1] != nums[i] :
            nums[insertIndex] = nums[i]
            insertIndex = insertIndex + 1
    return insertIndex
```

Rotate Array

==========

[1,2,3,4,5,6,7]
k=7

[7,1,2,3,4,5,6]

[6,7,1,2,3,4,5]

[5,6,7,1,2,3,4]

size = 7

rotate it by k = 9

k%size

9%7 = 2

```
class Solution:
    def reverse(self, nums:list, start:int, end:int) -> None:
        while start < end:
            nums[start] , nums[end] = nums[end], nums[start]
            start = start + 1
            end = end - 1

    def rotate(self, nums: List[int], k: int) -> None:
        size = len(nums)
        k = k%size
        self.reverse(nums,0,size-1)
        self.reverse(nums,0,k-1)
        self.reverse(nums,k,size-1)
```

Maximum Subarray

==========

[-2,1,-3,4,-1,2,1,-5,4]
  .

maxsum = 6
sum = 5

2 important things whenever sum < 0
you make sum = 0 and move ahead

if sum > maxsum
maxsum = sum

[5,4,-1,7,8]

maxsum = 23
sum = 23

[-2,1,-3,4,-1,2,1,-5,4]
.
maxsum = 6
sum = 5

```
class Solution:
   def maxSubArray(self, nums: List[int]) -> int:
      sum = 0
      maxsum = nums[0]
      for num in nums:
         if sum < 0:
            sum = 0
         sum = sum + num
         if sum > maxsum:
            maxsum = sum
      return maxsum
```

time complexity = O(n)

space complexity = O(1)

Zero's and one's
==========

given an array of 0's and 1's

input
[0,1,0,1,0,0,1,1,1,0,1,0]

[0,0,0,0,0,0,1,1,1,1,1,1]

O(n)

in the first pass we have to take the count of 0's and 1's

and in the second pass we replace the elements...

O(n)

[0,0,0,0,0,0,1,1,1,1,1,1]
. .

whenever first pointer get a 0 we move it right

whenever second pointer gets a 1 we move it left

if first pointer is pointing on 1 and second pointer is pointing on 0 then we swap the elements

```
class Solution:
  def zerosAndOnes(self,nums: list) -> None :
    size = len(nums)
    first = 0
    second = size - 1
    while first < second:
      if nums[first] == 0:
        first = first + 1
      if nums[second] == 1:
        second = second - 1
      if nums[first] == 1 and nums[second] == 0 :
        nums[first] , nums[second] = nums[second] , nums[first]
    print(nums)

s = Solution()
s.zerosAndOnes([0,1,0,1,0,0,1,1,1,0,1,0,1,1,1,1])
```

Leaders in the array
==========

maxelement = 17

input {16, 17, 4, 3, 5, 2}

output = {2,5,17}

output { 17, 5, 2 }

brute force O(n2)

O(n)

{16, 17, 4, 3, 5, 2}

size = len(nums) = 6

ans = [nums[size-1]]

ans = [2]
max = 2

start index = size-2 = 4
end index = -1

for i in range (5,-1,-1)
print(i)

```
class Solution:
  def leaders(self,nums: list) -> list :
    size = len(nums)
    ans = [nums[size-1]]
    max = nums[size-1]
    for i in range(size-2,-1,-1):
      if nums[i] > max:
        max = nums[i]
        ans.append(nums[i])
    return ans

s = Solution()
ans = s.leaders([16,17,4,3,5,2])
print(ans)
```

Max Consecutive Ones
==========

input [1,1,0,1,1,1]

max_count = 3
count = 3

time complexity will be O(n)

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        max_count = 0
        count = 0
        for num in nums:
            if num == 1:
                count = count + 1
            else:
                count = 0
            if count > max_count:
                max_count = count
        return max_count
```

Merge Sorted Array
==========

[1,2,3,0,0,0] - n p1

.

[2,5,6] - m p2

.

[1,2,2,3,5,6]

time complexity O(n+m)

space complexity O(n+m)

[1,2,3,2,5,6] - n+m

now you sort this...

O((n+m)log (n+m))

[1,2,3,3,5,6] - n

.

[2,5,6] - m

.

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        p1 = m-1
        p2 = n-1
        for x in range (n+m-1,-1,-1):
            if p2 < 0 :
                break
            if nums1[p1] > nums2[p2] and p1 >= 0:
                nums1[x] = nums1[p1]
                p1 = p1 - 1
            else:
                nums1[x] = nums2[p2]
                p2 = p2 - 1
```