Python DSA

Day –1

Github- https://github.com/im-amit-kumar/Python-DSA

Problem –1 Longest Element in an array

https://www.geeksforgeeks.org/problems/largest-element-in-array4009/0

Solution

Brute force

```
class Solution:
    def largest(self, arr):
        arr.sort()
        return arr[-1]
```

Time Complexity –

Python Default uses Tim Sort algorithm for sorting . Worst Case will be O(n log n)

Space Complexity -

Sort in in-place operation does not uses any extra variable/space, so space c omplexity will be O(1)

Optimal Solution

Loop Approach

```
class Solution:
    def largest(self, arr):
        max_num = arr[0]
        for num in arr:
```

```
      if num > max_num:
        max_num = num
    return max_num
```

Time Complexity –

The function iterates through the entire array exactly once.

Each comparison (if num > max_num) is a constant-time operation.If n is the number of elements in arr:

Time Complexity: O(n)

Space Complexity –

Only a single variable max_num is used to store the current maximum.

No additional space or data structures are used.

Space Complexity: O(1)

Problem –2 Second Largest Number

https://www.naukri.com/code360/problems/ninja-and-the-second-order-elements_6581960

Optimal

```
from typing import List

def getSecondOrderElements(n: int , a:List[int])-> List[int]:
  smallest = float("inf")
  second_smallest = float("inf")
  largest = float("-inf")
  second_largest = float("-inf")

  for i in range(0, len(a)):
    if a[i] < smallest:
        second_smallest= smallest
        smallest=a[i]
    elif a[i] < second_smallest and a[i] != smallest:
        second_smallest = a[i]
    if a[i] > largest:
        second_largest =largest
        largest= a[i]
    elif a[i] > second_largest and a[i] !=large:
        second_largest=a[i]
  return [second_largest,second_smallest]
```

Time Complexity: O(n) – because it iterates through the list once.

Space Complexity: O(1) – because it uses only a fixed number of variables regardless of input size.

Problem –3 Check if Array is Sorted

https://www.geeksforgeeks.org/problems/check-if-an-array-is-sorted0701/1

Optimal

```
class Solution:

    def isSorted(self, arr) -> bool:

        for i in range(0, len(arr)-1):

            if arr[i] > arr[i+1]:

                return 0

        return 1
```

Time Complexity - O(n) – because it iterates through the list once.
Space Complexity – O(1)- The function uses only a loop variable i and returns an integer (0 or 1)

Problem –4 Remove Duplicates from Sorted Array
https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/

Brute Force

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        my_dict = dict()
        for i in nums:
            my_dict[i] = 0


        j = 0
        for n in my_dict:
            nums[j] = n
            j += 1
        return j
```

Dry Run

Step 1: Build the dictionary my_dict

We iterate through nums:

i = 0 → my_dict = {0: 0}

i = 0 → already exists

i = 1 → my_dict = {0: 0, 1: 0}

i = 1 → already exists

i = 1 → already exists

i = 2 → my_dict = {0: 0, 1: 0, 2: 0}

i = 2 → already exists

i = 3 → my_dict = {0: 0, 1: 0, 2: 0, 3: 0}

i = 3 → already exists

i = 4 → my_dict = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0}

Result after this loop:

my_dict = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0}

Step 2: Overwrite nums with unique values from my_dict

We iterate through the keys of my_dict and replace values in nums.

j = 0

n = 0 → nums[0] = 0, j = 1

n = 1 → nums[1] = 1, j = 2

n = 2 → nums[2] = 2, j = 3

n = 3 → nums[3] = 3, j = 4

n = 4 → nums[4] = 4, j = 5

Now nums is:

[0, 1, 2, 3, 4, 2, 2, 3, 3, 4]

Final Step:

return j # which is 5

Final Output:

Return value: 5

Modified nums (first 5 elements): [0, 1, 2, 3, 4]

Time Complexity: O(n)

We loop through the list twice:

1. First to build the dictionary of unique elements.
2. Second to overwrite the original list with those unique elements.

Each loop runs at most `n` times, so:

O(n) + O(n) = O(2n) ≈ O(n)

Space Complexity: O(n)

We use a dictionary to store unique elements.

In the worst case, if all elements are unique, the dictionary will store all `n` elements.

So, space complexity is: O(n)

Optimal

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return 1
        i = 0
        j = i + 1
        while j < len(nums):
            if nums[j] != nums[i]:
                i += 1
                nums[i] = nums[j]
            j += 1
        return i + 1
```

Dry Run

nums = [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]

Initial values:

- nums: [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]
- i = 0, j = 1

Iteration details:

j	nums[j]	nums[i]	Condition (nums[j] != nums[i])	Action	Updated i	Updated nums
1	0	0	False	skip	0	[0, 0, 1, 1, 1, 2, 2, 3, 3, 4]
2	1	0	True	i += 1, nums[i] = nums[j]	1	[0, 1, 1, 1, 1, 2, 2, 3, 3, 4]
3	1	1	False	skip	1	[0, 1, 1, 1, 1, 2, 2, 3, 3, 4]
4	1	1	False	skip	1	[0, 1, 1, 1, 1, 2, 2, 3, 3, 4]
5	2	1	True	i += 1, nums[i] = nums[j]	2	[0, 1, 2, 1, 1, 2, 2, 3, 3, 4]
6	2	2	False	skip	2	[0, 1, 2, 1, 1, 2, 2, 3, 3, 4]

7	3	2	True	i += 1, nums[i] = nums[j]	3	[0, 1, 2, 3, 1, 2, 2, 3, 3, 4]
8	3	3	False	skip	3	[0, 1, 2, 3, 1, 2, 2, 3, 3, 4]
9	4	3	True	i += 1, nums[i] = nums[j]	4	[0, 1, 2, 3, 4, 2, 2, 3, 3, 4]

Final values:

- i = 4 → which means 5 unique elements (i + 1)
- nums after removing duplicates (first 5 values): [0, 1, 2, 3, 4, ...]

Summary:

- Input: [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]
- Modified nums: [0, 1, 2, 3, 4, _, _, _, _, _] (values after index 4 are irrelevant)
- Return value: 5

Time Complexity: O(n)

Space Complexity: O(1) (in-place modification, no extra space used)