1. **Two Sum**

Input: nums = [2,7,11,15], target = 9

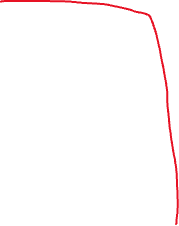
Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

**Brute Force:** By Using a 2 nested ForLoop

A computer screen shot of a program code

Description automatically generated **Complexity**  
Time complexity: O(N^2);



Space Complexity: O(1);

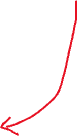
Problem with this code is: Above Code will waits until it has checked all possible pairs before returning the result.

**Complexity:**TC : O(N^2) : Because of 2 nested For Loop  
SC: O(1) : because we are using array of size 1.

|  |  |
| --- | --- |
|  | Mistakes |
|  |  |

**Bit improvement:** I have removed the array and it might be faster in cases where a valid pair is found early in the iteration.  
A computer screen shot of a program code

Description automatically generated



**Complexity: same as above.**

**Optimize Approach :** Using a HashMap because is save as key value pairs.

|  |  |
| --- | --- |
|  | Mistakes |
|  | I forgot how to use Map , HashMap . and it should be I key, value pairs |
|  | I forgot the hashmap functions : **containsKey() map.get() map.put()** |

A screen shot of a computer code

Description automatically generated



**Complexity:**TC : O(N) : Because of 1 For Loop  
SC: O(N) : because we are using HashMap of size N(means size of the array).

1. **Sort Colors (0,1,2)**

Input: nums = [2,0,2,1,1,0]

Output: [0,0,1,1,2,2]  
**Brute Force: Just use a in-built sort function.**

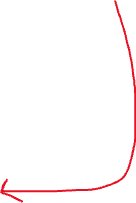
**Better Approach:** using 2 for loop ( one loop split into 3 parts).  
Note: Make sure when you are working on the 2nd loop and its sub-parts check the limit otherwise it will show a different output.

|  |  |
| --- | --- |
|  | **Mistakes** |
| 1 | I forgot the for loop Condition eg: i< (zeroes + ones) |

|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(2N) | 2 for loop |
| SC : O(1) | As we are not using any extra space. |

A computer screen shot of a program code

Description automatically generated



**Optimize Approach: Using 3 Pointers(Low, Mid, High) for 0,1,2   
Point to Remember: Low and Mid starts with 0th location**

|  |  |
| --- | --- |
|  | **Mistakes** |
| 1 | I forgot to use “else-if “ |

|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(N) | We are using a single loop that can run at most N times. |
| SC : O(1) | As we are not using any extra space. |

|  |
| --- |
| Key Notes: |
| We can also use a swap function for swapping the values. It will reduce the code length. |

A screenshot of a computer program

Description automatically generated



1. Majority Element

Input: nums = [2,2,1,1,1,2,2]

Output: 2

**Brute Force : using nested loop**

|  |  |
| --- | --- |
|  | **Mistakes** |
| 1 | I forgot to keep “count=0” inside 1st for loop. because “count” should be equal to 0 when we exit from the 2nd loop. |

|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(N^2) | Nested for loop |
| SC : O(1) | As we are not using any extra space. |

A computer screen shot of a program code

Description automatically generated  
  
  
**Better Approach :** Using HashMap.



|  |  |
| --- | --- |
|  | Key Note |
| 1 |  |
| 2 | As we are not using any extra space. |

|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(N\*logN) + O(N) | We are using a map data structure. Insertion in the map takes logN time. And we are doing it for N elements. So, it results in the first term O(N\*logN). The second O(N) is for checking which element occurs more than floor(N/2) times. If we use unordered\_map instead, the first term will be O(N) for the best and average case and for the worst case, it will be O(N2). |
| SC : O(N) | As we are using a map data structure. |

A computer screen shot of a program code

Description automatically generated



|  |
| --- |
| Key Notes: |
| Make sure that check for constraints otherwise it will create a problem when you put any value while initializing the “candidate”. |

|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(N) + O(N), where N = size of the given array. | The first O(N) is to calculate the count and find the expected majority element. The second one is to check if the expected element is the majority one or not.  Note: If the question states that the array must contain a majority element, in that case, we do not need the second check. Then the time complexity will boil down to O(N). |
| SC : O(1) | As we are not using any extra space. |

**Optimize:** Using Moore Voting algorithm  
A screen shot of a computer program

Description automatically generated



1. Maximum SubArray

A computer screen shot of a program code

Description automatically generated

Time Limit Exceeded



A computer screen shot of a program code

Description automatically generated

Time Limit Exceeded



A screen shot of a computer program

Description automatically generated

1. Best time to buy and sell Stock

A screen shot of a computer code

Description automatically generated

Time Limit Exceeded



Optimized Solution :

A computer screen with colorful text

Description automatically generated

1. Next Permutation

A screen shot of a computer program

Description automatically generated



1. Leaders is Array :

A computer screen shot of a program code

Description automatically generated



1. Longest Consecutive Sequence in an Array

Brute Force Approach: Using Array.sort

A screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

Test Case:



|  |  |
| --- | --- |
| **Complexity** | **Reason** |
| TC : O(n \* log(n)) + O(n) | Sorting: The Arrays.sort() operation takes O(n \* log(n)) time complexity, where 'n' is the number of elements in the input array.  Loop: After sorting, the loop iterates through the sorted array once, which takes O(n) time complexity.  Combining these two factors, the overall time complexity is O(n \* log(n)) + O(n), which simplifies to O(n \* log(n)). |
| SC : O(1) | As we are not using any extra space. |

Optimized Approach : Using Sets  
  
  
Things I learned in 1D- Arrays Problem:

1. Using Nested for-loop

* Usually start with nested for loop
* 1st loop where int i=0;
* 2nd loop where j=0 and it will check for all the element w.r.t “i” where “i” remains at it position.

1. Using 2 pointer approach with Binary Search,
   * 1st pointer is at 0th location 2nd pointer it at end location.
   * us can also do a swap.
2. Use of Moore Voting algorithm.
3. Learn Map and its Functions.