Arrays

Easy

① Find the largest ele

- initialising a variable to the first element of the array then loop throughthe whole array if it gets anelement larger than the initial variable then store that in largest & repeat the same till the end of the array.

② largest & Second largest

- for the given array initialize 2 variable largest & slargest both to int min
- iterate throughthe loop & with 2 ifs check for the largest first update slargest to take largests value & then largest for a new value

1st if

2nd if to get the second largest the current ele should be greater than current slargest but less than largest then that value is put into slargest.

__/__/____

(3) is sorted or not

- just check if the previous element of the current element is larger if its larger then the array is not sorted so return saying saying not sorted
- If not if the loop goes till the end that means its sorted.

(4) Remove duplicates

Brute force

- assuming that the array has consecutive duplicates (works only for this)
- we keep a tracker i from the start of the array to keep track of the last updated position of the non duplicate element.
- compare jth element to currentElement if its not the same then bring the current ele to i+1

basically what this i would've stopped at a place it found a duplicate so when you find a non duplicate element you're bringing it in front of i

__/__/____

using set

- just insert all the array elements into the set it stores only non duplicates so the print out the set.

(5) left rotate array by one place

BF just create a temp array keep the first element aside then copy all elements from 1 - n to the temp array then place the first element in the last place of temp array (n-1)

OP without creating another array just keep the first ele seperate & then ~~overwrite~~ start copying all elements one place back (basically overwrites) then at last place the first element (n-1)

(6) left rotate array by D places.

OP- create a temp array of size D and store D elements into temp array (ex: D=2, temp[D])

- now in the main array store shift all elements to start of the array from D^{th} place till end of array (arr[i-D] = arr[i])
- store the temp array elements back to OG array at the end looping through temp array (arr[n-D-i] = temp[i])

__/__/____

(7) Put all 0s to end of the array.

#BF

- first create an empty temp array to store all non 0 elements and get a count of non zero elems
- then copy back all non zero elements to OG array (0 → nz)
- then for the remaining last places in the array fill it with 0. (nz → n)

#OP work on one single array keep a track of non 0 elements whenever you find non zero elements swap it to beginning & increment i will stop when it finds 0 then the next non 0 element will be swapped with a j.

(8) Linear Search

- just check the array if the given query is there if yes then returns its place.

__/__/____

(9) union of 2 arrays

- just insert both arrays elements into a set then copy back from set to new array.

(10) find the missing number 1 to N

BF by keeping a counter from 1 you check the array if its equal to counter then do counter++ if not return the counter which gives the missing number.

OP you keep 2 variables one for sum of n nos and another for sum of all array elements then when you do [sum of n - sum of array] u get the missing number.

(11) find the max consecutive ones

OP basically just keep count of consecutive ones in the array then but to return the max consecutive ones keep another variable to store the max ones count (maxi = max(maxi, count))

__/__/____

⑫ find the num that's present only once
{ 2, 3, 3, 4, 4}

BF - for every single element in the array check how many times its repeated then return the element that's present only once using 2 for loops

Better - just insert all array elements into a map then fetch the element with value one by iterating through the map.

OP - Using XOR starting with XOR=0 do XOR with all array elements since only one ele is present once the XOR result will return that element since XOR of all double repeated elements will be 0 & 0 ^ single will return that single ele.

__/__/____

(13) & (14) find the longest subarray that sums to k
(Better approach for array with negatives)

BF just runs 2 loops the outer loop(i) to move index 1 by 1 from start of the array & the inner loop(ii) starting from the first element of the outer loop you start adding all elements when the sum = k
then you get that subarray length by doing j - i + 1
then update to maxi - maxi is updated in such a way that in every subarray that sums to k holds the length of the longest subarray till the end & returns that.

Better

- Using a loop you add up all the elements moving indexes one by one calculating the sum upto each index.

- Now check if the sum upto current index = k
if its true then update the subarray length to maxi

- Now check if there are previous subarrays &
if there are subarrays check its length & compare the old & new subarray size & update to maxi.

__/__/__

- Now just store the current sum as key & it's index as the value in map.
- finally return the maxi which has the longest subarray size.

DP find

- basically ~~crossed~~ in a given array you keep 2 pointers starting from the start of the array left & right

right - tracks current index & the sum till current index

left - will be at the start of the array but when current sum ~~is~~ > K then we subtract left with current sum & move ahead

- if the current sum is higher than k then subtract the elements from the previous subarray with current ~~sub~~ sum

ie subtract left pointer elements with current sum & move left pointer till sum <= K

__/__/____

2- Now you check if the sum=k if its true then update the max subarray length to maxi

3- Now again move the right pointer & calculate sum

- finally return the maxi